Improving Network Availability with Protective ReRoute

David Wetherall, Abdul Kabbani*, Van Jacobson, Jim Winget, Yuchung Cheng,
Charles B. Morrey III, Uma Moravapalle, Phillipa Gill, Steven Knight, and Amin Vahdat
Google

ABSTRACT

We present PRR (Protective ReRoute), a transport technique for shortening user-visible outages that complements routing repair. It can be added to any transport to provide benefits in multipath networks. PRR responds to flow connectivity failure signals, e.g., retransmission timeouts, by changing the FlowLabel on packets of the flow, which causes switches and hosts to choose a different network path that may avoid the outage. To enable it, we shifted our IPv6 network architecture to use the FlowLabel, so that hosts can change the paths of their flows without application involvement. PRR is deployed fleetwide at Google for TCP and Pony Express, where it has been protecting all production traffic for several years. It is also available to our Cloud customers. We find it highly effective for real outages. In a measurement study on our network backbones, adding PRR reduced the cumulative region-pair outage time over TCP with application-level recovery by 63–84%. This is the equivalent of adding 0.4–0.8 "nines" of availability.

CCS CONCEPTS

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

KEYWORDS

Network availability, Multipathing, FlowLabel

ACM Reference Format:
David Wetherall, Abdul Kabbani, Van Jacobson, Jim Winget, Yuchung Cheng, Charles B. Morrey III, Uma Moravapalle, Phillipa Gill, Steven Knight, and Amin Vahdat. 2023. Improving Network Availability with Protective ReRoute. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10, 2023, New York, NY, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3603269.3604867

1 INTRODUCTION

Hyperscalers operate networks that support services with billions of customers for use cases such as banking, hospitals and commerce. High network availability is a paramount concern. Since it is impossible to prevent faults in any large network, it is necessary to repair them quickly to avoid service interruptions. The classic repair strategies depend on routing protocols to rapidly shift traffic from failed links and switches onto working paths.

In our experience, the main barrier to high availability in large networks is the non-negligible number of faults for which routing does not quickly restore connectivity. Sometimes routing fails to help due to configuration mistakes, software bugs, or unexpected interactions that are only revealed after a precipitating event, such as a hardware fault. For example, bugs in switches may cause packets to be dropped without the switch also declaring the port down. Or switches may become isolated from their SDN controllers so that part of the network cannot be controlled. And routing or traffic engineering may use the wrong weights and overload links.

While unlikely, these complex faults do occur in hyperscaler networks due to their scale. They result in prolonged outages due to the need for higher-level repair workflows. This has an outsize impact on availability: a single 5 min outage means <99.99% monthly uptime. Qualitatively, outages that last minutes are highly disruptive for customers, while brief outages lasting seconds may not be noticed.

Perhaps surprisingly, routing is insufficient even for the outages that it can repair. Traffic control systems operate at multiple timescales. Fast reroute [3, 4] performs local repair within seconds to move traffic to backup paths for single faults. Routing performs a global repair, taking tens of seconds in very large networks to propagate topology updates, compute new routes, and install them at switches. Finally, traffic engineering responds within minutes to fit demand to capacity by adjusting path weights.

For routing alone to maintain high availability, we need fast reroute to always succeed. Yet a significant fraction of outages only fully recover via global routing and traffic engineering. This is because fast reroute depends on limited backup paths. These paths may not cover the actual fault because they are pre-computed for the Shared Risk Link Group (SRLG) [9] of a planned fault. Backup paths that do survive the fault are less performant than the failed paths they replace. Because the number of paths grows exponentially with path length, there are fewer local options for paths between the switches adjacent to a fault than global end-to-end paths that avoid the same fault. With limited options plus the capacity loss due to the fault, backup paths are easily overloaded.

Our reliability architecture shifts the responsibility for rapid repair from routing to hosts. Modern networks provide the opportunity to do so because they have scaled by adding more links, not only faster links. To add capacity, the links must be disjoint. This leads to multiple paths between pairs of endpoints that can fail independently. Thus the same strategy that scales capacity also adds diversity that can raise reliability.

Hosts typically see bimodal outage behavior: some connections take paths to a destination that are *black holes* which discard packets in the network, while other connections to the same destination take paths that continue to work correctly. This partial loss of connectivity is a consequence of multiple paths plus standard techniques to avoid single points of failure. For example, it is unlikely

*The author contributed to this work while at Google.

ACM SIGCOMM '23, September 10, 2023, New York, NY, USA

ACM ISBN 979-8-4007-0236-5/23/09.
https://doi.org/10.1145/3603269.3604867

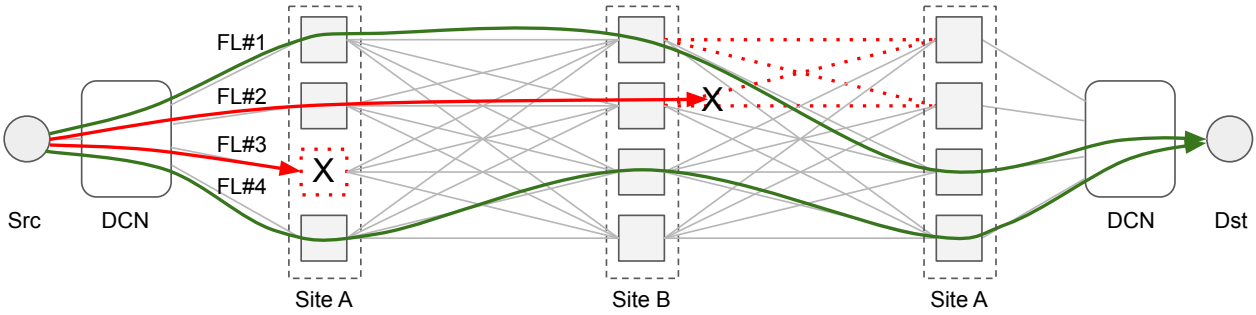


Figure 1: Multipath between hosts can route around faults shown in dotted red.

that a fault will impact geographically distinct fiber routes, or all shards of a network entity that is partitioned for reliability.

To restore connectivity, hosts must avoid failed paths and use working ones. This poses a dilemma because the IPv4 architecture does not let hosts influence the network path of a flow. Instead, with the rise of multiple paths, Equal-Cost Multi-Path (ECMP) [24, 48] routing at switches uses transport identifiers to spread flows across paths. This design ties a connection to a path. Fortunately, now that we have migrated Google networks to IPv6, we can leverage the IPv6 FlowLabel [2, 34] in its intended architectural role to influence the path of connections.

With Protective ReRoute (PRR), switches include the FlowLabel in the ECMP hash. Hosts then repath connections by changing the FlowLabel on their packets. Linux already supports this behavior for IPv6, building on the longstanding notion of host rerouting on transport failures, e.g., TCP timeouts. We completed this support by handling outages encountered by acknowledgement packets.

PRR repairs failed paths at RTT timescales and without added overhead. It is applicable to all reliable transports, including multipath ones (§2.5). It has a small implementation, and is incrementally deployable across hosts and switches.

For the last several years, PRR has run 24x7 on Google networks worldwide to protect all TCP and Pony Express [31] (an OS-bypass datacenter transport) traffic. Recently, we have extended it to cover Google Cloud traffic. The transition was lengthy due to the vendor lead times for IPv6 FlowLabel hashing, plus kernel and switch upgrades. But the availability improvements have been well worthwhile: fleetwide monitoring of our backbones over a 6-month study found that PRR reduced the cumulative region-pair outage time (sum of outage time between all pairs of regions in the network) over TCP with application-level recovery by 63–84%.

This work makes two contributions. We describe the nature of outages in a large network, using case studies to highlight long outages that undermine availability. And we present data on the effectiveness of PRR for our fleet as a whole, plus learnings from production experience. We hope our work will encourage others to leverage the IPv6 FlowLabel with an architecture in which routing provides hosts with many diverse paths, and transports shift flows to improve reliability.

This work does not raise ethical issues.

2 DESIGN

We describe PRR, starting with our network architecture, and then how outage detection and repathing work in it.

