CSIG: Fine-Grained Congestion Signals for ML Workloads

ML Networking Tutorial
SIGCOMM 2025

Brad Karp (Google LLC, UCL CS)

CC and Load Balancing for ML Workloads Need Enhanced Congestion Signals

Performance of tightly synchronous training gated by collective tail flow completion time:

- Congestion control (CC) must ramp up quickly yet safely to fill abruptly released fabric capacity
- CC must react to precise extent of congestion at bottleneck hop
- Multi-path load balancing (LB) essential, guided by congestion feedback

CC and Load Balancing for ML Workloads Need Enhanced Congestion Signals

Performance of tightly synchronous training gated by collective tail flow completion time:

- Congestion control (CC) must ramp up quickly yet safely to fill abruptly released fabric capacity
- CC must react to precise extent of congestion at bottleneck hop
- Multi-path load balancing (LB) essential, guided by congestion feedback

These CC and LB needs not fully served by status-quo congestion feedback signals (e.g., drops, RTT path delay, ECN marks):

- Sender can learn of congestion, but not magnitude of path's spare capacity, except by slow, conservative search (else risks overshooting)
- In delay-based CC, path RTT includes all queues on forward path, not just bottleneck hop
- Single-bit ECN is low-resolution feedback on path's congestion state; does not indicate degree of congestion for LB path selection beyond "congested"/"uncongested"

Telemetry for ML Workloads Needs Enhanced Congestion Signals

Scenario: deploy ML training job, would like to understand its congestion behavior in fabric to refine model's design (e.g., to use more of capacity, or avoid congesting fabric too severely)

- Synchronized transmissions in collective communication create intense, brief traffic bursts, and congestion episodes that last under 1 ms
  - Today's typical switch telemetry measures over 1 s or longer
  - Cannot even **observe** workloads' fabric congestion behavior if cannot measure at short enough timescale

Telemetry for ML Workloads Needs Enhanced Congestion Signals

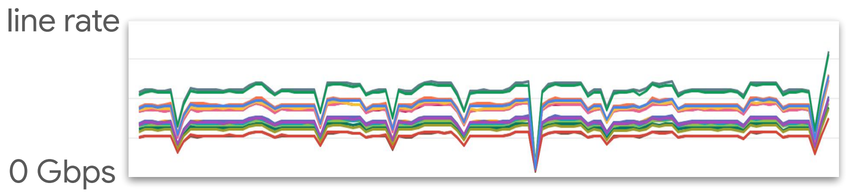
Scenario: deploy ML training job, would like to understand its congestion behavior in fabric to refine model's design (e.g., to use more of capacity, or avoid congesting fabric too severely)

- Synchronized transmissions in collective communication create intense, brief traffic bursts, and congestion episodes that last under 1 ms
  - Today's typical switch telemetry measures over 1 s or longer
  - Cannot even **observe** workloads' fabric congestion behavior if cannot measure at short enough timescale
- Bulk telemetry from status-quo switches (e.g., traces of port queue lengths, link traffic levels) utterly divorced from application
  - Need to **interpret** congestion in application context: identify which collective ops in training job encountered congestion in fabric
  - Status quo: to match port/link switch congestion data to application sends, must log transmitted packets, routing, load balancing decisions, all finely time-sync'ed, and "join" these data sets
  - While perhaps not impossible, huge effort; not scalable to always-on capability

Example: The Need for Fine-Grained Observability in Time for ML Workload Telemetry

Real-world example: port tx utilization from ToR in Google GPU cluster during ML training job.

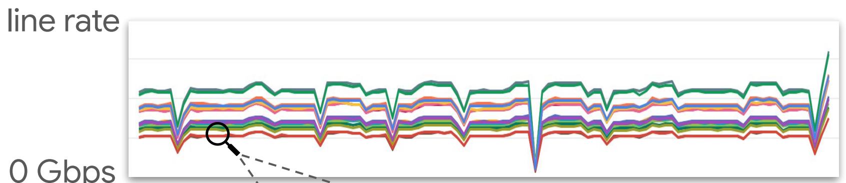
1-second measurement granularity. No congestion evident:



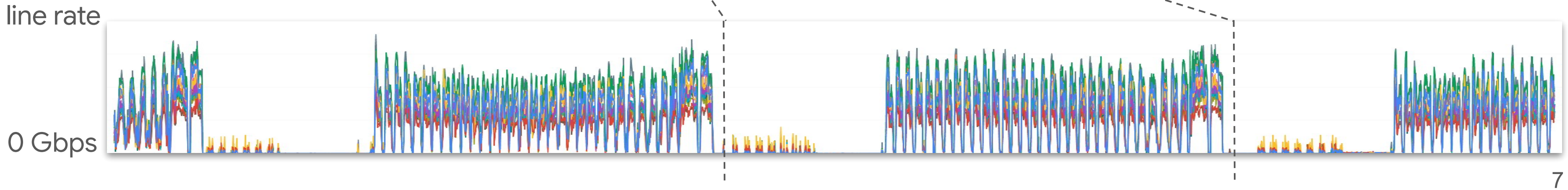
Example: The Need for Fine-Grained Observability in Time for ML Workload Telemetry (2)

Real-world example: port tx utilization from ToR in Google GPU cluster during ML training job.

1-second measurement granularity. No congestion evident:



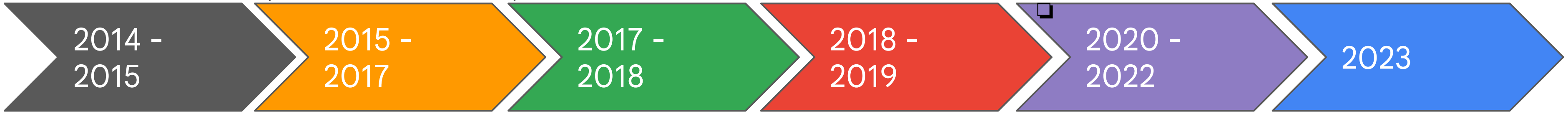
At 100 us measurement granularity (same workload), hallmark alternating idle periods and brief, intense bursts and congestion episodes within a training step readily visible:



Fine-Grained Observability and Interpretability: Design Goals for Congestion Signals

- Fine-grained observability in time: measurements over fine enough timescales to expose sub-ms congestion episodes typical in ML workloads
- Fine-grained observability in space: measurements of congestion state at bottleneck hop alone, rather than for all hops along path combined; identify location of bottleneck hop
- Interpretability: must be able to place congestion signals from fabric in application context; must measure congestion experience of application's packets to reason about impact on that application

A Brief History of In-Band Network Telemetry (INT)



- Millions of Little Minions: Using packets for low latency network programming and visibility, [Jeyakumar et al., SIGCOMM 2014]
- In-band Network Telemetry via Programmable Data Planes, [Kim et al., SIGCOMM 2015 Demo]
- Flow Table in Cisco Cloud Scale ASICs, 2015

Initial P4-INT Spec [Barefoot Networks]

