REPS: Recycled Entropy Packet Spraying for Adaptive Load Balancing and Failure Mitigation

Tommaso Bonato
ETH Zürich
Microsoft

Abdul Kabbani
Microsoft

Ahmad Ghalayini
Microsoft

Michael Papamichael
Microsoft

Mohammad Dohadwala
Microsoft

Lukas Gianinazzi
ETH Zürich

Mikhail Khalilov
ETH Zürich

Elias Achermann
ETH Zürich

Daniele De Sensi
Sapienza University of Rome

Torsten Hoefler
ETH Zürich
Microsoft

ABSTRACT

Next-generation datacenters require highly efficient network load balancing to manage the growing scale of artificial intelligence (AI) training and general datacenter traffic. Existing solutions designed for Ethernet, such as Equal Cost Multi-Path (ECMP) and oblivious packet spraying (OPS), struggle to maintain high network utilizations as datacenter topologies (and network failures as a consequence) continue to grow. To address these limitations, we propose REPS, a lightweight decentralized per-packet adaptive load balancing algorithm designed to optimize network utilization while ensuring rapid recovery from link failures. REPS adapts to network conditions by caching good-performing paths. In case of a network failure, REPS re-routes traffic away from it in less than 100 microseconds. REPS is designed to be deployed with next-generation out-of-order transports, such as Ultra Ethernet, and introduces less than 25 bytes of per-connection state. We extensively evaluate REPS in large-scale simulations and FPGA-based NICs.

1 INTRODUCTION

Network architecture of distributed AI clusters is inherited from cloud workload deployments: WebSearch, Hadoop, Storage [25, 50]. It typically relies on Infiniband [1] and, more recently, commodity *in-order* Ethernet (e.g., RoCEv2 [10]) for cost efficiency and ease of deployment. These solutions will struggle to deliver peak network bandwidth when scaled from ≈ 10K training endpoints (e.g., to train models of GPT-4, Llama 3 size [3, 23]) to more than 100K (e.g., next-generation model scales) due to:

(1) increased traffic volume and burstiness in collective communication compared to traditional workloads [39, 66],
(2) management complexity and operational cost of lossless in-order network management at this scale due to link failures and degradation [25].

Thus, the community recognized the need for network stacks tailored to distributed training traffic while remaining compatible with commodity datacenter Ethernet infrastructure [25]. Such proposals include SDR by Amazon [54], Falcon by Google [2], TTPoE by Tesla [42], and the upcoming Ultra Ethernet (UE) [19], developed in collaboration between major tech players. Key open questions in these proposals are how to address ***load balancing*** and ***mitigate link failures***.

Current-generation in-order Ethernet-based training systems (such as RoCEv2-based clusters) typically rely on ECMP [32] or similar mechanisms for decentralized routing and load balancing. ECMP load balancing logic applies a hashing function to the 5-tuple header of each data packet to determine the next hop to take. The benefit of this scheme is that, ignoring link failures, it is unlikely to receive out-of-order packets at the destination NIC as the packets that belong to the same connection will be routed through the same network path.

However, ECMP routing is fragile when different connections get hashed to the same link [5, 6, 25]. In this scenario, flows can get hashed to the same path even when other paths are free, resulting in congestion and queue build up, which, in turn, can result in drops and go-back-N retransmission cycles [31].

Moreover, recent works have shown that link failures drastically impact both training times and economic costs. A single link failure can have ≈ 20× higher cost impact in distributed training workloads than in cloud workloads[25, 50]. This observation, along with the increasing scale these systems are growing at, highlights the need for a transport layer with a load balancing scheme that can adapt near-instantly, e.g., within a few round-trip times (RTTs), to network link failures and, consequently, bandwidth asymmetries across the topology.

Several solutions have been proposed to overcome ECMP's limitations. MPTCP, PLB, FlowBender, Flowlet Switching, and Flowcell divide flows into subflows or flowlets and then route each one individually [29, 35, 36, 53, 60]. However, these solutions are still designed for in-order networks, making them inherently sensitive to collisions, prone to handling failures poorly, and requiring significant memory for each connection [41]. Load balancers, such as Oblivious Packet Spraying (OPS) and Multi-Path RDMA (MPRDMA), which operate at a per-packet granularity, can mitigate ECMP-based collisions [21, 41]. However, both approaches lack effective mechanisms to load balance effectively in the presence of failed network paths. Additionally, MPRDMA is constrained by its limited support for receiving out-of-order (OOO) packets and its requirement for per-packet acknowledgments (ACKs).

Our key insight is that OPS problems can be addressed by *adaptive* packet spraying, paired with a transport layer that natively supports *out-of-order* packet delivery, as seen in SDR, UE, and Falcon [19, 21, 25, 41, 46]. Based on this insight, we design and contribute to the Ultra Ethernet Consortium (UEC) our decentralized load-balancing scheme, **R**ecycled **E**ntropy **P**acket **S**praying aka *REPS*. REPS caches "good" network paths in a circular buffer and quickly recovers (within a few RTTs) from network failures by adaptively discovering or freezing network paths.

REPS does not require any specific hardware support from switches beyond ECMP-like header hashing and ECN, which are standard features in modern switches [31, 45]. REPS requires only ≈ 25 bytes of state per-connection, whereas MPTCP requires 368 extra bytes for 8 sub-flows [41].

We extensively evaluate REPS in simulation and augment the simulation findings with real hardware results in Section 4. We deploy REPS in a cluster with modified FPGA-based RDMA-capable NICs. In large-scale network simulations, REPS consistently outperforms state-of-the-art load balancing algorithms. REPS outperforms ECMP and OPS by up to 6× and 1.25× in symmetric networks, and by up to 4.5× and 1.5× in asymmetric networks, outperforming OPS by as much as 100× during short-term transient link failures.

2 BACKGROUND

In this section we introduce the different building blocks necessary to understand REPS logic. We first describe different congestion signals (ECN, loss). We then introduce key terms and concepts related to load balancing (ECMP, EV).

2.1 Congestion Signals

Explicit Congestion Notification (ECN) marking allows switches to notify congestion by setting a bit in the traffic class field of the IP header. The receiver then sends back this marked ECN bit to the sender in its ACK packet header. The sender can choose to react to this congestion signal by adjusting its sending rate [7, 62, 68]. Switches can employ various strategies for marking packets. For instance, in *Random Early Detection* (RED)[24], switches probabilistically mark packets based on queue size, with the marking probability increasing linearly between two thresholds (K_{min} and K_{max}).
In REPS, we use ECN as the congestion signal to detect path congestion due to its simplicity and widespread adoption [31, 45]. Since ECN is not marked for packets when the queue is smaller than K_{min}, ECN effectively filters out cases of minor queuing due to packet collisions across multiple hops, while identifying true congestion at a single bottleneck. In contrast, delay-based signals struggle to differentiate between these scenarios unless enhanced by advanced switch features, such as in-network telemetry (INT) [61].

Packet loss has long been a key indicator of severe congestion in networks [24, 44]. However, using packet losses as the sole signal for congestion detection can result in delayed responses, as losses typically indicate a point of significant congestion. Packet loss detection, often based on timeouts, can be challenging to calibrate and may lead to unnecessary retransmissions. We categorize packet losses in two categories: losses because of severe congestion and losses due to networking failures. To differentiate the two type of losses and to improve reaction times, packet trimming, which only triggers for congestion drops, can be employed [4, 16, 27, 47]. Trimming can be implemented on many existing switches [4] and is starting to be supported by some switch vendors. It is also being pushed as a UE protocol feature [18]. REPS can optionally use trimming, if supported, to distinguish congestion losses from network failure, which are indicated by timeouts (Appendix A).

Congestion Control (CC) algorithms rely on congestion signals, such as ECN marks or packet loss, to adjust a flow's sending rate or window with the goal of maintaining high network utilization while preventing queue build up. REPS is designed to work well with any CC algorithm as long as they support receiving and acknowledging packets out-of-order for a given message. In particular we show later how REPS works well with EQDS, a variant of DCTCP, and an internal CC algorithm [8, 47].

2.2 Load Balancing

Equal Cost Multi-Path (ECMP) is one of the most commonly used and simple load balancing mechanisms. It works by using a hashing function to randomly choose one of the available paths for a given packet [32]. The hashing functions usually takes as input 5 elements (aka *five tuple*) from the packet header: the protocol number, source address, destination address, source port, and destination port (some variations utilize only the four tuple without the protocol number). Recent approaches have proposed incorporating additional fields, such as the Time-to-Live (TTL) or the Flow Label (in IPv6), to further refine the hash calculation [35, 51]. Under normal circumstances, all packets belonging to the same flow are assigned statically to a given path since the hashing function will use the same values as input. This assignment is done statically and ignores the current network congestion and failure conditions. As a result, two or more flows might be assigned to the same path even if there are many more paths available. This will inevitably result in heavy congestion and possibly packet drops as a consequence. Such ECMP *hash collisions* are a well-documented limitation of standard ECMP [6, 25, 65].

Entropy Value (EV) is a value in the packet header that can be configured to be an input to the hashing function in switches. Such a value, which is set by the sender, allows it to alter a packet's path in the network. Possible header fields to be used as an EV are the Source Port field in the packet header [41] or the Flow Label field in IPv6 [51]. We leverage EVs in REPS to improve load balancing and address the ECMP collision limitation, without needing to know the exact mapping between a packet's EV and its resulting path.

Entropy Values Set (EVS) is a fixed-size set of EVs, since the number of possible values is constrained by the number of bits they can occupy in the packet header. For example, the source port field in a UDP header is assigned 16 bits, giving the EVS a size of 65536 possible values (excluding some reserved values) [41]. While different numbers of bits could be allocated for the EVS, we analyze in Section 4.5.1 and Appendix B how many are required for optimal performance. It is generally advantageous for an algorithm to achieve good load balancing performance with a *small* EVS size since that often reduces the algorithm's memory overhead.

Oblivious Packet Spraying (OPS), also known as Random Packet Spraying (RPS) [21], randomly distributes individual packets across all available paths between a sender and a receiver. This is done by selecting a random EV for every packet at the sending host or by choosing a random output port at the switches. OPS has the advantage of distributing traffic evenly across multiple paths, addressing most ECMP issues. However, it is unaware of asymmetries or failures and can still be sub-optimal even in a perfectly symmetrical network (Section 4.3.1).

3 REPS

Recycled **E**ntropy **P**acket **S**praying aka *REPS* is a load balancing algorithm that relies on simple and memory-efficient endpoint logic. By design, REPS can be implemented in NIC hardware or firmware with minimal memory/area footprint. REPS does not need any switch support besides ECMP hashing and ECN marking. The key idea behind REPS is straightforward: when congestion is detected on a certain path, we explore alternative paths while caching and reusing paths with little to no congestion. Specifically, REPS uses a circular buffer of a fixed size to cache EVs of uncongested paths. Algorithm 1 details the pseudocode for a REPS sender when receiving an ACK and when detecting a failure, and Algorithm 2 describes the pseudocode for a REPS sender when sending out a data packet.

3.1 Core Logic: Path Exploration and Reuse

During the first Bandwidth-Delay Product (BDP) packets of a new flow, a REPS sender *explores* random entropies from the EVS. This exploration is necessary because, initially, there is no knowledge about the network's state. In this warm up phase, REPS operates similarly to OPS.

Upon receiving a data packet, the receiver copies the EV from the received packet into the acknowledgement (ACK) packet, forwarding it back to the sender. More specifically, ACKs can use that same EV for their own header instead of using a new header field, eliminating the need for extra header space and for any changes to the packet wire format.

When an ACK arrives at the sender, if it is not ECN-marked, the EV it carries is cached in the circular buffer, and its validity bit is set to 1. Otherwise, if the ACK is ECN-marked, REPS does not cache the EV and discards it. When set, the validity bit indicates that an entropy has not been used after it has been added to the the circular buffer. When sending a data packet out, REPS first checks if there are any valid EVs in its buffer. In case there is any valid EV, REPS *reuses* the oldest valid EV from the circular buffer and resets its validity bit. Otherwise, REPS *explores* a random EV from the EVS.

