Decentralizing Software Defined Networking

The Hidden Complexities of SDN & What We Can Do About Them

Alexander Krentsel, on behalf of colleagues at UC Berkeley & Google...
...Sylvia Ratnasamy, Ali Al-Shabibi, Anees Shaikh, Rob Shakir, & others

Google

Context: Who am I?

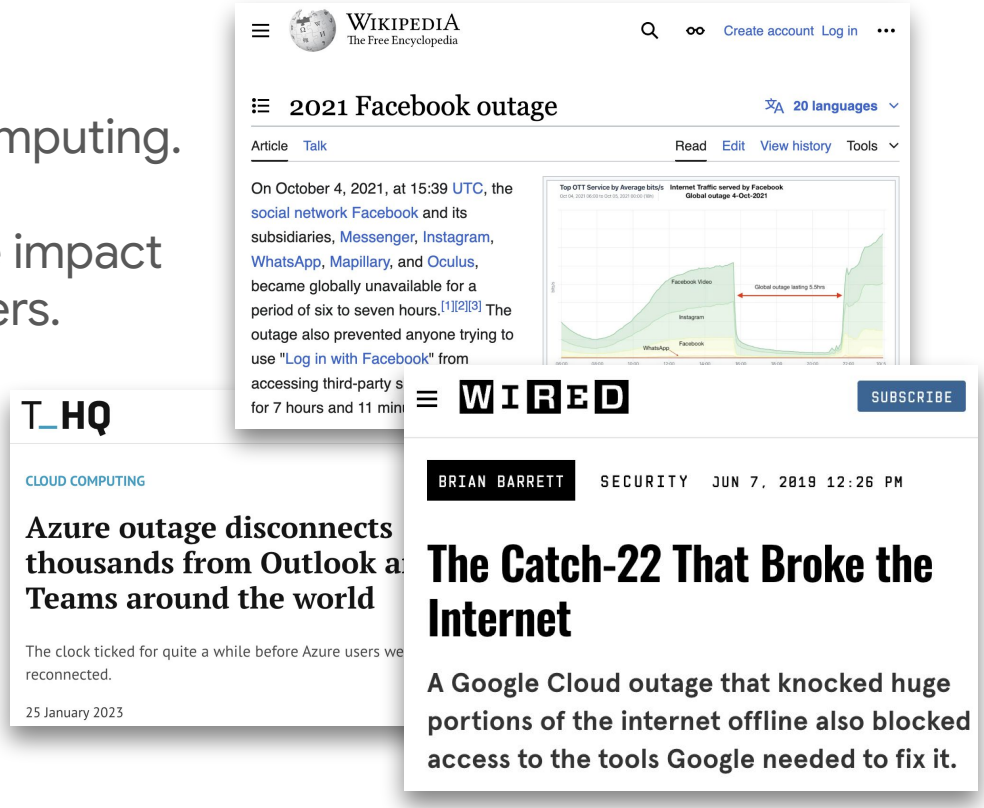
- PhD Student in Networking @ UC Berkeley
  - Advised by Scott Shenker & Sylvia Ratnasamy
  - Year 4 of PhD
- Networked Systems Researcher @ Google
  - 6.5 years in engineering at Google, across YouTube, Cloud, Ads

Research Interests:

- **Wide-Area Network control system architectures**
- Network verification/validation
- Cellular Networks for autonomous vehicle control systems

Problem statement

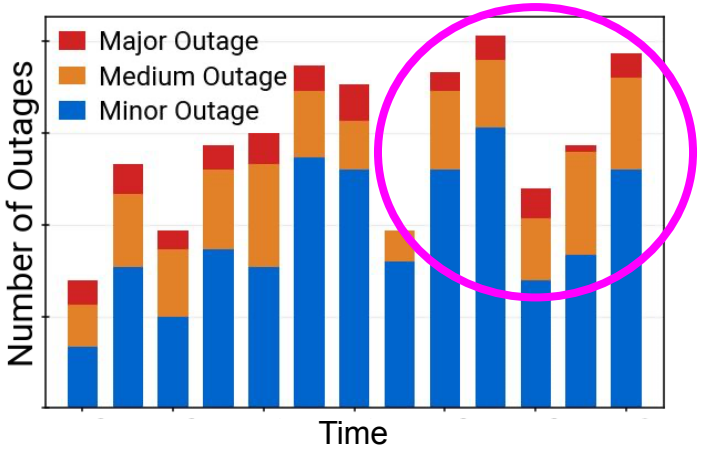
- Global WANs enable planet-scale computing.
- When outages occur, they have large impact on cloud providers and their customers.



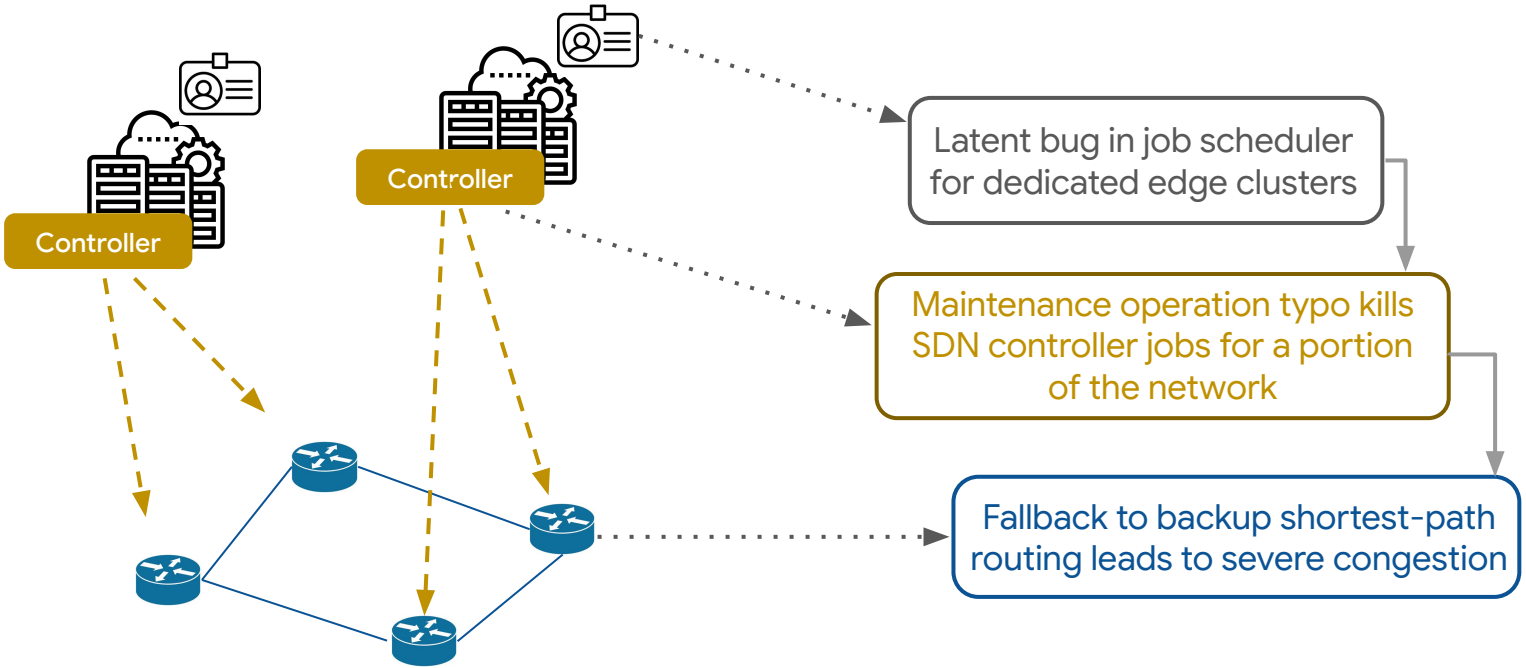
Problem statement

- Outages continue *despite* decades of experience and vast literature of best practices