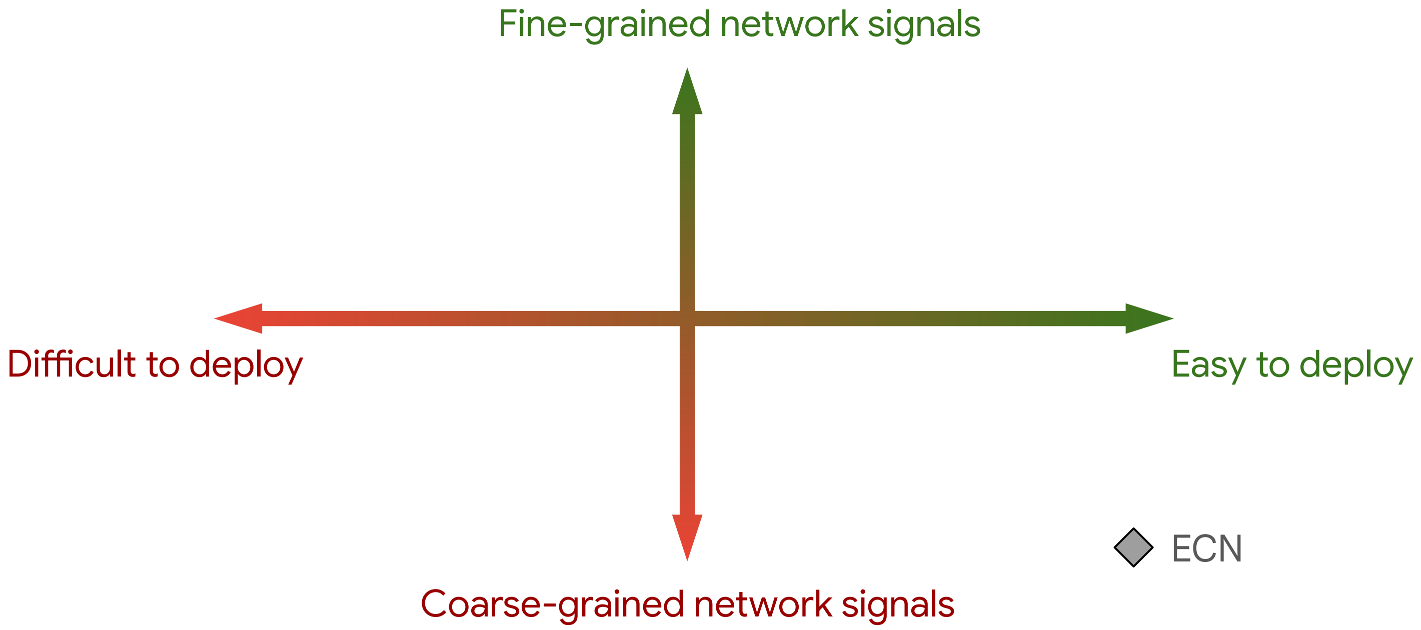
- INT v1.0 Spec [P4 Applications WG, 2018]
- Barefoot Deep Insights GA
- Inband Flow Analyzer (IFA 1.0) [Kumar (Broadcom) et al., ID, 2018]

- IFA 2.0 [Kumar (Broadcom) et al., ID, 2019]
- In-situ OAM (iOAM) [Cisco, ID, 2018]
- Streaming Statistics eXport (SSX) in Cisco Cloud Scale ASICs, 2018
- HPCC: High Precistion Congestion Control [Li et al., SIGCOMM 2019]

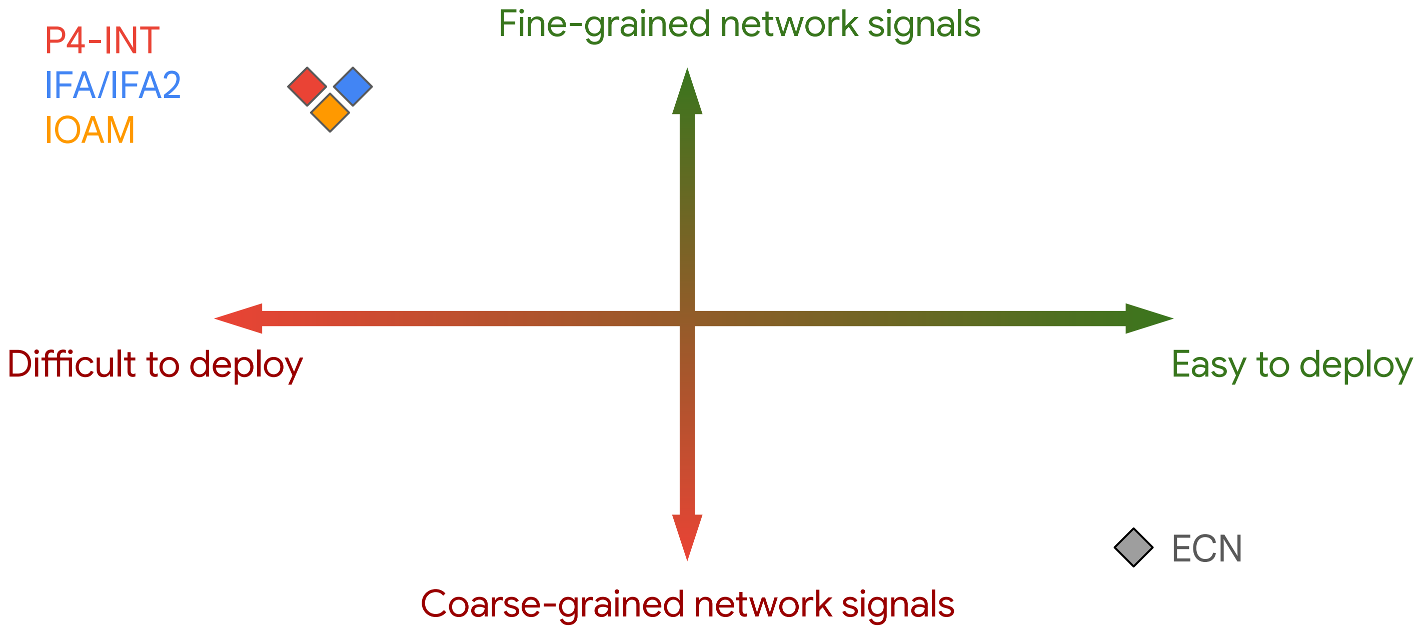
- INT v2.1 Spec [P4 Applications WG, 2020]
- PINT: Probabilistic In-Band Network Telemetry [Ben Basat et al., SIGCOMM 2020]
- HPCC++ proposal [Miao (Alibaba) et al., ID, 2021]
- TCP-INT proposal [Wass (Intel) and Miao (Baidu), P4 Workshop, 2022]

- Congestion Signaling (CSIG) initial proposal [Ravi (Google) et al., ID, 2023]