The circular buffer in REPS ensures that bursts of back-to-back ACKs with "good" entropies are correctly cached and reused. Moreover, it guarantees stable load balancing in the case of failures as shown in Section 3.2. We use a circular buffer of 8 elements based on the bounds from Theorem 5.1.

3.2 Failure Mitigation: Freezing Mode

Once the network experiences any kind of transient (e.g. link flap) or persistent unrecoverable failure (e.g., a link or switch failure), it will take the system some time to recover from it: ranging from several milliseconds to update the ECMP routing group to several seconds if a reboot is needed, and much more if a swap is needed [9, 34].

If we assume that it takes 10 ms to exclude a failed cable from a routing group, packets will still be routed to this failing group during this transient period, resulting in packet drops. Specifically, with a 4 KiB MTU and a 400 Gbps link, this could potentially result in over 120,000 packets (approximately 0.5 GB) being lost (ignoring congestion control). This becomes even more critical in the case of other failures where it takes longer to update the routing.

REPS detects such failures via indirect feedback from the network and enters *freezing mode*. REPS uses a simple timeout heuristic (Section 2.1) that can be enhanced with packet trimming (Appendix A) as a natural feedback from the network to detect failures along a path.

When in *freezing mode*, REPS:

(1) avoids exploring new EVs at random since this could result in the hashing function picking a failing path,
(2) reuses the elements that are currently in the circular buffer even if they might be invalid.

While this strategy could result in slightly worse load balancing (due to potentially reusing the same EV several times), it comes with the major benefit of guaranteeing that REPS will almost never pick the failing path again since the recent received EVs point to healthy paths. Considering the example above: by enabling freezing mode, the number of packets dropped decreases from over 120K packets to only about 1K.

To decide when to exit freezing mode, we set a timer which can be configured by the operator. For instance, in our internal testing with real hardware, we have found that this can be equal to the maximum observed time that it takes for a failure to recover plus a certain buffering period. Once we exit freezing mode, we use random EVs to allow REPS to explore new paths and assess whether we detect new packets failing or not. This prevents REPS from getting stuck in a suboptimal state if the EVs in the buffer were all pointing to a dead path, a rare scenario that can theoretically happen with properly timed back-to-back network failures. Moreover, if REPS exits freezing mode before the issue is fully resolved, it will simply re-enter the mode shortly afterward with minimal impact on performance (Figure 7 and Figure 8).

The intuition behind freezing mode is that once we suspect there is a failure, we want to start avoiding it as soon as possible. Interestingly, we observe that even if we enter freezing mode unnecessarily (i.e., by mistaking a congestion drop for a network failure), REPS would still load balance well, as discussed in Section 4.5.1 and Appendix A. This observation means that even if there is doubt about whether a real failure occurred, REPS can be conservative and can safely enter freezing mode.

Algorithm 1 REPS logic upon ACK receive and failure detection.

```
repsBuffer = [ ]                                        ▷ State variables.
isFreezingMode = false
head, numberValidEVs, exploreCounter = 0

procedure onAck(ackPacket)
    if ackPacket.ecn is set then
        return
    end if
    if not repsBuffer[head].isValid then
        numberValidEVs++
    end if
    repsBuffer[head].cachedEV = ackPacket.ev
    repsBuffer[head].isValid = true
    head = (head + 1)%REPS_BUFFER_SIZE
    if isFreezingMode and now() > exitFreezingMode then
        isFreezingMode = false
        exploreCounter = NUM_PKTS_BDP
    end if
end procedure

procedure onFailureDetection()
    if not isFreezingMode and exploreCounter == 0 then
        isFreezingMode = true
        exitFreezingMode = now() + FREEZING_TIMEOUT
    end if
end procedure
```

Algorithm 2 REPS logic on send datapath.

```
                                ▷ Variables already listed in Algorithm 1
procedure getNextEntropy()
    if numberValidEVs > 0 then
        offset = (head − numberValidEVs)%REPS_BUFFER_SIZE
        repsBuffer[offset].isValid = false
        numberValidEVs−−
    else                                        ▷ Must be in freezing mode.
        offset = head
        head = (head + 1)%REPS_BUFFER_SIZE
    end if
    return repsBuffer[offset].cachedEV
end procedure

procedure onSend(dataPacket)
    if repsBuffer.isEmpty() or (numberValidEVs == 0 and not isFreezingMode) or exploreCounter then
        dataPacket.ev = rand()% EVS_SIZE
        exploreCounter = max(exploreCounter − 1, 0)
    else
        dataPacket.ev = getNextEntropy()
    end if
end procedure
```

3.3 REPS Design Advantages

Simple and versatile algorithm: REPS is simple for cost-efficient hardware support, as it does not require any change of the packet headers format or existing network components. Moreover, its code is short and simple to implement and understand. REPS works best with per-packet ACKs, but

Component	Footprint (bits)
Circular Buffer Element (× elements in buffer):	
Entropy Value (*cachedEV*)	16
Entropy Validity Bit (*isValid*)	1
Global Variables:	
Head Buffer (*head*)	8
Number Valid Entropies (*numberValidEVs*)	8
Exit Freezing Time (*exitFreezingMode*)	32
Is Freezing Mode (*isFreezingMode*)	1
Explore Counter (*exploreCounter*)	8
Total (1 elements in buffer)	**74 ≈10 bytes**
Total (8 elements in buffer)	**193 ≈25 bytes**

Table 1: Per-connection memory footprint of REPS.

we show in Section 4.5.2 that it still performs well even with ACK coalescing.

Minimal NIC memory footprint: A key advantage of REPS is that it does not need to track per-EV metrics and statistics. As will be discussed in Section 4.5.1, achieving good performance with OPS requires a relatively large EVS. If OPS were to maintain metrics for each EV, the memory overhead would be excessive for a hardware NIC implementation, e.g., 8 KiB to store 1 byte per entropy value for an EVS with 8K EVs. However, REPS only needs a fixed number of bytes in memory regardless of the EVS size. More specifically, as detailed in Table 1, REPS requires only around $\approx$ 25 bytes. Moreover, even when constrained to a small EVS, REPS is still able to perform well (Section 4.5.1), which can further reduce REPS' memory footprint by 1 byte since Table 1 assumed 16 bits per EV.

There is a subtle observation as to why REPS can achieve a great performance without needing a lot of state: while the REPS buffer is used to cache good entropies, it is really only useful in certain scenarios like when receiving a burst of ACKs or during freezing mode. In reality, most of REPS' state is on the wire, stored in the inflight data and ACK packets, which will inform REPS about the good paths in the network.

Quick failure mitigation: The general approach of REPS is that it only keeps track of good paths and avoids keeping statistics on congested or failing paths. This approach enables it to promptly load balance away from a congested link or failing link as, especially for the latter, it is never going to take a random guess once a link is failing. Any alternative method that tries to avoid selecting a failing path by tracking bad EVs would need to keep records of all the EVs that map to that path for a given flow, which would involve tracking not only the failing EVs but also all those still in flight.

4 EVALUATION

Our evaluation consists of simulations that stress test REPS at large scale with a number of workloads. We also evaluate REPS at a meaningful scale on real hardware. Our goal is to answer the following research questions:

- Does REPS offer an advantage over OPS under baseline conditions in a healthy network with symmetrical topology? (Section 4.3.1 and 4.4.1)
- Does REPS perform well under network topology asymmetries? (Section 4.3.2 and 4.4.2)
- Is REPS able to quickly recover from failures? (Section 4.3.3 and 4.4.3)
- Can REPS work well even with different network parameters and settings? (Section 4.5)

4.1 Evaluation Setup

Baseline load balancers: We compare REPS with OPS. In the large-scale simulations, we also compare REPS with ECMP [32], PLB [51], MPRDMA [41], Flowlet Switching [60], MPTCP [53], a bitmap approach where we keep per EV statistics similarly to STrack [37], and adaptive RoCE by NVIDIA [46]. We configure PLB to have more aggressive parameters similarly to FlowBender to improve its performance [35]. For Flowlet Switching we set an aggressive flowlet timeout at half of the RTT. For MPTCP, we divide each message into 8 subflows and route each one with different EVs similarly to what happens with multiple QPs (Queue Pairs) [25].

NIC congestion control: In all simulated baseline runs, we use the same DCTCP [7] variant used in MPRDMA [41]. It applies per-ACK congestion window updates, allows the receiver to accept and acknowledge out-of-order packets, and reduces the congestion window by one MTU in case of packet drops. In the FPGA-based experimentation, we use a similar but proprietary CC algorithm that relies on ECN marking, congestion notification packets, and per-flow congestion window adjustments.

Network setup: Regardless of the workloads that we discuss in Section 4.2, in the evaluation we simulate 3 different scenarios: (1) healthy symmetric topology network conditions, (2) asymmetric network conditions (e.g., due to failures, in-order ECMP-hashed background traffic that increase load on specific paths or incremental deployments), and (3) a network encountering various failures. We focus on the most relevant ones for real-world deployments [25] and we report some of the remaining ones in Appendix D.

Simulation model: We implement REPS by extending the *htsim* packet-level network simulator [27]. Our simulations consider different fat-tree topologies with 1024 nodes and 128 nodes and with different levels of oversubscription ranging from 1:1 (no oversubscription) to 4:1. We test 2- and 3-tier fat trees (TOR or Top-of-rack as *T0*, Aggregate as *T1* and Core *T2*). Such topologies are commonly deployed in production datacenters designed for distributed training [25, 55].

We reflect the specifications of current-generation switches in simulation parameters: a 4 KiB MTU size, a bandwidth of 400 Gbps, and a switch traversal latency of 500 ns [14, 20].

We assume uniform link lengths and latencies, with each link exhibiting a latency of 500 ns. We set the retransmission timeout (RTO) to 70 μs which is the amount of time it takes to traverse every queue in the network if it was full plus the network-wide RTT. For each queue K_{min} is set to 20% of the queue size (one BDP) and K_{max} to 80% of it.

REPS-FPGA: To demonstrate the effectiveness and resilience of entropy recycling in a real network environment, we also evaluate REPS in an end-to-end setting using a modified production-grade FPGA-based RDMA-capable NIC. Our testbed consists of a two-tier fat-tree Ethernet network with 100G NICs and 12.8T switches. The default MTU for our FPGA NICs is 8KB and typical RTT incorporating NIC buffer delay and ACK processing through T0 and T1 are in the order of 10 and 15 us, respectively.

4.2 Workloads

We evaluate REPS on a mix of synthetic benchmarks, real datacenter traces, and distributed training collectives.

Synthetic benchmarks set consists of (1) *incast*, (2) *permutation*, and (3) *tornado* traffic patterns. Incast happens when multiple senders simultaneously send to one receiver. It is very common in storage workloads [15, 64] but also, with a small incast degree, in distributed training [25]. In the permutation pattern, each node sends to a random receiver, and we ensure that each node is sending and receiving to exactly one node [7]. The tornado pattern is a special case of the permutation where each node sends to its "twin" node in the other half of the tree. For example, with 128 nodes, node 0 would send to 64 and vice-versa, node 1 to 64 and so on. Tornado is an important worst case for load balancing, as each packet is required to traverse the full tree [49].

Datacenter traces: We use real datacenter traces from similar previous work [7, 63]. We use a series of traces used for web search in production clusters. In such distribution the majority of flows are quite small (less than 100 KB) while a small number of flows are large. For each node we select randomly the receiver and run the simulation for 5 ms. More details are available in Appendix E.

AI collectives: We show simulated results for two commonly used collectives in AI training: the AllReduce implemented via the ring and butterfly algorithm [39], the AllToAll implemented using an algorithm where we limit the number of parallel connections per node (n connections) [30, 43].