2.1 Multipath Architecture

PRR is intended for networks in which routing provides multiple paths between each pair of hosts, as is the case for most modern networks. For example, in Fig 1 we see an abstract WAN in which a pair of hosts use four paths at once out of many more possibilities. When there is a fault, some paths will lose connectivity but others may continue to work. In the figure, one path fails due to a switch failure, and another due to a link failure. But two paths do not fail despite these faults.

Multipath provides an opportunity for hosts to repair the fault for users. To do so, we need hosts to be able to control path selection for their connections. We shifted our IPv6 network architecture to use the FlowLabel [2] to accomplish this goal.

In the original IPv4 Internet, hosts were intentionally oblivious to the path taken by end-to-end flows. This separation between end hosts and routing was suited to a network that scaled by making each link run faster and thus had relatively few IP-level paths between hosts. Over the past two decades, networks have increasingly scaled capacity by adding many links in parallel, resulting in numerous additional viable paths. Flows must be spread across these paths to use the capacity well, and so multipathing with ECMP [24, 48] has become a core part of networks. The same paths that spread flows to use capacity for performance also increase diversity for reliability.

The IPv6 architecture allows hosts to directly manage the multiplicity of paths by using the FlowLabel to tell switches which set of packets needed to take the same path, without knowing the specifics of the path [34]. This arrangement lets transports run flows over individual paths while letting the IP network scale via parallel links. However, IPv6 was not widely deployed until around 2015, and this usage did not catch on. In the interim, ECMP used transport headers for multipathing, e.g., each switch hashes IP addresses and TCP/UDP ports of each packet to pseudo-randomly select one of the available next-hops. This approach eases the scaling problem, but limits each TCP connection to a fixed network path.

The adoption of IPv6 allows us to use the FlowLabel as intended. At switches, we include the FlowLabel in the ECMP hash, a capability which is broadly available across vendors[1]. As a pragmatic step, we extend ECMP to use the FlowLabel along with the usual 4-tuple. The key benefit is to let hosts change paths without changing transport identifiers. In Fig 1, one connection may be shifted across the four paths simply by changing the FlowLabel.

2.2 High-Level Algorithm

PRR builds on the concept of host rerouting in response to transport issues. An instance of PRR runs for each connection at a host to protect the forward path to the remote host. This is necessary because connections use different paths due to routing and ECMP, so one instance of PRR cannot learn working paths from another.

An outage for a connection occurs when its network path loses IP connectivity. If the connection is active during an outage, PRR will detect an outage event via the transport after a short time, as described in §2.3. The outage may be a forward, reverse, or bidirectional path failure. Unidirectional failures are quite common since routes between hosts are often asymmetric, due to traffic engineering onto non-shortest paths [23].

PRR triggers repathing intended to recover connectivity in response to the outage event, as described in §2.4. If the new path successfully recovers connectivity, the outage ends for the connection (even though the fault may persist). If not, PRR will detect a subsequent outage event after an interval, and again trigger repathing. It will continue in this manner until either recovery is successful, the outage ends for exogenous reasons (e.g. repair of the fault by the control plane), or the connection is terminated.

A fundamental ambiguity in this process is that a host alone cannot distinguish forward from reverse or bidirectional path failure. As a result, outage events may cause PRR to perform spurious repathing, which can slow recovery by causing a forward path failure where none existed. Fortunately, this does not affect correctness because outage events will continue to trigger repathing until both forward and reverse paths recover.

2.3 TCP Outage Detection

PRR can be implemented for all reliable transports. We describe how it works with TCP as an important example. Since network outages are uncommon, PRR must have acceptable overhead when there is no outage. This means PRR must be very lightweight in terms of host state, processing and messages. Our approach is to detect outages as part of normal TCP processing.

Data Path. There are many different choices for how to infer an outage, from packet losses to consecutive timeouts. For established connections, PRR takes each TCP RTO (Retransmission TimeOut) [35] in the Google network as an outage event. This method provides a signal that recurs at exponential backoff intervals while connectivity is sufficiently impaired that the connection is unable to make forward progress. It is possible that an RTO is spurious or indicates a remote host failure, but repathing is harmless in these situations (as it is either very likely to succeed or won't help because the fault is not network related).

[1]Support was limited when we began but has increased steadily.

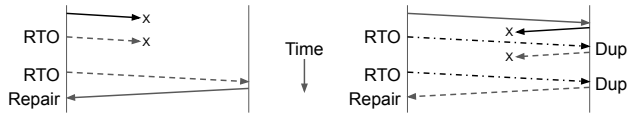


Figure 2: Example recovery of a Unidirectional Forward (left) and Reverse (right) fault. Non-solid lines indicate a changed FlowLabel.

ACK Path. Interestingly, RTOs are not sufficient for detecting reverse path outages. Failure of the ACK path will not cause an RTO for the ACK sender because ACKs are not themselves reliably acknowledged. Consider request/response traffic. The response will not be sent until the request has been fully received, which will not happen if the pure ACKs for a long request are lost.

We use the reception of duplicate data beginning with the second occurrence as a signal that the ACK path has failed. A single duplicate is often due to a spurious retransmission or use of Tail Loss Probes (TLP) [10], whereas a second duplicate is highly likely to indicate ACK loss.

Control Path. Finally, we detect outages in an analogous way when a new connection is being established. We use SYN timeouts in the client to server direction, and reception of SYN retransmissions in the server to client direction.

Performance. The performance of PRR depends on how quickly these sequences are driven by RTOs. Outside Google, a reasonable heuristic for the first RTO on established connections is RTO = 3RTT, with a minimum of 200ms. Inside Google, we use the default Linux TCP RTO formula [36] but reduce the lower-bound of RTTVAR and the maximum delayed ACK time to 5ms and 4ms from the default 200ms and 40ms, respectively [8]. Thus a reasonable heuristic is RTO = RTT + 5ms. It yields RTOs as low as single digit ms for metropolitan areas, tens of ms within a continent, and hundreds of ms for longer paths. These lower RTOs speed PRR by 3–40X over the outside heuristic. For new connections, a typical first SYN timeout is 1s, at the upper end of RTOs. This implies that connection establishment during outages will take significantly longer than repairing existing connections.

Examples. Recovery for request/response traffic with a unidirectional path fault is shown in Fig 2. For simplicity the figures focus on the initial transmission and later ones involving repathing, while omitting other packets present for fast recovery and TLP, etc. Each non-solid line indicates a changed FlowLabel.

In the forward case, the behavior is simple: each RTO triggers retransmission with repathing (dashed line). This continues until a working forward path is found. Recovery is then direct since the reverse path works. The number of repaths (here two) is probabilistic. The expected value depends on the fraction of working paths.

The reverse case differs in two respects. The RTOs cause spurious repathing (dot-dash line) since the forward path was already working. However, it is not harmful because only the reverse direction is faulty. Duplicate detection at the remote host (after TLP which is not shown) causes reverse repathing until a working path is found. The recovery time is the same as the forward case despite these differences.

Bidirectional faults are more involved because the repair behavior depends on whether the connection initially failed on the

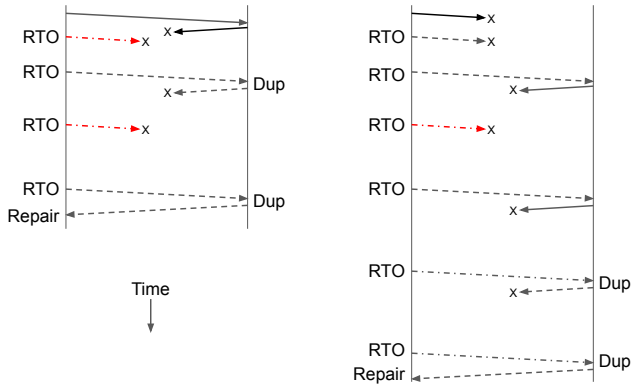


Figure 3: Example recovery of a Bidirectional fault when only the Reverse path (left) or both directions (right) initially failed. Non-solid lines indicate a changed FlowLabel. Red dash-dot lines indicate harmful repathing.

forward path, reverse path, or in both directions. Unlike the case of a unidirectional fault, we may have harmful spurious repathing and delayed reverse repathing.