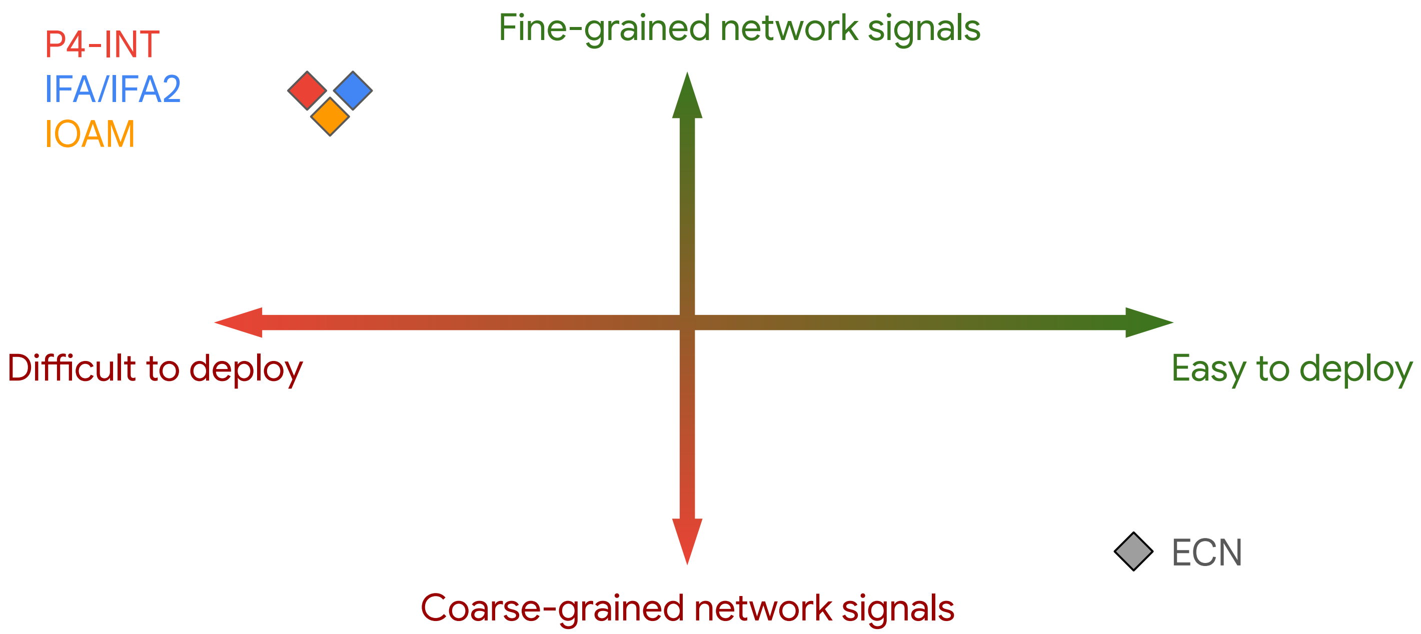
Axes of INT: Signal Granularity vs. Ease of Deployment



Axes of INT: Signal Granularity vs. Ease of Deployment



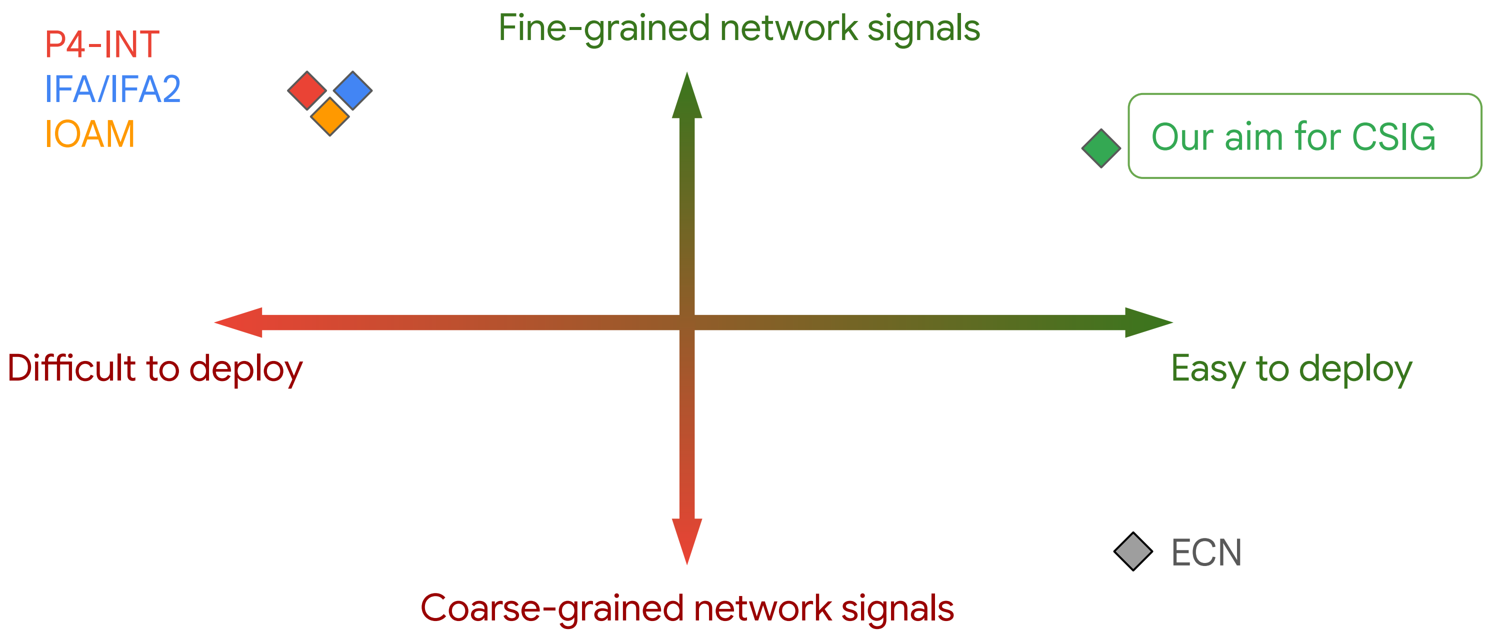
Axes of INT: Signal Granularity vs. Ease of Deployment



Prior industry standards for in-band network telemetry (P4-INT, IFA, IOAM) explored promising signals, but costly in header overhead and processing at switches:

- header space proportional to path length
- switch processing at variable packet offsets
- modifications to headers that may be encrypted

Axes of INT: Signal Granularity vs. Ease of Deployment



Prior industry standards for in-band network telemetry (P4-INT, IFA, IOAM) explored promising signals, but costly in header overhead and processing at switches:

- header space proportional to path length
- switch processing at variable packet offsets
- modifications to headers that may be encrypted

CSIG Protocol: Simple, In-Band Congestion Signals

Packet format

Dest MAC	Src MAC	CSIG tag	EtherType	IP header	Transport	Payload

CSIG Tag Fields

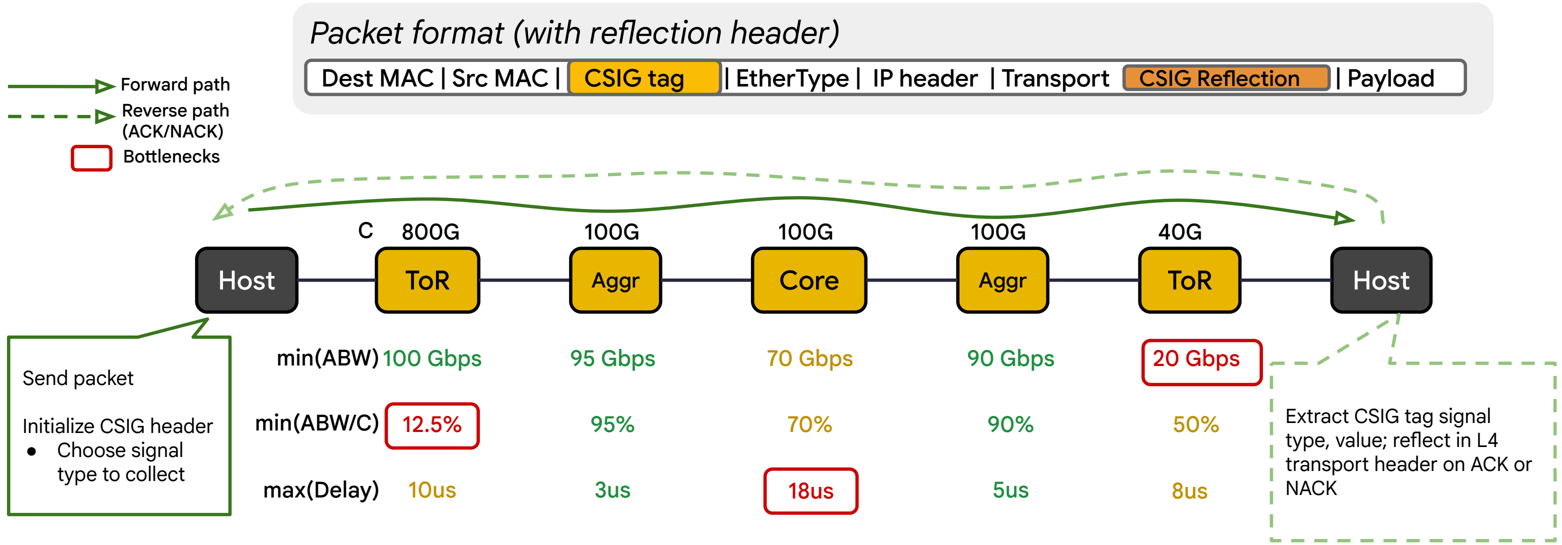
Field	Width (4B tag)	Width (8B tag)
Type (T)	3 bits	4 bits
Signal Value (S)	5 bits	20 bits
Locator Metadata (LM)	6 bits	15 bits

