CAPA: An Architecture For Operating Cluster Networks With High Availability

(Draft under submission)

Bingzhe Liu* Colin Scott† Mukarram Tariq† Andrew Ferguson† Phillipa Gill† Richard Alimi†
Omid Alipourfard† Deepak Arulkannan† Virginia Jean Beauregard† Patrick Conner†
P. Brighten Godfrey* Xander Lin† Joon Ong† Mayur Patel† Amr Sabaa† Arjun Singh†
Alex Smirnov† Manish Verma† Prerepa V Viswanadham† Amin Vahdat†

†Google *UIUC

Abstract

Management operations are a major source of outages for networks. A number of best practices designed to reduce and mitigate such outages are well known, but their enforcement has been challenging, leaving the network vulnerable to inadvertent mistakes and gaps which repeatedly result in outages. We present our experiences with CAPA, Google's "containment and prevention architecture" for regulating management operations on our cluster networking fleet. Our goal with CAPA is to limit the systems where strict adherence to best practices is required, so that availability of the network is not dependent on the good intentions of every engineer and operator. We enumerate the features of CAPA which we have found to be necessary to effectively enforce best practices within a thin "regulation" layer. We evaluate CAPA based on case studies of outages prevented, counter-factual analysis of past incidents, and known limitations. Management-plane-related outages have substantially reduced both in frequency and severity, with a 82% reduction in cumulative duration of incidents normalized to fleet size over five years.

1 Introduction

Cloud applications require high availability from cloud infrastructure. Application deployment patterns vary from non-replicated (single-zone) services to multi-regional replicated services designed for 24/7 global availability [6]. To accommodate this range of deployment patterns, cloud infrastructure must ensure *high baseline availability* within a zone, and *failure domain independence* across zones. Networking, as a baseline dependency for other infrastructure and services, is particularly critical for availability.

Despite the criticality of networking, single zone and correlated multi-zone network failures take place frequently (*e.g.,* [5, 44, 68] in March 2022 alone). At Google, we find that $\geq 58\%$ of cluster (datacenter) network outages since 2018 result from management operations. Previous research has recommended a number of best practices for executing operations in order to improve availability [1, 7, 23]. These recommendations include defining failure domain boundaries, ensuring progressive and supervised change rollout, defense-in-depth, consistency across planes, invariant monitoring, etc. While these best practices are well understood, major cloud providers continue to suffer outages that in hindsight could have been prevented by application of these practices.

In our experience managing Google's fleet of cluster networks, several challenges make it difficult for engineers and operators to consistently follow best practices: **(C1) Complexity and diversity of system interactions.** Cluster networking has evolved into a complex distributed system of many interdependent subsystems. The cross product of the system's behavioral axes, the variety of operations it supports, and the range of services that perform operations result in a huge surface area where best practices might need to be adhered to. **(C2) Difficulty isolating failure domains.** Failure domain isolation reduces correlated failures. But cluster networks are fundamentally coupled as they must connect to each other and exchange routing messages through a wide area network (WAN) [27, 28]. Additionally, centralized operational teams and systems with global responsibilities create coupling through their day-to-day operations. **(C3) Need to balance safety vs. operational velocity.** Aggressive rate limiting can create a logjam of business critical network updates. In practice, operators must resort to some degree of concurrency to keep up. Moreover, certain issues might only be triggered in later stages such that progressive rollout alone is not sufficient. **(C4) Software development velocity overhead.** Strict adherence to best practices comes with high development costs, particularly for a foundational service like cluster networking that cannot always make use of common implementations of convenient abstractions (e.g., distributed storage). Mandating a high-level of rigor for best practice adherence across the dozens of teams and hundreds of engineers that maintain individual subcomponents is at odds with the desire to maintain high software development velocity.

In this paper, we ask: is it feasible to limit the systems where strict adherence to best practices is required, so that availability of the network is not dependant on the best in-

tentions of every engineer and operator? We describe our experiences with CAPA, the "containment and prevention architecture" we have designed and implemented with the goal of systematically enforcing best practices while avoiding the need for all subsystems to be encumbered by enforcement. We enumerate all of the features we have found through experience to be necessary in order to achieve our goal.

Following [31], CAPA organizes subsystems into three layers: a *regulation layer* separates the *production critical layer* below from the *operations workflow layer* above. CAPA defines rules for each layer.

- The *production critical layer* encompasses the network control and data planes responsible for handling application payload. The components in this layer adhere to strict failure domain containment.
- The *operations workflow layer* contains capacity planning, modeling, and rollout workflows that produce behavioral intents for the production layer based on business needs. The services in this layer make no changes to the production critical layer directly, and instead proxy intended changes via the regulation layer.
- The *regulation layer* initiates behavioral changes to components in the production critical layer, and modulates changes by providing rate limiting, health supervision, and fast rollback of certain operations.

Going well beyond [31], we describe in detail throughout the paper how CAPA addresses challenges C1-C4. To address the complexity of system interactions (C1) CAPA's regulation layer (i) ensures that management operations first 'drain' (move traffic away from) the data plane entities they are about to mutate, to mitigate unexpected interactions even in cases where operators believe the operation will not disrupt traffic, and (ii) enforces *protected cross-section bandwidth*, a novel invariant that ensures 'drain' requests (and therefore operations per (i)) are blocked until there is enough available capacity to safely accommodate the operation (§3.1, §4.1).

To prevent correlated failures across zones (C2), the regulation layer rate-limits management operations across the fleet to prevent them from concurrently affecting too many zones, and the production critical layer ensures via ACL enforcement that outbound control messages originating from within a zone are disallowed thereby limiting blast radius and ensuring "low-dependency" bootstrap capability (§4.1, §4.2). One of the few unavoidable exceptions for outgoing ACL enforcement are BGP routing updates, which are needed because cluster networks must maintain connectivity via the WAN. To mitigate correlated failures that propagate through BGP, we have developed *fail-static* fallback routing that allows cluster networks to maintain WAN connectivity even when dynamic BGP routing fails (§4.3). When dynamic BGP routing fails, our border routers continue forwarding packets through pre-programmed static routes to existing traffic engineering tunnels in the data plane. Although these backup routes may be less optimal, many packets will continue arriving at their destination thanks to end-host mechanisms such as PLB [50], as long as a working path exists in the data plane.

Safety policies such as protected cross-section bandwidth and the maximum allowable concurrency of operations need to be adjusted to meet the velocity needs of each type of management operation (C3) (§4.1). The regulation layer provides a centralized place for these adjustments.

In CAPA, only systems in the production critical layer have strict failure domain isolation requirements. Developers at the operations workflow layer are unhindered by these requirements (C4), e.g., they can rely on high-level abstractions and build globally scoped systems which are easier to build and reason about [41]. Since all outputs from the workflow layer are filtered through the regulation layer before propagating to the production critical layer, we largely avoid coupling between global failure domains in the workflow layer and the carefully contained failure domains in the production layer.

CAPA has helped us make considerable progress towards systematic enforcement of production best practices, which improves both baseline availability and failure domain independence. We perform a qualitative retrospective analysis of relevant outages (including before CAPA's deployment) to show that 84% could have been prevented or mitigated by CAPA. Moreover, our fleet's annual incident rate has decreased by more than 73% from pre- to post-deployment. As CAPA was our primary investment in reliability improvements, given the substantial improvements and a number of "near-miss" success stories, it is fair to attribute most of this improvement to the design and deployment of CAPA. At the same time, our work is not complete; we characterize incidents where CAPA did not help, highlighting key limitations.

2 Background

2.1 Cloud deployment patterns

The grouping of servers connected by a datacenter fabric is called a *cluster*, and we therefore refer to the datacenter network fabric as a *cluster network*. Compute resources are presented as *zones* ($\geq$ 1 co-located clusters) and *regions* ($\geq$ 3 zones) to customers. Cloud customers use these building blocks for a range of deployment patterns, from non-replicated services to multi-region replication [6, 13, 21].

In order to provide 99.9% zonal availability, each zone can only tolerate ~4 minutes per month of downtime. This implies that any failure event that requires human intervention will almost certainly render a zone as out of service-level-objective (SLO). Correlated failures (those that simultaneously affect multiple zones) are particularly problematic as they negate customer replication.

2.2 Cluster networking

As described elsewhere [16, 47, 54], Google's software-defined network (SDN), a.k.a. "cluster network service", includes:

- The data plane switches of a Jupiter [47, 54] fabric, including their operating system and firmware stack.
- SDN applications, responsible for programming switches, tracking link state, traffic matrices, etc.
- The dedicated machines called network control servers (NCS) that run the SDN applications. These machines are completely independent from normal machines.
- The dedicated instance of the Borg [61] cluster scheduler that manages the NCS.
- A separate control plane network (CPN) that provides out-of-band connectivity between the NCS machines and data plane switches.

