Unlocking ECMP Programmability for Precise Traffic Control

Yadong Liu, *Tencent;* Yunming Xiao, *University of Michigan;* Xuan Zhang, Weizhen Dang, Huihui Liu, Xiang Li, and Zekun He, *Tencent;* Jilong Wang, *Tsinghua University;* Aleksandar Kuzmanovic, *Northwestern University;* Ang Chen, *University of Michigan;* Congcong Miao, *Tencent*

https://www.usenix.org/conference/nsdi25/presentation/liu-yadong

This paper is included in the Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation.

April 28–30, 2025 • Philadelphia, PA, USA

978-1-939133-46-5

Open access to the Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation is sponsored by

Unlocking ECMP Programmability for Precise Traffic Control

Yadong Liu[1*], Yunming Xiao[2*], Xuan Zhang[1], Weizhen Dang[1], Huihui Liu[1], Xiang Li[1], Zekun He[1], Jilong Wang[3], Aleksandar Kuzmanovic[4], Ang Chen[2], Congcong Miao[1]
[1] *Tencent,* [2] *University of Michigan,* [3] *Tsinghua University,* [4] *Northwestern University* *

Abstract

ECMP (equal-cost multi-path) has become a fundamental mechanism in data centers, which distributes flows along multiple equivalent paths based on their hash values. Randomized distribution optimizes for the aggregate case, spreading load across flows over time. However, there exists a class of important *Precise Traffic Control* (PTC) tasks that are at odds with ECMP randomness. For instance, if an end host perceives that its flows are traversing a problematic switch/link, it often needs to change their paths before a fix can be rolled out. With randomized hashing, existing solutions resort to modifying flow tuples; since hashing mechanisms are unknown and they vary across switches/vendors, it may take many trials before yielding a new path. Many other similar cases exist where precise and timely response is critical to the network.

We propose *programmable ECMP* (P-ECMP), a programming model, compiler, and runtime that provides precise traffic control. P-ECMP leverages an oft-ignored feature, ECMP groups, which allows for a constrained set of capabilities that are nonetheless sufficiently expressive for our tasks. An operator supplies high-level descriptions of their topology and policies, and our compiler generates PTC configurations for each switch. End hosts can reconfigure specific flows to use different PTC policies precisely and quickly, addressing a range of important use cases. We have evaluated P-ECMP using simulation at scale, and deployed one use case to a real-world data center that serves live user traffic.

1 Introduction

ECMP (equal-cost multipath) is foundational to modern data centers, for distributing traffic stochastically along multiple paths. ECMP is widely adopted not only because Fat-tree topologies [12, 26] naturally expose equivalent paths, but also because of its ease of implementation. The flow hashing mechanism at its core is efficiently realizable in ASICs and is widely available in commodity off-the-shelf switches.

Randomized flow hashing has turned out to be a simple yet effective mechanism. In data center networks, traffic patterns are hard to predict and they fluctuate over time. ECMP embraces this observation and optimizes for the aggregate. Individual flows might still fall victim to unfortunate choices of randomness (e.g., hash collisions between large flows); however, given enough flows, and when considering a long timespan, this randomness produces a good traffic spread and utilizes the underlying network efficiently. All production data center networks that we are aware of make use of ECMP.

However, strong aggregate performance does not remove the pitfalls of randomness in specific scenarios. In particular, we have identified a set of *Precise Traffic Control* (PTC) scenarios where rapid and precise response to network anomalies is key but randomness hinders it. Consider the case of switch/link failures that render an ECMP path unavailable or unstable [49, 56, 66]: it is then critical to quickly redirect network traffic off this path before we roll out a fix. In the presence of ECMP, this is no easy task: state-of-the-art solutions (e.g., Google's PLB [49] or PRR [56]) resort to random modification of flow tuples (e.g., by varying TCP ports), hoping that it would eventually yield a new path. This trial-and-error process could take minutes to complete [49]. Likewise, network monitoring systems [28, 31, 32] often need to quickly probe all paths to localize a fault; with randomness, we again need repeated flow modifications until all paths are covered. What these PTC scenarios have in common is the need to 1) exert precise control over traffic paths 2) against the backdrop of ECMP—the latter is equally important, as we cannot afford to *disable* ECMP, even if momentarily. In other words, the majority of network traffic should still be subjected to ECMP, while we seek a way to impose precise control *over* ECMP.

We propose to achieve this by carefully leveraging a commodity switch feature, ECMP groups, which is as simple, efficient, and widely available as basic ECMP itself. Basic ECMP maps a flow f to a preconfigured list of outgoing ports $l = [p_0, p_2, \cdots, p_n]$ based on the range that $hash(f)$ falls into. ECMP groups allow a flow to carry a selector s in its packet header, where each s maps to a its own port list l_s. Operators

*Equal contribution.

can configure the default ECMP group, say $\langle s = 0, l_{s=0} \rangle$, in conjunction with other $\langle s, l_s \rangle$ in an ECMP *matrix*. By doing so, operators could ensure, for instance, paths in l_s and $l_{s'}$ are always disjoint, and that precise control can be achieved by a single modification to the packet header field s. By leveraging this long-overlooked feature, we could instantiate ECMP matrices for each switch, so that traffic can enter/exit random modes of control by the end hosts toggling a packet header.

We call our key design *programmable ECMP* (P-ECMP), which does not require fanciful features (e.g., P4 programmability [29, 35, 39, 62]) but only exposes a restricted amount of reconfiguration for ECMP. The key challenge is to develop the 1) programming model, 2) compilation support, and 3) update mechanism for P-ECMP. While we could ask the operator to handcraft an ECMP matrix for each switch in her network, this is a tedious and error-prone undertaking. Instead, the operator describes her network in a topology matrix T, her PTC policies in control matrix C, and our compiler generates the ECMP matrices across the topology to realize precise control—while taking into account SRAM constraints across heterogeneous switches. Our consistent update mechanism further provides atomic transactions within and across switches, whether to repopulate the ECMP matrices or to dynamically change the selectors while traffic is still in flight.

We demonstrate a set of P-ECMP use cases that were previously difficult to achieve due to the lack of precise control:

- Network failover: End hosts can precisely re-path their flows to avoid perceived failures, whether silent packet drops or link flaps along an existing ECMP path.
- Failure localization: Network monitoring systems can precisely cover a set of underlying paths with their probes, e.g., to localize failures efficiently.
- Load imbalance: When skewed traffic patterns lead to performance degradation, end hosts can act upon congestion signals (e.g., ECN) to re-path select flows.
- Multipath protocols: With multipath protocols (e.g., MPTCP [50], MPRDMA [40]), we can ensure that each flow traverses a disjoint path for best performance.
- Packet spraying: Workloads that are resilient to network reordering [24] can spray packets across multiple paths for higher fairness and efficiency [21].
- Segment routing: Hosts can even precisely dictate all or parts of the forwarding paths for their flows, e.g., so that they traverse specific middleboxes.

The last use case, segment routing, points to the upperbound flexibility of P-ECMP in exerting precise control: over the entirety of a network path. The first use case, network failover, has been deployed and evaluated in a live data center network at scale. At the same time, all use cases are deployable in commodity data centers with legacy devices, and traffic that does not trigger PTC policies will continue to be handled by ECMP. Our evaluation with simulation and testbed experiments shows the effectiveness and low overhead of P-ECMP.

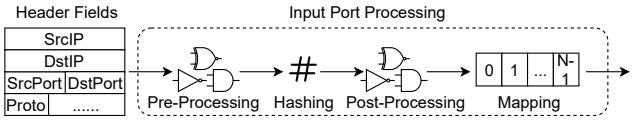


Figure 1: Four stages of ECMP processing.

2 Motivation

Many equal-cost paths exist in Clos topologies [12, 26], offering a natural opportunity for ECMP-based load balancing. When a packet arrives at an ingress port, relevant headers such as IP addresses and protocol types are parsed for further processing, including routing decisions and ECMP. The SAI (switch abstraction interface) [7] standard defines ECMP processing in four stages: pre-processing, hashing, post-processing, and mapping, as shown in Figure 1. During pre-processing, the switch performs bit-wise operations to obtain the flow ID, *e.g.,* the five-tuple <src_ip, dst_ip, src_port, dst_port, protocol>. The flow ID is then hashed by the hashing stage using ASIC functions (e.g., CRC) and reduced to a small integer. The post-processing stage also performs bit-wise operations, usually fixed in their logic, on the hash outputs. Finally, the mapping stage bins the hash output into one of the next hops $\{0, 1, \ldots, N-1\}$ for forwarding. All four stages are simple to realize in ASIC, so ECMP is widely available in commercial off-the-shelf switches.

2.1 The Pitfalls of Randomness

Since ECMP decisions are determined by the flow ID and the hash functions, across many flows the spreading pattern is essentially random. Furthermore, this randomness is difficult to predict or control, since switch vendors might employ different hash functions and their details are opaque to the network operator. While the pitfalls of ECMP regarding hash collisions resulting in (temporary) load imbalance are well-known, we have identified a more severe class of problems that, once they manifest, produce a more persistent issue inside the network.

Example: Network failover by re-pathing flows. Timely response to network failures is critical for high availability and service-level objectives (SLOs). Network failures can stem from a variety of issues, including hardware malfunctions in switches, software bugs, or configuration errors. While many of these failures can be addressed by the network control plane – for example, by using protocols like BFD to quickly detect failures and reroute traffic – others are more challenging. Failures such as silent packet drops, whether caused by misconfigured ACLs, routing blackholes, or errors in CRC computation, often affect only a subset of traffic and evade timely detection by traditional control-plane mechanisms.

In these cases, applications will perceive the failures earlier and the traditional wisdom is to produce a faster response from the end host side before the control plane fixes come in. Google [56] and Alibaba [66] have proposed using additional

header fields to modify the hash input used in network path selection, effectively tricking ECMP into recalculating the path. Applications can therefore re-path affected flows to bypass failures [56, 66]. Figure 2(a) shows an example where Flow B successfully avoids the failure encountered by Flow A on the original path.

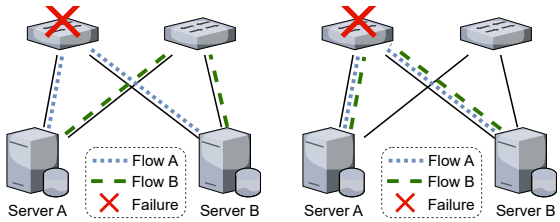
Problem: Lack of precise control. Since ECMP makes random decisions, and hash functions are opaque to operators, this re-pathing effort is essentially a trial-and-error process. It is common to encounter multiple failed re-pathing attempts due to hash collisions [56]. Figure 2(b)[1] illustrates a failed re-path attempt, where the re-pathed flow (Flow B) is mapped to the same path as the original flow (Flow A), encountering the same failure at Switch A.

Quantification of Failed Attempts. Quantitatively, the probability that either the "to" or "from" path of a new flow is mapped to a healthy path under ECMP is $\frac{N-1}{N}$, where N is the total number of equal-cost switches in the network tier where the failure occurs. The probability that the flow successfully circumvents the failure on its first re-path attempt, *i.e.,* both "to" and "from" paths avoid the failure point, is therefore $p = \frac{(N-1)^2}{N^2}$. If the flow fails to circumvent the failure on the first attempt, both "to" and "from" paths need to be re-selected randomly. Each re-path attempt is independent of the previous attempts and has the same probability of encountering the network failure as before, *i.e.,* $\frac{1}{N}$. Let T be the number of re-paths needed for successful failover. Then T has a geometric distribution with parameter p, where $P(T=k) = p(1-p)^{k-1}$. The expectation of T is given by:

$$\mathbb{E}[T] = \frac{1}{p} = \frac{N^2}{(N-1)^2}. \quad (1)$$

While the average number of new flows needed for a successful failover is relatively low, with T being at most 4 when $N = 2$ and decreasing when $N > 2$, its long-tail distribution can lead to extremely slow failovers for critical flows, resulting in SLA violations. For example, privious study [49] has shown that during a link failure, random re-pathing can cause certain flows to be disabled for 1-2 minutes due to unlucky failed re-pathing. This may further lead to unstable loads on other links for more than 20 minutes. Another proposal from Google is to re-path flows relying on TCP's RTO signal to indicates failures [56], where RTO's duration is influenced by RTT and subject to exponential backoff. Without a maximum limit on RTO and with a success probability $p \leq 0.5$, the average time for failure recovery using PRR cannot converge. Fortunately, RTO typically has an upper bound, and $p > 0.5$ when there are three or more switches in the ECMP group. Despite this, the recovery time in such setups is significantly longer compared to situations where failover is guaranteed to succeed on the initial attempt.