CSIG Signal Types

max(Delay)	Greatest queueing delay among all hops along packet's forward path
min(ABW)	Least absolute available output port bandwidth among all hops along packet's forward path
min(ABW/C)	Least relative available output port bandwidth (as fraction of output port link speed) among all hops along packet's forward path

- CSIG information carried in L2 Ethernet tag; 4B and 8B tag formats
- Unlike previously proposed INT headers, a CSIG tag carries only a single signal value (sufficient for measuring signal at bottleneck)
- Signal values in CSIG tags are bucketized (4B tag) or quantized (8B tag)

CSIG End-to-End: Origination, Switch Forwarding Behavior, Reflection


Packet format (with reflection header)
Dest MAC | Src MAC | CSIG tag | EtherType | IP header | Transport CSIG Reflection | Payload
Forward path
Reverse path (ACK/NACK)
Bottlenecks
C
800G
100G
100G
100G
40G
Host
ToR
Aggr
Core
Aggr
ToR
Host
min(ABW) 100 Gbps 95 Gbps 70 Gbps 90 Gbps 20 Gbps
min(ABW/C) 12.5% 95% 70% 90% 50%
max(Delay) 10us 3us 18us 5us 8us
Send packet
Initialize CSIG header
• Choose signal type to collect
Extract CSIG tag signal type, value; reflect in L4 transport header on ACK or NACK

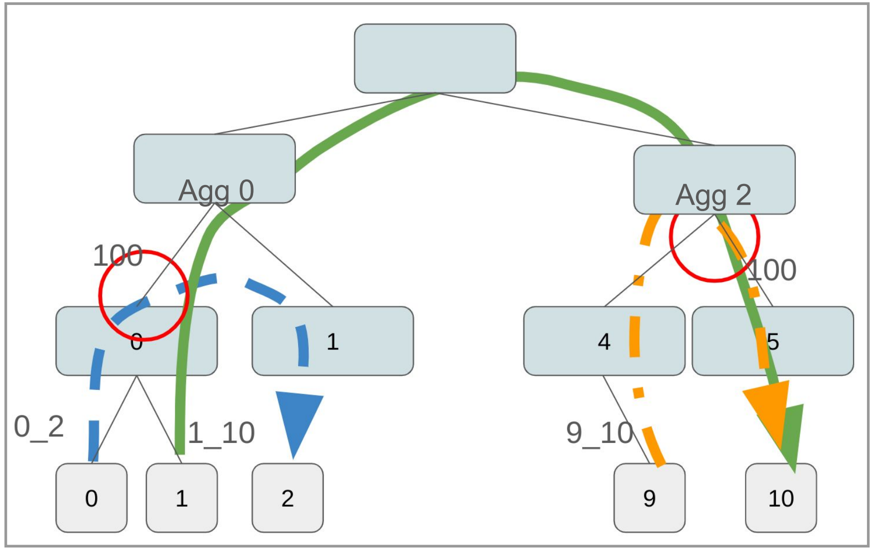
- Each switch compares local signal value for signal type in tag with signal value in tag; conditionally overwrites value in tag with local value according to aggregation function in signal type

Designing CSIG for Efficiency, Fine-Grained Observability, Interpretability, Deployability

- Per-hop signal scope pinpoints congestion state at bottleneck
- Space-efficient telemetry collection
    - Fixed-length 4B or 8B tag per packet, independent of path length; no insertions (compare with prior INT)
    - Central design insight: only measurement from bottleneck germane
- Compute-efficient telemetry collection
    - Switch operations to process CSIG tags simple
- In-band collection interpretable in application/transport context
    - Given low space overhead of tag, can attach to every packet of flow
    - Reflected signals indicate congestion observed by application's actual packets, and match switch congestion state with affected application
- Transport- and encapsulation-agnostic
    - L2 tag at fixed offset for switches, regardless of transport and encap; not encrypted
- Deployable on pre-CSIG switch silicon
    - L2 tag format similar to 802.1Q VLAN tag; amenable to processing using VLAN tag processing support
- Extensible
    - Protocol framework allows definition of future signal types

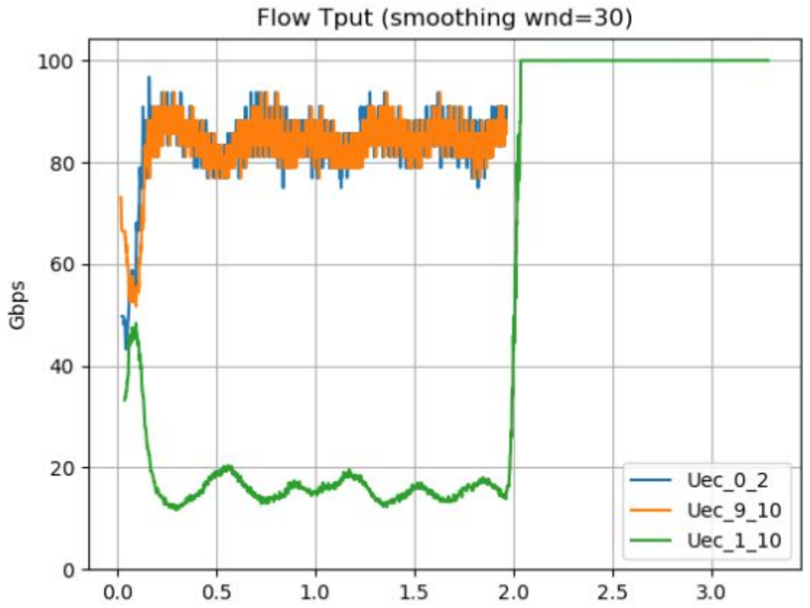
CSIG CC Use Case: Improving Bottleneck Sharing Behavior for Delay-Based CC

