PLB: Congestion Signals are Simple and Effective for Network Load Balancing

Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, Qiaobin Fu, Gautam Kumar, Masoud Moshref, Junhua Yan, Van Jacobson, David Wetherall, Abdul Kabbani*
Google

ABSTRACT

We present a new, host-based design for link load balancing and report the first experiences of link imbalance in datacenters. Our design, PLB (Protective Load Balancing), builds on transport protocols and ECMP/WCMP to reduce network hotspots. PLB randomly changes the paths of connections that experience congestion, preferring to repath after idle periods to minimize packet reordering. It repaths a connection by changing the IPv6 Flow Label on its packets, which switches include as part of ECMP/WCMP. Across hosts, this action drives down hotspots in the network, and lowers the latency of RPCs.

PLB is used fleetwide at Google for TCP and Pony Express traffic. We could deploy it when other designs were infeasible because PLB requires only small transport modifications and switch configuration changes, and is backwards-compatible. It has produced excellent gains: the median utilization imbalance of highly-loaded ToR uplinks in Google datacenters fell by 60%, packet drops correspondingly fell by 33%, and the tail latency (99p) of small RPCs fell by 20%. PLB is also a general solution that works for settings from datacenters to backbone networks, as well as different transports.

CCS CONCEPTS

• **Networks** → **End nodes**; **Data path algorithms**;

KEYWORDS

Congestion control, Datacenter fabric, Load balancing, Distributed

ACM Reference Format:
Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, Qiaobin Fu, Gautam Kumar, Masoud Moshref, Junhua Yan, Van Jacobson, David Wetherall, Abdul Kabbani. 2022. PLB: Congestion Signals are Simple and Effective for Network Load Balancing. In *ACM SIGCOMM 2022 Conference (SIGCOMM '22), August 22–26, 2022, Amsterdam, Netherlands.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3544216.3544226

1 INTRODUCTION

Modern networks scale capacity by using many links in parallel, often in Clos topologies. This design leads to routes with many paths from a source to each destination. In this setting, effective mechanisms for spreading load across the available network paths are vital for application performance and network efficiency.

The most widely-used load balancing mechanism is ECMP [40] (Equal Cost Multi-Path), in which each flow is randomly hashed to one of the set of switch outputs, or more generally WCMP (Weighted Cost Multi-Path) [45], which is a weighted version for a non-uniform allocation. However, it is well-known that ECMP does not produce a balanced load and can exacerbate congestion hotspots, which has led to many alternative designs [3, 4, 9, 15, 19, 20, 24, 26, 27, 31, 37, 39, 41, 43, 44].

We observed hotspots degrading performance when there was unused capacity in Google datacenters. When we sought to mitigate them, we were largely unable to apply the designs in the literature. They would have added significant complexity to the network, resulting in increased costs, reliability risks, and slow evolution. For example, regardless of their other merits, it is expensive and time-consuming to adopt schemes that require new switch processing (e.g., CONGA [4]), difficult to deploy solutions that make wholesale transport changes (e.g., MPTCP [9]), and risky to rely on designs with global controllers (e.g., Hedera [3]). Even simple and attractive ideas like switch flowlets [39] pose per flow state scaling challenges.

Given this difficulty, we developed our own load balancing design, called PLB (Protective Load Balancing). PLB leverages existing transports to identify flows that are experiencing congestion and need to be repathed. It builds on earlier work on FlowBender [24] with two ideas. Where FlowBender lacked an architecturally clean way to repath connections, PLB uses the IPv6 Flow Label for this role. The switch simply forwards IPv6 packets with ECMP/WCMP flow hashing on the usual four-tuple plus the Flow Label; this configuration is supported by modern switch hardware. It lets hosts randomly change the path of a flow within the set of available paths without application involvement. Where FlowBender simply moved connections when they experienced congestion, PLB prefers to repath after idle periods to minimize transport interactions due to packet reordering. This strategy makes small RPCs repath more often than large RPCs at hotspots, resulting in lower tail latencies. PLB still operates at near-RTT timescales to respond to volatile workloads, and quickly removes connections from congested paths. Across hosts, this behavior probabilistically diffuses load away from congestion hotspots towards coldspots to increase the effective carrying capacity of the network.

PLB is used throughout the Google fleet for datacenter and backbone traffic, covering all internal applications and services. We report its impact with A/B production measurements for RPCs using TCP and low-latency messaging using Pony Express [33]. We find large gains across the board for the network infrastructure and applications. After deploying PLB, the median utilization imbalance of highly-loaded ToRs in Google datacenter networks fell by 60%, packet drops correspondingly fell by 33%, and the tail latency (99p) of small RPCs fell by 20%.

We claim both the design of PLB and production experience with its global deployment as contributions. Specifically, the design of PLB shows how the IPv6 Flow Label can cleanly separate the roles of pathing and connection demultiplexing. And the heuristic

*This author contributed to work while at Google

SIGCOMM '22, August 22–26, 2022, Amsterdam, Netherlands

ACM ISBN 978-1-4503-9420-8/22/08.
https://doi.org/10.1145/3544216.3544226

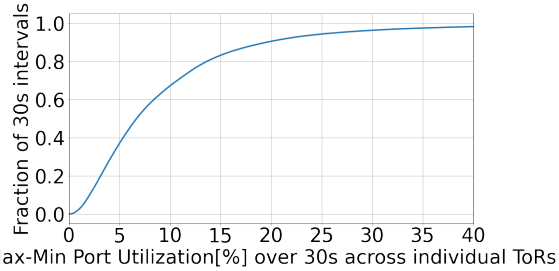


Figure 1: Pre-PLB CDF of *max − min* uplink utilization for busy ToRs

of repathing after idle is highly effective for heavy-tailed traffic at optimizing small RPC latency and large RPC throughput. In terms of production experience (§4), we believe this to be the first paper that reports on link utilization imbalance in datacenters. Among other lessons (§6), we value the robustness of PLB, including how it coexists with traditional ECMP traffic and compensates for sub-optimal WCMP weights. Finally, while our design for PLB was constrained for production deployment, we believe it has an excellent cost/benefit compared to other designs, as we explain in §7. To this end, we have open-sourced PLB code for TCP [21]. This work does not raise any ethical issues.

2 PROBLEM & SOLUTION REQUIREMENTS

2.1 ECMP and Hotspots

The impetus for our work on load balancing is congestion hotspots. We observed hotspots in datacenter fabrics where some Top of Rack (ToR) switch ports were busy while others were not, and some fabric switches were busy while other equivalent ones were not. In a balanced network, in which the workload is well-matched to carrying capacity, this should not happen. Rather, utilization should increase uniformly across groups of ports and switches until the busy part of the network is overloaded as a whole. When there is imbalance then the effective capacity of the network is lowered, sometimes substantially for the affected users.

We traced the imbalance to ECMP/WCMP. ECMP randomly maps each 4-tuple, e.g., TCP connection, to a switch output to spread load across the links to a destination; WCMP is a weighted version of ECMP for a non-uniform allocation. They are a critical part of networks such as datacenters that scale capacity by using many parallel links. Both rely on the assumption that when there are many flows then each link will have roughly the expected number of flows, and hence have a balanced load. But this does not happen in practice because ECMP/WCMP ignore flow sizes.

Datacenter workloads are known to be heavy-tailed: most bytes traversing the network are carried in a small fraction of the RPCs [7, 36]. For instance, in Google datacenters, only a single-digit percentage of RPCs are longer than 64KB but they make up over three-quarters of the overall traffic. In effect, only a proportionally small number of flows matter in terms of load. When ECMP randomly assigns those few flows to links then per the Binomial distribution uneven assignments are likely.

We see this effect throughout Google datacenters. To show it, we focus on busy ToRs because imbalance only degrades performance at high utilization. We define a busy ToR using an arbitrary threshold of >70% utilization (bytes over a 30s interval) on any of its uplink ports for over 4 hours in a day. To measure imbalance, we define the Load Imbalance (LI) to be *max − min* utilization across the ToR uplinks (toward the datacenter core switches) for each ToR, again over 30s.

Figure 1 shows the CDF of LI for the uplinks of 1000+ busy ToRs for a whole day. This data comes from before PLB was deployed. Around a third of the measurements have >10% utilization spread, and many have a much larger spread. This imbalance may seem minor, but it is detrimental beyond causing performance variation. There are always locations of overload in datacenters, despite the best provisioning and traffic engineering efforts, due to heterogeneity and workload variability. Imbalance raises the maximum port utilization at these locations, which super-linearly increases queuing and packet drops [36]. We will see later that PLB greatly reduces the overall network loss by reducing imbalance.