Steady state behavior refers to how—in the absence of ongoing management operations—the cluster network service treats the user payload (network packets) while responding to changes in the runtime environment (e.g., shifts in traffic, links failures). The primary expectation from users is reachability from cluster machines to destinations both inside the cluster and outside (through the WAN) with expected latency, packet loss rates, and bandwidth.

We define an *outage* as an event where the network service behavior fails to meet user expectations, defined precisely as active prober packet loss or latency exceeding predefined thresholds. Examples include blackholes, capacity loss that leads to congestion, or suboptimal pathing.

2.3 Best practices for operations and their implementation challenges

Since management operations are a common cause of outages, several studies have analyzed outage incidents [1, 7, 23, 42] and proposed best practices for operating networks with high availability. We recap the most relevant practices, as well as challenges we have faced in enforcing them.

Failure domain containment. System designers should identify failure domains (e.g., groupings of physical entities which share power supply, cooling, software control, etc.) and enforce as much as isolation as possible between domains in order to minimize the risk that a failure's "blast radius" will extend beyond domain boundaries.[1] Failure domain isolation also reduces the risk of lockout during disaster recovery, since proper isolation requires systems to avoid cyclic dependencies involved in bootstrapping the failure domain (e.g., the cluster network service cannot depend on a distributed storage service which in turn depends on the cluster network service for its operation).

In practice system designers need to strike a balance between minimizing blast radius and minimizing system complexity and cost (C1, C2). For example, some exceptions need to be made in practice for communication (e.g., BGP routing messages) that is allowed to cross failure domain boundaries.

Progressive, supervised change rollout. Faulty operations can lead to service disruptions or violations of failure domain containment. To mitigate these risks, changes should be rolled out gradually. Progressive rollout alone does not improve safety; the health of the system needs to be closely supervised throughout the rollout, otherwise the network remains vulnerable to "slow wrecks" where service behavior gradually but continuously degrades. To keep up with the volume of operations needed to sustain the business, the system must strike a good balance between safety and velocity (e.g., level of concurrency) of operations (C3).

Monitoring and upholding invariants. Safety invariants should be monitored during change rollout to maintain system health [23]. Black-box metrics (those that do not assume any knowledge of the internal implementation details of the network, and therefore capture the user's perspective), such as remote procedure call (RPC) latency or success rates, are effective at detecting failures regardless of root cause as soon as the service behavior begins degrading [1] since they capture the exact conditions that the user experiences, but they are not effective at detecting the degradation before it occurs. Subject matter experts can also identify white-box invariants, i.e. those that leverage implementation details of their subsystems (e.g., "at least 2 replicas should be responsive"), but white-box invariants that generalize to a diversity of system interactions have remained elusive (C1).

Provisioning capacity headroom. Provisioning more capacity than strictly needed to accommodate traffic improves safety. For example, headroom ensures that when a management operation (especially those that require involvement from human technicians) becomes stuck, the operation can remain in a paused state without risking congestion on the unaffected portions of the network. The costs of deploying extra capacity can be ameliorated by serving low priority (QoS) traffic during periods without ongoing operations.

Fast recovery. Fast recovery limits outage duration. In practice, recovery often requires humans to understand the situation and determine a recovery strategy. In a system with many ongoing changes it is not always obvious what needs to be rolled back. Systems can also fail in ways that cannot be recovered by rollback, e.g., if the outage wipes out data, or prevents access to the system itself – a real concern for networking. Automated recovery that rolls back too aggressively

[1] Failure domains are hierarchical. For example, top-of-rack (ToR) switches reside in the same physical failure domain as the machines on the rack; each ToR connects to multiple middleblocks (another failure domain); the middleblocks reside within an aggregation block which are designed to be both physically independent from other aggregation blocks (separate power and cooling supplies) and logically independent (separate software control) [16].

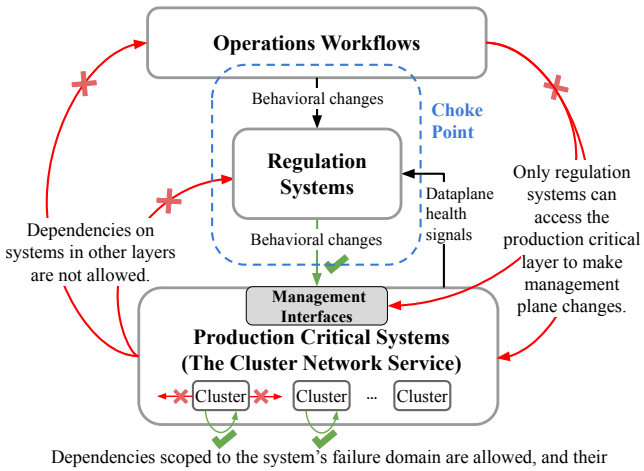

Figure 1: CAPA's layering and enforcement architecture.

can also be at odds with operational velocity (C3).

3 CAPA architecture Overview

CAPA is Google's system for safely operating our cluster networks. Following [31], CAPA organizes system components into three layers, as shown in Figure 1: (a) the *production critical layer*, comprising the components (control plane and data plane) that are responsible for delivery of application payload. We must ensure failure independence (§4.2, §4.3) and high baseline availability in this layer to meet service expectations (SLOs). (b) the *regulation layer*, a small number of centrally maintained services with exclusive access to the management interfaces in the production critical layer. It upholds best practices by applying changes progressively (§4.1) while monitoring system health and invoking automatic rollback of operations ("behavioral changes") if needed (§3.1). The regulation layer maintains the key invariant of ensuring sufficient bandwidth at all cross-sections of the network (§3.1), and enforces global rate limiting to reduce the risk of correlated failures triggered by faulty operations concurrently executing in different zones (§4.1). (c) the *operations workflow layer* contains all other systems needed for operations. A key function of this layer is generating the intended behavior (configuration) of the production critical layer, and realizing that intent by invoking operations via the regulation layer.

3.1 Life cycle of an operation

Before we dive into the details of CAPA's components, we will walk through the life cycle of an example management operation in CAPA.[2] We choose a capacity expansion as our example. Table 1 lists other types of operations.

Capacity expansion starting in the workflow layer. Based on projected traffic demands, a forecasting service in the workflow layer predicts that capacity will be needed on the

[2] §6 gives an overview of network management at Google prior to CAPA.

Category of Operation	Frequency
Capacity expansion/contraction	1 per 3.6 months
Switch software upgrade	1 per 4 weeks
Controller software upgrade	1 per 2 weeks
Hardware repairs	1 per 35 minutes

Table 1: Categories of behavioral changes ("operations") that our networks undergo. Frequencies are computed as averages per cluster network across all clusters over one year.

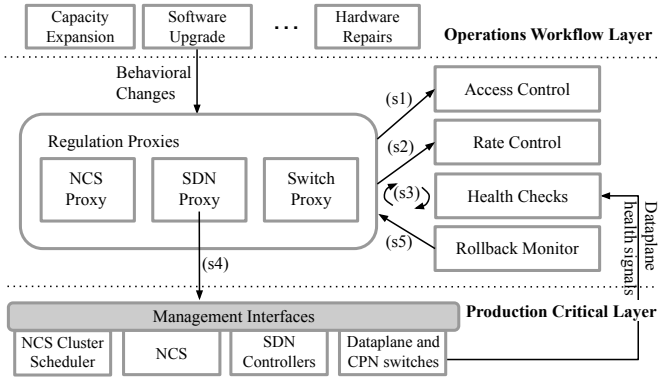

Figure 2: Detailed view of the regulation layer.

WAN-facing links of the fabric border router (FBR) portion of a cluster network. This in turn initiates a capacity expansion workflow that begins by retrieving a model of which entities (e.g., fibers, transceivers) will need to be added to the network [41]. The workflow dispatches purchase orders and waits for delivery of the new materials to the datacenter floor. Once the materials arrive at the datacenter floor, the capacity expansion workflow dispatches instructions to technicians on how to wire and qualify (e.g., verify that bit error rate is sufficiently low for) new physical links. The physical changes needed for the capacity expansion are "append-only," meaning that technicians do not need to mutate any pre-existing links carrying live traffic. We reserve spare ports on both the FBR side and the WAN side to make this possible.

Capacity expansion processed by the regulation layer. For scalability of routing updates, we maintain link aggregation ("trunks") at the WAN boundary: we split all WAN-facing physical links into four trunks (each with distinct power and cooling failure domains), and assign a single SDN control domain responsible for routing over each trunk. Trunks are the level of abstraction that the SDN routing stack operates on; BGP sessions, next-hops, metric computations, etc. are maintained on a per-trunk basis.

