Staying Alive: Connection Path Reselection at the Edge

Raul Landa, Lorenzo Saino, Lennert Buytenhek, and Joao Taveira Araujo, *Fastly*

https://www.usenix.org/conference/nsdi21/presentation/landa

This paper is included in the Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation.

April 12–14, 2021

978-1-939133-21-2

Open access to the Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation is sponsored by

NetApp®

Staying Alive: Connection Path Reselection at the Edge

Raul Landa, Lorenzo Saino, Lennert Buytenhek and João Taveira Araújo
Fastly

Abstract

Internet path failure recovery relies on routing protocols, such as BGP. However, routing can take minutes to detect failures and reconverge; in some cases, like partial failures or severe performance degradation, it may never intervene. For large scale network outages, such as those caused by route leaks, bypassing the affected party completely may be the only effective solution.

This paper presents Connection Path Reselection (CPR), a novel system that operates on *edge networks* such as Content Delivery Networks and edge peering facilities [52, 64] and augments TCP to deliver transparent, scalable, multipath-aware end-to-end path failure recovery.

The key intuition behind it is that edge networks need not rely on BGP to learn of path impairments: they can infer the status of a path by monitoring transport-layer forward progress, and then reroute stalled flows onto healthy paths. Unlike routing protocols such as BGP, CPR operates at the timescale of round-trip times, providing connection recovery in *seconds* rather than minutes. By delegating routing responsibilities to the edge hosts themselves, CPR achieves *per-connection* re-routing protection for *all* destination prefixes without incurring additional costs reconstructing transport protocol state within the network. Unlike previous multipath-aware transport protocols, CPR is *unilaterally deployable* and has been running in production at a large edge network for over two years.

1 Introduction

Survivability is a core design tenet of the Internet, and a key factor in its enduring success. In reviewing the formative years of the DARPA Internet Protocols, Clark [19] listed survivability second in importance only to the top level goal of interconnecting existing networks:

> It was an assumption in this architecture that synchronization would never be lost unless there was no physical path over which any sort of communication could be achieved. In other words, at the top of transport, there is only one failure, and it is total partition. The architecture was to mask completely any transient failure.

The Internet today falls short of this assumption. Failures are not only common on the Internet, they are often visible [25, 28, 29], sometimes spectacularly so [39, 41, 42, 53, 54]. While large outages are rare, transient reachability fluctuations, colloquially referred to as *internet weather*, are frequent [25]. Attempting to prevent all sources of outages is an exercise in futility: failures are endemic to every component at every layer along every path on the Internet, and subsets of components interact to form complex failure conditions which cannot be anticipated. Instead, the most cost-effective way of improving reliability on the Internet is to *circumvent failures when they occur*. Traditionally, this task has fallen upon network providers, who rely on routing protocols such as the Border Gateway Protocol (BGP) and Open Shortest Path First (OSPF) to route around failures. Routing alone, however, is not enough.

Firstly, since the successful delivery of keepalive messages does not imply the successful delivery of client traffic, routing protocols can only detect a subset of failure conditions. For example, a BGP session may be hashed onto a healthy Link Aggregation Group (LAG) member, while other links in the same group falter. Misconfigurations, such as those that result in route leaks, can impact large swathes of the Internet [26, 35, 41, 53]. Such events, undetected by routing protocols, often require manual intervention to mitigate at significant cost to the stakeholders involved.

Secondly, the time required by routing to re-establish a consistent state after a failure increases with the size of the network and can take minutes [37], during which loops and blackholes can occur [28, 29]. Even detection itself can be slow. The recommended value for the *BGP hold timer* (the time after which a non-responsive BGP peer is marked as failed) is 90s [33, 48], but many implementations set the default value as high as 180s [4, 5, 11, 18] or even 240s [23]. Even with sophisticated monitoring infrastructure [17, 32, 51],

transient performance degradations have often disappeared by the time corrective updates can fully propagate.

Rather than relying on routing reconfiguration, protocols such as SCTP [55] and MPTCP [24] propose pushing the responsibility for mitigating path failures to the transport layer. Unfortunately, multipath transport protocols have struggled with adoption, and are today still circumscribed to niche use cases. Although multipath-aware protocols provide compelling reliability improvements, the current Internet architecture provides limited means (and incentives) for network providers to push multipath options out towards clients. With deployments limited to subsets of traffic or client population [16, 36, 46], their promise remains largely unfulfilled.

Edge networks [51, 52, 64] provide a natural vantage point from which to improve reliability, acting as a critical intermediary between application/content providers, hosted on highly centralized cloud infrastructure, and a globally distributed set of clients. By providing caching, security and compute functions as close to clients as possible, edge networks have positioned themselves to carry most of the customer facing traffic on the Internet [51, 52, 64]. Further, because they are expected to commit to SLAs guaranteeing the successful processing of end user requests, they end up bearing the costs of network layer failures and have a tremendous economic incentive to improve reliability. Conveniently, they also have the means. Unlike access/transit providers, edge networks have end-to-end visibility of traffic, and can therefore detect and react to failures faster and at finer granularity. By design, edge networks are multihomed and have access to better path diversity than end clients.

This paper presents Connection Path Reselection (CPR), a software-based approach improving the end-to-end reliability of edge network traffic. The key intuition behind it is that edge networks do not need to rely on BGP to learn of path degradation: they can infer the status of a path by monitoring transport-layer forward progress, and then reroute stalled flows onto healthy paths. This not only improves connection recovery, but also allows traffic to be shifted on a per-flow basis, greatly reducing the likelihood of load-induced cascading failures. Unlike previous proposals, CPR is unilaterally deployable, applicable to all flows, and simple to configure. Its implementation is entirely contained in a server-side kernel patch; it does not require programmable switches or any extra infrastructure. CPR has been in production for over two years at Fastly, a multi-Tbps edge cloud provider, where it successfully mitigates ~120 degradation events every day, each ~8 minutes long (over ~16 hours per day).

Having described our motivation, the remainder of this paper is organized as follows. First, we discuss the background of this work (§2), explain first how CPR detects path impairments (§3) and reroutes traffic as a result (§4). We then share results from production measurements (§5), followed by some operational considerations (§6). Finally, we compare CPR with related work (§7) and present our conclusions (§8).

2 Background and motivation

Edge networks (as understood in *e.g.,* [52, 64]) have unique characteristics that must be considered when designing a mechanism to improve end-to-end customer traffic reliability.

Edge networks support a diverse and changing set of applications. Early edge networks such as Content Delivery Networks (CDNs) were designed to support a narrow segment of Internet traffic: large, static content that could benefit from caching. This narrow traffic profile allowed for a wide set of potential optimizations. Edge networks have since evolved to support a much wider set of use cases (security, edge compute) and applications which no longer fit a neat traffic profile: a video client may need to retrieve small manifest files before requesting video chunks, or a browser session may download cached assets while maintaining a long-polling connection over which it receives update notifications. As such, edge networks today represent a microcosm of Internet traffic, not a segment. While it may be tempting to focus our efforts on improving reliability for a subset of traffic, performance degradation on any flow can adversely impact an entire application. A further complication is that it is not always a given that end clients will retry. For example, packaging a container image can involve retrieving potentially hundreds of individual assets. Failure to acquire any single one of these assets can result in the entire build process failing, at which point a user must decide whether to retry. This implies that we must detect *any* potential source of failure, for *every* type of flow, independently of its source, length, or capabilities of the end-client. We must also take into account that most traffic towards end-users will likely flow through middleboxes.

Edge networks are constrained by physical capacity. Points of Presence (POPs) are limited by physical space, and are designed to maximize the number of requests per second (RPS) they can serve [59]. Peak RPS is primarily dictated by storage and compute capacity - not bandwidth. Unlike traditional cloud environments, we cannot increase the physical footprint of network hardware, since that would necessarily reduce the amount of hardware dedicated to serving requests. Given our motivation for improving reliability is to reduce costs, we can not do so at the expense of efficiency.

Edge networks have unpredictable traffic patterns. Edge networks are subject to sudden fluctuations in demand due to flashcrowds or DDoS attacks. While physical capacity at any given POP is fixed, operators can shift traffic between POPs by adjusting DNS and BGP anycast configurations. This traffic engineering wreaks havoc on any potential solution that acts only on a set of heavy-hitter prefixes which is periodically updated. For instance, a POP located in Los Angeles may only observe traffic from prefixes in southern California in normal conditions, but this can change abruptly if *e.g.,* a POP in San Jose undergoes maintenance, or if a DDoS attack targets POPs in Japan. Traffic patterns shift dramatically during significant congestion and routing events, which is

precisely when reliability suffers the most. Our design takes these constraints into account, and presents a system which strives to:

- detect and react to failures affecting *any flow, at any point in its lifetime*;
- minimize operational and infrastructure costs;
- interact safely with concurrent traffic engineering and routing processes.

Such a system is possible because edge networks support routing architectures which expose multipath capability to end servers [57,64]. By maintaining multiple routing tables in the kernel and allowing transport sockets and userspace applications to select which one to use on a per-packet basis, edge servers can override standard BGP route selection and instead implement objective-driven routing policy by themselves.

The benefits of path diversity have been amply studied (*e.g.,* [21,60]), in particular for stub networks [27]; even when performance gains are not forthcoming, cost benefits can be achieved [6]. Previous work on path-switching revealed that it is possible to improve average path loss performance by an order of magnitude on average by dynamically switching paths [56]. Previous work on CDN multihoming demonstrated 25% performance improvement for 3 out of 4 metro areas simply by selecting the best of two transit providers, with comparable reliability improvement [7]. One specific type of simple path diversity is highly prevalent: *path load balancing*. 72% of source-destination network pairs explored in [12, 13] show evidence of load balancing; for ~12% of these, load balanced paths are asymmetric and explicit selection can significantly improve end-to-end latency. A more recent study [47] showed even more significant benefits: not only do paths from large cloud providers show latency differences between load-balanced paths exceeding 20ms to 21% of public IPv4 addresses; 8 pairs of datacenters were found to have latency differences between load-balanced paths exceeding 40ms. Path diversity is high for edge networks: in the CPR deployment presented here each POP typically connects to a few transit providers and several peers[1].

This is the starting point for Connection Path Reselection (CPR). *Given the multipath capabilities of edge servers, how can we extend TCP to circumvent failures?* Focusing on TCP is appealing not only because its congestion control and loss recovery mechanisms are sufficient on their own for a large class of end-to-end impairments, but also because *any* automated, short-timescale re-routing of large volumes of traffic can override traffic engineering and create undesirable traffic distributions. By performing path selection decisions at the *connection* (rather than the *route/prefix*) level, CPR minimizes its impact on traffic volumes.

CPR is embedded within TCP and implemented as a server-side patch to the Linux kernel, and relies on extensions to the `tcp_sock` struct to keep essential re-route-related state.

[1] See §5.3 and appendix A.1 for further details.

