Evolve or Die

High-Availability Design Principles Drawn from Google's Network Infrastructure

Govindan et al. · SIGCOMM 2016

Lead discussion: Vincent Nguyen, Cheng Peng

vincentnguyenduc.github.io/cse561-au2026-lead-discussion

TL;DR

- **Data:** 103 post-mortems of Google network failures over two years
- **Finding:** failures hit every network and every plane, with no dominant root cause
- **Cause:** ~70 happened **while the network was being changed** (a management operation)
- **Lesson:** no silver bullet

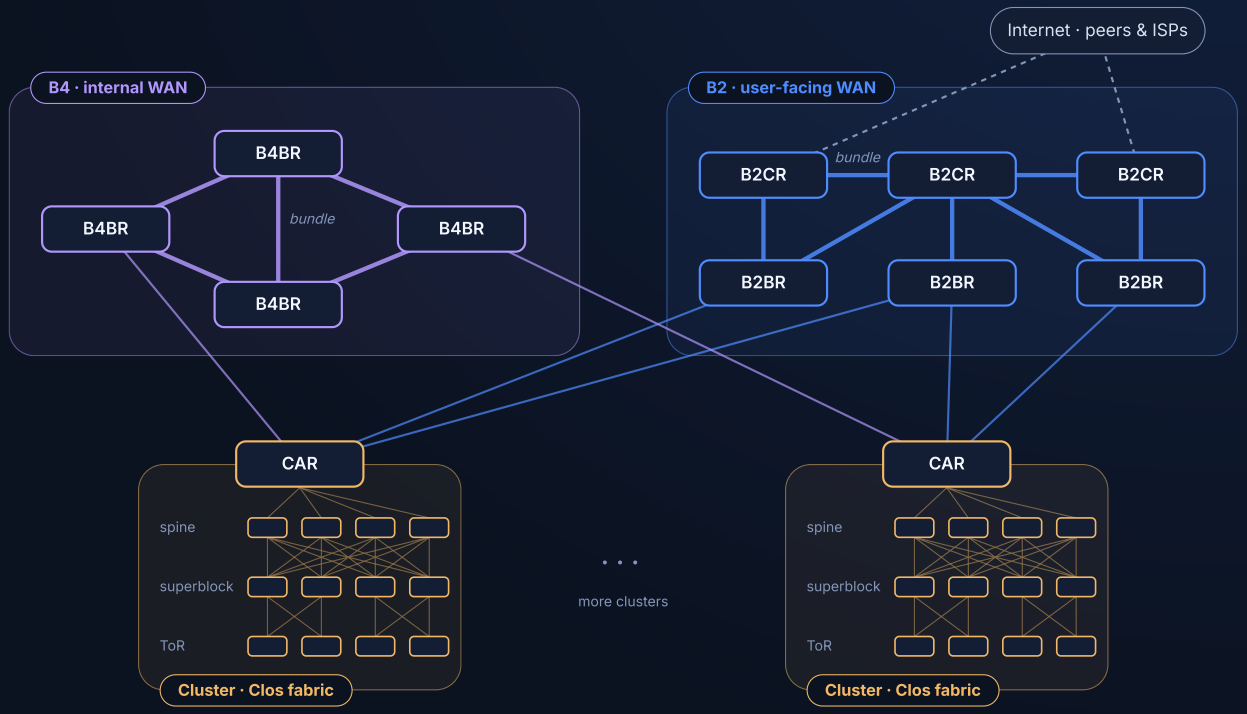
> ~~If it works, don't touch it.~~ Touch it often, and make touching it safe.