To bring the physical capacity into service, the capacity expansion workflow needs to reconfigure each of the SDN control domain's configuration for the trunk it routes over. The workflow first drains traffic from the links under the control domain's purview before initiating any reconfiguration, primarily to protect against any *unexpected* side-effects of the operation; as discussed in §5, unexpected side-effects have

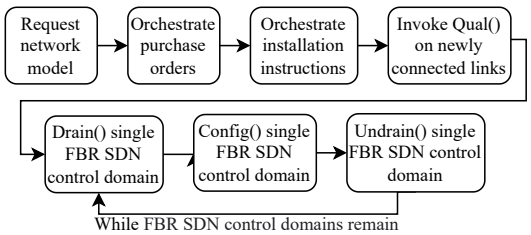

Figure 3: The capacity augment workflow's state machine. The workflow repeatedly invokes APIs in the regulation layer, which are enumerated in Table 2.

historically been a prevalent cause of outages. The regulation layer refuses to process operations that do not already have drains covering the entities they affect.

As depicted in Figure 3, the capacity expansion workflow invokes the `Drain()` API on the SDN proxy in the regulation layer before proceeding with trunk reconfiguration for each FBR SDN control domain. As shown in Figure 2, the SDN Proxy first (s1) confirms whether the workflow has permission to perform a `Drain()`, and (s2) whether the rate limits across the fleet for performing FBR `Drain()` would not be violated. Concurrency control and rate limits reduce the scope of temporally correlated failures by limiting the number of domains that may be affected (§4.1).

Enforcing cross-section capacity. The regulation layer enforces that all target elements should be drained before proceeding with the operation: besides protecting against inadvertent impact on live traffic, this invariant allows the `Drain()` API to double as a semaphore that guards the network's bandwidth while allowing concurrent operations.

Figure 4 depicts how `Drain()` doubles as a semaphore. Multiple operations, each requesting drains, can proceed concurrently so long as the effective ("residual") capacity in all relevant cross-sections of the network are above a threshold (see §4.1 for details). Example network cross-sections on a per cluster network basis include: all WAN-facing links, inter-aggregation-block links, intra-aggregation-block links, top-of-rack (ToR) to middleblock links, and control plane network links between SDN controllers and data plane switches [54]. Cross-sections can be overlapping and hierarchical, e.g., we seek to prevent partitions of any particular aggregation block, but we also protect aggregate interblock bandwidth.

Operations are blocked when the threshold is reached, but they can proceed as soon as the previously in-progress operations conclude and restore capacity by issuing an `Undrain()`. The semaphore also gracefully handles organic failures; in case some network elements lose their ability to service traffic (e.g., due to hardware failures), the effective capacity is automatically reduced, thus blocking subsequent operations.

From a workflow's perspective, `Drain()` is atomic, i.e., if it succeeds, the residual network is in a safe state and the drained entities are safe to operate on. The SDN proxy processes `Drain()` requests in a serial order within each cluster network, which prevents deadlock so long as each workflow requests a single `Drain()` at a time. In the case that there is insufficient effective bandwidth (or some other health check fails, as we describe below), the `Drain()` invocation is blocked and leaves no side effects until it is unblocked.

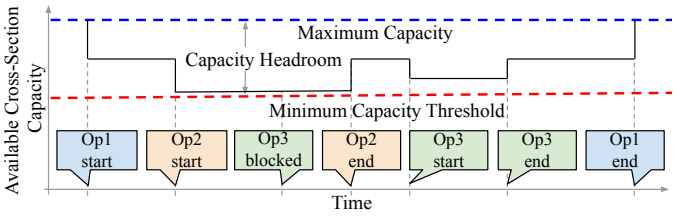

Figure 4: Example of drain thresholds ("ensure minimum cross-section bandwidth") acting as a semaphore for regulating concurrent operations. Op3 is blocked until capacity is released by Op1/Op2.

Interface	Description
SDN Controllers	
Config()	Configuration used by SDN apps. Includes fabric topology, routing policies, etc.
Drain/ Undrain()	Deprefer the targeted elements from the routing solution. May require coordination across domains.
Dataplane and CPN Switches	
Install()	Updates software running on the switch.
Config()	Modifies non-routing behavior, e.g., port speeds.
Qual()	Initiates disruptive bit-error-rate test on a port.
NCS Cluster Scheduler	
Add/Remove Machine()	Modify set of servers running NCS jobs.
Create/Update/ DeleteJob()	Create, update or delete the jobs to run in the cluster.
Config()	Updates resource management policies.
NCS	
Install/Update()	Install or update the firmware and OS of the server.
Restart/ Shutdown()	Reboot or shutdown the server.

Table 2: List of RPC interfaces for management plane operations on various components in the cluster networking service (non-exhaustive).

Applying the drain and configuration changes. The SDN Proxy actuates the `Drain()` by invoking management interfaces on the production critical layer (s4). Once the `Drain()` completes, the workflow may proceed with reconfiguration of trunks by invoking a `Config()` API[3] on the SDN Proxy within the regulation layer. Table 2 describes other management APIs provided by the regulation layer.

The proxies execute the same steps (s1-s4) to actuate the reconfiguration. Once reconfiguration is complete, the workflow will issue an `Undrain()` and repeat starting from the `Drain()` for the next FBR SDN control domain.

Continuous health checks. Network health can change rapidly and unexpectedly, e.g., from hardware failures, sud-

[3]The format of the `Config()` change is arbitrarily defined by application developers; in this case it describes which physical links the routing controllers should aggregate together as a trunk.

den traffic shifts, or unexpected side-effects of operations.

The regulation layer continuously checks two types of health signals—black-box and white-box signals (introduced in §2.3)—both before (s3) and during (s4) the execution of operations. While black-box signals are a lagging indicator of problems (e.g., discards occur only after congestion reaches unsafe levels), some white-box signals can act as preventive leading signals. Protected cross-section bandwidth is an example of a white-box invariant, and is the only invariant we have found so far that generalizes to all operations. Other examples include ensuring a minimum number of SDN application replicas are live, checks on whether any parts of the cluster networks have entered fail-static [16], bans on certain orderings of operations that are known to cause issues, and outputs from operation simulators or analyzers.

Automated drain rollbacks. When servicing a `Drain()` invocation, the SDN Proxy actuates the request incrementally by repeatedly invoking management interfaces in the production critical layer for a subset of the entities referenced in the `Drain()`. If health checks ever fail (e.g., we detect packet discards, or effective capacity drops below safe thresholds) and only a subset of the entities in the `Drain()` have completed, the SDN Proxy can automatically roll back (s5) the partially executed `Drain()`. If a `Drain()` has already been acknowledged as completed to the workflow, we cannot currently rollback those drains. As part of our future work (§6), we are exploring automated rollback of the operations that proceed after the `Drain()` has been acknowledged.

4 Detailed Design

4.1 Rate Limiting & Concurrency Control

As described in §3.1, the regulation layer performs rate limiting and concurrency control. Rate-limits help maintain failure independence by limiting temporally correlated outages that arise from concurrent operations. Rate limiting also maintains baseline availability by increasing the probability that issues are caught early in low-risk cluster networks.

Rate limiting enforcement. Google uses MALT [41] to represent the elements in our network, their attributes, and their relationships with each other. Using this representation, a global service in the regulation layer builds a fleet-wide model of cluster level failure domains and sub-cluster failure domains (e.g., aggregation blocks) within each cluster. The global service also provides a rate-limiting policy configuration to set the maximum concurrency allowed across (hierarchical) failure domains to balance velocity and production risk. Policies are reviewed by senior members in engineering and operations teams on a per use-case basis.

Equipped with modeling and policy, the global rate limiting system ensures that only the desired number of entities are operated on concurrently or within a time epoch. For example, we can set policies to limit concurrent capacity expansions to only N clusters globally, or require 15 minutes between successive operations on each of all 4 FBR domains within each cluster such that at most N clusters can lose at most 25% of FBR capacity within a 15-minute window.

The rate limiting system accounts for the full hierarchy of failure domains in making its decisions, e.g., to perform an operation such as a `Drain()` upon a middleblock in a particular cluster, all rate-limiting policies that apply to `Drain()` operations will be checked. Any ongoing operations in that middleblock will be checked against policies that apply to middleblocks; then, ongoing operations in the aggregation block that the middlebock resides in will be checked against policies that apply to aggregation blocks, and so on through the cluster, zone, region, and global levels [54]. The operation will be allowed to proceed only if all policies defined at all levels of the hierarchy deem that the operation is in compliance with rate limiting.