- Existence of small outages is expected, *large* outages is not
  - complex, cascading root causes; rarely due to loops/deadends, simple link cuts, protocol bugs

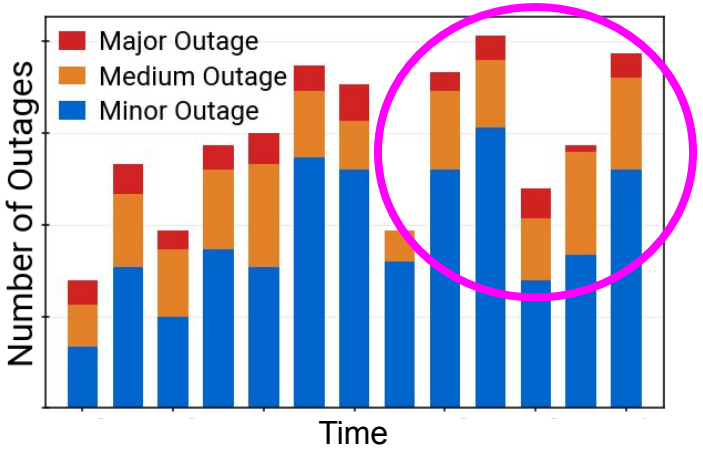


A complex outage example



Problem statement

- Outages continue *despite* decades of experience and vast literature of best practices



- Existence of small outages is expected, *large* outages is not
  - complex, cascading root causes; rarely due to loops/deadends, simple link cuts, protocol bugs

What can we do at *design* time to limit the occurrence of complex failures?

Designing networks to avoid complex failures

- Practitioners apply some techniques today:
  - isolation: shard/partition the network
  - diverse redundancy: minimize single points of failure
  - formal verification: formally guarantee correct behavior

- Orthogonal approach in our work: ***simplification***

Concretely: what components can we take out?

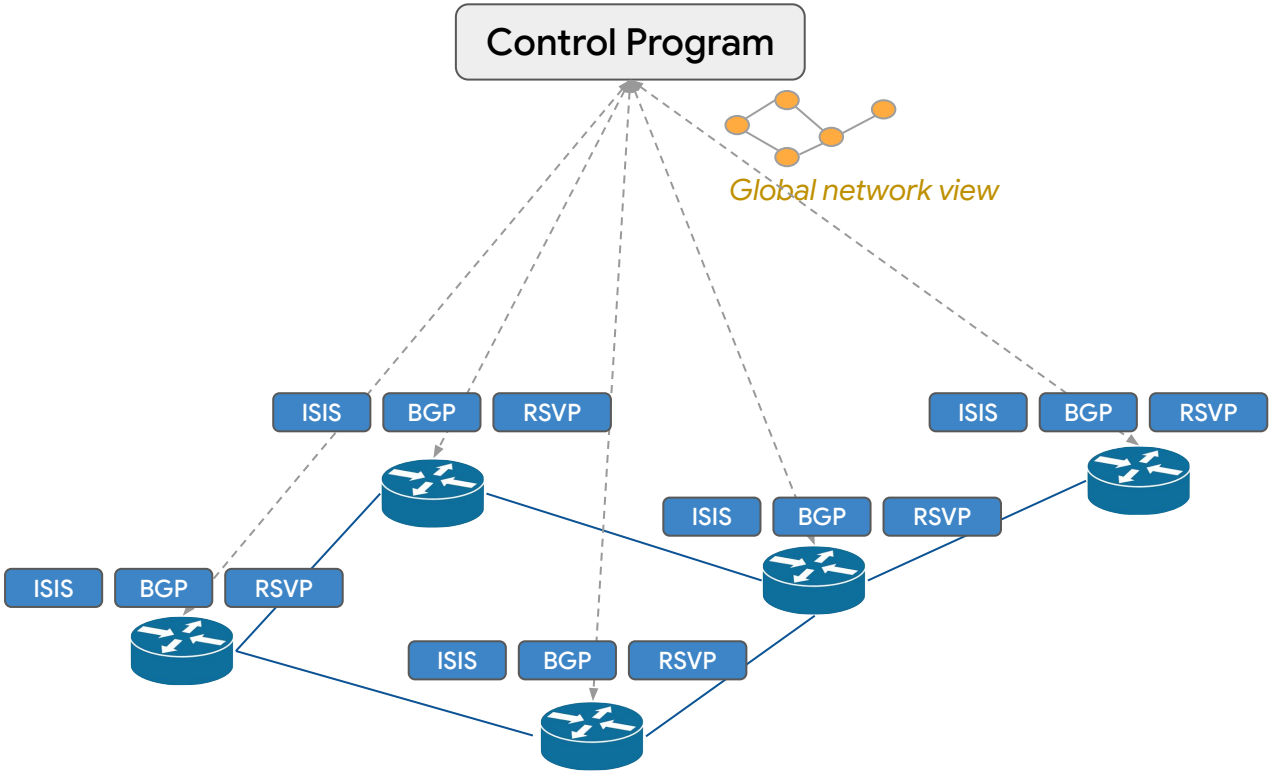
...while maintaining functionality, performance, and cost

Outline

Rest of this talk...

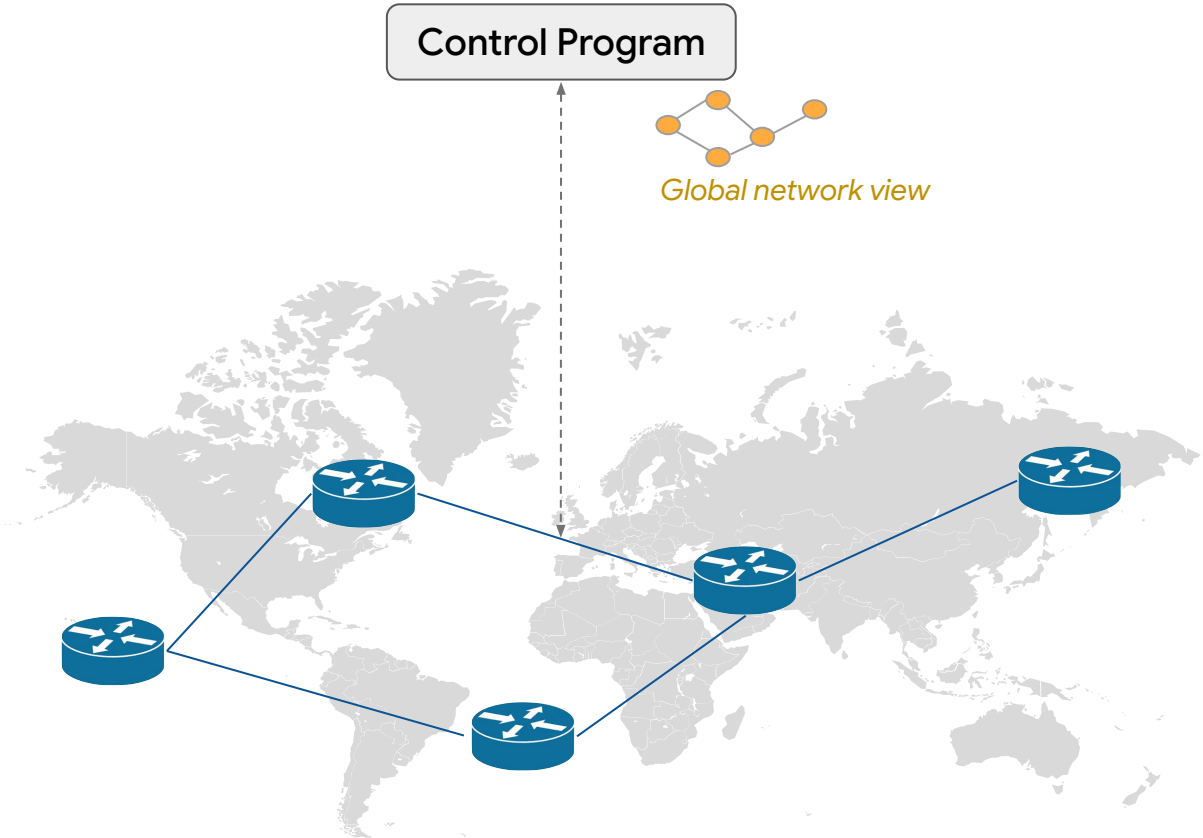
- SDN WAN architectures today
- Opportunities for simplification
- Our approach: **Decentralized SDN (dSDN)**
- dSDN evaluation