If the connection initially experiences a forward path failure but (by chance) no reverse failure, then recovery proceeds as for the unidirectional case. When the forward repair succeeds, it is the first time the packet reaches the receiver. Thus the receiver will not repath. It will use its working reverse path to complete recovery.

Conversely, if the connection initially fails only on the reverse path (Fig 3 left) then we see different behavior. At the first RTO, spurious forward repathing occurs. Now it can be harmful, and is in this example (dot-dash red line). It causes a forward path loss that requires subsequent RTOs to repair. When the forward path works, duplicate reception causes reverse repathing. We must draw both a working forward and reverse path at the same time to recover.

The longest recovery occurs when the connection has failed in both directions (Fig 3 right). Reverse repathing is needed for recovery but delayed until after two repairs of the forward path. This is because the receiver first repaths on the second duplicate reception and the original transmission and TLP are lost. To complete recovery, we need a subsequent joint forward and reverse repair. As before, spurious forward repathing may slow this process.

2.4 Repathing with the FlowLabel

PRR repaths as a local action by using the FlowLabel. It does not rely on communication with other network components (e.g., SDN controllers or routing) during outages.

Random Repathing. For each outage event, PRR randomly changes the FlowLabel value used in the packets of the connection. This behavior has been the default in Linux via `txhash` since 2015, with our last upstream change (for repathing ACKs) landing in 2018. The OS takes this action without involving applications, which greatly simplifies deployment.

With a good ECMP hash function, different inputs are equivalent to a random draw of the available next-hops at each switch. If we define a path as the concatenation of choices at each switch, then paths more than a few switches long will change with very high probability. However, the key question is how often a new path will intersect the fault.

The fraction of failed paths for the outage must be measured empirically since it depends on the nature of the fault. It also varies for each prefix-pair and changes over time. Often the outage fraction is small, leading to rapid recovery. For example, with a 25% outage (in which a quarter of the paths fail) a single random draw will succeed 75% of the time. More generally, for an IP prefix-pair with a p% outage, the probability of a connection being in outage after N rerouting attempts falls as p^N.

Avoiding Cascades Repathing needs to avoid cascade failures, where shifting a large amount of traffic in response to a problem focuses load elsewhere and causes a subsequent problem. PRR is superior to routing in this respect: fast-reroute shifts many "live" connections in the same way at the same time, while PRR shifts traffic more gradually and smoothly.

The shift is gradual because each TCP connection independently reacts to the outage, which spreads reaction times out at RTO timescales. Each connection is quiescent when it moves, following an RTO, and will ramp its sending rate under congestion control.

The shift is smooth because random repathing loads working paths according to their routing weights. The expected load increase on each working path due to repathing in one RTO interval is bounded by the outage fraction. For example, it is 50% for a 50% outage: half the connections repath and half of them (or a quarter) land on the other half of paths that remain. This increase is at most 2X, and usually significantly lower, which is no worse than TCP slow-start [25] and comfortably within the adaptation range of congestion control. Moreover, PRR can only use policy-compliant paths which have acceptable properties such as latency, while there are no such guarantees for bypass routes.

A related concern is that repathing in response to an outage will leave traffic concentrated on a portion of the network after the outage has concluded. However, this does not seem to be the case in practice: routing updates spread traffic by randomizing the ECMP hash mapping, and connection churn also corrects imbalance.

2.5 Alternatives

Multipath Transports. A different approach is to use multipath transports such as MPTCP [6] or SRD [38] since connections that use several paths are more likely to survive an outage; these transports can also improve performance in the datacenter [33]. However, PRR may be applied to any transport to boost reliability, including multipath ones. For example, MPTCP can lose all paths by chance, and it is vulnerable during connection establishment since subflows are only added after a successful three-way handshake. PRR protects against these cases.

Moreover, many connections are lightly used, so the resource and complexity cost of maintaining multiple paths does not have a corresponding performance benefit. The prospect of migrating all datacenter usage to a new transport was also a consideration. Instead, we apply PRR to TCP and Pony Express to increase the reliability of our existing transports.

Application-Level Recovery. Applications can approximate PRR without IPv6 or the FlowLabel by reestablishing TCP connections

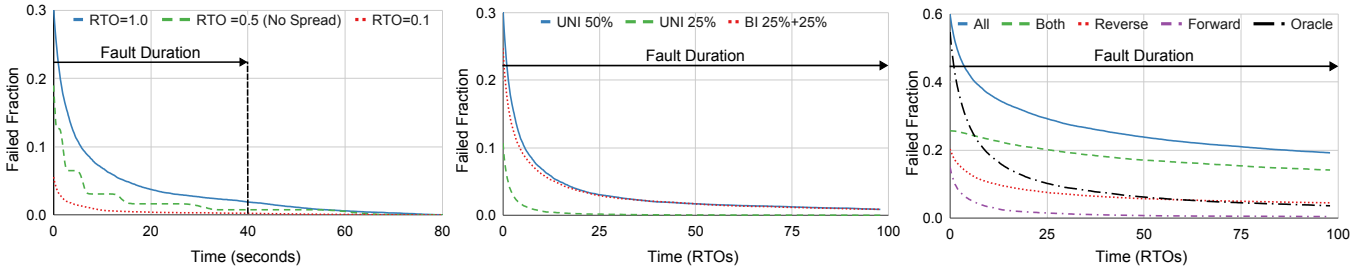


Figure 4: (a) Effect of RTO (b) Uni- and bi-directional repair curves (c) Breakdown of bidirectional repair

that have not made progress after a timeout[2]. We relied on this approach before PRR. However, using low RPC timeout values is much more expensive than PRR, as well as less performant. This is because establishing an RPC channel takes several RTTs and has computational overhead for security handshakes. Coverage is also more limited since it relies on application developers. Adding PRR to TCP covers all manner of applications, including control traffic such as BGP and OpenFlow, whether originating at switches or hosts, and achieves the ideal of avoiding application recovery.

PLB. Finally, PRR is closely related to Protective Load Balancing [32]. In our implementation, they are unified to use the same repathing mechanism but for different purposes. PLB repaths using congestion signals (from ECN and network queuing delay) to balance load. PRR repaths using connectivity signals (e.g., timeouts) to repair outages. These signals coexist without difficulty, but there is one point of interaction. PRR activates during an outage to move traffic to a new working path. Since outages reduce capacity, it is possible that PLB will then activate due to subsequent network congestion and repath back to a failed path. Therefore, we pause PLB after PRR activates to avoid oscillations and a longer recovery.

3 SIMULATION RESULTS

We use a simple model to predict how PRR reduces the fraction of failed connections. Our purpose is to provide the reader with a mental model for how PRR is expected to perform; we will see these effects in our case studies (§4.2).

We simulate repathing driven by TCP exponential backoff for an ensemble of 20K long-lived connections under various fault models. This workload represents the active probing that we use to measure connectivity. The fault starts at $t = 0$ and impacts each connection when it first sends at $t \geq 0$. We model black hole loss and ignore congestive loss. A connection is considered failed if a packet is not acknowledged within 2s. The repair behavior is strongly influenced by two factors that we consider in turn: the RTO (Retransmission TimeOut), and the outage fraction of failed paths.

RTO. The RTO depends on the connection RTT and state, and the TCP implementation. For our network the RTT ranges from O(1ms) in a metro, to O(10ms) in a continent, to O(100ms) globally. For new connections, the SYN RTO is commonly 1s. This variation has a corresponding effect on the speed of recovery.

Fig 4(a) shows the repair of a 50% outage (i.e., half the paths fail) in one direction for three different RTO scenarios. The middle line in the graph is the repair curve for connections having RTOs clustered around a median of 0.5s, with most mass from 0.45 to 0.55s. They are generated with a log-normal distribution, LogN(0, 0.06), scaled by the median RTO of 0.5s. The connections also have 1s of jitter in their start times. The aggregate behavior is a "step" pattern, where the failed fraction is reduced by 50% when the connections repath at each step. Note that the failed fraction starts at around 0.2, much lower than the 50% of connections that were initially black holed. This is because the black holed connections RTO and most recover before the 2s timeout.