We see the same kind of imbalance at datacenter fabric switches, and even backbone switches. It is reduced with greater flow aggregation but does not resolve due to the heavy-tailed workload – imbalance is a network-wide issue.

2.2 Solution Requirements

The problem we seek to solve is how to assign flows to the available paths to minimize the network traffic at hotspots. A hotspot is a congested switch output port that has significant queuing or loss. Routing and traffic engineering compute the available paths; they are unchanged by our solution. Note that this formulation does not require link utilization to be balanced at all locations, only at points of congestion. This is because our end goal is to improve application performance.

Any solution used in production must accommodate the heterogeneity of the Google fleet. Network fabrics are composed of multiple generations of switches from multiple vendors. Network designs are similar at a high level but differ in many respects. Hosts use multiple transports with different forms of congestion control. And applications generate a diverse set of workloads which can change quickly.

This heterogeneity leads us to focus on incrementally-deployable solutions that build on the installed base of switches and hosts. In particular, we limit our consideration of switches to configuration changes only. It takes several years to replace switches in the fleet. If we need new hardware along network paths then the benefits of a solution will not be realized for a long time. Hosts' software can be far more easily upgraded. However, we limit our consideration to solutions that work with the existing transports, since it would take years to migrate to new protocols. Even with these restrictions, a solution must be incrementally deployable for incremental benefit. It is not possible to upgrade even an individual datacenter at once. The network also supports a mix of local and remote traffic, including Internet traffic originating outside Google that is beyond our control.

We do not believe these solution requirements are unique to Google. All large providers likely face the pressures we have described. Solutions that are simple and general will always have a strong value proposition because of the combination of reduced costs with broad applicability. We also do not believe that our pragmatic choices lead to a solution that is inferior to state-of-the-art designs. We find PLB to be highly effective, and contrast it to related work (§7) to explain the differences.

2.3 Congestion-Aware Approach

Our approach for PLB is to use congestion as a signal to drive rebalancing. This approach arose from our unsuccessful explorations with obliviously spreading traffic. We first broke TCP connections

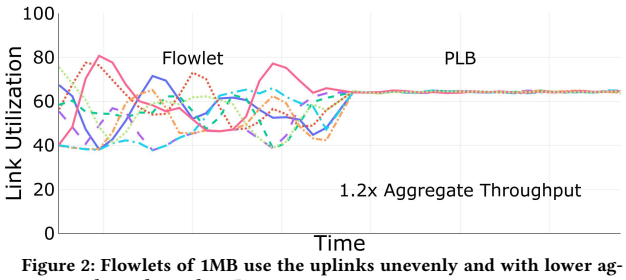


Figure 2: Flowlets of 1MB use the uplinks unevenly and with lower aggregate throughput than PLB.

into flowlets, each subject to ECMP, to address the flow size problem. The hope was that over many flowlets, the link utilization would even out. *But at each moment the number of flows is not increased.* Consider a ToR with 8 uplinks and 32 flows, for an expected 4 flows per uplink. From the Binomial distribution, we have a 20% chance of underloading links with ≤ 2 flows, and 20% chance of overloading links with ≥ 6 flows. If we repath all flows every 1s or 100ms or 10ms then we simply change which links are underloaded and overloaded.

In fact, we repathed connections every 1MB and the result was worse. This is because connections on lightly-loaded links send faster, so they are repathed more often, which clumps connections on heavily-loaded links, where they run more slowly. Eventually the clumps break up and reform elsewhere so the utilization of each link fluctuates. Figure 2 shows this flowlet experiment on the left side; the link utilizations vary over time and from each other, by up to 2X. PLB is shown on the right side for comparison. It has balanced utilization that is higher in aggregate as links are not under-used.

Congestion-aware designs also lend themselves to incremental deployment as they work when only a portion of the traffic is load-balanced. The load-balanced traffic will respond to congestion even if it is caused by legacy traffic. Conversely, designs that obliviously spray traffic across the datacenter amplify the impact of other points of congestion.

We were inspired by the FlowBender [24] congestion-aware design. However, it requires overloading the VLAN tag, which means that VLANs could no longer be used freely, and limited the length of network paths that could be influenced. After our datacenters transitioned to IPv6, the IPv6 Flow Label [6] provides exactly the architectural support we need. It identifies packets from a host that require the same network handling; hosts should label connections for this purpose. Its value has no other semantics that need to be maintained. And switches are encouraged to use it for load balancing [12].

3 PLB DESIGN

3.1 High-Level Design

PLB is a host design that integrates load-balancing with the transport for IPv6 traffic. It is complementary to ECMP/WCMP routing, except that flow hashing is extended to include the Flow Label in addition to the usual 4-tuple.

PLB design has two parts: congestion detection and re-pathing. A sender host detects a connection is experiencing congestion via the transport. It then repaths the connection by assigning it a new, randomly-generated Flow Label for subsequent outgoing packets. This action has the effect of selecting the flows that collide to form hotspots and probabilistically spreading them across the available paths, repeating as necessary to minimize hashing collisions. PLB uses the existing congestion signals in the corresponding transport's congestion control and doesn't require any additional congestion instrumentation. This design is applicable to many transport protocols and network settings. We present PLB design details for TCP and Pony Express.

3.2 PLB for TCP

We develop PLB-TCP with BBRv2 [10] congestion control, without any specific change to BBRv2. (We discuss other interactions between PLB and congestion control later in this section.) The entire PLB-TCP implementation is about 50 lines of code in the Linux kernel TCP stack. PLB is backwards-compatible, and can be incrementally enabled on senders to repath flows when both sides support ECN, and switches are configured for Flow Label hashing and ECN marking.

Detecting congestion. A PLB-TCP sender uses a simple DCTCP-like heuristic [5] to detect that a connection is congested. The pseudocode is depicted in Algorithm 1. Switches mark CE on packets when the queue exceeds a certain threshold. The receiver echos back CE marks to the sender. This is all existing TCP/IP behavior for ECN. For every ACK received, the sender calls *TCP_CongDetection()*. The sender computes the fraction of packets with CE marks per round-trip. When this fraction is larger than a constant K, we say the round is congested. After experiencing M consecutive congested rounds, we mark the flow as congested. The flow remains congested only while consecutive rounds are congested. We give values for K, M and other parameters later.

Algorithm 1: PLB congestion detection algorithm. Default K=0.5

```
1  function TCP_CongDetection(ACK)
2      is_congested = (ACK has CE echoed)
3      PLBDetection(ACK, is_congested)
4  end
5  function Swift_CongDetection(ACK)
6      is_congested = (ACK RTT > swift_target_delay)
7      PLBDetection(ACK, is_congested)
8  end
9  function PLBDetection(ACK, is_congested)
10     pkts_delivered += pkts_acknowledged
11     if is_congested then
12         pkts_congested += pkts_acknowledged
13     end
14     for every round trip do
15         if pkts_congested ≥ K * pkts_delivered then
16             congested_rounds++
17         else
18             congested_rounds=0
19         end
20         pkts_delivered = 0
21         pkts_congested = 0
22     end
23 end
```

Repathing with minimal reordering. PLB repaths by changing the Flow Label on subsequent packets over the connection. However, this can lead to packet reordering at the receiver when PLB

Algorithm 2: PLB repathing for a TCP or Pony Express sender. A data packet is a TSO jumbo frame in TCP. Default M=3, N=12.

```
SendDataPacket():
if (congested_rounds ≥ M AND pkts_in_flight = 0) OR
   congested_rounds ≥ N OR RTO_retransmit then
    Assign flow to a new random flow label
    congested_rounds=0
Proceed to rest of the sending routine
```

naively moves a flow away from a longer path with high queuing to a shorter one. For modern TCP stacks, RACK [13] minimizes false loss recoveries due to reordering but reordering still poses a challenge for Generic Receive Offload (GRO). GRO aggregates consecutive TCP segments into a jumbo frame of up to 64KB, which is passed to the kernel TCP stack, where it triggers a single stretched ACK. Hence GRO substantially reduces NIC upcalls and ACKs for high-speed transfers, which results in large CPU savings. When packets are received out-of-order, GRO aggregation ends early. This inflicts a high CPU penalty at both the sender and receiver.

To avoid reordering, PLB tries to defer repathing a congested flow until the flow becomes idle. When the flow restarts, no packets are inflight and a path change will not cause reordering. This strategy is effective because most RPCs are small. Distributed applications commonly use sharding and distribute RPCs over many connections, so small RPCs are likely sent on idle connections. To handle the remaining heavy flows that send large RPCs and are rarely idle, we force a repath after N consecutive congested rounds. This logic is shown in Algorithm 2. Here, we restrict Flow Label changes to TCP Segmentation Offload (TSO) jumbo frame boundaries to increase the likelihood of in-order reception of TSO bursts. At worst, packet reordering may occur every N round-trips, which we find infrequent enough to avoid CPU penalties and spurious loss recovery.