Concurrency control and capacity headroom. Rate limiting policies are defined according to the scope of entities that operations intend to mutate. In some cases, operations affect a larger set of entities than intended. Protected cross-section bandwidth and `Drain()` as a semaphore further protect against unintended side-effects and "slow wrecks" where rollout supervision fails to detect degradation.

The threshold we use for protected cross-section bandwidth (capacity headroom) is largely determined by the level of redundancy designed into the network. Across most cross-sections (exceptions in §4.4) of the dataplane, we employ 4 independently operable (sub-cluster) failure domains, and only 3 are needed to maintain sufficient capacity to avoid outages, leading to a minimum 25% capacity headroom. In our example (§3.1), there are 4 independent FBR domains. Similarly, within an aggregation block, we have 4 middleblocks, and we have 4 interblock routing domains [47]. In the control plane network, we generally have 2-way redundancy of switches and paths, so CAPA does not allow draining either unless both are healthy.

4.2 Failure Domain Containment

Separating the operations workflow layer from the production critical layer, and rate limiting operations through the regulation layer helps prevent temporally correlated failures. However, this protection is only effective if the failure domains themselves are well contained.

Appropriately defined failure domain boundaries should align physical points of failure (e.g., power and cooling) with software control. Once failure domain boundaries are defined, we must audit dependencies between software components to ensure that there is no unintentional communication across boundaries. Most interactions among our system components take place over RPCs.[4] For containment, cluster network ser-

[4] Exceptions include BGP (§4.3), NTP (§4.4), and DHCP.

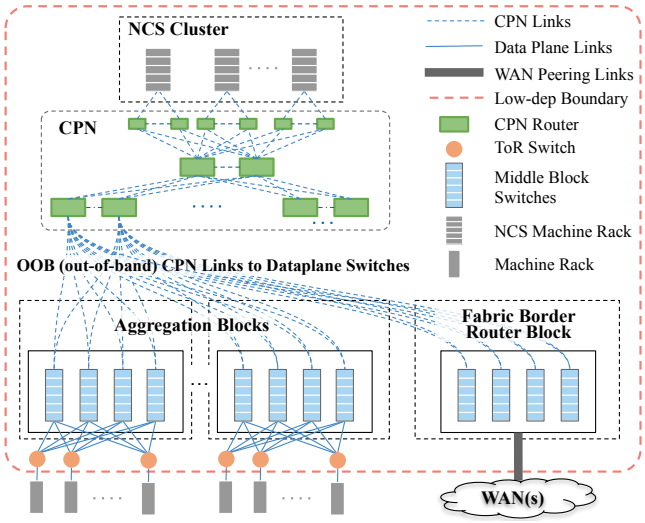

Figure 5: The low-dependency boundary encompasses all elements of the data plane, control plane network (CPN), network control servers (NCS), and their software. RPCs to any services running in normal machine racks or across the WAN are disallowed. (Interblock connectivity and certain other elements not shown for legibility.)

vice components within one datacenter network are not allowed outbound or inbound *modify*[5] RPCs to other cluster networks or other layers (depicted in Fig. 1). To enforce this rule we use a combination of (1) token-based authentication [24], e.g., ACLs to prevent all outbound RPCs, (2) locking down all inbound Management Interfaces in the production critical layer to the proxies in the regulation layer,[6] and (3) monitoring cross-domain RPCs to/from system components to detect any interfaces that may inadvertently remained unprotected.

Failure domain containment enforcement extends to the transitive closure of all the dependencies of the components in the production critical layer, creating a foundational failure domain containment structure that is robust to erosion as the system evolves. We encourage but do not enforce the same failure domain enforcement rules for sub-cluster failure domains (e.g., individual aggregation blocks) in the cluster.

Low-Dependency Requirements Our restriction on outbound RPCs implies that all components within the production critical layer must be "low-dependency", meaning that dependencies outside the cluster network service are disallowed. Figure 5 presents all the elements within the low-dependency boundary. All cluster network service software on switches and SDN control applications on NCS machines must be designed without any outside dependencies on software services that run in the normal machine racks either in the same datacenter or another datacenter. Low-dependency is an expensive requirement to meet as it precludes use of convenient, broadly understood, high-level infrastructure services, such as multi-zonal replicated storage systems, work queues, load balancers, authentication mechanisms, etc.

Ultimately this burden is required considering (i) avoiding circular dependencies on network connectivity simplifies bootstrap and disaster recovery (further discussion in §6), and (ii) the requirement encourages a minimal "trusted computing base" of services that are absolutely needed to maintain steady state behavior. Minimalism reduces the components in the production critical layer and therefore reduces the number of subsystems that can cause an outage.

4.3 Static routes for WAN interconnects

Inter-domain BGP routing updates are one of the few unavoidable exceptions to RPC boundary containment. WAN connectivity requirements are in tension with failure domain isolation as BGP messages themselves are a vector for correlated failures. BGP has multiple properties that make it source of historical outages at Google, including:

1. BGP speakers disseminate information rapidly, which can cause multi-zonal outages from a bad update.

2. BGP speakers lack a global view and make uncoordinated best-path decisions; this can result in replicated speakers creating loops (each peer believing the other has the preferred next-hop) and unintendend route withdraws (e.g., when loops are detected, or when each speaker deprefers its own next-hops).

3. In default configurations, BGP ties together liveness of the data plane and the control plane through keep-alive messages. Upon three missed keep-alive messages, the peer is assumed to have failed and routes are withdrawn, even if the data plane is healthy. This is particularly problematic for SDN where the BGP speaker is separate from the dataplane switches.

To achieve failure domain isolation we require additional defense-in-depth for BGP. We rely on BGP message filtering [11] for defending against (1) and (2), and implement a static route fallback policy for the WAN and CPN routes for (3). Conceptually, fail-static routing splits BGP-based connectivity into two parallel routing systems: traditional, dynamic (optimal) routing via BGP, and a simpler, rarely changing, safe but suboptimal set of "static" routes. The fail-static policy makes traffic fall back to the static routes whenever dynamic routes become unavailable, thus substantially reducing correlated failures that are triggered by unavailability of the BGP control plane without complete loss of the data plane.[7]

From the perspective of the fabric border routers within the cluster network, static routes are configured as a virtual output port that switches can send traffic out of when the SDN

[5] We allow read-only inbound RPCs to facilitate monitoring.

[6] We are further strengthening these boundaries using separate cryptographic scopes for each failure domain. This reduces the principals that may act across failure domains even in the face of security incidents.

[7] In the event of complete data plane partition, packet loss will still occur once the packets reach the partition.

controller detects loss of dynamic BGP messages. Within the WAN, static routes still need to be available at every hop to preserve end-to-end connectivity. We implement static routes in the WAN as either traffic-engineering tunnels that fail-static (for B4 [28]), or redistribution of cached routes even when BGP keep-alive timers have expired.

Static routes can be updated progressively in accordance with best practices, at the cost of temporary mismatch with primary (BGP) routes. In an application of the end-to-end principle [51], these periods of mismatch are acceptable thanks to an end-host mechanism that repaths onto healthy paths [50]. Using knowledge of the network topology, the progressive rollouts are sequenced to avoid creating loops.

4.4 Exceptions

We allow a few exceptions to the `Drain()`-before-operate rule when we conclude that the benefits of simplification outweigh the safety risks. For example, we do not require `Drain()` for most changes to ToR switches (with the exception of decommissioning) as their failure domain is only a handful of machines; further, many racks do not have redundant ToRs, so draining the ToRs would require vacating the underlying machines, creating second order disruption. Similarly, although we require `Drain()` before config changes to (sharded) FBR control domains, we have concluded that sharding of SDN control domains is not worth the complexity cost for our aggregation blocks, and we therefore allow config changes to those control domains without requiring the data plane entities under their purview to be drained.

We also allow a few exceptions to the "low-dependency" restriction, which we permit either to ease the transition from a historical dependency, e.g., NTP from another cluster during switch boot, or as an optimization, e.g., host level telemetry to drive more optimal traffic engineering. These exceptions create a lockout risk, but we require the teams to have well-tested fallback mechanisms so that the cluster service can recover in absence of these dependencies.

5 Evaluation

We evaluate CAPA with three methodologies:

1. We enumerate examples of real production incidents where CAPA successfully prevented or mitigated outages, to examine CAPA's mechanisms in practice.

2. We conduct a comprehensive retrospective analysis of outages affecting our cluster networks from 2018 - 2022 that had behavioral change as their root cause to understand where CAPA would have been beneficial and where CAPA still has limitations. Some of these outages pre-dated CAPA, due to the fact that we incrementally rolled out CAPA's features over multiple years.

3. We examine quantitative data (active network probes) to show incidence trend and severity of management plane related outages. Although this does not establish causality, CAPA was our primary investment in reliability improvements; given quantitative correlation and the first two qualitative evaluations, we can reasonably attribute these gains to CAPA.