Because the kernel itself abstracts the IP address version at the socket level, CPR naturally covers both IPv4 and IPv6 traffic. Since CPR is parsimonious with both its execution and instrumentation state, it remains scalable even under extreme situations such as flashcrowds or DDoS attacks. The operation of CPR can be decomposed into two independent sub-problems: how to detect genuine path failures through *impairment detection*, and how to circumvent an impaired route through *path reselection*. We will tackle each of these problems in turn.

3 Impairment detection

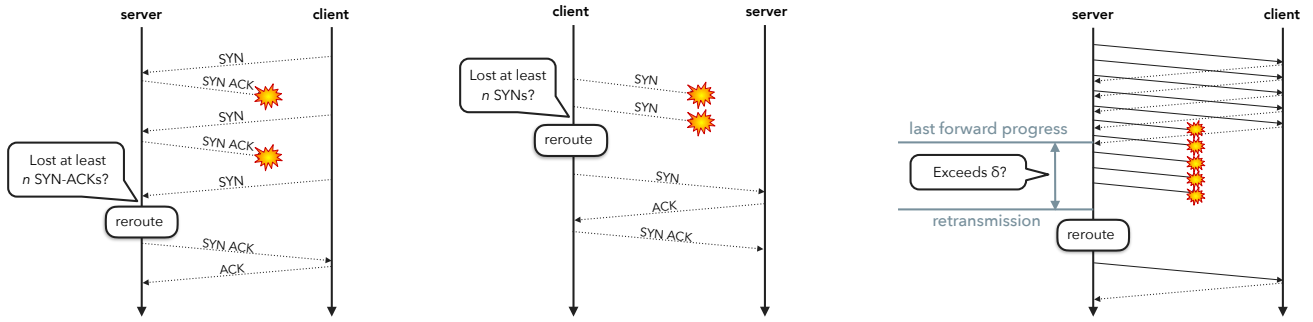
The main task of CPR's *impairment detection* algorithms is to accurately identify path failures based on transport layer performance in a timely manner. A key challenge is to distinguish between spurious packet loss, which should be recoverable with retransmission over the same path, and persistent failures, which are better addressed by selecting a different one. From a transport layer perspective, connections routed over an impaired path experience *stalls*, *i.e.,* fail to make forward progress for some amount of time in spite of retransmissions. CPR works by detecting stalls and using them as a signal to select an alternate path, with the objective of eventually resuming forward progress.

Two different impairment detection mechanisms are necessary, each addressing complementary stages in the TCP connection lifecycle. The first (§3.1) deals with path impairments before connection establishment; the second (§3.2) deals with impairments arising after connections have successfully established.

3.1 Pre-establishment impairments

A TCP connection is initiated by a client transmitting a SYN packet to commence a three-way handshake, to which the server will reply with a SYN-ACK. If the server's outbound path has failed before a connection is established, the SYN-ACK transmitted by the server will be lost. The client then retransmits the SYN after a predefined interval (1s on Linux [3]), to which the server will reply again with a SYN-ACK that will also be lost. This retry behavior continues until the path becomes available again or the client stops retrying (a Linux client retries 6 times by default). In such cases (fig. 1a) CPR will declare a stall upon exceeding the threshold of n presumed lost SYN-ACKs. Upon detecting a stall, route reselection is triggered on every subsequent SYN-ACK retransmission until the connection is successfully established or the client times out.

This detection mechanism is quite coarse because its ability to detect a failure is limited by the frequency of SYN retransmissions at the client. The precision and speed of failure detection could be improved if the server proactively retransmitted SYN-ACKs at a higher frequency without waiting for



(a) *Inbound* pre-establish (b) *Outbound* pre-establish (c) *Established*

Figure 1: Impairment detection and reroute

SYN retransmissions from the client. This would however create an amplification vector that could be exploited by SYN flood attacks and therefore would not be safe to deploy in untrusted environments.

CPR's pre-establishment impairment detection algorithm cannot be used when SYN cookies are enabled, because then the kernel will not keep any state for incoming pre-establish connections. This is a desirable feature. By default, the Linux kernel sends SYN cookies upon listen queue overflow, which is typically triggered by SYN floods. This will result in SYN-ACK stall detection being disabled during an attack, and SYN-ACKs using the preferred path.

Finally, we note that the same mechanism could be applied to connections initiated by edge servers (see fig. 1b), whereby we reroute after a given number of lost SYN packets rather than SYN-ACK packets. This case is less relevant in practice however, since most edge server traffic results from inbound connection requests.

3.2 Post-establishment impairments

For established connections (see fig. 1c) CPR verifies, before a retransmission, whether the connection has failed to make forward progress for a time threshold δ. If so, it declares a stall and selects a new egress path. CPR marks a connection as making forward progress whenever an (S)ACK is received for data that has been sent, but not yet acknowledged. This requires storing a timestamp variable in each TCP socket to keep track of forward progress. The algorithm operates as follows:

- clear the timestamp as long as there are no outbound segments in flight, and set it to the current time when the segment that is at the front of the transmit queue is transmitted *for the first time*;
- update the timestamp to the current time whenever the connection makes forward progress, *i.e.,* receives an (S)ACK that acknowledges data byte ranges previously transmitted but not yet acknowledged;
- clear the timestamp when the last outstanding byte range has been fully acknowledged;
- declare a stall when (re)transmitting a TCP segment if 1) the timestamp is set, and 2) the time elapsed since the timestamp exceeds a threshold δ.

Because this algorithm declares stalls *on retransmission*, connections that become idle whilst using paths that subsequently fail cannot declare a stall until their first retransmission. This behavior minimizes spurious path reselection.

Algorithm parameters. As with n, setting an appropriate value of δ needs to strike an appropriate tradeoff between reactivity and accuracy. The key challenge is to ensure that the threshold works consistently well across connections, regardless of their RTT. Setting δ to a fixed, global value would lead to either spurious stalls for high RTT connections, or sluggish response for low RTT connections. This can be addressed by defining δ in terms of path properties already estimated by TCP. We could, for instance, define δ as a multiple μ_{rto} of the connection retransmission timeout (RTO) T_{rto}, so that $\delta = \mu_{\text{rto}} T_{\text{rto}}$. As usual, $T_{\text{rto}} = \texttt{srtt} + 4 \times \texttt{rttvar}$, where `srtt` is the smoothed round-trip time and `rttvar` is the round-trip time variance [50]. Unfortunately, this solution on its own could be problematic for connections with very low `srtt` and `rttvar`, because, given a low value of δ, temporary router queue build-ups and subsequent increased latency may be misidentified as stalls and trigger spurious reroutes. We guard against this issue by defining δ_{min}, a lower bound for δ, and setting $\delta = \max(\delta_{\text{min}}, \mu_{\text{rto}} T_{\text{rto}})$.

Using a small value for μ_{rto} (*e.g.,* $\mu_{\text{rto}} = 1$) may spuriously trigger reroutes during the slow start phase of the connection. For paths with moderate background packet loss, RTO expiration is more likely to happen when there are few TCP segments in flight, *e.g.,* during slow start: once the connection has filled its *bandwidth-delay product*, a constant stream of incoming ACKs makes the triggering of RTOs less likely.

Rate limiting reroutes. Re-routing onto a new path does not necessarily result in recovery: the new egress path could share a failure with the original one, or the impairment responsible for the stalling could be on the inbound path. When this occurs, CPR will simply continue to probe paths until forward progress is made. If the reroute threshold δ of the connection is low, the connection may end up being rerouted multiple times in rapid succession. This could result in CPR not being able to gather enough data about the state of a new

path to make an accurate decision about its suitability before trying yet another path. We addressed this by implementing a *rate limiting* over periods of length `wait`, so that connections will not be re-routed more than once in every period.

Triggering stall detection logic. Since CPR performs stall detection on retransmission with a timeout expressed as a multiple of the RTO, it is natural to ask whether stall detection logic could be co-located with the RTO triggering logic of the TCP state machine. The answer is negative, because not all retransmissions relevant to connection forward progress are triggered by RTO expiration. Consider the case where the outbound path used by a connection has failed and both the local and the remote ends have outstanding (unacknowledged) data. It is possible for retransmissions by the local end to keep being triggered by the reception of retransmissions from the remote end, before a local RTO can elapse. For this reason, stall detection logic *reuses* the RTO value, but is evaluated *independently* of RTO logic at relevant points in the TCP state machine (*e.g.,* when sending retransmissions).

4 Path reselection

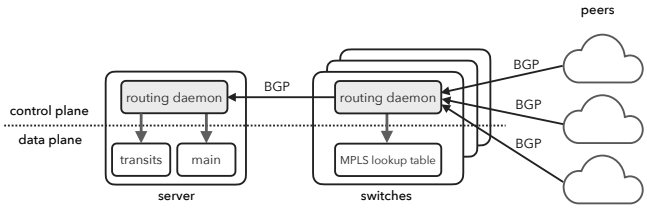


Figure 2: Routing architecture

CPR leverages a routing architecture similar to Espresso [64] and Silverton [14, 57] which push visibility of all available routes down to edge hosts. In this architecture (depicted in fig. 2) each host is connected to a number of switches, which are in turn connected to a number of upstream providers. These include both transit providers and settlement-free peers, connected directly through Private Network Interconnects (PNI) or Internet Exchange Points (IXP).

Each switch performs two tasks. First, an MPLS label is configured for each upstream provider, and a corresponding nexthop entry is inserted into the local routing table. Second, BGP route updates received from upstream providers are tagged with the associated MPLS label, and forwarded to routing daemons on the host. The routing daemon on end hosts populates two routing tables:

- The **main** table contains all policy-preferred routes among those learned from all peers, and is used for routing traffic under *normal* circumstances *i.e.,* when path reselection has not been requested. It contains routes preferred under performance, capacity and cost constraints.
- The **transits** table is populated with all *default routes* (*i.e.,* 0.0.0.0/0 or ::/0) learned from upstream providers. Since settlement-free peers do not provide universal reachability (*i.e.,* export a full routing table), only default routes provided by transit providers are included.

Given the routing architecture in fig. 2, and having access to the *main* and *transits* table, the next objective of CPR is to provide a mechanism to allow the stall detection algorithms in §3 to select a new path for a stalled connection. We achieve this by associating a reroute counter r with each connection, and incrementing it every time a stall is declared for that connection. This counter is stored in the 4 most significant bits of the firewall mark (*fwmark*) of a connection, a 32 bit value that can be used to tag packets traversing the Linux network stack and make routing decisions about them. To make rerouting based on r possible, we made two relevant changes. First, we changed the Equal Cost Multipath (ECMP) hashing function used by the Linux kernel. The standard Linux implementation of ECMP selects a nexthop by hashing the connection *5-tuple* (*i.e.,* source and destination IP addresses, source and destination ports and protocol number). *CPR includes the value of the reroute counter r into the hash computation*. Second, we configure an *ip rule* to ensure that, for any connection for which $r > 0$, a next hop is looked up from the *transits* table rather than from the *main* table. Hence, *we use r as a flag that triggers CPR-specific routing for connections that have suffered stalls*. The combined effect of these two changes is that simply incrementing the reroute counter will force a new route lookup.