Moving small RPCs away from heavy flows. Datacenter traffic is known to be heavy-tailed: Google datacenter applications mostly use persistent connections to send RPCs. The vast majority of RPCs are smaller than 64KB but they make up less than a quarter of overall traffic. In this situation, the heavy flows that make up the bulk of the traffic cause significant queuing [34] that impacts the latency of small RPCs. Triggering repathing quickly upon restart from idle is beneficial for reducing the tail latency of small RPCs. Since the connections that send small RPCs idle frequently, PLB conveniently tends to move them off the paths of the big queues contributed by heavy flows or large RPCs.

Note that PLB does not assume or depend on RPC traffic patterns. It only exploits the idle periods in between RPCs due to distributed nature of datacenter applications. Rehashing due to PLB is a Markov process where the transition to a new random path depends on the congestion along the current path. The likelihood of hitting congestion on the new path depends on the available bandwidth.

Dealing with link failures. PLB also repaths when a Retransmission TimeOut (RTO) occurs. This can happen under exceptional situations when the TCP feedback loop is broken, such as link failures and small transfers under heavy congestion. Small transfers (e.g. single packet RPCs) are vulnerable because they have few ACKs to return CE marks. While ECN is the common case for detecting congestion, we find RTO to be a key part of a complete solution.

3.3 PLB Congestion Detection for Pony Express

Applications within Google use a transport called Pony Express [33] as well as TCP. Pony Express runs in user-space and accesses the NIC directly for low-latency transfers [38]. The PLB-Pony Express implementation is also about 50 lines of code, and is used in conjunction with Swift delay-based congestion control [30].

PLB-Pony Express only differs from PLB-TCP in how it detects congestion, as depicted in Algorithm 1. For every ACK received, the sender calls *Swift_CongDetection()*. Instead of ECN marks, Swift congestion control in Pony Express uses NIC timestamps to measure the network path and queuing delay, and aims to keep the RTT below a target value. For each round-trip, PLB computes the fraction of RTT measurements exceeding $swift\_target\_delay$. The target delay is a Swift configuration parameter based on expected path propagation delays and a queuing target. As with TCP, we detect congestion when there are M consecutive round-trips with at least a K fraction of high RTTs.

Pony Express does not use TSO and GRO as it bypasses the kernel. However, it is still important to minimize reordering to avoid spurious loss recovery and congestion reactions. To do so, we use the same repathing logic in Algorithm 2.

3.4 Algorithm Dynamics

We describe how PLB interacts with congestion control, and how it converges to a network-wide load balance.

Interaction with congestion control. We use timescale separation to let PLB and congestion control operate concurrently without adverse interactions. When a flow experiences congestion, PLB waits several round trips (configured by M, N in Algorithm 2) before repathing. This gives congestion control time to react to transient issues. If PLB were to repath immediately, then congestion control would become an open-loop, with every round-trip potentially measuring a different path. Similarly, PLB prefers to repath after idle, at the start of an application duty cycle. Congestion state is likely stale at this time and probing is already required to acquire fresh state. In this way, PLB does not require modifications to the underlying congestion control module.

On the other hand, we do not want to endure prolonged heavy congestion (i.e. large N, K in Algorithm 1). This could lower performance as congestion control may slow the flows down instead of PLB seeking more available bandwidth. We conducted a parameter sweep and empirically found that K = 0.5, M = 3 and N = 12 strikes a good balance across a range of workloads. We found no particular sensitivity, only the need for different M and N to simultaneously repath more quickly to speed small RPCs, and more slowly to avoid disrupting large RPCs. For other workload patterns, different values may work better. For example, a lower K value may be needed if the switch ECN marking threshold is high. A higher M is needed if the corresponding congestion control needs more round trips to react and stabilize congestion.

There is a further interaction for link failures. During a failure, flows on failed links will experience RTOs and PLB will repath them, which will tend to shift load away from the failed links. This is all as expected. However, if the workload is larger than the now reduced capacity, then flows are more likely to experience congestion than normal. In this situation, we do not want to repath again and risk shifting flows back to the failed link, which would trigger yet more repathing. Instead, PLB damps repathing after an RTO by pausing the PLBDetection() module in Algorithm 1. We damp for a short interval based on the expected link recovery time to let the network

recompute routes and congestion control reduce the workload. To avoid synchronization across flows, the pause time is randomly selected between one to two times the expected recovery time.

PLB as Thermal Diffusion. To see how PLB balances load, consider the uplinks of a ToR switch. When the imbalance is high with most flows clumped on few uplinks, host repathing is highly likely to increase the spread of the flows, and quickly reduce the magnitude of the imbalance. When the imbalance is slight, then few flows will be repathed. The system will then explore low imbalance states until a balanced state is found by chance – these states are sticky as heavy flows will remain in them for their duration with no further repathing.

The same argument applies across switches, since host repathing explores all available paths. In fact, we can make a much stronger statement. Much as the analysis of Kelly [28] and Low [32] shows that the independent action of host congestion control achieves global bandwidth fair shares, we can leverage theoretical results to say that the independent action of PLB hosts achieves a global load balance.

The seminal work of Feller [16] relates discrete Markov processes to the Fokker-Planck equation that describes diffusion processes. Under very general conditions, it lets us relate PLB to a thermal diffusion process. For this to hold, we require a measure of temperature, which comes from hotspots as the volume of traffic experiencing congestion, and an action that probabilistically follows a gradient, which is provided by spreading. As with thermal diffusion, PLB will then cool the global network to an equilibrium state.

In practice, datacenter workloads have high connection churn rate and heavy tail distribution, PLB actively tries to remove hotspots created by hashing imbalance or due to large flows clumped together. What we do not know is how quickly PLB will cool the network. This is important in practice because the network workload changes quickly, which means that the system is always moving towards equilibrium. PLB has excellent prospects because it reacts at near RTT timescales by using transport signals. In the next section, we examine how PLB performs in production.

4 PLB IN PRODUCTION

We deployed PLB for TCP and Pony Express on all Google datacenter servers. Our production network consists of many large datacenters spread across the globe. They support a multitude of applications, ranging from storage workloads to latency-sensitive workloads like in-memory key-value stores. IPv6 usage is widespread, and all switches use the Flow Label in addition to the four-tuple for ECMP/WCMP hashing. There are no other changes to network controllers or traffic engineering (e.g. ECMP and WCMP weights, routing).

PLB covers nearly all internal application transfers within the datacenters and across the B4 [23] backbone. TCP uses BBRv2 [10] congestion control, while Pony Express uses Swift [30] congestion control. Both protocols share the same underlying networks hence their PLB implementations interact with each other.

PLB allows rapid deployment because it can be unilaterally enabled at the sender on a per-flow basis without even restarting a flow. Hence PLB was deployed quickly and progressively without disrupting applications in individual datacenters. It can also be selectively disabled for troubleshooting, although we did not need to do so.

4.1 Evaluation Methodology

We compare measurements from before and after the deployment of PLB to assess its benefits. As the global rollout takes many days, we measure each datacenter for one day before and one day after the local PLB deployment. The two days are within a week. We then aggregate results to see fleetwide trends. For confidentiality reasons, we normalize the measurement scale. This method provides better workload stability than using a single period for all datacenters.

The effects of PLB are directly visible at switches, where hotspots are reduced, so we begin here. We focus on changes in utilization imbalance and packet discard metrics. Load imbalance is computed as $max - min$ utilization over the switch ports, which we term as LI, with more details given below. It is an imperfect but useful metric for load balancing behavior. PLB reduces imbalance that causes congestion, but does not otherwise drive it to zero. Nor is balanced utilization the target operating point in the less common case of flows in asymmetric topology. As well as being artificially high, LI can be artificially low during overload, when link utilization becomes saturated. Despite these caveats, reductions in imbalance due to PLB always reflect network improvements. Similarly, while switch discards are caused by both high link load and unbalanced link load, reductions in discards due to PLB always reflect network improvements.

Then we follow up with transport measurements to confirm that network improvements translate into application gains. We focus on RPC network transfer latencies to see changes in latency distribution for small and large RPCs. Finally, we check transport metrics, e.g., spurious retransmissions, for unwanted side-effects.

4.2 Improving Load Balance

We use our global telemetry monitoring to obtain for each switch port 30-second counters of the number of bytes forwarded and packets discarded due to output buffer overruns. From this telemetry, we derive datapoints for: (i) Load Imbalance (LI), the $max - min$ utilization over the switch ports; (ii) Aggregate discards, the sum of discards across the ports; and (iii) Maximum/Minimum utilization over the ports.