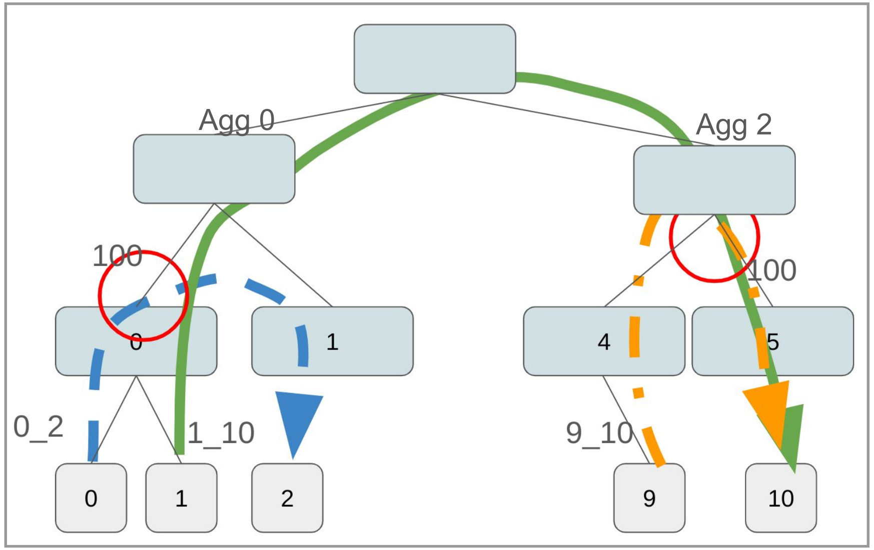
- Core intuition: using queueing delay of entire forward path can result in unfair sharing when competing flows traverse different numbers of congested hops
- "Victim" flow 1⟶10 encounters two congested queues at ToR 0 (uplink), Agg 2 (downlink)
- Flows 0⟶2 and 9⟶10 each encounter one congested queue each at ToR 0 (uplink), Agg 2 (downlink)
- Perfect max-min fairness: 50-50 sharing between victim flow and single competing flow at each of the two congested queues
- Expected delay-based CC behavior: victim flow observes two queues' worth of queueing delay, but other two flows observe one queue's worth; status-quo delay-based CC will throttle victim more severely than other two [Poseidon, NSDI 2023]



- 3-tier fat tree, 16 hosts, 100 Gbps links, 2:1 oversubscribed, link delays 1us
- Only single paths exist between host pairs
- (Figure omits Agg 1 and Agg 3 subtrees for clarity)

Baseline NSCC Throughput Fairness in 3-Flow Scenario





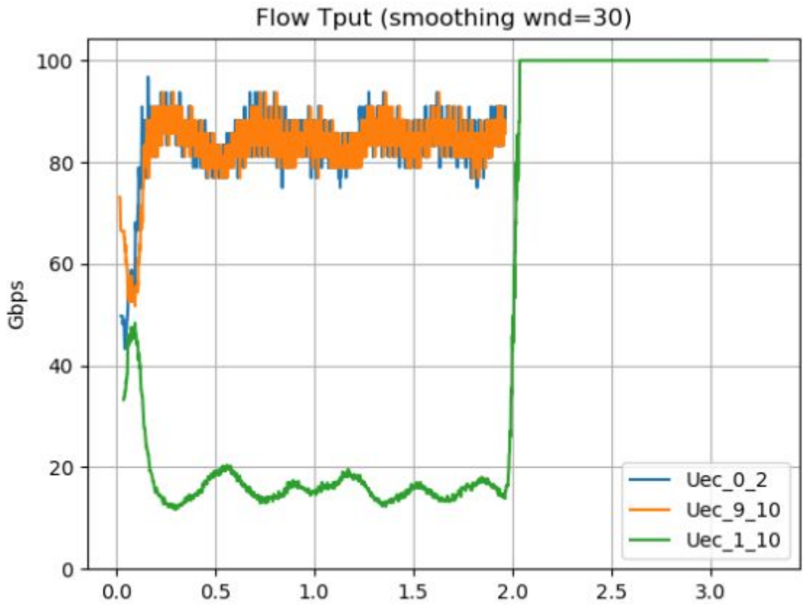
~5:1 steady-state throughput ratio between non-victim and victim flows while all three flows active (matching intuition for delay-based CC behavior)

Extending NSCC to Use CSIG max(Delay)

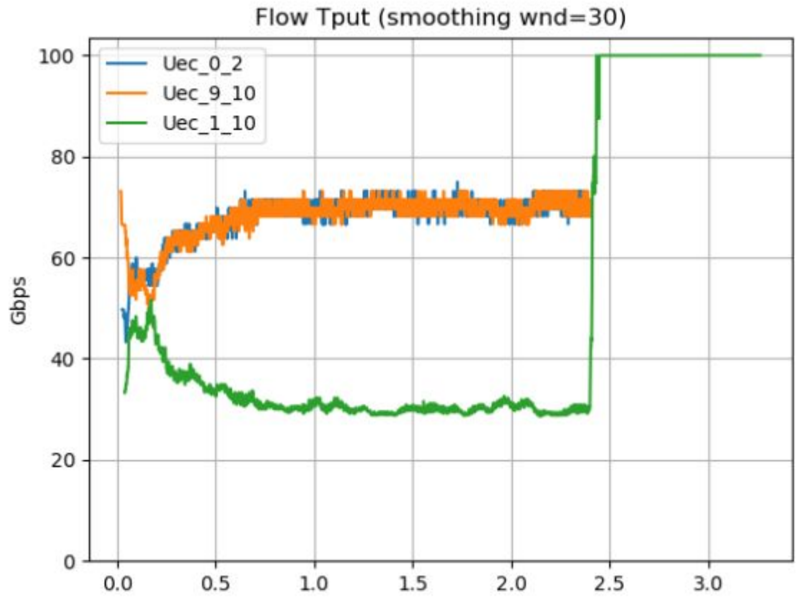
- Aim: sender adapts cwnd based only on queueing delay at single true bottleneck queue on path
  - Intuition: should reduce relative penalization of flows that encounter more congested hops
- Sender places CSIG max(Delay) 4B tag in all data packets
- Receiver reflects CSIG max(Delay) value in ACKs
- In sender control loop, replace path queueing delay samples from ACKs with CSIG max(Delay) feedback from ACKs
- No change to target_Qdelay calculation
- No regressions in full htsim validation suite

UET CC CSIG Use Case: max(Delay) Improves NSCC Fairness (3-Flow Scenario)

Baseline NSCC



NSCC + CSIG max(Delay)

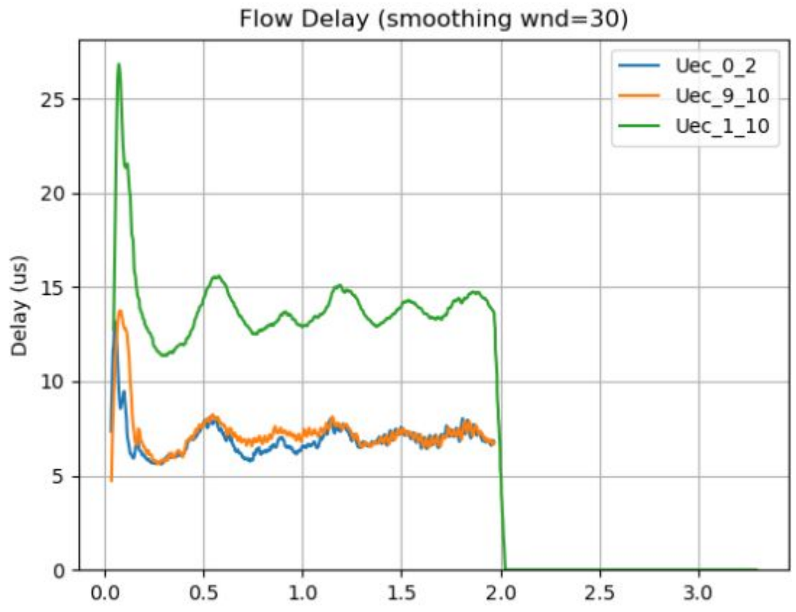


Steady-state throughput ratio improvement:

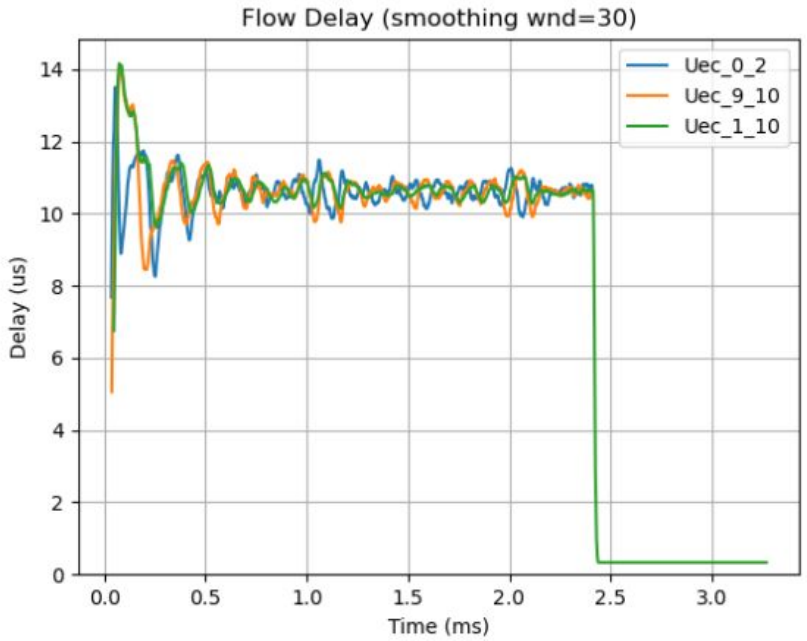
From ~5:1 (baseline) to ~2.3:1 (with CSIG max(Delay)

How does CSIG max(Delay) Improve NSCC's Fairness? (1/2)

Baseline NSCC

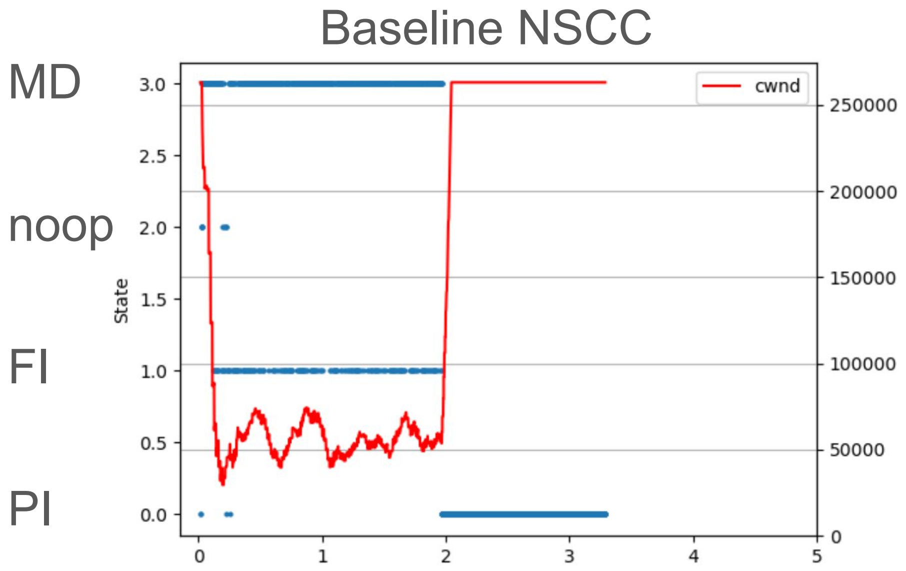


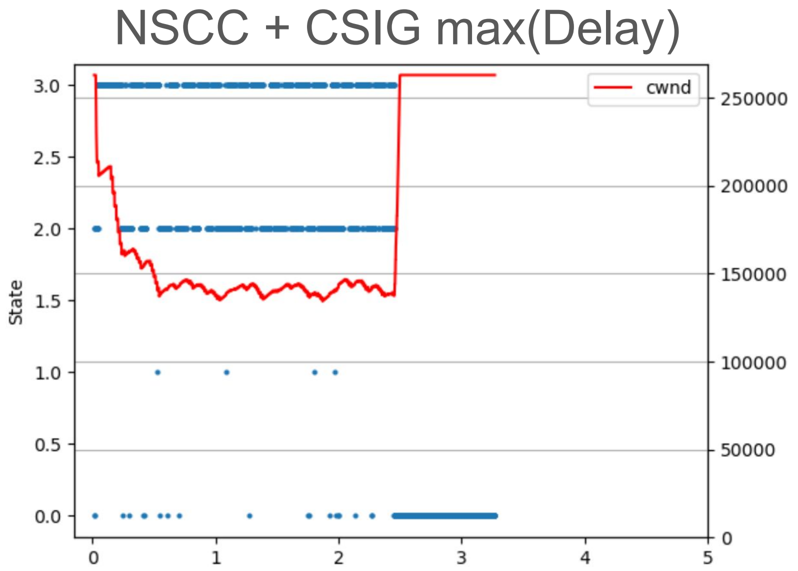
NSCC + CSIG max(Delay)



On this topology, target_delay = 10.5us (default computed value in htsim)
Under baseline NSCC, victim flow's queueing delay remains persistently above target, while other flows' queueing delays remain persistently below it. Effect on control loop?

How does CSIG max(Delay) Improve NSCC's Fairness? 2/2





When queueing delay > target_delay persistently, baseline NSCC's control loop performs almost no noops or proportional increases (PI) for victim flow's cwnd, and instead performs multiplicative decreases (MD) and fair increases (FI)

By allowing victim flow's queueing delay to converge around target_delay, CSIG max(Delay) causes NSCC's control loop to perform significantly more noops and PIs (replacing MDs and FIs in baseline NSCC)

Bottleneck Sharing Behavior in Delay-Based CC: Collective-Like Scenario

Traffic pattern: pipeline + all reduce traffic

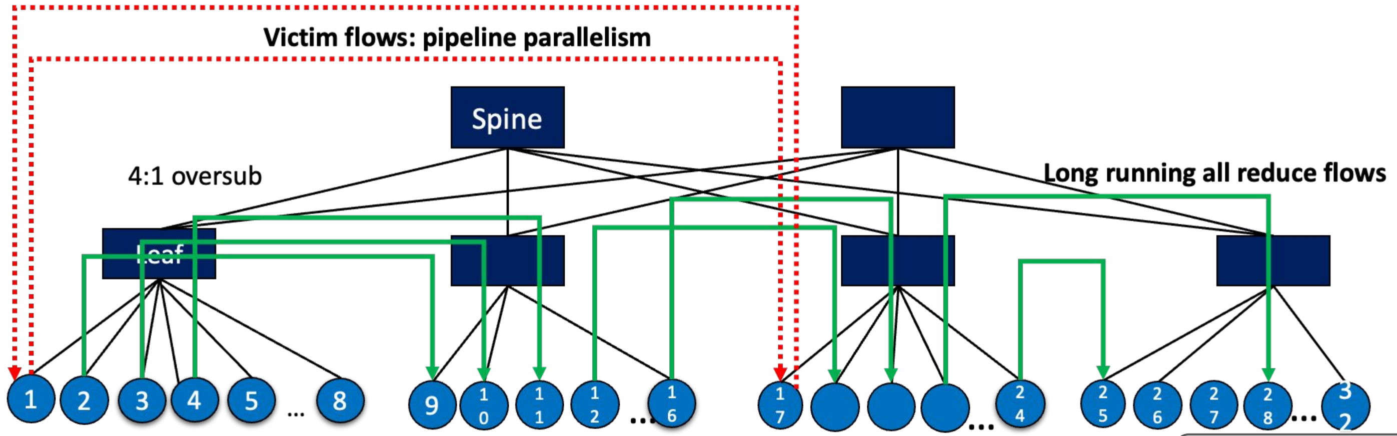

Victim flows: pipeline parallelism
Spine
4:1 oversub
Leaf
Long running all reduce flows
1
2
3
4
5
...
8
9
10
11
12
...
16
17
...
24
25
26
27
28
...
32