Our baseline traffic for REPS-FPGA consists of 128 endpoints under two T0 switches continuously performing 4 MB ring-based AllReduce collective operations, with the logical ring laid out such that all connections traverse the T1 spine to maximize the pressure on the spine of the topology.

4.3 Simulation Results

We conduct a detailed analysis of REPS behavior for each network condition (see Sec. 4). We first examine a specific case in depth and then summarize key takeaways.

4.3.1 Healthy Symmetric Network Conditions. In this section we evaluate the performance of REPS in a simple setting where there is no oversubscription and there are no failures, meaning the network is perfectly symmetrical. Intuitively, this seems the best situation for oblivious packet spraying since evenly splitting the packets across multiple links should result in the best performance. However, as we will see later in this section and based on our simple theoretical model in Section 5, this is not the case as REPS still offers an up to 25% advantage over OPS. This is because of ECMP collisions that still happen with OPS. While over long period of time, each link will be evenly used, there will still be short-term collisions happening that will increase and decrease the link utilization of certain links.

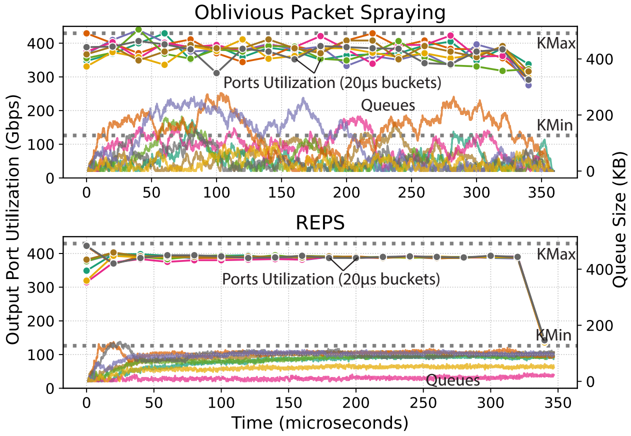


Figure 1: A tornado workload with 16 MiB messages with CC disabled and using OPS and REPS as load balancers.

Microscopic analysis: We study this effect in a tornado pattern, which, theoretically, can be managed entirely by optimal load balancing. We study what happens at a TOR switch and register the link utilization of the uplinks both over the entire simulation and also at smaller time buckets. For visualization purposes we limit these runs to a 2-tier network where each switch has 8 uplinks. However, we note that this problem is present, to an even bigger degree, when using a larger number of uplinks as explained in Section 5. In Figure 1 we visualize statistics for a single T0 switch during the workload run. In particular we show two key metrics over time: 1) on the left Y-axis we show the *output port utilization* at fixed time intervals of 20 us. If it goes above

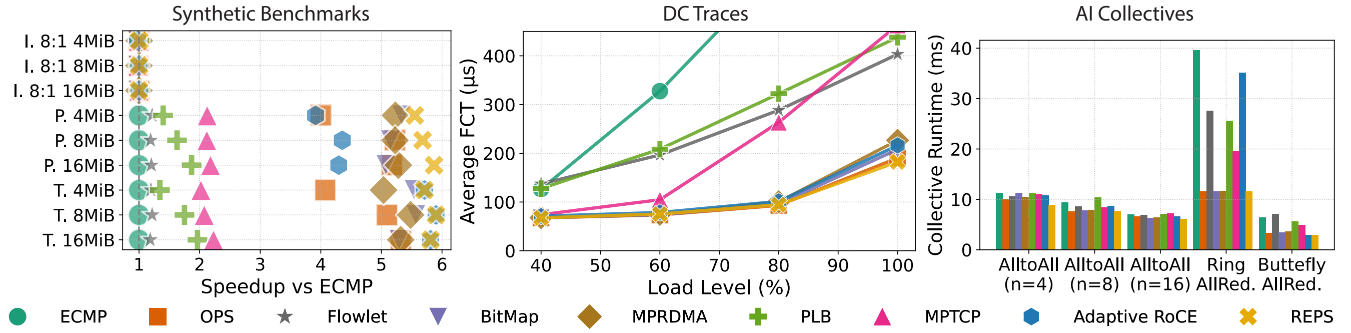


Figure 2: REPS performance in synthetic benchmarks (I.=Incast, P.=Permutation, T.=Tornado), DC traces and AI collectives.

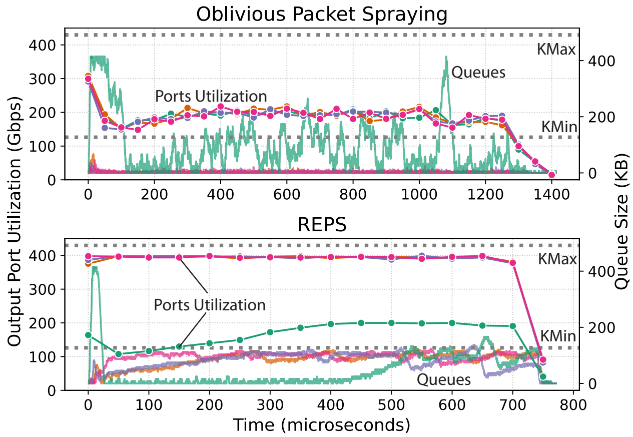


Figure 3: REPS vs. OPS in a 32 MiB message send.

400 Gbps it means that during the studied time bucket some queueing was created. If it goes below it means the output port was slightly under-utilized. 2) On the right Y-axis we show the *queue size* over time of the 8 output ports.

In the case of OPS, we can see that due to the random nature of it, queues are created over time and that the link utilization of each port, at small timeframes, sometimes goes significantly above and below it (15% more or less). This shows that while OPS does still a decent job at completing the workload close to the ideal completion time, it does inevitably create unpredictable queues (potentially even exceeding K_{max} and causing drops) over short time periods. Such queues will cause the CC to kick in and slightly reduce the sending rate and average output port utilization.

In the same configuration with REPS, we notice a major difference in the bottom plot of Figure 1. In particular REPS converge quickly to a configuration where each queue is kept below K_{min} (note that the only guarantee here is that all queues will be below K_{min}, not necessarily all at the same value). At the same time, we can also notice that all the ports converge to the perfect selection rate of 400 Gbps. While the overall completion time is only about 4% better than OPS, the smaller queues provide a better guarantees for system low latency traffic. Moreover, as we can see in Figure 2, this gap expands as we increase the message size.

Looking at the port selection rate over the entire run of the workload for OPS vs. REPS, we observe that they are nearly equivalent. This is again because the problem is with the short-term collisions that OPS can experience at microscopic scale. Finally, we note that, while the main advantages of REPS are not experienced with these perfectly symmetric scenarios, we consistently observe lower max out-of-order distance thanks to its improved stability.

Macroscopic analysis: We now focus more on the overall view comparing REPS with all the other state of the art algorithms in a series of benchmarks.

In Figure 2, we visualize a summary of the performance of the various algorithms by looking at the runtime of the workloads (max FCT). As expected, in the case of incast, the performance is driven almost exclusively by the CC and, hence, we do not see any major difference between all the load balancers and even ECMP performs well. However, once we move to permutation and tornado workloads, ECMP collisions start to drastically reduce the performance of ECMP. In most cases REPS outperforms all the other algorithms.

In the tornado case, Adaptive RoCE is able to match REPS since this is the ideal scenario for it: REPS, unlike Adaptive RoCE, still needs to guess during its initial BDP worth of packets. On the other hand, REPS outperforms Adaptive RoCE in the permutation pattern where taking a local best decision might not always lead to the best global outcome. We also see the difference between algorithms that were designed to reduce the number of out-of-order packets versus algorithms that do not have such hard constraints. Additionally, there is a distinction between algorithms that operate at packet-level granularity, such as REPS, OPS, BitMap, and MPRDMA, and those that operate at a coarser granularity, such as Flowlet and PLB.

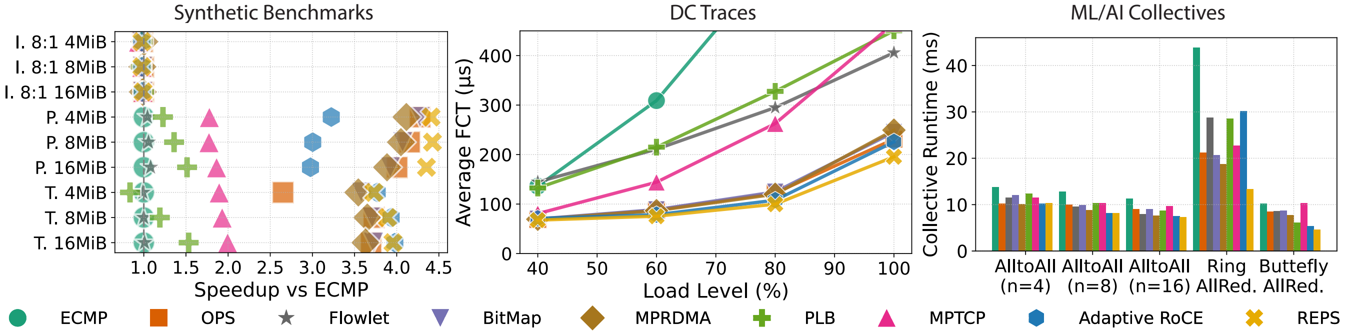


Figure 4: REPS performance in synthetic benchmarks (I.=Incast, P.=Permutation, T.=Tornado), DC traces and AI collectives and an asymmetric network due to 2% of the TOR uplinks being offline.

For datacenter traces, we analyze the results for different load levels, ranging from 40% to 100%. Here we observe again, the clear difference between per-packet algorithms and less granular options. Even at higher load REPS is able to work well with a 5% advantage over OPS.

We also run distributed collectives and report their completion times. We observe how, by design, the ring AllReduce has the same performance regardless of the load balancing algorithm utilized. This is because, due to its ring design, there is no opportunity for congestion to accumulate. In AllToAll, REPS gets an up to 20% advantage over the alternatives.

4.3.2 Asymmetric Network Conditions. We evaluate REPS under different scenarios where some degree of asymmetry is created. We focus on two scenarios: 1) the network is not perfectly symmetrical because of some missing (or degraded) cables, 2) there is some background traffic in the network that is using ECMP routing.

Microscopic analysis: We visualize this problem with a simple scenario where we have a switch with n input and output ports and n flows active, each from a different source sending a 32 MiB message. To create an asymmetry, we reduce one of the uplinks speed to 200 Gbps while all the other links remain at 400 Gbps. In Figure 3 we visualize the output port utilization rate for OPS and REPS.

We observe that while OPS chooses each port equally, irrespective of its actual bandwidth, REPS eventually converges to a stable configuration where the slower uplink is used less frequently. This results in both stabler queues but, more importantly, a much faster completion time (1400 μs for OPS and 756 μs for REPS).

Macroscopic analysis: We now shift our focus to more general results when encountering asymmetries in a network. For space constraints, we focus mostly on the case where some of the links have a lower sending rate. In our first experiment we run synthetic benchmarks where 2% of the TOR uplinks, chosen randomly, have been downgraded to 200 Gbps. In Figure 4, we can see results similar to before where REPS gets an up to 350% advantage over ECMP and 10% advantage over the second best algorithm (usually BitMap). In the DC traces we can see a higher difference due to the asymmetry in the network. At 100% load REPS gets a 25% advantage over the second best algorithm and a 1000% advantage over ECMP. We show the results for several AI collectives. We note how for AllToAll REPS keep a small but significant advantage and in the AllReduce a sizable 50% advantage over the second best performing algorithm.