While instructive, we only see this step pattern on homogeneous subsets of data. More typical are the smooth curves of the top and bottom lines due to connections repathing at different times. They have median RTOs of 1s and 100ms, respectively, and are generated with LogN(0, 0.6) scaled by the median RTO. This distribution spreads the RTOs to have a standard deviation that is 10X larger than before. The bottom line shows a much faster repair due to the lower median RTO and has become smooth due to the RTO spread. The smaller 100ms RTO makes a large difference. It both reduces the initial failed fraction and reaches successive RTOs more quickly. The top line, with initial RTOs around 1s, models the failure rate for new connections as well as long RTTs. It shows a correspondingly slower repair due to the larger RTO.

For both curves, the failed fraction of connections, f, falls polynomially over time. Suppose the outage fails a fraction paths p. After N RTOs, f is p^N below its starting value, which is exponentially lower. However, the RTOs are also exponentially-spaced in time, t, so we have $t \approx 2^N$ for the Nth RTO. Combining expressions, $f \approx p^{log_2(t)} = 1/(t)^K$, for $K = -log_p(2)$. Thus for $p = \frac{1}{2}$, the failure probability falls as $1/t$. For $p = \frac{1}{4}$, it falls as $1/t^2$.

A notable effect is that the fault (dashed line) ends at $t = 40$s, yet some connections still lack connectivity until $t = 80$s. That is, the failures visible to TCP can last longer than the IP-level outage. The reason is exponential backoff. It is not until the first retry after the fault has resolved that the connection will succeed in delivering a packet and recover. If the fault ends at $t = 40$s then some connections may see failures in the interval $[20, 40)$. These connections will increase their RTO and retry in the interval $[40, 80)$.

Outage Fraction. The severity of the fault has a similarly large impact on recovery time. Fig 4(b) shows repair for three different long-lived faults. We normalize time in units of median initial RTOs, spread RTOs as before, and use a timeout of twice the median RTO.

The top solid line is for a 50% outage in one direction. It corresponds to the 1s RTO curve from before. The bottom solid line shows the repair of a 25% outage in one direction. It has the same

[2]Linux fails TCP connections after ~15 mins by default. Application timeouts are shorter.

effects, but starts from a lower failed fraction and falls more quickly. Now, each RTO repairs 75% of the remaining connections.

The dashed line shows the repair of a bidirectional outage in which 25% of paths fail in each direction. For each connection, the forward and reverse paths fail independently to model asymmetric routing. This curve is similar to the 50% unidirectional outage, even though it might be expected to recover more quickly since the probability of picking a working forward and reverse path is $\frac{9}{16}$, which is larger than $\frac{1}{2}$. The reason is that the bidirectional outage has three components that repair at different rates, as we see next.

In Fig 4(c), we break the repair of a long-lived 50% forward and 50% reverse outage (solid line) into its components (dashed lines) as described in §2.3. This outage is demanding, with 75% of the round-trip paths having failed, so the tail falls by only one quarter at each RTO. Connections that initially failed in one direction only, either forward or reverse, are repaired most quickly. Connections that initially failed in both directions are repaired slowly due to spurious repathing and the delayed onset of reverse repathing. To see the cost of these effects, the Oracle line (dotted) shows the how the failed fraction improves without them.

Summary. We conclude that for established connections with small RTOs, PRR will repair >95% of connections within seconds for faults that black hole up to half the paths. This repair is fast enough that the black holes are typically not noticed. Larger RTOs and new connections will require tens of seconds for the same level of repair, and show up as a small service interruption. PRR cannot avoid a noticeable service interruption for the combination of large RTOs and faults that black hole the vast majority of paths, though it will still drive recovery over time.

4 PRODUCTION RESULTS

PRR runs in Google networks 24x7 and fleetwide for TCP and Pony Express traffic. We present a measurement study to show how it is able to maintain high network availability for users, beginning with case studies of outages and ending with fleet impact.

Our study observes real outages at global scale across the entire production network. It covers two backbones, B2 and B4 [23, 26], that use widely differing technologies (from MPLS to SDN) and so have different fault and repair behaviors. Further, the results we derive are consistent with service experience to the best of our knowledge. One limitation of our study is that we are unable to present data on service experience. We report results for long-lived probing flows instead. Still, our measurements are comprehensive, with literally trillions of probes.

4.1 Measuring loss

We monitor loss in our network using active probing. We send probes between datacenter clusters using multiple flows, defined as source/destination IP and ports. Flows take different paths due to ECMP. Each flow sends ~120 probes per minute. Each pair of clusters is probed by at least 200 flows. This arrangement lets us look at loss over time and loss over paths, with high resolution in each dimension. We also aggregate measurements to pairs of network regions, where each region is roughly a metropolitan area and contains multiple clusters.

We use three types of probes to observe loss at different network layers. First, UDP probes measure packet loss at the IP level. We refer to these probes as L3. They let us monitor the connectivity of the underlying network, which highlights faults and routing recovery, but not how services experience the network.

To measure application performance before PRR, we use empty Stubby RPCs as probes; Stubby is an internal version of the open source gRPC [18] remote procedure call framework. We refer to these probes as L7. They benefit from TCP reliability and RPC timeouts, which reestablish TCP connections that are not making progress. An L7 probe is considered lost if the RPC does not complete within 2s. Stubby reestablishes TCP connections after 20s to match the gRPC default timeout.

Finally, to measure application performance with PRR, we issue the L7 probes with PRR enabled, which we refer to as L7/PRR. These probes benefit from PRR repathing as well as TCP reliability and RPC timeouts. Note that the network may be in outage while applications are not, due to PRR, TCP and RPC repair mechanisms. Thus comparing the three sets of probes lets us understand the benefits of PRR relative to applications without it and the underlying network.

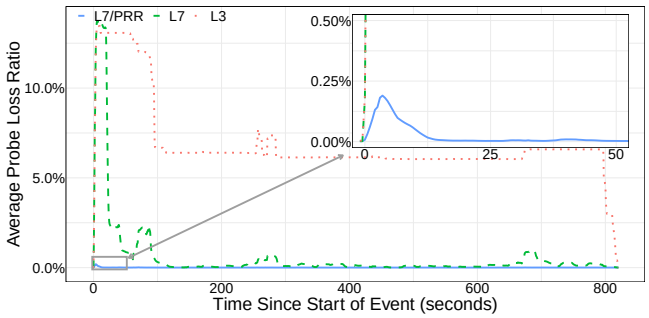
Our fleet summaries use probe data for 6 months to the middle of 2023 between all region-pairs in our core network, and on both backbones. Since the functionality to disable PRR is not present on all probe machines, we note that there are slightly fewer L7 probes (29%) than L7/PRR (37%) and L3 probes (33%), but we have no reason to believe the results are not representative.

4.2 Case Studies

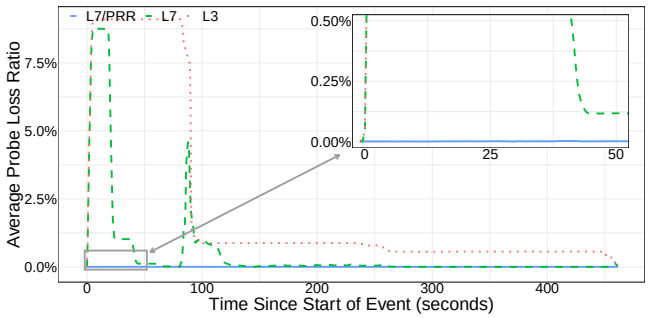
We begin with case studies of how PRR behaves during a diverse set of significant outages. Most outages are brief or small outages. The long and large outages we use for case studies are exceptional. They are worthy of study because they are highly disruptive to users, unless repaired by PRR or other methods.

Case Study 1: Complex B4 Outage. The first outage is the longest we consider, lasting 14 mins. It impacted region-pairs connected by the B4 backbone. We use it to highlight multiple levels of repair operating at different timescales. It was caused by a rare dual power failure that took down one rack of switches in a B4 supernode [23] and disconnected the remainder of the supernode from an SDN controller. It was prolonged by a repair workflow that was blocked by unrelated maintenance. Long outages have diverse causes and typically complex behaviors. In this case, a single power failure would not have led to an outage, and a successful repair workflow would have led to a much shorter outage, but in this unlucky case three events happened together.