For both the qualitative analysis and the quantitative analysis, we consider all outages which exceeded $\geq 5\%$ packet loss across all active probers (§5.4) in at least one cluster network, and lasted for ≥ 5 minutes. We refer to such incidents as "bad fabric incidents." A single outage may cause multiple bad fabric incidents if it affects multiple cluster networks.

5.1 Methodology Discussion

We must cope with three challenges in evaluating CAPA:

1. Our customers will not tolerate us running randomized control trials ("A/B tests") on our deployed reliability mechanisms. Moreover, even if we were able to do so, there is strong evidence that the underlying distribution of outage severity is memoryless ("long-tailed"), implying that we would need to run A/B tests over a multi-year period for statistical soundness.

2. Cluster networking systems, operational workflows, fleet size and CAPA's own capabilities are constantly evolving; we have incrementally rolled out its features over several years, and have continuously made modifications to our safety policies. Hence, there is no single system that can be evaluated.

3. We cannot gather a comprehensive list of all "near-miss" success stories where a CAPA mechanism prevented an outage, largely because it is infeasible to precisely define those circumstances.

While we cannot establish causality, we believe our evaluation methodology effectively illuminates CAPA's impact, particularly given that CAPA was our only investment into hardening against faulty management operations. We employ the same qualitative analyses internally to evaluate new investments into availability improvements.

5.2 Success stories

We first describe a selection of incidents where CAPA successfully prevented or mitigated outages.

Layer separation, rate limiting, health monitoring and fail stop. As noted earlier, Google uses a network model [41] to express intended state of our networks. Operational automation systems continuously monitor differences in the current and intended state of the network, initiating management plane operations to apply the intended state. A bug in the network modeling system mistakenly updated the models

for all the clusters globally. Automation systems, unaware of the buggy nature of this change, detected the differences across the fleet and initiated a huge number of concurrent operations to apply new configuration to dataplane switches to align them with the intended state in the model.

CAPA limited the concurrency and rate of changes, monitored the health of cluster networks, and raised alerts about an unusual backlog of operations. The Drain()-before-operate rule prevented the few operations that were allowed within the rate limits from causing damage to the dataplane. Separation between the operations workflow layer and the production critical layer prevented a buggy global intent from propagating unabated to the fleet, and rate limiting at the regulation layer created temporal separation between operations that—if allowed to execute immediately—would have created correlated failures across multiple zones.

Ensuring disruptive operations are covered by drain. A workflow responsible for decommissioning ToR switches had a bug resulting in an incorrect intent to decommission ToRs across multiple aggregation blocks instead of the specific planned block. The bad intent propagated undetected through several planning systems, but the "ensure operations are covered by drain" validation at the regulation layer rejected the downstream intent after detecting that some ToRs were still serving traffic. If the intent had been allowed to propagate to routing systems, the machines under those ToRs would have become partitioned.

Protected cross-section bandwidth. Our switch config rollout workflow first checks if a switch is drained for repairs, and if so skips pushing configs to that switch until the repair case is resolved (since the switch is likely unresponsive). The database that tracks switches under repairs was undergoing a migration to a new format; to reduce human toil during the migration, the config rollout workflow was modified to query either the old or new format. The rollout workflow misinterpreted the new format, causing it to skip pushing configs to switches that were not under repairs. At a later point, another workflow started to upgrade the software of the switches. The upgraded software on the skipped switches was unable to interpret the stale config, causing the switches to enter an unstable state, and consequently to be excluded from pathing by the routing application. This exclusion led to capacity imbalance at the WAN boundary. The regulation layer detected that cross-section bandwidth had dropped below safe thresholds, and stopped the switch software upgrade workflow from breaking additional switches.

BGP fail static. Maglev [12] is a datacenter-scale load balancing service. It relies on the cluster network to encapsulate packets destined to virtual IP addresses (VIPs) which map to machines in the datacenter. Jupiter's traffic engineering system [47] relies on telemetry about traffic volume to program optimal paths to prefixes. A bug in Maglev software created telemetry reports with invalid network prefixes, which were duly discarded by Jupiter's traffic engineering system. However, without telemetry data, routing entered a fall-back mode using equal-cost-multi-path (ECMP) instead of the more efficient weighted-cost-multi-path (WCMP) scheme. Via a sequence of other optimizations and bugs, several equal-cost prefixes were then aggregated to a single, larger IP prefix that was not present in the export list for the WAN, causing the BGP prefix to be withdrawn. Fortunately, when the WAN stopped receiving dynamic advertisements, static routes kicked in, successfully keeping WAN traffic flowing.

CAPA's rate limiting mechanism prevented concurrent rollouts that could have resulted in simultaneous bugs in multiple cluster networks. Additionally, BGP static routes prevented an outage even when the behavior was triggered by management plane operations for a separate service.

Mechanism	Prevent or Mitigate
Rate limiting, monitoring, and fail stop	74%
Protected cross-section bandwidth	47%
Failure domain containment	11%
Fail-static for WAN interconnects	21%
At least one applicable mechanism	84%
No applicable mechanism*	16%

Table 3: Counterfactual analysis of which CAPA mechanisms either *were* applicable or *would have been* applicable in mitigating or preventing all outages which resulted in at least one bad fabric incident and which had an operation as root cause trigger from 2018 - 2022. Multiple mechanisms could be applicable for any particular outage. *See §5.3.2.

5.3 Retrospective analysis of outages

We retrospectively analyze all management plane triggered outages that resulted in at least one bad fabric incident between 2018 - 2022. Many of these incidents pre-dated CAPA. We analyze which of CAPA's mechanisms either *were*, *would have been*, or *still are not* applicable. Our analysis does not establish statistical significance since the number of outages are insufficient; our goal is instead to gain qualitative insights into CAPA's mechanisms in the context of real incidents.

As Table 3 shows, CAPA's mechanisms either were or would have been applicable to mitigate or prevent 84% of outages. Table 3 demonstrates that there is no single 'magic-bullet' prevention mechanism, due to the wide diversity of failure modes in our systems. Some level of rate limiting was in place during most of these outages, but without additional mechanisms (e.g., protected capacity), management workflows caused "slow wrecks" where the network gradually degraded until an outage occurred. Similarly, gaps in failure domain containment allowed a small number of operations to affect a large number of failure domains.

5.3.1 *Retrospective case studies*

GCP Incident 19009. A significant outage in 2019 [22] affected Google services globally due to misalignment of job scheduling and SDN controller (Orion) failure domains, where a single job controller managed multiple Orion domains. A reconfiguration intended for a single job scheduler triggered cascading failures that brought down all Orion jobs for a large number of zones. This outage underscored the need for considering the transitive closure of dependencies for failure containment: although the SDN controllers, NCS machines, dataplane switches were all in separate failure domains, the job scheduler was not considered as part of the containment.

This outage could have been limited to a single cluster with proper failure domain containment. BGP fail-static would have further mitigated the outage since WAN dataplane connectivity was functional despite failed Orion jobs.

IncidentB. Dataplane switches require a reboot after certain management operations, and are expected to signal the management systems to trigger the reboot. For a new management operation, the switch failed to signal the reboot requirement. A switch configuration rollout exercising this new operation gradually caused a large number of switches to enter an unhealthy state. Orion misinterpreted the lack of responsiveness from these switches as being due to CPN connectivity issues, and then entered massive fail-open (MFO) where it refused to program new routes [16].[8] MFO complicated recovery, since engineers had to find an emergency override that would allow them to reboot undrained switches rather than requesting Orion to drain them before rebooting.

If CAPA had required drains before pushing configs to fabric switches (as it does now), impact could have been avoided. Even if drained switches still broke upon receiving the new config, not all switches would have been affected since protected cross-section bandwidth would have halted drain requests (and therefore config pushes) once the affected entites in the cross-section exceeded a threshold.

IncidentC. The workflow responsible for choosing locations of new machine racks on the datacenter floor relies on human provided layout. An error caused racks to be targeted to the wrong datacenter. The associated turnup workflow concluded that rather than adding new ToRs, it needed to update locations of *existing* ToRs. To do this, the turnup workflow deletes entities in the model and adds them back with updated locations. A different workflow designed to continually monitor model differences and push new configs to the network detected the difference between the delete step and the add-back step and pushed configs that decommissioned a large number of live ToRs. Black-box monitoring did not detect packet loss, since the monitoring system itself decommissioned probing jobs running under those deleted ToRs.

The impact could have been avoided if CAPA had checked whether drains on ToRs were in place before removing the ToR entities from the configuration of aggregation block SDN controllers.[9] Rate limiting configuration pushes to the controllers of the aggregation blocks would not have stopped the operations since the health signal was also affected.

