CSIG: Congestion Signaling for Datacenter Transports

Abhiram Ravi, Nandita Dukkipati, Weiwu Pang, Neal Cardwell, Brad Karp†, MJ Akhbari, Weida Huang, Konstantinos Prasopoulos*, KK Yap, Arjun Singh, Amin Vahdat
Google LLC

Abstract

Optimizing burst-heavy datacenter workloads necessitates fine-grained network control and visibility. We introduce *CSIG*, a protocol that delivers precise, multi-bit bottleneck congestion signals via a fixed-length Ethernet header. The architecture captures μs-granularity switch metrics, such as available bandwidth, and signals them to end-hosts using in-band, line-rate operations. We propose Fast Ramp-Up, a congestion control primitive that leverages these bottleneck signals to reduce median RPC latency by 20% and unclaimed bandwidth by 60% in production. Beyond transport-level performance, CSIG enables flow-aware observability by embedding μs-scale metrics into every packet, allowing individual application transfers to pinpoint their bottleneck location, such as the topology tier limiting their performance. CSIG thus transforms network telemetry from post-hoc correlation into a real time, context-aware capability. We demonstrate CSIG's broad deployability by validating it across five generations of commodity switch hardware (up to 102.4 Tbps), four NIC generations, and five transport stacks. Our design proves that a streamlined Layer 2 approach, focusing exclusively on the principal path bottleneck, provides transport-agnostic gains without requiring forklift hardware upgrades.

CCS Concepts

• **Networks** → **Data center networks**; **Link-layer protocols**; **Transport protocols**; **Network measurement**.

Keywords

Network Protocols, In-band Signaling, Datacenter Transports, Congestion Control, Network Telemetry

ACM Reference Format:

Abhiram Ravi, Nandita Dukkipati, Weiwu Pang, Neal Cardwell, Brad Karp, MJ Akhbari, Weida Huang, Konstantinos Prasopoulos, KK Yap, Arjun Singh, Amin Vahdat. 2026. CSIG: Congestion Signaling for Datacenter Transports. In *ACM SIGCOMM 2026 Conference (SIGCOMM '26), August 17–21 2026, Denver, CO, USA*. ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3789240.3829179

1 Introduction

Datacenter workloads, especially large-scale AI/ML, demand extreme predictability, massive burst bandwidth, and ultra-low latency. While tail latency dictates job completion times, synchronized microbursts drive rapid fluctuations between acute congestion and idle capacity. Achieving *precise* control and resource efficiency during these transients remains a formidable challenge for modern transport protocols, which are fundamentally constrained by a lack of fabric *observability*. To bridge this gap and respond effectively, telemetry systems and control loops require congestion signals fine-grained enough to render these burst dynamics visible across three critical axes:

- **In time:** over the interval of a datacenter flow's round-trip time (RTT) for prompt detection and response;
- **In resolution:** multi-bit congestion signals for precise telemetry and response;
- **In space:** at the bottleneck switch that limits performance, rather than summing along an entire end-to-end path.

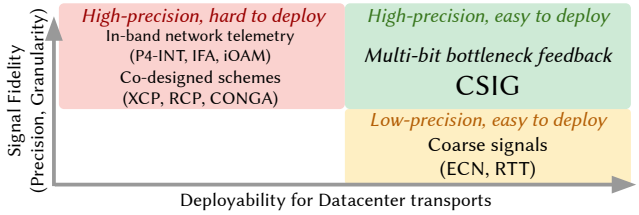

Figure 1: Fidelity-deployability tradeoff for DC transports

Indeed, for over two decades, in-band signaling has promised better transports through finer-grained congestion metrics. Yet production adoption remains stymied by tradeoffs between fidelity and deployability (Fig. 1). Early protocols such as XCP [28] and RCP [14, 15] were limited by switch vendors' reluctance to implement special-purpose L3/L4 processing for use cases perceived as narrow. Subsequent general-purpose frameworks such as P4-INT [52] and IFA/IFA2 [32] prioritized flexibility but introduced architectural barriers, including performance overhead from variable-length headers, incompatibility with NIC hardware offloads, and entanglement with L3+ tunneling protocols and encryption schemes. While Explicit Congestion Notification (ECN) [19, 20] is easily deployed, its single-bit nature provides only coarse telemetry, forcing control loops to average signals [29], sacrificing responsiveness.

We introduce *Congestion Signaling* (*CSIG*), a protocol that provides precise, multi-bit bottleneck congestion signals to end hosts while overcoming prior deployment barriers to in-band signaling. CSIG achieves this through a transport-neutral design and a fixed-size Ethernet header, using line-rate compare-and-replace operations to deliver fine-grained signals in every data packet. The architecture comprises two components: the CSIG tag, an L2 header where switches record real-time bottleneck metrics, and CSIG Reflection, which returns signals to the sender via L4 transport protocols. The CSIG tag, positioned as the final tag in the Layer 2 header, is available in two variants: a Compact 4-byte tag (5-bit signal plus locator) or a Wide 8-byte tag (20-bit signal plus locator). This placement ensures compatibility with end-to-end encryption and L3+ tunneling schemes. At its core, CSIG leverages high-precision switch telemetry to provide signals at line rate, either natively via

† Work done while on leave from UCL. *Work done while at Google.

SIGCOMM '26, Denver, CO, USA

ACM ISBN 979-8-4007-2467-1/26/08
https://doi.org/10.1145/3789240.3829179

switch ASICs or through software-assisted low-latency polling of hardware counters. By carrying signals in-band with application data, the signals map naturally to flow context. Furthermore, by decoupling the switch-side L2 tag from the host-side L4 reflection, CSIG allows transports such as TCP, PonyExpress [34], and Falcon [49] to evolve their control loop responses independently of switch hardware.

Contributions. We designed and deployed CSIG to bridge the long-standing gap between high-fidelity network control and the pragmatic constraints of production environments. For two decades, in-band signaling research has been stymied by complex, all-hop telemetry mechanisms that proved difficult to deploy at scale. We shift this paradigm by identifying that transport control loops and performance analysis are optimally served by bottleneck-only, μs-scale signals—an observation that enables a streamlined, L2, fixed-size header capable of achieving both generality and ubiquity. Crucially, we demonstrate that these high-precision signals can be realized efficiently on commodity, fixed-function ASICs dating back to 2010. By prioritizing deployability as a core architectural requirement, we provide a protocol that integrates seamlessly across the entire datacenter ecosystem. Our contributions are as follows:

(1) *CSIG protocol:* We present the architectural principles that underlie CSIG, an in-band signaling protocol that provides end hosts with μs-scale visibility into path bottleneck signals in an Ethernet Layer 2 header. In collaboration with industry partners, we are standardizing the protocol, described elsewhere [46, 54].

(2) *High-precision switch telemetry (HST).* Via hardware or a software-assisted approach usable on existing ASICs, HST captures microbursts at 100μs or finer timescales, and updates CSIG headers at line rate.

(3) *Fast Ramp-Up (FRU) Algorithm*: FRU, a congestion control primitive using CSIG signals, cuts median RPC latency by 20% and reduces unclaimed bandwidth by 60% in production scenarios. Additionally, CSIG's precise bottleneck location metadata enables dynamic load balancing that reduces tail latency by 2.5× and unstable path switching by 3.75×.

(4) *Application context-aware congestion observability:* By embedding μs-scale bottleneck signals into every packet, CSIG enables the first real-time mapping of network dynamics to specific application flows, e.g., annotated distributed traces with CSIG metrics allows operators to instantly pinpoint the exact topology tier impacting specific RPCs.

(5) *Broad Deployability:* CSIG delivers significant transport gains and context-aware observability without requiring a "fork-lift" hardware upgrade; its design is proven across five generations of switch silicon (up to 102.4 Tbps), four NIC generations, and five transport stacks.

Ethics. This work does not raise any ethical issues.

Gen AI. Gemini was used to polish author-written text.

2 Challenges and Requirements

The networking community has long recognized the potential of in-band switch signaling to solve problems in congestion control, load balancing, and fault localization. However, widespread adoption in large-scale production environments has proven elusive. We examine the evolution of prior work to identify the trade-offs between generality and deployability that have hindered success.

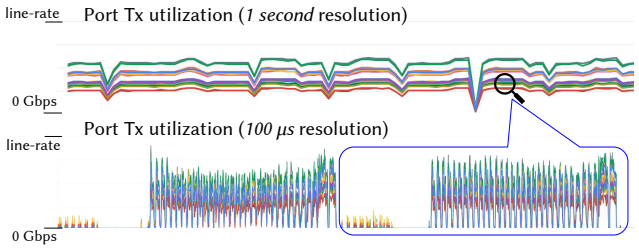

Figure 2: Output port bandwidth from a GPU ToR: Shifting from 1-second to 100-μsec telemetry exposes the synchronized, alternating congestion episodes and idle gaps in ML training.

We then articulate the requirements for a signaling mechanism that is precise, fast, expressive, and, most importantly, deployable.

2.1 In-Band Signaling Challenges

In-band signaling has spanned two eras, with unique barriers to adoption in each. Early proposals took a *dedicated, per-protocol* approach. Congestion Control (CC) protocols such as XCP [28], RCP [14], and VCP [57] defined L3/L4 header fields for switches to update, while CONGA [1] reserved bits in VXLAN overlay headers for load balancing. These proposals piqued the community's interest in in-band signaling, but lacked adoption because switch vendors rarely implement specialized L3/L4 header processing for single-use protocols. To overcome this, the next generation of proposals prioritized *generality*: TPPs [27], P4-INT [52], IFA [32], and iOAM [9]. These support diverse signals and multi-hop telemetry within a single header. However, this flexibility introduced significant complexity that is at odds with the operational realities of hyperscale networks:

Path-Length Proportional Headers: Carrying telemetry for every hop creates variable-length headers that grow with network diameter. This consumes excessive bandwidth and risks exceeding the MTU, leading to unacceptable fragmentation and reassembly overheads.

Variable Offsets & Parsing Complexity: Variable-length headers shift the offsets of subsequent headers (e.g., L4). For NICs, this breaks critical offloads, e.g., Generic Receive Offload, Receive-side Zero-copy, that rely on fixed-offset parsing. For switches, appending data at variable offsets creates a costly sequential dependency: the switch must parse the variable-length header before writing to the right offset, hindering line-rate processing.

Entanglement with L3+ encapsulations: Embedding signaling at Layer 3 or above, as do P4-INT, IFA/IFA2, and iOAM, complicates encapsulation. Because parsing *inner* headers at line rate is prohibitively expensive, encapsulating switches must copy telemetry data to the *outer* header to make it visible to the fabric, and decapsulating switches must merge it back. This entangles the signaling with encapsulation, requiring distinct copy-logic for every supported tunneling scheme, e.g., VxLAN, IP-in-IP.

Encryption Incompatibility: When telemetry is embedded within the payload (as in some IFA variants) or L4 headers protected by encryption (e.g., PSP, IPsec), it becomes inaccessible to forwarding switches, rendering the mechanism ineffective in secure environments.