The probe loss during the outage is shown in Fig 5. The top graph shows the loss versus time for L3, L7 and L7/PRR probes over impacted inter-continental region-pairs over B4. The bottom graph shows the same for intra-continental pairs. We separate them since the behaviors are qualitatively different. One datapoint covers 0.5s, so the graphs show a detailed view of how the fault degraded connectivity over time. Each datapoint is averaged over many thousands of flows sending in the same interval, which exercised many thousands of paths.

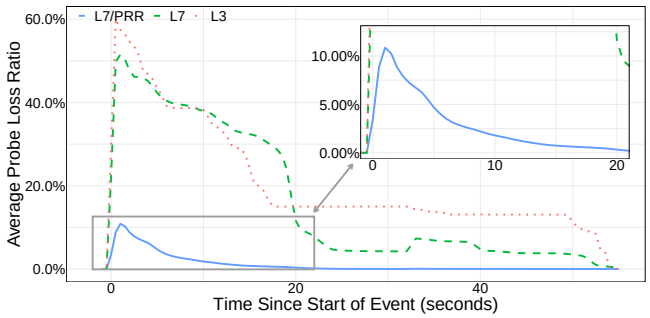


(a) Inter-continental probe loss

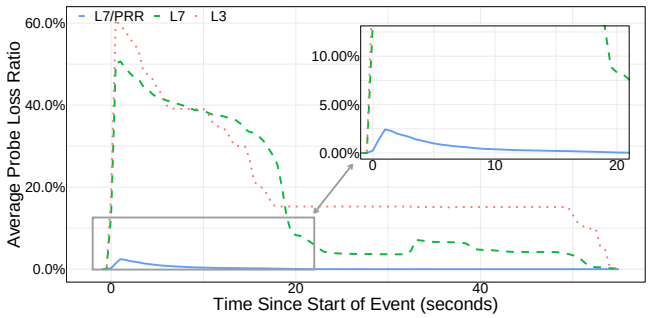


(b) Intra-continental probe loss

Figure 5: Probe loss during a complex B4 outage.



(a) Inter-continental probe loss



(b) Intra-continental probe loss

Figure 6: Probe loss during an optical link failure on B4.

Consider first the L3 line, which shows the IP-level connectivity. Since the SDN control plane was disconnected by the fault, it could not program fixes to drive a fast repair process. Around 100s, global routing systems intervened to reroute traffic not originating from or destined for the outage neighborhood. This action reduced the severity of the outage but did not fix it completely. After more than 10 mins, the drain workflow removed the faulty portion of the network from service to complete the repair.

The loss rate stayed below 13% throughout the outage because only one B4 supernode was affected by the fault so most flows transited healthy supernodes. However, the outage was more disruptive than the loss rate may suggest because the failure was bimodal, as is typical for non-congestive outages: all flows taking the faulty supernode saw 100% loss, while all flows taking the healthy supernodes saw normal, low loss. For customers using some of the faulty paths, the network was effectively broken (without PRR) because it would stall applications even though most paths still functioned properly.

The L7 line shows the outage recovery behavior prior to the development of PRR. The L7 loss rate started out the same as L3, but dropped greatly after 20s, after which it decayed slowly, with occasional spikes. The L7 improvement is due to the RPC layer, i.e., application-level recovery, which opened a new connection after 20s without progress. These new connections with different port numbers avoided the outage by taking different network paths due to ECMP. Most of the new paths worked by chance because the L3 loss rate tells us that on average only 13% of paths initially failed. This is similar to the repathing done by PRR except that (1) by using the FlowLabel, PRR can repath without reestablishing the TCP connection; and (2) PRR operates at RTT timescales, which are much shorter than the 20s RPC timeout.

Some TCP flows required multiple attempts to find working paths, so there was a tail as connectivity was restored. The subsequent spikes arose when routing updates changed paths, altering the ECMP mapping and causing some working connections to become black holed. TCP retransmissions were of little help in this process, since a connection either worked or lost all of its packets at a given time.

Finally, the L7/PRR line shows the outage recovery using the same RPC probes as the L7 case but with PRR enabled. The loss rate was greatly reduced, to the point of being visible only in the inset panel. The repair was roughly 100X more rapid than the L7 case, especially for the intra-continental case due to its shorter RTT. It achieved the desired result: most customers were unaware that there was an outage because the connectivity interruption is brief and does not trigger application recovery mechanisms.

This case study highlights outage characteristics that we observe more broadly. Many outages black hole a fraction of paths between a region-pair while leaving many other paths working at the same time. And some outages are not repaired by fast reroute so they have long durations that are disruptive for users without quick recovery. Outages with both characteristics provide an opportunity for PRR to raise network availability.

Case Study 2: Optical failure. Next we consider an optical link failure that resulted in partial capacity loss for the B4 backbone. Fig 6 shows the probe loss over time for inter- and intra-continental paths during the outage. In this case, L3 loss was around 60% when

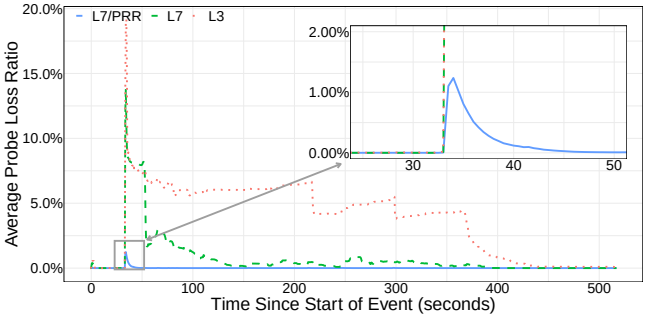


Figure 7: Inter-continent probe loss during a device failure on B2. (No intra-continent probe loss was observed.)

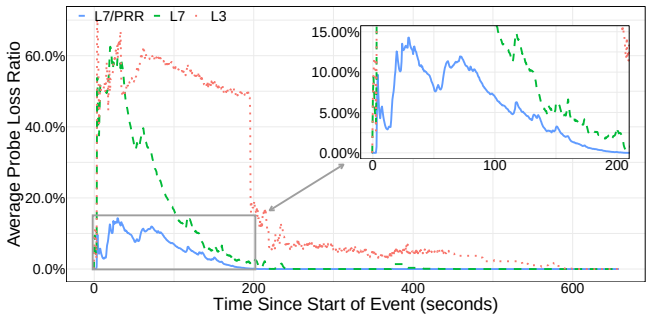


Figure 8: Intra-continental probe loss for a regional fiber cut in B2. (The inter-continental graph is omitted as similar.)

the event began, indicating that most paths had failed but there was still working capacity.

Immediately after the outage began, fast routing repair mechanisms acted to reduce the L3 loss to ~40% within 5s. Further improvements gradually reduced the L3 loss to ~20% by around 20s from the start of the event. The cause of this sluggish repair was congestion on bypass links due to the loss of a large fraction of outgoing capacity, and SDN programming delays due to the need to update many routes at once. Finally, the outage was resolved after 60 seconds when unresponsive data plane elements were avoided using traffic engineering.

The L7 line starts out slightly lower then the L3 one because L7 probes have a timeout of 2s, during which time routing was able to repair some paths. We see that TCP was unable to mitigate probe loss during the first 20s since retransmissions do not help more than routing recovery. In fact, after around 10s, L7 loss exceeded L3 loss because the detection of working paths was delayed by exponential backoff. Around 20s, RPC channel reestablishment roughly halves the loss rate for the remainder of the outage for both intra- and inter-continental paths. All these effects are consistent with our simulation results (§3).

In contrast, PRR lowered peak probe loss and quickly resolved the outage. For intra-continental paths, L7/PRR reduced the peak probe loss to 2.4% and had completely mitigated the loss by 20s into the outage. L7/PRR similarly performed well for inter-continental paths where probe loss peaked at around 11%, which is over 5X less than the peak L3 probe loss. This outage illustrated how the path RTT affects PRR. Consistent with simulation (§3), intra-continental paths that have lower RTTs observed a lower peak and faster resolution than inter-continental paths. In both cases, PRR greatly reduced probe loss beyond L7.