5.3.2 *Retrospective and other known limitations*

We share experiences with a selection of outages where CAPA's mechanisms did not fully prevent an outage. In addition we list other known limitations of the architecture. These highlight that as much as CAPA has helped improve reliability for cluster networking, there is still work to do.

IncidentD. During a capacity augment, operators inadvertently disconnected undrained physical links. CAPA's `Drain()`-before-operate works well to prevent software from making such mistakes, but not humans in the physical world. We have invested in user experience improvements (e.g., pagers, real-time feedback) to reduce these risks, but actions in the physical world cannot be programmatically stopped.

IncidentE. A bug in interaction between `Drain()` and routing applications caused BGP announcements to be revoked before intra-fabric routing finished moving outbound traffic away from WAN-facing links. This resulted in an abrupt loss of forwarding state and packet loss blips for 7 minutes until the intra-fabric routing converged. We have drawn two lessons from this incident: (i) Unsurprisingly, CAPA's own mechanisms can have bugs and hence it is important to have a multi-layer defense. Draining traffic (even if buggy), together with rate limiting as a second layer of defense, contained the duration and the blast radius. (ii) It is important to monitor that each layer is working as intended, so that multiple layers are not simultaneously impaired thereby increasing risk.

IncidentF. A workflow made a modeling change that affected both switch and Orion configs. Later, an unrelated workflow inadvertently pushed those configs to Orion but not switches. This resulted in a version mismatch, causing switches to fall back to ECMP rather than WCMP. ECMP resulted in higher utilization on links with slower speeds. The issue went undetected until the workflow pushed config to enough domains to cause congestion and packet loss. Although CAPA's cross-section capacity monitoring was in place, the implementation at the time (now fixed) failed to account for ECMP behavior over heterogeneous link speeds. This outage illustrates a broader challenge: although cross-section capacity is a remarkably effective health signal, we must continuously update our interpretation of capacity signals to keep astride with advancements in routing.

[8] MFO is a feature in Orion wherein routing apps halt upon unresponsiveness from a large number of dataplane switches. The premise is that switches may become disconnected from the routing apps, but may be serving dataplane just fine, so it is better to keep the current dataplane forwarding state instead of avoiding the unresponsive switches and steering traffic towards a small number of responsive switches, thereby creating congestion.

[9] We except ToRs from `Drain()`-before operate for config pushes direct to a single ToR (§ 4.4), but we still require drains for configuration changes whenever a large multiple ToRs are affected simultaneously.

Metric	2018	2019	2020	2021	2022
Bad Fabric Incidents	1.0x	2.36x	0.28x	0.23x	0.27x
Cumulative Duration	1.0x	5.96x	0.44x	0.10x	0.18x

Table 4: Counts and cumulative duration of "bad fabric incidents" across the fleet (cluster networks with ≥5% loss for ≥5 minutes) that had an operation as their trigger, normalized to the number of switches in the fleet at the beginning of the year, relative to the normalized count for 2018. A single cluster network may undergo ≥1 bad fabric incident.

Other known limitations: Beyond these three cases, we are aware of a number of other limitations of CAPA:
Organic failures: CAPA is designed to mitigate impact from planned changes rather than organic failures such as link cuts (though it does protect capacity headroom, thereby increasing the chance that the network can accommodate traffic despite failures). Similarly, CAPA does not address outages triggered by coincidental or malicious user behavior (e.g., DDoS attacks, or packets of death). We consider protection against such problems as part of the baseline functionality of the network service which should be addressed through multi-path routing, testing, packet filtering, etc.
Infrequently used disaster recovery tools: In CAPA we have allowed both the workflow and regulation layers to be high dependency, i.e., the services in this layer can depend on higher layer services such as distributed storage that potentially depend on multiple cluster networks. This allows creation of cyclic dependencies which create lockout risk during large-scale outages (e.g., GCP Incident 19009 [22]), where sufficient clusters are degraded to render the higher level services unavailable. To address this gap, we provide separate "breakglass" management tools that can be used when low-dependency recovery is required. Fortunately these tools are infrequently used; but infrequency of use increases the chances of encountering bugs when the tools are actually needed. We address this by regular testing at smaller scale and maximizing the shared libraries between the breakglass tools and systems used on a day-to-day basis.

5.4 Quantitative data

Application-perceived availability is determined by the source, destination, and service class (priority) the application uses to send messages. We employ active probing where a subset of machines continuously send packets of various QoS classes to each other in order to monitor network availability.

Table 4 demonstrates trends in total bad fabric incidents caused by management operations in a five-year period (2018 - 2022). We normalize these counts to the size of our fleet (measured in terms of # of fabric switches across all cluster networks) since the number of behavioral changes (and therefore potential # of outages) we execute is proportional to the size of the fleet. Table 4 shows a 73% reduction in normalized bad fabric incidents over the period.

Table 4 also shows cumulative duration of bad fabric incidents caused by management operations. We observe a 82% reduction in normalized duration. A single large outage in 2019 [22] accounts for 73% of cumulative bad fabric duration.

6 Discussion

Does CAPA apply beyond cluster networking? Failure domain containment enforced via RPC ACLs is a generic mechanism that is already used by other systems (e.g., cluster schedulers). Layered enforcement and regulation (Fig. 1) is also applicable to other production services, but it requires clear separation of management plane interfaces from data-plane interfaces, something that not all services do consistently. Lastly, the idea of defense-in-depth through progressive and supervised rollout is applicable to other services, but it needs to be adapted to the specific context. For example, for cluster scheduling it could mean that the resources allocated to a service should be reduced progressively and should never go below a configured threshold.

Is CAPA provably safe? CAPA's architecture is based on principled reasoning about failure domain containment and separation of operational systems from the cluster network service, but we cannot claim that it is provably safe, especially given reliance on domain specific heuristics, exceptions (§4.4), and known limitations (§5.3.2). It is unclear whether provably safe systems for continuously evolving and complex systems like cluster networking are feasible, but the data (§5.2, §5.4) clearly demonstrate that CAPA is effective for real-world large-scale systems, while also providing motivation to continue to invest into improvements.

Should we use progressive updates in the production critical layer itself? In §5, we see a number of incidents that are caused by rapidly propagating updates (e.g., route programming) within the production critical layer. We have pondered whether to rate-limit messages in the production critical layer itself. To date, we have decided against it, as fast responses to organic events—such as switch/link failures, changes in traffic conditions, or route availability from adjacent domains—are critical for predictable performance and low packet loss in the steady state. While rapid route programming undoubtedly increases risk of failures, we believe the tradeoff is worthwhile so long as failures are rare and contained to one cluster (ideally sub-cluster failure domains).

How has CAPA evolved? What's next? CAPA's design is informed by experience accumulated since Google built its first datacenter network. The datacenters were built with failure domain separation at physical, data and control plane levels, but this separation was not enforced. We relied on manual management operations (scripts invoked by humans) in the early days, and then factored out management plane functionality into dedicated services to reduce duplication

across scripts. Security and reliability concerns lead to access controls for these services. Datacenters becoming the basis for Google Cloud accelerated both capacity and feature growth, and underscored the need for more reliable operations. We invested in more advanced network modeling [41], automation in operations, and principles of layering for production operations. Large-scale outages like GCP incident 19009 [22] brought additional emphasis on considering the full transitive closure of dependencies in failure domain containment, BGP fail-static, and low dependency recovery. Our ongoing focus is on preventing erosion of enforcement over time and driving new directions that will further improve the safety and velocity of operations.

Automated rollback of operations beyond `Drain()`: An interesting side-effect of 82% reduction in elapsed outage duration is that newer outages are almost always unique and difficult to prevent with simple principles, even with the benefit of hindsight. This elevates the need for faster recovery. We are working towards fully intent-driven network management that will allow us to quickly roll back to known good state upon failures. For this, we are extending rollback support beyond in-flight `Drain()` operations, as well as *uplevelling* the regulation layer APIs to more directly express intent instead of using the current imperative APIs (Table 2).

Up-leveling SDN's management interface: Today, workflows reach their intended state by sequencing low-level imperative APIs. Workflows are not aware of each other's existence or intent. CAPA has created safe guards that prevent outages, but workflows are still vulnerable to data races and indefinite blockages. We are working on up-leveling APIs and increased awareness of concurrent workflows within the regulation and production critical layers to prevent these outcomes.

7 Related Work

Layering for network management. In a keynote lecture, Koley describes ZTN, a three-layered architecture that regulates network management operations applied to Google's B2 and B4 wide area networks [31]. CAPA adopts the same principle of isolating operation workflows from data/control planes by interposing a regulation layer.