Our datacenter SDN controller [17] uses WCMP for traffic engineering. ToR uplinks mostly use ECMP unless the datacenter fabric capacity is partially drained (e.g. for maintenance or upgrade). For ToRs, we compute these metrics across all the uplink ports feeding the fabric core.

Top of Rack (ToR) uplinks. ToRs are an excellent test case for load balancing because they are the first point of over-subscription for application traffic, and have relatively few flows that require careful assignment for balance. We start on busy switches where imbalance matters the most. As mentioned earlier, we say a switch is busy if it has maximum utilization of >70% utilization for 4+ hours a day. Note flows experience congestion below 100% utilization because the datapoints average bursty traffic over 30 seconds.

Figure 3 shows the LI distribution of a particular datacenter that was badly unbalanced. Only 1% of the flows were elephants, but hashing collisions due to the birthday paradox [29] skewed the port utilization. After PLB rollout, the 50^{th} and 99^{th} percentile LI were reduced by 70%, with a clear shift in mass to lower LI. The average LI for busy ToRs fell from 13% to 4.5%, which is a substantial utilization drop. Packet discards at those switches fell by ~50%. This case shows how PLB can help when ECMP falls short.

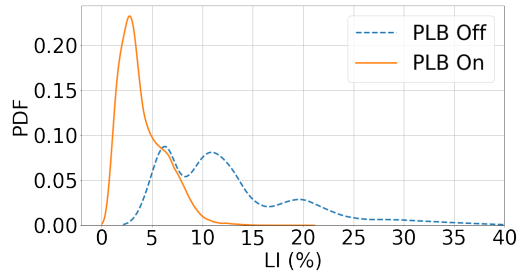


Figure 3: PDF of LI of uplinks of busy ToRs in a badly unbalanced datacenter

Next we study the ToR uplinks across the entire fleet. At such a large operation scale, ToRs across the fleet cover a range of utilization levels with varying load imbalance states. We want to evaluate the effects of PLB for these different combinations of utilization and load imbalance. For every ToR in our fleet, we collect the following two metrics from measurements spanning one day before and after PLB rollout: (1) LI and (2) Minimum port utilization. Minimum port utilization along with LI define the utilization range across ToR uplink ports over 30s measurement interval.

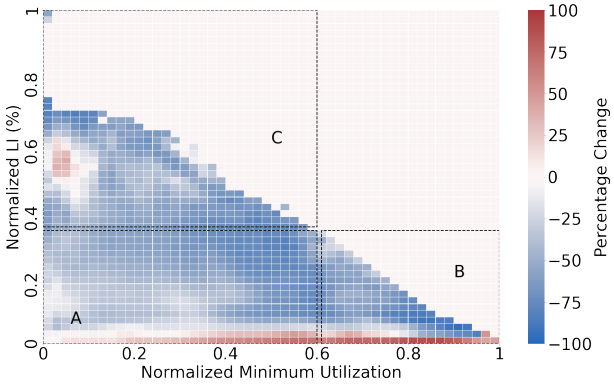


Figure 4: Percentage change in bin count after PLB is enabled where (x,y)=(min_utilization,LI). Red(hot) indicates increase in bin count whereas blue(cool) indicates a decrease.

We compute 2D histogram of data before and after the rollout, H_{before} and H_{after} respectively. H is the normalized number of measurement samples in each bin, where a bin (x,y) corresponds to minimum utilization=x and LI=y. Figure 4 shows $(H_{after} - H_{before})/max(H_{before}, H_{after})$. The color of each bin reflects the percentage change the bin underwent when PLB was enabled. A red bin indicates increase in bin count whereas blue color shows a decrease.

The effect of PLB is evident across the board in that it moves mass away (blue shade) from high LI regions towards 0 (the deep red) across all utilization levels. For high LI and high utilization regions B and C, the percentage decrease is also higher, as shown by a darker shade of blue. Near the tail of utilization in region B, we see the impact of PLB on the hottest ToRs in our fleet: it strongly pushes the load towards balance. At low utilization and low LI region A, the ports are less utilized and congested. They show less mass shift away, as indicated by a white to light shade of blue. The increase in mass around (0.05, 0.6) is because of routing changes due to drains that PLB simply shows more clearly. The sharp and large increase in volume of LI=0 bins show how PLB is effective across all utilization levels, and especially at higher levels. For busy ToRs across all our datacenters, PLB reduced 50^{th} percentile LI by 60% and 99^{th} percentile by 25%.

Datacenter core switches. For switches deeper in the fabric, like switch ports feeding the spine (aggregation layer) and fabric border gateway switch ports feeding the WAN, the load-balancing is more sophisticated. Topology asymmetry causes switches between the two layers, e.g. aggregation and spine, to be connected through different numbers of links. The connected group is referred to as a block. The SDN controller uses WCMP to assign weights per block to optimally distribute the workload across multiple paths inside the datacenter to handle the capacity differences [17].

Despite the routing optimization, highly transient workload can still cause imbalance and congestion to trigger PLB. Since the flow label change affects every hop on the path, PLB may steer a flow not only within the links in a congested block, but also away from a congested block entirely. This reduces the imbalance in each WCMP group (where links have equal weights) and across different WCMP groups.

As an example, we study the imbalance of a datacenter with heavily used spine switches, before and after PLB rollout. Figure 5b shows the maximum utilization across ports of all aggregation blocks in that datacenter. PLB shifts and flattens the peak from 0.6 max utilization towards 0.5 by load balancing traffic across all blocks by moving traffic away from hotspots. Figure 5a shows the significant reduction in LI across busy aggregation blocks, by both PLB moving traffic among the links in each block, or to other blocks. Figure 5c tells a similar story for the border links directed towards the WAN, where overall LI distribution shifts towards left by ~2.5%. These examples demonstrate PLB helps significantly where it is needed the most (i.e. network hotspots).

To confirm that PLB delivers a consistent gain, we computed the same metrics across the fleet. Across spine blocks, we observed a LI reduction in 50^{th} percentile by 46% and 99^{th} percentile by 30%. For fabric border switches to WAN, 50^{th} percentile is reduced by 41% and 99^{th} percentile by 24%.

4.3 Reducing Packet Drops

Lower packet discard rates at switches provide compelling evidence that PLB is alleviating congestion by lowering the maximum link utilization. Since the packet discard rate at individual ports is volatile, we inspect cumulative discards over long time scales to visualize the impact of PLB.

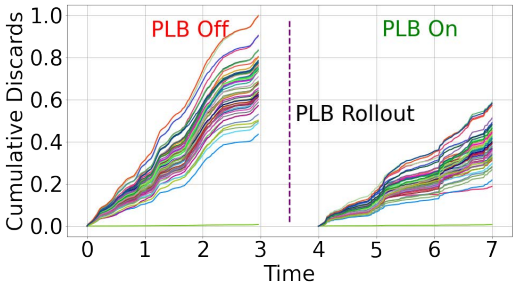
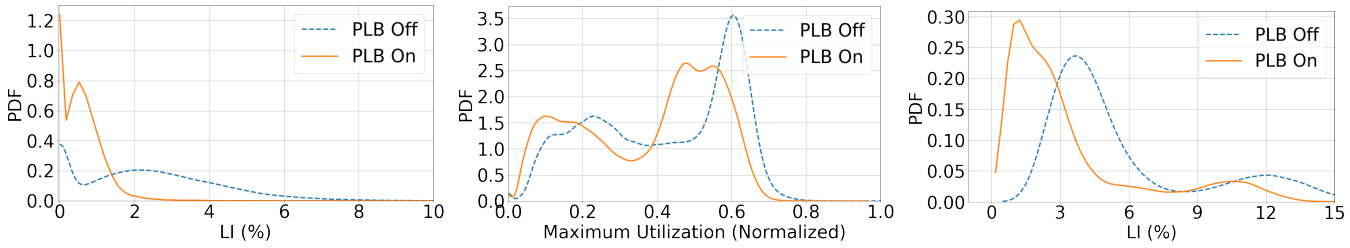


Figure 6: Cumulative Discards across ToR Uplinks



(a) LI (%) across busy aggregation blocks (b) Max util across all aggregation blocks (c) LI (%) across border switches feeding WAN

Figure 5: Impact of PLB on core switches in individual datacenters

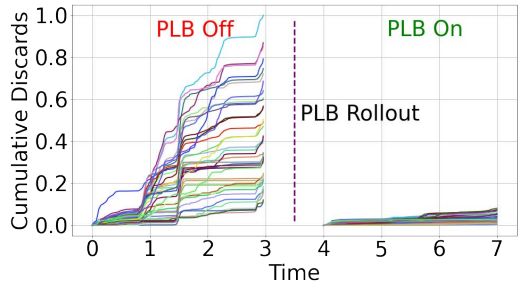


Figure 7: Cumulative Discards across spine switches

Figures 6 and 7 show the cumulative discards for busy ToR switches and spine blocks in one Jupiter fabric before (left) and after (right) the PLB rollout over 7 days. Cumulative discards are the sum of the 30 second packet drop counters for all switch ports versus time; each colored curve represents a single switch/block. Wiggles in the curves correspond to short-term changes in behavior due to events, while the overall slope of the line gives the long-term discard rate.