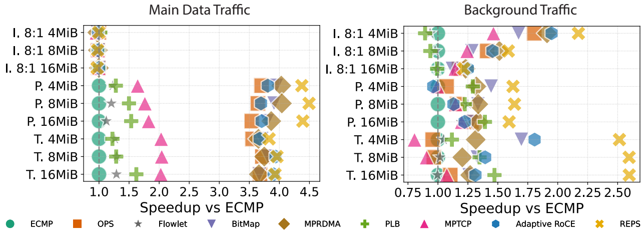


Figure 5: Synthetic traffic with background ECMP traffic.

In Figure 5, we showcase one example of REPS sharing traffic together with background ECMP traffic (we assume 10% of the traffic is ECMP). In this case, REPS: 1) shifts REPS traffic away from ECMP traffic in order to not slow down REPS traffic, 2) helps background traffic by ensuring that it will not be slowed down by REPS traffic. This also highlights the possibility of incrementally deploying REPS on ECMP-base systems.

Finally, we note that WCMP [67] could be used to enhance ECMP performance in the case of a topology with known asymmetries, but would not help as much in the case of unpredictable mixed traffic or sudden temporary asymmetries.

4.3.3 Network Failures. We focus our attention to cases where the network encounters a failure during operation. We collect data from several previous works on networking failures and also study internal logs to simulate the most

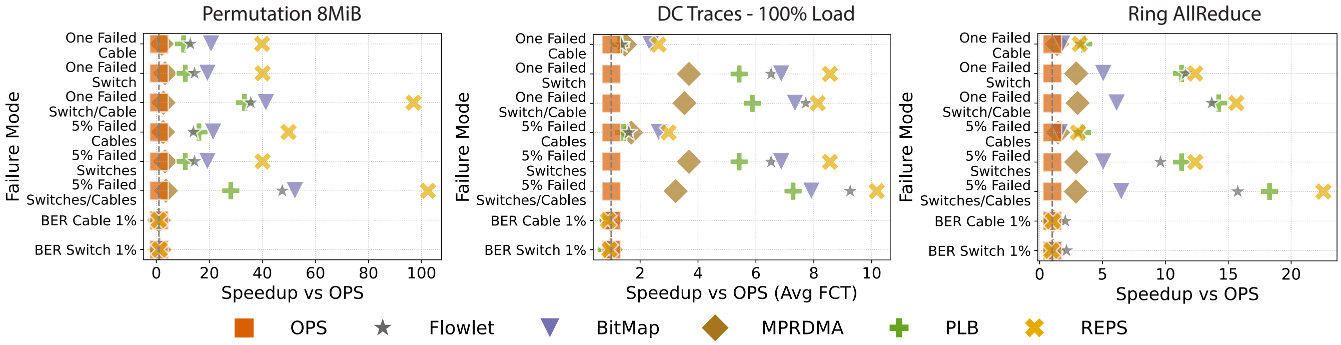


Figure 6: REPS performance under different failure modes in a 8 MiB permutation, DC traces at 100% load and a ring AllReduce.

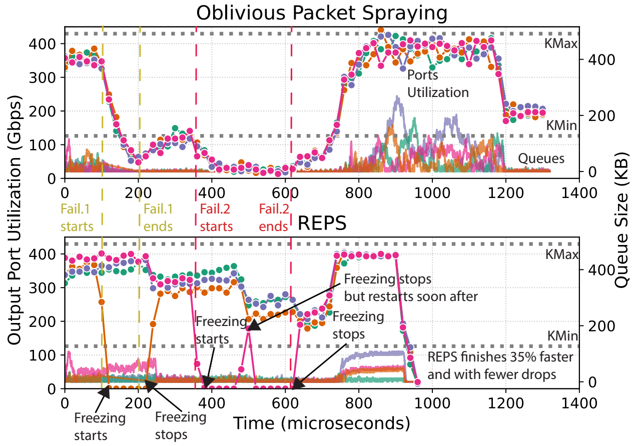


Figure 7: REPS vs. OPS in a 32 MiB permutation with two cables' failure (a shorter one and a longer one).

commonly reported cases [26, 56–58]. Since the probability of a failure happening during a short simulation is low, we simulate worst case scenarios where we force individual failures to happen. Most particularly we focus on total or partial failures of cables and switches. We note that we limit our failures to components that would not prevent the workload from completing (single point of failure).

Microscopic analysis: We study a simple synthetic case where we fail, at different times, two uplinks of a TOR switch: a shorter 100 μs failure and a longer 300 μs failure. As suggested in Section 3.2, we advise to set the timeout for freezing mode of REPS to a number which is consistent with the average failure duration (which we artificially lower only for this experiment).

In Figure 7, OPS keeps choosing all paths equally (although at lower rate due to CC activation), while REPS once it enters freezing mode, stops selecting the failing path all together after only one timeout period (order of tens of microseconds). Afterwards, once the failure stops, REPS also exits freezing mode and converges once again quickly to use all paths. The overall result is that, compared to OPS, REPS completes the workload more than 35% faster even with such a short failure and, more importantly, reduces the number of dropped packets by 2.5×. We showcase a similar analysis but for incremental failures in Appendix D.3.

Macroscopic analysis: We showcase three cases with a series of failure modes in Figure 6. We start each failure mode after a fixed amount of time to ensure initial CC convergence. We can see when dealing with total failures that REPS provides a dramatic speedup over OPS but also other load balancer algorithms. This is because of *freezing mode* that helps REPS to quickly converge to a safe configuration after detecting a failure, considerably faster than the time needed for ECMP routing to update to exclude the failing path. Positively, we note that the gains with REPS inrease with the amount of failures. Furthermore, random drops (e.g., because of BER) does not affect negatively REPS performance.

To further demonstrate the resilience of REPS, we evaluate its performance under an extreme scenario characterized by increasingly large and long-lasting network failures during a permutation. As shown in Figure 8, REPS performs close to an ideal load balancer, even with 50% of network cables failing, while PLB, the second-best alternative, significantly lags behind.

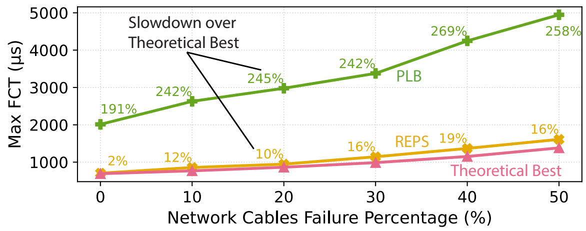


Figure 8: Extreme failures scenario.

4.4 REPS-FPGA Evaluation

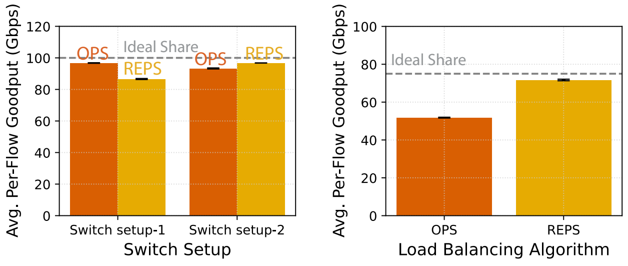
4.4.1 Healthy Symmetric Network Conditions. The first set of results focuses on a baseline healthy symmetric network configuration. Figure 9a shows per-flow goodput defined as end-to-end useful bit rate observed by application, after header, overheads, retransmissions, etc. We use OPS and REPS across two experimental configurations: denoted as *setup-1* and *setup-2*. In *setup-1*, all FPGA endpoints under the two T0s are active while in *setup-2*, 40 out of 64 FPGA endpoints are active.

We present results from both of these configurations as we observe small unexpected performance variations depending on which and how many switch ports are active in experiments near the peak network performance levels using all switch ports (*setup-1*). These variations appear to be related to internal switch microarchitectural details such as port-buffer affinity and vendor-specific scheduling policies. The *setup-2* uses a subset of the switch ports and eliminates most of these vendor-specific and implementation-related behaviors. To get a better understanding of this behavior we performed a sweep where we capped the TX rate of our FPGA NICs and discovered that the slight degradation for *setup-1* when using REPS appears to subside if the TX rate is capped at 95 Gbps.

4.4.2 Asymmetric Network Conditions. We evaluate the performance of REPS under asymmetric network conditions. We connect 16 endpoints through two T0 switches (8 endpoints each) with a total of 4 links to a pair of T1 switches. To demonstrate the adaptive load balancing capabilities of REPS, we change the link speed of one T0-T1 link from 400 Gbps to 200 Gbps, creating asymmetry in the network. Fig. 9b shows the per-flow goodput as observed by the application while Fig. 10a the FCT distribution. OPS sends packets across all paths (including those crossing the 200 Gbps link) with equal probability and is ultimately capped by the slower 200 Gbps path. The ECN marking on the 200 Gbps path causes the CC algorithm to throttle all flows and eventually match the capacity of that single slower link, thus leading to underutilization of the remaining 400 Gbps links (that are running at 50% utilization).

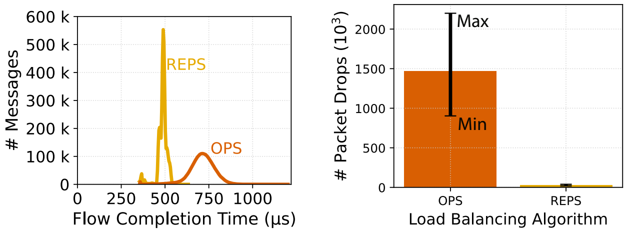
REPS can gracefully adapt in such a scenario as the cached entropies will reflect the network asymmetry and result in a path distribution that is skewed to tailor to the relative capacity of the available paths. In this example, REPS can reach high utilization with average per-flow goodput within 5% of the ideal fair-share target.

4.4.3 Network Failures. We also evaluate the performance of REPS in the presence of network failures. To demonstrate the robustness and resilience of REPS in the context of link failures, Fig. 10b shows total packet drops (average across five runs each) observed in a large-scale 128 endpoint run (endpoints split across 2 T0s connected through 8 T1s) where we abruptly bring down a T0-T1 link during the experiment. While the network is trying to recover from the impact of this event (which in our environment can take in the order of 100s of milliseconds), OPS continues sending packets across all paths (including those affected by the link that went down). The freezing capability of REPS can quickly adapt to such events (within the order of an RTO) and avoid sending packets down the affected paths as the entropy cache is replenished from packets traversing remaining healthy paths.



(a) Symmetric Network **(b) Asymmetric Network**

Figure 9: REPS-FPGA impact on goodput.



(a) Asymmetric Network **(b) Link Failure**

Figure 10: REPS-FPGA impact on FCT and packet drops

4.5 REPS Applicability

In this section, we briefly evaluate REPS, in simulations, under different scenarios by changing the EVS size, the ACK coalescing ratio and the underlying CC algorithm. Generally, we believe that REPS can work well under many different circumstances, topologies and workloads.

4.5.1 EVS Size. In Appendix B, we prove that OPS routing, done through EVs and ECMP hashing, requires a large EVS to work correctly and reduce collisions. Moreover, this requirement also grows with the number of output ports in a switch as demonstrated in Section 5.1. On the other hand, REPS, due to its adaptive nature, can drastically reduce the EVS size and still work well. We show this in the left plot of Figure 12 where we compare, in a real scenario, OPS and REPS when using 32, 256, and 64K EVs. REPS works equally well with 256 and 64K EVs and is only 8% slower with 32 EVs. On the other hand, OPS is 21% and 64% slower with 256 and 32 EVs when compared to 64K. This confirms that REPS could potentially work well even just with 1 byte for the EVS. We note that while OPS could be implemented without using EVS—such as through round-robin selection or by making random choices directly at the switch—these approaches introduce additional challenges (Section 5.2).

4.5.2 ACK Coalescing. We have primarily evaluated REPS without ACK coalescing, as this configuration allows REPS to operate with the most up to date data. However, some transport protocols permit ACK coalescing, where the receiver sends an ACK packet only after receiving *n* data packets from the sender. In theory, the coalesced ACK packet could return all previous non-ECN marked entropies in its header, but we focus on the worst case scenario for REPS where the ACK only carries the entropy of the packet that triggered it.