SDN WAN architectures today: from protocols to SDN

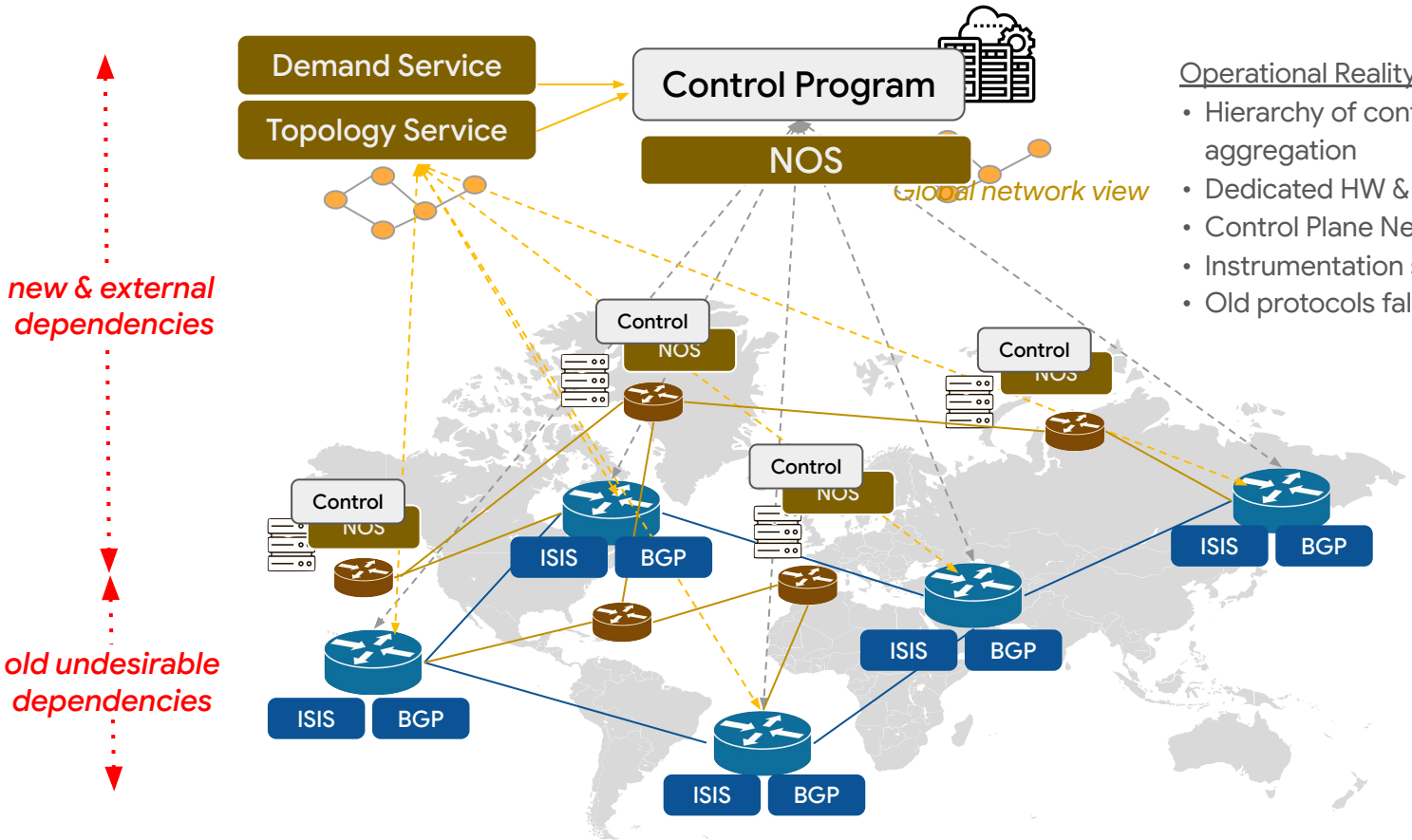


Protocols survivable, but inefficient and difficult to manage, evolve, and scale

SDN WAN architectures today: from protocols to SDN



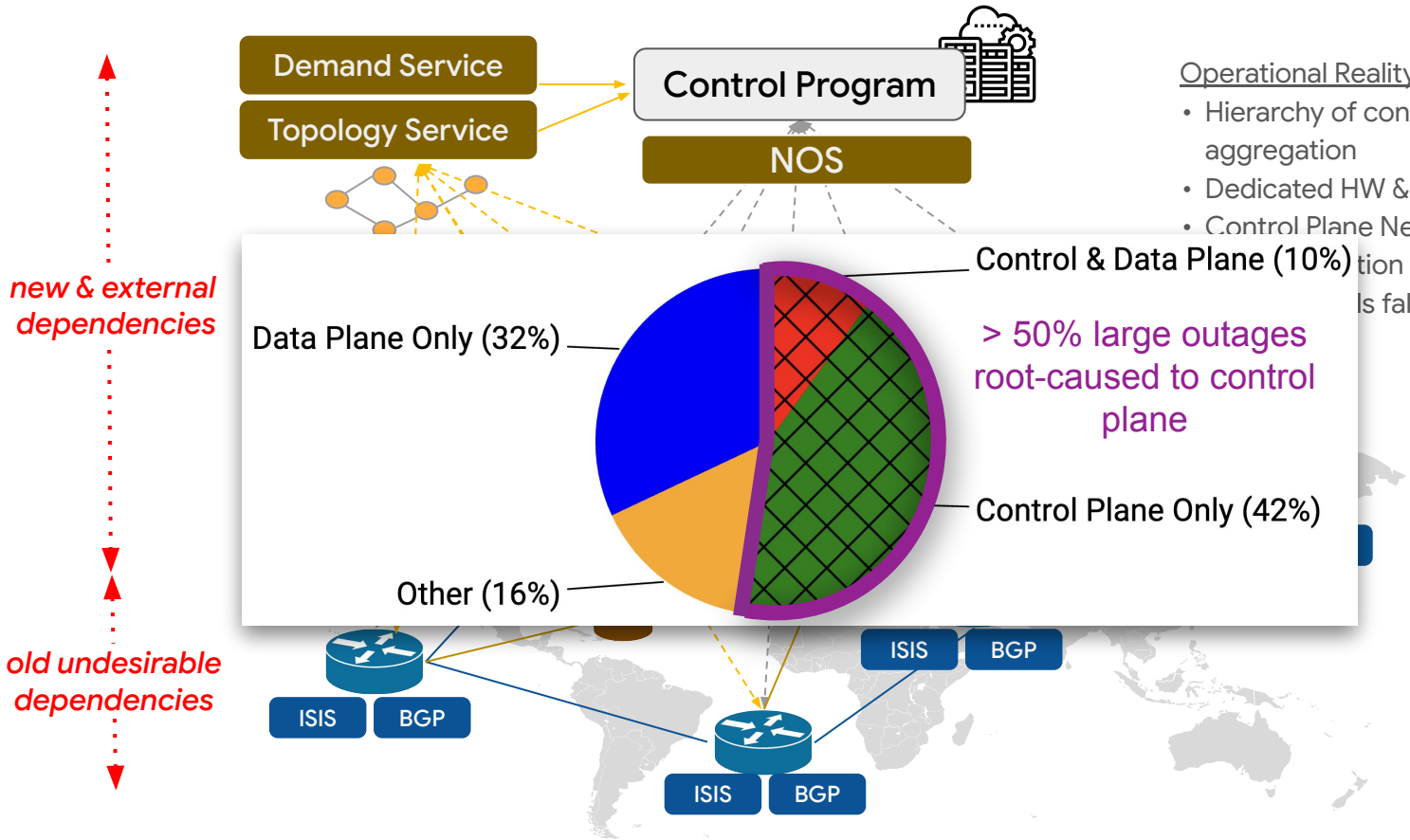
SDN WAN architectures today: reality is complex