We see that, for this datacenter with PLB, the long-term discard rate for busy ToR switches and spine blocks fell by 50% and 80% respectively. The absolute number of discards is normalized in the graphs, but the ToR discards are more than an order of magnitude higher than spine discards. This is expected since the ToRs are the first point of over-subscription and have greater imbalance since the datacenter topologies are designed to evenly distribute the ToR traffic across the spine blocks. Correspondingly, we see that the ToR curves uniformly shift to lower long-term discard rates, while the spine blocks are affected more unequally, as there are fewer places at which imbalance need to be corrected.

Fleetwide graphs would have been too cluttered to show at this level, but the long-term discard rate for all busy ToRs and spine blocks went down by 33% and 15% respectively. That is, PLB can avoid a large portion of the packet drops for the entire fleet by improving on ECMP/WCMP.

Note that LI is computed over 30s and may underestimate the instantaneous load as burst arrivals get averaged out over 30s. However, PLB's goal is to reduce the sustained load imbalance instead of transient ones. LI does capture these long term congestion episodes. As shown in this section, improvement in LI directly translates into lowering of maximum utilization across switch ports and aggregate packet discards.

4.4 Reducing Application Latency

We study two representative workloads to show that the network gains we have presented correspond to application benefits: a storage service using TCP, and a distributed file system using Pony Express.

Storage Service (TCP). Storage comprises the majority of datacenter traffic at Google. It is used by many applications for both reads and writes via RPCs over TCP. ToRs hosting these storage servers can easily become hotspots due to rapid workload changes.

We study RPC network transfer latency of small and large RPCs. The data is sampled from our fleet telemetry system that instruments applications which use RPCs with kernel network timestamps. For a TCP write system call, we measure the transfer latency from when the first byte leaves the sender till the ACK is received. This latency includes the network delays and congestion control reactions and loss recovery. It does not include TCP ACK delay at the receiver, which is factored out.

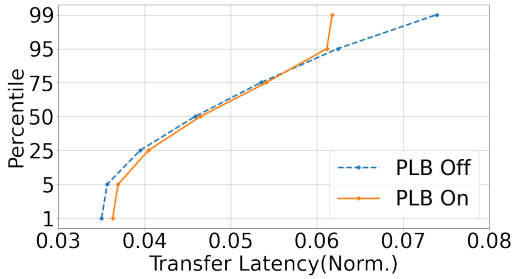


Figure 8: Latency for small(≤1KB) RPCs of storage workload

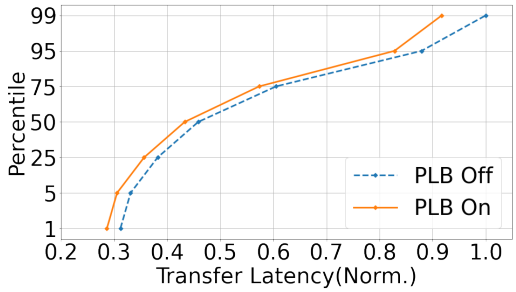


Figure 9: Latency for large (2MB+) RPCs of storage workload

We compare the RPC transfer latency of the storage servers in one representative datacenter for 7 days before and after the PLB deployment, excluding the rollout time window; we did not observe any significant change of the overall workload in this period. Note that this data covers RPCs whether or not they are impacted by switch hotspots and can benefit from PLB, so even small improvements are significant.

Figure 8 shows how the tail latency for small RPCs (≤1KB) decreased after PLB deployment. These small RPCs are often control messages for the storage system that need to be delivered quickly.

Their latency is lower-bounded by the physical propagation delay (which PLB cannot change). We see the 99th percentile has dropped by about 20%, while the first 25th percentile has very slightly increased. This is because before PLB a fraction of these transfers suffer much heavier congestion due to load imbalance. After PLB, all small transfers experience more similar congestion hence the curve tends to level out.

On the other hand, the latency of large RPCs (≥2MB), which carry storage chunks, is reduced across all percentiles, as shown in Figure 9. The reduction starts at ~5% and increases to 10% at 99p. This is because as PLB resolves collisions by spreading heavy flows it is finding more bandwidth and increasing the effective capacity for all flows.

Distributed File System (Pony Express). Next we look at a service that is sensitive to tail latency: transactions that fail to complete by the deadline are cancelled and prompt a more expensive recovery. The deadline should be high enough that it is missed only in exceptional situations, and low enough that recovery commences quickly. This service runs in a separate datacenter from the previous storage service.

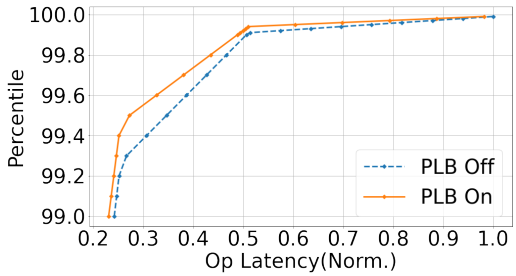


Figure 10: Op Latency over one day before/after PLB rollout. Note that the y-axis starts from the 99th percentile

We see a reduction in the op latency when we compare the 10 days before and after PLB rollout. Op latency, which includes both the network and host delays, sees an obvious improvement between 99 and 99.9 percentiles, as shown in Figure 10, with up to ~20% improvement at 99.5 percentile. The lower tail has reduced application errors as the file operation has tight deadlines. Figure 11 shows that PLB shifts the curve of the hourly deadline-exceeded rate to the left, reducing the median by 66%. That is, better handling of network congestion can reduce application errors.

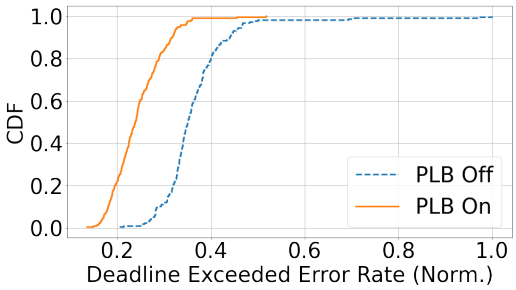


Figure 11: Hourly Deadline Exceeded Error Rate

4.5 Transport Layer Health

The PLB algorithm (§3.2 and §3.3) prefers to defer repathing until restart from idle to minimize reordering, and changes the Flow Label at TSO jumbo frame boundaries to minimize CPU costs. However, reordering may still occur if repathing happens during active sending, so we check that the transport remains healthy when running PLB.

First, we observed no noticeable CPU penalty with PLB deployment for both TCP and Pony Express. Second, we checked for spurious retransmissions that signal false loss recovery. We measured the spurious retransmission ratio for each flow as the number of packets reported in duplicate SACK options [18] for TCP (and the equivalent for Pony Express) divided by the total retransmitted packets.

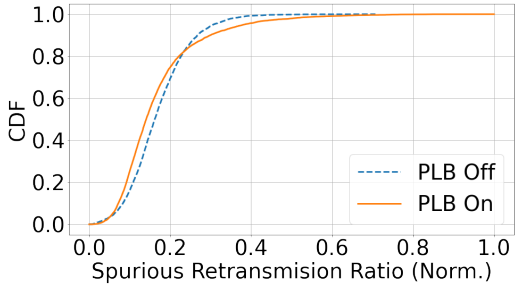


Figure 12: Little change in TCP spurious retransmissions.

Figure 12 shows that PLB-TCP has little effect on spurious retransmissions. Interestingly, the ratio is somewhat reduced below the 80th percentile and somewhat increased beyond that. We speculate that PLB is reducing the spurious retransmissions caused by the TCP Tail Loss Probe (TLP) [13]. TLP is the retransmission of the last unacknowledged packet (without repathing). It becomes spurious when the TCP sender underestimates the packet delivery rate due to a sudden congestion episode. As PLB reduces congestion for many small RPCs, it reduces this failure mode. Meanwhile, PLB repathing may still incur reordering that falsely triggers loss recovery on the smaller number of large transfers experiencing congestion. Pony Express has a similar inconsequential change of spurious retransmissions.