[1] For the sake of simplicity, examples in Figure 2 assume that the flow's "to" and "from" paths are identical. However, in reality, the "to" and "from" paths of the same flow may be different, as the flow identifier, *i.e.,* five-tuple header fields, is different when the source and destination are reversed.



(a) Succeeded case. (b) Failed case.

Figure 2: Demonstration of (a) succeeded and (b) failed failover by flow re-pathing or multi-path protocols.

2.2 Precise Traffic Control

Re-pathing is not an isolated case where the lack of precision causes operational challenges. In fact, there are many scenarios where precise traffic control (PTC) is critical.

Network failure localization. ECMP also hinders network failure localization, a crucial function for maintaining and ensuring high availability in modern data centers. Many data centers employ systems similar to Pingmesh [28], where servers regularly send ping (ICMP) packets to probe network availability. However, due to the randomness of ECMP, these probes are randomly distributed across available paths, without guaranteed coverage of all network hops (i.e., switches). As a result, achieving full path coverage requires an excessive number of probe packets, often exceeding the number of actual paths between any two servers. Further, the randomness in ECMP limits monitoring tools' ability to effectively identify localized network issues. While it can detect regional problems, it lacks the granularity needed to precisely pinpoint failures at the switch or link level [54].

Load imbalance. The goal of ECMP is to distribute flows based on their five-tuples, achieving an even spread. However, as the traffic loads are generally long tailed, the randomness in ECMP may lead to imbalances in real-world workloads, or even exacerbate congestion hotspots [13, 34]. To mitigate this, Google introduced a method called PLB, where applications re-path their flows upon detecting congestion signals – such as a few ECN-marked packets from switches – thereby improving load balancing [49]. However, addressing load imbalance in this way can lead to failed attempts. At certain cases, flows can be disrupted for 1-2 minutes due to unlucky attempts, destabilizing load distribution across other links for over 20 minutes, as reported by Google's PLB [49].

Multi-flow applications and multi-path protocols. Applications may initiate multiple flows for their transfer, and some transport protocol such as MPTCP [16] or MPRDMA [40] are inherently multipath. By exposing path diversity, we can leverage multiple paths for transmission, identify best paths, and achieve better balance in terms of path utilization – assuming that paths are independent. However, multi-flow applications and multi-path protocols do not interact well with ECMP. Its randomness may mean that paths could be overlapping. This could lead to (sub-)flows being routed along the same path,

diminishing the benefits of using multiple paths.

Packet spraying. Packet spraying is a technique that distributes packets across multiple available paths on a per-packet manner. This is particularly useful for flows that do not require strict packet ordering – such as UDP flows or networks with reordering-resilient network stack [24]. Previous results show that packet spraying can greatly improve network fairness and efficiency [21]. Unfortunately, the randomness in ECMP routing also undermines the effectiveness of packet spraying, as it prevents precise distribution of packets across paths, leading to potential hotspots and imbalance.

Segment routing. Finally, segment routing is a network function that enables the sender (host machines) to specify all or part of the forwarding path for packets. It is widely employed in wide-area networks (WANs) to support traffic engineering [44]. In data center networks, applications can benefit from segment routing such as diverting specific packets to designated middleboxes. However, segment routing is a network function that demands precise control, which is not compatible with the randomness of ECMP.

In all of these cases, we need a solution that enables the majority of the traffic to still use ECMP, while temporarily allowing certain flows or applications to exert precise control over ECMP. One solution is to implement these use cases in programmable switches, so that certain flows are processed differently when they traverse the network. However, this requires a universal deployment of programmable switches inside data center networks. Another solution is to use out-of-band mechanisms, as proposed in RePaC [66], which precomputes the mappings between inputs and outputs of the hash algorithm in ECMP and distributes these mappings to host machines. This allows host machines to exercise precise control of relative path (offset). However, this assumes uniform switch configurations within the data center network, whereas operators often source equipment from multiple vendors to avoid vendor monopoly and increase resilience against single-product flaws.

2.3 Idea: Leverage ECMP Groups

To support PTC use cases on commodity switches, our key observation is that ECMP has a certain amount of flexibility that we have not fully used. This is signaled by the fact that a significant portion of the SRAM memory resources reserved for ECMP routing remains underutilized. Indeed, the abundant ECMP table capacity is primarily designed to support ISP use cases, where asymmetric topologies and a large number of destination sets are common. However, this is not the case in DCNs which have symmetric topologies and thus require fewer ECMP groups at normal times. Our measurement of the commodity switches in a production DCN revealed that, while hardware providers (e.g., Trident5 ASIC) allocate space for 4 pipelines (e.g., uplink or downlink), with 8K ECMP groups and 64K ECMP members per pipeline, current switches use only up to 8 groups (0.1%) and 512 members (<1%) at most (spine switches). ToR and leaf switches use only one group (0.01%) and up to 64 members in uplinks (0.1%). Our idea is to extend the basic ECMP mechanism both *horizontally* and *vertically* leveraging *ECMP groups*, a common feature in ECMP-capable switches.

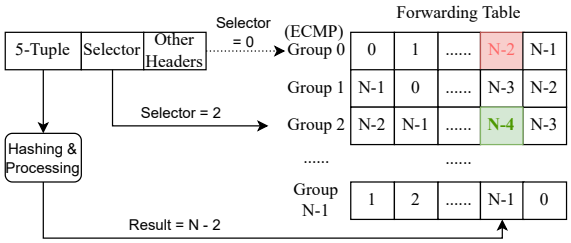


Figure 3: An example of PTC of path offset by P-ECMP where the selector is 2 and the outcome of the ECMP processing is $N-2$. By redirecting the ECMP group by 2, P-ECMP re-directs the forwarding port from $N-2$ to $N-4$.

In basic ECMP, the hash value of a flow tuple $Hash(f)$ is used as the index into an array to determine the output port. The ECMP group feature allows for multiple such arrays as depicted in Figure 3. These multiple (M) ECMP groups transform the forwarding table from a simple array (with only group 0) into a matrix C. The selection of an ECMP group can be driven by an additional field in the packet header, referred to as the selector s. As a result, the forwarding decision is based on the tuple $< s, Hash(f) >$, where s determines the row, and the hash of the flow's selected field $Hash(f)$ determines the column. That is, the final output port is given by $C[s, Hash(f)]$. This extended flexibility in the forwarding table enables us to tune the C matrix for precise control.

Horizontal: Let us first consider only one ECMP group, with C being a one-dimensional array $[0, 1, \ldots, N-1]$. This already allows for extending the forwarding behavior by inserting, deleting, or replacing items in this array. For instance, by adjusting the number of entries in the table according to specific rules, we can implement weighted ECMP, where the weight of each next hop reflects its proportion in the table. If the array contains only a single entry, this effectively achieves precise control, eliminating randomness so that packets are always forwarded to the same next hop.

Vertical: Considering multiple ECMP groups provides further flexibility, in two dimensions: (i) multiple ECMP policies can coexist simultaneously, allowing applications to select a policy on demand, and (ii) it introduces an additional layer of determinism on top of the inherent randomness of ECMP. Specifically, applications can embed a value in a custom header field to specify which row of the matrix to use. The row selection is deterministic, while the column selection within each row (*i.e.,* the ECMP routing) remains random.

2.4 P-ECMP contributions

The contributions we made in developing *programmable ECMP* (P-ECMP) are as follows.

Programming model. Our first contribution is a systematic study and exploration of the inherent flexibility in ECMP, upon which we distill the abstraction of ECMP *programmability*. Specifically, we categorize the programmability of ECMP into three sets of intent: (i) PTC of path offset, (ii) PTC of the exact next hops, and (iii) Weighted-Cost Multi-Path (WCMP). Since our approach to WCMP aligns with previous work [29], this paper focuses on the first two intents.

Compiler. We also develop a compiler that automates the process of generating ECMP forwarding tables for all switches in the network. The compiler takes the programmability intent, along with the network topology and resource constraints, as inputs. This streamlines the development and deployment of ECMP programmability, making it intuitive and easy for network operators to implement and manage.

Runtime updates. A challenge with ECMP programmability lies in ensuring consistency during deployment. Specifically, when deploying ECMP forwarding tables on commodity switches in production DCNs, issues can arise. For example, a host machine might have an outdated database and send a packet with a flag corresponding to an invalid ECMP policy, or in-flight packets might not be able to adjust their embedded flags and thus hit an invalid policy at next switches. This could lead to packet loss or out-of-order delivery. Given the constant high volume of traffic, an inconsistent update could have a significant impact. To address this, we propose a distributed transaction scheme that ensures consistency during updates, preventing packet loss or reordering issues.

Rich use cases. P-ECMP enables a range of PTC tasks, such as the ones motivated before. We evaluate these use cases and report our experience in testing the network failover use case in a production data center at scale.

3 System Design

3.1 Where Can We Program ECMP?

To better understand the potential programmability residing in commodity switches, let us first trace back to the architecture of commodity switches and its forwarding process. In the ECMP processing described above, three elements can be modified: the input fields, the hashing algorithm, and the mapping stage. Below, we explore whether modifying each of these components can alter the output port and enable programmability. Table 1 summarizes the analysis.

First, modifying the input fields in ECMP processing enables the generation of a different hash output, which in turn determines a different output port. This approach has been employed in several studies, such as PLB [49] for improved load balancing, and PRR [56] and RePaC [66] for network failover. These studies introduce an additional header field as an input to ECMP, beyond the standard five-tuple flow definition. This allows for changing the forwarding output port by adjusting the value of the extra header field without disrupting ongoing flow sessions.

Table 1: Programmability potential of commodity switch at different forwarding stages.

Stage		PTC of Offset	PTC of Hop	Compatibility	Practicality
	Input	Yes*	Yes*	Yes	No
	Hashing	No	No	No	No
Mapping	Member Redirect	No	Yes	No	Yes
	Group Redirect	**Yes**	**Yes**	**Yes**	**Yes**

However, if the hashing algorithm used by the switches is unknown, this modification may result in random outcomes, thereby limiting programmability. When the hashing algorithm is known in advance, such as what was assumed in RePaC [66], it is possible to deterministically adjust the value of the extra header field to achieve the desired output port. Specifically, let f represent the selected flow fields for hashing, and $Hash(f)$ denote the hashing result. RePaC deterministically alters the output port by modifying the input fields with a difference $\Delta \in \{0x0...01, 0x0...10, \ldots, 0x1...00\}$, *i.e.,* the output becomes $Hash(f+\Delta)$. Leveraging the linearity of hashing algorithms, we have $Hash(f+\Delta) = Hash(f) + Hash(\Delta)$. This gives that $Hash(\Delta)$ can be used to control the output port offset. However, programmability can still be constrained by the significant heterogeneity of switches. For example, in heterogeneous environments where ToR, leaf, and spine switches employ different hashing algorithms [61], finding a suitable value for the additional header field may become infeasible due to the diversity of hashing functions across switches. For further details, please refer to Appendix A. In summary, RePaC only enables PTC of offsets in limited scenarios. Thus, it is not feasible for production deployment.

A second approach involves changing the hashing algorithm to generate different outputs for different input packets. However, this requires specialized switch chips capable of selecting the hashing algorithm based on a specific header field value in incoming packets. In our data center, only a small fraction of switches support such modifications, making this approach impractical. Additionally, changing the hashing algorithm does not necessarily guarantee that a different output port will be selected, as different algorithms may still produce the same output port for certain inputs (see Appendix B). Even when successful, this method demands extensive computation of all possible input-to-output mappings to achieve the desired network functions by selecting appropriate selectors. Moreover, the range of possible functions is constrained by the limited number of available hash functions.