Outline

1. Google's network
2. Why high availability is hard
3. What breaks: 103 post-mortems
4. Failures and their lessons

Google's network

Three networks



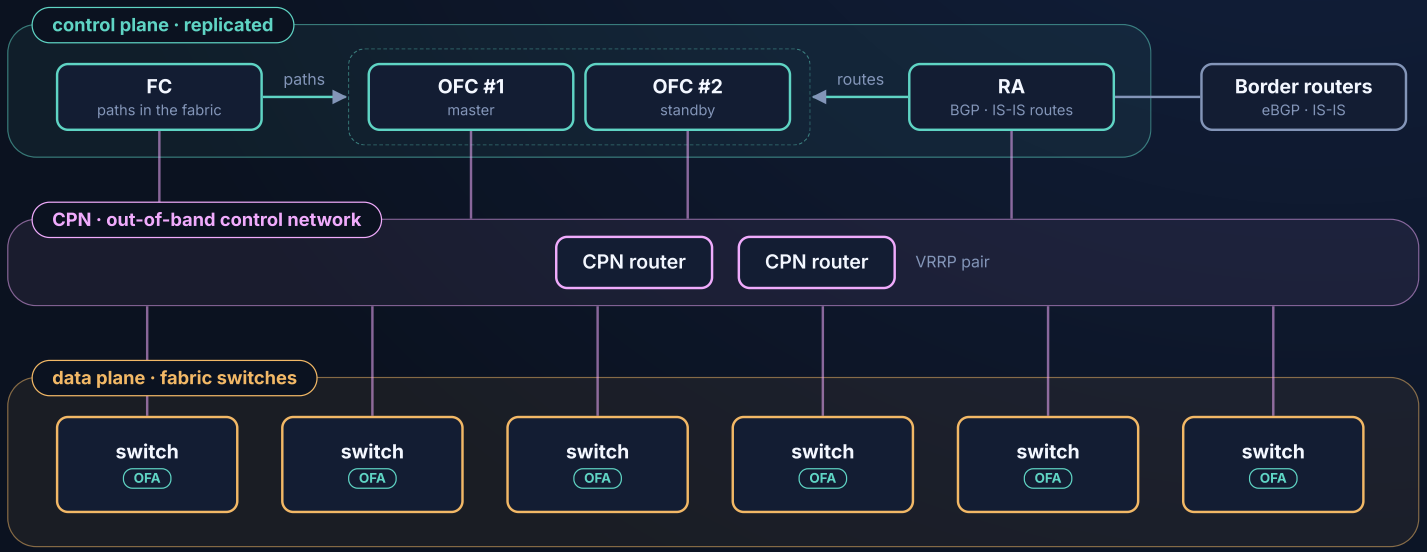
Clusters, plus two WANs that connect them. Hover any box for details

Why two WANs?

	B4 can break the rules	B2 has to be ordinary
Carries	Data center ↔ data center: **elastic** bulk copies	Users ↔ data centers, Internet peering: needs the **highest availability**
Sites	A **few dozen** data centers, so central control works	**Dense**: border, core and edge routers
Hardware	Merchant-silicon switches	Vendor routers, **many protocols**
Routing and TE	Central TE server, tunnels	IS-IS, eBGP, MPLS tunnels via RSVP
Priorities	Google owns **both ends**: priority queues, BwE rate-limits at the edge	Traffic marking, vendor QoS, low-priority tunnels
Utilization	Busy links near **100%**	Typical WANs run at 30–40%, costing **2–3×**

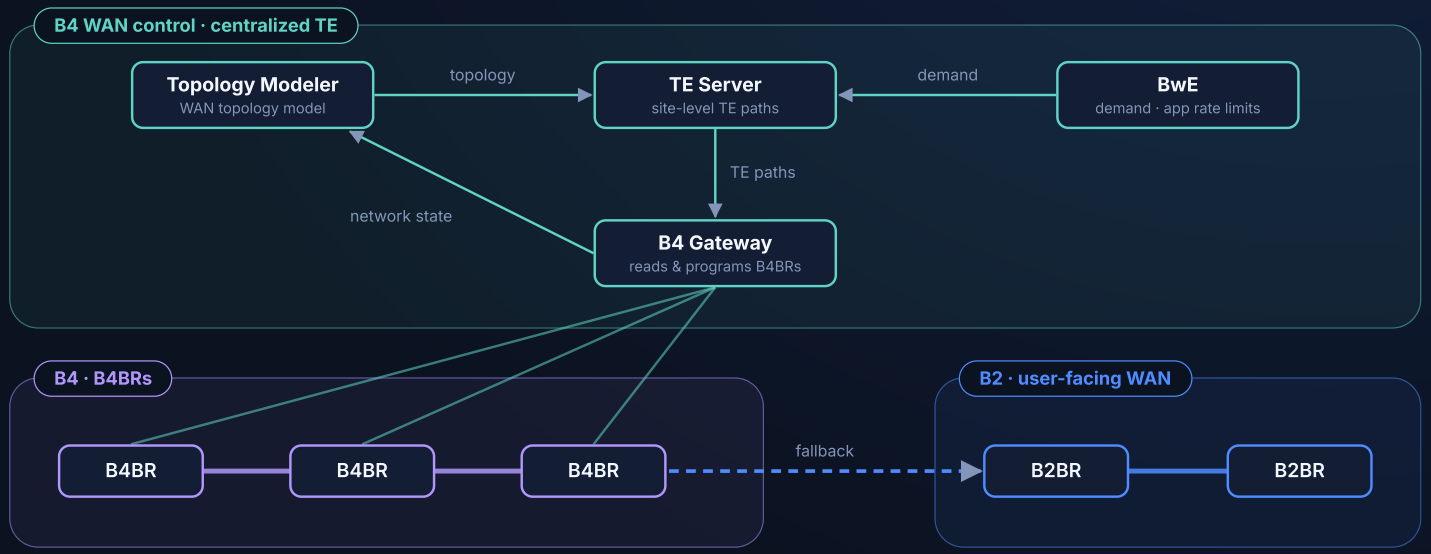
B4 design rationale from Jain et al., "B4: Experience with a Globally-Deployed Software Defined WAN", SIGCOMM 2013 (reference [18])