Case Study 3: Line card issues on a single device. The next outage involved a single device in our B2 backbone (Fig 7). During the outage, the device had two line-cards malfunction, which caused probe loss for some inter-continental paths. Due to the nature of the malfunction, routing did not respond. The outage was eventually mitigated when an automated procedure drained load from the device and took it out of service.

While the cause of this outage is different than the others, we see similar results: PRR was able to greatly reduce loss. In this case, the

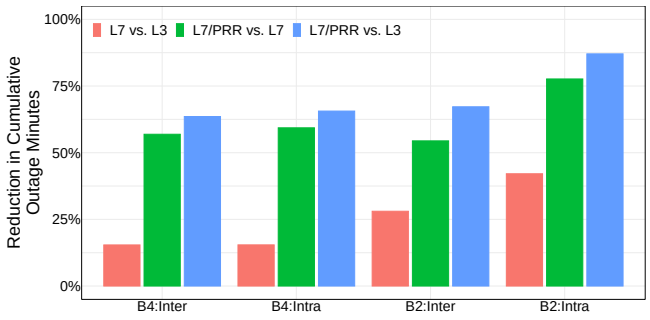


Figure 9: Reduction in outage minutes for B2 and B4 intra- and inter-continental paths.

peak probe loss seen by L3 was 19%. L7/PRR reduced this peak loss over 15X to 1.2% and, as with the prior outage, quickly lowered the loss level to near zero after 20 seconds. Conversely, the L7 probe loss has a large peak of 14% and persists for significantly longer than L7/PRR.

Case Study 4: Regional fiber cut. Finally, we present an outage that challenged PRR (Fig 8). In this outage, a fiber cut caused a significant loss of capacity. The average L3 probe loss peaked at 70% and remained around 50% or higher for 3 mins. This severe fault impacted many paths. Fast reroute did not mitigate it because the bypass paths were overloaded due to the large capacity reduction. After ~3 mins, global routing moved traffic away from the outage, lowering the loss rate and alleviating congestion on the bypass paths.

L7/PRR reduced the peak loss to 14%, a 5X improvement. It was much more effective than L7, which reduced peak loss to 65%, but was not able to fully repair this outage because of the large fraction of path loss. This path loss was exacerbated by routing updates during the event: PRR moved connections to working paths only for some of the connections to shift back to failed paths when the ECMP mapping was changed. As a result, we see a pattern in which L7/PRR loss fell over time but was interrupted with a series of spikes.

4.3 Aggregate Improvements

Case studies provide a detailed view of how PRR repairs individual outages. To quantify how PRR raises overall network availability, we aggregate measurements for all outages across all region-pairs in the Google network for the 6-month study period. The vast majority of the total outage time is comprised of brief or small outages. Our results show that PRR is effective for these outages, as well as long and large outages.

Outage Minutes Our goal is to raise availability, which is defined as $MTBF/(MTBF + MTTR)$, where $MTBF$ is the Mean Time Between Failures and $MTTR$ is the Mean Time to Repair. This formula is equivalent to 1 minus the fraction of outage time. Since we are unable to report absolute availability measures due to confidentiality, we report relative reductions in outage time across L3, L7, and L7/PRR layers. These relative reductions translate directly to availability gains. For instance, a 90% reduction in outage time is equivalent to adding one "nine" to availability, e.g., hypothetically improving from 99% to 99.9%

We measure outage time for each of L3, L7 and L7/PRR in minutes derived from flow-level probe loss. Specifically, we compute the probe loss rate of each flow over each minute. If a flow has more than 5% loss, such that it exceeds the low, acceptable loss of normal conditions, then we mark it as lossy. If a 1-minute interval between a pair of network regions has more than 5% of lossy flow, such that it is not an isolated flow issue, then it is an outage minute for that region-pair. We further trim the minute to 10s intervals having probe loss to avoid counting a whole minute for outages that start or end within the minute.

In Fig 9, we show the percent of total outage minutes that were repaired for L7/PRR probes relative to L3 probes. We give results for both B2 and B4 backbones, and broken out by intra- and inter-continental paths. Similarly, we compare L7 (without PRR) with L3 and L7/PRR with L7 to understand where the gains come from. The results are computed across many thousands of region-pairs and include hundreds of outage events.

PRR reduces outage minutes by 64-87% over L3. PRR combined with transport and application layer recovery mechanisms is very effective at shortening IP network outages for applications. We see large reductions in outage minutes when using L7/PRR for both backbone networks. The reductions range from 64% for inter-continental paths on B4, to 87% for intra-continental paths on B2. Note that, unlike for individual outages, we do not see a consistent pattern between inter- and inter-continental results across outages. This is because PRR effectiveness depends on topological factors and not only the RTT.

PRR reduces outage minutes by 54-78% over L7. PRR is able to repair most of the outage minutes that are not repaired by the TCP and application-level recovery of L7 probes. This result confirms that most of the L7/PRR improvement over L3 is not due to the RPC layer and TCP retransmissions; L7 reduces the cumulative outage minutes by only 15–42% relative to L3. We believe that a large portion of this gain is coming from RPC reconnects, since TCP retransmission is ineffective for black holes.

PRR performs well over time. As a check, we also look at how PRR behavior varies over time. Fig 10 shows the Generalized Additive Model (GAM) smoothing [43] of the fraction of daily outage minutes repaired. We see some variation over time, reflecting the varying nature of outages, while PRR consistently delivers large reductions in outage minutes throughout the study period.

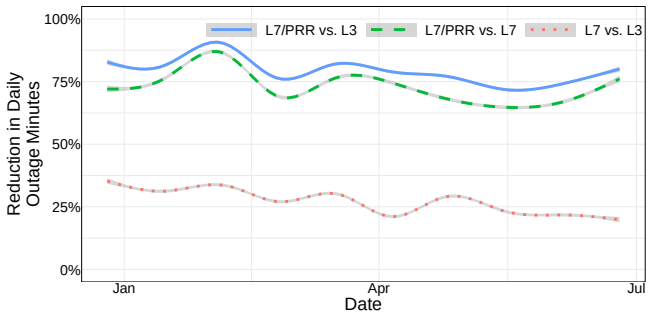


Figure 10: Fraction of outage minutes reduced over time.

4.4 Effectiveness for Region-Pairs

Outages affect multiple region-pairs, each of which may see a different repair behavior. We next look at how the benefit of PRR is distributed across region-pairs in our network. Fig 11 shows the Complementary Cumulative Distribution Function (CCDF) over region-pairs of the fraction of outage minutes repaired between layers. This graph covers all region-pairs in the fleet over our entire study period. Points higher and further to the right are better outcomes as they mean a larger fraction of region-pairs repaired a greater fraction of outage minutes.

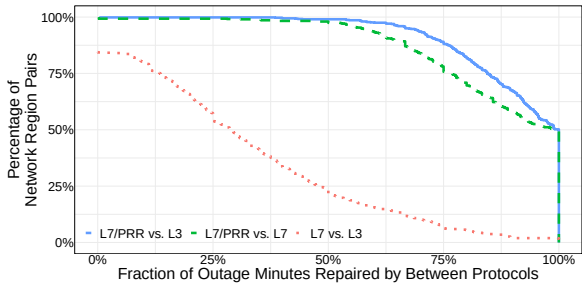
PRR performs well for a variety of paths. We see that the vast majority of region-pairs see a large benefit from L7/PRR over L3 on both backbones. It is able to repair 100% of outage minutes for 50% and 16% of B2 intra- and inter-continental region-pairs, respectively. PRR performance is more varied for B4 where outage minutes are decreased by half for 63% and 77% of intra- and inter-continental region-pairs, respectively.

PRR improves significantly over L7. As expected, L7/PRR provides much greater benefit than L7. The lines showing PRR gain are quite similar whether they are relative to L3 or L7 and show a reduction in outage minutes for nearly all region-pairs. (The exceptions tend to be region-pairs with very few outage minutes for which L7/PRR dynamics for sampling were unlucky.) Conversely, L7 without PRR increases the number of outage minutes relative to L3 for 3-16% of region pairs. This counter-intuitive result is possible because TCP exponential backoff on failed paths tends to prolong outage times (until RPC timeouts are reached).