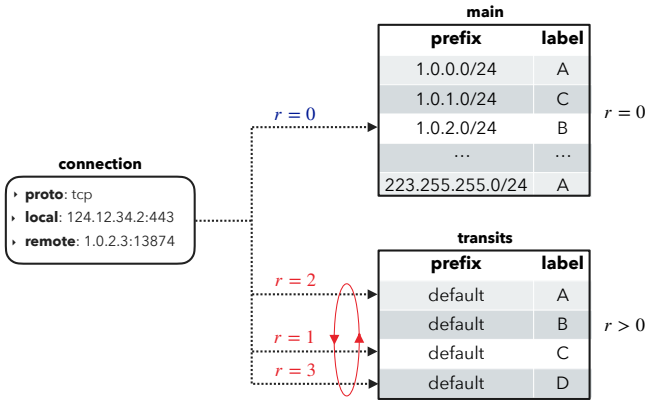


Figure 3: Path reselection upon stall detection

As an illustration, consider a hypothetical connection that has not yet experienced a stall, as shown in fig. 3. At this point in its lifetime, $r = 0$, route lookups are performed using the *main* table, and the BGP-defined egress path is used. When the stall detection algorithm (§3.2) declares a stall, it increments r. From that point on, since $r > 0$, route lookups are performed using the *transits* table, and the next hop for IP packets forming this connection is pseudorandomly selected among all the nexthops of the default route present in the transits table according to ECMP(5-tuple, r). Since each increment of r forces a new route lookup, as r increases the stalled connection will follow a unique, pseudorandom sequence of egress paths which will depend on both its 5-tuple

and the ECMP hash used. The aggregate effect of this process is that rerouted connections are homogeneously "load balanced" among all available egress paths.

We now show that CPR should be able to resolve recoverable stalls with relatively few retries. If n_p is the number of egress paths available, of which n_s lead to *stalled* connections, each path reselection is a Bernoulli trial with a success probability $p = \frac{n_p - n_s}{n_p}$. The number of re-routes k required until recovery will be geometrically distributed [2] so that $P(X \leq k) = 1-(1-p)^k$. Hence, the expected maximum number of re-routes that will be required to find a good path with a given probability β is $k^* = \frac{\log(1-\beta)}{\log(1-p)}$. Even a conservative[2] $p = 50\%$ and $\beta = 95\%$ results in only $k^* \approx 4$ re-routes.

We note that, although fig. 2 states that the *main* table should contain a full routing table, CPR does not *require* this to be the case. As noted above, path reselection only relies on the *transits* table. Further, although our implementation uses MPLS to steer specific flows towards a given provider, this can also be done using GRE tunneling [64] or DSCP marking [52]. Routes can be pushed down to the host using BGP add-path [62] or proprietary mechanisms. Our architecture is just one of many that could support CPR deployment; the basic primitives of CPR are applicable to many scenarios.

5 Evaluation

This section evaluates the performance of CPR in a large edge cloud production deployment with daily traffic peaks on the order of tens of Tbps. All results were collected through passive measurements of production traffic.

5.1 Parameter tuning

Tuning CPR involves resolving a tradeoff: whereas unnecessarily rerouting connections could place them on paths with potentially lower performance and higher cost, failing to react to a recoverable path impairment increases its potential to harm client connections. This section discusses how we tuned the CPR parameters (§3) to resolve this tradeoff between accuracy and reactivity.

Tuning pre-establishment impairment detection. The only parameter involved in detecting impairments prior to connection establishment is n, the number of presumed lost SYN-ACKs after which a reroute is executed (§3.1). Hence, at this stage tuning involves determining the value of n after which timely connection establishment becomes unlikely without CPR intervention.

We proceeded by instrumenting servers in three distinct geographical regions (North America, Europe and Asia) with CPR disabled. We then measured how many SYN retransmissions occurred for all connections that were eventually established[3]. As shown in table 1, only ~0.63% to ~0.7% of connections experience impairments before establishing, depending on the region. This means that more than ~99.3% of connections establish without any retransmissions. Further retransmissions help connection establishment, but with noticeable diminishing returns. For example, after two consecutive retransmissions without a reroute, the probability of a connection being successfully established was between ~0.05% to ~0.12%. Based on these findings, we set $n = 2$ in our production configuration.

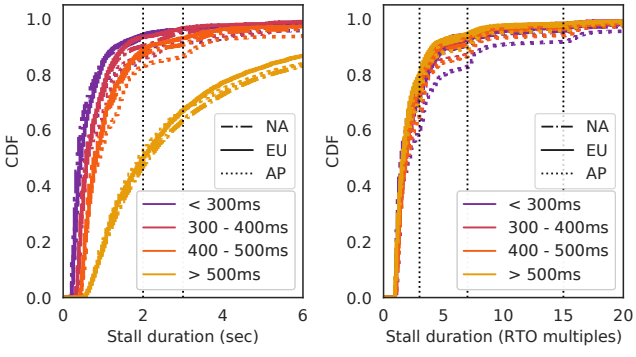
n	APAC	EU	NA
0	.63	.64	.70
1	.13	.17	.28
2	.05	.09	.12
3	.03	.06	.08
4	.01	.05	.05
5	.01	.03	.03

Table 1: Proportion of connections not establishing after n consecutive SYNACK losses (%)

Stall duration lower bound	APAC	EU	NA
RTO	1.79	.57	2.18
2s	.34	.36	.10
3s	.23	.24	.06

Table 2: Proportion of connections experiencing at least one stall during their lifetime, as a function of the stall duration (%)

Tuning post-establishment impairment detection. We addressed the tuning of δ_{min} and μ_{rto} (§3.2) in two steps. First, we followed a similar measurement methodology as that used to tune n, this time focusing on connections experiencing at least one RTO expiration during their lifetime. Our results, reported in table 2, provide evidence of significant regional variability. Whereas ~0.1% of connections in North America experience stalls lasting for 2 seconds or more, this increases to ~0.35% for connections in Europe or Asia-Pacific[4]. However, these results also show that the probability that a connection recovers after experiencing a stall for 2 or 3 seconds was very low irrespective of geographical region.



(a) (b)

Figure 4: Duration of TCP stalls by region and RTO

[2] See §5.3 and appendix A.1 for further details.

[3] This was done during a period free of obvious biases such as outages, high-load client events, DDoS attacks, etc.

[4] These values constitute a lower bound; during the active phase of an impairment event the proportion of affected connections can rise by an order of magnitude or more (see §5.2).

The next step was to investigate the time required for a connection to resume forward progress after an RTO (fig. 4). We begin by noting the two vertical lines in fig. 4a: these correspond to the two candidates found in the previous study. For connections with RTO $\leq$ 500ms, most of the probability mass in the dataset lies to the left of the first line, providing evidence that for these connections the length of a stall is largely independent of RTO and $\delta_{\text{min}} = 2$s is sufficient to ensure adequate impairment detection.

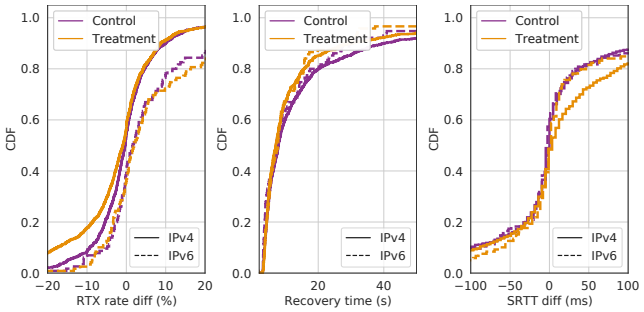
To understand how best to handle connections with RTO > 500ms, we first note the probability mass "bumps" in fig. 4b. These intervals of 3, 7 and 15 RTOs stem from TCP retransmission behavior under exponential backoff. Since the majority of the probability mass lies to the left of the first line, a large proportion of connections will recover on their own before $\mu_{\text{rto}} = 3$, irrespective of their RTO. This addresses the high RTO connections that were not already covered by $\delta_{\text{min}} = 2$s, and provides a rationale to set $\mu_{\text{rto}} = 3$.

Finally, we set `wait` (§3.2) by observing day-to-day operation of the system in production. We select `wait` = 1s sec on the basis of keeping the volume of steady-state rerouted traffic to a sufficiently low level.

5.2 Evaluating benefit and non-harm

Methodology. To ascertain the impact that CPR has on ongoing connections we follow an *experimental* approach, in which we (pseudo)randomly label some connections as part of a *treatment* group, and the remainder as part of a *control* group for which path reselection logic is disabled. While the state and output of the impairment detection algorithms are maintained for both groups, network-layer path changes are triggered only for treatment connections. This setup allows us to explore the degree to which the benefits of CPR during path outages outweigh its potential costs when no impairments are present. We begin by exploring whether rerouting a stalled connection could make its performance significantly worse than doing nothing. To the extent that the answer to this question is negative, CPR will be innocuous when triggering due to stalls not associated with path impairments, and hence, non-recoverable by rerouting. We then move on to analyze the benefits that CPR provides during path impairment episodes.

Evaluating non-harm in the steady state. For both treatment and control we quantify the *reroute effect* on connection properties such as RTT or retransmission rate. Since connection properties need time to settle to their new values after a reroute in order to be meaningful, we focus on long-lived connections. We define the *onset* of a stall as the TCP sending event immediately prior to an RTO, and the *full resolution* as the first sending event after forward progress is restored and 30[5] segments have been sent. We define the reroute effect on a connection property as the difference between its value at *onset* and at *full resolution*.



(a) (b) (c)

Figure 5: Steady-state reroute effect experiment results

The CDF of the reroute effect on the TCP retransmission rate[6] is presented in fig. 5a. Whereas it seems to be negligible for IPv6, for IPv4 the treatment connections have a greater proportion of their probability mass on the negative effect sizes, implying that rerouted IPv4 connections tend to have lower retransmission rates after the reroute than before it. This points towards CPR having *some* small benefit during the measurement period. Otherwise, the curves are very close to one another, providing evidence that CPR is not introducing significant retransmissions during steady state.