Control plane: fabrics and B4BRs



The same three components run every cluster fabric and every B4BR. Hover any box for details

Control plane: the B4 WAN



Above the B4BRs sits a global control plane. Hover any box for details

Three planes

Plane	What it does	Examples
Data	Forwards packets	Line cards, links, switch OS
Control	Computes and propagates forwarding state	FC/RA/OFC, TE Server, BGP
Management	Changes the network: MOps	Upgrades, config pushes, drains

> Every failure in the paper is attributed to one of these planes.

Why high availability is hard

Why it's hard

- **Target:** only a few minutes of downtime per month for some traffic classes
- **Scale and heterogeneity:** two vendors on purpose, 2–3 generations of cluster design at once
- **Velocity:** some part of the network is being upgraded every day
- **Device management:** 20 Tb/s boxes, still managed device by device, often through a CLI
- **Tight targets vs. slow upgrades:** a border router upgrade can take 8+ hours, so either drain (costs capacity) or upgrade in place (fragile)

They were already doing it right

- Capacity planning, redundant routers, disjoint fiber and power
- Every controller replicated with failover; WAN replicas spread geographically
- Services register traffic priorities; BwE marks and rate-limits them
- Regression tests (even "Testing on the Toilet" posters), canaries, disaster drills
- Every MOp documented with steps, duration and a risk assessment

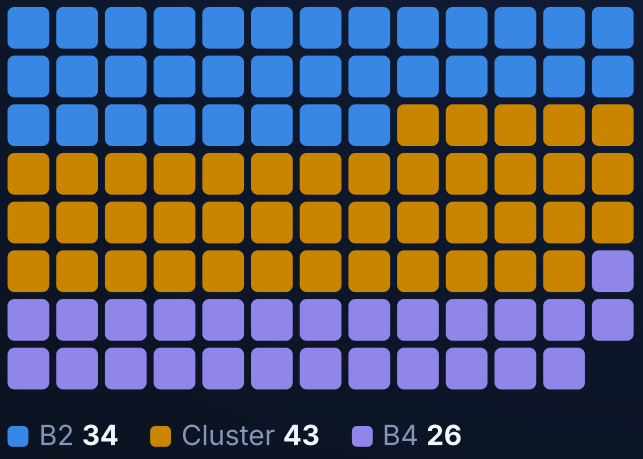
> ...and still: 103 high-impact failures in two years.