The final approach involves modifying the mapping stage, which can be done in two ways: (i) redirecting the packet to a different member within the same ECMP group or (ii)

redirecting it to a different ECMP group while keeping the member fixed. However, the first approach faces compatibility issues. Many switches do not support performing arbitrary mathematical operations on the post-processing results; the only feasible modification is to replace the post-processed result with a predefined value (such as selector s). This limitation reduces programmability, allowing only PTC of hops but preventing the use of the hashing process to support PTC of offset. As a result, we discard this approach.

On the other hand, the second approach of redirecting the packet to a different ECMP group is more feasible and aligns well with our goals. Therefore, we choose this method as the foundation for P-ECMP.

3.2 Programming Model and Use Cases

Let $l = [p_0, p_1, \cdots, p_{N-1}]$ represent an ECMP group, where each p_i ($i \in [0, N-1]$) corresponds to an output port, and N is the total number of available ports. For each ECMP group, the port selection for a flow f is determined by a hashing function, $Hash(f)$. P-ECMP generalizes this by using multiple (M) ECMP groups instead of just one, transforming the forwarding table into an ECMP matrix, referred to as the control matrix C, whose dimension is $M \times N$. We assume that users can select a specific ECMP group by embedding a selector s in the packet header, i.e., l_s. If the selector value exceeds the total number of ECMP groups ($s > M$), it will be taken modulo M, i.e., $s \bmod M$. As a result, the port selection for flow f becomes an item selection in the matrix, $C[s \bmod M, Hash(f)]$.

Next, we demonstrate how PTC can be implemented. We categorize two types of PTCs, each associated with a different control matrix C. For simplicity, we will describe each type individually before addressing how to support them concurrently in § 3.3.

Precise traffic control of path offsets. A useful form of PTC involves adding an offset to the existing randomness introduced by ECMP. Specifically, let us assume a default selector s_0 and a custom selector s. The desired property is that the output port selection shifts by a constant offset corresponding to the difference between s_0 and s for a given flow. Formally, this means $F[s, Hash(f)] - F[s_0, Hash(f)] = G(s) - G(s_0)$, where G is a linear function. Taken $G(x) = x$, we have $F[s, Hash(f)] - F[s_0, Hash(f)] = s - s_0$. In this way, the randomness of ECMP is maintained, *i.e.,* with $Hash(f)$ used for column selection, while PTC of path offset is achieved through the selector $s - s_0$.

The control matrix that satisfies this property can be constructed by recursively copying the last available ECMP group and shifting each subsequent group by one position, either to the right or left. For example, if the first ECMP group is $l_0 = [p_0, p_1, \ldots, p_{N-1}]$, the next group would be $l_1 = [p_{N-1}, p_0, \ldots, p_{N-2}]$. This process continues until all N ECMP groups are generated, with the final group being $l_{N-1} = [p_1, p_2, \ldots, p_{N-1}, p_0]$, as illustrated in Figure 3.

The PTC of offset supports various use cases:

- Network failover. Google has proposed randomly re-routing flows upon detecting a network failure signal, such as an RTO [56], by adding a random value in the IPv6 header to alter the ECMP hash outcome. However, this approach is prone to re-pathing failures caused by hash collisions, where the re-pathed flow may still be mapped to the same failed path.

  In contrast, the ability to perform packet forwarding with precise offset control enables deterministic re-pathing, ensuring successful failover. For example, assume the default selector is $s_0 = 0$. When a flow needs to be re-pathed, it can select any s that is not equal to the default selector and does not result in the same path, i.e., $s - s_0 \not\equiv 0 \bmod M$.

  In practice, the value of M (the number of ECMP groups) may not be knowledgeable to applications, and there could be multiple values of M corresponding to different layers of switches in the network. A robust solution is to adopt a sufficiently large prime number for M, which works across any network topology. In most modern data centers, a prime number greater than 16 is typically adequate.

  Another concern is that re-pathed flows should ideally remain load-balanced. We provide a detailed discussion on selector selection to realize this in Appendix C.

- Load imbalance. Similar to network failover, Google proposed PLB [49], which randomly re-path flows to avoid congestion points. In contrast, PTC of offsets allows applications to re-path flows with a specific offset that ensures they will not be mapped to the original path, guaranteeing successful flow re-pathing. The selector value used is the same as in the network failover scenarios.

- Multipath protocols. PTC of offset can also enhance multipath protocols by ensuring that no sub-flows are mapped to the same path, thereby improving resilience against network failures or congestion.

  Unlike flow re-pathing, multipath protocols using P-ECMP should assign increasing integers starting from 0 as the selector s[2]. For instance, if an MPTCP flow creates 4 sub-flows, the selectors s should be $0, 1, 2, 3$, respectively. When all network tiers have at least 4 equal-cost paths, each sub-flow will be mapped to a distinct path with no overlap. However, if a network tier has fewer than 4 available paths, two sub-flows may overlap at one hop, reducing MPTCP's effectiveness. This limitation is not due to P-ECMP but is an inherent consequence of the pigeonhole principle. Determining the optimal number of sub-flows requires analysis of the network topology and workloads, which is beyond the scope of this study.

[2] A native implementation of MPTCP uses random port numbers for its sub-flows, which makes assigning increasing integers starting from 0 as the selector s ineffective. Instead, we have implemented a custom MPTCP protocol where sub-flows are distinguished based on the selector s.

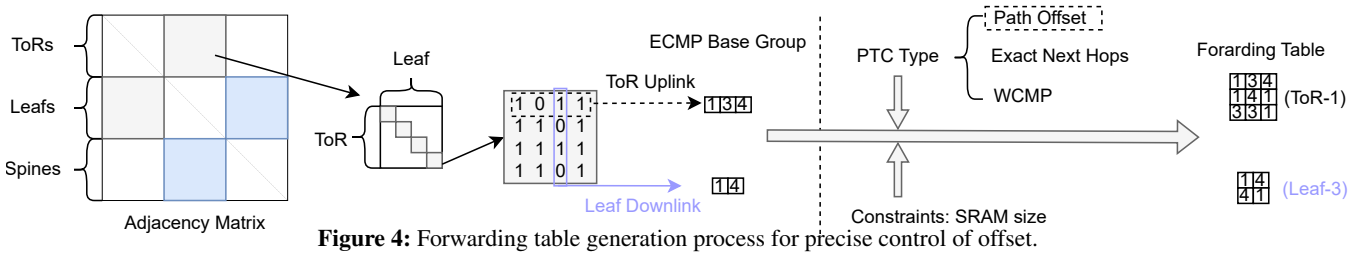


Figure 4: Forwarding table generation process for precise control of offset.

Precise traffic control of exact next hops. P-ECMP can achieve greater precision by removing the hashing process. This can be done by either setting all output ports in an ECMP group to the same port or reducing the number of output ports to just one (i.e., making $l_s = [p_s]$), allowing the output port to be determined entirely by the selector s. At different network tiers, commodity switches can use different packet headers or different bits of the same header field as the selector. As a result, multiple selectors can be configured by the sender, enabling precise control over the entire network path.

This also supports various use cases:

- Failure localization. Existing network failure localization tools, such as Pingmesh [28], need to generate an excessive number of probe packets to cover all paths due to the inherent randomness in ECMP routing. To overcome this limitation, systems like NetBouncer [54] employ IP-in-IP encapsulation to systematically probe all possible paths, providing more accurate and fine-grained failure localization at the cost of incurring additional overhead due to the encapsulation technique. In contrast, P-ECMP offers an optimized alternative for network failure localization, allowing for precise control over all paths without the overhead associated with IP-in-IP encapsulation.
- Packet spraying. Previous approaches to packet spraying have typically followed a random manner [21], relying on ECMP routing. In contrast, P-ECMP enables deterministic packet spraying, enhancing efficiency and fairness beyond what can be achieved through random packet distribution.
- Segment routing. Commodity switches used in today's data centers do not genuinely support segment routing natively, which contributes to its limited adoption. As a workaround, segment routing is often achieved using techniques like IP-in-IP encapsulation [47], though this comes with the tradeoff of increased header size.

  By leveraging the ability to specify the exact output ports with P-ECMP, segment routing can be implemented without the need for IP-in-IP encapsulation. In this approach, the selector s serves as an indicator of the intended path, and together with coordinated forwarding tables across different network tiers realizes the segment routing.

3.3 Compilation Support

To deploy the aforementioned PTC, operators must offload the control matrix as forwarding tables into every switch in the data center network. Manually performing this task would be highly labor-intensive and prone to errors. Instead, we propose automating this process using a compiler. At a high level, the compiler takes as input the type of PTC (as described in § 3.2), the network topology, and the SRAM constraints of the heterogeneous switches. The output of the compiler is the forwarding tables for all switches in the network.

Figure 4 shows the network's adjacency matrix for a topology with switches organized into three tiers: top-of-rack (ToR), leaf, and spine. While P-ECMP can adapt to any topology, here we assume a fat-tree topology with partial connectivity for simplicity. The adjacency matrix is sparse due to the absence of interconnectivity within the same tier (e.g., between ToR switches). We omit sections of the matrix where all values are zero and focus on the portion where ToR switches connect to leaf switches. By scanning each row, we can identify the ECMP base group, representing the available next hops for each ToR switch. In Figure 4, the first ToR switch connects to all leaf switches except the second, resulting in an ECMP base group of $[1,3,4]$ for its uplinks. Similarly, scanning each column reveals the ECMP base group for the leaf switches' downlinks. Note that these numbers represent the global switch IDs across the entire data center. During forwarding table deployment, translating the global switch IDs to the corresponding output ports on each switch is needed.

After determining the ECMP base group for each switch, the next step is to incorporate additional inputs, such as the desired PTC type and the SRAM constraints of each switch, to generate the forwarding table. The process for generating the forwarding table for each PTC type is straightforward, as outlined in § 3.2. The control matrices for different PTC types can be concatenated, with the matrix for precise control of offset appended after the base ECMP group being preferred. This is because the default ECMP group is also part of the control matrix for precise control of offset. We discuss the design of the selector bits in Appendix E.

Note that the PTC of offsets does not necessarily require knowledge of the network topology, whereas the PTC of hops does. When both types of PTC are supported concurrently, servers need to be aware of the network topology and the mappings between selectors and specific PTC policies.

3.4 Update Mechanism

The dynamic nature of data center networks, such as topology changes, requires updates to the forwarding tables in switches.

However, these updates can lead to inconsistencies between the PTC policies in the switches and the host machines, potentially causing packet drops or unintended forwarding behavior for a short period. Given the high bandwidth in data centers – often in the tens or hundreds of Gbps – even brief periods of inconsistency, or no delay but with packets already in transit, can significantly impact data. For instance, at 100 Gbps, a 10 ms disruption may affect 125 MB of data per link. The impact will be significantly larger considering the entire data center network.

To mitigate this issue, we propose a transactional runtime update mechanism that ensures global consistency inspired by previous work [45, 60, 64]. Specifically, we treat the forwarding tables as versioned entities, where each version corresponds to a specific range of selectors. When an update is imposed at runtime, instead of modifying the current version of the forwarding table during an update, a new version is created. This ensures that the existing forwarding table remains unchanged during runtime. Once the new version of the forwarding table is ready, servers update their selectors to align with the new version, guaranteeing a smooth runtime update without forwarding glitches.

In practice, we divide the SRAM space at switches into two symmetric halves, with each half occupying equal space. One half stores the current version of the forwarding table, while the other holds the new version during updates. For instance, if the SRAM can accommodate up to M ECMP groups, ECMP groups 0 to $\lfloor \frac{M-1}{2} \rfloor$ would represent the current version, while groups $\lfloor \frac{M-1}{2} \rfloor + 1$ to $M-1$ would represent the new version. When the new table is in place, servers update their selectors by adding $\lfloor \frac{M}{2} \rfloor$ to adapt to the new version. Once all servers have transitioned, the new version becomes the active one, freeing the 0 to $\lfloor \frac{M-1}{2} \rfloor$ range for the next update cycle.