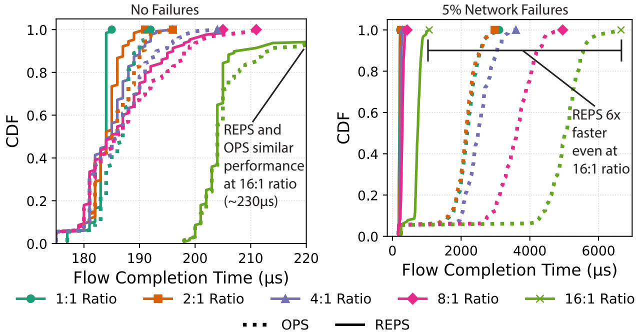


Figure 11: Performance with different ACK coalescing ratios during a 8MiB permutation.

We run a real 8 MiB permutation workload simulating different ACK coalescing ratios. As we can see in the left part of Figure 11, REPS with 2:1, 4:1 and 8:1 coalescing ratios does significantly better than OPS, while with 16:1 it starts losing its advantage. However, it should be noted that in case of asymmetries or failures (right Figure 11), REPS does much better due to its various improvements over OPS even at 16:1 ratio. We confirm these results theoretically in Appendix D.1.

Finally, we note that the trade-off of sending more ACKs is worth the effort if the underlining hardware supports such rate and the impact on the network traffic is minimal (1%) since ACKs are relatively small (64B) compared to the packet size usually used in modern interconnects (4KiB or 8KiB).

4.5.3 Different CC Algorithms. In principle, REPS has been designed to work with any CC algorithm as long as there is no over-reaction to out-of-order packets and ECN support. In this paper, we have, so far, evaluated REPS working alongside a tuned version of DCTCP. However, as we will see, REPS can work well even with other algorithms. Moreover, we envision also a version of REPS that could work just with delay if ECN is not supported but we do not go into details here. For example, in the right plot of Figure 12, we run a simple 8 MiB permutation workload without failure for DCTCP, EQDS and a proprietary CC algorithm. REPS can help all of these CC algorithms when compared to OPS.

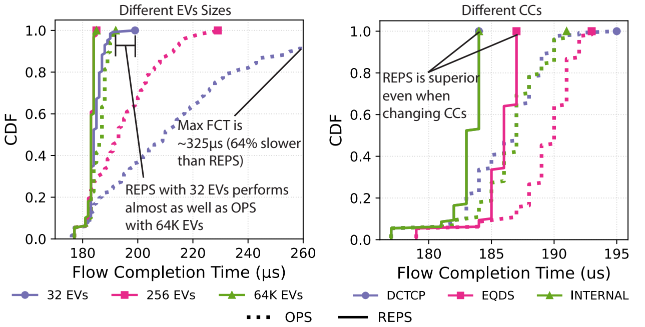


Figure 12: Performance with different EVs sizes and CCs during a 8MiB permutation.

5 THEORETICAL VERIFICATION

To support our experimental findings, we present a theoretical first principles analysis of OPS and demonstrate how it can lead to arbitrarily long queues. To address this issue, we develop a new *recycled balls-into-bins* model and prove its local convergence. This model serves as the theoretical intuition behind the REPS protocol (Section 3).

5.1 Recycled Balls-into-Bins Model

At maximum injection rate, OPS suffers from severe load-imbalance that eventually leads to queues of arbitrary length building up. We explain this behavior with an infinite batched balls-into-bins model [11, 13, 40]. In contrast, we show that recycling good paths such as in REPS leads to a convergent behavior and logarithmically-bounded queues.

In our switch model, each output port corresponds to a bin. At each time step, every non-empty bin removes one element. Afterward, a new set of balls (packets) arrives and is distributed among the bins. In our setting, we focus on the case where n balls arrive in each time step, representing full throughput. The maximum queue length at any time step corresponds to the maximum load of any bin at that time step. Balls are removed in FIFO order from the bins.

In OPS, balls are allocated to bins uniformly at random. In what follows, we assume the EVS is sufficiently large (i.e., 16 bits), allowing us to model the assignments as uniformly random. If balls arrive at a rate of λn for $\lambda < 1$, the process remains stable. The maximum load at any time step is $O\left(\frac{1}{1-\lambda}\log\frac{n}{1-\lambda}\right)$ with high probability [13] and with probability approaching 1, there is always a bin containing $\Omega\left(\frac{1}{1-\lambda}\log n\right)$ balls [13]. In the limit as $\lambda \to 1$, this implies that some bin will eventually become arbitrarily overloaded. In the context of load-balancing, this means that at the maximum injection rate, *oblivious random spraying leads to unbounded queue lengths.* Intuitively, this occurs because n balls are introduced at each step, but fewer than n may be removed, as some output ports may remain unselected.

As n increases, the maximum load grows, exacerbating congestion in OPS. Figure 13 illustrates this effect for $\lambda = 0.99$, where larger numbers of bins (output ports) result in faster-growing maximum queues.

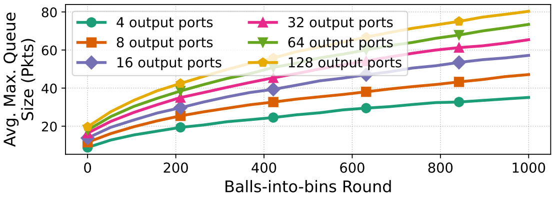


Figure 13: Simulating 10000 rounds of balls-into-bins.

We propose a new model, called *recycled balls-into-bins* and prove it converges locally to a load-balanced state with bounded queue lengths, even at maximum injection rate.

In the recycled balls-into-bins model, we keep a set of $b \cdot n$ colors (for some constant b) and a threshold τ. We cycle through the colors in round-robin fashion in batches of n colors. In each time step, we remove a ball from each nonempty bin. If a bin has at most τ balls, the color of the removed ball remembers the bin, unless it already remembers another bin. If the bin has more than τ balls, the color forgets its bin. Then, for each color in the batch that remembers a bin, throw a ball into the remembered bin. For all colors in the batch without a remembered bin, throw a ball uniformly at random. We show that, for a single switch, *recycled balls-into-bins converges*, meaning all colors remember a bin and keep the same bin remembered.

Theorem 5.1. *For $n \geq 16$, $\tau \geq 4\ln n$, $b \geq 2.4\ln n$, recycled balls-into-bins converges in $O(n\log n)$ expected steps. Every bin has $O(\log n)$ elements throughout, with probability $1 - o(1)$.*

The proof in Appendix C shows that recycled balls-into-bins converges as bins fill and stabilize below the threshold.

To visualize these behaviors, we model the balls-into-bins problem with both OPS and with the recycled balls improvement. In Figure 14, we set $n = 8$ (for visualization purposes, but this holds true for more realistic n values) and showcase the queues' evolution for oblivious packet spraying and the recycled balls model. The results confirm our findings that oblivious packet spraying will see the queues grow unbounded while the recycled balls model will eventually converge and keep all queues below the threshold (τ) value. This is also consistent with our previous simulation results (e.g. Figure 1 and Figure 3).

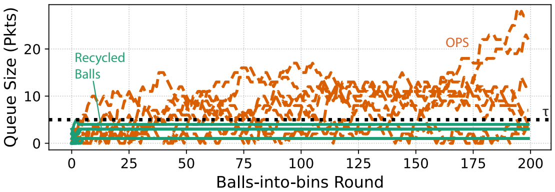


Figure 14: Simulating 200 rounds of balls-into-bins.

5.2 Limitations and Alternatives

While the provided models perform well in an idealized setting with maximal injection rates, in a real network scenario, CC algorithms would reduce the sending rate if a large queue were to build up. However, such activations of the CC would result in a slowdown of the sending rate and, hence, an increase of the completion time of a workload as we can see in Section 4.3.1. Moreover, we note that while real scenarios would have more subtle interactions not modelled here (like using distinct *KMin* and *KMax*), the key results hold and are consistent with our evaluations in Section 4. Finally, we note that here we provide a proof only about the local convergence of REPS and that other models such as static path assignments and round-robin across the ports would also work well and potentially provide no queueing. However, such approaches are not viable in reality for a series of reasons: they cannot react well to partial or total failures, they would struggle with multiple tiers topologies and they require knowing the workload in advance to properly assign paths [25].

6 RELATED WORK

The literature for load balancing designs is vast, however most of it focuses on TCP-based deployments where out-of-order packets are not desirable [5, 12, 21, 60, 63]. In contrast, REPS aligns with the new transports (UE [19], Adaptive RoCE by NVIDIA [46], Falcon [2]) where out-of-order packets help to leverage the capacity of multi-path networks.

Generally, most load balancers work at different granularity: per-flow, sub-flow or per-packet. ECMP [32] is a per-flow solution and susceptible to flow collisions and are oblivious to congested paths. Solutions like Hedera [5] and MicroTE [12] require a global controller which is not desirable in production datacenters [25]. For these reasons, many sub-flow solutions like Flowlet Switching [60], Presto [29], CONGA [6], PLB [51], FlowBender [35] have been proposed. However, some of these solutions are still congestion oblivious

(Flowlet, Presto), react too slowly for AI/ML bursty and intense traffic (PLB, FlowBender), require specialized switches (CONGA). Moreover, most of these solutions are unable to quickly deal with blackholes and failures.

Load balancers with per-packet granularity, such as OPS [21], help drastically reduce ECMP collisions, but still are oblivious to asymmetries. As demonstrated in Section 5 OPS can suffer even in symmetric cases. MPRDMA [41] also uses ECN for load balancing like REPS, however it requires probing and ACK clocking and does not offer caching of entropies. Hermes [63] combines both ECN and delay but it works best with TCP-like protocols and has many parameters to tune.

ConWeave [59] is designed specifically for RDMA networks and offers a solution by masking out-of-order packets in commodity RNICs but it requires changes to TOR switches and has limited scalability.

Proteus [33] focuses on optimizing the load balancing for lossless PFC networks while REPS focuses on lossy networks.

7 CONCLUSION

We presented REPS, a simple, lightweight, yet highly effective load-balancing mechanism designed to meet the constraints of next-generation datacenter networks tailored for AI workloads. As demonstrated in our extensive evaluations, conducted through both simulations and FPGA hardware, REPS' adaptive entropy caching enhances end-to-end performance across multiple critical metrics, including average flow completion time, runtime, and packet loss. REPS outperforms ECMP and OPS by up to 6x and 1.25x in symmetric networks, and by up to 4.5x and 1.5x in asymmetric networks, outperforming OPS by as much as 100x during short-term transient link failures while reducing packet drops by over 70x. We also showed how REPS can work well under various network configurations demonstrating its flexibility to adapt to different scenarios while remaining light-weight on memory requiring only 25 bytes of state per-connection.

REFERENCES

[1] 2004. Infiniband Performance Review. In *2004 USENIX Annual Technical Conference (USENIX ATC 04)*. USENIX Association, Boston, MA. https://www.usenix.org/conference/2004-usenix-annual-technical-conference/infiniband-performance-review

[2] 2023. Introducing Falcon: a reliable low-latency hardware transport | Google Cloud Blog. https://cloud.google.com/blog/. (2023). [Accessed 16-09-2024].

[3] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[4] Popa Adrian, Dumitrescu Dragos, Handley Mark, Nikolaidis Georgios, Lee Jeongkeun, and Raiciu Costin. 2022. Implementing packet trimming support in hardware. (2022). arXiv:cs.NI/2207.04967

[5] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. 2010. Hedera: dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI'10)*. USENIX Association, USA, 19.