4.6 Summary

Our results consistently show that PLB: reduces switch link load imbalance; reduces switch packet drops; lowers application RPC tail latency and deadline misses; and incurs negligible transport CPU or retransmission penalties by repathing. PLB cannot help when there are no congestion hotspots, but where there are, it is highly effective. These results hold across our production fleet for TCP and Pony Express, and for ToR, spine, and WAN border switches.

5 TESTBED EVALUATION

We complement the production results with controlled testbed experiments to let us evaluate specific aspects of PLB. For widespread use, it is important that PLB works well in a variety of challenging situations:

(1) Poor WCMP weights. The WCMP weights that match traffic to capacity may be suboptimal for many reasons. Does PLB reduce congestion when this happens?
(2) Network failures. How does PLB react to link failures, which may congest the remaining links, and link repairs, which suddenly increase capacity?

	Small RPCs		Large RPCs		LI	#Repaths
	p50	p99	p50	p99		
1:1 PLB ON	1	3.5	5.33	20	1x	1x
1:2 PLB ON	1	3.5	5.33	20	1.3x	1.06x
1:2 PLB OFF	1.06	4.25	6.33	43	2x	

Table 1: Normalized Transfer Latency, LI and #repaths over TCP with 1:2 routing weight split across ToR uplinks.

(3) Co-existence. How does PLB interact with non-PLB traffic and co-exist with different transports (e.g. congestion control using ECN vs delay)?

We will see that PLB performs well in all these scenarios.

5.1 WCMP Interactions

ECMP is for situations that need an equal amount of traffic on each path. This is not always the case because outgoing links may not have the same capacity, and so WCMP adds weights to allow non-equal allocations. However, even WCMP weights may not match the network workload to its carrying capacity. Weights are often approximate due to switch table space limitations. Moreover, network topologies change abruptly with failures and datacenter workloads can change faster than updates from the SDN controller [17]. It is important that PLB reduce hotspots even in these scenarios.

To evaluate WCMP interactions, we use two ToRs in a production datacenter. The ToRs do not carry other traffic to minimize interference. Each ToR has 4 uplinks of 40 Gbps and 12 hosts that are connected via 40 Gbps links such that the ToR uplinks are oversubscribed. Hosts under one ToR act as clients, and connect in an all-to-all fashion to hosts under the other ToR, which act as servers. The clients use TCP and send small requests (16kB) over a total of 1440 connections and large requests (1MB) over 144 connections; the servers send 1-byte responses. Each connection sends RPCs in a closed loop, one after the other. Note that consecutive RPCs have (small) idle gaps due to application turn-around time.

1:2 WCMP Imbalance. In this experiment, we assign a weight of 2 to half the client ToR uplinks and weight 1 to the rest even though all paths have the same capacity. This imbalanced setup simulates suboptimal weights due to approximations in WCMP or traffic engineering, or capacity loss due to a failure. Actual capacity along all paths is the same but double traffic is routed towards paths with weight 2 with the expectation of double capacity along those paths.

Table 1 shows the LI and latency metrics for small and large RPCs for the baseline 1:1 weights with PLB ON, and the 1:2 weights with and without PLB. We see that PLB with 1:2 imbalance achieves performance close to the baseline by moving connections away from the congested paths. There is no latency degradation and little network degradation as the LI rises to 1.3X. Repathing across all connections increases from 470/sec to 500/sec (1.06X) since PLB needs to work harder to move traffic off congested links, but 99.9% of the repaths still happen after restarting from idle. Conversely, without PLB, both the latency and network are significantly degraded. The 99p latency for small and large RPCs increase by 20% and 115% respectively, and load imbalance rises 2x. These results show that PLB is very effective at matching traffic to the available capacity even when the WCMP weights are only approximate.

1:100 WCMP Imbalance. To understand the limits of PLB, we also experiment with extreme imbalance. We use the same setup and metrics as before with weights of 100 and 1. This setup might represent a misconfiguration where the network has inadvertently been put in a broken state.

	Small RPCs		Large RPCs		LI	#Repaths
	p50	p99	p50	p99		
1:1 PLB ON	1	3.5	5.33	20	1x	1x
1:100 PLB ON	0.91	2.5	12.3	46.67	3.5x	14x
1:100 PLB OFF	10.83	14.16	225	300	100x	

Table 2: Normalized Transfer Latency, LI and #repaths over TCP with 1:100 routing weight split across ToR uplinks

Table 2 shows a surprising result. Without PLB, a portion of network is largely unusable. Load imbalance grows by 100x as the paths with weight of 1 are little used, which lowers aggregate throughput by ~20%. Due to congestion, the 50p and 99p latency rise by 10x and 3x, respectively, for small RPCs, and by 41x and 14x, respectively, for large RPCs. With PLB, it is a different story. Repathing increases 14x over the baseline as PLB must work very hard to overcome the 99% chance of picking a congested link. Load imbalance then falls to 3.5x, latency falls to 2.5x or better, and the aggregate throughput recovers. That is, PLB can achieve degraded but usable performance despite the disastrous choice of weights.

Interestingly, the latency of small RPCs actually improves with PLB and 100:1 weights. The chance of repathing to a link of weight 1 is only 1%. Small RPCs repath more frequently in response to congestion and eventually find the links with low utilization, which reduces their 50p and 99p latency. Large RPCs are less likely to find these links, and more likely to congest them and be moved away. They see 50p and 99p latency increases of 130%. These results highlights how PLB differentiates small and large RPCs.

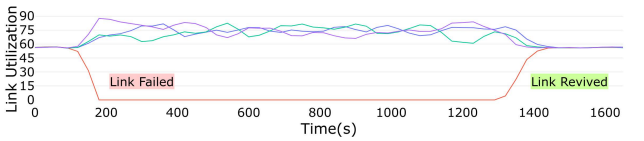
5.2 Link Failure and Recovery

We evaluate how PLB-TCP repaths traffic in response to link failure and recovery. We use the same setup as §5.1 with uniform weights. We simulate a link failure by disabling one of the four destination (server) ToR uplinks such that it drops all packets routed towards it for ~20 minutes. Routing does not remove the failed link from paths. Recall (§3.4) that connections pause ECN-triggered repathing after an RTO to avoid returning to a failed link right away. We use a pause period of 1 to 2 minutes.

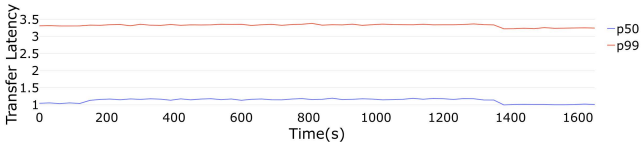
Figure 13 shows the time series of link utilization as the link fails and recovers. Note that while the 30-second average link utilization before the failure is 60%, the instantaneous utilization can reach 100% and trigger PLB repathing. When the link fails, impacted connections timeout, and PLB repaths them until they are spread over the working links (and no longer RTO). The working links become less balanced after the failure because some of the flows on them have disabled PLB for 1-2 minutes following an RTO. This happens throughout the failure as flows that experience congestion repath and are unlucky enough to move to the failed link, from which they must RTO to recover.

Despite some link imbalance, none of the connections are aborted. During the failure, the 50p and 99p latency of large RPCs increases by 22% and 140%, respectively, as to the working paths become more congested. For small RPCs, 50p latency increases by 12% whereas the 99p increase is negligible. Small RPCs repath more often (before and after PLB pauses) so they are more likely to find a less congested path. Finally, when the link recovers, PLB smoothly re-balances all links since there is no further RTO.

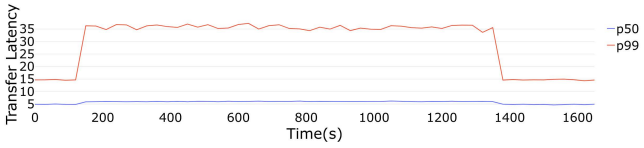
This test demonstrates that PLB is resilient to link failures, even prolonged ones that are not quickly repaired by routing. It is a vast



(a) Link Utilization



(b) Normalized small RPC (16kB) transfer latency



(c) Normalized large RPC(1MB) transfer latency

Figure 13: PLB behavior during link failure and recovery

improvement from the situation without PLB: RPCs using the failed link would be stuck until the TCP connection is reset, at which time they would abort and require application recovery.

5.3 Co-existence with Non-PLB Traffic

By design, PLB can be deployed to hosts incrementally to repath their traffic away from hotspots. The other hosts (even those receiving PLB traffic) do not need to be upgraded because PLB will react to congestion regardless of the cause. This is an important consideration in multi-tenant systems since tenants may upgrade at different times.