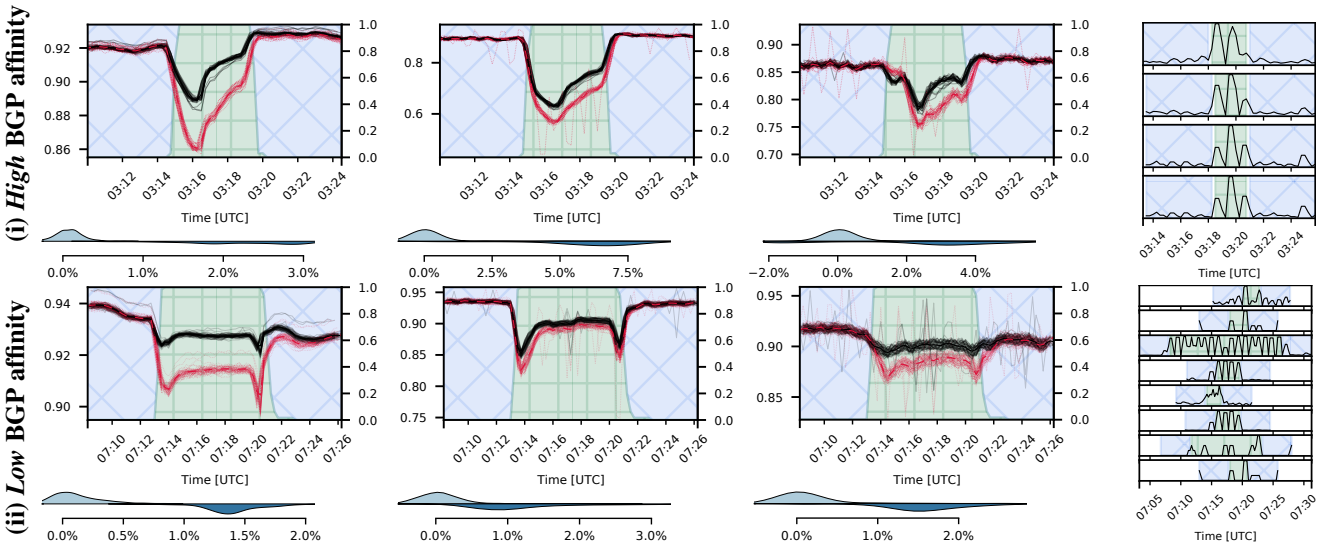
We can also use the results of the previously described reroute effects experiment to understand the effect of CPR on stall recovery speed. From fig. 5b we can see that treatment connections tend to have shorter resolution times compared to control connections, both for IPv4 and IPv6. The effect during steady state is small, as expected: ~80% of control connections recover within 20s of a reroute, compared to ~85% of treatment connections.

Finally, fig. 5c shows the effect of reroutes on `srtt` (§3.2). Again, a seemingly negligible effect for IPv6 is accompanied by a clear effect for IPv4, this time demonstrating higher `srtt` values after the reroute (usually by less than ~30 ms, but sometimes more than ~100 ms). This performance penalty is expected since our BGP traffic engineering policy optimizes for latency, and is more finely tuned in IPv4 than in IPv6 due to operational maturity. Since CPR explicitly reroutes away from paths selected by this policy, we are more likely to experience an increase in RTT than not. In this light, our configuration of CPR is an expression of the extent we are willing to subvert local routing policy in an attempt to recover from failure. Since this is a matter of policy, we observe there is no single correct configuration, but a range of potentially acceptable outcomes.

Evaluating benefit during stall events. During path impairments, the most common outcome for control connections is *failure* (rather than *e.g.,* increased RTT or retransmission

[5] This number was arbitrarily selected to provide enough samples for TCP connection properties to stabilize.

[6] Every connection in this dataset experienced a stall during the measurement period. Hence, retransmission rates are expected to be higher than the blended averages typically reported.



(a) $\pi_\bullet(\texttt{HEALTHY})$ (b) $p^o_\bullet(\texttt{HEALTHY}|\texttt{SYNRCVD})$ (c) $p^i_\bullet(\texttt{CLOSE}|\texttt{HEALTHY})$ (d) BGP events

Figure 6: CPR operation during two stall events (i and ii). The timeseries plots in figs. 6a to 6c show the *control* • (dashed) and *treatment* • (solid) group values (**left y-axis**). Each one of the *thin* lines corresponds to a single host; *thick* lines track the average for all hosts in a POP. In the same plot we also show the proportion of hosts in a POP for which the timestamp has been classified as part of an *active event* ⊞ or its *context* ⊠ (**right y-axis**). Beneath each timeseries plot we show a violin plot for the PDF of $\Delta\pi$ (fig. 6a) or $\Delta p^\bullet$ (figs. 6b and 6c), the difference between treatment and control values, for both *context* • (top) and *event* • (bottom) periods. Each subplot in fig. 6d shows BGP update/withdrawal rates for peering sessions which triggered anomaly detection in proximity to the active event span. Since only relative changes are relevant, y-axis tickmarks have been removed.

rates). Hence, to accurately assess the benefits of CPR we must look beyond the reroute effect measures presented above. Broadly, we resort to anomaly detection on the performance differences between treatment and control groups in order to identify *stall events*. The benefit of CPR for a given stall event can then be measured by comparing treatment and control performance differences during the event, with the corresponding performance differences during the immediately adjacent timespans. We refer to the union of timespans immediately preceding and following a stall event as its *context*.

In order to identify stall events, we instrumented the kernel to export additional metrics. First, we store per-connection information on the operation of the recovery mechanism as part of the socket metadata; this includes the reroute counter r, last forward progress timestamp, etc. Second, we maintain aggregate counters in the kernel which track state transitions as each connection traverses both TCP and CPR state machines. Each edge server aggregates separate counters for treatment and control group connections, allowing us to estimate the efficacy of CPR on each host according to multiple performance measures[7], including:

- $\pi_\bullet(\texttt{HEALTHY})$, the proportion of TCP connections in the HEALTHY state;
- $p^o_\bullet(\texttt{HEALTHY}|\texttt{SYNRCVD})$, the proportion of TCP connections that transition out from the SYNRCVD state towards the HEALTHY state, per unit time. This gives an indication of the instantaneous probability of connections connecting successfully; and
- $p^i_\bullet(\texttt{CLOSE}|\texttt{HEALTHY})$, the proportion of TCP connections that transition into the CLOSE state from the HEALTHY state, per unit time. This gives an indication of the instantaneous probability of connections closing while healthy (rather than stalled).

We present two CPR stall events in fig. 6, embedded in their *context* so that their impact is clearly visible. Each event is presented along its *BGP affinity score*, a synthetic measure of the degree to which observed routing plane events are correlated with transport layer anomalies (appendix C). Our examples were selected to juxtapose the case where there is a *high* BGP affinity score (i), and therefore an association between CPR and BGP behavior, and a *low* BGP affinity score (ii). In both cases we can observe that immediately after the start of the stall event (at the left context/event boundary) CPR treatment connections start experiencing better performance, as inferred from a higher $\pi_\bullet(\texttt{HEALTHY})$ (fig. 6a). Treatment connections also have better chances of establishing (fig. 6b), evidencing connection setup distress for control connections, and exhibit a greater chance of closing while HEALTHY, rather than when STALLED (fig. 6c). When taken together, these facts point towards a small but significant proportion of affected connections, both *established* and *pre-establish*, clearly benefiting from CPR, irrespective of the BGP affinity of the event. When a clear association with BGP is present, such as with fig. 6d (i),

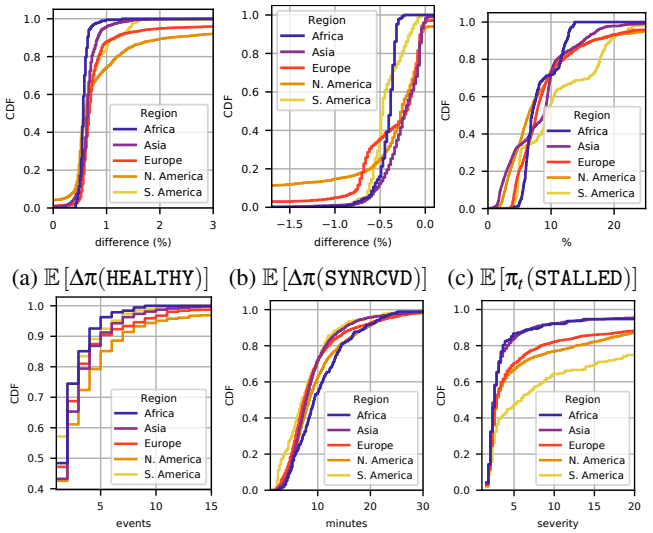
[7] A more detailed overview of stall event detection is included in appendices B and C.

the benefits provided by CPR are materialized significantly before the associated BGP event is resolved. Conversely, fig. 6d (ii) demonstrates that CPR can recover just as effectively in cases for which BGP provides no remediation.

Each violin plot beneath a CPR time series event subplot in figs. 6a to 6c shows the benefit that CPR provided to connections assigned to the treatment group during the event. For state occupancies we rely on $\Delta\pi = \pi_t - \pi_c$, the difference between treatment and control values (similarly, for state transitions $\Delta p^\bullet = p_t^\bullet - p_c^\bullet$). For instance, in fig. 6a we can see that although the distribution of $\Delta\pi(\texttt{HEALTHY})$ was centered at zero outside the event period, it moved to the right during the active phase of the event, irrespective of BGP affinity. For (i) we see that CPR allowed an additional ~3% of connections (over the entire POP and to all destinations) to remain in the HEALTHY state compared with the control group; for (ii) this is reduced to ~1.4%. On the other hand, from $\Delta p^o(\texttt{HEALTHY}|\texttt{SYNRCVD})$ (fig. 6b) we see that during (i) CPR allowed an additional ~7% of connections in the POP moving out from a SYNRCVD state to enter the HEALTHY state; during (ii) this is reduced to ~1%.

Evaluating benefit globally. Having explained the typical characteristics of stall events using individual examples, we now focus on 1) measuring CPR effectiveness at a global scale, and b) identifying the circumstances under which CPR is most effective. To this end we perform statistical aggregation on data collected from ~80 POPs over ~12 months. As before, we focus on *recovered events*, defined as stall events which show evidence of improvement in performance measures (*e.g.,* those presented in fig. 6). We do this for two reasons. First, whereas improved treatment group performance unequivocally points towards TCP stalls that can be resolved by path reselection, anomalies where CPR provides no benefit are uninformative: there are many possible causes of TCP stalls unrelated to resolvable path impairments (*e.g.,* wireless access roaming or dis-association) for which re-routing cannot be expected to help. The second reason is implementation-related. Since performance data such as presented in fig. 6d is only kept aggregated at host level (rather than at prefix or connection granularity) it is difficult to directly associate a CPR event with a set of destination prefixes/ASes, and hence, to a BGP-based ground truth. For recovered events this is not an issue: CPR itself provides direct confirmatory evidence of path impairment mitigation[8].

We will denote the daily per-POP average CPR state occupancy during the active phase of stall events (rather that their context) as $\mathbb{E}[\Delta\pi]$. This measure can be used to directly understand which geographies benefit the most from the deployment of CPR, as shown in figs. 7a and 7b. First, in fig. 7a we can see that most of the probability mass for $\mathbb{E}[\Delta\pi(\texttt{HEALTHY})]$ lies on the *positive* semi-axis regardless of geography, implying that $\pi_t \geq \pi_c$ and hence that the proportion of HEALTHY connections was higher for the treatment than the control group (median $\in [0.57\%, 0.68\%]$). Conversely, fig. 7b shows that most of the probability mass for $\mathbb{E}[\Delta\pi(\texttt{SYNRCVD})]$ lies on the *negative* semi-axis (median $\in [-0.46\%, -0.19\%]$), so that $\pi_t \leq \pi_c$. This means that CPR allows connections to both 1) connect faster and 2) spend less time stalled. We note that whereas $\pi_t(\texttt{HEALTHY})$ benefits are greatest in Europe (median = 0.68%) and Asia (median = 0.65%), $\pi_t(\texttt{SYNRCVD})$ benefit is particularly strong in South America (median = −0.46%) and Africa (median = −0.38%). However, as shown in fig. 7c, these benefits are accompanied by a cost: every day more than ~10% of connections on up to ~30% of PoPs experience stall events. Although the immense majority of these stalls are associated with abandoned connections and hence carry negligible traffic, we note that non-recoverable stalls induce a sustained level of background reroutes.