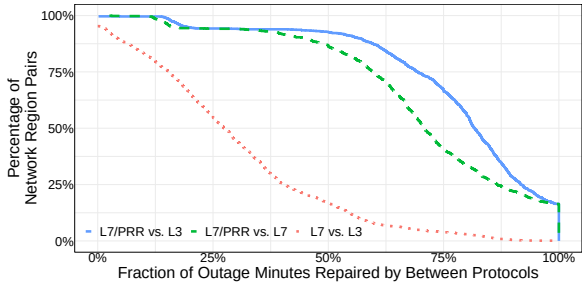
5 DISCUSSION

We briefly discuss some additional aspects of PRR.

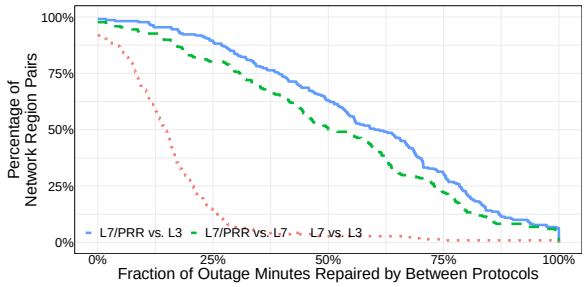
Other Transports. PRR can be applied to protect all transports, including multipath ones, since all reliable transports detect delivery failures. For example, we use PRR with Pony Express [31] OS-bypass traffic with minor differences from TCP. User-space UDP transports can implement repathing by using syscalls to alter the FlowLabel when they detect network problems. Even protocols



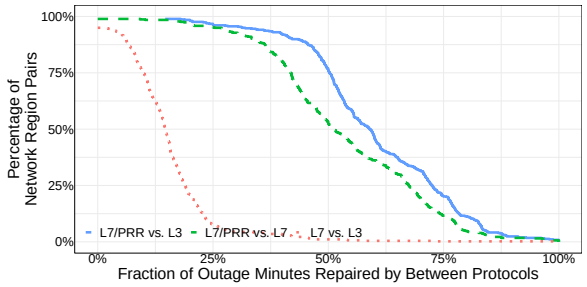
(a) intra-continental (B2)



(b) inter-continental (B2)



(c) intra-continental (B4)



(d) inter-continental (B4)

Figure 11: CCDF of improvement across region pairs.

such as DNS and SNMP can change the FlowLabel on retries to improve reliability.

Cloud & Encapsulation. PRR must be extended to protect IPv6 traffic from Cloud customers because virtualization affects ECMP. Google Cloud virtualization [12] uses PSP encryption [19], adding IP/UDP/PSP headers to the original VM packet as shown in Fig 12. In the network, switches use the outer headers for ECMP and ignore the VM packet headers. To enable the VM to repath via the FlowLabel, we hash the VM headers into the outer headers.

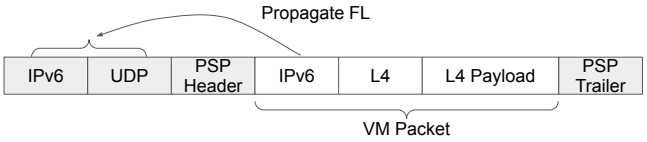


Figure 12: PSP Encapsulation

Now, when the guest OS has PRR and a TCP connection detects an outage and changes its FlowLabel, the encapsulation headers also change and hence ECMP causes the connection to be repathed. For different encapsulation formats, e.g., IPSEC, the details will vary, but the propagation approach is the same.

We also use encapsulation to enable PRR for Cloud IPv4 traffic. The gve [20] driver passes connection metadata to the hypervisor, which hashes it into the encapsulation headers. This technique works for other IPv4 networks as well: encapsulate with IPv4 to provide a layer of indirection, and propagate inner header entropy to an outer UDP header (since there is no FlowLabel).

Deployment. Backwards-compatibility and incremental deployment are highly desirable to protect existing deployments. PRR excels in this respect. Its rollout was lengthy due to vendor participation, upstream kernel changes, and switch upgrades, but otherwise straightforward. Hosts could be upgraded in any order to use the FlowLabel for outgoing TCP traffic. Switches could be concurrently upgraded in any order to ECMP hash on the FlowLabel; this change is harmless given the semantics of the FlowLabel.

It is not necessary for all switches to hash on the FlowLabel for PRR to work, only some switches upstream of the fault. Often, substantial protection is achieved by upgrading only a fraction of switches. This property has implications for the reliability of IPv6 traffic in the global Internet. Not only can each provider enable FlowLabel hashing to protect their own network, but upstream providers have some ability to work around downstream faults by changing the ingress into downstream networks.

6 RELATED WORK

Most work on improving network reliability focuses on network internals such as fast reroute [3, 4], backbones [23, 40, 47], or datacenters [44]. Fewer papers report on the causes of outages [17, 21, 41] or availability metrics suited to them, such as windowed-availability [22], which separates short from long outages.

Hosts have the potential to raise availability using the FlowLabel [5], but no large-scale deployment or evaluation has been presented to the best of our knowledge. Most host-side work focuses on multipath transports like MPTCP [6], SRD [38], and multipath QUIC [13] that send messages over a set of network paths. While they primarily aim to improve performance, these transports also increase availability, e.g., MPTCP may reroute data in one subflow to another upon RTO. However, they use only a small set of paths and may not protect the reliability of connection establishment, e.g., MPTCP adds paths only after a successful three-way handshake [5]. PRR can be added to multipath transports to increase reliability by exploring new paths until it finds working ones, and protecting connection establishment.

There is a large body of work on a related host behavior: multipathing for load balancing [1, 6, 14–16, 27–30, 32, 37, 39, 42, 45, 46]. Some of this work considers reliability, for example, CLOVE [28] uses ECMP to map out working paths. PRR shows this mapping is not necessary, as random path draws work well. Most of this work is not done in the context of IPv6 and does not consider the FlowLabel; we find it apt for multipathing. In our network, PRR and PLB [32] are implemented together, using repathing for both load balancing and rerouting around failures.

Finally, [7] argues that hosts should play a greater role in path selection, instead of routers reacting to failures. PRR is one realization of this argument.

7 CONCLUSION

PRR is deployed fleet-wide at Google, where it has protected the reliability of nearly all production traffic for several years. It is also available to our Cloud customers. PRR greatly shortens user-visible outages. In a 6-month study on two network backbones, it reduced the cumulative region-pair outage time over TCP with application-level recovery by 63–84%. This is the equivalent of adding 0.4–0.8 "nines" to availability. We now require that all our transports use PRR.

PRR (and its sister technique, PLB [32]) represent a shift in our network architecture. In the early Internet [11], the network instructed hosts how to behave, e.g., ICMP Source Quench. This did not work well, and in the modern Internet neither hosts nor routers instruct each other: hosts send packets, and routers decide how to handle them.

In our architecture, hosts instruct the network how to select paths for their traffic by using the IPv6 FlowLabel. This shift has come about because networks scale capacity by adding parallel links, which has greatly increased the diversity of paths. To make the most of this diversity, we rely on routing to provide hosts access to many paths, and hosts to shift traffic flows across paths to increase reliability and performance. We hope this architectural approach, enabled by the FlowLabel, will become widespread.

REFERENCES

[1] 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

[2] Shane Amante, Jarno Rajahalme, Brian E. Carpenter, and Sheng Jiang. 2011. IPv6 Flow Label Specification. RFC 6437. (Nov. 2011). https://doi.org/10.17487/RFC6437

[3] Alia Atlas, George Swallow, and Ping Pan. 2005. Fast Reroute Extensions to RSVP-TE for LSP Tunnels. RFC 4090. (May 2005). https://doi.org/10.17487/RFC4090

[4] Alia Atlas and Alex D. Zinin. 2008. Basic Specification for IP Fast Reroute: Loop-Free Alternates. RFC 5286. (Sept. 2008). https://doi.org/10.17487/RFC5286

[5] Alexander Azimov. 2020. Self-healing Network or The Magic of Flow Label. https://ripe82.ripe.net/presentations/20-azimov.ripe82.pdf. (2020).