It is important to note that updates to PTC policies are only necessary for addressing stationary topology changes, such as network expansion, reconstruction, or long-lasting failures and maintenance. These updates are not required for transient failures, such as temporary congestion or brief hardware/software glitches, which are short-lived and self-resolving. As a result, compiling a new version of ECMP groups is not a time-critical task. Nevertheless, our system is capable of compiling the new configurations and offloading them to all switches in the production network within a few milliseconds.

4 Implementation

Switch. We have implemented the aforementioned functionalities across all switches used in our production data centers, primarily including various series of Trident and Tomahawk. Our implementation is built on SONiC [9], where we ensure each operation at a switch – such as adding, replacing, or deleting ECMP groups and their members – is atomic by using locks. Additionally, when making network-wide changes, we utilize the transactional runtime update mechanism (§ 3.4) to maintain consistency across the entire network.

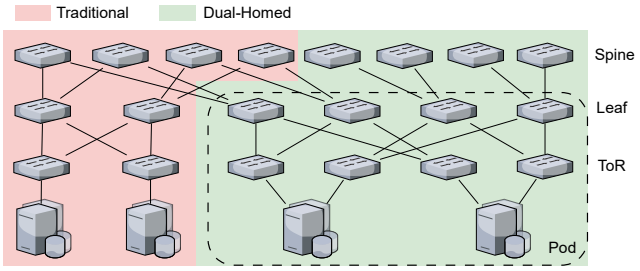


Figure 5: Comparison of traditional and dual-homed data center network architectures.

Host machine. When a host machine needs to invoke the various network functions supported by P-ECMP, such as re-pathing flows after detecting a failure, it populates the selector s in the packet header with the appropriate value corresponding to the desired network function.

Several prior works have explored different methods for encoding the selector s within the packet header. For example, FlowBlender [34] uses the VLAN tag, while PLB [49] and PRR [56] leverage the IPv6 Flow Label. Both of these fields are compatible with P-ECMP. However, in our implementation within the production network (§ 5.3), we avoid both options because (*i*) VLAN tags has been used for other purposes in our production network, and (*ii*) IPv6 Flow Labels are not useful for IPv4 traffic, which is still prevalent on the Internet and in our data centers. Instead, we select the Differentiated Services Code Point (DSCP) in the IP header to encode the selector s value. This is reasonable as DSCP is intended to be used for classifying and managing network traffic, and providing quality of service (QoS) in modern IP networks [27]. it also avoids conflicts with other packet or flow-related operations in our data center. Note that DSCP, VLAN, and Flow Label are not the only viable options – data center operators should select the appropriate header field for carrying the selector s based on their specific environment and requirements.

There are various approaches for modifying the DSCP value. For instance, one can modify the DSCP at the kernel level using eBPF [3] or by introducing a loadable kernel module [4]. Alternatively, one can modify the DSCP at the application level by calling `SetSocketOpt` [8], a function provided by the Unix socket interface, to set the desired value. We adopt the last approach in our implementation.

Note that the focus of this paper is on exploring the programmability of ECMP forwarding on commodity switches, rather than addressing when and how specific network functions should and can be applied, such as how to detect silent packet drops or congestion. For these aspects, we simply rely on state-of-the-art approaches. For instance, in network failover using deterministic re-pathing, we follow PRR [56] by using TCP RTOs as indicators of silent packet drops. For load

balancing, we adopt PLB's approach [49], using a fixed number of packets with ECN marks to detect network congestion. Additionally, our implementation supports application-level signals that can be customized based on user requirements.

Multi-homed server. As a leading cloud provider, we have built and been managing multiple large data centers. Nevertheless, we adopted a different approach when constructing them. One critical revision is that we adopt a dual-homed server bonding architecture where each server connects to two ToR switches. As illustrated in the green section of Figure 5, the numbers of ToR, leaf, and spine switches are hence doubled compared to the standard fat-tree topology. The dual-homed server bonding provides several benefits, including improved network resiliency and reduced downtime. It has also been adopted by other providers recently [48].

Given our dual-homed setup, it is essential for host machines to support similar functionalities provided by P-ECMP-enabled switches across different network interface cards (NICs) as well. A common and convenient practice is to bond multiple NICs to a single logically bonded interface, which allows flows to be distributed across different NICs and paths [5]. However, bonding often relies on hashing algorithms, which introduce randomness as ECMP.

Instead, we adopt a design approach similar to our switch-based solution. Specifically, we add the selector value to the output of the hashing algorithm and perform modulo calculations afterwards. This ensures deterministic path selection and avoids any randomness or the need to modify the hashing algorithm itself. Our approach can be implemented by a hot patch to the NIC bonding driver, making the support for multi-homed networks easy and light.

5 Evaluation

We begin by evaluating the compilation and update mechanism of P-ECMP (§ 5.1). Next, we demonstrate the effectiveness of utilizing P-ECMP to address various critical network corner cases (§ 5.2). Most experiments in § 5.1 and § 5.2 are done using the NS3 simulator [6]. Finally, we share our experience deploying network failover enabled by P-ECMP's PTC of path offset capability in our production network (§ 5.3).

5.1 P-ECMP Compilation and Update

We first present the resource requirements for deploying P-ECMP after compilation. We evaluate multiple topologies, as listed in Table 2, including whether dual-homed architecture is adopted, the number of leaf and spine switches (determined by the product of the number of spines per plane and the number of planes, where the number of planes equals the number of leaf switches per pod), and the resulting maximum number of paths between two servers. Among them, the largest topology #12 is the same as our production network.

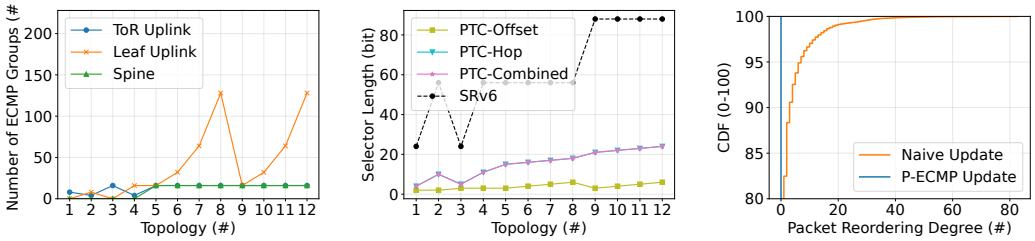
Table 2: Topologies for evaluation. OSR stands for oversubscription ratio.

Topo	Dual-Homed	# Leaf Per Pod	# Spine	OSR	Max # Paths Between Servers
#1	N	4	N/A	N/A	4
#2	Y	4	N/A	N/A	8
#3	N	8	N/A	N/A	8
#4	Y	8	N/A	N/A	16
#5	N	8	8×8	8:1	64
#6	N	8	16×8	4:1	128
#7	N	8	32×8	2:1	256
#8	N	8	64×8	1:1	512
#9	Y	8	8×8	8:1	128
#10	Y	8	16×8	4:1	256
#11	Y	8	32×8	2:1	512
#12	Y	8	64×8	1:1	1,024

Figure 6(a) shows the number of ECMP groups required for P-ECMP across different switches under various topologies, with both PTCs enabled. At the ToR uplink, between 4 and 16 ECMP groups are needed. For leaf switches (uplink), the SRAM resources range from 4 to 128 ECMP groups, except in cases where spine switches are not available, requiring 0 groups. For both ToR and leaf switches, the available SRAM resources are more than sufficient to accommodate P-ECMP. At spine switches, the SRAM resources required range from 4 to 16 ECMP groups per pod. However, it is important to note that this number must be multiplied by $(x-1)$ for a network with x pods. Given that the Trident 5 ASIC provides 8K ECMP groups and accounting for our transactional update mechanism – which halves the available ECMP space – P-ECMP can support networks with up to 512 pods based on ECMP considerations. Nevertheless, the actual number is capped at 128 due to the spine switch's 128-port limit. In other words, P-ECMP's consumption of SRAM is not the bottleneck. Notably, when using either the PTC of offset or hop, the number of required ECMP groups is halved (see Figure 11 in Appendix D).

We move on to the bit length of selector s. As shown in Figure 6(b), when only the PTC of offset is required, the selector ranges from 2 to 6 bits, making a 6-bit DSCP sufficient. If the PTC of path is needed, whether combined with the offset PTC or not, the selector will require more bits. For single-homed topologies (up to topology #8), fewer than 20 bits are needed, making the 20-bit IPv6 FlowLabel a suitable option for carrying the selector. For larger and dual-homed topologies, up to 24 bits may be required, necessitating the use of multiple header fields or other headers not covered in § 4. Nevertheless, compared to SRv6 or IP-in-IP (which incurs even higher overhead than SRv6 but is not depicted in Figure 6(b) for clarity), P-ECMP reduces the header overhead by 3x to 6x.

Next, we evaluate our transactional runtime update mechanism. At each switch, updating ECMP groups typically takes less than 2ms. While updating a single switch is atomic, updating all switches without our transactional update mechanism (§ 3.4) can result in packet reordering due to inconsistencies



(a) Number of ECMP groups. (b) Bit length of selector. (c) Packet reordering degree.

Figure 6: Evaluation of P-ECMP compilation and update mechanism.

across switches and in-flight packets using outdated P-ECMP selectors. We generate a Hadoop workload [10, 51] on topology #4 to assess the packet reordering degree at the receiver, defined as the number of packets arriving earlier or later than their expected sequence position. As illustrated in Figure 6(c), without our transactional update mechanism, the reordering degree can reach up to 83, with a 99th percentile of 19. Depending on the receiver's tolerance for packet reordering, such high levels of reordering may lead to performance degradation, or even packet loss in severe cases.

5.2 Network Functions

Next, we evaluate the support of various network functions by P-ECMP. We use topology #12 for all experiments below.

Network failover. We begin by evaluating the failure recovery of P-ECMP with a single flow. Specifically, we initiate a flow at full rate and introduce a network failure later. We follow PRR [56] to use TCP RTO as the indicator for network failures. We also change the RTO calculation following PRR to better adapt to data center networking scenarios, where RTO $\approx$ RTT + 5ms and has a cap of 1 second. It is noteworthy that the indicator for network failure and the trigger for flow re-path operation is up to user customization, which can react faster or slower than in our experiment.

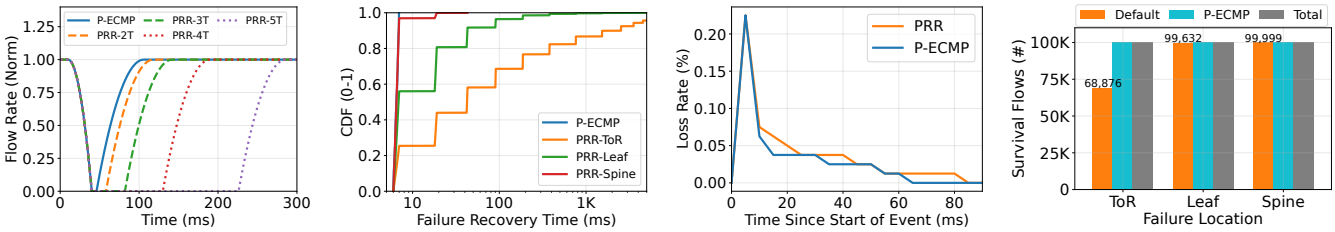
Figure 7(a) compares the performance of P-ECMP to PRR as a representative of random re-pathing. The failure occurs at 10ms. After that, the host takes roughly 6ms to detect the failure and trigger flow re-path. If the first re-path fails, the flow will pause and wait for another RTO, which grows exponentially, before attempting another re-path. In this experiment, P-ECMP succeeds in the first re-path attempt, resulting in full flow recovery at 106ms. However, PRR or other random re-pathing may require two (PRR-2T), three (PRR-3T), four (PRR-4T), or more attempts, resulting in flow recovery at 118ms, 142ms, 190ms, or even later.