What breaks

Dataset: 103 post-mortems

103

high-impact failures, two years



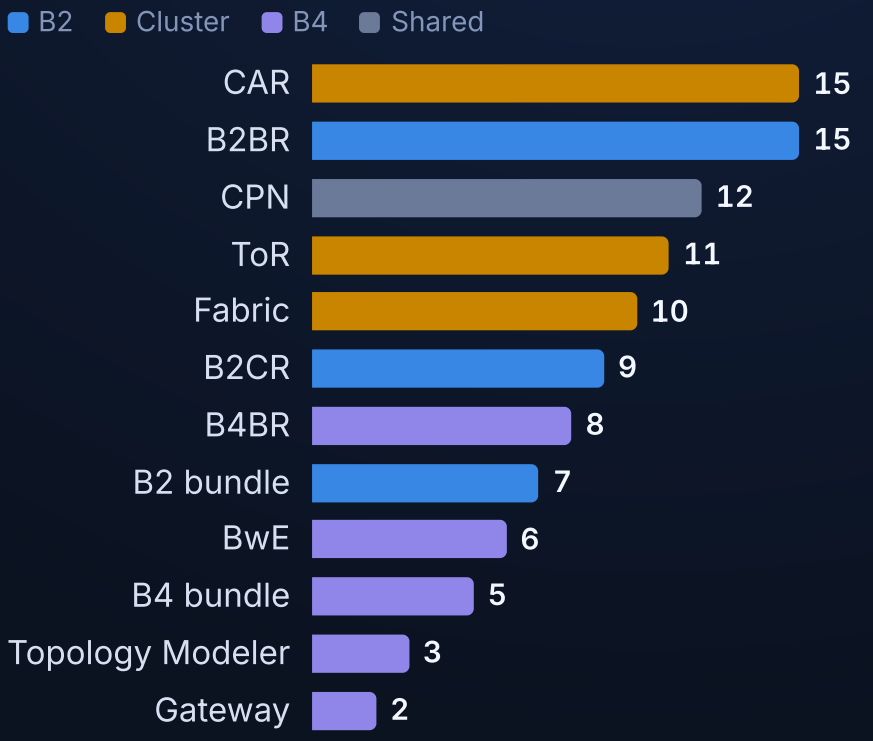
Blameless, peer-reviewed, root causes often reproduced in a lab. Only **unique** failures: a repeat isn't written up again.

No single place to fix

By network and plane

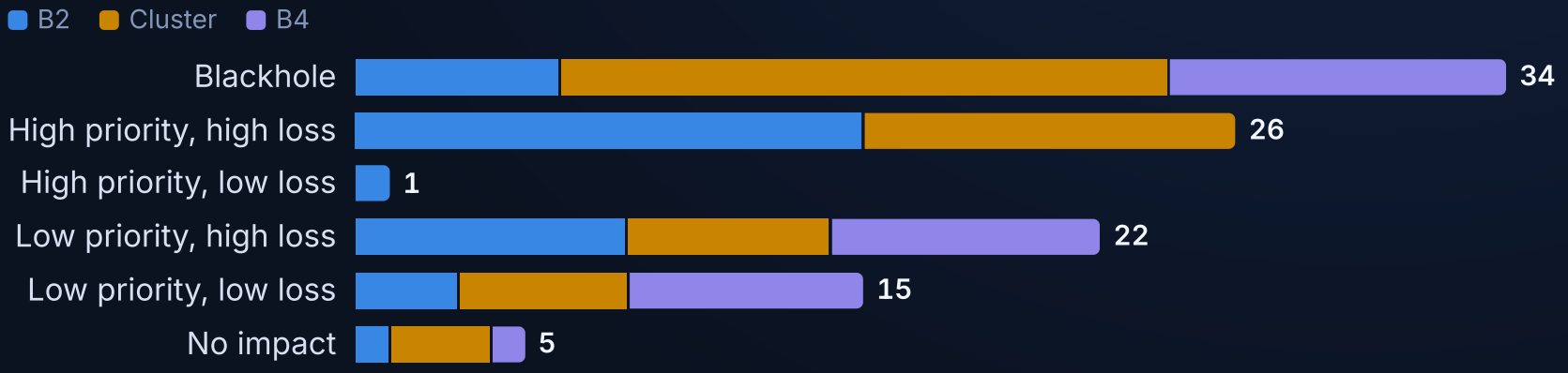
	Control	Data	Mgmt	Total
Cluster	18	10	15	43
B2	9	13	12	34
B4	16	3	7	26