2.2 In-Band Signaling Requirements

For effective in-band signaling, we define seven key requirements for *control loops*, *observability*, and *deployability*.

(R1) Promptness and Frequency. Congestion signals must arrive at a sending host within one RTT and be obtainable on *every* data packet. Feedback delayed beyond an RTT slows control loop convergence, as with prior in-band signaling schemes that can only keep pace with one "probe" of switch congestion levels per RTT [32]. Evaluations in §7.3 show that obtaining signals on every packet provides the best latency improvements and §6 captures the corresponding telemetry enhancements.

(R2) Fine-grained observability in time. In addition to capturing instantaneous queuing or delay, switches must measure bandwidth rates over intervals short enough to capture microbursts. As shown in Fig. 2, standard 1-second telemetry masks the rapid, alternating congestion characteristic of ML training workloads on production GPU clusters. Only at ~100μs resolution do these dynamics become visible. For CC loops, bandwidth measurement timescales should also not be so short as to cause overshooting during unrepresentative transients (§7.3).

(R3) Fine-grained observability in space. Signals must isolate the value and location of the *single true bottleneck*. Signals such as ECN [19, 20] or whole-path queuing delay accumulate across the entire path. Reacting to this sum penalizes flows with multiple congested hops for extraneous delays, yielding max-min unfair allocations [55]. Reacting only to the bottleneck restores fairness. Bottleneck localization also aids multipathing; it can prevent counter-productive multipathing when congestion occurs at the last hop (§5.1). For telemetry and debugging, it aids in pinpointing the hop that's limiting performance.

(R4) Fine-grained observability in signal resolution. To be sufficiently responsive to microbursts, datacenter transports need multi-bit congestion signals. Schemes such as DCTCP [2] that rely on binary ECN must average many ECN reports to derive a signal with enough resolution for robust, stable CC response. This *primal* CC approach slows down reactions to microbursts, which may be tolerable in long-RTT WAN settings where CC adapts slowly to begin with. However, in short RTT datacenter settings, *dual* algorithms where switches provide multi-bit congestion signals are preferred because they are immediately actionable and reduce queue variability, as noted by Kelly [29] .

(R5) End-host Deployability. The signaling scheme must be transport-neutral, integrating with both SW and HW transports. It must coexist with NIC offloads–to maintain line-rate performance at 200–1600 Gbps, the mechanism must support signal ingestion and reflection even alongside techniques such as receive-side coalescing or Rx Zero-Copy.

(R6) Switch Deployability. To avoid prior pitfalls, the signaling header must be *small, fixed-length, and at a predictable offset* to minimize bandwidth overhead and enable efficient hardware parsing. It must remain accessible even when the packet is encapsulated (tunneling) or the payload is encrypted, and be implementable in commodity ASICs at line rate within pipeline latency budgets.

(R7) Application Context. Signals must map intrinsically to the application flow context. Given data center scales, operators cannot rely on complex post-hoc correlation of separate "probe" packets to debug performance. Signals must be carried in-band with data, providing immediate, unambiguous mapping of network behavior to application flows.

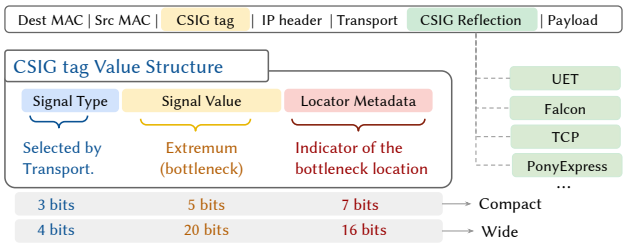

Figure 3: CSIG packet format and value structure

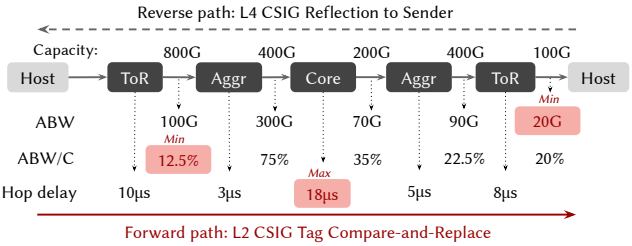

Figure 4: CSIG's L2 tag collects path extrema signals and L4 headers reflect signals to the sender. Different signal types capture distinct bottlenecks.

3 CSIG Design

CSIG's design reconciles application needs with the operational realities of production data center networks.

3.1 Overview

CSIG provides simple, low-overhead, fixed-length summaries of path extrema over packet headers. It employs two distinct components: the CSIG tag, an L2 header where transit devices record real-time bottleneck metrics via compare-and-replace operations, and CSIG Reflection, an L4+ protocol where end hosts return these signals to the sender (Fig. 3).

The CSIG tag is positioned as the final tag in the Layer 2 header and is implemented in two variants, each using a distinct Tag Protocol Identifier (TPID). The *Compact* variant (focus of this paper) resembles a single VLAN tag, while the *Wide* variant resembles a double VLAN tag in size.

Fig. 4 shows the CSIG packet lifecycle, starting at the sending host where the transport initializes a Compact or Wide L2 tag and selects a signal type, such as minimum available bandwidth or maximum hop delay. During network transit, switches perform compare-and-replace operations, overwriting the CSIG tag and locator metadata if the switch's local condition is more severe (e.g., lower available bandwidth). This process ensures the tag captures the forward path's true bottleneck extremum, noting that different signal types may be set at different bottleneck locations. Upon arrival, the receiver extracts the signal to track congestion and piggybacks it via an CSIG Reflection Header in the next outbound L4 packet. Finally, the sender extracts the reflected data to drive congestion control algorithms and export telemetry.

We first detail the CSIG tag design elements below, followed by their switch-side realization (§4) and their integration across transport stacks (§5) and telemetry systems (§6).

	Attribute	CSIG	P4-INT	IFA 2.0	iOAM	TCP-INT
Core Design	Protocol Layer	**L2**	L3/L4	L3/L4	L3	L4
	Datapath Operation	**Compare-Replace**	Append-at-hop			Compare-Replace
	Congestion Locator	**Bottleneck**	Full Path Trace			Hop TTL
	No Per-Flow Switch State	✓	✓			✓
Scalability	Packet Overhead	**4B/8B (Fixed)**	Variable (N × hops)			12B (Fixed)
	Overhead Indep. of # Hops	✓	✗			✓
	No MTU Modification Req.	✓	✗			✓
	Telemetry Coverage	**All Pkts**	Probes Only			Sampled
Deployability	Transport-Agnostic	✓	✓			✗
	Encrypted L4+ Payload	✓	✗			✗
	L3+ Tunnels (VxLAN, IP-in-IP)	✓	✗			✗
	L3/L4 Hash Compat. (ECMP)	✓	✗			✓
	Legacy Switch Participation	✓	✗			✗
	Support Reflection to sender	✓	✗			✓

Table 1: Comparison of in-band signaling schemes

3.2 Design principles

We describe CSIG's design principles that enable it to satisfy the requirements in §2. Tab. 1 summarizes how CSIG's core design, scalability, and deployability compare to existing in-band signaling schemes.

Multi-bit bottleneck signaling. CSIG provides multi-bit congestion signals that quantify path bottlenecks. It captures only the path extremum and the bottleneck location, and encodes them into a 4B or 8B CSIG tag. A single packet carries only one such congestion signal as per the signal type, which avoids the overhead and complexity of managing metadata for multiple signals in the packet. End hosts can, however, request different signal types across different packets within a flow to gather diverse telemetry. Generally, the feedback frequency can support round-robin signal collection without impacting congestion control fidelity. To support future extensibility, its specification reserves specific signal type values for future use, such as defining new congestion signals or alternative aggregation functions.

Fixed-size, Compare-and-replace header. The CSIG tag is fixed-size, imposing a constant, negligible bandwidth cost (e.g., < 0.2% for 4k/9k MTUs) and packet-per-second (PPS) overhead. This low overhead is critical, as it permits CSIG to be enabled on all data packets, rather than relying on sampling or probing. This enables direct, real-time quantification of the bottlenecks experienced by the data traffic itself. CSIG employs a fast `compare-and-replace` operation (akin to [1, 14, 28, 30]) to compute max or min aggregations across a path. Unlike INT protocols that *append* per-hop metadata, CSIG's fixed-size header does not grow based on the network diameter and does not require MTU modifications.

Layer 2 tag placement. A key design choice is placement of the CSIG tag at L2, as the last tag in the L2 header stack (just before the L3 header). This is key to the system's simplicity, compatibility, and deployability:

- *Hardware simplicity*: At L2, the tag's placement is invariant to L3 and L4 headers. This fixed-offset hardware parsing design avoids the hardware complexity of parsing and updating the tag at variable offsets, a significant challenge for L3+ INT schemes that must handle numerous L3+ encapsulations. A CSIG-capable switch or NIC thus only needs to parse L2 headers to process the tag. Moreover, headers such as IPv6 extension headers (e.g., hop-by-hop options or destination options used by iOAM) are typically processed on the "slow path" in switching ASICs to keep the fast path hardware simple and performant. Slow-path processing cannot achieve line-rate performance.
- *Tunneling & Encryption compatibility*: As an L2 tag, CSIG is transparently compatible with pervasive L2.5/L3+ tunnels (e.g., MPLS, IP-in-IP, VxLAN, Geneve)—including those used in virtualization stacks to enable guest VMs to use CSIG—and ubiquitous end-to-end encryption (e.g., PSP, IPsec, MACsec). The tag is forwarded *under* these encapsulations, obviating complex logic to copy or relocate metadata across inner and outer headers. Q-in-Q encapsulation, such as done by IEEE 802.1ad, is outside the scope of CSIG, as it is not typically used in data centers.
- *Deployability & Backward compatibility*: CSIG tag variants are small enough to fit within the packet parsing depths of commodity ASICs and do not break ECMP hashing. The compact tag's structural similarity to VLAN tags is deliberate and allows legacy devices to participate in the protocol, making CSIG uniquely brownfield-deployable (see §4.1).

These datacenter realities make L2 the only viable layer for robust, deployable signaling that bypasses L3+ ossification.

Use-case-driven, HW-friendly signals. CSIG employs simple, use-case-driven metrics—summarized in Tab. 2—to characterize path bottlenecks. These signals close a visibility gap for end hosts, which otherwise must rely on heuristics that frequently overestimate or underestimate capacity. Specifically, ABW/C enables rapid yet precise rate ramp-up in CC, ABW allows transport load balancers to select paths with high available bandwidth, and per-hop delay ensures CC responds exclusively to bottleneck queuing. ABW/C and ABW also enhance observability by revealing network headroom; unlike delay, which spikes only after saturation, available bandwidth provides a proactive signal of remaining capacity.

Further, these metrics are hardware-optimized, requiring no per-flow state and avoiding the expensive per-packet arithmetic of prior schemes [15, 28]. For available bandwidth computation, switches use a fixed, operator-configured timescale independent of the flow's RTT. The transports maintain the onus of accounting for this timescale in their control decisions, such as temporal averaging to derive a signal suitable for the flows' RTT.