Figure and simulation: Costin Raiciu, Broadcom

2-tier Fat-Tree, 4:1 oversubscribed, 800 Gbps links, 1us link latency

Collective-like, oversubscribed communication: 30 long-running all-reduce flows (2 GB flow sizes) compete with “victim” shorter pipeline parallelism flows (64 MB flow sizes)

Unlike in prior 3-flow example, multiple paths exist between host pairs

UET CC CSIG Use Case:
max(Delay) Improves NSCC's Fairness and FCT (Collective-Like Scenario)

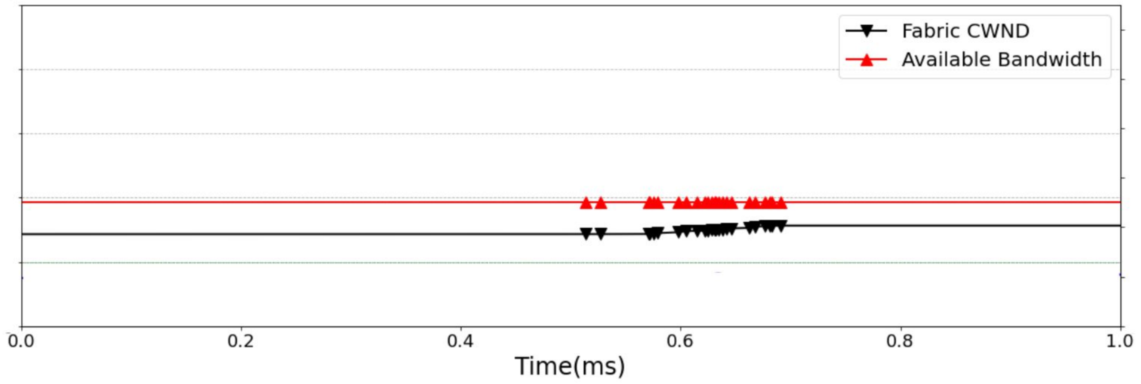
- Victim (pipeline parallelism) flows' throughputs:
  - Baseline NSCC: 13.7 and 14.0 Gbps
  - NSCC + CSIG max(Delay): 22.6 and 23.1 Gbps
  - CSIG max(Delay) improves each victim flow's throughput by ~65%
- Victim (pipeline parallelism) flows' FCTs:
  - Baseline NSCC: 37.3ms and 36.7ms
  - NSCC + CSIG max(Delay): 22.6 and 22.2ms
  - CSIG max(Delay) reduces each victim flow's FCT by ~39%
- Under a collective-like oversubscription scenario, CSIG max(Delay) can offer fairness and FCT benefits

Fast and Safe CC Ramp-Up for Swift with CSIG min(ABW/C)

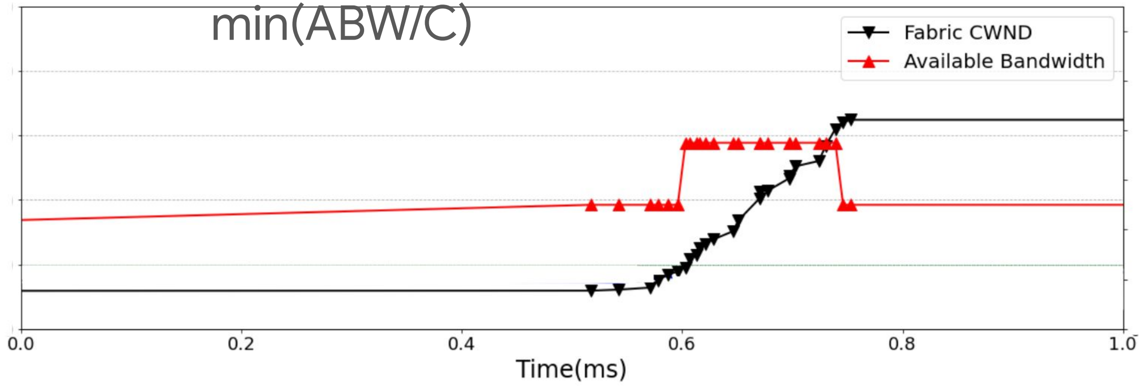
- Like many CC protocols, Swift searches for available bandwidth on path using additive increase (additive increment) (AI)
- When flows at a shared bottleneck terminate, continuing flows may need many RTTs of AI to claim abruptly released capacity
- Baseline Swift fabric cwnd update (per RTT)
    - if (rtt < target rtt)
      $\text{fcwnd} \longleftarrow \text{fcwnd} + \text{ai}$
- Fast Ramp-Up (FRU): CSIG-Enhanced Swift fabric cwnd update (per RTT)
    - if (rtt < target rtt)
      $\text{fcwnd} \longleftarrow \text{fcwnd} + \text{ai} + \lambda \cdot \text{fcwnd} \cdot \min(\text{ABW/C})$
    - Senders increase windows faster than AI would permit, in proportion to path bottleneck's relative available bandwidth
    - Intuition: total proportional increase by senders should remain within bottleneck's available bandwidth; $\lambda < 1$ for additional margin of safety under traffic dynamics

Swift+FRU with CSIG min(ABW/C) Reduces RPC Latencies

Baseline Swift



Swift+FRU with CSIG min(ABW/C)



Results from testbed benchmark of Swift+FRU with CSIG min(ABW/C) at Google.

Workload:

- Foreground cross-rack RPC traffic: 640 KB ops
- Bursty background cross-rack RPC traffic: 10 MB ops, burst duration 1 ms, gap between bursts 4 ms

15% median improvement in foreground ops' latency; background ops' latency unchanged.

Applications of CSIG Signals in Load Balancing

- min(ABW): Beyond binary ECN mark indication of “congestion” or “no congestion,” quantitative signal of available capacity to inform sender path choice and traffic weights across paths
- Locator metadata for max(Delay): distinguishes last-hop congestion (incast) from “core” congestion
  - LB should not move away from a path when congestion at last hop
  - Alternative mechanism to UET’s use of two DSCP codepoints to distinguish NACKs of packets trimmed at last hop from NACKs of those trimmed in core

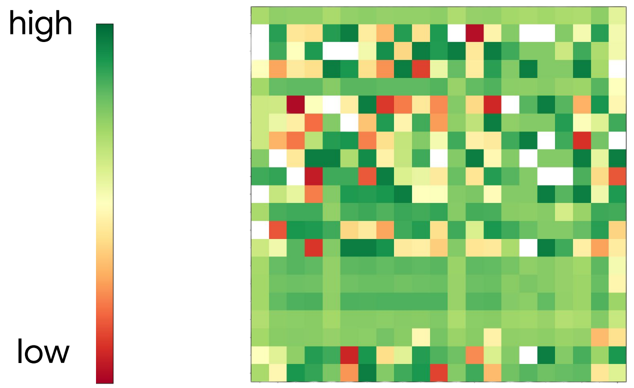
CSIG Telemetry for TPU Training Jobs at Google: Fine-Grained Congestion Observability

- CSIG min(ABW) telemetry for an ML training job maps even brief intervals of congestion in the fabric, enabling:
  - ML training job refinement: “Can my training job perform more communication without encountering congestion, and if so, where?” “Where is my training job congesting the fabric?”
  - Fabric capacity planning: “Is the fabric adequately provisioned for the models I am running?”