Operational Reality

- Hierarchy of control programs to provide aggregation
- Dedicated HW & SW to run them
- Control Plane Network to reach routers
- Instrumentation services to collect inputs
- Old protocols fallback in case of split-brain

SDN WAN architectures today: reality is complex



Operational Reality

- Hierarchy of control programs to provide aggregation
- Dedicated HW & SW to run them
- Control Plane Network to reach routers
- tion services to collect inputs
- ls fallback in case of split-brain

Achieving simplification

We tackle *complex* outages head-on by aiming for *simplification*.

Why not go back to distributed protocols? Because shift to SDN gave us benefits that we've come to rely on…

1. **Operator-defined** code, which enabled innovation
2. **Optimized** computations (TE) on global view of network
3. **Simplicity** of "consensus-free" path selection

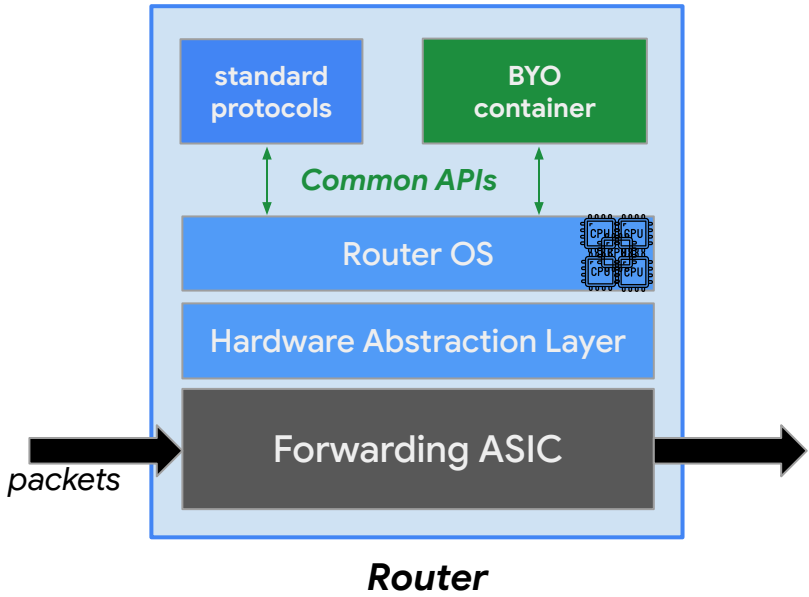
Goal: Simplify while *retaining SDN's benefits*.

Opportunity: operator applications on-router

1. Router vendors now support running 3rd-party containerized code directly "on the box"
   → operators can run custom control code on router CPU

2. New generation of standardized control/config APIs (*e.g.*, gRIBI, gNMI, OF/P4)
   → control code is uniform across vendors

3. Expanded on-router CPU resources
   → from single-core to multi-core multi-GHz CPUs



Operator-defined control applications can run on-box.

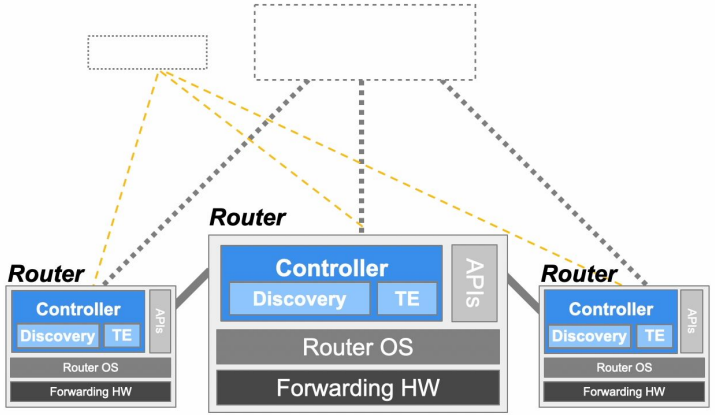
Essential idea: decentralizing SDN

Decentralize SDN by moving *operator-written controller code onto the router*

➡ **...<u>replicate</u> controller on each router**

Concretely, every router runs a ***dSDN controller*** that

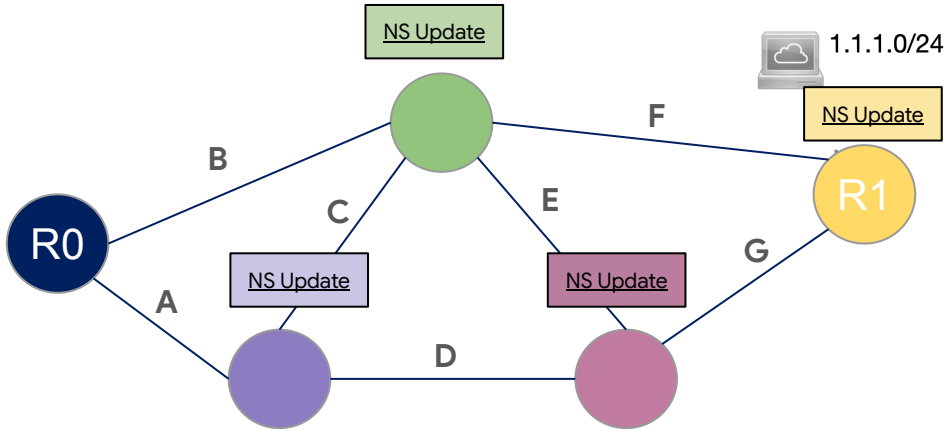
1. **floods** its local node state; learns global network view
2. locally **computes *all* paths** (using a traffic eng. algorithm)
3. "programs" locally originating paths as ***source-routes***



Walkthrough →

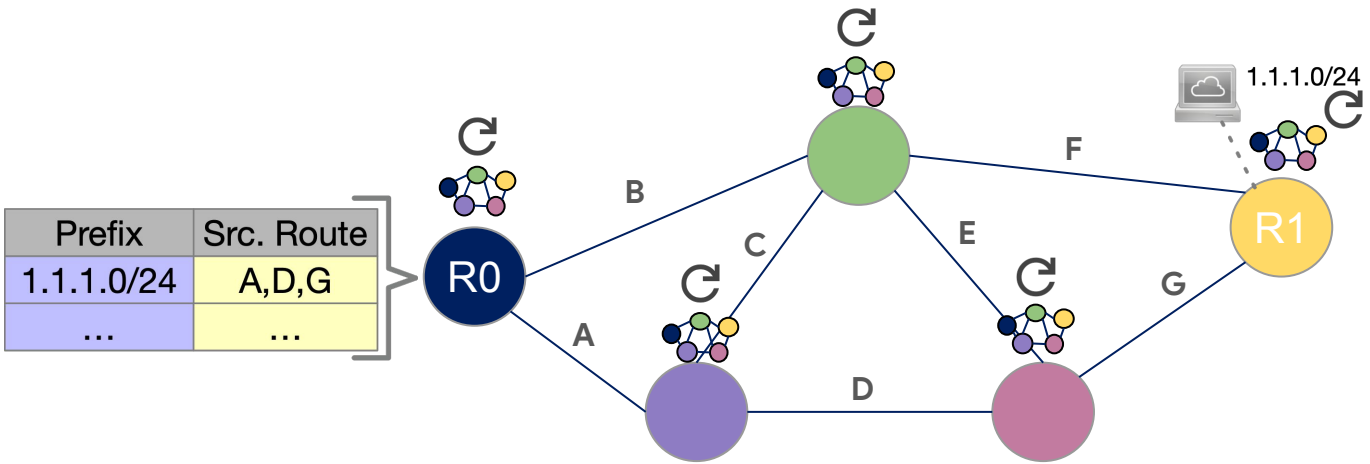
(1) Building a global view

Node State Update
NodeID: R0
Prefixes: 1.1.2.0/24
Neighbors: ...
Link Capacity: (A:10G), (B:20G)
Demand Vector: <...>