[6] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. 2014. CONGA: distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM '14)*. Association for Computing Machinery, New York, NY, USA, 503–514. https://doi.org/10.1145/2619239.2626316

[7] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). *SIGCOMM Comput. Commun. Rev.* 40, 4 (aug 2010), 63–74. https://doi.org/10.1145/1851275.1851192

[8] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference (SIGCOMM '10)*. Association for Computing Machinery, New York, NY, USA, 63–74. https://doi.org/10.1145/1851182.1851192

[9] Anix Anbiah and Krishna M. Sivalingam. 2022. Efficient failure recovery techniques for segment-routed networks. *Computer Communications* 182 (2022), 1–12. https://doi.org/10.1016/j.comcom.2021.10.033

[10] Infiniband Trade Association. 2024. Supplement to InfiniBand Architecture Specification Volume 1 Release 1.2.1 Annex A17: RoCEv2. (2024).

[11] Luca Becchetti, Andrea Clementi, Emanuele Natale, Francesco Pasquale, and Gustavo Posta. 2019. Self-stabilizing repeated balls-into-bins. *Distributed Comput.* 32, 1 (2019), 59–68. https://doi.org/10.1007/S00446-017-0320-4

[12] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. 2011. MicroTE: fine grained traffic engineering for data centers. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies (CoNEXT '11)*. Association for Computing Machinery, New York, NY, USA, Article 8, 12 pages. https://doi.org/10.1145/2079296.2079304

[13] Petra Berenbrink, Tom Friedetzky, Peter Kling, Frederik Mallmann-Trenn, Lars Nagel, and Chris Wastell. 2018. Self-Stabilizing Balls and Bins in Batches - The Power of Leaky Bins. *Algorithmica* 80, 12 (2018), 3673–3703. https://doi.org/10.1007/S00453-018-0411-Z

[14] Broadcom. 2024. Tomahawk 5 Switch. (2024). https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm78900-series (accessed 01/24).

[15] Yanpei Chen, Rean Griffith, Junda Liu, Randy H. Katz, and Anthony D. Joseph. 2009. Understanding TCP incast throughput collapse in datacenter networks. In *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking (WREN '09)*. Association for Computing Machinery, New York, NY, USA, 73–82. https://doi.org/10.1145/1592681.1592693

[16] Peng Cheng, Fengyuan Ren, Ran Shu, and Chuang Lin. 2014. Catch the Whole Lot in an Action: Rapid Precise Packet Loss Notification in Data Center. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. USENIX Association, Seattle, WA, 17–28. https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/cheng

[17] Fan Chung and Linyuan Lu. 2006. Concentration inequalities and martingale inequalities: a survey. *Internet Mathematics* 3, 1 (2006), 79 – 127.

[18] Ultra Ethernet Consortium. [n. d.]. Ultra Ethernet Specification Update - Ultra Ethernet Consortium — ultraethernet.org. https://ultraethernet.org/ultra-ethernet-specification-update/. ([n. d.]). [Accessed 16-09-2024].

[19] Ultra Ethernet Consortium. 2024. Ultra Ethernet. (2024). https://ultraethernet.org/.

[20] Daniele De Sensi, Salvatore Di Girolamo, Kim H. McMahon, Duncan Roweth, and Torsten Hoefler. 2020. An In-Depth Analysis of the Slingshot Interconnect. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14. https://doi.org/10.1109/SC41405.2020.00039

[21] Advait Dixit, Pawan Prakash, Y. Charlie Hu, and Ramana Rao Kompella. 2013. On the impact of packet spraying in data center networks. In *2013 Proceedings IEEE INFOCOM*. 2130–2138. https://doi.org/10.1109/INFCOM.2013.6567015

[22] Benjamin Doerr. 2020. Probabilistic Tools for the Analysis of Randomized Optimization Heuristics. In *Theory of Evolutionary Computation - Recent Developments in Discrete Optimization*, Benjamin Doerr and Frank Neumann (Eds.). Springer, 1–87. https://doi.org/10.1007/978-3-030-29414-4_1

[23] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[24] S. Floyd and V. Jacobson. 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking* 1, 4 (1993), 397–413. https://doi.org/10.1109/90.251892

[25] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, Shuqiang Zhang, Mikel Jimenez Fernandez, Shashidhar Gandham, and Hongyi Zeng. 2024. RDMA over Ethernet for Distributed Training at Meta Scale. In *Proceedings of the ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 57–70. https://doi.org/10.1145/3651890.3672233

[26] Haryadi S. Gunawi, Riza O. Suminto, Russell Sears, Casey Golliher, Swaminathan Sundararaman, Xing Lin, Tim Emami, Weiguang Sheng, Nematollah Bidokhti, Caitie McCaffrey, Gary Grider, Parks M. Fields, Kevin Harms, Robert B. Ross, Andree Jacobson, Robert Ricci, Kirk Webb, Peter Alvaro, H. Birali Runesha, Mingzhe Hao, and Huaicheng Li. 2018. Fail-Slow at Scale: Evidence of Hardware Performance Faults in Large Production Systems. In *16th USENIX Conference on File and Storage Technologies (FAST 18)*. USENIX Association, Oakland, CA, 1–14. https://www.usenix.org/conference/fast18/presentation/gunawi

[27] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. 2017. Re-Architecting Datacenter Networks and Stacks for Low Latency and High Performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 29–42. https://doi.org/10.1145/3098822.3098825

[28] Jun He and Xin Yao. 2004. A study of drift analysis for estimating computation time of evolutionary algorithms. *Nat. Comput.* 3, 1 (2004), 21–35. https://doi.org/10.1023/B:NACO.0000023417.31393.C7

[29] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. 2015. Presto: Edge-Based Load Balancing for Fast Datacenter Networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 465–478. https://doi.org/10.1145/2785956.2787507

[30] Torsten Hoefler, Tommaso Bonato, Daniele De Sensi, Salvatore Di Girolamo, Shigang Li, Marco Heddes, Jon Belk, Deepak Goel, Miguel Castro, and Steve Scott. 2022. HammingMesh: a network topology for large-scale deep learning. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '22)*. IEEE Press, Article 11, 18 pages.

[31] Torsten Hoefler, Duncan Roweth, Keith Underwood, Robert Alverson, Mark Griswold, Vahid Tabatabaee, Mohan Kalkunte, Surendra Anubolu, Siyuan Shen, Moray McLaren, Abdul Kabbani, and Steve Scott. 2023. Data Center Ethernet and Remote Direct Memory Access: Issues at Hyperscale. *Computer* 56, 7 (2023), 67–77. https://doi.org/10.1109/MC.2023.3261184

[32] C. Hopps. 2009. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992. (Nov. 2009). https://www.ietf.org/rfc/rfc2992.txt

[33] Jinbin Hu, Chaoliang Zeng, Zilong Wang, Junxue Zhang, Kun Guo, Hong Xu, Jiawei Huang, and Kai Chen. 2023. Enabling Load Balancing for Lossless Datacenters . In *2023 IEEE 31st International Conference on Network Protocols (ICNP)*. IEEE Computer Society, Los Alamitos, CA, USA, 1–11. https://doi.org/10.1109/ICNP59255.2023.10355615

[34] A. Iselt, A. Kirstadter, A. Pardigon, and T. Schwabe. 2004. Resilient routing using MPLS and ECMP. In *2004 Workshop on High Performance Switching and Routing, 2004. HPSR.* 345–349. https://doi.org/10.1109/HPSR.2004.1303507

[35] Abdul Kabbani, Balajee Vamanan, Jahangir Hasan, and Fabien Duchene. 2014. FlowBender: Flow-level Adaptive Routing for Improved Latency and Throughput in Datacenter Networks. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT '14)*. Association for Computing Machinery, New York, NY, USA, 149–160. https://doi.org/10.1145/2674005.2674985

[36] Abdul Kabbani, David J. Wetherall, Gautam Kumar, Junhua Yan, Kira Yin, Masoud Moshref, Mubashir Adnan Qureshi, Qiaobin Fu, Van Jacobson, and Yuchung Cheng. 2022. PLB: Congestion Signals are Simple and Effective for Network Load Balancing.

[37] Yanfang Le, Rong Pan, Peter Newman, Jeremias Blendin, Abdul Kabbani, Vipin Jain, Raghava Sivaramu, and Francis Matus. 2024. STrack: A Reliable Multipath Transport for AI/ML Clusters. (2024). arXiv:cs.NI/2407.15266 https://arxiv.org/abs/2407.15266

[38] Johannes Lengler. 2020. Drift Analysis. In *Theory of Evolutionary Computation - Recent Developments in Discrete Optimization*, Benjamin Doerr and Frank Neumann (Eds.). Springer, 89–131. https://doi.org/10.1007/978-3-030-29414-4_2

[39] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. 2020. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704* (2020).

[40] Dimitrios Los and Thomas Sauerwald. 2023. Tight Bounds for Repeated Balls-Into-Bins. In *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany (LIPIcs)*, Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté (Eds.), Vol. 254. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 45:1–45:22. https://doi.org/10.4230/LIPICS.STACS.2023.45

[41] Yuanwei Lu, Guo Chen, Bojie Li, Kun Tan, Yongqiang Xiong, Peng Cheng, Jiansong Zhang, Enhong Chen, and Thomas Moscibroda. 2018. Multi-path transport for RDMA in datacenters. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation (NSDI'18)*. USENIX Association, USA, 357–371.

[42] Tesla Motors. 2024. Tesla Transport Protocol (TTPoE). (2024). https://github.com/teslamotors/ttpoe (accessed 09/24).

[43] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).

[44] Kathleen Nichols and Van Jacobson. 2012. Controlling Queue Delay: A modern AQM is just one piece of the solution to bufferbloat. *Queue* 10, 5 (may 2012), 20–34. https://doi.org/10.1145/2208917.2209336

[45] Nvidia. 2024. Networking for the Era of AI: The Network Defines the Data Center. (2024). https://nvdam.widen.net/s/bvpmlkbgzt/networking-overall-whitepaper-networking-for-ai-2911204 (accessed 01/24).

[46] NVIDIA. 2024. NVIDIA Spectrum-X Network Platform Architecture. (2024). https://resources.nvidia.com/en-us-accelerated-networking-resource-library/nvidia-spectrum-x.

[47] Vladimir Olteanu, Haggai Eran, Dragos Dumitrescu, Adrian Popa, Cristi Baciu, Mark Silberstein, Georgios Nikolaidis, Mark Handley, and Costin Raiciu. 2022. An edge-queued datagram service for all datacenter traffic. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 761–777. https://www.usenix.org/conference/nsdi22/presentation/olteanu

[48] Christos Pelekis. 2017. Lower bounds on binomial and Poisson tails: an approach via tail conditional expectations. (2017). arXiv:math.PR/1609.06651 https://arxiv.org/abs/1609.06651

[49] Bogdan Prisacari, German Rodriguez, Cyriel Minkenberg, and Torsten Hoefler. 2013. Bandwidth-optimal all-to-all exchanges in fat tree networks. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing (ICS '13)*. Association for Computing Machinery, New York, NY, USA, 139–148. https://doi.org/10.1145/2464996.2465434

[50] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi Fangbo Zhu, Rui Miao, Chao Wang, Peng Wang, Pengcheng Zhang, Xianlong Zeng Zhiping Yao, Ennan Zhai, and Dennis Cai. 2024. Alibaba HPN: A Data Center Network for Large Language Model Training. (2024).

[51] Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, Qiaobin Fu, Gautam Kumar, Masoud Moshref, Junhua Yan, Van Jacobson, David Wetherall, and Abdul Kabbani. 2022. PLB: congestion signals are simple and effective for network load balancing. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 207–218. https://doi.org/10.1145/3544216.3544226

[52] Martin Raab and Angelika Steger. 1998. "Balls into Bins" - A Simple and Tight Analysis. In *Randomization and Approximation Techniques in Computer Science, Second International Workshop, RANDOM'98, Barcelona, Spain, October 8-10, 1998, Proceedings (Lecture Notes in Computer Science)*, Michael Luby, José D. P. Rolim, and Maria J. Serna (Eds.), Vol. 1518. Springer, 159–170. https://doi.org/10.1007/3-540-49543-6_13

[53] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. 2011. Improving datacenter performance and robustness with multipath TCP. *SIGCOMM Comput. Commun. Rev.* 41, 4 (aug 2011), 266–277. https://doi.org/10.1145/2043164.2018467

[54] Leah Shalev, Hani Ayoub, Nafea Bshara, and Erez Sabbag. 2020. A Cloud-Optimized Transport Protocol for Elastic and Scalable HPC. *IEEE Micro* 40, 6 (2020), 67–73. https://doi.org/10.1109/MM.2020.3016891

[55] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2015. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. In *Sigcomm '15*.