Transport-friendly design. For broad transport compatibility, CSIG decouples the switch-updated in-band tag from the host-processed reflection header. This allows transport-specific logic without switch changes. Further, by carrying signals in every data packet, bottlenecks map intrinsically to flow context, eliminating the need to correlate out-of-band probes. This also enables distinct tracking of forward and reverse path bottlenecks—essential for reacting solely to forward-path congestion. The small, fixed-length tag also minimizes reflection header overhead.

4 High-precision Switch Telemetry

The foundation of CSIG is *high-precision switch telemetry (HST)*, which acquires high-resolution telemetry to capture microbursts, and updates CSIG headers within the HW latency budget for line-rate performance. The ideal realization is in ASICs such as Tomahawk 6 [6], which can implement HST and CSIG natively in the HW datapath. To deploy CSIG on existing commodity switches, we developed a compatible software-assisted version. We validated and deployed SW-assist CSIG in switches across two operational environments: a mature, proprietary switch software stack and a contemporary stack built upon SONiC [41, 51], over five switch

Signal Name	Use Cases
min(ABW/C) - Relative Available bandwidth (%)	Fast Ramp-up
min(ABW) - Absolute Available bandwidth (Gbps)	PLB
max(Delay) - Per-hop Packet delay (μs)	NSCC, Poseidon
max(nQD) - Normalized Queue depth (μs)	HPCC

Table 2: CSIG signal types & control loops they augment

ASICs spanning the Broadcom Tomahawk series [7], Cisco Silicon One series [11], and older generations of the Broadcom Trident series [8]. We first detail congestion signals in their optimal HW-native realization, followed by the SW-assist CSIG alternative. To support broad deployment across our fleet for many generations of switches, we support both versions.

4.1 HW-native vs SW-assist CSIG

Tab. 2 summarizes the HW-native CSIG signals, which provide fine-grained visibility into congestion. Native logic in new ASICs can perform per-packet computation directly in the fastpath and compare-and-replace operations on the packet header. ABW is computed per egress port using byte counters over a configurable time window (as low as 1 μs). ABW/C normalizes ABW by the egress port's capacity. Packet delay (Delay) is computed as the time spent by the packet in the device pipeline by differencing ingress and egress timestamps, excluding serialization delays. nQD is computed by measuring the instantaneous queue depth at packet dequeue and normalizing by the egress port capacity, yielding the drain time—a quantity comparable across links of different speeds. HW-native implementation supports both 4B and 8B CSIG tags.

SW-assist CSIG, in contrast, targets brownfield deployments on a large swath of commodity ASICs dating back to 2010 by decoupling signal computation from the data plane. This approach is sufficiently vendor-agnostic and proven to work in different vendor ASICs. A control-plane agent computes signals using port and queue counters and updates HW tables for the data plane to read. While this maintains line-rate tagging and minimal pipeline latency, it limits signals such as available bandwidth to coarser ~100μs intervals and precludes signals for metrics measured per-packet, such as per-hop delay (which can at best be approximated using available port/queue metrics). Furthermore, HW constraints typically restrict this approach to 4B CSIG tags with values of five bits or less to fit available TCAM space. We use 3 out of the 5 bits available for signal values due to resource constraints on switch ASICs for the SW-assist approach (§4.3). For ABW/C, we employ a non-linear range encoding (Tab. 4 in App. A) for these 3 bits: `{1,5,10,20,40,60,90,100}%`, providing higher granularity at lower values of the signal (closer to congestion). We implemented the locator to only encode the stage and link direction attributes of the bottleneck hop (e.g., ToR→Aggr), which fits comfortably within the available 7 bits and is sufficient for our use cases. We did not directly encode switch IDs or TTL since they are complex and need more bits. Tab. 5 in App. A contrasts the tradeoffs between SW-assist and the HW-native CSIG.

4.2 High-Res Signals for SW-assist CSIG

Fig. 5 illustrates the SW-assist CSIG architecture. We developed a user-space *HST Agent* that runs as a standalone process on the switch's x86 CPU, ensuring functional isolation from the core switch stack. In SONiC-based platforms, we isolate the agent within

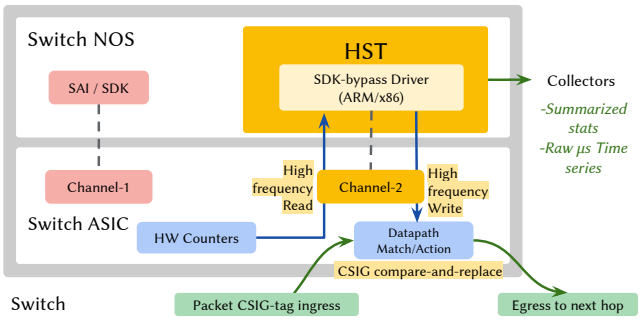

Figure 5: SW-assist CSIG architecture

a dedicated container and pin it to a single hyperthread to minimize resource contention and performance intrusion. While the agent can run on embedded ARM cores, we utilized the x86 CPU to simplify engineering and expedite integration with the existing switch stack. Prior work such as [50, 59] do not meet our requirements: (i) they rely on best-effort sampling or socket-based streaming which are prone to sampling loss and significant software overheads, and (ii) they provide only read-only unidirectional telemetry and lack mechanisms to write back to datapath tables at the same frequency. Instead, our strategy employs a deterministic poll-and-update cycle to meet CSIG's stringent low-loss, low-latency requirements.

To obtain high-resolution telemetry, we engineered a *dedicated CPU-to-ASIC hardware channel* for a deterministic, low-latency polling loop in the HST agent. This bypasses vendor SDK and SAI [40] API constraints, which are often limited to multi-second intervals that fail to capture transient microbursts. Modern ASICs provide multiple hardware channels for CPU-to-ASIC interaction. While vendor SDKs typically utilize the primary channel, we repurpose a separate, unutilized channel intended for ARM cores to establish a private, high-speed conduit that does not contend with SDK operations. We memory-map the interface's PCI base address to enable direct MMIO access to control registers, allowing the HST agent to either perform indirect HW register read/write operations, or configure the ASIC's internal DMA engine for bulk reads/writes. Crucially, this direct hardware access eliminates software overheads to enable μs-latency telemetry reads and table writes, bound only by the limits of PCIe access.

4.3 SW-assist CSIG header update datapath

The signals acquired by HST Agent must be applied to packet headers without performance loss. Each cycle, the HST agent polls counters, computes congestion signals such as *ABW*, and encodes them into compact values. Using the high-speed SDK-bypass channel, the agent performs low-latency writes to program these "signal-of-the-moment" metrics directly into hardware tables. The ASIC then references these fields for line-rate, per-packet tagging.

Our packet-processing pipeline repurposes existing ASIC capabilities by treating CSIG tags like conventional VLAN tags. By configuring the CSIG EtherType as a legitimate Tag Protocol Identifier (TPID), packets traverse standard forwarding pipelines without hardware modification. We prevent Layer 3 mechanisms from stripping the tag by disabling tag rewrites on next-hop entries. Finally, the ingress pipeline passes CSIG fields as VLAN metadata to the egress stage, where Ternary Content Addressable Memory

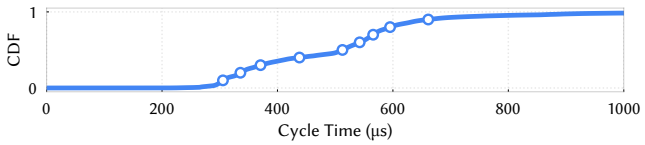

Figure 6: CDF of *ABW* intervals (HST cycle) across switch fleet

(TCAMs) or Network Processing Units (NPUs) perform compare-and-replace operations. Because CSIG updates depend on egress port congestion, they execute after port resolution in late egress stages. For each port, locator metadata is statically stored in a hardware table alongside its current signal-of-the-moment. The compare-and-replace operation then integrates this signal and locator into the CSIG header. By bypassing slow SDK operations, the cycle from counter polling to datapath updates completes in sub-millisecond time, delivering real-time feedback rather than seconds-stale data. Fig. 6 shows a CDF of HST cycle times across our production fleet. SW-assist CSIG re-purposes switch VLAN resources, potentially precluding concurrent VLAN use. We could not deploy SW-assist in certain baremetal settings where tenant isolation required use of VLANs. In contrast, HW-native implementations co-exist with VLAN tags without such limitations.

Beyond enabling deployment on brownfield hardware, this poll-and-update model also serves as a flexible framework for extending HW-native implementations with new, experimental signal types computed in the control plane (e.g., link health metrics, switch reliability signals) in both 4B and 8B formats without waiting for ASIC redesign cycles. Independently, while the user-space HST agent plays a critical role for datapath updates in SW-assist mode, it remains useful even in HW-native mode to export fabric telemetry in the form of summarized digests and raw traces (§6).

5 CSIG Realization in Transports

5.1 Using CSIG in Control Loops

We next describe how the CSIG signals provided by HST (§4) are used by congestion control, multipathing, and load-balancing algorithms to enhance performance.

CSIG Fast Ramp-Up in Congestion Control: Fast Ramp-Up (FRU) is a CC mechanism that uses CSIG normalized Available Bandwidth (ABW/C) telemetry to increase the congestion window (*cwnd*) *rapidly and safely* when available bandwidth is detected. With FRU, if only $\frac{1}{f}$-th of the bottleneck bandwidth is used then all flows can safely increase their sending rate by a factor of f over the next RTT, fully utilizing the bottleneck. This adaptive increase claims idle capacity faster than standard additive increase (AI) mechanisms, which take O(BDP) RTTs to converge [23]—increasingly a challenge for short flow completion time as we move to 400Gb/s paths and beyond. While endpoint CC has evolved from coarse-grained loss signals (Reno [26], CUBIC [23]) to more precise signals for detection of congestion, such as ECN (DCTCP, DCQCN) and delay measured by hardware timestamps (TIMELY [36], Swift [31]), most CC algorithms deployed at-scale have until now used slow, static increase mechanisms such as AI, because they lacked a signal such as ABW/C. Unlike CUBIC [23], which defaults to Reno's linear increase at low BDPs and requires hundreds of RTTs to recover from loss at high BDPs, or BBR [10], which

Algorithm 1 Fast Ramp-Up (FRU) for Swift CC

```
1:  Input: acknowledged (# packets ACKed in current ACK),
2:    tx_in_flight (flight size at time of packet transmission),
3:    fabric_delay (measured RTT), abwc (bottleneck ABW/C),
4:    current_fcwnd (current window size),
5:  Params: Q_th (queue delay threshold)
6:  function GetSwiftFcwndInc(...)
7:    ▷ Baseline Swift Additive Increase
8:    per_ack_inc ← acknowledged / current_fcwnd
9:    new_fcwnd ← current_fcwnd + per_ack_inc
10:   ▷ CSIG FRU Logic
11:   queue_delay ← fabric_delay − GetMinFabricDelay()
12:   if queue_delay < Q_th then
13:     utilization = 1.0 − abwc        ▷ bottleneck utilization as a fraction
14:     f = 1.0 / utilization           ▷ safe multiplicative growth factor
15:     est_flow_capacity ← tx_in_flight × f
16:     fru_target ← max(new_fcwnd, est_flow_capacity)
17:     fru_inc ← (fru_target − new_fcwnd) × per_ack_inc
18:     new_fcwnd ← new_fcwnd + fru_inc
19:   end if
20:   return new_fcwnd
21: end function
```

caps growth at 1.25× per RTT, FRU leverages precise ABW/C feedback to safely and rapidly claim newly idle bottleneck capacity and outpaces these heuristics.