By network element



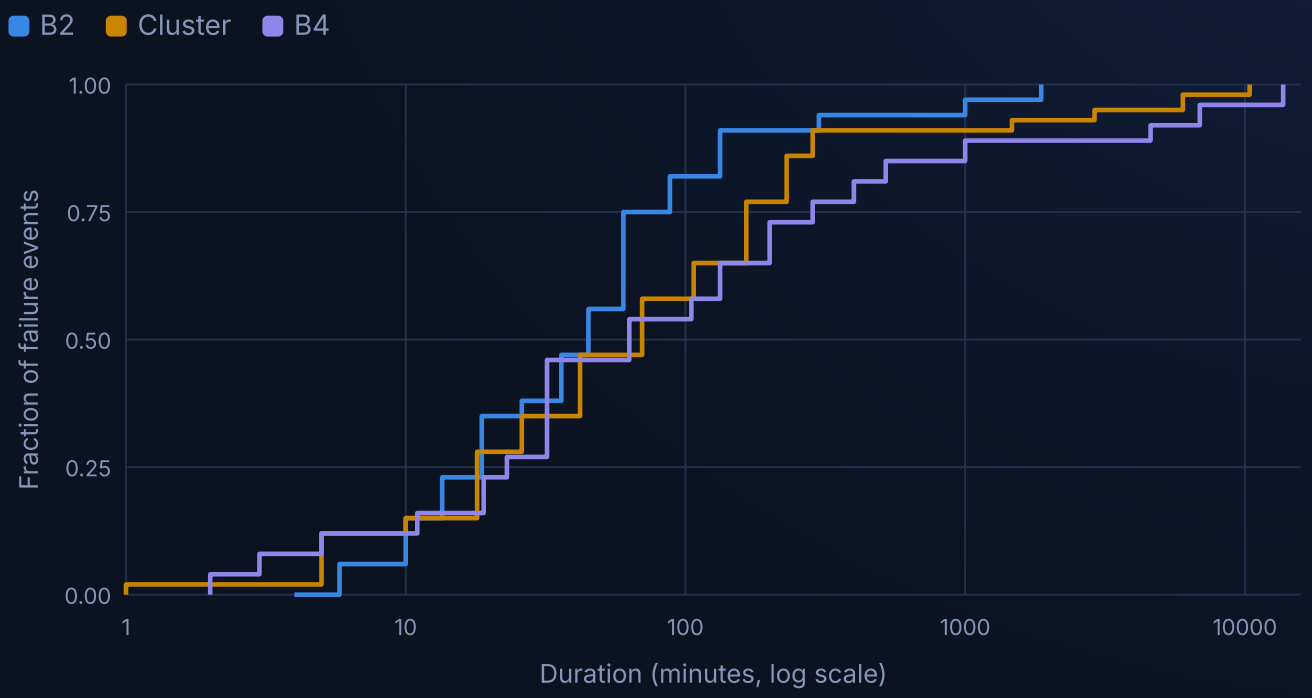
Every network fails in every plane. Only B4 leans one way: 16 of its 26 failures are control-plane.

Failures are big



34 events blackholed traffic, often for whole clusters. B4 carries no high-priority traffic, so it is absent from those rows.

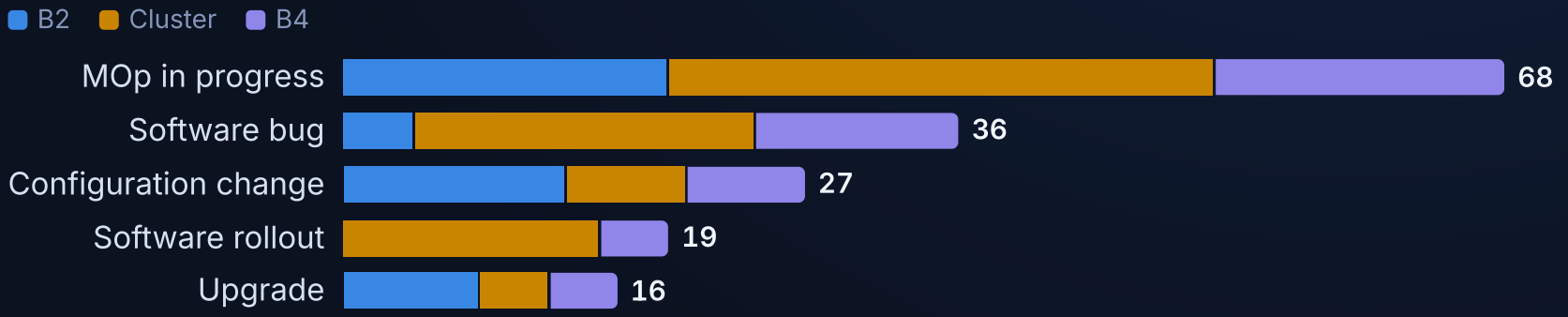
…and slow



Time from discovery to repair. Most take 10–100 minutes; the budget for high-priority traffic is a few *minutes per month*.