Beyond layering, ZTN describes how operations workflows can be made device-agnostic by acting on an abstract model of the network [41], which is particularly relevant in the wide area network where Google purchases "traditional" routers from hardware vendors. ZTN only briefly touches on other topics such as rate limiting and invariant monitoring. We focus on cluster networking, where failure domains are "leaf nodes" of the overall network, and where network designers have many more degrees of freedom in defining failure domains as a result of our software-defined control plane. We go well beyond ZTN by describing the multiple mechanisms we have built into CAPA which we have found to all be necessary to protect the availability of the network, including 'drain-before-operate', protected cross section bandwidth, fleet-wide rate limiting, ACL enforcement and auditing, low-dependency requirements, and BGP static routes.

Production best practices. Several papers [1, 7, 23, 38, 42] describe reliability best practices for general systems and networked systems in particular. We do not propose novel best practices; we describe our experience in what mechanisms are necessary to systematically enforce them within a thin "regulation" layer.

Network management. The literature on network analysis and testing [26, 53, 64], simulation and emulation [33, 36, 66], verification [9, 14, 17, 18, 29, 30, 48, 55] and synthesis [35, 37, 40, 43, 57, 62] can be applied to change management. These mechanisms test or verify if changes are safe to proceed before releasing them, or automatically generate correct configurations from intent. WAN failure domain containment [32], network update and change planning [2, 34, 49, 52, 56, 65], drain operations [60], impact analysis [19, 58, 65] and failure mitigation [15, 63] are also applicable. These mechanisms allow network operators to manage changes safely and effectively. This paper describes our experience piecing together such mechanisms into a cohesive architecture; while existing literature describes singular aspects of systemically enforcing best practices, we are unaware of any work describing experiences holistically enforcing all best practices.

Failure studies. Prior studies [4, 25, 46, 46] have shown the network to be among the major causes of cloud service outages. Along that line, many failure studies particularly focus on data center networks [3, 8, 10, 20, 23, 35, 36, 39, 45, 46, 59, 63, 67]. Among them, some studies [23, 38, 39] have conclusions that motivate our paper or align with our observations: (1) planned changes and maintenance contribute to a majority of the outages; (2) failures are inevitable in large scale complex systems, and we need to proactively prepare for failure (e.g., with failure domain isolation and fast recovery). Though our paper examines outage case studies as a qualitative evaluation of CAPA, our goal is not to comprehensively analyze the characteristics of observed outages.

8 Conclusion

Production best practices are only effective if they can be reliably enforced. CAPA demonstrates an approach for providing enforcement of best practices without relying on the good intent of every engineer and operator. CAPA has allowed us to maintain availability while facing the practical realities of a continually evolving and large scale production environment.

Even with a 73% reduction in incident rate and 82% reduction in outage duration, the pressure to deliver higher availability, higher operational and feature development velocity, and increased complexity in cluster networking means that this area remains ripe for further innovation.

References

[1] Heather Adkins, Betsy Beyer, Paul Blankinship, Piotr Lewandowski, Ana Oprea, and Adam Stubblefield. *Building Secure and Reliable Systems: Best Practices for Designing, Implementing, and Maintaining Systems*. O'Reilly Media, 2020.

[2] Omid Alipourfard, Jiaqi Gao, Jérémie Koenig, Chris Harshaw, Amin Vahdat, and Minlan Yu. Risk based planning of network changes in evolving data centers. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019*. ACM, 2019.

[3] Peter Bailis and Kyle Kingsbury. The network is reliable. *Commun. ACM*, 2014.

[4] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Ed.* Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2013.

[5] Tom Bedford. Internet outages: Spotify, Discord and more are finally back up. In *TechRadar*, 2022. https://www.techradar.com/news/live/internet-down-spotify-discord-facebook-and-more-are-all-suffering-outages.

[6] Anna Berenberg and Brad Calder. Deployment Archetypes for Cloud Applications. *ACM Computing Surveys*, 2022.

[7] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, Inc., 1st edition, 2016.

[8] Ayush Bhardwaj, Zhenyu Zhou, and Theophilus A. Benson. A Comprehensive Study of Bugs in Software Defined Networks. In *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2021*. IEEE.

[9] Yiyang Chang, Sanjay G. Rao, and Mohit Tawarmalani. Robust Validation of Network Designs under Uncertain Demands and Failures. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017*. USENIX Association.

[10] Sean Choi, Boris Burkov, Alex Eckert, Tian Fang, Saman Kazemkhani, Rob Sherwood, Ying Zhang, and Hongyi Zeng. FBOSS: building switch software at scale. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2018*. ACM.

[11] Jérome Durand, Ivan Pepelnjak, and Gert Doering. BGP operations and security. *RFC*, 7454:1–26, 2015.

[12] Daniel E. Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. Maglev: A Fast and Reliable Software Network Load Balancer. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*.

[13] Google Compute Engine. Designing resilient systems. https://cloud.google.com/compute/docs/tutorials/robustsystems.

[14] Seyed Kaveh Fayaz, Tushar Sharma, Ari Fogel, Ratul Mahajan, Todd D. Millstein, Vyas Sekar, and George Varghese. Efficient Network Reachability Analysis Using a Succinct Control Plane Representation. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*. USENIX Association.

[15] Sifat Ferdousi, Massimo Tornatore, Ferhat Dikbiyik, Charles U. Martel, Sugang Xu, Yusuke Hirota, Yoshinari Awaji, and Biswanath Mukherjee. Joint Progressive Network and Datacenter Recovery After Large-Scale Disasters. *IEEE Trans. Netw. Serv. Manag.*, 2020.

[16] Andrew D. Ferguson, Steve Gribble, Chi-Yao Hong, Charles Killian, Waqar Mohsin, Henrik Muehe, Joon Ong, Leon Poutievski, Arjun Singh, Lorenzo Vicisano, Richard Alimi, Shawn Shuoshuo Chen, Mike Conley, Subhasree Mandal, Karthik Nagaraj, Kondapa Naidu Bollineni, Amr Sabaa, Shidong Zhang, Min Zhu, and Amin Vahdat. Orion: Google's Software-Defined networking control plane. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, 2021.

[17] Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd D. Millstein. A General Approach to Network Configuration Analysis. In *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15*. USENIX Association.

[18] Aaron Gember-Jacobson, Raajay Viswanathan, Aditya Akella, and Ratul Mahajan. Fast Control Plane Analysis Using an Abstract Representation. In *Proceedings of the ACM SIGCOMM 2016 Conference*. ACM.

[19] Aaron Gember-Jacobson, Wenfei Wu, Xiujun Li, Aditya Akella, and Ratul Mahajan. Management Plane Analytics. In *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015*. ACM.

[20] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*.

[21] Google. Google Cloud Basics: Geography and regions. https://cloud.google.com/docs/geography-and-regions.

[22] Google. Google Cloud Networking Incident 19009. https://status.cloud.google.com/incident/cloud-networking/19009.

[23] Ramesh Govindan, Ina Minei, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. Evolve or Die: High-Availability Design Principles Drawn from Googles Network Infrastructure. In *Proceedings of the ACM SIGCOMM 2016 Conference*. ACM.

[24] gRPC Authors. gRPC: Authentication. https://grpc.io/docs/guides/auth/.

[25] Haryadi S. Gunawi, Mingzhe Hao, Riza O. Suminto, Agung Laksono, Anang D. Satria, Jeffry Adityatama, and Kurnia J. Eliazar. Why Does the Cloud Stop Computing? Lessons from Hundreds of Service Outages. In *Proceedings of the Seventh ACM Symposium on Cloud Computing, 2016*. ACM.

[26] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. I know what your packet did last hop: Using packet histories to troubleshoot networks. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014*. USENIX Association.

[27] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven WAN. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, 2013.

[28] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jonathan Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a Globally Deployed Software Defined WAN. In *Proceedings of the ACM SIGCOMM Conference*, Hong Kong, China, 2013.

[29] Peyman Kazemian, George Varghese, and Nick McKeown. Header Space Analysis: Static Checking for Networks. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012*. USENIX Association.

[30] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and Philip Brighten Godfrey. VeriFlow: Verifying Network-Wide Invariants in Real Time. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013*. USENIX Association.

[31] Bikash Koley. Keynote lecture: The Zero Touch Network. In *International Conference on Network and Service Management*, 2016.

[32] Umesh Krishnaswamy, Rachee Singh, Nikolaj S. Bjørner, and Himanshu Raj. Decentralized cloud wide-area network traffic engineering with BLASTSHIELD. In *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022*. USENIX Association.

[33] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.