In "clean-slate" protocols such as XCP/RCP, switches used terms akin to ABW/C to compute flow window/rate changes based on available bottleneck capacity, while hosts sent flow RTTs in headers for stable behaviors. Like VCP, CSIG FRU is the dual solution: switches report bottleneck utilization signals, and host CC uses this in concert with a flow's RTT and application demand.

FRU can be integrated into any CC algorithm; we have integrated it into Swift [31] (see Algorithm 1). Standard Swift CC grows the fabric congestion window (*fcwnd*) by a constant 1 packet per RTT by incrementing the window by *per_ack_inc* upon each ACK (Lines 8–9). FRU augments this baseline with an adaptive growth component. By pairing the bottleneck ABW/C reported in an ACK (*abwc*) with the flight size recorded at the corresponding data packet's transmission time (*tx_in_flight*), FRU estimates the flow's share of the bottleneck's capacity, *est_flow_capacity* (Lines 13–15). It finds an *fru_inc* for this ACK to smoothly grow *fcwnd* to *fru_target* over the next RTT (Line 17). For example, when a flow with *fcwnd* = 10 and *tx_in_flight* = 10 receives an ACK for 5 packets (*per_ack_inc* = 0.5) carrying an *abw/c* value of 50%, it will estimate a capacity of 20 packets; FRU adds 5 ((20 − 10) × 0.5) to the baseline increase for this ACK, smoothly driving *fcwnd* to the target over a single RTT.

Finally, to account for switch ABW/C updates that may occur at intervals longer than the RTT, FRU applies queuing thresholding (Q_{th}) as a real-time circuit breaker (Line 12). These guards allow the sender to pause an increase if stale ABW/C feedback risks an overshoot of actual available bandwidth. They maintain FRU's responsiveness, even under delayed explicit signaling.

FRU works similarly with other CC algorithms: §B.2 describes FRU for CUBIC (BBRv3 would be similar), and §B.1 demonstrates how FRU works with packet-sprayed NSCC.

CSIG-Enabled Dynamic Multipathing degree: Transports such as Falcon [49] and PonyExpress [34] employ multipathing with a

static number of flows per connection. This generally improves bandwidth utilization and flow completion times, but is counterproductive in incast, where the bottleneck is the (single) ToR-to-Host link. In such cases, a higher multipathing degree exacerbates last-hop queues without boosting throughput, adding unnecessary latency. We use CSIG to implement *dynamic multipathing*, by monitoring the fraction of CSIG bottleneck locator metadata fields pinpointing the last hop. When this fraction exceeds a configured threshold, indicating last-hop saturation, the sender cuts its multipathing degree—"draining" subflows to reduce last-hop queuing delay. This reduces tail latency by 2.5× in experiments (§7.2).

CSIG-informed Protective Load Balancing: Fundamentally, load balancing seeks paths with high per-flow available bandwidth. Standard Protective Load Balancing (PLB) [44] relies on indirect signals—triggering random reroutes when RTT or ECN indicates congestion for N consecutive RTTs. However, in highly loaded networks, such rerouting can blindly move flows from one congested path to another.

We find that rerouting is better served using direct ABW signals. First, CSIG-informed PLB redefines the congestion trigger: a path is deemed congested if its bottleneck ABW drops near zero for N consecutive RTTs. This prevents congestion control from suppressing the trigger even if it keeps delays low. Second, it replaces random rerouting with a *power of two choices* [3, 37] strategy. The sender evaluates two candidate paths by switching data packets between two different flow labels over consecutive RTTs, then migrates the flow to use the label with the higher ABW. This requires minimal additional state at the sender. This direct approach reduces path switching by 3.75× and increases throughput by 16% in experiments (§7.2).

Network Signal Congestion Control (NSCC): Integrating CSIG into Ultra Ethernet Transport's NSCC [5, 12] serves as a test of generality for unseen, packet-spraying transports. Implementation details are in §B.1. We focus here on correcting a structural bias common to delay-based protocols: the reliance on cumulative whole-path queuing delay. This penalizes flows traversing multiple congested hops, even when they share the same bottleneck. By substituting NSCC's cumulative signal with CSIG's `max(Delay)`, flows adapt to the true bottleneck, and decouple rate adaptation from delay elsewhere, a change shown to restore fairness (Poseidon [55]). §7.3 shows the resulting improvements in NSCC.

High Precision Congestion Control (HPCC): We demonstrate CSIG's generality by adapting HPCC [33], an INT-enabled protocol. This integration challenges CSIG's fixed-length design against two INT assumptions: whole-path visibility and multi-signal aggregation. First, while INT collects telemetry from every hop, HPCC's control algorithm acts strictly on the maximum congestion metric along the path. CSIG's bottleneck-only signaling is thus functionally equivalent for control purposes, validating that full path visibility is often superfluous. Second, HPCC relies on simultaneous measurements of queue length and link utilization. Since CSIG carries only a single signal type per packet, we adapted the sender to interleave requests for distinct signal types across alternating packets. This approach successfully drives HPCC's multi-variable control loop, demonstrating that CSIG's compact header can support multi-signal protocols. §B.3 details this implementation, showing the resulting equivalence in control fidelity. Additionally, CSIG-based HPCC outperforms "vanilla" INT-based HPCC, as it receives in-band signals on every packet rather than one probe per RTT.

5.2 CSIG End Host & NIC Implementations

This section details the implementation of CSIG in datacenter software and hardware transports (Linux TCP, PonyExpress, Falcon, RoCE) spanning four NIC platforms (a proprietary 100Gb/s NIC, 200Gb/s Intel IPU E2100 [24], 400Gb/s Intel IPU E2200 [25], and NVIDIA ConnectX-7 [38]). We also validated CSIG-tagging implementation in the Andromeda [13] virtualization stack.

CSIG for Linux TCP, PonyExpress: We integrated CSIG into our production software transport stacks—Linux TCP and the user-space stack PonyExpress [34]—by implementing an L2 tag for forward-path signaling and an L4 reflection header for the reverse path. See Fig. 7 for an overview. Our design addresses two critical challenges. First, for performance and feature integrity, the design ensures that new headers do not regress high-performance features nor break hardware offloads. Second, because these stacks serve production traffic in heterogeneous environments with varying host and switch capabilities, our design ensures reliable operation and interoperability with legacy infrastructure.

Shared aspects of PonyExpress and Linux TCP CSIG:

- *Negotiating CSIG:* Connection handshakes negotiate CSIG capabilities, and only use CSIG tagging and reflection upon mutual agreement between endpoints. This allows CSIG-enabled hosts to interoperate safely with straggler host SW versions & transparent upgrades [34].
- *Reliable Connectivity*: We use several layers of protection to prevent blackholing due to heterogeneous mixes of CSIG and non-CSIG hosts/switches. First, we configure transport connections to use CSIG only for intra-fabric traffic in fabrics for which all switches are wholly CSIG-enabled. Our Software Defined Networking (SDN) system, Orion [18], configures and checks for CSIG-compliance, and switches are programmed to strip CSIG tags when forwarding packets to neighboring switches that don't support CSIG. This ensures a set of verified-CSIG-compliant fabrics, and that CSIG is only used within these fabrics. Second, to protect against blackholing by devices that might drop packets due to bugs, transport *Protective Re-route* [56] repaths upon each Retransmission Timeout (RTO), repeatedly trying to find a path with end-to-end CSIG support. Finally, if an CSIG-enabled connection experiences k consecutive RTOs, it reverts to untagged packets for the remainder of its lifetime. This prioritizes reliability over CSIG telemetry. See App. D for details about the CSIG rollout.
- *Flexible Processing*: For backwards compatibility with devices running legacy/straggler SW, the Rx and Tx packet processing logic supports packet headers with or without L2 CSIG tags and L4 reflection headers over the same flow.
- *Dual state*: Given bidirectional flows with ACKs and Selective ACKs (SACKs), the transport has dual per-flow state: forward CSIG to track the outgoing path bottleneck (local→remote) and reverse CSIG to track the incoming path (remote→local). This

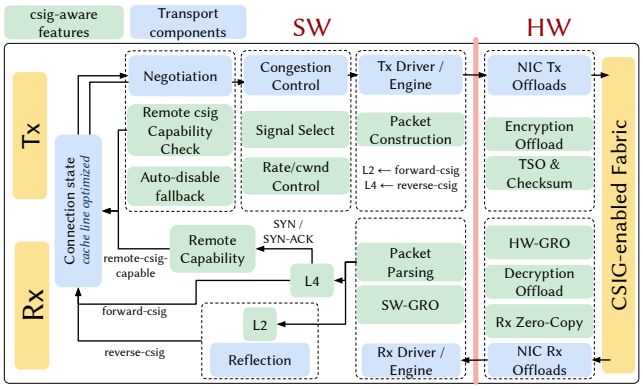

Figure 7: CSIG realization in SW Transports

enables using forward-path feedback for CC/LB while saving reverse-path data to set L4 reflection headers.

- *Reflection Mechanism:* The receiver's transport reads the L2 CSIG value, caches it in flow state, and reflects it in an L4 header on outgoing ACKs. Senders then read this reflected value and pass it to the active CC module.
- *Support with Encryption:* To support PSP encryption, we adapt NIC transmit and receive packet processing (steering rules and packet parsers) so that NIC cryptographic offload engines apply encryption and decryption at offsets adjusted for the CSIG tag. Since the reflection header is within the encrypted L4 data, it does not affect these offsets. Upon reception, the NIC extracts the CSIG tag and exposes it to transports through Rx descriptors.
- *Zero-copy Support:* CSIG support preserves zero-copy Tx and Rx optimizations. Rx Zero Copy relies on *header-splitting*, where headers are kept in host-stack buffers and the payload is placed in application buffers. The logic must account for the variable size of the L2 header due to the optional CSIG tag. The small size of the CSIG tag and reflection header remains within the header-split sizes supported by commodity NICs, enabling Rx zero-copy.
- *Cache-sensitive layout*: We optimized the layout of CSIG state to fit in already-hot cache lines, avoiding cache miss latency penalties and sustaining line-rate performance.