We repeat the experiments for 10,000 times. Figure 7(b) presents the CDF of the failover duration for P-ECMP and PRR under different failure scenarios: at the ToR switch (PRR-ToR), at the leaf switch (PRR-Leaf), or at the spine switch (PRR-Spine). The results demonstrate the long-tail distribution of flow recovery duration by PRR (note that X-axis is in log scale). Failures at the ToR switch have the most significant impact on flows, requiring a median recovery duration of 42ms and 95th percentile recovery duration of notably over 4.5s. Failures at the leaf switch have an intermediate impact, with a median recovery duration of 7ms and a 95th percentile recovery duration of 91ms. Note that in practice, failures indeed mostly occur at ToR or leaf layers. Finally, failures at the spine switch have the smallest impact, with only a slight chance ($\sim$3%) of failure during the first re-path attempt. In contrast, P-ECMP always succeeds in re-pathing the flow in the first attempt regardless of the failure location, and hence realizing a stable flow recovery duration of 6ms.

We further present a case study of link failure between a ToR and a leaf switch. Prior to the failure, we initiated data flows of varying sizes mimicking web search workloads [10, 15] from each host machine to randomly selected destination machines. Figure 7(c) illustrates the packet loss rates over time, starting from the onset of the failure. Both PRR and P-ECMP rapidly decrease loss rates, with PRR reducing the loss from over 0.2% to 0% within 85ms. Notably, P-ECMP surpasses PRR in reducing loss rates, achieving failure mitigation within 65ms – a 24% improvement over PRR. Although our current data limitations prevent a similar emulation to more outage events presented in PRR report [56], we expect that P-ECMP will consistently outperform PRR across various scenarios, owing to its PTC capability.

Load imbalance. Next, we evaluate P-ECMP's performance against load imbalance. For each host machine, a destination is randomly chosen, and we generate flows with sizes following web search and Hadoop workload [10, 14, 15, 51]. We compare P-ECMP to PLB [49], a leading load balancing scheme leveraging random flow re-pathing. We incorporate PTC of offset by P-ECMP into the re-pathing phase of PLB [49], and employ the same parameters as PLB.

Table 3 presents the total number of flow re-paths and normalized flow completion time (FCT) results. Our analysis shows that P-ECMP requires fewer re-paths in half of the cases, while in the other half, it incurs a marginally higher number of re-paths (1% to 3%) compared to PLB. This does not indicate that P-ECMP fails to re-path flows on the first attempt; rather, the timing of successful re-paths (e.g., P-ECMP succeeding in the first attempt while PLB requires multiple attempts) can alter the workload distribution across remaining paths and influence the overall network dynamics. What is more important is that P-ECMP achieves significant improvements across various metrics, including p50 FCT, p99



(a) Single-flow recover example. (b) Single-flow recovery duration. (c) Failover from a link failure event. (d) MPTCP enhancement.

Figure 7: Use case evaluation of P-ECMP.

Table 3: Normalized flow completion time with load balancers. LF stands for last flow (*i.e.,* 100th percentile).

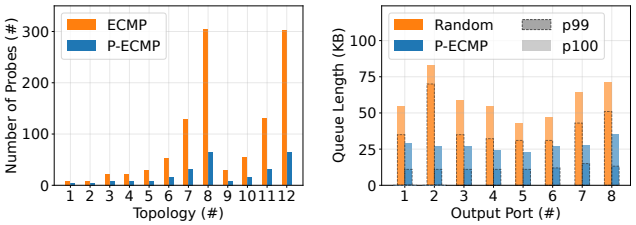
Trace	LB Scheme	30% Load				50% Load				80% Load			
		p50	p99	LF	#RePath	p50	p99	LF	#RePath	p50	p99	LF	#RePath
Web Search	PLB	1	21.5	33.0	1x	1	**22.5**	214.3	1x	1	54.9	366.7	**1x**
	P-ECMP	**0.90**	**8.1**	**13.6**	**0.96x**	**0.91**	23.1	**182.4**	**0.93x**	**0.92**	**45.0**	**189.0**	1.01x
Hadoop	PLB	1	147.2	334.0	**1x**	**1**	212.4	4,700.0	1x	**1**	41.9	1,278.3	**1x**
	P-ECMP	**0.92**	**135.0**	**296.7**	1.02x	1.01	**188.3**	**2,246.3**	**0.97x**	1.01	**40.7**	**769.1**	1.03x

FCT, and last flow (LF) completion time, in most scenarios. Notably, under high network load, P-ECMP nearly halves the last flow completion time, highlighting its effectiveness in addressing the randomness and delays associated with PLB.

Multi-path protocol. We then evaluate P-ECMP's enhancement to the robustness of multi-path protocols. We have implemented a custom MPTCP based on P-ECMP, where subflows are distinguished using the P-ECMP selector. We generate 100K MPTCP flows between two servers, with each MPTCP flow consisting of four subflows, resulting in a total of 400K subflows. A subflow fails if either direction (uplink or downlink) is affected by a switch failure, but an MPTCP flow fails only when all its subflows are lost. Our results show that spine failure barely affects the results. When a leaf switch fails, less than 400 MPTCP flows fail because all their sub-flows are mapped to the same failed leaf switch out of four available leaf switches. When a ToR switch fails, more than half of the MPTCP flows fail because there are only two equal-cost ToR switches. These results match the theoretical expectation[3] and demonstrate the vulnerabilities of multi-path protocols to network failures, especially at the ToR level. In contrast, when P-ECMP is enabled, all MPTCP flows survive regardless of where the failure occurred, demonstrating the effectiveness of P-ECMP in enhancing the robustness of multi-path protocols.

Failure localization. Figure 8(a) illustrates the number of probes required to cover all available paths between a pair of host machines across various topologies. Previous ECMP-based mechanisms (such as Pingmesh [28]) unfortunately require 2x to 5x more probes compared to P-ECMP which enables the PTC of next hops. There are also proposals to use techniques like IP-in-IP to achieve similar functionality.

[3]Assume there are N switches at the network layer where failure occurs. That leaves $N-1$ healthy hops. As mentioned in § 2.1, the probability that a subflow avoids the failure is $\frac{(N-1)^2}{N^2}$, so its failure rate is $1-\frac{(N-1)^2}{N^2}=\frac{2N-1}{N^2}$. Assume there are K subflows. Given that an MPTCP flow fails only if all subflows fail, the survival rate of an MPTCP flow is $1-\frac{(2N-1)^K}{N^{2K}}$.



(a) Failure localization. (b) Packet spray.

Figure 8: More use case evaluation of P-ECMP.

However, these approaches incur significantly higher packet header overhead compared to P-ECMP (see Figure 6(b)).

Packet spraying. We evaluate packet spraying using a 30% Hadoop workload, monitoring ToR switch output queue lengths. Figure 8(b) reveals that random spraying causes uneven packet distribution at a microscopic level, with 99th percentile queue lengths ranging from 31KB to 70KB, likely degrading performance in low-latency environments. This imbalance worsens at the 100th percentile. In contrast, P-ECMP's PTC capability ensures a more balanced queue length distribution across ports, with most queues measuring 11KB at the 99th percentile and a maximum of only 15KB. This demonstrates superior load balancing. Further results confirm that P-ECMP consistently outperforms random spraying in terms of FCT, as detailed in Appendix D.

Segment routing. Finally, we address segment routing. Previous approaches to implementing segment routing on commodity switches typically rely on IP-in-IP or SRv6 protocols, both of which introduce significantly higher (3x-6x) header overhead compared to P-ECMP, as shown in Figure 6(b).

5.3 In Production

Over the past year, we have incrementally deployed P-ECMP's PTC with offset across multiple data centers. The rollout was seamless, with P-ECMP coexisting with existing workloads without disrupting or interfering with live traffic.

Application Recovery. We focus on Cloud Block Storage

Table 4: CBS application performance with P-ECMP (numbers before slashes) and without P-ECMP (numbers after slashes) under different network failures and configurations. #MDS refers to the number of Metadata Service daemons.

Failure Type	#MDS	IO Jitters				IO Hang			
		Occurrence (%)	Min (s)	Med (s)	Max (s)	Occurrence (%)	Min (s)	Med (s)	Max (s)
Leaf Switch Unidirectional Black Hole	0	**100** / 100	**1** / 5	**7** / 7	**9** / 9	**56** / 72	**1** / 1	**2** / 3	**3** / 6
	28	**100** / 100	**6** / 7	**7** / 8	**8** / 11	**60** / 68	**1** / 1	**2** / 3	4 / **3**
	36	**100** / 100	**6** / 9	**7** / 11	**11** / 12	**68** / 70	**1** / 1	**1** / 4	**4** / 6
Leaf Switch Bidirectional Black Hole	28	**100** / 100	**3** / 3	**4** / 4	**6** / 10	**52** / 56	**1** / 1	**3** / 2	**3** / 4
	36	**100** / 100	**3** / 3	**3** / 4	**6** / 8	**44** / 54	**1** / 1	**2** / 2	**4** / 4
ToR Switch Unidirectional Black Hole	28	**100** / 100	**7** / 7	**10** / 15	**15** / 25	**90** / 98	**1** / 1	**5** / 8	**12** / 18

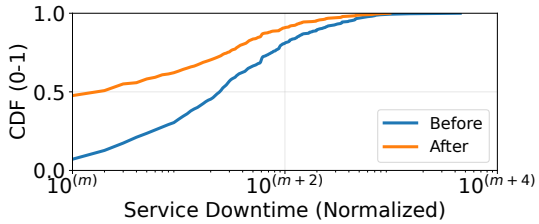


Figure 9: The distribution of service downtime by network failures in our production networks, before and after deploying P-ECMP.

(CBS) to illustrate the effectiveness of P-ECMP. CBS is a high-performance application that transmits large chunks of data across data center storage nodes. CBS may spawn multiple flows at the same time, and the completion of a CBS IO task usually incorporates a series of transport-layer flows. When a network failure occurs, CBS may encounter IO jitters or IO hang, *i.e.,* all the ongoing flows are lost, multiple times through its lifetime as the new flows spawned may be unfortunately mapped to the path with failure.

Before P-ECMP is introduced, the network failure recovery for CBS relies on FullPingmesh, a custom service similar to Pingmesh [28], and a custom socket-level failure detection and reconnection mechanism. FullPingmesh is able to detect and locate network failures within 20s. In addition, our custom socket-level failure detection for CBS can detect potential network failures by sending heartbeat packets and monitoring the socket buffer. When CBS considers that a network failure occurs and starts flow re-pathing by creating new TCP flows with different port numbers. According to our production log, the failure detection takes on average 1s.

Table 4 presents the CBS application performance with and without P-ECMP under different network failures and configurations, *i.e.,* different numbers of Metadata Server (MDS) daemons [1], where more MDS daemons generally help to serve larger workloads [2]. For each configuration and failure type, we repeat the experiments for 50 times. The results show that P-ECMP improves the application performance in most scenarios. Specifically, P-ECMP reduces the IO jitters minimum, median, and maximum durations by up to 80%, 36%, and 40%, respectively. P-ECMP also reduces the occurrence of IO hang by up to 16%, and the recovery duration for IO hang as well. The results demonstrate the effectiveness of P-ECMP in enhancing service availability.

Overall Impact. Our data centers have implemented and integrated multiple standalone solutions to detect and localize network failures, including active and passive probing, as well as data plane telemetry. Figure 9 shows the distribution of service downtime caused by network failures in our data centers. Many network failures can be mitigated within one minute. However, some failures such as hardware failures, take orders of magnitude more – *e.g.,* tens of minutes or even hours – to recover, which falls short of our needs. For example, the switches may encounter optical module failures. This requires the on-site operator to manually inspect and replace the optical module. This may take hours or even days depending on the availability and schedules of the on-site operators. In the worst case, the monthly occurrence of such failures could surpass ten times per pod, lasting for more than 20% of the total operational time.

After deploying P-ECMP in our data centers, the overall downtime due to network failures significantly decreased, as shown in Figure 9. Most downtimes were reduced to within one minute, and tail downtimes were also significantly improved. It is noteworthy that the figure still shows some long downtimes after P-ECMP is deployed. This is because these failures are multi-point or global network failures, *e.g.,* campus electricity outages, and are out of the capacity of P-ECMP or any flow re-pathing signals. Overall, P-ECMP is effective in production and meets our expectations.