Curves drawn by Claude to mimic the shape of the paper's Figure 5, not actual data.

Evolution is in the loop



Categories overlap: upgrades, config changes and rollouts are all kinds of MOp. With 585 MOps a week, this is not a failure rate.

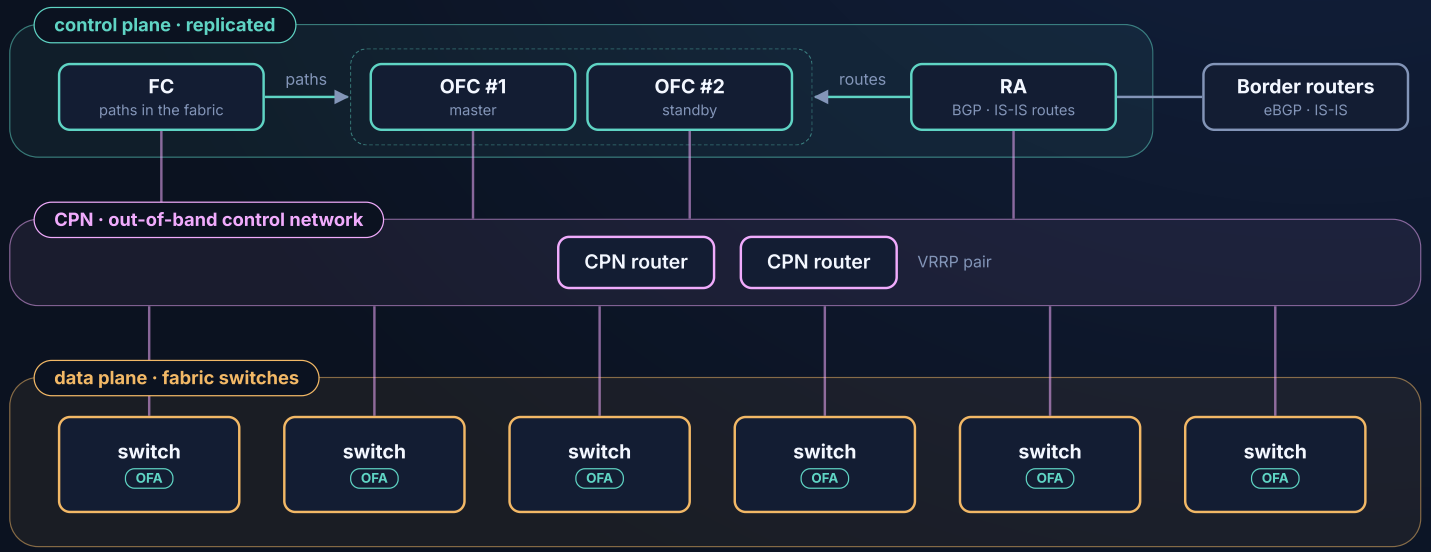
Root causes: 19 categories



One failure can have several root causes. 6+ categories appear in all three networks: systemic, not one bad design.