Specific to kernel/TCP: We integrated CSIG into NIC firmware and Linux's TCP stack, maintaining HW/SW offload performance:

- *CSIG-Aware GRO*: High-performance TCP stacks rely on Generic Receive Offload (GRO) to coalesce sequential, in-order packets belonging to the same TCP flow. However, CSIG-tagged packets within a single flow may carry varying signals, causing standard GRO to skip coalescing. To address this, we modified both kernel software GRO and NIC-offloaded hardware GRO to be CSIG-aware, coalescing packets with differing CSIG values while preserving the signal from the most recent segment. This maintains offload efficiency while giving the transport the freshest bottleneck information. We found that the configurability of existing commodity NICs and IPU-based Smart NICs [24] was sufficient to implement this.
- *TCP Offloads*: Inserting an CSIG tag at L2 introduces a variable offset for subsequent headers, potentially breaking hardware offloads such as Checksum, Receive Side Scaling (RSS), and Protocol Decryption. We addressed this while avoiding vendor-specific dependencies: (i) For foundational NICs with limited programmability or with TCAM-based steering, we updated the microcode to recognize the custom CSIG EtherType. The parsing logic dynamically adjusts the pointers used by the decryption and RSS engines, ensuring that L3/L4 headers are correctly located. (ii) For IPU-based Smart NICs, we use HW processing to decouple the wire format from the host driver. The HW strips the L2 CSIG tag before the packet is DMA'd to the host, copying the signal directly into the Rx descriptor. This simplifies the kernel, as it processes standard Ethernet frames while reading CSIG signals separately via the Rx descriptor.

CSIG for Falcon: CSIG is integrated into the Falcon [49] hardware transport on the NIC to support high-performance RDMA transfers. CSIG signal selection and congestion control logic, such as Fast Ramp-Up, is delegated to the Falcon Adaptive Engine, a programmable unit that manages transport policies. Concurrently, Falcon's fixed-function hardware datapath handles line-rate CSIG tag generation and signal reflection, ensuring fine-grained, in-band congestion feedback without sacrificing the performance advantages of a hardware transport. The Falcon protocol specification v1.1 [17] defines the mapping of the Compact CSIG tag to various Falcon protocol operations. Independently, we also validated CSIG tagging and reflection in hardware on NVIDIA CX7 NICs [38] for the RoCE transport [22].

CSIG for Virtualization Stacks: We integrated CSIG into Andromeda [13] to capture telemetry for VM-to-VM traffic. Andromeda inserts CSIG tags via standard Match-Action flow table operations during encapsulation (e.g., PSP, GRE, IP-in-IP) with negligible overhead, keeping the guest OS unaware of the signaling. To extract telemetry, we leverage ingress packet sampling at Rx hosts. This is an ideal vantage point, as receiver packet headers contain CSIG tags capturing path bottleneck signals. These samples flow into Andromeda's monitoring backend to provide real-time insights into Cloud traffic bottlenecks. Furthermore, this design can be extended to support guest OS transports transparently: at Tx, the virtual switch copies the guest's inner L2 tag to the outer L2 header during encapsulation so that transit switches can update them. At Rx, the virtual switch copies the updated outer L2 tag back into the inner L2 header or descriptor fields of the decapsulated packet before delivering it to the VM, allowing the guest OS transport to receive bottleneck signals and perform reflection transparently.

6 Bottleneck observability for applications

The transport stack's flow telemetry tracks end-to-end metrics such as retransmission rates, RTTs, and congestion windows, but it cannot localize the specific fabric bottlenecks driving application tail latency. Conversely, state-of-the-art datacenter observability frameworks use out-of-band, switch-centric telemetry to characterize fabric congestion, but face fundamental limitations in metric resolution and metric-to-flow attribution. Recent studies [4, 21] analyze minute-level switch counters to infer core RDMA congestion patterns post-hoc or characterize ToR hotspots. These approaches fail to capture microbursts *at the bottleneck* that determine performance; moreover, their port-centric metrics are application-blind, requiring complex offline correlation to associate network behaviors with specific workloads.

CSIG resolves these limitations by surfacing precise bottleneck data that can be interpreted *in application context* across three telemetry planes—RPC, SDN, and fabric—enabling operators to trace the causal link between application latency spikes and transient microbursts in fabric.

Remote Procedure Call (RPC) telemetry: Distinguishing between host inefficiencies and fabric congestion remains a fundamental challenge in RPC systems, turning the root-causing of elevated tail latency into a laborious process of manual correlation between host and fabric data sources. To resolve this, we surface CSIG signals through Fathom [45], which annotates distributed traces in Dapper [48] with transport-layer metrics. Since a single RPC consists of multiple packets, we record the most recent signal and locator observed during the RPC's lifetime. By including CSIG data in the traces, operators can directly correlate high RPC latency with specific network bottlenecks. Annotations such as `CSIG{ABW:<5%, loc:ToR→host}` quickly pinpoint whether the network was a bottleneck for a slow RPC and, if so, exactly where the congestion occurred and its severity. This allows for rapid exoneration of the network when bandwidth is plentiful or provides a clear location for investigation when congestion is indicated, directly linking application performance to real-time network conditions. We evaluate this capability in §7.1.2, where CSIG distinguishes the specific network bottlenecks affecting storage read vs. write RPCs.

Software Defined Networking (SDN) telemetry: Our host-based SDN telemetry system generates detailed traffic matrices across aggregates, such as racks, aggregation blocks, clusters, or accelerator pods, and provides granular breakdowns by specific applications. To construct these matrices, the system utilizes packet sampling at ingress and egress sides of hosts with probabilities ranging from 1:1K to 1:1M. By reporting headers up to L4 alongside timestamps and application metadata to centralized collectors, the system accurately estimates traffic volumes and maps end-to-end connectivity across the fabric.

We integrated CSIG into this infrastructure to generate *bottleneck matrices*, providing per-application visibility into where and to what extent workloads experience congestion. For ingress samples, the receiver extracts CSIG signals directly from the L2 packet header. For egress samples, the sender attaches the reflected forward-path congestion data, typically maintained as per-flow state. We aggregate these samples over fixed intervals to map the distribution of bottleneck locations and available bandwidth for each source-destination pair. These matrices enable operators to identify network-constrained workloads and optimize job scheduling & placement. §7.1.2 shows how bottleneck matrices capture structural shifts in congestion following network optimizations.

Fabric telemetry: Complementing workload-centric SDN and RPC telemetry, we surface signals computed by HST Agent (§4.2) to capture the network's perspective. This third level of analysis enables deep-dive investigations once CSIG in RPC and SDN telemetry identifies a bottleneck's location and severity. To balance fine-grained visibility with overheads, a two-tiered architecture exports CSIG signals of port utilization, *(1 - ABW/C)*, and queue depths. First as *continuous summarized statistics*, switches export bucketed histograms and cumulative digests of port utilization, e.g., time spent at >95% utilization, over 30s windows via gNMI [42] streaming telemetry. This enables operators to detect fabric-wide microbursts and anomalies without the bandwidth overhead of raw time-series data. Second as *on-demand raw traces*, to facilitate precise forensic analysis of transient events, switches maintain a circular in-memory buffer (in tmpfs) that stores the most recent ~30s of raw, sub-millisecond timeseries data. This buffer is managed by the user-space HST agent, decoupled from whether CSIG is realized via software-assist or native hardware. These high-fidelity traces of CSIG signals are retrieved on-demand via a new orchestration system. This system coordinates synchronized trace collection across switches for a desired duration, time-aligning and stitching together the ring-buffer segments—both across switches and across time by chaining multiple 30s buffers from the same switch—to reconstruct the exact micro-dynamics of congestion events. §7.1.2 demonstrates using the summaries for fabric-wide analyses, and raw traces for diagnosing ML training slowdowns.

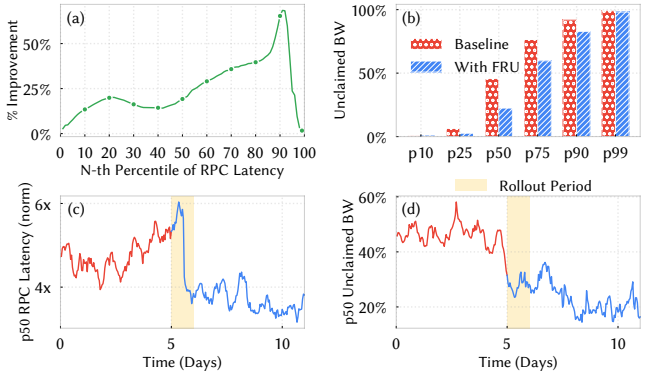

Figure 8: Impact of FRU in production for a distributed inference workload. Model server RPC latency p50 improves by up to 20%. Unclaimed bandwidth reduces with FRU by up to 60%.

7 Evaluation

We evaluate CSIG across three dimensions: production results covering congestion control and observability at scale; testbed results demonstrating load balancing performance and validating implementation overheads; and simulation results for sensitivity analysis. All results are with the 4B tag, unless otherwise specified.

7.1 Production Results

We evaluate CSIG in a large-scale production deployment comprising hundreds of clusters and handling intra-datacenter traffic on our Jupiter [43] network. The system serves a heterogeneous mix of latency-critical and throughput-oriented workloads carried over Pony Express and Linux TCP. These results demonstrate CSIG's efficacy for real-world compute, storage, and ML applications under dynamic traffic conditions of production fabrics.

7.1.1 Congestion Control benefits. We evaluate CSIG Fast Ramp-Up (FRU) for Swift [31] CC in production. To account for natural traffic variations (diurnal and weekly), we compared performance against the same time period from the week prior to rollout. We also analyzed the relevant timeseries spanning ±5 days around the rollout to confirm that gains were sustained and not transient.

Fig. 8(a) shows that FRU improves RPC latency across all %-iles for latency-sensitive *distributed LLM serving traffic*. The most significant gains occur between the 20th and 90th %-iles–including a

Cluster	Median unclaimed BW	FRU's impact on Median RPC Latency
Cluster A	~ 90% (high)	~20% improvement
Cluster B	~ 40% (medium)	~10% improvement.
Cluster C	~ 15% (low)	< 1% improvement

Table 3: Impact of FRU on cluster-wide median RPC latency in relation to the cluster's unclaimed BW.

~20% improvement in the median. Fig. 8(c) shows the time series evolution of median RPC latency over hourly windows during the rollout. These gains translate to faster prefill-to-decode KV-cache transfer time which is critical in disaggregated LLM serving. We compared intervals with identical (within 1%) serving QPS and prefill lengths and found that FRU reduces median KV-cache transfer time by 5.8%, and by 9.3% under peak QPS load (not shown). We further verified these gains are from FRU by confirming that the natural variation (A/A and B/B controls) for equivalent load conditions was within 0.5%. Similarly, a large-scale *In-Memory Filesystem* which exhibits bursty shuffle phases achieved a 10% median RPC latency improvement (not shown) from FRU. Deep tails such as 99th %-ile do not exhibit noticeable improvement; this is expected since the tail is determined by congested scenarios where available bandwidth is zero and FRU is not expected to trigger.