[56] Preeti Singh, J. K. Rai, and Ajay K. Sharma. 2020. Bit Error Rate Analysis of AWG Based Add-Drop Hybrid Buffer Optical Packet Switch. In *2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*. 454–458. https://doi.org/10.1109/ICACCCN51052.2020.9362921

[57] Rachee Singh, Muqeet Mukhtar, Ashay Krishna, Aniruddha Parkhi, Jitendra Padhye, and David Maltz. 2021. Surviving switch failures in cloud datacenters. *SIGCOMM Comput. Commun. Rev.* 51, 2 (may 2021), 2–9. https://doi.org/10.1145/3464994.3464996

[58] Rachee Singh, Muqeet Mukhtar, Ashay Krishna, Aniruddha Parkhi, Jitendra Padhye, and David Maltz. 2021. Surviving switch failures in cloud datacenters. *SIGCOMM Comput. Commun. Rev.* 51, 2 (may 2021), 2–9. https://doi.org/10.1145/3464994.3464996

[59] Cha Hwan Song, Xin Zhe Khooi, Raj Joshi, Inho Choi, Jialin Li, and Mun Choon Chan. 2023. Network Load Balancing with In-network Reordering Support for RDMA. In *Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 816–831. https://doi.org/10.1145/3603269.3604849

[60] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. 2017. Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 407–420. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/vanini

[61] Weitao Wang, Masoud Moshref, Yuliang Li, Gautam Kumar, T. S. Eugene Ng, Neal Cardwell, and Nandita Dukkipati. 2023. Poseidon: An Efficient Congestion Control using Deployable INT for Data Center Networks. https://www.usenix.org/system/files/nsdi23-wang-weitao.pdf

[62] Jin Ye, Renzhang Liu, Ziqi Xie, Luting Feng, and Sen Liu. 2019. EMPTCP: An ECN Based Approach to Detect Shared Bottleneck in MPTCP. In *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. 1–10. https://doi.org/10.1109/ICCCN.2019.8847013

[63] Hong Zhang, Junxue Zhang, Wei Bai, Kai Chen, and Mosharaf Chowdhury. 2017. Resilient Datacenter Load Balancing in the Wild. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 253–266. https://doi.org/10.1145/3098822.3098841

[64] Jiao Zhang, Fengyuan Ren, and Chuang Lin. 2011. Modeling and understanding TCP incast in data center networks. *2011 Proceedings IEEE INFOCOM* (2011), 1377–1385. https://api.semanticscholar.org/CorpusID:16461175

[65] Zhehui Zhang, Haiyang Zheng, Jiayao Hu, Xiangning Yu, Chenchen Qi, Xuemei Shi, and Guohui Wang. 2021. Hashing Linearity Enables Relative Path Control in Data Centers. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 855–862. https://www.usenix.org/conference/atc21/presentation/zhang-zhehui

[66] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. 2023. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277* (2023).

[67] Junlan Zhou, Malveeka Tewari, Min Zhu, Abdul Kabbani, Leon Poutievski, Arjun Singh, and Amin Vahdat. 2014. WCMP: weighted cost multipathing for improved fairness in data centers. In *Proceedings of the Ninth European Conference on Computer Systems (EuroSys '14)*. Association for Computing Machinery, New York, NY, USA, Article 5, 14 pages. https://doi.org/10.1145/2592798.2592803

[68] Yibo Zhu, Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, Ming Zhang, and Jitu Padhye. 2015. Congestion Control for Large-Scale RDMA Deployments. In *SIGCOMM* (sigcomm ed.). ACM - Association for Computing Machinery. https://www.microsoft.com/en-us/research/publication/congestion-control-for-large-scale-rdma-deployments/

A FREEZING MODE IN REPS

Ideally, REPS should enter freezing mode only upon detecting a network failure. To achieve this, we employ two strategies:

(1) **Packet Trimming Support**: When packet trimming is available, distinguishing between packets lost due to congestion and those lost because of network failures becomes more straightforward. We use trimming to identify and separate lost packets with greater accuracy.

(2) **Absence of Packet Trimming**: In the absence of packet trimming, we analyze the maximum round-trip time (RTT) observed during a period preceding the timeout event. If the maximum RTT immediately before the timeout is high, it indicates that the packet was likely lost due to congestion. Conversely, if the maximum RTT was low, the packet was more likely lost due to a network failure.

In our paper, we focused on scenarios where packet trimming was not supported. However, REPS performs optimally when packet trimming is available, benefiting from both an enhanced loss detection algorithm and a more responsive CC loop.

Regardless of the employed strategy, REPS maintains high performance even if it inadvertently enters freezing mode without an actual network failure. This is because entering freezing mode effectively reduces the EVS size of REPS. As demonstrated in Section 4.5.1, REPS remains effective with as few as 32 EVs. For instance, we tested REPS with the 16 MiB tornado workload, running it first under normal conditions and then with forced freezing mode activated after 150 μs. The results showed only a 1% increase in completion time for REPS with forced freezing mode, with the 99th percentile RTT remaining very close, as illustrated in Figure 15.

As an extension for REPS, probing can be incorporated to make the failure detection more precise, but we decided to not add this as part of REPS for now in order to keep things simple. Moreover, in this paper we do not discuss fast loss recovery mechanisms as they are anyway orthogonal to REPS behaviour and could be used to further improve its performance.

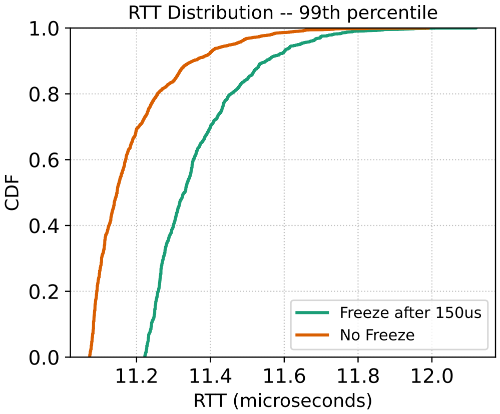


Figure 15: Studying REPS while forcing freezing mode without failures.

B ENTROPY VALUES SET ANALYSIS

Practical implementations make use of hashing functions to map EVs onto output ports. We show that while using a small EVS can lead to high inherent load balancing issues, using 2^{16} EVs is very close to uniformly random.

We first show theoretically that the EVS size is very important for OPS and later that REPS is much more flexible and can work well even with small EVS.

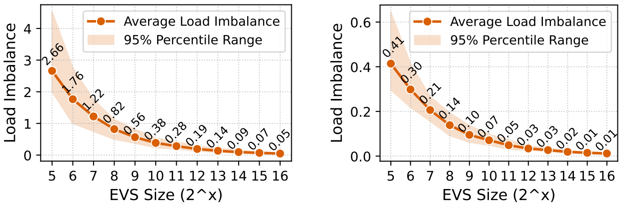
We investigate the phenomenon that a small EVS leads to poor load balancing of the output ports of a switch in the case of a fat tree topology.

We use a balls-into-bins model [52], where m balls are thrown uniformly and idependently at random into n *bins*. The goal is to determine the largest number of balls in any bin, referred to as the *maximum load* $l(m,n)$. We define the *load imbalance* $\lambda_{m,n}$ as $\frac{l(m,n)}{m/n} - 1$, representing the extent to which the most heavily loaded bin exceeds the average.

In our setting, the output ports correspond to bins and the EVs correspond to balls. In our model, $\lambda_{m,n}$ represents the load imbalance of the EVs onto uplinks. Since EVs are chosen for packets uniformly at random, the load balancing of EVs directly affects the load balancing of packets onto output ports.

The load imbalance depends on the average number of balls per bin (i.e., $\frac{m}{n}$): if $m = n$, then the load imbalance is $\Theta(\frac{\log n}{\log \log n})$ with high probability. However, when the ratio of balls to bins $\frac{m}{n} \gg \log n$, the load imbalance tends to zero with high probability [52]. In conclusion, for OPS and a fixed number of flows, we expect high load imbalance with a small EVS and near-zero load imbalance as the EVS increases.

We confirm this theoretical analysis with simulations of the load imbalance. Figures 16a and 16b show the distribution of the load imbalance for 1 and 32 unique flows, respectively. We note that each flow is from a different sender and will hence have different header fields that will be used in the hashing function regardless of the EV value. We note that for each case, we throw for each active flow a number of balls equal to the EVS size, with each ball being a unique EV. We can see that for 32 flows, choosing less than 2^8 EVs can lead to more than 10% load-imbalance, whereas 2^{16} EVs guarantee less than a 1% load imbalance.



(a) 1 flow active

(b) 32 flows active

Figure 16: The expected load imbalance at a switch with 32 uplinks.

C REPS SINGLE SWITCH CONVERGENCE PROOF

We prove the main theorem that demonstrates the convergence of the recycled balls-into-bins process for a single switch.

The process has two distinct phases. In the first phase, there are still empty bins. The number of balls can still grow in this 'warmup' phase. We will bound the steps it takes to undergo the first phase and bound the maximum load. In the second phase, all bins are non-empty. Once a bin is non-empty, it remains non-empty. Moreover, once all bins are non-empty, the number of balls in the system remains constant, as for every ball removed a ball is thrown. In this second phase, we will show that the system converges to a state where all bins have at most τ balls by means of a drift theorem [38]. Note that once all bins have at most τ balls and every ball has a distinct color, they will keep having at most τ balls. Moreover, we will argue about the maximum load and we will show that throughout, as long as the number of colors is at least $b \cdot n \geq (2.4 \ln n)n$, there is at most one ball of any given color with high probability.

Note that much of the proof is devoted to showing convergence of the second phase. If we set the threshold large enough, no bin ever overflows and essentially we only have the first phase, which converges in expected $n \ln n + n$ steps. Since queues are an expensive hardware resource, we are interested in showing good constant factor bounds for the queue lengths rather than giving a weaker asymptotic bound for τ.

Proof of Theorem 5.1. **First Phase.** We say a ball is fresh if it was thrown randomly. After $m = 2n \ln n$ fresh balls are thrown, every bin received at least one fresh ball with probability $1 - \frac{1}{n}$, by the coupon collector problem [22]. Either all bins are non-empty before m steps and we are done, or we have thrown at least m fresh balls and all bins are non-empty. Either way, a bin receives at most $O(\log n)$ fresh balls, by balls into bins with $m+n$ balls [52]. Observe that the number of balls in a bin only increases if it receives a fresh ball. Hence, the number of received fresh balls bounds the number of balls in a bin.

Since $b > 2.4 \ln n$, we have at most one ball per color when the first phase ends.

Second Phase At the beginning of the second phase, we have at most $2n \ln n$+n balls in total and all bins are nonempty. We define a potential function to measure the drift towards a completely load-balanced state. Let X_t^i be the number of balls in bin i at time step t and let $Y_t^i = \max(0, X_t^i - \tau)$. Our potential is $Y_t = \sum_i Y_t^i$. Clearly, when the potential is 0, all bins have at most τ elements. We need to bound the drift $Y_{t+1} - Y_t$.