To evaluate this co-existence, we use another testbed[1]. Each ToR has 8 uplinks of 100Gbps and is connected to 18 hosts via 50Gbps links. Hosts under one ToR act as clients and the others act as servers. We send small 16KB RPCs over a total of 180 (18 x 10) connections and large 1MB RPCs over 90 (18 x 5) connections, all using TCP. There is no production traffic between these ToRs, though the test traffic does interact with production traffic in the fabric. PLB is enabled for 4 clients (~20% PLB) and disabled for the other 14 clients (~80% non-PLB). We also run an iteration where PLB is disabled for all clients to get a baseline for comparison.

We find that even this modest 20% deployment benefits PLB hosts and also non-PLB hosts. Table 3 gives the transfer latencies normalized using the baseline. PLB helps directly by moving flows that experience congestion to less-used paths. This shows up in the tail of large RPCs. The same movement indirectly helps the non-PLB flows because they face less competition for bandwidth. The system improvement is also seen in the ToR LI, which falls from 14% for the baseline (PLB disabled) to 6% (57% reduction). With these dynamics, PLB can deliver most of its benefit well before it is fully deployed.

5.4 Incast Fairness

Surprisingly, we observed that PLB helps with incast congestion by balancing load on the ingress links of the bottleneck switch, e.g., a destination ToR. This benefit is especially noticeable in certain switches with a chip architecture for which ingress imbalance causes egress unfairness. In these switches, the ports are split into

[1]Our experiments were run at different times, requiring a new testbed.

	Small RPCs			Large RPCs		
	p50	p99	p99.99	p50	p99	p99.99
PLB	0.97	1.57	2.43	4.56	14.38	19.01
non-PLB	1	1.61	2.74	5	15.24	20.52
Baseline	1	1.60	2.94	4.85	16.73	23.28

Table 3: Transfer latencies with 20% PLB and 80% non-PLB traffic. Normalized using 100% non-PLB traffic.

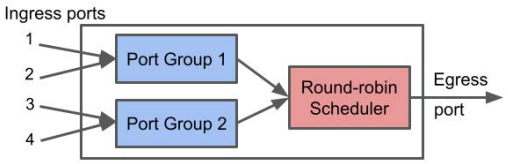


Figure 14: Scheduling over multiple port groups in a switch

	Max-Min Throughput Unfairness	Op Latency(ms)			
		median	p99	p99.9	p99.99
PLB ON	6.04%	0.90	7.04	15.31	28.78
PLB OFF	16.50%	1.07	7.40	17.02	35.26

Table 4: PLB improves incast fairness.

groups, each with its own data path and buffer. When many input ports contend for the same output port, the switch scheduler round-robins over the groups. This structure is shown in Figure 14. If the incast flows are unbalanced across these groups, then some flows will receive less service and see higher drops and lower bandwidth than other senders [1]. With PLB, the balance across groups is improved and fairness increases.

We show this benefit with an incast experiment on switches with two port groups. 45 servers send Pony Express traffic to the same destination client continuously over persistent connections. 22 servers are in the local rack, and 23 servers are in a remote rack. The results in Table 4 show that PLB reduced the throughput unfairness (max-min throughput divided by the fair-share) across servers by 60%. Latency is also reduced at both the median (15%) and tail (99.99p down by 18%). In effect, PLB has repathed around an unexpected hop internal to the switch that can become congested.

Note we focus on continuous incast workload instead of one-shot incast. The former workload causes persistent load imbalance and congestion which are more common and problematic in datacenters. Since PLB requires several round trips to react, it is not active in the startup phase of transient incast workload.

5.5 Applicability Across Congestion Controls

Our results show that PLB works with BBRv2 and Swift congestion control. We also study a third form of congestion control, BBR.swift [11]. This study lets us compare delay and ECN congestion controllers with the same TCP transport. Our intent is not to recommend one congestion control over another, but to show that PLB has broad applicability.

We use the testbed setup of §5.3 with small and large RPCs. We do one run for BBRv2, and another for BBR.swift. As expected, we did not observe significant differences in load balance or latency metrics. The LI values for each run remain within 1% of each other, while latency metrics remain within 5% of each other. The small latency difference favors BBR.swift and is due to subtle congestion signal and reaction differences. ECN is more noisy under bursty traffic, while delay is better at capping the tail latency.

6 LESSONS LEARNED

We give qualitative experiences to complement our results.

Problem. Initially, we were unclear about the extent of the load imbalance problem in practice, as reducing imbalance only helps between underload and overload. Our fleetwide results show that there is a large opportunity for improvement, even when the network is optimized with WCMP. Moreover, heavy-tails imply that load imbalance is not resolved by aggregation. We see it at ToRs, fabric core switches, and gateways to the WAN. This led to our focus on hotspots.

Design. The most striking feature of PLB is that it leverages transport protocols. This was a pragmatic choice for us, but we have found it to have important synergies. It lets us change the end-to-end path, rather than outputs at a single switch, so that upstream choices can push traffic away from downstream hotspots. It lets us repath after application idle periods, not simply packet timing gaps, which smoothly integrates load balancing and congestion control. The two need to play well together for application gains.

Congestion is a powerful metric for aligning traffic to the true carrying capacity. Designs that align traffic levels with WCMP weights can fail to resolve problems when the weights do not reflect the actual capacity or match the current workload (§4.2). Designs that equalize traffic along paths can fail to model behaviors that affect performance, such as switch chip limitations (§5.4). PLB defensively *protects* applications from suffering these harmful imbalances.

Deployment. We were pleasantly surprised by the extent of backwards-compatibility we could achieve, and the benefits of partial deployments. Switch upgrades generally need to precede host upgrades to deliver a benefit, but both can roll out in waves. Senders can turn on PLB for their connections at any time, without receiver upgrades or coordination. PLB delivers benefits to adopters when it is enabled, even if legacy traffic is causing congestion. In fact, PLB helps legacy traffic by moving traffic away from legacy hotspots (§5.3). PLB was smoothly rolled out globally without any interruption.

Limitations. PLB assumes IPv6 traffic, which is not yet the standard for the public Internet and may not be an option for some providers. It also requires the ability to modify host stacks and configure switches to use the IPv6 Flow Label. While convenient for us, we acknowledge that these requirements may not be viable for others. We hope that sharing our experience with PLB will lead to more widespread host and switch support for load balancing using the Flow Label.

7 RELATED WORK

There are many load balancing designs in the literature, but to the best of our knowledge none have been deployed at scale in datacenters.

The closest prior work is FlowBender [24] which needs to overload the VLAN tags to repath. We improve with a Flow Label architecture, preferential repathing of small transfers, greater attention to packet reordering, and other learnings.

Many switch designs spread load over output ports with a local measure of load. Vendor dynamic load balancing [2, 22] shifts flows based on link utilization, LocalFlow [37] bin-packs flows into outputs, Flare [39] spreads flowlets to even the load, and Drill [20] and DeTail [43] pick less-loaded outputs for each packet. Unlike PLB, these schemes do not handle asymmetric scenarios, which are important in datacenters. They are also prone to greater packet reordering, and require stateful switch processing that PLB does not.

Similarly, some host designs spread load over paths with static rules. Presto [31] spreads TSO-sized bursts across paths, while RPS [15] randomly sprays packets. These designs have the same difficulty with asymmetric scenarios and reordering as above. In addition, and unlike PLB, they do not interact well with unbalanced legacy traffic.

Other designs make balancing decisions based on the path load or congestion. At switches, CONGA[4], HULA [27] and Expeditus [42] exchange remote state to send flowlets along less-utilized paths. PLB also makes congestion-aware decisions, but it does not need switch support or extra control messages to do so. Nor does it restrict its design to apply to datacenter networks. LetFlow [41] is an interesting case because it randomly sprays flowlets but argues that transport behavior biases towards congestion-aware decisions. PLB makes congestion-aware decisions directly, without depending on transport behavior that may change, e.g., flowlets may reflect application duty cycles with pacing.

Host based congestion-aware load balancing designs are closest to PLB. Hermes [44] senses and avoids congestion, but it relies on actively probing path conditions. This is expensive for volatile traffic and unnecessary with PLB. Clove [26] detects congestion with ECN or INT [25], but it relies on overlay and virtualization technologies that are not always present. PLB integrates with the transport directly, and does not probe paths. MPTCP [9] breaks a connection into subflows that are adaptively filled to increase throughput.