6 Discussion

Dual-homed server bonding architecture. We adopted a dual-homed server bonding architecture as in § 4. It is noteworthy that this is not a strict requirement for P-ECMP, as P-ECMP can work with any topology. Instead, the dual-homed server architecture guarantees that a failure-free path is always available in case of any single-point failures in the network, and thus allows obtaining full benefits from P-ECMP. Indeed, when there is only one ToR connected to the host machines, P-ECMP cannot help in the presence of a ToR failure.

PTC policy with asymmetric topologies. When only the PTC of path offset is enabled, P-ECMP functions effectively in any topology, as long as there is more than one equal-cost output port available at each switch (otherwise P-ECMP is not useful). Appendix C details how to select the appropriate selector value for arbitrary topologies. However, enabling PTC of next hops introduces additional complexity. So far, our

discussion is restricted to tree-like topologies. P-ECMP can handle tree-like topologies with varying numbers of switches in different tiers. However, for non-tree-like topologies, such as mesh networks, hypercube topologies, or random graphs, where the concept of "network tiers" may not be as clear, the PTC policy requires more careful design. We leave further exploration of these scenarios to future work.

Interaction between P-ECMP and switch's local failover. For network failover issues, we note that commodity switches are often equipped with built-in local failover mechanisms, such as rapid link failure detection and fast reroute (FRR) protocols [17]. These mechanisms enable switches to quickly reroute traffic to alternative paths when a link or port failure is detected, often within milliseconds. As a result, not all failures require intervention from P-ECMP, which operates as an application-layer failover mechanism. However, certain types of errors fall outside the capabilities of commodity switches' local failover (though programmable switches can resolve some of these issues [23, 55]). Examples include silent packet drops and gray failures [31, 32], which are difficult to detect and mitigate using traditional failover mechanisms. Furthermore, P-ECMP addresses a broader range of scenarios beyond failover, such as congestion avoidance and load balancing. By operating at the application layer, P-ECMP complements the switch's local failover and provides a more comprehensive solution for ensuring network reliability and performance.

7 Related Work

ECMP and host-based path control. To defend against the randomness of ECMP and enable host-based path control, previous studies have mainly focused on removing ECMP, typically by redesigning or using different routing protocols [30, 33, 54]. For instance, XPath [30] identifies all end-to-end paths, compresses them, and pre-installs them into the IP TCAM tables of commodity switches. However, this approach requires a large number of forwarding table entries, which becomes impractical at the scale of data centers. More recently, programmable switches provides another solution for explicit host control. Nevertheless, their deployment in production networks remains limited so far.

Other studies attempt to work within the constraints of ECMP. For example, Volur [65] uses a centralized controller to collect routing information from switches, predict network behaviors, and send these predictions to servers. However, this approach faces two major issues: limited scalability and inaccurate predictions. RePaC [66] leverages the linearity of hashing functions to achieve relative path control. Unfortunately, this method assumes uniform switch deployment, making it impractical in production environments with diverse switches and hashing functions.

Our approach, P-ECMP, aligns more closely with the second group of studies. However, we leverage a different feature – additional ECMP groups – to extend ECMP's capabilities, allowing for precise traffic control. P-ECMP requires acceptable resources at switches and adapts to heterogeneous switch environments. Notably, P-ECMP has been successfully deployed in production.

Data center network failure. Existing proposals for network failure detection are mostly based on active or passive probing [11, 19, 20, 28, 42, 63], or data plane telemetry monitoring and analysis [18, 22, 25, 38, 41, 46, 52, 68, 69]. After detecting and locating the network failures, data center operators then need to mitigate the failures. Various automated failover solutions have been proposed based on SDN [36, 37, 43, 53, 58, 59, 67].

Nevertheless, the first-detection-then-mitigation fashion of failover is inherently slow. To speed up this process, another line of work proposes to mitigate the failures before accurately locating them. For instance, NetPilot [57] proposes to rely on the data center redundancy and blindly apply mitigation methods. Fast failovers can also be done by the host machines alone. For example, MPTCP [16] or MPRDMA [40] can be employed to address the network failures in data centers [50] by actively adjusting its sub-flows to circumvent network failures. In single flow scenarios, PRR circumvents the failure by re-pathing the flow leveraging the ECMP capability of data center networks [56]. Nevertheless, all the above solutions are vulnerable to hashing collisions. P-ECMP enhances fast failover, multipath protocols, and many other network functions by enabling precise traffic control.

8 Conclusion

This paper introduces programmable ECMP (P-ECMP), a programming model that achieves precise traffic control (PTC) by reconfiguring ECMP groups on commodity switches. P-ECMP enables PTC for both path offset and exact next hops, extending its support to multiple use cases, from network failover to segment routing, without disrupting the majority of traffic flowing through standard ECMP forwarding. Our evaluations demonstrate that P-ECMP requires minimal resources at switches and outperforms several existing approaches across various use cases. P-ECMP has been deployed in production for network failover, where it has proven effective in significantly reducing failover duration.

Acknowledgements

We sincerely thank our shepherd Kaihui Gao and anonymous reviewers for their insightful comments. We also thank software engineers at Tencent for their support in deploying P-ECMP in production. Jilong Wang was supported by the National Key Research and Development Program of China under Grant No. 2020YFE0200500. Congcong Miao is the corresponding author.

References

[1] ceph-mds – ceph metadata server daemon. https://docs.ceph.com/en/latest/man/8/ceph-mds/.

[2] Chapter 2. Configuring Metadata Server Daemons. https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/3/html/ceph_file_system_guide/configuring-metadata-server-daemons.

[3] eBPF - Introduction, Tutorials & Community Resources. https://ebpf.io/.

[4] modprobe(8) - Linux man page. https://linux.die.net/man/8/modprobe.

[5] networking:bonding. The Linux Foundation DokuWiki. https://wiki.linuxfoundation.org/networking/bonding.

[6] ns-3 | a discrete-event network simulator for internet systems. https://www.nsnam.org/.

[7] opencomputeproject/SAI. https://github.com/opencomputeproject/SAI.

[8] setsockopt(3) - Linux man page. https://linux.die.net/man/3/setsockopt.

[9] Software for Open Networking in the Cloud (SONiC). https://github.com/anish-n/SONiC/.

[10] Traffic Generator. HPCC-PINT/traffic_gen. https://github.com/ProbabilisticINT/HPCC-PINT/tree/master/traffic_gen.

[11] A. Adams, P. Lapukhov, and J. H. Zeng. Netnorad: Troubleshooting networks via end-to-end probing. *Facebook White Paper*, 2016.

[12] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM'08, Seattle, WA, USA, August 17-22, 2008*, pages 63–74. ACM, 2008.

[13] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese. CONGA: distributed congestion-aware load balancing for datacenters. In *Proceedings of ACM SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014*, pages 503–514. ACM, 2014.

[14] M. Alizadeh, A. G. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *Proceedings of ACM SIGCOMM'10, New Delhi, India, August 30 -September 3, 2010*, pages 63–74. ACM, 2010.

[15] R. B. Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher. PINT: probabilistic in-band network telemetry. In *Proceedings of ACM SIGCOMM'20, Virtual Event, USA, August 10-14, 2020*, pages 662–680. ACM, 2020.

[16] O. Bonaventure, C. Paasch, and G. Detal. Use cases and operational experience with multipath TCP. *RFC*, 8041:1–30, 2017.

[17] Cisco. IPv4 Loop-Free Alternate Fast Reroute. https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_pi/configuration/15-s/iri-15-s-book/iri-ip-lfa-frr.pdf.

[18] B. Claise. Cisco systems netflow services export version 9. *RFC*, 3954:1–33, 2004.

[19] A. Dhamdhere, D. D. Clark, A. Gamero-Garrido, M. J. Luckie, R. K. P. Mok, G. Akiwate, K. Gogia, V. Bajpai, A. C. Snoeren, and K. Claffy. Inferring persistent interdomain congestion. In *Proceedings of ACM SIGCOMM'18, Budapest, Hungary, August 20-25, 2018*, pages 1–15. ACM, 2018.

[20] A. Dhamdhere, R. Teixeira, C. Dovrolis, and C. Diot. Netdiagnoser: troubleshooting network unreachabilities using end-to-end probes and routing data. In *Proceedings of the 2007 ACM Conference on Emerging Network Experiment and Technology, CoNEXT 2007, New York, NY, USA, December 10-13, 2007*, page 18. ACM, 2007.

[21] A. A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella. On the impact of packet spraying in data center networks. In *Proceedings of the IEEE INFOCOM 2013, Turin, Italy, April 14-19, 2013*, pages 2130–2138. IEEE, 2013.

[22] B. Eriksson, P. Barford, R. A. Bowden, N. G. Duffield, J. Sommers, and M. Roughan. Basisdetect: a model-based network event detection framework. In *Proceedings of the 10th ACM SIGCOMM Internet Measurement Conference, IMC 2010, Melbourne, Australia - November 1-3, 2010*, pages 451–464. ACM, 2010.

[23] K. Gao, S. Wang, K. Qian, D. Li, R. Miao, B. Li, Y. Zhou, E. Zhai, C. Sun, J. Gao, D. Zhang, B. Fu, F. Kelly, D. Cai, H. H. Liu, Y. Li, H. Yang, and T. Sun. Dependable virtualized fabric on programmable data plane. *IEEE/ACM Trans. Netw.*, 31(4):1748–1764, 2023.

[24] Y. Geng, V. Jeyakumar, A. Kabbani, and M. Alizadeh. Juggler: a practical reordering resilient network stack for datacenters. In *Proceedings of the Eleventh European Conference on Computer Systems, EuroSys 2016, London, United Kingdom, April 18-21, 2016*, pages 20:1–20:16. ACM, 2016.

[25] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *Proceedings of the ACM SIGCOMM'11, Toronto, ON, Canada, August 15-19, 2011*, pages 350–361. ACM, 2011.

[26] A. G. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *Proceedings of ACM SIGCOMM'09, Barcelona, Spain, August 16-21, 2009*, pages 51–62. ACM, 2009.

[27] D. Grossman. New terminology and clarifications for diffserv. *RFC*, 3260:1–10, 2002.

[28] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. A. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z. Lin, and V. Kurien. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proceedings of ACM SIGCOMM'15, London, United Kingdom, August 17-21, 2015*, pages 139–152. ACM, 2015.

[29] K. Hsu, P. Tammana, R. Beckett, A. Chen, J. Rexford, and D. Walker. Adaptive weighted traffic splitting in programmable data planes. In *SOSR '20: Symposium on SDN Research, San Jose, CA, USA, March 3, 2020*, pages 103–109. ACM, 2020.

[30] S. Hu, K. Chen, H. Wu, W. Bai, C. Lan, H. Wang, H. Zhao, and C. Guo. Explicit path control in commodity data centers: Design and applications. In *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA, May 4-6, 2015*, pages 15–28. USENIX Association, 2015.

[31] P. Huang, C. Guo, J. R. Lorch, L. Zhou, and Y. Dang. Capturing and enhancing in situ system observability for failure detection. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 1–16. USENIX Association, 2018.

[32] P. Huang, C. Guo, L. Zhou, J. R. Lorch, Y. Dang, M. Chintalapati, and R. Yao. Gray failure: The achilles' heel of cloud-scale systems. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems, HotOS 2017, Whistler, BC, Canada, May 8-10, 2017*, pages 150–155. ACM, 2017.

[33] S. A. Jyothi, M. Dong, and B. 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, Santa Clara, California, USA, June 17-18, 2015*, pages 10:1–10:8. ACM, 2015.

[34] A. Kabbani, B. Vamanan, J. Hasan, and F. 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 2014, Sydney, Australia, December 2-5, 2014*, pages 149–160. ACM, 2014.

[35] Q. Kang, L. Xue, A. Morrison, Y. Tang, A. Chen, and X. Luo. Programmable in-network security for context-aware BYOD policies. In *USENIX Security 2020, August 12-14, 2020*, pages 595–612. USENIX Association, 2020.

[36] J. Li, J. Hyun, J. Yoo, S. Baik, and J. W. Hong. Scalable failover method for data center networks using openflow. In *2014 IEEE Network Operations and Management Symposium, NOMS 2014, Krakow, Poland, May 5-9, 2014*, pages 1–6. IEEE, 2014.