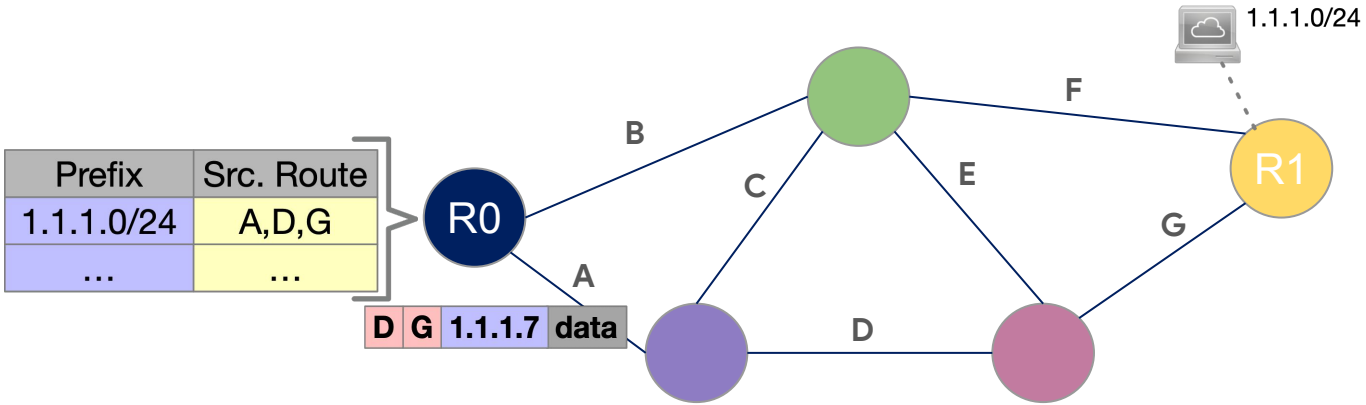
- Each node floods “Node State” (NS) updates to all other nodes
  - Contains *local* topology and demand info
  - ⇒ each node discovers a global network view

(2) Each node computes *all* paths



- Each node…
  - …locally runs operator's *global* TE algorithm
  - …encodes the set of paths it originates as strict source routes
  - …programs them into its forwarding table

(3) Forwarding follows source route

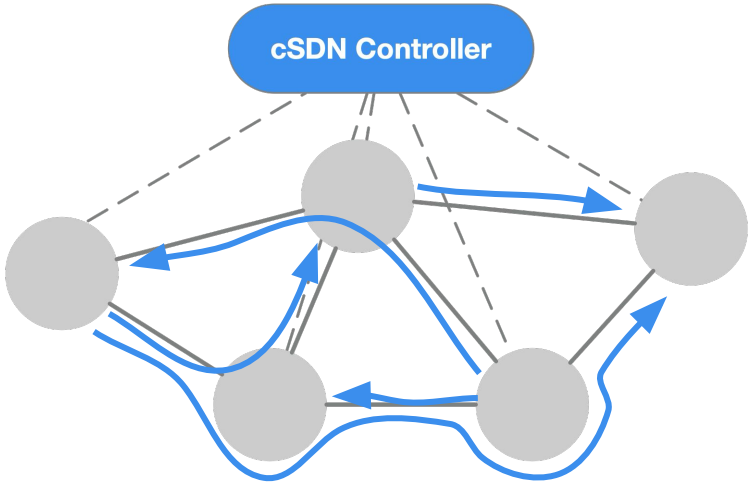


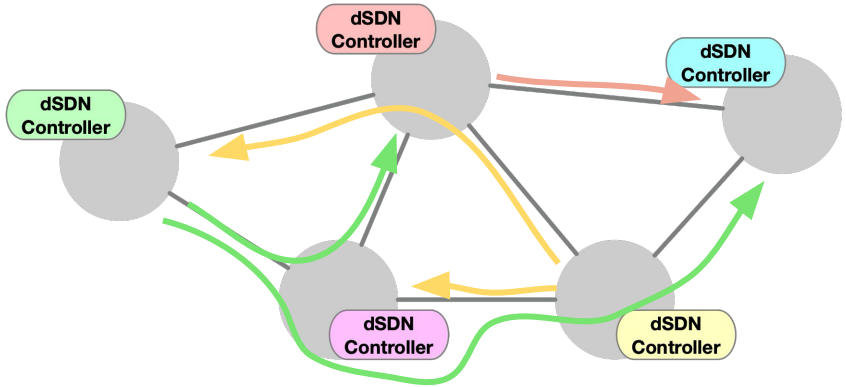
- Full source route inserted into packet at ingress node
  - path followed by all transit nodes
  - ⇒ **“consensus-free”** pathing; no coordination across routers to program

Source routing enables *simple* decentralization

Strict source routing ⇒ ingress router dictates path *authoritatively*

- requires no consensus across transit routers to program path

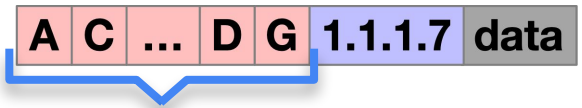




Source routing enables *simple* decentralization

Strict source routing ⇒ must enumerate path in header

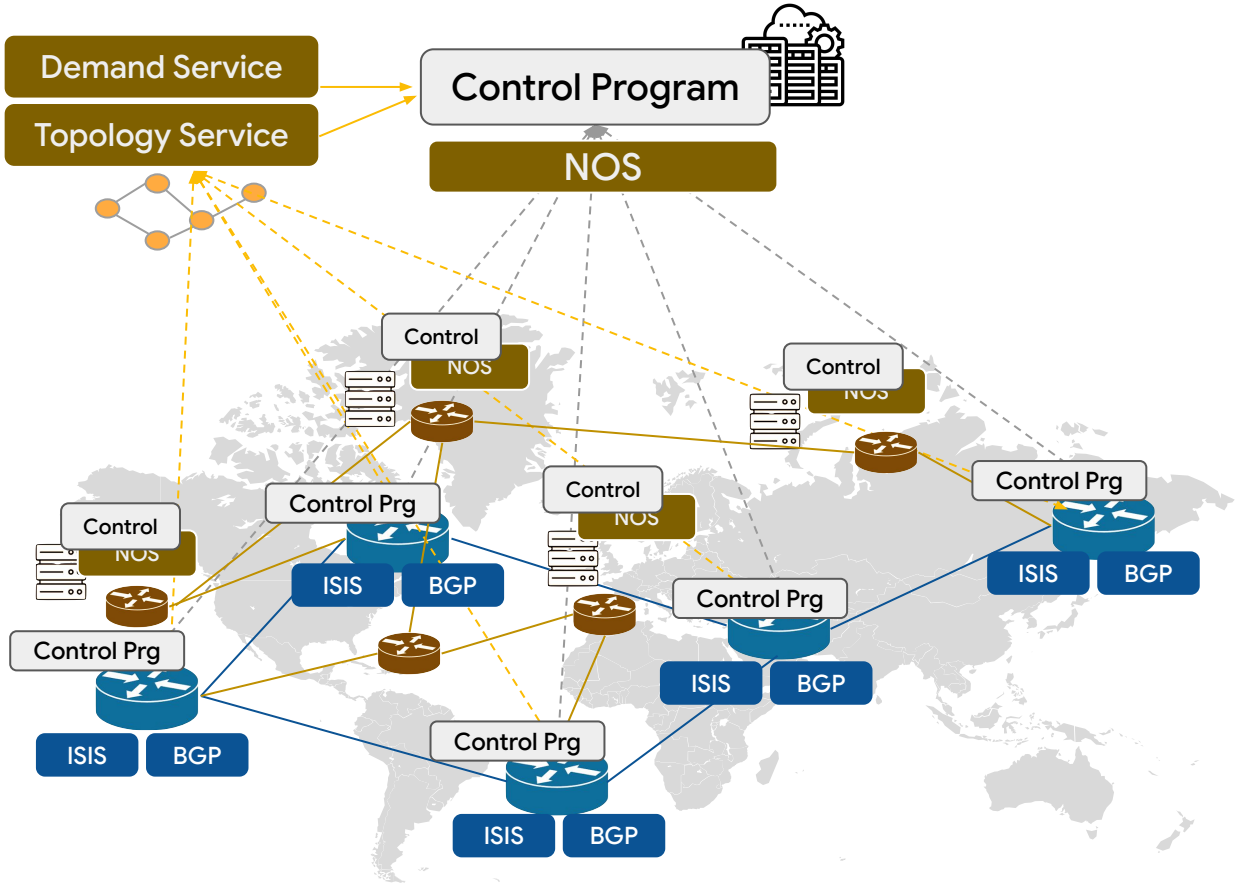
- Historically infeasible due to length of WAN paths