Finally, Hedera [3], Mahout [14], MicroTE [8], FastPass [35] use a centralized approach to load balancing. By its nature, this approach requires control messages to collect information at the central server, which makes it hard for such designs to scale and be responsive at global scale. PLB free-rides the congestion state that already exists at hosts to make globally optimal decisions.

8 CONCLUSION

PLB is a load balancing solution that repaths IPv6 flows which experience congestion, driving down link imbalance and hotspots in the network. It is simple, requiring <50 lines of code update to transports plus ECMP/WCMP flow hashing on the Flow Label. It is general, being applicable to TCP and Pony Express on networks from datacenters to backbones. And our results show that it is highly effective at lowering link imbalance, packet loss, and the tail of RPC transfer latency. Global deployment of PLB gives us confidence that PLB provides across the board benefits in practice.

We hope that this paper will encourage others to leverage the IPv6 Flow Label, and that PLB will become widely available as a load balancing solution. We have open-sourced PLB code for TCP [21].

Acknowledgements: We would like to thank the anonymous reviewers and our shepherd James Hongyi Zeng for their insightful feedback. We are also grateful to Neal Cardwell and Kevin Yang for their invaluable help on PLB design and implementation.

REFERENCES

[1] 2018. Tolly Report: Mellanox Spectrum Switch vs. Broadcom Tomahawk. https://community.mellanox.com/s/article/tolly-report--mellanox-spectrum-switch-vs--broadcom-tomahawk. (2018).

[2] 2018. Trident 3 Dynamic Load Balancing. https://www.broadcom.com/video/b4684311367445439131 29cd6a0caa30. (2018).

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

[4] Mohammad Alizadeh, Tom Edsall, et al. 2014. CONGA: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM conference on SIGCOMM*. 503–514.

[5] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). In *SIGCOMM*.

[6] S Amante, B Carpenter, S Jiang, and J Rajahalme. 2011. RFC 6437: IPv6 flow label specification. *IETF, November* (2011).

[7] Theophilus Benson, Aditya Akella, and David A. Maltz. 2010. Network Traffic Characteristics of Data Centers in the Wild. In *IMC*. 267–280. https://doi.org/10.1145/1879141.1879175

[8] 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*. 1–12.

[9] Olivier Bonaventure, Christoph Paasch, Gregory Detal, et al. 2017. Use cases and operational experience with multipath TCP. *RFC 8041* (2017).

[10] Neal Cardwell, Yuchung Cheng, et al. 2019. BBR v2: A Model-based Congestion Control. IETF 105. https://datatracker.ietf.org/meeting/105/materials/slides-105-iccrg-bbr-v2-a-model-based-congestion-control-00. (2019).

[11] Neal Cardwell, Yuchung Cheng, et al. 2020. BBR Update:1: BBR.Swift; 2: Scalable Loss Handling. IETF 109. https://datatracker.ietf.org/meeting/109/materials/slides-109-iccrg-update-on-bbrv2-00. (Nov 2020).

[12] Brian Carpenter and Shane Amante. 2011. *Using the IPv6 flow label for equal cost multipath routing and link aggregation in tunnels*. Technical Report. RFC 6438, November.

[13] Yuchung Cheng, Neal Cardwell, Nandita Dukkipati, and Priyaranjan Jha. 2021. RFC 8985 The RACK-TLP Loss Detection Algorithm for TCP. (2021).

[14] Andrew R Curtis, Wonho Kim, and Praveen Yalagandula. 2011. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *2011 Proceedings IEEE INFOCOM*. IEEE, 1629–1637.

[15] Advait Abhay Dixit, Pawan Prakash, Y. Charlie Hu, and Ramana Rao Kompella. 2013. On the impact of packet spraying in data center networks.. In *INFOCOM*.

[16] William Feller. 1954. Diffusion processes in one dimension. *Trans. Amer. Math. Soc.* 77, 1 (1954), 1–31.

[17] Andrew D Ferguson, Steve Gribble, Chi-Yao Hong, Charles Edwin Killian, et al. 2021. Orion: Google's Software-Defined Networking Control Plane.. In *NSDI*. 83–98.

[18] Sally Floyd, Jamshid Mahdavi, Matt Mathis, and Matt Podolsky. 2000. RFC2883: An extension to the selective acknowledgement (SACK) option for TCP. (2000).

[19] Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, and Mohammad Alizadeh. 2016. Juggler: a practical reordering resilient network stack for datacenters. In *Proceedings of the Eleventh European Conference on Computer Systems*. 1–16.

[20] Soudeh Ghorbani, Zibin Yang, P Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. 2017. Drill: Micro load balancing for low-latency data center networks. In *Proceedings of the ACM SIGCOMM*. 225–238.

[21] Google. 2022. TCP-PLB source code. (2022). https://github.com/google/plb.

[22] Douglas Richard Hanks. [n. d.]. Juniper QFX10000 Series. Chapter 4. Performance and Scale. https://www.oreilly.com/library/view/juniper-qfx10000-series/9781491922248/ch04.html. ([n. d.]).

[23] Chi-Yao Hong, Subhasree Mandal, et al. 2018. B4 and after: managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined WAN. In *Proceedings of the ACM SIGCOMM*. 74–87.

[24] 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 ACM CoNEXT*. 149–160.

[25] Abdulkadir Karaagac and Jeroen Hoebeke. [n. d.]. *In-band Network Telemetry for 6TiSCH Networks*. Internet-Draft draft-karaagac-6tisch-int-00. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-karaagac-6tisch-int-00 Work in Progress.

[26] Naga Katta, Aditi Ghag, Mukesh Hira, Isaac Keslassy, Aran Bergman, Changhoon Kim, and Jennifer Rexford. 2017. Clove: Congestion-aware load balancing at the virtual edge. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. 323–335.

[27] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. 2016. Hula: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*. 1–12.

[28] Frank Kelly. 2003. Fairness and stability of end-to-end congestion control. *European journal of control* 9, 2-3 (2003), 159–176.

[29] William Knight and D. M. Bloom. 1973. E2386. *The American Mathematical Monthly* 80, 10 (1973), 1141–1142. http://www.jstor.org/stable/2318556

[30] Gautam Kumar, Nandita Dukkipati, et al. 2020. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *Proceedings of the SIGCOMM*.

[31] Ming Li, Deepak Ganesan, and Prashant Shenoy. 2009. PRESTO: Feedback-driven data management in sensor networks. *IEEE/ACM Transactions on Networking* 17, 4 (2009), 1256–1269.

[32] Steven H Low. 2003. A duality model of TCP and queue management algorithms. *IEEE/ACM Transactions On Networking* 11, 4 (2003), 525–536.

[33] Michael Marty, Marc de Kruijf, et al. 2019. Snap: a Microkernel Approach to Host Networking. In *SOSP*.

[34] Kihong Park, Gitae Kim, and Mark E Crovella. 1997. Effect of traffic self-similarity on network performance. In *Performance and Control of Network Systems*, Vol. 3231. International Society for Optics and Photonics, 296–310.

[35] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. 2014. Fastpass: A centralized zero-queue datacenter network. (2014).

[36] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. 2015. Inside the social network's (datacenter) network. In *Proceedings of the ACM SIGCOMM*. 123–137.

[37] Siddhartha Sen, David Shue, Sunghwan Ihm, and Michael J Freedman. 2013. Scalable, optimal flow routing in datacenters via local link balancing. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. 151–162.

[38] Arjun Singhvi, Aditya Akella, Dan Gibson, Thomas F Wenisch, et al. 2020. 1RMA: Re-Envisioning Remote Memory Access for Multi-Tenant Datacenters. In *Proceedings of the ACM SIGCOMM*. 708–721.

[39] Shan Sinha, Srikanth Kandula, and Dina Katabi. 2004. Harnessing TCP's burstiness with flowlet switching. In *Proc. 3rd ACM Workshop on Hot Topics in Networks (Hotnets-III)*.

[40] D Thaler and C Hopps. 2000. RFC 2991 Multipath Issues in Unicast and Multicast Next-Hop Selection. (2000).

[41] 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)*. 407–420.

[42] Peng Wang, Hong Xu, Zhixiong Niu, Dongsu Han, and Yongqiang Xiong. 2016. Expeditus: Congestion-aware load balancing in clos data center networks. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*. 442–455.

[43] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, and Randy Katz. 2012. DeTail: Reducing the flow completion time tail in datacenter networks. In *Proceedings of the ACM SIGCOMM*. 139–150.

[44] Hong Zhang, Junxue Zhang, Wei Bai, Kai Chen, and Mosharaf Chowdhury. 2017. Resilient datacenter load balancing in the wild. In *Proceedings of the ACM SIGCOMM*. 253–266.

[45] 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*. 1–14.