[6] Olivier Bonaventure, Christoph Paasch, and Gregory Detal. 2017. Use Cases and Operational Experience with Multipath TCP. RFC 8041. (Jan. 2017). https://doi.org/10.17487/RFC8041

[7] Matthew Caesar, Martin Casado, Teemu Koponen, Jennifer Rexford, and Scott Shenker. 2010. Dynamic Route Recomputation Considered Harmful. *ACM SIGCOMM Computer Communication Review* 40, 2 (Apr. 2010), 66–71. https://doi.org/10.1145/1764873.1764885

[8] Neal Cardwell, Yuchung Cheng, and Eric Dumazet. 2016. TCP Options for Low Latency: Maximum ACK Delay and Microsecond Timestamps, IETF 97 tcpm. https://datatracker.ietf.org/meeting/97/materials/slides-97-tcpm-tcp-options-for-low-latency-00. (2016).

[9] Sid Chaudhuri, Gisli Hjalmtysson, and Jennifer Yates. 2000. Control of lightpaths in an optical network. In *Optical Internetworking Forum*.

[10] Yuchung Cheng, Neal Cardwell, Nandita Dukkipati, and Priyaranjan Jha. 2021. The RACK-TLP Loss Detection Algorithm for TCP. RFC 8985. (Feb. 2021). https://doi.org/10.17487/RFC8985

[11] David Clark. 1988. The Design Philosophy of the DARPA Internet Protocols. *SIGCOMM Comput. Commun. Rev.* 18, 4 (aug 1988), 106–114. https://doi.org/10.1145/52325.52336

[12] Mike Dalton, David Schultz, Ahsan Arefin, Alex Docauer, Anshuman Gupta, Brian Matthew Fahs, Dima Rubinstein, Enrique Cauich Zermeno, Erik Rubow, Jake Adriaens, Jesse L Alpert, Jing Ai, Jon Olson, Kevin P. DeCabooter, Marc Asher de Kruijf, Nan Hua, Nathan Lewis, Nikhil Kasinadhuni, Riccardo Crepaldi, Srinivas Krishnan, Subbaiah Venkata, Yossi Richter, Uday Naik, and Amin Vahdat. 2018. Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018*. USENIX Association, Renton, WA, 373–387.

[13] Quentin De Coninck and Olivier Bonaventure. 2017. Multipath QUIC: Design and Evaluation. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*.

[14] 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

[15] 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 (EuroSys '16)*. Association for Computing Machinery, New York, NY, USA, Article 20, 16 pages. https://doi.org/10.1145/2901318.2901334

[16] 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 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 225–238. https://doi.org/10.1145/3098822.3098839

[17] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. 2011. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. *SIGCOMM Comput. Commun. Rev.* 41, 4 (aug 2011), 350–361.

[18] Google. 2015. gRPC Motivation and Design Principles (2015-09-08). https://grpc.io/blog/principles/. (2015).

[19] Google. 2022. PSP Architecture Specification (2022-11-17). https://github.com/google/psp/blob/main/doc/PSP_Arch_Spec.pdf. (2022).

[20] Google. 2022. Using Google Virtual NIC. https://cloud.google.com/compute/docs/networking/using-gvnic. (2022).

[21] Ramesh Govindan, Ina Minei, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. 2016. Evolve or Die: High-Availability Design Principles Drawn from Googles Network Infrastructure. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 58–72. https://doi.org/10.1145/2934872.2934891

[22] Tamás Hauer, Philipp Hoffmann, John Lunney, Dan Ardelean, and Amer Diwan. 2020. Meaningful Availability. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 545–557. https://www.usenix.org/conference/nsdi20/presentation/hauer

[23] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Kondapa Naidu B., Chandan Bhagat, Sourabh Jain, Jay Kaimal, Shiyu Liang, Kirill Mendelev, Steve Padgett, Faro Rabe, Saikat Ray, Malveeka Tewari, Matt Tierney, Monika Zahn, Jonathan Zolla, Joon Ong, and Amin Vahdat. 2018. B4 and after: Managing Hierarchy, Partitioning, and Asymmetry for Availability and Scale in Google's Software-Defined WAN. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 74–87. https://doi.org/10.1145/3230543.3230545

[24] Christian Hopps and Dave Thaler. 2000. Multipath Issues in Unicast and Multicast Next-Hop Selection. RFC 2991. (Nov. 2000). https://doi.org/10.17487/RFC2991

[25] Van Jacobson. 1988. Congestion Avoidance and Control. *SIGCOMM Comput. Commun. Rev.* 18, 4 (aug 1988), 314–329. https://doi.org/10.1145/52325.52356

[26] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2013. B4: Experience with a Globally-Deployed Software Defined Wan. *SIGCOMM Comput. Commun. Rev.* 43, 4 (aug 2013), 3–14.

[27] 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

[28] 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 (CoNEXT '17). Association for Computing Machinery, New York, NY, USA, 323–335. https://doi.org/10.1145/3143361.3143401

[29] 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 (SOSR '16)*. Association for Computing Machinery, New York, NY, USA, Article 10, 12 pages. https://doi.org/10.1145/2890955.2890968

[30] 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. https://doi.org/10.1109/TNET.2008.2006818

[31] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Mike Dalton, Nandita Dukkipati, William C. Evans, Steve Gribble, Nicholas Kidd, Roman Kononov, Gautam Kumar, Carl Mauer, Emily Musick, Lena Olson, Mike Ryan, Erik Rubow, Kevin Springborn, Paul Turner, Valas Valancius, Xi Wang, and Amin Vahdat. 2019. Snap: a Microkernel Approach to Host Networking. In *In ACM SIGOPS 27th Symposium on Operating Systems Principles*. New York, NY, USA.

[32] 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

[33] 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.

[34] Jarno Rajahalme, Alex Conta, Brian E. Carpenter, and Dr. Steve E Deering. 2004. IPv6 Flow Label Specification. RFC 3697. (March 2004). https://doi.org/10.17487/RFC3697

[35] Matt Sargent, Jerry Chu, Dr. Vern Paxson, and Mark Allman. 2011. Computing TCP's Retransmission Timer. RFC 6298. (June 2011). https://doi.org/10.17487/RFC6298

[36] Pasi Sarolahti and Alexey Kuznetsov. 2002. Congestion Control in Linux TCP. In *2002 USENIX Annual Technical Conference (USENIX ATC 02)*. USENIX Association, Monterey, CA. https://www.usenix.org/conference/2002-usenix-annual-technical-conference/congestion-control-linux-tcp

[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 (CoNEXT '13)*. Association for Computing Machinery, New York, NY, USA, 151–162. https://doi.org/10.1145/2535372.2535397

[38] 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

[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] Sucha Supittayapornpong, Barath Raghavan, and Ramesh Govindan. 2019. Towards Highly Available Clos-Based WAN Routers. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 424–440. https://doi.org/10.1145/3341302.3342086

[41] Daniel Turner, Kirill Levchenko, Alex C. Snoeren, and Stefan Savage. 2010. California Fault Lines: Understanding the Causes and Impact of Network Failures. *SIGCOMM Comput. Commun. Rev.* 40, 4 (aug 2010), 315–326. https://doi.org/10.1145/1851275.1851220

[42] 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

[43] Simon N Wood. 2017. *Generalized additive models: an introduction with R* (second ed.). Chapman and Hall/CRC, Boca Raton. https://doi.org/10.1201/9781315370279

[44] Dingming Wu, Yiting Xia, Xiaoye Steven Sun, Xin Sunny Huang, Simbarashe Dzinamarira, and T. S. Eugene Ng. 2018. Masking Failures from Application Performance in Data Center Networks with Shareable Backup. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 176–190. https://doi.org/10.1145/3230543.3230577

[45] 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 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '12)*. Association for Computing Machinery, New York, NY, USA, 139–150. https://doi.org/10.1145/2342356.2342390

[46] 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

[47] Zhizhen Zhong, Manya Ghobadi, Alaa Khaddaj, Jonathan Leach, Yiting Xia, and Ying Zhang. 2021. ARROW: Restoration-Aware Traffic Engineering. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 560–579. https://doi.org/10.1145/3452296.3472921

[48] 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. Article No. 5. https://dl.acm.org/doi/10.1145/2592798.2592803