We first consider the case where each ball has a unique color. The drift depends on the number of bins with more than τ elements, which we denote by K_t. Let us consider the expected drift for one particular bin i. The easy cases are:

$$E[Y_{t+1}^i - Y_t^i | X_t^i = x, K_t = k] = \begin{cases} \frac{k}{n} - 1 & \text{if } x > \tau, \\ \frac{k}{n} & \text{if } x = \tau \\ 0 & \text{if } x \leq \tau - k, \end{cases}$$

If x is in between $\tau - k$ and τ, it gets interesting. The expected drift is still bounded by $\frac{n}{k}$, but this estimate is not strong enough overall. Instead, we need to show that the expected drift is very small for all such bins. To this end, we express the expected drift using a binomial random variable B with parameters $\frac{1}{n}$ and k as $E[\max(0, B - (\tau - x))]$. We cannot directly use linearity of expectation because of the maximum function. Instead, using total expectation leads to the following expression (for $z \geq 3$):

$$E[\max(0, B - z)] = (E[B \mid B \geq z] - z)\, P[B \geq z]$$
$$\leq \frac{1}{z-1} e^{-(z-1)},$$

where we show the tail bound in Lemma C.2.

We now compute the expected drift overall by considering each type of bin separately. By summing over all bins, the total expected drift contribution of bins with less than $\tau - \ln n$ balls is therefore bounded by $\frac{1}{\ln n}$ (since $z \geq \ln n + 1$). Note that these bins only contribute to the drift if $k > \ln n$. Next, consider the contribution from bins with at least $\tau - \ln n$ but at most τ balls. Since we have at most $2n \log n + n$ balls, of which at least $k(\tau + 1)$ balls are in overfull bins, there can be at most $\frac{2n \log n + n - k(\tau+1)}{\tau - \ln n} \leq \frac{3}{4}n - k$ such bins. They each contribute at most $\frac{k}{n}$, so their total contribution is at most $\frac{3}{4}k - \frac{k^2}{n}$. Finally, each overfull bin contributes $\frac{k}{n} - 1$ to the expectation, for a total contribution of $\frac{k^2}{n} - k$. Hence, the total expected drift is

$$E[Y_{t+1} - Y_t | K_t = k] \leq \begin{cases} \frac{1}{\ln n} - \frac{k}{4} & \text{if } k > \ln n, \\ -\frac{1}{4} & \text{else.} \end{cases}$$

Next, we bound the expected drift when there is more than one ball per color. We have two ball of the same color only if a ball does not get removed before the next ball of the same color is thrown. This happens if more than b balls arrive in the same bin in the same time step as the ball (this uses the FiFO-property of the queues). By a tail bound on the binomial distribution, this happens with probability at most $\frac{e^{5/4}}{n^3}$, by Lemma C.1 using $b \geq \frac{12}{5} \ln n$. By a union bound, the probability that *any* color in a batch has more than one ball is at most $\frac{e^{5/4}}{n^2}$. The drift is bounded by n, so the contribution to the expected drift by this case is at most $\frac{e^{5/4}}{n}$.

Note that for $n \geq 16$, $\frac{e^{5/4}}{n} + \frac{1}{\ln n} - \frac{1 + \ln n}{4} \geq -\frac{1}{8}$ and $\frac{e^{5/4}}{n} - \frac{1}{4} \geq -\frac{1}{32}$. Hence, the expected drift is at most $-\frac{1}{32}$. We conclude by an additive drift theorem [28, 38] that the expected time until all bins are below the threshold is at most $E[32 \cdot Y_0] = O(n \ln n)$. If all balls have unique colors at the end, the process converges then. This happens with probability $1 - O(\frac{\ln n}{n})$. So with high probability, a single such $Y_t = 0$ event suffices and the overall expectation is $O(n \ln n)$.

Observe that the number of balls in the queues is bounded by τ plus the maximum load of a batched balls-into-bins process [13] with expected injection rate $\lambda = \frac{k}{n} \leq \frac{1}{2}$. Hence, the number of balls in a bin is $O(\log n)$ throughout with high probability.

□

We use the following tail bound:

Lemma C.1. *Let B be binomially distributed with parameters $\frac{1}{n}$ and $k \leq n$. Then, for any $x \geq 16$:*

$$P[B \geq x] \leq e^{-\frac{5}{4}(x-1)}$$

Proof. We use an additive Chernoff bound [17]:

$$\begin{aligned} P[B \geq E[B] + \delta] &\leq e^{-\frac{\delta^2}{2(E[B]+\delta/3)}} \\ &\leq e^{-\frac{\delta}{\frac{2}{\delta}+\frac{2}{3}}} \\ &\leq e^{-\frac{5}{4}\delta} && \text{using } \delta \geq 15 \end{aligned}$$

□

We use the following lemma to bound the expected drift of bins that are far below the threshold:

Lemma C.2. *Let B be binomially distributed with parameters $\frac{1}{n}$ and $k \leq n/2$. Then, for any $x \geq 7/2$:*

$$(E[B \mid B \geq x] - x) P[B \geq x] \leq \frac{1}{x-1} e^{-(x-\frac{1}{2})}$$

Proof. We use a bound from Pelekis [48] for $E[B \mid B \geq x]$. By their theorem on this conditional expectation:

$$E[B \mid B \geq x] \leq x + \frac{(n-x)\frac{1}{n}}{x - \frac{k}{n} + \frac{1}{n}} \leq x + \frac{1}{x-1} \quad .$$

We bound $P[B \geq x]$ similarly as in Lemma C.1:

$$\begin{aligned} P[B \geq E[B] + \delta] &\leq e^{-\frac{\delta^2}{2(E[B]+\delta/3)}} \\ &\leq e^{-\frac{\delta}{\frac{1}{\delta}+\frac{2}{3}}} && \text{using } E[B] \leq \frac{1}{2} \\ &\leq e^{-\delta} && \text{using } \delta \geq 3 \end{aligned}$$

Because $E[B] \leq \frac{1}{2}$, $P[B \geq x] \leq e^{-(x-\frac{1}{2})}$. □

D ADDITIONAL RESULTS

This section presents additional results that could not be included in the main body of the paper.

D.1 ACK Coalescing Theoretical Modeling

To assess REPS performance under different ACK coalescing ratios, we validate it using the theoretical model from Section 5.1.

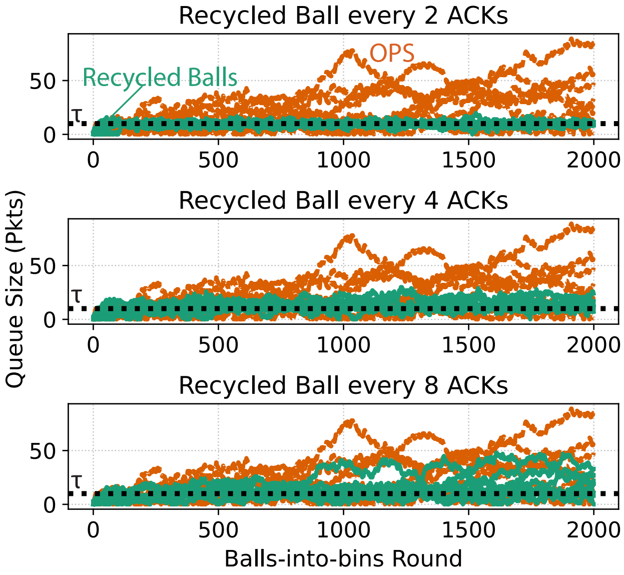


Figure 17: Performance with different ACK coalescing ratios using the balls-into-bins models.

Figure 17 shows how the recycled balls model performs well, even with less frequent recycling. While we lose the guarantee of consistently staying below τ, with 2:1 and 4:1 recycling ratios, the queues barely exceed this threshold. An 8:1 ratio still proves slightly more advantageous than OPS.

D.2 Different Tiers

We aim to verify that REPS performs effectively with fat-tree topologies that have three tiers. This scenario poses a slightly greater challenge for REPS, as a single EV must manage two hops. Nonetheless, there is no intrinsic reason why REPS should not perform well in such a topology.

To validate this, we execute the synthetic benchmark using the symmetric topology described in Section 4.3.1, but with three tiers instead of two. The results, shown in Figure 18, indicate that REPS performs comparably to the two-tier topology.

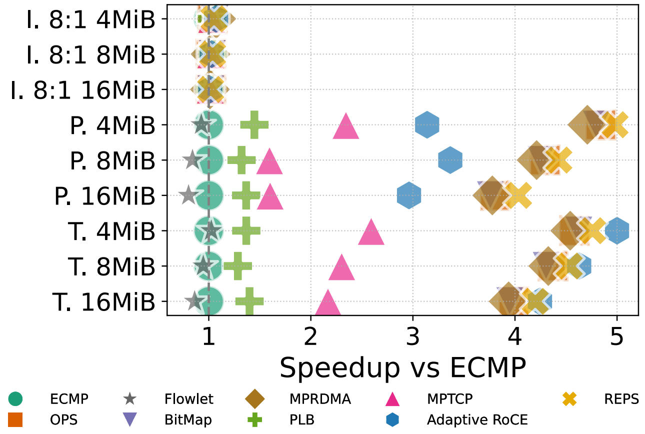


Figure 18: Studying REPS with a 3 tiers fat tree topology.

D.3 Incremental Failures

To further demonstrate the resilience of REPS under failures, we conducted an experiment where we incrementally failed all but one of a switch's uplinks at 200 µs intervals. Figure 19 presents the permutation from the perspective of the failing switch, where three of the four uplinks were permanently disabled in a staggered manner. As expected, REPS enters freezing mode immediately after the first failure, ensuring that failing output ports are avoided. Notably, small utilization spikes are observed on the failing links when REPS exits freezing mode to verify if the failure has been resolved. Since the failures are permanent for the duration of the experiment, REPS promptly re-enters freezing mode after detecting unresolved issues.

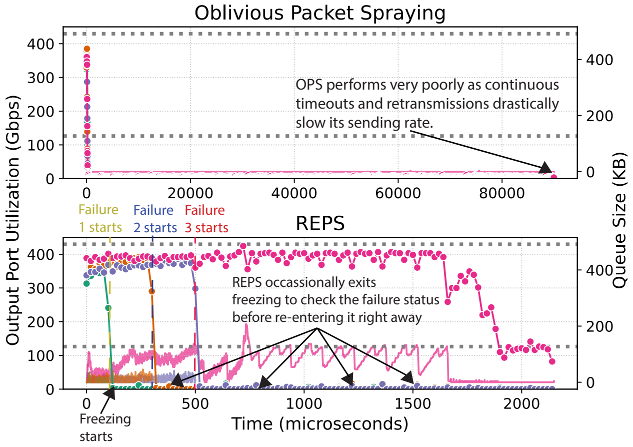


Figure 19: REPS vs. OPS in a 32 MiB permutation with incremental persistent failures.

In this drastic scenario, OPS performs 40× worse than REPS, primarily due to its inability to avoid broken links, resulting in numerous retransmissions and reduced congestion windows.

E ADDITIONAL DATA

The datacenter traces used throughout this paper have been previously used in a number of similar works [7, 60]. In particular we use traces provided by Alibaba and Facebook and a series of traces used for web search in production clusters. The CDF distribution for such traces can be seen in Figure 20. For most of the paper we focus exclusively on the WebSearch traces.

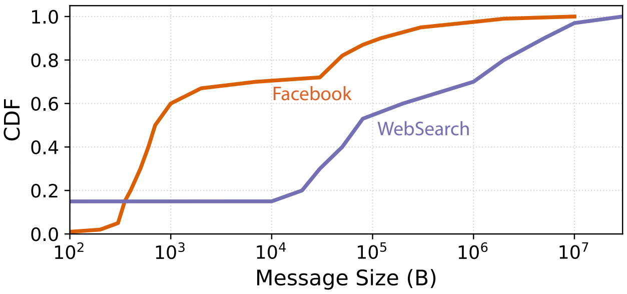


Figure 20: CDF for different data center traces