(a) $\mathbb{E}[\Delta\pi(\texttt{HEALTHY})]$ (b) $\mathbb{E}[\Delta\pi(\texttt{SYNRCVD})]$ (c) $\mathbb{E}[\pi_t(\texttt{STALLED})]$

(d) Count (e) Duration (f) Severity Index

Figure 7: Per-day, per-POP recovered events

In contrast to figs. 7a to 7c, figs. 7d and 7e show weaker region dependence. Although fig. 7d provides some evidence that recovered stall events occur roughly proportionally to both 1) the aggregate traffic volume and 2) the interconnection density of a region, the difference between regions is small. Similarly, fig. 7e shows that most recovered events last under 20 minutes, irrespective of region (median $\in [7.1, 9.4]$). However, we note that this distribution can be heavy tailed: we have observed uncommon, longer events that span multiple hours. These can have "macroscopic" effects, triggering operational responses and re-routing over 20% of PoP traffic.

We finally turn our attention to the severity of recovered events. We define a heuristic *severity index* based on the normalized performance difference between treatment and control connections during the event and its associated context. The higher the number of standard deviations between

[8]Note however that, since our CPR deployment could *in principle* fail to recover from some stall events, our reported prevalence findings suffer from survivorship bias [63] and must be interpreted as lower bounds.

treatment and control variables during the event itself, compared to before and after it, the higher the severity for the event (see appendix C). Perhaps unsurprisingly, fig. 7f shows that CPR routinely recovers from higher severity events in densely connected regions. However, the distinction between the North and South America curves in figs. 7d and 7f is instructive. Whereas in fig. 7d the former is always beneath the latter, in fig. 7f the opposite occurs. This shows that CPR helps recover from more events in regions with nontrivial BGP routing where complex business agreements and undersea cable topology can make BGP-only recovery more challenging.

As a complement to the per-POP statistics presented in fig. 7, we report that globally CPR mitigates on average ~120 impairment events every day with a median duration of ~8 minutes. Informally, this amounts to over ~16 hours per day. While this improvement may appear quantitatively modest, in practice these performance degradations bear a high cost in the absence of mitigation. Many of them exhibit no proximal cause, making them notoriously hard to debug. Even when an underlying cause is understood, many performance problems are short-lived and will be resolved by the time engineering or customer support resources have been mobilized to address them. In opportunistically mitigating either type of deterioration, CPR has an outsized impact in improving the reliability of services provided by edge networks, at virtually no cost.

5.3 Path coverage

Given that CPR deflects traffic among upstream providers when mitigating path impairments, it is informative to estimate the degree to which reroutes are likely to result in disjoint data plane paths. To this end, we analyze all routes learned from every provider in a given POP. Since all transits offer full routing tables, every destination prefix will have a set of available routes. In addition to these routes, we may have additional peering routes learnt from PNIs or IXPs. One of these paths (learnt from either peering or transits) will be *selected* as an outcome of BGP policy; the remaining routes (learnt over transits) represent valid *alternate* paths.

We proceed by defining each AS in the selected path to be *node-protected* if there is an alternate path for the same destination that does not include it. Then, we define *AS-node diversity* as the proportion of node-protected ASes in the selected path. Similarly, we can define an AS pair in the selected path to be *link-protected* if there is an alternate path that does not include the same AS pair in the same order, which leads to the entirely analogous definition of *Inter-AS-path diversity* as the proportion of link-protected AS pairs in the selected path. Since CPR will eventually explore every available path to every destination, these two measures can be used as proxies for the probability of recovery given a failure event involving any given destination prefix[9].

An analysis of ~900k globally distributed routes revealed traffic-volume-weighted AS-node diversities of 90.8% and 86% for IPv4 and IPv6 respectively; these rise to 93.7% and 90.5% when considering Inter-AS-path diversity. The significant path diversity available to CPR evidenced by these numbers is a consequence of a strict requirement for edge networks: POPs must be interconnected to multiple transits in order to survive outages of individual providers or local exchange points. Since most client traffic is preferentially exchanged over peering, the addition of a transit provider has a limited impact in improving latency. Instead, multihoming is primarily a form of insurance. In this light, CPR is of great interest to edge network operators in that it extracts greater value from what is otherwise a necessary cost.

6 Operational considerations

CPR has been in use for over two years. This section revisits some of our original assumptions in the light of our accrued experience, as well as highlight some of the tweaks that were required along the way.

Per-route overrides. While it is beneficial to enable CPR by default on edge hosts, and the parameters derived in §5.1 are adequate for a wide range of conditions, there are cases where it is desirable to override configuration on a per-route basis. For example, rerouting intra-POP traffic onto transits is ineffective as a mitigation strategy, and costly due to the large volume of traffic exchanged between hosts within a POP.

Linux already allows per-route configuration of *features*, such as ECN or SACK. We expanded this method to allow the configuration of CPR on a per-route basis, and extended our edge host routing daemon to be able to inject routes into the *main* routing table accordingly. Over time, this has allowed us to selectively disable or experiment with CPR on a per-route basis as an integral part of our BGP policies. This evolution was gradual: what started exclusively as a server-side, transport layer extension has been progressively incorporated operationally as an extension of our routing infrastructure.

Middleboxes. Rerouting ongoing connections could in theory adversely interact with stateful middleboxes, which would be presented with packet streams for which the TCP three-way handshake happened in a different path (and hence was not observed). The expected outcome of this potential problem, increased TCP RST rates or silently dropped packets for treatment connections compared with control (see §5.2), has not been observed in practice after multiple years of sustained, worldwide, multi-Tbps usage. This is not unexpected, since

[9] Two practical qualifications are in order. First, since a packet can traverse different router-level paths within the same AS between given ingress and egress pairs, and multiple peering points between two adjacent ASes in an AS path, both quantities are lower bound estimates of actual dataplane path diversity. Second, under these definitions a number of high-volume destination prefixes advertised by directly connected peers will be counted as presenting zero diversity whilst in reality presenting negligible impairment risk as their path reliability is very high. These have been removed from the analysis in §5.3, but are reported in their entirety in appendix A.1.

CPR does not modify the TCP header in any way. From the perspective of intermediate devices, CPR reroutes are indistinguishable from other reroutes events, such as those triggered by BGP or ECMP rehashes.

Prefix aggregation. It is natural to consider extending CPR by aggregating per-connection information and using it to influence routing. If many connections in a given prefix stall, only to recover successfully after a reroute, we could avoid future stalls by overriding the route for that prefix. Evaluating this approach is outside the scope of this paper, but the implementation is far from straightforward once we consider its risks. Firstly, as evidenced by §§5.1 and 5.2, not all transport-layer impairments will be correctable by path reselection. This means that there is always the danger of overriding routing and traffic engineering policy in an attempt to resolve a suspected problem that cannot be fixed that way. This risk is severely amplified by performing routing interventions on entire prefixes. A related threat is that, in addition to natural variability, stall signals are also susceptible to adversarial manipulation. Using stall recovery information to route entire prefixes would amplify the impact of such an attack, potentially burdening the edge network with higher transit costs or end-users with higher latency. Finally, there is no *a priori* reason to assume that the same routing intervention will have the same effect on every subprefix covered by an Internet-advertised prefix. Whereas this tradeoff is irreconcilable at the prefix level, it is trivially accommodated at the connection level.

Avoiding peering offload. As noted in §4, whereas routes in the *main* table are learned from *both* settlement-free peers and transits, routes in the *transits* table are *only* learned from transits. Therefore, CPR path reselection can re-route connections away from PNIs and IXPs and onto transits, which would typically result in greater costs and potentially less direct paths to destinations. During our design phase we accepted this limitation because the alternative is for connections to remain stalled; an opportunity to recover then makes the cost benefit positive. Although the nature of colocation facilities makes it relatively rare for a single POP to be connected to multiple IXPs, it would be beneficial to be able to failover from PNIs to IXPs. In our current architecture, this capability can only be provided at the cost greater routing complexity (defining the set of failover routes on a per-destination basis).

Path reselection. While one could envision more complex path selection algorithms than those presented in §3, in our experience the additional complexity does not translate to significant benefits. The simplicity of CPR is a core feature, providing quick recovery and effective load balancing without burdening operators with configuration overhead.

Userspace support. CPR need not be limited to the kernel, it can be leveraged or selectively disabled by any userspace application. To support this, we reserved bits in *fwmark*, which can be set via a `setsockopt` system call, to communicate application intent to the Linux kernel networking stack.

We reserve one bit in *fwmark* to signal that the connection for the given socket must not be rerouted. This allows CPR to be disabled on a per-connection basis for measurement and debugging purposes. For example, it may be necessary for a connection to be sent over an application-defined path to test path performance or to debug networking issues.

In order to support UDP based transport protocols, we allow userspace applications to directly manipulate the value of *r*. Surfacing control over the reroute counter through the *fwmark* effectively pushes route reselection out of the kernel.

QUIC. One of the primary drivers for userspace support of CPR has been the ongoing deployment of QUIC [38]. QUIC supports connection migration [31], and over time we expect this to be the primary mechanism for path failover. As of today, however, QUIC connection migration may be disabled by either peer, and can only be used after connection establishment. Furthermore, migration must be initiated by the receiver, who must send a probe packet from a new local address. CPR on the other hand covers connection establishment, and can be triggered unilaterally on the sender side, where stalls are more quickly detected. Since there is nothing intrinsic to CPR that makes it fundamentally incompatible with connection migration, we view both methods as complementary and are actively experimenting with CPR within QUIC. While validating the use of CPR within QUIC is the subject of future work, the implementation itself is straightforward and bears testament to the overall elegance and simplicity of CPR.

7 Related work

Although multiple solutions in the literature address the detection and mitigation of path impairments, none provides feature parity with CPR. To the best of our knowledge, the design of CPR is unique in its simplicity, which makes it extremely easy to deploy. It is the only solution providing path failover at a connection granularity, which does not require switches with programmable dataplanes or support from the client endpoint, and which makes it possible to identify and mitigate performance degradation even if a connection is not currently established.