Failures and their lessons

CPN-1: split brain in a B4BR



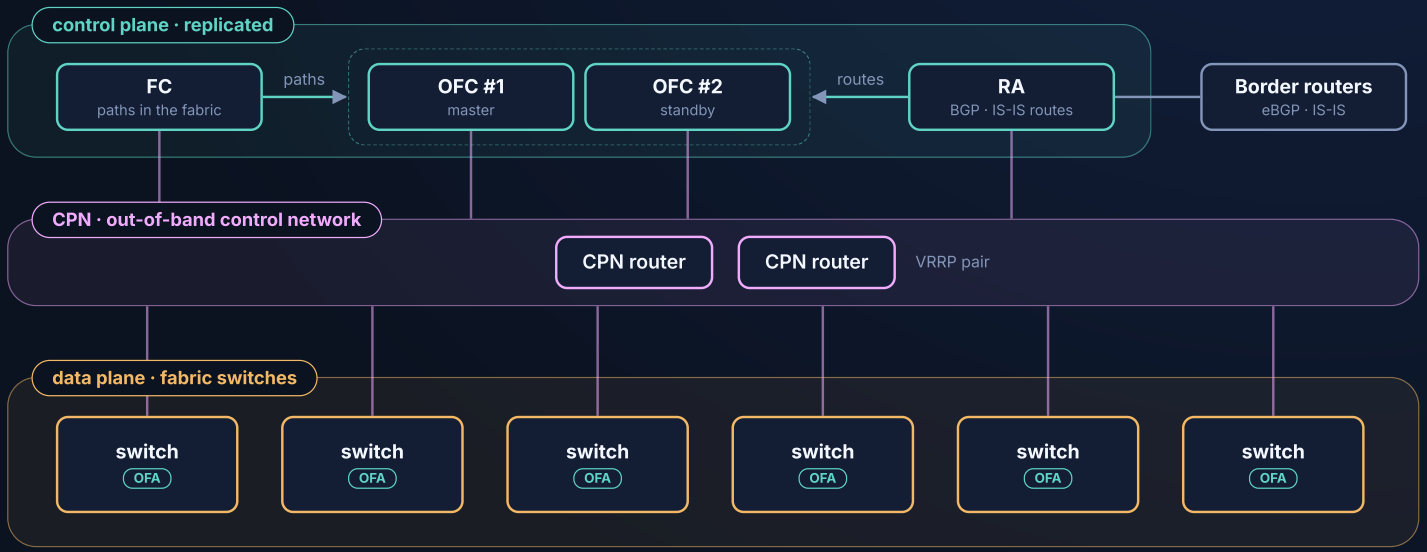
A B4BR runs replicated OFCs that reach their switches over the CPN

PRINCIPLE

Defense in depth

- **Contain the radius:** *k* virtual routers, so a failure costs 1/*k*
- **Fallbacks:** B4 → B2; TE → plain IP routing
- **Big red buttons:** keep the old feature, with a switch back

Consis-1: RA and OFC disagree



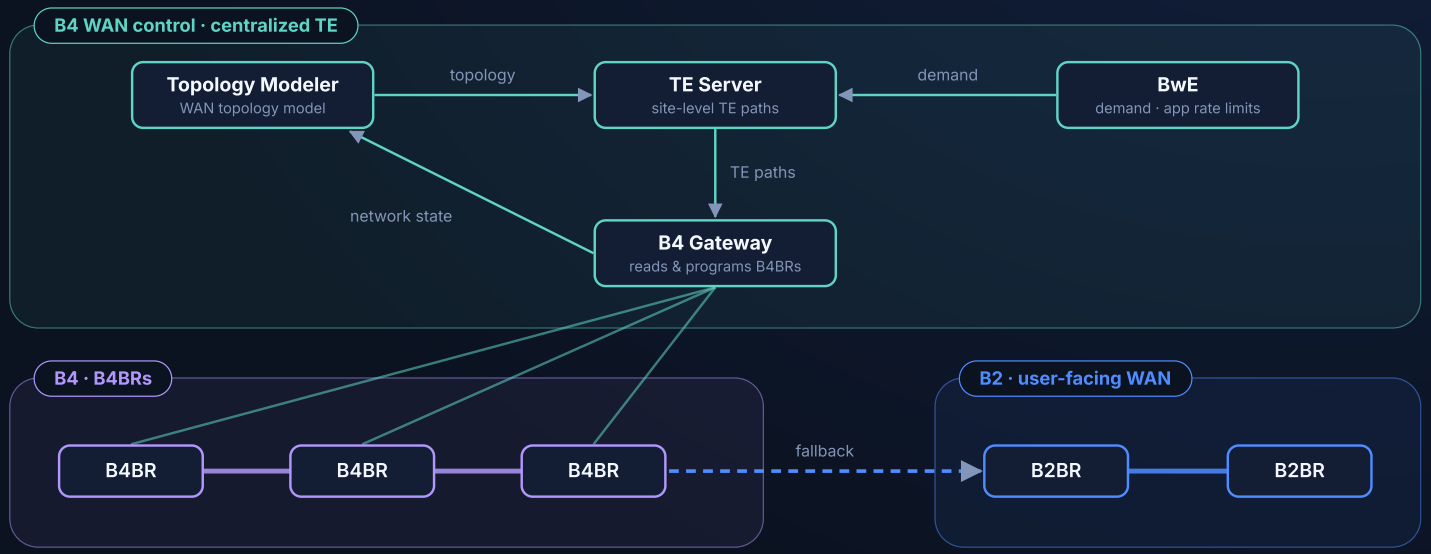
RA turns BGP/IS-IS into routes plus **nexthop groups** (ECMP sets); the OFC stores them and programs switches