CSIG Telemetry for TPU Training Jobs at Google: Fine-Grained Congestion Observability

- CSIG min(ABW) telemetry for an ML training job maps even brief intervals of congestion in the fabric, enabling:
  - ML training job refinement: “Can my training job perform more communication without encountering congestion, and if so, where?” “Where is my training job congesting the fabric?”
  - Fabric capacity planning: “Is the fabric adequately provisioned for the models I am running?”

Telemetry from a production TPU ML training job at Google:



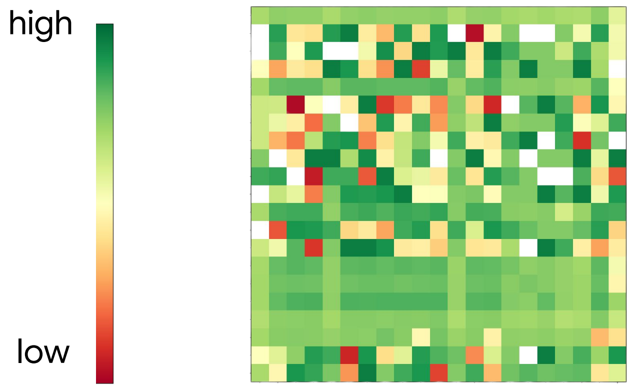
Heatmap of mean available bandwidth observed by TPU job’s flows

Each small square summarizes data for packets between one portion of cluster (x coord) and another (y coord)

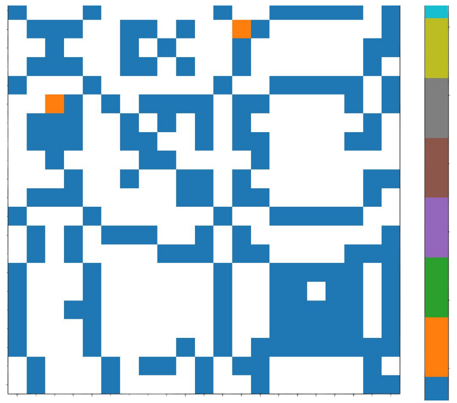
CSIG Telemetry for TPU Training Jobs at Google: Fine-Grained Congestion Observability

- CSIG min(ABW) telemetry for an ML training job maps even brief intervals of congestion in the fabric, enabling:
  - ML training job refinement: “Can my training job perform more communication without encountering congestion, and if so, where?” “Where is my training job congesting the fabric?”
  - Fabric capacity planning: “Is the fabric adequately provisioned for the models I am running?”

Telemetry from a production TPU ML training job at Google:



Heatmap of mean available bandwidth observed by TPU job’s flows

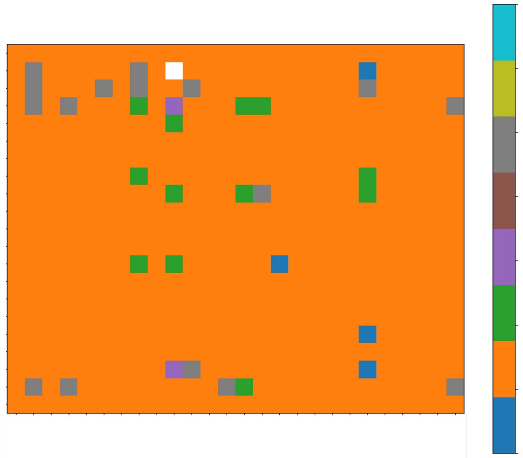


Most frequent (mode) location of bottleneck experienced by TPU job’s flows, only among heavily loaded bottleneck links

Each small square summarizes data for packets between one portion of cluster (x coord) and another (y coord)

CSIG Telemetry for Evaluation of Protocol Improvements at Google

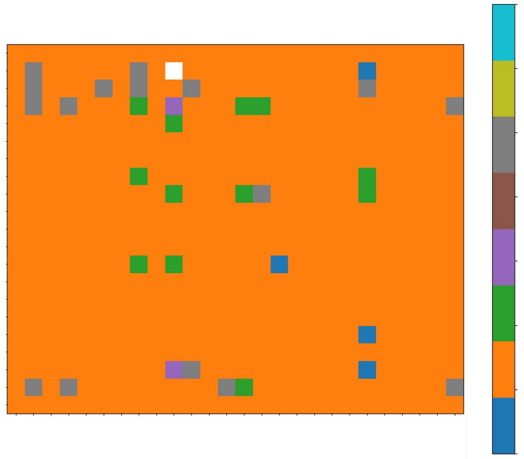
- CSIG ABW/C telemetry maps how a change to congestion control or load balancing shifts an app's (or multiple apps') load within the fabric
- Before-and-after views of ABW/C locator metadata show shift in bottleneck location after deployment of a load balancing scheme at Google:



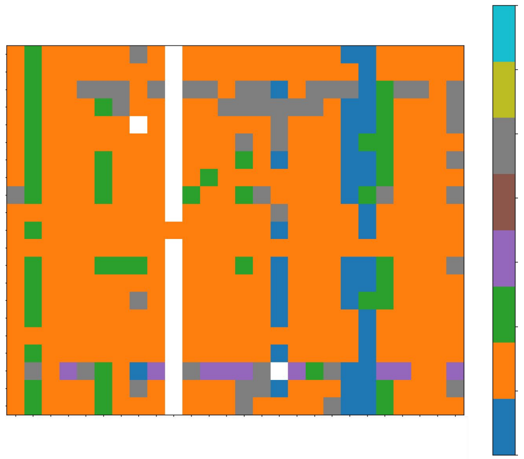
BEFORE: Most prevalent ABW bottleneck location is ToR uplinks

CSIG Telemetry for Evaluation of Protocol Improvements at Google

- CSIG ABW/C telemetry maps how a change to congestion control or load balancing shifts an app's (or multiple apps') load within the fabric
- Before-and-after views of ABW/C locator metadata show shift in bottleneck location after deployment of a load balancing scheme at Google:



LB scheme deployed



BEFORE: Most prevalent ABW bottleneck location is ToR uplinks

AFTER: Most prevalent ABW bottleneck location has moved elsewhere; LB has alleviated ToR uplink congestion

CSIG: Simple, Fine-Grained, Efficient Congestion Signals

- ML workloads demand observability and interpretability of fabric congestion
  - Observable at fine granularity in time (100 us, not 10 s)
  - ...and at fine granularity in space (at one bottleneck hop, not black-box summarized for a whole path)
  - Interpretable in application context, for per-flow CC and to reveal application congestion impact
- Central design principle: fabric congestion state at path bottleneck is sufficient signal, both for CC and telemetry
  - Unlocks deployability: space-efficient tag format, processing-efficient switch-side implementation
- L2 tag with L4 end-host reflection avoids complexity/cost of variable header offset, insertion into packets, encrypted tag update at switches
- Broad scope for use across CC, load balancing, scheduling, traffic engineering, and telemetry; what will you design?
- To benefit from these many uses, Google is deploying CSIG at scale in production
- To foster broad availability of CSIG in NICs and switches, Google and industry collaborators are standardizing CSIG at the Ultra Ethernet Consortium (UEC)