[37] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. A. Maltz. zupdate: updating data center networks with zero loss. In *Proceedings of ACM SIGCOMM'13, Hong Kong, August 12-16, 2013*, pages 411–422. ACM, 2013.

[38] V. Liu, D. Halperin, A. Krishnamurthy, and T. E. Anderson. F10: A fault-tolerant engineered network. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013, Lombard, IL, USA, April 2-5, 2013*, pages 399–412. USENIX Association, 2013.

[39] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016*, pages 101–114. ACM, 2016.

[40] Y. Lu, G. Chen, B. Li, K. Tan, Y. Xiong, P. Cheng, J. Zhang, E. Chen, and T. Moscibroda. Multi-path transport for RDMA in datacenters. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018*, pages 357–371. USENIX Association, 2018.

[41] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, B. Godfrey, and S. T. King. Debugging the data plane with anteater. In *Proceedings of the ACM SIGCOMM'11, Toronto, ON, Canada, August 15-19, 2011*, pages 290–301. ACM, 2011.

[42] P. Marchetta, A. Botta, E. Katz-Bassett, and A. Pescapè. Dissecting round trip time on the slow path with a single packet. In *Passive and Active Measurement - 15th International Conference, PAM 2014, Los Angeles, CA, USA,*

March 10-11, 2014, Proceedings, volume 8362 of *Lecture Notes in Computer Science*, pages 88–97. Springer, 2014.

[43] C. Miao, Y. Xiao, M. Canini, R. Dai, S. Zheng, J. Wang, J. Bu, A. Kuzmanovic, and Y. Wang. TENSOR: lightweight BGP non-stop routing. In *Proceedings of ACM SIGCOMM'23, New York, NY, USA, 10-14 September 2023*, pages 108–121. ACM, 2023.

[44] C. Miao, Z. Zhong, Y. Xiao, F. Yang, S. Zhang, Y. Jiang, Z. Bai, C. Lu, J. Geng, Z. He, Y. Wang, X. Zou, and C. Yang. Megate: Extending WAN traffic engineering to millions of endpoints in virtualized cloud. In *Proceedings of ACM SIGCOMM'24, Sydney, NSW, Australia, August 4-8, 2024*, pages 103–116. ACM, 2024.

[45] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of ACM SIGCOMM'17, Los Angeles, CA, USA, August 21-25, 2017*, pages 15–28. ACM, 2017.

[46] T. Mizrahi, G. Navon, G. Fioccola, M. Cociglio, M. G. Chen, and G. Mirsky. AM-PM: efficient network telemetry using alternate marking. *IEEE Netw.*, 33(4):155–161, 2019.

[47] C. E. Perkins. IP encapsulation within IP. *RFC*, 2003:1–14, 1996.

[48] K. Qian, Y. Xi, J. Cao, J. Gao, Y. Xu, Y. Guan, B. Fu, X. Shi, F. Zhu, R. Miao, C. Wang, P. Wang, P. Zhang, X. Zeng, E. Ruan, Z. Yao, E. Zhai, and D. Cai. Alibaba HPN: A data center network for large language model training. In *Proceedings of ACM SIGCOMM'24, Sydney, NSW, Australia, August 4-8, 2024*, pages 691–706. ACM, 2024.

[49] M. A. Qureshi, Y. Cheng, Q. Yin, Q. Fu, G. Kumar, M. Moshref, J. Yan, V. Jacobson, D. Wetherall, and A. Kabbani. PLB: congestion signals are simple and effective for network load balancing. In *Proceedings of ACM SIGCOMM'22, Amsterdam, The Netherlands, August 22 - 26, 2022*, pages 207–218. ACM, 2022.

[50] C. Raiciu, S. Barré, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving datacenter performance and robustness with multipath TCP. In *Proceedings of the ACM SIGCOMM'11, Toronto, ON, Canada, August 15-19, 2011*, pages 266–277. ACM, 2011.

[51] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the social network's (datacenter) network. In *Proceedings of ACM SIGCOMM'15, London, United Kingdom, August 17-21, 2015*, pages 123–137. ACM, 2015.

[52] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen. csamp: A system for network-wide flow monitoring. In *5th USENIX Symposium on Networked Systems Design & Implementation, NSDI 2008, April 16-18, 2008, San Francisco, CA, USA, Proceedings*, pages 233–246. USENIX Association, 2008.

[53] B. E. Stephens and A. L. Cox. Deadlock-free local fast failover for arbitrary data center networks. In *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*, pages 1–9. IEEE, 2016.

[54] C. Tan, Z. Jin, C. Guo, T. Zhang, H. Wu, K. Deng, D. Bi, and D. Xiang. Netbouncer: Active device and link failure localization in data center networks. In *16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, Boston, MA, February 26-28, 2019*, pages 599–614. USENIX Association, 2019.

[55] S. Wang, K. Gao, K. Qian, D. Li, R. Miao, B. Li, Y. Zhou, E. Zhai, C. Sun, J. Gao, D. Zhang, B. Fu, F. Kelly, D. Cai, H. H. Liu, and M. Zhang. Predictable vfabric on informative data plane. In *Proceedings of ACM SIGCOMM'22, Amsterdam, The Netherlands, August 22 - 26, 2022*, pages 615–632. ACM, 2022.

[56] D. Wetherall, A. Kabbani, V. Jacobson, J. Winget, Y. Cheng, C. B. M. III, U. Moravapalle, P. Gill, S. Knight, and A. Vahdat. Improving network availability with protective reroute. In *Proceedings of ACM SIGCOMM'23, New York, NY, USA, 10-14 September 2023*, pages 684–695. ACM, 2023.

[57] X. Wu, D. Turner, C. Chen, D. A. Maltz, X. Yang, L. Yuan, and M. Zhang. Netpilot: automating datacenter network failure mitigation. In *Proceedings of ACM SIGCOMM'12, Helsinki, Finland - August 13 - 17, 2012*, pages 419–430. ACM, 2012.

[58] Y. Wu, A. Chen, A. Haeberlen, W. Zhou, and B. T. Loo. Automated network repair with meta provenance. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks, Philadelphia, PA, USA, November 16 - 17, 2015*, pages 26:1–26:7. ACM, 2015.

[59] Y. Wu, A. Chen, A. Haeberlen, W. Zhou, and B. T. Loo. Automated bug removal for software-defined networks. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 719–733. USENIX Association, 2017.

[60] J. Xing, K. Hsu, M. Kadosh, A. Lo, Y. Piasetzky, A. Krishnamurthy, and A. Chen. Runtime programmable switches. In *19th USENIX Symposium on Networked*

Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022, pages 651–665. USENIX Association, 2022.

[61] Y. Xu, K. He, R. Wang, M. Yu, N. Duffield, H. M. G. Wassel, S. Zhang, L. Poutievski, J. Zhou, and A. Vahdat. Hashing design in modern networks: Challenges and mitigation techniques. In *Proceedings of the 2022 USENIX Annual Technical Conference, USENIX ATC 2022, Carlsbad, CA, USA, July 11-13, 2022*, pages 805–818. USENIX Association, 2022.

[62] Z. Yu, Y. Zhang, V. Braverman, M. Chowdhury, and X. Jin. Netlock: Fast, centralized lock management using programmable switches. In *Proceedings of ACM SIGCOMM'20, Virtual Event, USA, August 10-14, 2020*, pages 126–138. ACM, 2020.

[63] H. Zeng, R. Mahajan, N. McKeown, G. Varghese, L. Yuan, and M. Zhang. Measuring and troubleshooting large operational multipath networks with gray box testing. *Mountain Safety Res., Seattle, WA, USA, Rep. MSR-TR-2015-55*, 2015.

[64] H. Zhang, A. Cardoza, P. B. Chen, S. Angel, and V. Liu. Fault-tolerant and transactional stateful serverless workflows. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*, pages 1187–1204. USENIX Association, 2020.

[65] Q. Zhang, D. Zhuo, V. Liu, P. Lapukhov, S. Peter, A. Krishnamurthy, and T. E. Anderson. Volur: Concurrent edge/core route control in data center networks. *CoRR*, abs/1804.06945, 2018.

[66] Z. Zhang, H. Zheng, J. Hu, X. Yu, C. Qi, X. Shi, and G. Wang. Hashing linearity enables relative path control in data centers. In *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*, pages 855–862. USENIX Association, 2021.

[67] W. Zhou, J. Croft, B. Liu, and M. Caesar. Neat: Network error auto-correct. In *Proceedings of the Symposium on SDN Research, SOSR 2017, Santa Clara, CA, USA, April 3-4, 2017*, pages 157–163. ACM, 2017.

[68] Y. Zhou, C. Sun, H. H. Liu, R. Miao, S. Bai, B. Li, Z. Zheng, L. Zhu, Z. Shen, Y. Xi, P. Zhang, D. Cai, M. Zhang, and M. Xu. Flow event telemetry on programmable data plane. In *Proceedings of ACM SIGCOMM'20, Virtual Event, USA, August 10-14, 2020*, pages 76–89. ACM, 2020.

[69] Y. Zhu, N. Kang, J. Cao, A. G. Greenberg, G. Lu, R. Mahajan, D. A. Maltz, L. Yuan, M. Zhang, B. Y. Zhao, and H. Zheng. Packet-level telemetry in large datacenter networks. In *Proceedings of ACM SIGCOMM'15, London, United Kingdom, August 17-21, 2015*, pages 479–491. ACM, 2015.

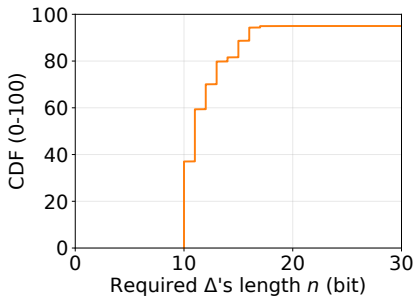


Figure 10: Required Δ's length to cover 64 paths.

A Evaluation of RePaC In Production

RePaC [66] introduces modifications to header fields by XORing a n-bit Δ. This generates a *pathmap* of 2^n entries where each possible Δ corresponds to an offset $Hash(\Delta)$. The output offset is subject to the number of available next hops. Assume the number of next hops is 2^m, then $Hash(\Delta)$ has a length of m bits. Let O be the matrix of all the outputs with n-bit Δ as input, where each $O_{i,j}$ represents the value of jth index of $Hash(\Delta_i)$ for $i \in [1, 2^n]$. To control the offset across all possible alternative paths, it requires $\text{rank}(O) = m$, meaning the hash results of all n-bit Δ must cover all 2^m possible paths. When there are multiple K layers of switches in the network with $2^{m_1}, 2^{m_2}, \ldots, 2^{m_k}$ switches per layer, then it requires $\text{rank}(O) = \sum_{i=1}^{K} m_i$, ensuring coverage of all possible path offsets across all switch layers.

We evaluated this requirement in a heterogeneous environment using all commonly used hash algorithms in commodity switches, including 8 variants of CRC-8, 8 variants of CRC-16, and 8 variants of CRC-32. We used topology #12, which mirrors our production network, with the same type of switches within each layer (e.g., ToR, leaf, or spine) but different switches across layers. We tested all permutations of hash algorithms across layers, totaling permutation(24,3)=12,144 settings, with different hashing algorithms assigned to different network layers.

Figure 10 shows the CDF of the required Δ's bit length n to cover all available paths across three network layers. The minimum bit length of Δ is 10 (1 bit needed for 2 ToR switches per server, 3 bits for 8 leaf switches per pod, and 6 bits for 64 spine switches). While 20 bits suffice in most settings, in 608 out of 12,144 (roughly 5%) switch permutations, no suitable Δ exists to cover all paths (tested with up to 1024-bit inputs, *i.e.,* $n = 1024$). In other words, RePaC fails if the deployed switches use hashing algorithms from these 608 settings. If switches are purchased randomly, there is a 5% chance that RePaC will not work. Moreover, if heterogeneity across pods

Table 5: Hashing collision rate between different hashing algorithms. Three numbers in each cell represent collision rates when there are 2, 4, and 8 equal-cost next hops, respectively.