PRINCIPLE

Maintain consistency

- **Track equivalence** between control-plane elements
- **Don't advertise** routes before they're installed
- **Check state** after every MOp step
- **Make invariants explicit**, and check them
- **Positive *and* negative** lists: "drain these" and "keep these"

Casc-1: a cascade through B4's controllers



Each B4 controller trusts what the one upstream of it says

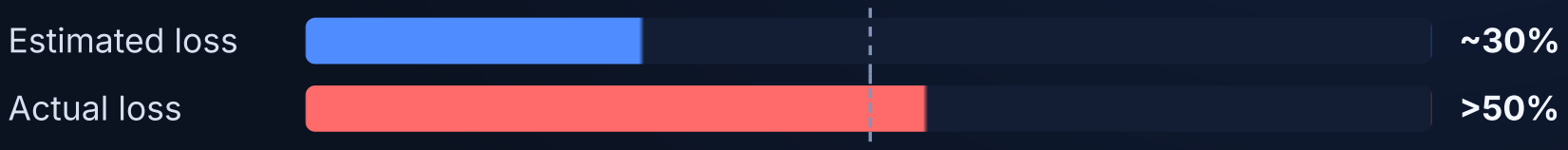
PRINCIPLE

Fail open

- Control plane dies: **keep the last good forwarding state**
- **Don't trust surprising updates:** cross-check before acting

Risk-1: the 30% that was really 50%

MOp: power off 30% of a fabric's switches, restore, repeat (30+ hours)



- Estimated as safe, so services were **not drained**
- **Risk-2:** splitting a B4BR, the estimate was off by 2×, *and* failures elsewhere sent more traffic through it

> **Assess risk continuously:** at every step of the MOp, using the control plane's own model of the fabric

PRINCIPLE

An ounce of prevention

- Assess risk **at every MOp step**, inside the control plane
- When unsure, **overestimate** the risk
- **One management stack, diverse systems** underneath
- **Canaries:** start small, watch for days, then grow

PRINCIPLE

Recover fast

- Root-causing takes **hours**, while a cluster sits drained
- Operators **chase the wrong anomaly**
- Monitoring has gaps, and must not fail with what it watches
- Next: **automated root-cause diagnosis**

Continuously upgrade the network!

- Slowing down means overbuilt capacity, missing features, and change running through rarely-exercised code
- Instead: push features weekly, be able to upgrade daily
- Many small changes are safer and easier to verify than one "big bang"
- Frequent change forces automation, and stops assumptions like "every component runs the same version"

> Touching the network breaks it. So touch it often, and make touching it safe.

For discussion: Is "upgrade continuously" in network a lesson for everyone, or only with Google-scale automation?

Since the paper: Google still breaks

When	What broke	Lesson it echoes
Jun 2019	A config change meant for one region hit several; control plane stopped, congestion slowed the fix	Contain the radius, recover fast
Dec 2020	Quota migration reported zero usage; login DB lost its quota, Gmail and YouTube down ~47 min	Assess risk
Nov 2021	Load balancer config bug hit an unpatched cluster mid-rollout; global 404s ~2h	Canaries
Jun 2025	Policy with blank fields went global in seconds; Service Control crashed, ~7h	Fail open, contain the radius

For discussion: Google published these principles in 2016. Why do they still happen?

PROPOSED PRINCIPLE

Failures are expected

Be calm, go fix it, and learn from it.

> **For discussion:** What's the worst network outage you've been through, or had to fix, or even better created?