Our work is most closely related in motivation and approach to INFLEX [58] and Blink [28]. Both use transport-layer triggers to reroute traffic and, unlike CPR, rely on programmable data plane switches. While INFLEX reroutes stalled flows on a per-connection basis, it does so by installing ephemeral route entries onto a switch. It can therefore not support failover for all connections at the scale edge networks operate. Blink on the other hand relies on the ability to monitor sequence numbers of ongoing TCP flows. This poses not only scale concerns, which Blink addresses by limiting itself to only monitoring high volume prefixes, but also cannot work on encrypted transport protocols such as QUIC. Finally, by detecting failures and re-routing on a route/prefix granularity, Blink suffers from the same pitfalls of traditional

Table 3: Comparison of path failover mechanisms

Feature	BGP [48]	MPTCP [24]	SWIFT [29]	INFLEX [58]	Blink [28]	CPR
Pre-establish connection support	✓	✗	✗	✓	partial	✓
Partial failures support	✗	✓	✗	✓	partial	✓
Granular failover	✗	✓	✗	✓	✗	✓
Support all connections and prefixes	✓	✓	✓	✗	✗	✓
No client support required	✓	✗	✓	✓	✓	✓
No switch support required	✓	✓	✗	✗	✗	✓
O(`seconds`) convergence	✗	✓	✗	✓	✓	✓
Production validation	✓	✓	✗	✗	✗	✓

routing protocols. Smaller failures can go undetected, and reroute decisions can result in substantial changes in traffic allocation, exacerbating the risk of cascading failures due to downstream congestion. CPR on the other hand can be enabled for all flows, and can detect outages and impairments on a per-connection basis. Re-routing is also performed on a per-flow basis, and as a result traffic can be offloaded and distributed in a more granular fashion. CPR can also afford more sophisticated failure detection implementations, for example based on RTT variation, that passive monitoring in switch dataplanes cannot support.

SWIFT [29] infers the extent of failures through the analysis of BGP updates. While it reduces the time to route convergence, its reliance on BGP route updates makes it oblivious to a wide array of failure scenarios. By the authors' own admission, it is also unable to improve the time taken to receive BGP updates, which itself is on the order of multiple seconds [28]. CPR bypasses routing messages altogether, and can more accurately detect more types of failure in less time by simply piggy-backing on existing transport mechanisms.

Although a number of egress routing control systems aim to mitigate some of the same issues as CPR (*e.g.,* Espresso [64] or Edge Fabric [52]), CPR again differentiates itself by operating on individual connections rather than prefixes. This makes it complementary to such systems: it is possible to use them to apply intelligent egress traffic engineering, whilst still relying on transport extensions like CPR to mitigate transient, sub-prefix path impairments. In fact, since CPR depends on specific routing primitives for its operation (*e.g.,* those provided by Silverton [14, 57]), CPR *requires* a subset of the functionality provided by these systems.

FlowBender [34], similarly to CPR, uses transport-layer metrics to trigger path reselection decisions, but for the purpose of intra-datacenter load balancing. However, in contrast to CPR, it 1) requires Explicit Congestion Notification (ECN) which is not safe to use with anycast traffic [49, 59]; and 2) requires control of the ECMP hashing configuration of routers along the path to implement its reroute mechanism. These two issues make it unsuitable for providing resilience against Internet path failures. Similarly, although SD-WAN solutions (*e.g.,* [22, 65]) can achieve for overlay networks results similar to CPR, they are neither unilaterally deployable nor a good fit to the business model of an edge cloud network.

Multipath TCP (MPTCP) [24] makes it possible to establish single connections over multiple paths and load balance traffic automatically to the best performing path. Inheriting from previous multipath-aware transport such as shim6 [44] and SCTP [55], MPTCP has slowly gained traction in niches such connection handoff [10]. In the same way the usage of CPR is not at odds with QUIC, we do not view CPR as inherently incompatible with MPTCP. CPR works well precisely where MPTCP often falls short, for example by improving reliability for short lived flows which dominate web traffic, protecting connection establishment, or exploiting path diversity without requiring clients to explicitly identify distinct paths (*e.g.,* with distinct addresses). In contrast with CPR, MPTCP provides no value when interacting with legacy TCP clients, since it is not unilaterally deployable. By tying itself to a legacy wire format, MPTCP is also less forward looking than QUIC. Given the significant economic investment required to deploy and support a new transport protocol, it is unsurprising that edge networks have collectively focused on the latter. Over time we expect QUIC to fully assimilate the explicit multipath capabilities of MPTCP, while still falling back to implicit multipath mechanisms such as CPR.

8 Conclusion

This paper presented Connection Path Reselection (CPR), a novel system that improves the reliability of *edge networks* [52, 64] by inferring the status of a path by monitoring transport-layer forward progress, and rerouting stalled flows onto healthy paths.

CPR is unabashedly simple, and follows in a long tradition of incremental improvements to the Internet which push the boundaries of best effort delivery. While it cannot protect flows against all failures, the cases in which it is effective come virtually for free. Our implementation is trivial to configure, effortless to operate, requires no additional hardware and exploits path diversity already available to edge networks. By operating at the transport layer, CPR provides faster recovery than is attainable by routing alone, as well as detecting a wider array of potential failures on a per-connection basis without incurring additional state.

Importantly, this paper is not a proposal - CPR has been deployed in production at a large scale edge network for over two years. We evaluate our design within this context, and document our assumptions for the benefit of a wider community. To the extent of our knowledge, CPR both complements existing routing and traffic engineering mechanisms, and can coexist with multipath enhancements to transport protocols in the future. As a bridge between the two, CPR is a step towards the collective goal of ensuring that only a complete partition is capable of stalling an Internet transport connection.

Acknowledgements

We would like to thank our shepherd Italo Cunha and the anonymous reviewers for their feedback. We are also very grateful to the engineers at Fastly who have contributed to the deployment, monitoring and operation of CPR over the years.

References

[1] The CAIDA UCSD AS classification dataset, 2020-07-01. https://www.caida.org/data/as-classification.

[2] Geometric distribution. In *Statistical Distributions*, chapter 23, pages 114–116. John Wiley & Sons, Ltd, 2010.

[3] Linux ip sysctls. https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt, Aug. 2020.

[4] SONiC default BGP hold timer. https://github.com/Azure/sonic-buildimage/blob/833584eff96fca37579d2d792807a8b69c47e701/dockers/docker-fpm-frr/frr/bgpd/templates/general/instance.conf.j2#L8-L9, Nov. 2020.

[5] FRRouting default BGP hold timer. https://github.com/FRRouting/frr/blob/ca7b6587b2a8d3d15e04c00b5201030340d5d1d2/bgpd/bgpd.h#L1756, Feb. 2021.

[6] F. Ahmed, M. Z. Shafiq, A. R. Khakpour, and A. X. Liu. Optimizing internet transit routing for Content Delivery Networks. *IEEE/ACM Transactions on Networking*, 26(1):76–89, 2018.

[7] A. Akella, B. Maggs, S. Seshan, A. Shaikh, and R. Sitaraman. A measurement-based analysis of multihoming. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, pages 353–364, New York, NY, USA, 2003. ACM.

[8] A. Akella, J. Pang, B. Maggs, S. Seshan, and A. Shaikh. A comparison of overlay routing and multihoming route control. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, pages 93–106, New York, NY, USA, 2004. ACM.

[9] A. Akella, S. Seshan, and A. Shaikh. Multihoming performance benefits: an experiment evaluation of practical enterprise strategies. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '04, page 9, USA, 2004. USENIX Association.

[10] Apple. Improving Network Reliability Using Multipath TCP. https://developer.apple.com/documentation/foundation/urlsessionconfiguration/improving_network_reliability_using_multipath_tcp.

[11] Arista. Border Gateway Protocol (BGP) - EOS 4.25.1F User Manual. https://www.arista.com/en/um-eos/eos-border-gateway-protocol-bgp, 2021.

[12] B. Augustin, T. Friedman, and R. Teixeira. Measuring load-balanced paths in the internet. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 149–160, New York, NY, USA, 2007. Association for Computing Machinery.

[13] B. Augustin, T. Friedman, and R. Teixeira. Measuring multipath routing in the internet. *IEEE/ACM Transactions on Networking*, 19(3):830–840, June 2011.

[14] D. Barroso. Developing and evolving your own control plane. NANOG '71. https://pc.nanog.org/static/published/meetings/NANOG71/1438/20171002_Barroso_Developing_And_Evolving_v1.pdf, Oct. 2017.

[15] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, oct 2008.

[16] O. Bonaventure and S. Seo. Multipath TCP deployments. *IETF Journal*, 2016, November 2016.

[17] M. Calder, R. Gao, M. Schröder, R. Stewart, J. Padhye, R. Mahajan, G. Ananthanarayanan, and E. Katz-Bassett. Odin: Microsoft's scalable fault-tolerant CDN measurement system. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 501–517, Renton, WA, apr 2018. USENIX Association.

[18] Cisco. Cisco IOS IP routing: BGP command reference. https://www.cisco.com/c/en/us/td/docs/ios/iproute_bgp/command/reference/irg_book/irg_bgp4.html, Oct. 2013.

[19] D. Clark. The design philosophy of the DARPA internet protocols. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, pages 106–114, New York, NY, USA, 1988. Association for Computing Machinery.

[20] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70:066111, Dec 2004.

[21] A. Dhamdhere and C. Dovrolis. ISP and egress path selection for multihomed networks. In *Proceedings IEEE*

INFOCOM 2006. 25th IEEE International Conference on Computer Communications, pages 1–12, 2006.

[22] Z. Duliński, R. Stankiewicz, G. Rzym, and P. Wydrych. Dynamic traffic management for SD-WAN inter-cloud communication. *IEEE Journal on Selected Areas in Communications*, 38(7):1335–1351, 2020.

[23] O. Filip, P. Machek, M. Mares, M. Matejka, and O. Zajicek. BIRD 2.0 user's guide. https://bird.network.cz/?get_doc&v=20&f=bird-6.html, Oct. 2019.

[24] A. Ford, C. Raiciu, M. J. Handley, O. Bonaventure, and C. Paasch. TCP extensions for multipath operation with multiple addresses. RFC 8684, RFC Editor, March 2020.

[25] V. Giotsas, C. Dietzel, G. Smaragdakis, A. Feldmann, A. Berger, and E. Aben. Detecting peering infrastructure outages in the wild. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 446–459, New York, NY, USA, 2017. ACM.

[26] D. Goodin. BGP event sends European mobile traffic through China Telecom for 2 hour. https://arstechnica.com/information-technology/2019/06/bgp-mishap-sends-european-mobile-traffic-through-china-telecom-for-2-hours/, June 2019.

[27] J. He and J. Rexford. Toward internet-wide multipath routing. *Netwrk. Mag. of Global Internetwkg.*, 22(2):16–21, Mar. 2008.

[28] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, and L. Vanbever. Blink: fast connectivity recovery entirely in the data plane. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 161–176, Boston, MA, Feb. 2019. USENIX Association.

[29] T. Holterbach, S. Vissicchio, A. Dainotti, and L. Vanbever. SWIFT:predictive fast reroute. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 460–473, New York, NY, USA, 2017. ACM.