Enabled by hardware advancements and **novel encoding technique**

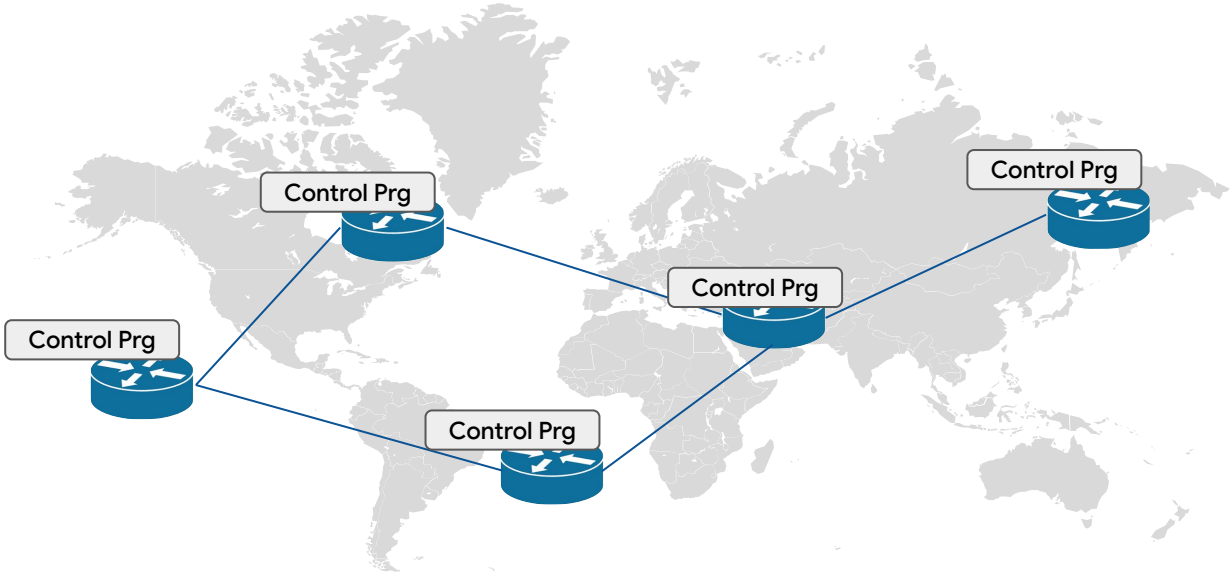
- See paper for details

dSDN cuts out a large fraction of infrastructure



dSDN cuts out a large fraction of infrastructure

- Smaller surface area for bugs
- Greatly decreased number of components on critical path for routing



dSDN achieves the benefits of SDN *and* decentralization

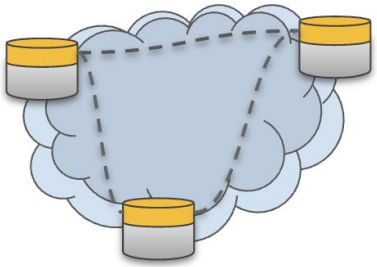
Original SDN Benefits

- **Operator-defined code**, which enabled innovation
  - ✅ *running our own containers on the router*
- **Optimized** computations (TE) on global view of network
  - ✅ *new APIs + simple dissemination ⟶ global view*
- **Simplicity** of “consensus-free” path selection
  - ✅ *source-routing; “ingress” router authoritatively decides path*

Decentralization Benefits

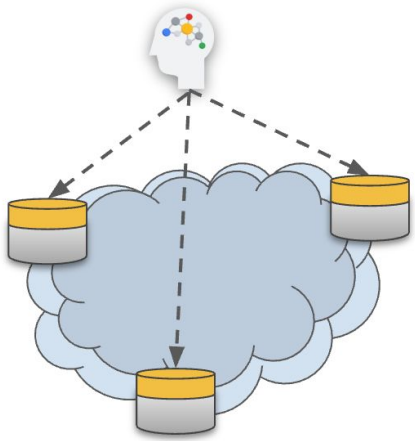
- Drastically **fewer** external **dependencies**
  - ✅ *control plane running in-band*
- Distributed **survivability**
  - ✅ *no central point of failure*

dSDN achieves the benefits of SDN *and* decentralization



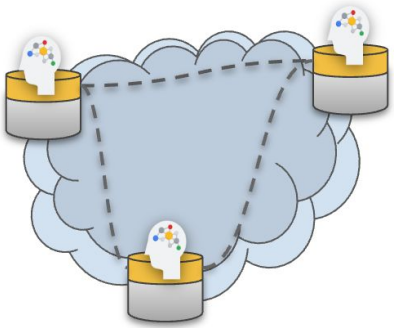
Traditional Protocols

Path selection logic implemented by vendor protocols that run at routers



Centralized SDN

Path selection logic implemented by operator code that runs on external server(s) and programs routers



Decentralized SDN

Path selection logic implemented by operator code that runs ***on routers***; source routing instead of path programming

Simplifying control infrastructure *improves* performance

Primary Goal: fewer outages, by cutting complexity from the WAN architecture

Removed	Added
(1) Central controller jobs (2) Regional controller jobs (3) Dedicated server hardware (4) Control plane network (5) Instrumentation services (6) Traditional protocols	On-router containers

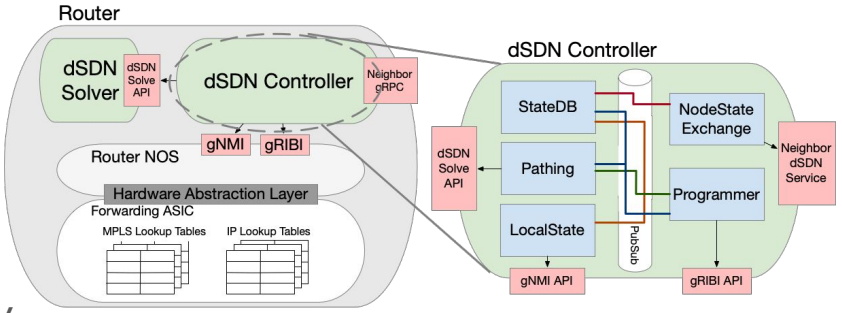
Have we lost performance in the process?