To understand these gains, we analyzed *unclaimed bandwidth* —defined as the ABW/C reported by CSIG during congestion control induced stalls i.e., when application has data to send but is *fcwnd* limited. This captures the available bandwidth that congestion control failed to utilize. This metric is the first direct measurement of how efficiently congestion control uses available bandwidth, which as [31] noted, has traditionally been much harder to measure than its ability to mitigate congestion. Fig. 8(b) compares the distribution of unclaimed BW measured by the application flows. It shows that FRU reduces this waste by 60% (30 percent points) at p50, converting idle capacity into useful throughput. Fig. 8(d) shows the time series evolution of this median over hourly windows tracking this reduction. This improvement is further supported by a 12% reduction in transport stalls and an upward shift in the congestion window distribution—22% at p50 (not shown)—allowing flows to maintain higher rates. This aggressive rate causes a marginal increase in RTT (<10% at p99.9) due to increased queuing in the switches, however, we trade this acceptable increase for significant gains in application latency.

We further analyze the efficacy of FRU across clusters with varying levels of overall network utilization, shown in Tab. 3. Our takeaway is that FRU benefits clusters with medium-to-high unclaimed bandwidth but offers minimal utility when unclaimed bandwidth is low. FRU primarily exploits unclaimed bandwidth that baseline additive increase is too conservative to claim. Thus performance improvements are highest when spare bandwidth is plentiful but naturally smaller otherwise. These observations also validate that FRU correctly scales back to not induce congestion under already heavy load.

7.1.2 Observability Case Studies. While metrics such as RTT indicate that congestion *exists*, they rarely reveal *where* or *why*. We use production case studies to evaluate how CSIG bridges this gap.

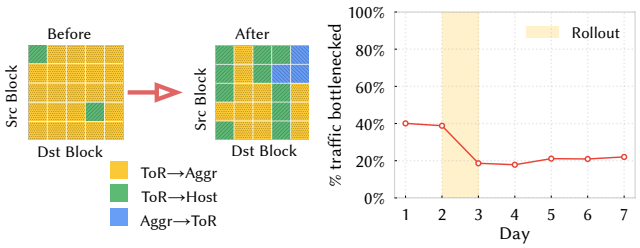

Figure 9: Impact of enabling multipathing for an in-memory filesystem job: (a) shift in primary bottleneck locations, and (b) reduction in traffic bottlenecked (ABW/C < 5%) at ToR uplinks.

Visualizing Congestion Shifts (SDN Telemetry): Using CSIG, we evaluated the production rollout of a new multipath load balancing scheme in Pony Express. Fig. 9 presents a bottleneck heatmap constructed from CSIG packet samples, capturing the *most frequent* bottleneck layer encountered by bandwidth-limited packets (ABW/C <5%) between each pair of src/dst compute blocks. Prior to the change, the bottlenecks were concentrated at ToR→Aggr uplinks. After enabling multipathing, the fraction of traffic bottlenecked at ToR→Aggr uplinks dropped from 40% to 20% (visualized in Fig. 9a and quantified in Fig. 9b), shifting the residual congestion to the other layers of the topology—primarily to the ToR→host last hop. While standard byte counters recorded aggregate throughput gains, CSIG revealed *why* this was happening. It proved that the throughput increase happened because the ToR→Aggr uplink bottleneck was relieved.

Contextual Network Hotspots (RPC Telemetry): We evaluated how CSIG advances bottleneck observability for RPCs by analyzing over 100 million TCP-driven RPC samples across dozens of storage-heavy production clusters.

Fig. 10 shows the correlation between storage HDD Read & Write latency (both total and the network-attributed portion) and CSIG-reported available bandwidth % recorded in Dapper annotations. The data confirms that Write latency is dominated by network transfers, and increases exponentially as available bandwidth drops. In contrast, Read latency is more governed by non-network factors such as disk I/O or host processing and is only linearly sensitive to the available bandwidth even when it is low.

Furthermore, CSIG confirmed that Reads and Writes are bottlenecked at different layers in the network. Fig. 11 shows the distribution of bottleneck locations and message size breakdowns for bandwidth-starved RPCs (ABW/C < 1%). Reads are primarily bottlenecked at ToR uplinks, while Writes are bottlenecked at the core layer Aggr → DCNI (Data Center Network Interconnect). By correlating these signals with message size, we found that core-layer congestion is driven by large Write payloads (>1 MB).

With CSIG, we performed this analysis using only Dapper traces, removing the need to correlate multiple telemetry sources. By pinpointing *where* congestion occurs for Reads vs. Writes, CSIG identified the exact network layers that require more capacity.

Diagnosing ML training slowdowns (Fabric telemetry): Large-scale ML training is highly sensitive to tail latency, as a single straggler can stall the entire job. We evaluate how CSIG aids in detecting and debugging the transient anomalies that cause these stragglers. Fig. 12 shows the raw μs-granularity traces of port utilization from CSIG fabric telemetry gathered from a ToR switch

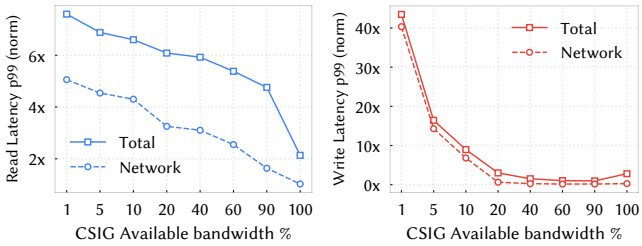

Figure 10: Storage RPC Latency (norm.) vs. Available Bandwidth.

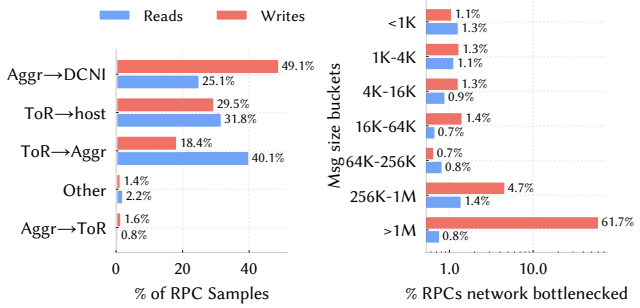

Figure 11: Bottleneck location and msg size distributions for BW-Starved RPCs (<1% ABW).

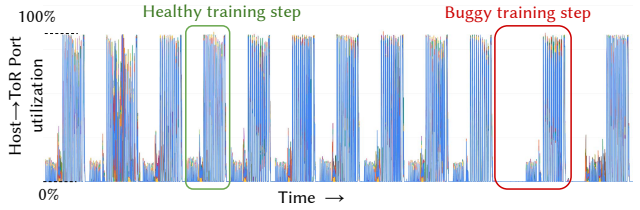

Figure 12: CSIG raw μs-granularity traces from a ToR switch serving an ML training job.

during a production TPU training run, where each color represents a unique host→ToR port. It reveals the unique network signature of the TPU training job, and we can precisely demarcate individual training steps. The traces detected an instance of an anomaly that coarse-grained health metrics such as steps/s missed: a prolonged stall between two specific training steps as seen in the idle section of the red box. By correlating the stall timestamp with logs from the Inter-chip Interconnect (ICI), we confirmed that a simultaneous physical link flap caused this stall.

Fabric-level congestion dynamics. To evaluate how CSIG enables large-scale congestion pattern analysis, we compared fabric telemetry from production storage and TPU environments. Density heatmaps in Fig. 13 summarize these distinct link dynamics over a two-day period. For every network link, we aggregate two metrics over identical five-minute windows: the Average Link Utilization (x-axis) and the Line-rate Microburst Duration (y-axis), representing the percentage of time the link spent at >95% saturation based on HST's 100μs measurements. These values are bucketed into integer bins, and heatmap shows the count of observations for each bin across the fabric. First, the two fabrics visibly exhibit different congestion signatures. Second, although both fabrics exhibit bursts at low avg utilization, the TPU fabric's isolated central patch (~50% avg utilization, ~50% microburst duration) indicates a larger prevalence of burst/idle cycles that's characteristic of ML workloads.

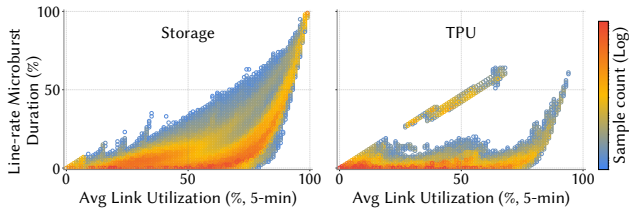

Figure 13: 100us cumulative line-rate burst duration (HST) vs 5-min Avg utilization across all links in a storage & TPU fabric

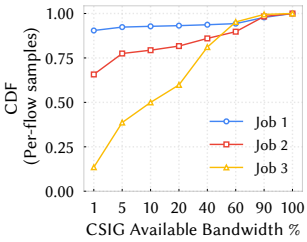

Figure 14: CSIG ABW/C CDF for jobs in TPU fabric.

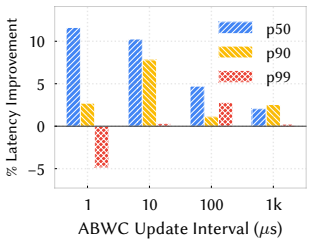

Figure 15: Effect of ABW/C measurement interval on op latency

Fig. 14 uses SDN telemetry to translate these fabric-level dynamics into application impact. It shows the CDF of ABW/C experienced by flows of three specific jobs in the TPU fabric. Despite sharing the same physical fabric, distinct jobs experience different available bandwidth. For example, 90% of Job 1's flows were severely bottlenecked by the bursty links identified in Fig. 13, while Job 3 is less impacted. This occurs because job placement determines which flows use or do not share the bursty links. CSIG not only characterizes fabric congestion dynamics but also identifies the specific jobs affected, which more precisely informs job scheduling decisions. This highlights CSIG's unique ability to isolate application-specific bandwidth availability, a dimension distinct from the per-RPC metrics shown earlier.

7.2 Testbeds & Controlled Experiments

To evaluate load balancing in a controlled setting, we used a *Production Mirror Testbed*, comprising a Jupiter network with two aggregation blocks and dozens of racks. This setup enables running application benchmarks against specific traffic patterns.

CSIG-Enabled Dynamic Multipathing degree. We evaluated dynamic multipathing (§5.1) using an incast scenario, where eight hosts saturated a 100 Gbps ToR downlink. As shown in Fig. 16 (left), static multipathing with a degree of 16 inflated p99 latency to 222ms due to increased last-hop queue pressure. In contrast, CSIG-guided dynamic multipathing correctly identified the ToR→host bottleneck and disabled multipathing, lowering p99 latency to 89ms—a 2.5× reduction. This approach matches single-path performance during incast while retaining the ability to scale up when a different layer of the fabric is the bottleneck.