[30] J. Hopcroft and R. Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June 1973.

[31] J. Iyengar and M. Thomson. QUIC: a UDP-based multiplexed and secure transport. Internet-Draft draft-ietf-quic-transport-34, IETF Secretariat, January 2021.

[32] Y. Jin, S. Renganathan, G. Ananthanarayanan, J. Jiang, V. N. Padmanabhan, M. Schroder, M. Calder, and A. Krishnamurthy. Zooming in on wide-area latencies to a global cloud provider. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, pages 104–116, New York, NY, USA, 2019. Association for Computing Machinery.

[33] Juniper. hold-time (protocols BGP). https://www.juniper.net/documentation/en_US/junos/topics/reference/configuration-statement/hold-time-edit-protocols-bgp.html, Dec. 2020.

[34] A. Kabbani, B. Vamanan, J. Hasan, and F. Duchene. Flowbender: flow-level adaptive routing for improved latency and throughput in datacenter networks. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, pages 149–160, New York, NY, USA, 2014. Association for Computing Machinery.

[35] N. Kephart. Route leak causes global outage in level 3 network. https://www.wired.com/2008/02/pakistans-accid/, Feb. 2015.

[36] N. Keukeleire, B. Hesmans, and O. Bonaventure. Increasing broadband reach with hybrid access networks. *IEEE Communications Standards Magazine*, 4(1):43–49, 2020.

[37] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed internet routing convergence. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '00, pages 175–187, New York, NY, USA, 2000. Association for Computing Machinery.

[38] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi. The QUIC transport protocol: design and internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 183–196, New York, NY, USA, 2017. ACM.

[39] S. Larson. Here's why you may have had internet problems today. https://money.cnn.com/2017/11/06/technology/business/internet-outage-comcast-level-3/index.html, June 2017.

[40] H. H. Liu, Y. Wang, Y. R. Yang, H. Wang, and C. Tian. Optimizing cost and performance for content multihoming. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 371–382, New York, NY, USA, 2012. ACM.

[41] D. Madory. Widespread impact caused by Level 3 BGP route leak. https://dyn.com/blog/widespread-impact-caused-by-level-3-bgp-route-leak/, Nov. 2017.

[42] A. Medina. CenturyLink / Level 3 outage analysis. https://blog.thousandeyes.com/centurylink-level-3-outage-analysis/, August 31st, 2020.

[43] K. P. Murphy. *Machine learning: a probabilistic perspective*. The MIT Press, Cambridge, MA, 2012.

[44] E. Nordmark and M. Bagnulo. Shim6: Level 3 multihoming shim protocol for IPv6. RFC 5533, RFC Editor, June 2009.

[45] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: machine learning in Python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[46] B. Peirens, G. Detal, S. Barre, and O. Bonaventure. Link bonding with transparent Multipath TCP. Internet-Draft draft-peirens-mptcp-transparent-00, IETF Secretariat, July 2016.

[47] Y. Pi, S. Jamin, P. Danzig, and F. Qian. Latency imbalance among internet load-balanced paths: a cloud-centric view. In *Abstracts of the 2020 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '20, pages 65–66, New York, NY, USA, 2020. Association for Computing Machinery.

[48] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271, RFC Editor, January 2006.

[49] L. Saino. Hashing on broken assumptions. NANOG '70. https://www.nanog.org/sites/default/files/1_Saino_Hashing_On_Broken_Assumptions.pdf, June 2017.

[50] M. Sargent, J. Chu, D. V. Paxson, and M. Allman. Computing tcp's retransmission timer. RFC 6298, RFC Editor, June 2011.

[51] B. Schlinker, I. Cunha, Y.-C. Chiu, S. Sundaresan, and E. Katz-Bassett. Internet performance from Facebook's edge. In *Proceedings of the Internet Measurement Conference*, IMC '19, pages 179–194, New York, NY, USA, 2019. Association for Computing Machinery.

[52] B. Schlinker, H. Kim, T. Cui, E. Katz-Bassett, H. V. Madhyastha, I. Cunha, J. Quinn, S. Hasan, P. Lapukhov, and H. Zeng. Engineering egress with Edge Fabric: steering oceans of content to the world. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 418–431, New York, NY, USA, 2017. ACM.

[53] R. Singel. Pakistan's accidental youtube re-routing exposes trust flaw in net. https://www.wired.com/2008/02/pakistans-accid/, Feb. 2008.

[54] B. Stelter. Time Warner Cable comes back from nationwide Internet outage. https://money.cnn.com/2014/08/27/media/time-warner-cable-outage/index.html, Aug. 2014.

[55] R. R. Stewart. Stream Control Transmission Protocol. RFC 4960, RFC Editor, September 2007.

[56] S. Tao, K. Xu, Y. Xu, T. Fei, L. Gao, R. Guerin, J. Kurose, D. Towsley, and Z.-L. Zhang. Exploring the performance benefits of end-to-end path switching. *SIGMETRICS Performance Evaluation Review*, 32(1):418–419, June 2004.

[57] J. Taveira Araújo. Building and scaling the Fastly network, part 1: fighting the FIB. https://www.fastly.com/blog/building-and-scaling-fastly-network-part-1-fighting-fib/, May 2016.

[58] J. Taveira Araújo, R. Landa, R. G. Clegg, and G. Pavlou. Software-defined network support for transport resilience. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–8, May 2014.

[59] J. Taveira Araújo, L. Saino, L. Buytenhek, and R. Landa. Balancing on the edge: Transport affinity without network state. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 111–124, Renton, WA, 2018. USENIX Association.

[60] R. Teixeira, K. Marzullo, S. Savage, and G. M. Voelker. Characterizing and measuring path diversity of internet topologies. In *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '03, pages 304–305, New York, NY, USA, 2003. Association for Computing Machinery.

[61] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro,

F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. Scipy 1.0 – fundamental algorithms for scientific computing in Python. *arXiv e-prints*, page arXiv:1907.10121, Jul 2019.

[62] D. Walton, A. Retana, E. Chen, and J. Scudder. Advertisement of multiple paths in BGP. RFC 7911, RFC Editor, July 2016.

[63] L. Wasserman. *All of Nonparametric Statistics (Springer Texts in Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[64] K.-K. Yap, M. Motiwala, J. Rahe, S. Padgett, M. Holliman, G. Baldus, M. Hines, T. Kim, A. Narayanan, A. Jain, V. Lin, C. Rice, B. Rogan, A. Singh, B. Tanaka, M. Verma, P. Sood, M. Tariq, M. Tierney, D. Trumic, V. Valancius, C. Ying, M. Kallahalla, B. Koley, and A. Vahdat. Taking the edge off with Espresso: Scale, reliability and programmability for global internet peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 432–445, New York, NY, USA, 2017. ACM.

[65] D. Zad Tootaghaj, F. Ahmed, P. Sharma, and M. Yannakakis. Homa: an efficient topology and route management approach in SD-WAN overlays. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 2351–2360. IEEE Press, 2020.

Appendix

A Path diversity

This section provides additional results on path diversity both at the AS path level (extending the analysis presented in §5.3) and at the dataplane level in the specific case of paths between pairs of POPs.

A.1 AS path diversity

This section extends the analysis of AS path diversity presented in §5.3 by reporting path diversity results disaggregated by remote AS type and IP version.

We classify destination ASes according to the CAIDA AS classification dataset [1]. This dataset provides an *AS type* for each candidate destination AS: *Transit/Access* for business providing Internet connectivity; *Content* for ASes which provide content hosting and distribution; and *Enterprise* for other entities that are mostly users, rather than providers of Internet access, transit or content. ASNs for which a classification is unavailable were tagged as *Unknown*.

Our findings are presented in fig. 8. Path diversity seems to induce a stable ordering in which *Content* destination ASes

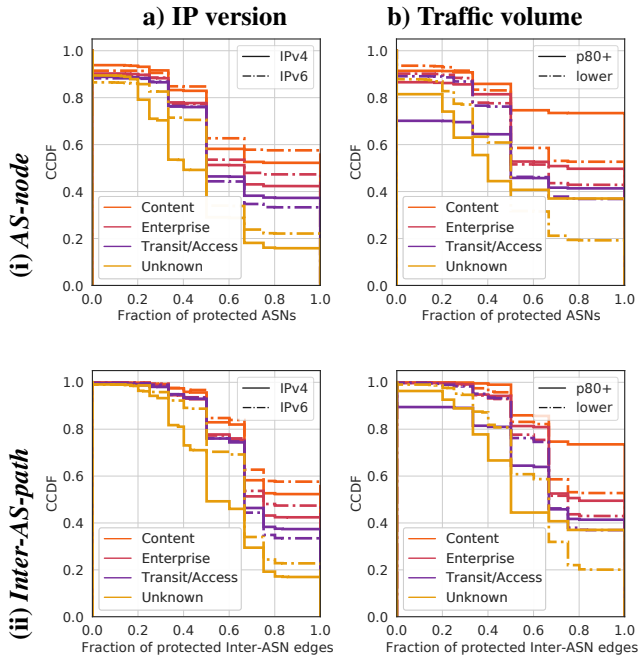


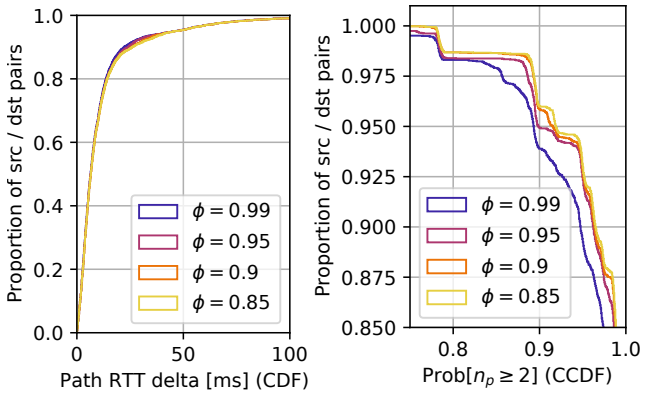
Figure 8: Relationship between both AS-node / Inter-AS path diversity and *AS type*, disaggregated by a) *IP version* of the advertised prefix and b) *Traffic volume* served.

have the highest diversity, followed by *Enterprise*, *Transit/Access* and *Unknown*. While Inter-AS path diversity is understandably higher than AS-node diversity, the ordering appears for both measures. This suggests that, in general, *Content* ASes value path diversity the most; this is unsurprising given the benefits multihoming provides in content delivery [7, 40]. Although similar arguments can be made for multihoming of *Access* [8, 21] and *Enterprise* [9] networks, in practice they exhibit lower path diversity, potentially because they operate at a different cost tradeoff point. Whereas every intermediate AS is node-protected for ~50% of *Content* ASes, this only happens for ~40% of the *Enterprise* and *Transit/Access* prefixes. Overall, there is slightly improved path diversity for IPv6 compared to IPv4, except for *Transit/Access* networks (see fig. 8 i-a and ii-a).