- On the contrary, we find simplification results in *better* performance

Evaluation Performance: dSDN running on real hardware

Methodology*

- ✦ Built production-grade dSDN implementation
  - Profiled on production-grade testlab routers
  - Fed live B4 topology and demand data
- ✦ Profiled cSDN performance in production
- ✦ Replayed production failure events in high-fidelity simulator of cSDN and dSDN



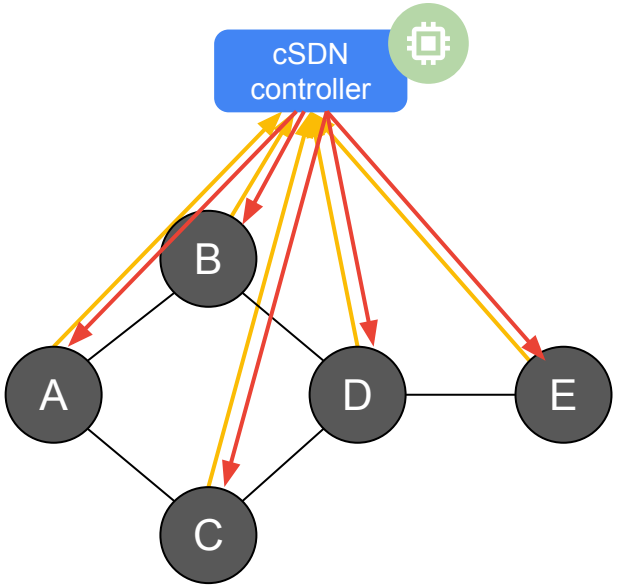
dSDN Implementation Architecture

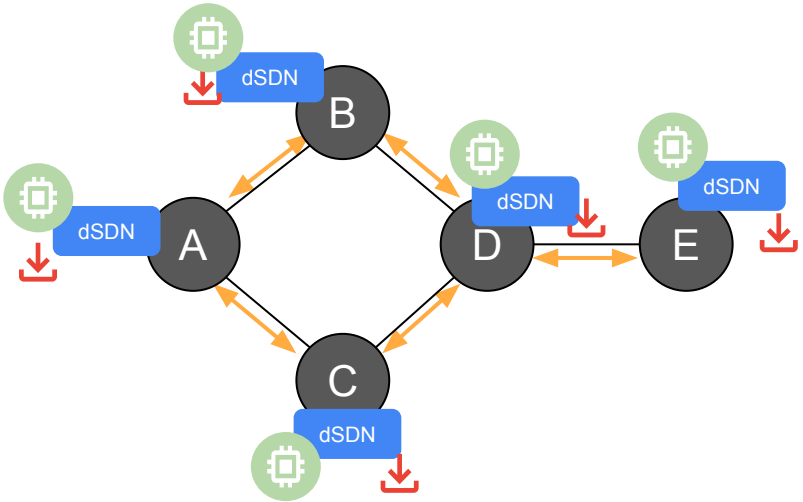
Evaluated (1) *convergence time* and (2) *convergence impact*

*See paper for more details

A closer look: convergence

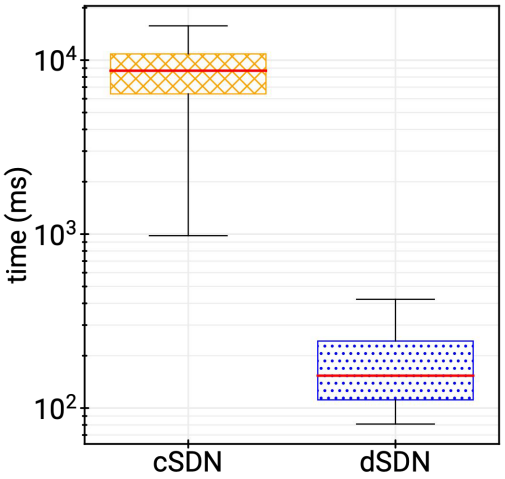
Three components: **(1) propagation**, **(2) computation**, **(3) programming**



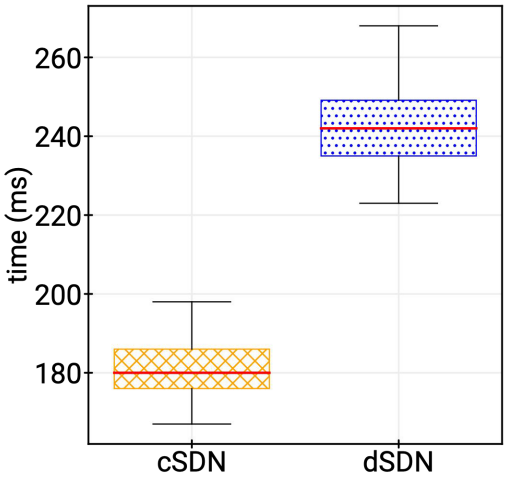


Convergence time: 120x-150x faster overall

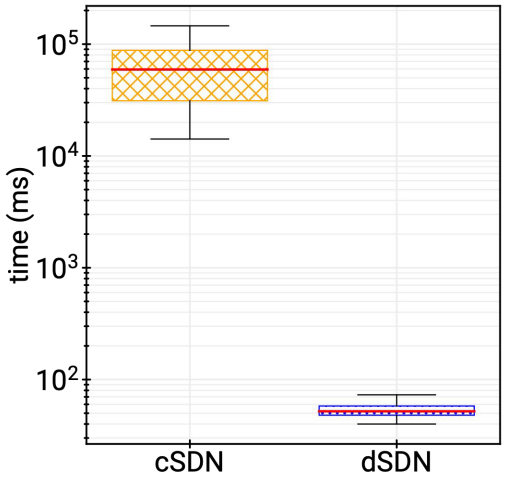
(Note differing Y axes)



Propagation



Computation



Programming

Network convergence time is the sum of these three components...

dSDN's faster propagation and programming far outweigh slightly slower computation

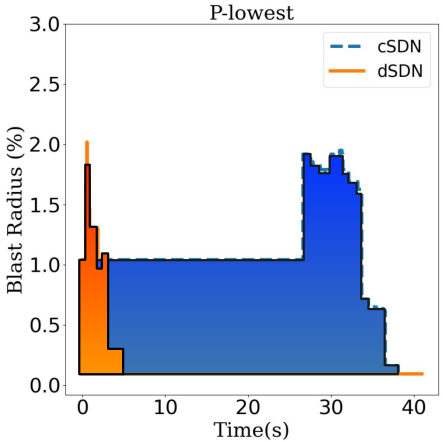
Convergence impact: significantly lower for all priorities

Convergence impact comprises both amount of time and number of flows affected

- **Bad seconds** metric defined in the paper, intuitively covers both

	cSDN	dSDN
Highest Priority	146.9	2.23
Lowest Priority	1122.9	57.3

Bad Seconds at the 98th Percentile



Impact during convergence for a single event for lowest-priority traffic

See paper for more details

dSDN sees big win in impact due to shorter convergence time

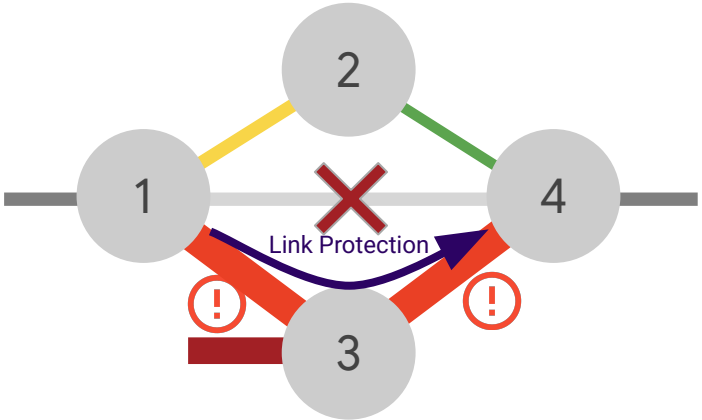
Enabling new on-router capabilities: Smart Fast ReRoute (FRR)

dSDN provides *platform* for additional on-router capabilities.