CSIG-informed Protective load balancing. We evaluated CSIG-informed PLB (§5.1) using two racks, each with 12 machines and 8×100 Gbps ToR uplink capacity. Under a heavy 730 Gbps background load, a single backlogged application flow achieved only 31.4 Gbps in the baseline scenario. Because standard PLB relies on indirect signals (RTT/ECN) to trigger reroutes that are blind to the available bandwidth on the path, it suffered from path thrashing—where the flows were unnecessarily rerouted (45 times/s)

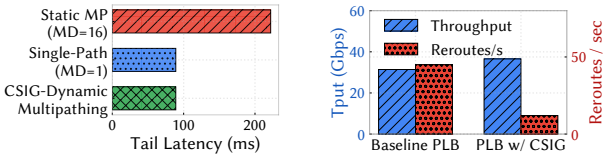

Figure 16: Performance improvements to Multipathing (left) and PLB (right) using CSIG

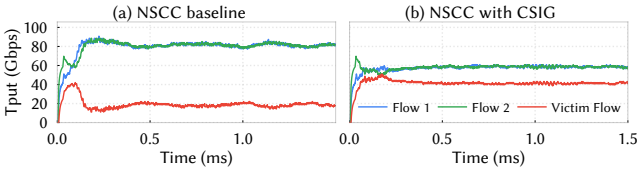

Figure 17: Bandwidth Fairness in Baseline NSCC & NSCC+CSIG.

from one path with no available bandwidth to another. By leveraging direct ABW signals to actively find paths capable of absorbing the load, CSIG-informed PLB directed traffic to less congested paths only when actually available. This reduced the reroute frequency to 12 reroutes/s and increased application throughput to 36.5 Gbps—a 16% improvement.

HW-native and SW-assist CSIG overheads. We validated CSIG overheads across four NIC generations, five host stacks (Pony Express, TCP, Falcon, Virtualization, RoCE), and five switch silicon generations —including Intel IPU E2200 and Broadcom Tomahawk 6 (first devices with HW-native CSIG). Adding headers to all packets incurs negligible overhead on goodput (<0.1%) and <0.01% impact on packets per second or latency even for 1B messages (with both 4B and 8B formats on TH6).

7.3 Simulations with CSIG

FRU Sensitivity Analysis. To evaluate how the switch's measurement interval for the ABW/C signal affects FRU's performance, we use Isekai [16] to simulate foreground incast RPCs competing with bursty background traffic, where all flows traverse inter-rack paths with a 10μs unloaded RTT. FRU captures capacity released between background bursts for foreground flows. Fig. 15 shows how FRU's median, p90, and p99 latency improvements (over baseline Swift without FRU) for foreground RPCs change as we vary the ABW/C measurement interval. For these homogeneous 10μs-RTT flows, FRU offers performance benefits for ABW/C measurement intervals between 10–100μs.

These results illuminate a key trade-off. When the ABW/C measurement interval is too short, ABW/C values may include sharp transients that do not reflect spare capacity that persists long enough to be claimable by end-host congestion control. Such transients can be caused by sub-RTT-burst-generating processes, such as CC operating unpaced while window limited. Transients in the 1μs ABW/C values cause the latency regression at p99 and the reduced improvement at p90 *vs.* for a 10μs interval. Lengthening the ABW/C measurement interval avoids exposing these transients. However, averaging over a longer period causes FRU to respond more slowly, both at the onset of spare capacity and once spare capacity has been claimed. This effect is clearly evident in the latency trend across the 10, 100, and 1000μs intervals. In production datacenter settings, where flow RTTs are heterogeneous, we expect that an ABW/C measurement interval in the range of 10 to 100μs will likely be adequate for FRU. We leave a full investigation of the best choice to future work, and offer details of how the choice of ABW/C measurement interval affects FRU's control loop in §B.4.

We further consider the role of the frequency of feedback, as determined by the fraction of a flow's data packets carrying a CSIG tag, on congestion control's performance. Fig. 21 in §B.3 demonstrates that for the vast majority of flow sizes, *flow completion time is shortest with per-packet feedback, vs. with less frequent feedback*. Per-packet feedback allows HPCC to learn of incipient queue buildup and link saturation earlier, so its control loop responds to incipient congestion earlier.

NSCC. We modified NSCC implementation in *htsim* [53] to use CSIG's bottleneck-only `max(Delay)` signal. In a canonical three-flow fairness scenario [55], where a victim flow traversing two congested queues competes with flows traversing only one, NSCC+CSIG improved the victim/non-victim throughput ratio from 4.5:1 to 1.4:1 (Fig. 17), significantly mitigating the unfairness of whole-path signals. Moreover, for ML collective-like traffic, NSCC+CSIG using packet spraying for multi-path load balancing increased victim throughput by ~65% and reduced FCT by ~39%. Simulation setups appear in §B.1.

8 Conclusion

CSIG bridges the long-standing gap between high-level application demands and low-level network dynamics. By delivering precise, μs-granularity bottleneck signals via an L2 protocol, we move beyond the limitations of coarse, black-box heuristics. Our deployment across five generations of silicon proves that high-fidelity signaling is achievable at line rate and transformative for transport performance and observability. Future extensions can evolve CSIG into a more comprehensive in-band signaling substrate by expanding the framework to expose network health and reliability signals directly to end-hosts.

Acknowledgments

We would like to thank David Wetherall, Jeff Mogul, the anonymous SIGCOMM reviewers, and our shepherd for their valuable feedback on the paper.

CSIG represents a multi-year effort at Google and across our industry partners. This work was made possible by the contributions of numerous individuals across Google's networking teams, including but not limited to: Ahmed Awad, Anthony Rebello, Arjun Singhvi, Arpitha Raghunandan, Augy StClair, Carl Mauer, Chris Alfeld, Dal Chand Choudhary, Elias Ahmed, Haseeb Ashfaq, JK Lee, Jeff Tikkanen, Kevin Yang, Meytal Pikin, Mike Beresford, Naoshad Mehta, Nipen Mody, Peixuan Gao, Praveen Kumar, Ryan Frappier, Sachin Menezes, Sagarika Sharma, Shijith Chempeth, Steven Sun, Volkan Mutlu, Willem de Bruijn. We also thank Jeffrey Ji, Liangcheng Yu, Moshe Wagner, Tyler Griggs, Yongzhou Chen for their contributions while at Google.

We thank Broadcom, Intel, Cisco, and Nvidia for collaborating with us to engineer CSIG support in their hardware platforms. We thank Jai Kumar and the Ultra Ethernet Consortium (UEC), and the Open Compute Project (OCP) community for their partnership in establishing CSIG as an open standard.

References

[1] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. 2014. CONGA: Distributed Congestion-Aware Load Balancing for Datacenters. In *Proceedings of the ACM SIGCOMM 2014 Conference*, Vol. 44. Association for Computing Machinery, New York, NY, USA, 503–514. doi:10.1145/2740070.2626316

[2] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference* (New Delhi, India). Association for Computing Machinery, New York, NY, USA, 63–74. doi:10.1145/1851182.1851192

[3] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. 1999. Balanced Allocations. *SIAM J. Comput.* 29, 1 (1999), 180–200. doi:10.1137/S0097539795288490

[4] Hamid Hajabdolali Bazzaz, Yingjie Bi, Weiwu Pang, Minlan Yu, Ramesh Govindan, Neal Cardwell, Nandita Dukkipati, Meng-Jung Tsai, Chris DeForeest, Yuxue Jin, Charles Carver, Jan Kopański, Liqun Cheng, and Amin Vahdat. 2025. Preventing Network Bottlenecks: Accelerating Datacenter Services with Hotspot-Aware Placement for Compute and Storage. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX Association, Philadelphia, PA, 317–333. https://www.usenix.org/conference/nsdi25/presentation/bazzaz

[5] Tommaso Bonato, Abdul Kabbani, Daniele De Sensi, Rong Pan, Yanfang Le, Costin Raiciu, Mark Handley, Timo Schneider, Nils Blach, Ahmad Ghalayini, Daniel Alves, Michael Papamichael, Adrian Caulfield, and Torsten Hoefler. 2024. SMaRTT-REPS: Sender-Based Marked Rapidly-Adapting Trimmed and Timed Transport with Recycled Entropies. arXiv:2404.01630v1 [cs.NI] https://arxiv.org/abs/2404.01630v1

[6] Broadcom Inc. 2025. Broadcom Ships Tomahawk 6: World's First 102.4 Tbps Switch. https://www.broadcom.com/company/news/product-releases/63146. Press Release.

[7] Broadcom Inc. 2026. Broadcom StrataXGS Tomahawk Switch Series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs. Accessed: 2026-06-03.

[8] Broadcom Inc. 2026. Broadcom StrataXGS Trident Switch Series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs. Accessed: 2026-06-18.

[9] Frank Brockners, Shwetha Bhandari, and Tal Mizrahi. 2022. Data Fields for in Situ Operations, Administration, and Maintenance (iOAM). RFC 9197. doi:10.17487/RFC9197

[10] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2017. BBR: Congestion-Based Congestion Control. *Commun. ACM* 60, 2 (Jan. 2017), 58–66. doi:10.1145/3009824

[11] Cisco Systems, Inc. 2026. Cisco Silicon One Architecture. https://www.cisco.com/site/us/en/products/networking/silicon-one/index.html. Accessed: 2026-06-03.

[12] Ultra Ethernet Consortium. 2025. Ultra Ethernet Specification V1.0.1. https://ultraethernet.org/wp-content/uploads/sites/20/2025/10/UE-Specification-1.0.1.pdf

[13] Michael Dalton, David Schultz, Jacob Adriaens, Ahsan Arefin, Anshuman Gupta, Brian Fahs, Dima Rubinstein, E. C. Causey, Krishnan Varadarajan, Harshad Priyadarshan, Ghazi Khabsa, Kan Qiu, Jean-Francois Lamy, and Amin Vahdat. 2018. Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, 373–388. https://www.usenix.org/conference/nsdi18/presentation/dalton

[14] Nandita Dukkipati, Masayoshi Kobayashi, Rui Zhang-Shen, and Nick McKeown. 2005. Processor Sharing Flows in the Internet. In *Quality of Service – IWQoS 2005*, Hermann de Meer and Nina Bhatti (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 271–285. doi:10.1007/11499169_22

[15] Nandita Dukkipati and Nick McKeown. 2006. Why Flow-Completion Time Is the Right Metric for Congestion Control. *ACM SIGCOMM Comput. Commun. Rev.* 36, 1 (2006), 59–62. doi:10.1145/1111322.1111336

[16] Falcon Transport Team. 2024. Isekai: an Event-Driven Network Simulator for the Falcon Protocol. https://github.com/falcon-transport/isekai.