We also consider *size* in our analysis, motivated by the intuition that large, well funded entities may plausibly enjoy better path diversity than smaller ones. We approach this issue by using traffic volume as a proxy for size, disaggregating those prefixes lying in the top 20th percentile (80th percentile overall) by traffic volume served from our edge cloud network. The result of this is presented in fig. 8-b. Although the conjecture that larger entities may enjoy better path diversity is confirmed for every *AS type*, it is emphatic for *Content* destination ASes: around ~75% of high-volume content provider prefixes are fully node- and link-protected (fig. 8 i-b and ii-b respectively). While ~30% of *Transit/Access* destination prefixes seem to not be node-protected at all,

as reported in fig. 8 these correspond to directly connected peers for which path reliability is very high. On the other hand, ~40% of *Transit/Access* destination prefixes are both AS-node and Inter-AS-path protected.

A.2 Dataplane path diversity between POPs



(a) RTT delta between the shortest and longest paths between POP pairs

(b) Proportion of time that two or more paths are available between POP pairs

Figure 9: Inter-POP end-to-end path measurements

Although the BGP-based results presented above are sufficient to justify our path selection approach, as stated in §5.3 they constitute lower bounds. For some use cases it may be advantageous to directly measure dataplane multipath diversity. In this section we present data indicative of the results of such an analysis. We make use of a measurement mesh between our ~80 globally distributed POPs, which continually perform independent RTT measurements over each one of their transit providers. By exploring loss episodes in this mesh we can estimate the probability that, if a BGP-preferred path is unavailable between two POPs, CPR can find another one by random choice.

During a given 30 second measurement window, we compute the *availability* α of a POP-to-POP path as the ratio between the number of *ping* probes for which a response was received and the total number of probes sent. We then compare this to a threshold ϕ, and define a path as *unavailable* if $\alpha < \phi$. We present two measurements using this setup. First, in fig. 9a we show the CDF of the RTT difference between the shortest and longest *available* paths between two given POPs. We can see that in ~80% of cases this RTT delta is of at most ~12.5ms; for ~95% of cases this upper bound rises to 50ms. Note that, in general, *the behavior of this RTT difference is not impacted by our choice of* ϕ. This suggests that, for a large proportion of paths between POPs, alternative paths with similar RTT are available simply by choosing an alternative egress - even for relatively high availability requirements. In some cases, though, these paths *can* induce significant additional delay, and hence this effect cannot be dismissed out of hand.

We also compute the proportion of measurement windows, over a 7 day period, in which at least two POP-to-POP paths were available between any two POPs. This is presented in fig. 9b, which shows that for $\phi = 0.9$, ~96% of POP pairs have two or more *available* paths between them for at least ~90% of the time. Since data collection noise from using *ping* invariably leads to packet losses that do not correspond to path impairments, these numbers constitute a lower bound for the actual data plane path availability. Nevertheless, even this rough approximation shows that the random retry strategy presented in §4 with alternative paths being simply defined by using alternative egress transits is good enough for CPR.

B CPR state probability reconstruction

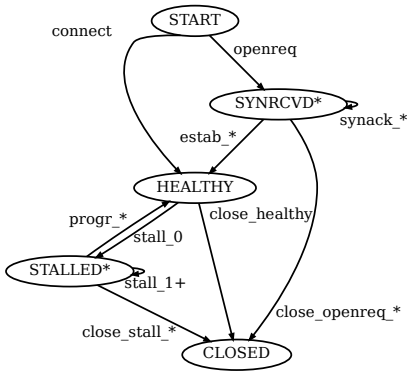


Figure 10: The CPR Finite State Machine

This section provides additional details regarding the computation of stall-event-related CPR performance measurements like those described in §5.2.

CPR state machine. A summarized representation of the CPR state machine is presented in fig. 10. Although CPR tracks the number of individual reroutes while in the `SYNRCVD*` and `STALLED*` states, these internal subdivisions and their associated transitions have been omitted (this is denoted in fig. 10 with * or +). Stall detection logic is evaluated at relevant TCP events (*e.g.,* retransmissions). Transitions between states represent either TCP events (*e.g.,* `openreq`, `estab*`), CPR events (*e.g.,* `stall*`, `progr*`) or both (*e.g.,* `synack*`, `close*`). `SYNRCVD*` and `STALLED*` correspond to points in which path reselection logic can be triggered.

For incoming connections, the `SYNRCVD` meta-state is entered when a SYN is received (`openreq`); this will elicit a SYN-ACK response from the local TCP stack. If this SYN-ACK is acknowledged, the TCP connection enters the `ESTABLISHED` state, and CPR the `HEALTHY` state. Outgoing connections enter the `HEALTHY` state directly upon termination of the three-way handshake. When connections terminate naturally (`close_healthy`), they leave both the TCP and the CPR state machines. The `SYNRCVD` meta-state subsumes one

`synack*` transition for every ACK/SYN-ACK retransmission, and the `STALLED` meta-state one `stall*` transition for every time a stall is declared (see §3.2). The `progr*` transitions capture the possibility of the connection making forward progress again after a number of stalls. The `close*` transition, which corresponds to connection termination, can happen before connection establishment completes (`close_openreq*`), after suffering one or more stalls (`close_stall*`), or as part of an orderly teardown while healthy (`close_healthy`).

Probability reconstruction. Each TCP connection will increment treatment or control counters *for every transition in* fig. 10. To turn these monotonically increasing counters into meaningful probability estimates they are processed in 15 second windows that we denote *export intervals*.

We begin with $\pi_\bullet(A)$, the instantaneous occupancy of a state A, where the subscript will be used to refer to the *treatment* or *control* groups. We note that the value of $\pi_\bullet(A)$ at a given time can be obtained by keeping track of the total number of connections that have entered A and subtracting the total number of connections that have left it; this can be directly computed from the exported counters.

We continue by defining $p^\bullet_\bullet$, the probability estimates for transitions in fig. 10. As before, the subscript will denote either the *treatment* or *control* groups; however, the superscript will be used to denote either *outflow* or *inflow* probabilities. Borrowing from the usual conditional probability notation, we denote an outflow probability $p^o_\bullet(B|A)$ as the probability that, when a connection transitions *out* from state A, it will move on to state B; likewise, we denote an inflow probability $p^i_\bullet(B|A)$ as the probability that a connection transitioning *in* to state B originated from state A. An estimate of $p^o_\bullet(B|A)$ at the end of an export interval can be obtained by computing the ratio of the rate at which connections transitioned from A to B and the *total* rate at which connections transition out from A during that export interval; a similar (but opposite) procedure can be used to compute $p^i_\bullet(B|A)$ (rates can be trivially computed by discrete differentiation). Using $p^\bullet_\bullet$ we can compute specialized measures that can be helpful when estimating the strength of the association between re-routing and stall-related connection properties, such as *e.g.,* the *risk* ratios

$$\rho(A) = \frac{\pi_t(A)}{\pi_c(A)}$$

and

$$\gamma^\bullet(B|A) = \frac{p^\bullet_t(B|A)}{p^\bullet_c(B|A)}.$$

C Stall event extraction

This section provides additional details regarding the extraction of CPR stall events, like those described in §5.2.

To extract *stall events* from raw performance counters we begin by estimating their *span*, defined as the set of timestamps over which they are considered to be *active* (having a correlated effect over the stall-related performance measures $\pi_\bullet/p^\bullet_\bullet/\rho/\gamma^\bullet$ of multiple hosts in the same POP). This is achieved by first identifying candidate per-host events, which can then be clustered at the POP level.

The identification of candidate per-host events begins with standard multidimensional anomaly detection (see *e.g.,* [45,61] and references therein) aimed to identify performance degradation event *candidates*; this yields a series of points in time when treatment/control performance differences over multiple time series may be suggestive of an event. In our case, we use standard peak-finding techniques (see *e.g.,* [45, 61] and references therein) to find "potentially interesting" timestamps for each stall-related performance measure time series, and use the intersection of all these subsets of active timestamps as our set of candidates. These are further refined by applying clustering and de-noising based on the inter-event durations between successive per-host event candidates. This can be achieved by identifying continuous sequences of candidate timestamps. If the separation between any two of these sequences is smaller than a given clustering threshold, they are aggregated as part of the same event candidate; conversely, if after this process their total time span is smaller than a given threshold, the whole candidate set is discarded. This will yield a number of per-host event *spans*, each subsuming multiple candidates.

To model the state of the world both before the event started and after it ended we assign to each *span* a *context*: two time periods, one preceding it and one following it. This can be used as an experimental framework of sorts: if the candidate event finding logic described above correctly identified a set of event candidates, we would expect to find a definite statistical difference between values of at least some $\pi_\bullet/p^\bullet_\bullet/\rho/\gamma^\bullet$ for timestamps within the candidate *span* and those within the *context*. For every performance time series $\rho/\gamma^\bullet$, we compute the average *Mahalanobis distance* [43] between its *context* distribution and its actual values within the candidate span. The resulting set of distances can then be used to reject the candidate if it does not provide enough evidence that the recovered span encompasses a genuine impairment. These distances will also be used to define an event *severity*, following the assumption that a larger context/span difference indicates a more impactful event.

The next stage of the computation is to bring together sets of events at the POP level that happen at approximately the same time and which may hence share an underlying cause. To achieve this, we create a graph where the vertices are the per-host events, and links denote that the spans of the two event nodes *overlap*. This will generate dense cliques when many host-level events overlap, which would happen if multiple edge hosts in a POP experienced correlated stalls as a result of the same event. Although any graph clustering algorithm could be used (*e.g.,* modularity optimization [15, 20]), trivial detection using connected components [30] has proven to be good enough. Once per-POP events have been identified,

event properties from their constituent per-host events (*e.g.,* duration, severity) are aggregated into comparable per-POP event properties.

The logic described can be reused to identify spikes in BGP updates or withdrawals for all the peers and transit providers in a given POP, yielding a number of *BGP events*. These can then be used to enrich the events found from $\pi_{\bullet}/p^{\bullet}_{\bullet}/\rho/\gamma^{\bullet}$ by using a *bookending* co-occurrence heuristic: if a BGP event happens immediately before the CPR event *starts*, it bolsters the conjecture that the CPR event was *caused* by the BGP event; if it happens immediately before the CPR event *ends*, it bolsters the conjecture that it *mitigated* it. By imposing time thresholds on the maximum time difference between the CPR and the BGP event spans and measuring the deviation between the center-of-mass of the BGP event and the start and end timestamps of the CPR event it becomes possible not only to assign suspected peers or transit providers to a given per-POP event, but also to provide a measure of the association strength between a BGP event and a CPR event. We use this to define a *BGP affinity* score to each per-POP event. Finally, the resulting per-POP events are discarded if they violate any one of a number of data quality policies, such as any of the hosts involved being put into maintenance during the per-POP event span; the total number of connections in an edge node being too low, which could induce numerical instability when calculating probabilities; the total number of nodes which observed the per-POP event being too low, which could point towards a false positive; etc.