[34] Bingzhe Liu, Kuan-Yen Chou, Pramod Jamkhedkar, Bilal Anwer, Rakesh K. Sinha, Kostas N. Oikonomou, Matthew Caesar, and Brighten Godfrey. Practical Automation for Management Planes of Service Provider Infrastructure. In *FlexNets '21: Proceedings of the 4th FlexNets Workshop on Flexible Networks Artificial Intelligence Supported Network Flexibility and Agility, 2021*. ACM.

[35] Hongqiang Harry Liu, Xin Wu, Wei Zhou, Weiguo Chen, Tao Wang, Hui Xu, Lei Zhou, Qing Ma, and Ming Zhang. Automatic Life Cycle Management of Network Configurations. In *Proceedings of the Afternoon Workshop on Self-Driving Networks, SelfDN@SIGCOMM 2018*. ACM.

[36] Hongqiang Harry Liu, Yibo Zhu, Jitu Padhye, Jiaxin Cao, Sri Tallapragada, Nuno P. Lopes, Andrey Rybalchenko, Guohan Lu, and Lihua Yuan. CrystalNet: Faithfully Emulating Large Production Networks. In *Proceedings of the 26th Symposium on Operating Systems Principles, 2017*. ACM.

[37] Ajay Mahimkar, Carlos Eduardo de Andrade, Rakesh K. Sinha, and Giritharan Rana. A composition framework for change management. In *ACM SIGCOMM 2021 Conference*. ACM.

[38] Ben Maurer. Fail at scale: Reliability in the face of rapid change. *Queue*, 13(8):30–46, sep 2015.

[39] Justin Meza, Tianyin Xu, Kaushik Veeraraghavan, and Onur Mutlu. A Large Scale Study of Data Center Network Reliability. In *Proceedings of the Internet Measurement Conference 2018, IMC 2018*. ACM.

[40] Jeffrey C. Mogul, Alvin AuYoung, Sujata Banerjee, Lucian Popa, Jeongkeun Lee, Jayaram Mudigonda, Puneet Sharma, and Yoshio Turner. Corybantic: towards the modular composition of SDN control programs. In *Twelfth ACM Workshop on Hot Topics in Networks, HotNets-XII, 2013*. ACM.

[41] Jeffrey C. Mogul, Drago Goricanec, Martin Pool, Anees Shaikh, Douglas Turk, Bikash Koley, and Xiaoxue Zhao. Experiences with Modeling Network Topologies at Multiple Levels of Abstraction. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 403–418, Santa Clara, CA, February 2020. USENIX Association.

[42] Jeffrey C. Mogul, Rebecca Isaacs, and Brent Welch. Thinking about Availability in Large Service Infrastructures. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems, HotOS 2017*. ACM.

[43] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. Composing software defined networks. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013*. USENIX Association.

[44] Jay Peters. Apple has resolved the outage affecting iMessage, Apple Music, the App Store, and other services. In *The Verge*, 2022. https://www.theverge.com/2022/3/21/22989393/apple-is-down-outage-music-imessage-maps-icloud-app-store.

[45] Rahul Potharaju and Navendu Jain. Demystifying the dark side of the middle: a field study of middlebox failures in datacenters. In *Proceedings of the 2013 Internet Measurement Conference, IMC*. ACM.

[46] Rahul Potharaju and Navendu Jain. When the network crumbles: an empirical study of cloud network failures and their impact on services. In *ACM Symposium on Cloud Computing, SOCC '13, 2013*. ACM.

[47] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve D. Gribble, Rishi Kapoor, Stephen Kratzer, Nanfang Li, Hong Liu, Karthik Nagaraj, Jason Ornstein, Samir Sawhney, Ryohei Urata, Lorenzo Vicisano, Kevin Yasumura, Shidong Zhang, Junlan Zhou, and Amin Vahdat. Jupiter evolving: transforming Google's datacenter network via optical circuit switches and software-defined networking. In *SIGCOMM '22: ACM SIGCOMM 2022 Conference*. ACM.

[48] Santhosh Prabhu, Kuan-Yen Chou, Ali Kheradmand, Brighten Godfrey, and Matthew Caesar. Plankton: Scalable network configuration verification through model checking. In *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020*. USENIX Association.

[49] Chaithan Prakash, Jeongkeun Lee, Yoshio Turner, Joon-Myung Kang, Aditya Akella, Sujata Banerjee, Charles Clark, Yadi Ma, Puneet Sharma, and Ying Zhang. PGA: Using Graphs to Express and Automatically Reconcile Network Policies. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015*. ACM.

[50] Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, Qiaobin Fu, Gautam Kumar, Masoud Moshref, Junhua Yan, Van Jacobson, David Wetherall, and Abdul Kabbani. PLB: congestion signals are simple and effective for network load balancing. In *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022.

[51] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-End Arguments in System Design. *ACM Trans. Comput. Syst.*, 1984.

[52] Tibor Schneider, Rüdiger Birkner, and Laurent Vanbever. Snowcap: synthesizing network-wide configuration updates. In *ACM SIGCOMM 2021 Conference*. ACM.

[53] Colin Scott, Andreas Wundsam, Barath Raghavan, Aurojit Panda, Andrew Or, Jefferson Lai, Eugene Huang, Zhi Liu, Ahmed El-Hassany, Sam Whitlock, Hrishikesh B. Acharya, Kyriakos Zarifis, and Scott Shenker. Troubleshooting blackbox SDN control software with minimal causal sequences. In *ACM SIGCOMM 2014 Conference*. ACM.

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

[55] Samuel Steffen, Timon Gehr, Petar Tsankov, Laurent Vanbever, and Martin T. Vechev. Probabilistic Verification of Network Configurations. In *SIGCOMM '20: Proceedings of the 2020 Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. ACM.

[56] Peng Sun, Ratul Mahajan, Jennifer Rexford, Lihua Yuan, Ming Zhang, and Ahsan Arefin. A network-state management service. In *ACM SIGCOMM 2014 Conference*.

[57] Yu-Wei Eric Sung, Xiaozheng Tie, Starsky H. Y. Wong, and Hongyi Zeng. Robotron: Top-down Network Management at Facebook Scale. In *Proceedings of the ACM SIGCOMM 2016 Conference*.

[58] Aisha Syed, Bilal Anwer, Vijay Gopalakrishnan, and Jacobus E. van der Merwe. DEPO: A platform for safe deployment of policy in a software defined infrastructure. In *Proceedings of the 2019 ACM Symposium on SDN Research, SOSR*.

[59] Daniel Turner, Kirill Levchenko, Alex C Snoeren, and Stefan Savage. California fault lines: understanding the causes and impact of network failures. In *Proceedings of the ACM SIGCOMM 2010 Conference*, 2010.

[60] Kaushik Veeraraghavan, Justin Meza, Scott Michelson, Sankaralingam Panneerselvam, Alex Gyori, David Chou, Sonia Margulis, Daniel Obenshain, Shruti Padmanabha, Ashish Shah, Yee Jiun Song, and Tianyin Xu. Maelstrom: Mitigating Datacenter-level Disasters by Draining Interdependent Traffic Safely and Efficiently. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018*. USENIX Association.

[61] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

[62] Andreas Voellmy, Junchang Wang, Yang Richard Yang, Bryan Ford, and Paul Hudak. Maple: simplifying SDN programming using algorithmic policies. In *ACM SIGCOMM 2013 Conference*.

[63] Xin Wu, Daniel Turner, Chao-Chih Chen, David A. Maltz, Xiaowei Yang, Lihua Yuan, and Ming Zhang. Netpilot: automating datacenter network failure mitigation. In *ACM SIGCOMM 2012 Conference*.

[64] Hongyi Zeng, Peyman Kazemian, George Varghese, and Nick McKeown. Automatic test packet generation. In *Conference on emerging Networking Experiments and Technologies, CoNEXT '12, 2012*. ACM.

[65] Ennan Zhai, Ang Chen, Ruzica Piskac, Mahesh Balakrishnan, Bingchuan Tian, Bo Song, and Haoliang Zhang. Check before You Change: Preventing Correlated Failures in Service Updates. In *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020*. USENIX Association.

[66] Qizhen Zhang, Kelvin K. W. Ng, Charles W. Kazer, Shen Yan, João Sedoc, and Vincent Liu. MimicNet: fast performance estimates for data center networks with machine learning. In *ACM SIGCOMM 2021 Conference*.

[67] Danyang Zhuo, Monia Ghobadi, Ratul Mahajan, Klaus-Tycho Förster, Arvind Krishnamurthy, and Thomas Anderson. Understanding and mitigating packet corruption in data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017.

[68] Ariel Zilber. PlayStation network goes down, sparking panic among gamers. In *The New York Post*, 2022. https://nypost.com/2022/03/23/playstation-network-goes-down-sparking-panic-among-gamers/.