Fast ReRoute (FRR): mechanism that establishes backup tunnels *per-link* when path is established.

⇒ If link goes down, traffic temporarily uses bypass...

⇒ ...which can lead to congestion.



Enabling new on-router capabilities: Smart Fast ReRoute (FRR)

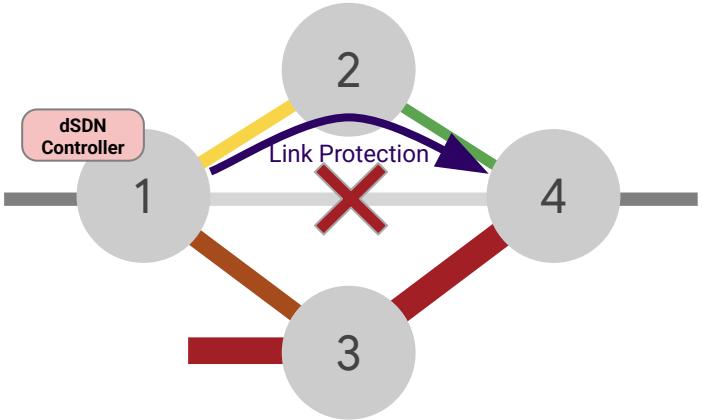
dSDN provides *platform* for additional on-router capabilities.

Fast ReRoute (FRR): mechanism that establishes backup tunnels *per-link* when path is established.

⇒ If link goes down, traffic temporarily uses bypass...

⇒ ...which can lead to congestion.

dSDN's on-router controller is utilization aware, can move bypass frequently and proactively.



Enabling new on-router capabilities: Smart Fast ReRoute (FRR)

dSDN provides *platform* for additional on-router capabilities.

Fast ReRoute (FRR): mechanism that establishes backup tunnels *per-link* when path is established.

⇒ If link goes down, traffic temporarily uses bypass...

⇒ ...which can lead to congestion.

dSDN's on-router controller is utilization aware, can move bypass frequently and proactively.

Recreated top 6 bypass congestion events over 2 week period, explored different dSDN-enabled algorithms...

#	FRR	Capacity Aware	k Shortest Paths	k Capacity Aware Paths
1	5.51% (1)	6.24% (1)	2.53% (1)	0.0% (1.24)
2	2.09% (1)	0.24% (1.06)	1.16 % (1)	0.0% (1.09)
3	1.25% (1)	0.0% (1.02)	0.08% (1)	0.0% (1.02)
4	1.58% (1)	0.25% (1.02)	0.11% (1)	0.0% (1.08)
5	1.54% (1)	0.03% (1.06)	1.34% (1)	0.0% (1.08)
6	1.20% (1)	0.13% (1.16)	1.09% (1)	0.0% (1.19)

Blast radius and (median end-to-end latency inflation) for high priority traffic

dSDN-enabled Smart FRR eliminated congestion during re-convergence in all 6 scenarios

Further details

See paper for discussion on additional questions...

- Does dSDN scale?
  Yes, paper discusses forward-looking projections, TE optimizations
- Does dSDN handle high churn in network state?
  Yes, paper evaluates up to 20x production churn rate
- Are there other benefits to having control loops on the router?
  Yes, paper demonstrates additional applications of on-router control
- Can source routing work in very large networks?
  Yes, paper presents supporting techniques

dSDN as a platform: building beyond routing

We've found dSDN to be a platform for a variety of applications.

- We presented (1) full routing, and (2) smart FRR for bypass routing...
- Many possibilities for on-router apps: auto-repair agent? telemetry validation?

More exciting on-going work applying dSDN to more routing applications...

Bunch of technical support developed, some open sourced:

- **Containerz** — vendor-neutral way to orchestrate containers on devices; deploy, start, stop, list, and log.
- **gNMI** – vendor-neutral telemetry reading

Decentralized SDN: a new point in the design space

- ✦ We've spent...
  - ➡ 30 years trying to fix and evolve distributed protocols
  - ➡ 15 years trying to make SDN-based networks more reliable
- ✦ We present **dSDN**, a new point in the design space
  - ➡ achieves the best of both by decentralizing the SDN controller in a way that...
    - ★ maintains benefits of SDN
    - ★ significantly ***simplifies*** the control plane infrastructure
    - ★ improves convergence performance

Paper: bit.ly/decentralized-sdn

Research beyond dSDN

Context: Who am I?

- PhD Student in Networking @ UC Berkeley
  - Advised by Scott Shenker & Sylvia Ratnasamy
  - Year 4 of PhD
- Networked Systems Researcher @ Google
  - 6.5 years in engineering at Google, across YouTube, Cloud, Ads

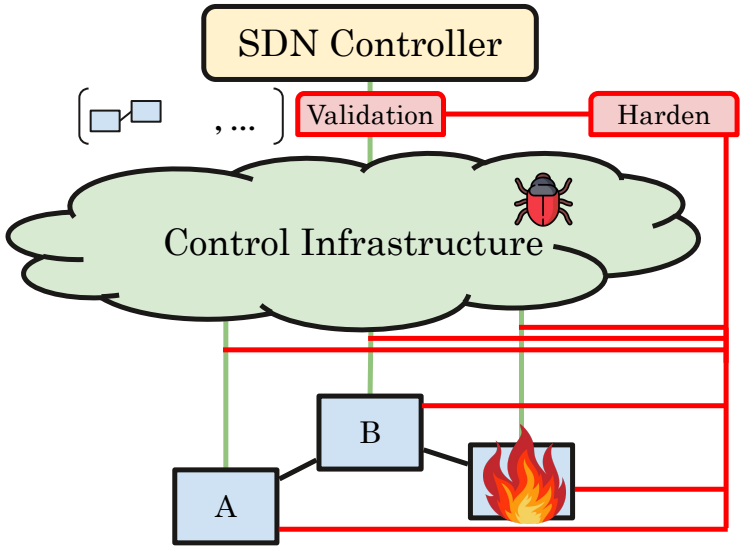
Research Interests:

- **Wide-Area Network control system architectures**
- Network verification/validation
- Cellular Networks for autonomous vehicle control systems

1

Research beyond dSDN: Input Validation

- → Many network outages caused by *incorrect* inputs to control systems
- → Testing/verification only ensure correct behavior for a given input...
  - → ...**assumes** inputs truly reflect reality
  - → Unbounded bad behavior if inputs are wrong



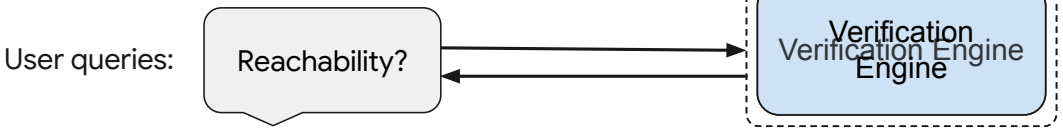
Key idea: cross-check controller inputs with low-level network telemetry, ensuring particular invariants hold between them.

Paper: bit.ly/input-validation

Research beyond dSDN: Model-Free Verification

- → Formal verification tools promise to **guarantee** network safety...
- → ...but limited real-world adoption because requires control plane *model*
  - → Models are incomplete, out-of-date, hard to maintain

Key idea: emulate control plane behavior in a full-fidelity emulator, then formally verify the resulting data plane.



Paper: bit.ly/model-free-verif

- Exciting new challenges as our networks continue to scale
- Academia has a lot to learn from network practitioners
- Looking forward to more engagement with network operators, engineers, and architects

Alexander Krentsel
akrentsel@berkeley.edu

Questions?

Google