Collison Rate (%)	Pearson	CRC	XOR	Random
Random	50.00 / 25.00 / 12.50	49.99 / 25.00 / 12.49	50.01 / 24.98 / 12.50	-
XOR	50.00 / 25.00 / 12.50	50.00 / 25.00 / 12.50	-	
CRC	50.01 / 24.97 / 12.50	-		
Pearson	-			

Table 6: The obtained maximum path load using selected selector set and the differences between it and the optimal (D.O.) or P-ECMP-allowable optimal (D.D.O) loads for an arbitrary network with a maximum of 4 equal-cost next hops.

$n\backslash p$	5	7	Maximum Path Load (%) Obtained	D.O.	D.D.O.
2	1	1	50	0	0
3	2	1	66.67	0	0
4	1	3	66.67	8.33	0

is considered, it will require longer Δ and increase the number of switch settings where no suitable inputs exist.

B Alternative of Changing Hashing Algorithm

We investigate the four most common hashing algorithms used in ECMP [66]: random-based, XOR, CRC, and Pearson hashing. The input to the hashing algorithms in ECMP is usually the five-tuple <Src_IP, Dst_IP, Src_Port, Dst_Port, Proto>, which includes 104 bits. However, given the huge input space, *i.e.,* 2^{96} possibilities for 104-bit inputs, we only select a random set of four-tuple inputs. We then feed them into the hashing algorithms respectively, and check whether the outputs would be the same. If two hashing algorithms produce identical outputs for the same input, switching between them would not alter path selection, rendering it incapable of achieving PTC of offset.

Table 5 illustrates the probabilities that two different hashing algorithms produce the same outputs for a random set of five-tuple inputs, *i.e.,* the re-pathing failure rates. The results show that the re-pathing failure rates are substantially high – 50%, 25%, and 12.5% when there are 2, 4, and 8 equal-cost next hops, respectively. This means that the PTC of offset cannot be guaranteed by swamping among any of these common hashing algorithms.

C Load balance After Flow Re-Path

Here, we discuss how to maintain load balancing for flows after they are re-pathed for failover, leveraging P-ECMP's PTC

Table 7: The obtained maximum path load using selected selector set and the differences between it and the optimal (D.O.) or P-ECMP-allowable optimal (D.D.O) loads for an arbitrary network with a maximum of 8 equal-cost next hops.

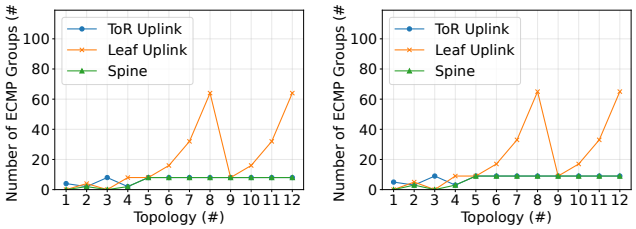
$n\backslash p$	11	13	17	19	23	29	Maximum Path Load (%) Obtained	D.O.	D.D.O.
2	1	1	1	1	1	1	50	0	0
3	2	1	2	1	2	2	60	6.67	6.67
4	3	1	1	3	3	1	66.67	8.33	0
5	1	3	2	4	3	4	75	5	5
6	5	1	5	1	5	5	60	23.33	6.67
7	4	6	3	5	2	1	85.71	0	0
8	3	5	1	3	7	5	75	12.5	5

of offset capability. We assume that the PTC of next hops is not simultaneously supported, and that servers have limited knowledge of the network topology. Otherwise, servers could simply select from the remaining available paths evenly, ensuring that all re-pathed flows are distributed uniformly across the paths.

First, if the bit efficiency of the selector s is crucial and the network topology follows a symmetric Clos-like structure – which is adopted by most data centers –, the servers should select selector s from the set of positive odd numbers less than the largest ECMP group size in the network. For instance, if the spine tier has 8 switches, then the selector s should be selected from the set $\{1,3,5,7\}$ with equal probability. In this case, selector s needs only three bits, providing good bit efficiency. It also guarantees that the load will be balanced across half of the remaining available switches in each network tier. It is important to only select odd numbers because the other network tiers may contain switches of even numbers, *e.g.,* 2 or 4, and selecting an even number could result in failed re-pathing at some network tiers.

Second, if the network topology is asymmetric/less structured, the host should select selector s from at most $N-1$ prime numbers greater than N, where N is the largest ECMP group size in the network. For example, if $N=8$, then the selector s should be selected from the set $\{11,13,17,19,23,29\}$ with equal probability. In this case, selector s occupies 5 bits, providing the capability to avoid any single-point failures and *good* load-balancing performance on most network tiers with a size equal to or less than N. We detail more analysis below.

We assume an arbitrary ECMP network where the largest ECMP group size is N. If the ability to circumvent single-point network failure is desired, then it means that there might be network tiers with the ECMP group size n to range from 2 to N, *i.e.,* $n \in [2,N]$. In such a network, our goal is to find a set of selector S that satisfies two conditions: (i) the re-pathing succeeds circumventing the network failure on the old path, *i.e.,* $\forall s \in S, p \bmod n \neq 0$; and (ii) good load balance is achieved after the flow re-pathing. Note that the minimum ECMP group size is 2 because we assume a dual-homed architecture (§ 4). If a traditional fat-tree topology is adopted,



(a) PTC of offset only. (b) PTC of hop only.

Figure 11: Number of ECMP groups with either PTC type.

the minimum ECMP group size is 1 at the ToR switch tier, meaning that flow re-pathability cannot ensure a successful failover from a ToR switch failure or link failure between the server and the ToR switch.

To quantify the second condition, let $D_n = \{s \bmod n, \forall s \in S\}, \forall n \in [2,N]$. A perfect load balance after flow re-pathing based on S means that $\forall n \in [2,N], D_n$ needs to be perfectly balanced, *i.e.,* every possible element has the same frequency. This might involve a large selector set S. In fact, the minimum size of S to achieve perfect load balance is the least common multiple of all prime numbers ranging from 2 to N. For example, with $N = 8$, the minimum $|S| = 2 \times 3 \times 5 \times 7 = 210$. To achieve a perfect balance an ECMP group of size n requires the number of selectors $|S|$ to be the product of its group size and any positive integer, *i.e., kn* where $k \in \mathbb{Z}^+$. If n is not a prime number, then it is the product of several prime numbers smaller than it. As long as $|S|$ can achieve perfect balance for its prime factors, $|S|$ can achieve perfect balance for n.

To find the set S for perfect load balance, one should start with the set of smallest prime numbers that are greater than N, *e.g.,* $\{11,13,17,\ldots\}$ if $N = 8$. This set should contain $|S|$ prime numbers. It follows that $\forall n \in [2,N], D_n$ can be calculated accordingly. If D_n is not balanced, then one should adjust the prime numbers in the set. The above process should be repeated until S is found. Some intuition might be helpful to optimize the prime number adjusting, but we do not go into details here as we do not deem the perfect load balance after flow re-pathing to be necessary given its high cost.

Rather, it may be more practical to target a smaller selector set to ensure *satisfactory* post-re-pathing load balance, rather than striving for perfection. This trade-off allows for a "good enough" load balance outcome while mitigating the high costs associated with the above prime number searching.

To achieve this objective, we propose a straightforward approach: simply select the subsequent $N_1 - 1$ prime numbers that are greater than N for the selector set, where N_1 represents the largest prime number less than or equal to N. For instance, when N is 8, the highest prime number fitting this criterion is $N_1 = 7$. Consequently, we arrive at $S = \{11,13,17,19,23,29\}$ where $|S| = 6$.

Assuming that load balance is established prior to the occurrence of any failure — meaning that all paths have identical loads — we define the degree of post-re-pathing load balance by examining the maximum load on each path. Employing the example provided, $D_4 = \{3,1,1,3,3,1\}$. In the scenario where one path becomes unavailable, its load is evenly distributed across two other paths. This results in the maximum load on each path reaching 66.67%.

For a state of perfect load balance, implying that the load on the failed path can be evenly redistributed across all other paths, the maximum load on each path can be up to 75%. Consequently, the disparity between our solution and the optimal path load equates to 8.33%.

However, it is crucial to acknowledge that the value 2 can never be attained via modulo 4 with any prime numbers. Consequently, we introduce the concept of P-ECMP-allowable degree of load balance by eliminating paths that cannot be derived through the modulo operation involving prime numbers. Returning to the aforementioned case, two paths can be re-pathed using P-ECMP. Hence, the difference between our solution and the optimal P-ECMP-allowable path load is 0%.

Table 6 and 7 illustrate the post-re-pathing load balance where N is 4 and 8, respectively. Overall, we find that the above straightforward approach demonstrates a good enough post-re-pathing load balance that meets our use case requirements adequately, with the disparity between the obtained and the optimal P-ECMP-allowable maximum path load remaining under 6.67%.

It is noteworthy that unlike RePaC which requires a central server to calculate the comprehensive path map and subsequently distribute it across all data center servers, the algorithm outlined above for ascertaining the optimal selector set S can be executed on any local server. Furthermore, each local server retains the freedom to determine its own desired properties, such as pursuing a state of perfect load balance or settling for a reasonably good load balance. More crucially, RePaC struggles to address scenarios involving an asymmetric network topology. In contrast, P-ECMP demonstrates the capacity to handle arbitrary network topologies and offers varying levels of load balance that can be fine-tuned against the size of the selector set.

D Additional Evaluation of P-ECMP

Figure 11 presents the number of ECMP groups with either PTC type.

Table 8 shows the normalized FCT with random packet spray and with P-ECMP.

E Selector Length

We examine the length of the selector when supporting either PTC type or both concurrently. For the sake of simplicity, we assume a small topology with one ToR switch connected to each server, four leaf switches, and two spine switches. Figure 12 shows the ECMP groups and members when different PTCs are supported.

Table 8: Normalized flow completion time with packet spray. LF stands for last flow (*i.e.,* 100th percentile).

Trace	LB Scheme	30% Load			50% Load			80% Load		
		p50	p99	LF	p50	p99	LF	p50	p99	LF
Web Search	Random Packet Spray	1	11.4	45.7	1	57.8	412.8	1	54.8	1,155.4
	P-ECMP	**0.98**	**10.0**	**45.1**	**0.84**	**56.0**	**396.7**	**0.98**	**47.5**	**1,071.8**
Hadoop	Random Packet Spray	1	57.8	412.8	1	166.5	2,062.1	1	27.8	475.2
	P-ECMP	**0.84**	**56.0**	**396.7**	**0.81**	**151.2**	**2,049.0**	**0.99**	**27.1**	**340.5**

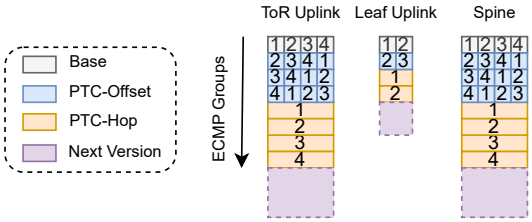


Figure 12: Forwarding table design.

When only the PTC of offset is required, the selector can range from 0 to 3 (for more details, see Appendix C). This value is effective across all network hops. For example, when $s = 1$, the flow will be re-pathed through a different leaf, spine, and another leaf switch in a different pod. As a result, the selector length is 2 bits. Our transactional update mechanism (§ 3.3) doubles the required space, increasing the selector length to 3 bits.

When only the PTC of next hops is needed, the situation changes, as different selector values are required at different switches. Without ECMP groups for PTC of offset, we find that 5, 3, and 5 selector values are required at the ToR, leaf, and spine switches, respectively, to specify the exact next hops. Therefore, the selector length is $\lceil \log(5) \rceil + \lceil \log(3) \rceil + \lceil \log(5) \rceil = 8$ bits. Including the transactional update, the selector length increases to 9 bits.

When both PTC types are supported, we again evaluate each switch individually. This results in 8, 4, and 8 selector values needed at the ToR, leaf, and spine switches, respectively. Thus, the selector length is $\lceil \log(8) \rceil + \lceil \log(4) \rceil + \lceil \log(8) \rceil = 8$ bits. Accounting for the transactional update, the selector length remains 9 bits, the same as when only the PTC of next hops is supported.