[17] Falcon Transport Team. 2026. *Falcon Protocol Specification V1.1.* Specification. Open Compute Project. https://www.opencompute.org/documents/ocp-specification-falcon-transport-protocol-v1-1-0-pdf

[18] 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. 2021. Orion: Google's Software-Defined Networking Control Plane. In *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI '21)*. USENIX Association, Berkeley, CA, USA, 83–98. https://www.usenix.org/conference/nsdi21/presentation/ferguson

[19] Sally Floyd. 1994. TCP and Explicit Congestion Notification. *SIGCOMM Comput. Commun. Rev.* 24, 5 (Oct. 1994), 8–23. doi:10.1145/205511.205512

[20] Sally Floyd and Van Jacobson. 1993. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking* 1, 4 (1993), 397–413. doi:10.1109/90.251892

[21] Soudeh Ghorbani, Yimeng Zhao, Srikanth Sundaresan, Ying Zhang, Yijing Zeng, Abhigyan Sharma, Prashanth Kannan, and Cristian Lumezanu. 2025. Congestion Patterns in a Large-Scale RDMA Datacenter. In *Proceedings of the 2025 ACM Internet Measurement Conference* (USA) *(IMC '25)*. Association for Computing Machinery, New York, NY, USA, 944–951. doi:10.1145/3730567.3764494

[22] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over Commodity Ethernet at Scale. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. ACM, 202–215. doi:10.1145/2934872.2934908

[23] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a New TCP-Friendly High-Speed TCP Variant. *SIGOPS Oper. Syst. Rev.* 42, 5 (July 2008), 64–74. doi:10.1145/1400097.1400105

[24] Intel. 2025. Intel E2100 200Gbps IPU. https://www.intel.com/content/www/us/en/products/details/network-io/ipu/adapter-e2100.html. Accessed: 2026-01-10.

[25] Intel. 2025. Intel E2200 400Gbps IPU. https://www.intel.com/content/www/us/en/products/sku/246030/intel-ipu-adapter-e2200/specifications.html. Accessed: 2026-01-10.

[26] Van Jacobson. 1988. Congestion Avoidance and Control. In *Symposium Proceedings on Communications Architectures and Protocols* (Stanford, California, USA). Association for Computing Machinery, New York, NY, USA, 314–329. doi:10.1145/52324.52356

[27] Vimalkumar Jeyakumar, Mohammad Alizadeh, Yilong Geng, Changhoon Kim, and David Mazières. 2014. Millions of Little Minions: Using Packets for Low Latency Network Programming and Visibility. In *Proceedings of the ACM SIGCOMM 2014 Conference*. Association for Computing Machinery, New York, NY, USA, 3–14. doi:10.1145/2619239.2626292

[28] Dina Katabi, Mark Handley, and Charlie Rohrs. 2002. Congestion Control for High Bandwidth-Delay Product Networks. In *Proceedings of the ACM SIGCOMM 2002 Conference* (Pittsburgh, Pennsylvania, USA). Association for Computing Machinery, New York, NY, USA, 89–102. doi:10.1145/633025.633035

[29] Frank Kelly. 2003. Fairness and Stability of End-to-End Congestion Control. *European Journal of Control* 9, 2-3 (2003), 159–176. doi:10.3166/ejc.9.159-176

[30] Christopher A. Kent and Jeffrey C. Mogul. 1987. Fragmentation Considered Harmful. *SIGCOMM Comput. Commun. Rev.* 17, 5 (Aug. 1987), 390–401. doi:10.1145/55483.55524

[31] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. 2020. Swift: Delay Is Simple and Effective for Congestion Control in the Datacenter. In *Proceedings of the ACM SIGCOMM 2020 Conference*. Association for Computing Machinery, New York, NY, USA, 514–528. doi:10.1145/3387514.3406591

[32] Jai Kumar, Surendra Anubolu, John Lemon, Rajeev Manur, Hugh Holbrook, Anoop Ghanwani, Dezhong Cai, Heidi Ou, Yizhou Li, and Xiaojun Wang. 2024. *Inband Flow Analyzer*. Internet-Draft draft-kumar-ippm-ifa-08. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-kumar-ippm-ifa/08/ Work in Progress.

[33] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. 2019. HPCC: High Precision Congestion Control. In *Proceedings of the ACM SIGCOMM 2019 Conference*. Association for Computing Machinery, New York, NY, USA, 44–58. doi:10.1145/3341302.3342085

[34] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Michael Dalton, Nandita Dukkipati, William C. Evans, Steve Gribble, Nicholas Kidd, Roman Kononov, Gautam Kumar, Carl Mauer, Emily Musick, Lena Olson, Erik Rubow, Michael Ryan, Kevin Springborn, Paul Turner, Valas Valancius, Xi Wang, and Amin Vahdat. 2019. Snap: a Microkernel Approach to Host Networking. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 399–413. doi:10.1145/3341301.3359657

[35] Alberto Medina, Mark Allman, and Sally Floyd. 2004. Measuring Interactions Between Transport Protocols and Middleboxes. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement* (Taormina, Sicily, Italy) *(IMC '04)*. Association for Computing Machinery, New York, NY, USA, 336–341. doi:10.1145/1028788.1028835

[36] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-Based Congestion Control for the Datacenter. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. ACM, 537–550. doi:10.1145/2785956.2787510

[37] Michael Mitzenmacher. 2001. The Power of Two Choices in Randomized Load Balancing. *IEEE Trans. Parallel Distrib. Syst.* 12, 10 (Oct. 2001), 1094–1104. doi:10.1109/71.963420

[38] NVIDIA. 2022. NVIDIA ConnectX-7 400Gbps NIC. https://www.nvidia.com/en-us/networking/ethernet-adapters/. Accessed: 2026-06-24.

[39] Open Compute Project. 2026. Multipath Reliable Connection (MRC) Specification. https://www.opencompute.org/documents/ocp-mrc-1-0-pdf Accessed: 2026-06-24.

[40] Open Compute Project. 2026. Switch Abstraction Interface (SAI) Specification. https://github.com/opencomputeproject/SAI. Accessed: 2026-06-07.

[41] Open Networking Foundation. 2021. P4 Integrated Network Stack (PINs). https://opennetworking.org/pins/. Accessed: 2026-06-29.

[42] openconfig. 2025. gNMI - Grpc Network Management Interface. https://github.com/openconfig/gnmi. Accessed: 2025-12-30.

[43] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve 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. 2022. Jupiter Evolving: Transforming Google's Datacenter Network Via Optical Circuit Switches and Software-Defined Networking. In *Proceedings of the ACM SIGCOMM 2022 Conference* (Amsterdam, Netherlands) *(SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 66–85. doi:10.1145/3544216.3544265

[44] Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, Qiaobin Fu, Gautam Kumar, Masoud Moshref, Junhua Yan, Van Jacobson, David Wetherall, and Abdul Kabbani. 2022. PLB: Congestion Signals Are Simple and Effective for Network Load Balancing. In *Proceedings of the ACM SIGCOMM 2022 Conference* (Amsterdam, Netherlands). Association for Computing Machinery, New York, NY, USA, 207–218. doi:10.1145/3544216.3544226

[45] Mubashir Adnan Qureshi, Junhua Yan, Yuchung Cheng, Soheil Hassas Yeganeh, Yousuk Seung, Neal Cardwell, Willem de Bruijn, Van Jacobson, Jasleen Kaur, David Wetherall, and Amin Vahdat. 2023. Fathom: Understanding Datacenter Application Network Performance. In *Proceedings of the ACM SIGCOMM 2023 Conference*. Association for Computing Machinery, New York, NY, USA, 394–405. doi:10.1145/3603269.3604815

[46] Abhiram Ravi, Nandita Dukkipati, Naoshad Mehta, and Jai Kumar. 2024. *Congestion Signaling (CSIG)*. Internet-Draft draft-ravi-ippm-csig-01. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ravi-ippm-csig/

[47] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. In *Proceedings of the ACM SIGCOMM 2015 Conference* (London, United Kingdom). Association for Computing Machinery, New York, NY, USA, 123–137. doi:10.1145/2785956.2787472

[48] Benjamin H. Sigelman, Luiz André Barroso, Mike Burrows, Pat Stephenson, Manoj Gupta, and Sarasvati Shanbhag. 2010. *Dapper, a Large-Scale Distributed Systems Tracing Infrastructure*. Technical Report. Google, Inc. https://research.google/pubs/dapper-a-large-scale-distributed-systems-tracing-infrastructure/

[49] Arjun Singhvi, Nandita Dukkipati, Prashant Chandra, Hassan M. G. Wassel, Naveen Kr. Sharma, Anthony Rebello, Henry Schuh, Praveen Kumar, Behnam Montazeri, Neelesh Bansod, Sarin Thomas, Inho Cho, Hyojeong Lee Seibert, Baijun Wu, Rui Yang, Yuliang Li, Kai Huang, Qianwen Yin, Abhishek Agarwal, Srinivas Vaduvatha, Weihuang Wang, Masoud Moshref, Tao Ji, David Wetherall, and Amin Vahdat. 2025. Falcon: a Reliable, Low Latency Hardware Transport. In *Proceedings of the ACM SIGCOMM 2025 Conference* (São Francisco Convent, Coimbra, Portugal). Association for Computing Machinery, New York, NY, USA, 248–263. doi:10.1145/3718958.3754353

[50] SONiC Community. 2024. High Frequency Telemetry (HFT) High Level Design. https://github.com/sonic-net/SONiC/blob/master/doc/high-frequency-telemetry/high-frequency-telemetry-hld.md. Accessed: 2026-06-07.

[51] SONiC Community. 2025. SONiC: Software for Open Networking in the Cloud. https://sonicfoundation.dev/. Accessed: 2026-01-10.

[52] The P4.org Applications Working Group. 2020. *In-Band Network Telemetry (INT) Dataplane Specification*. Technical Report Version 2.1. P4.org. https://p4.org/wp-content/uploads/sites/53/p4-spec/docs/INT_v2_1.pdf Accessed: 2025-11-04.

[53] Ultra Ethernet Consortium. 2025. htsim: High-Performance Discrete Event Simulator for UEC Transport. https://github.com/ultraethernet/uet-htsim. Reference simulator for the Ultra Ethernet Consortium.

[54] Ultra Ethernet Consortium. 2025. Ultra Ethernet Consortium Congestion Signaling (CSIG) Specification. https://purl.org/csig/spec

[55] Weitao Wang, Masoud Moshref, Yuliang Li, Gautam Kumar, T. S. Eugene Ng, Neal Cardwell, and Nandita Dukkipati. 2023. Poseidon: Efficient, Robust, and Practical Datacenter CC Via Deployable INT. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 255–274. https://www.usenix.org/conference/nsdi23/presentation/wang-weitao

[56] David Wetherall, Abdul Kabbani, Van Jacobson, Jim Winget, Yuchung Cheng, Charles B. Morrey III, Uma Moravapalle, Phillipa Gill, Steven Knight, and Amin Vahdat. 2023. Improving Network Availability with Protective Reroute. In *Proceedings of the ACM SIGCOMM 2023 Conference* (New York, NY, USA). Association for Computing Machinery, New York, NY, USA, 684–695. doi:10.1145/3603269.3604867

[57] Yong Xia, Lakshminarayanan Subramanian, Ion Stoica, and Shivkumar Kalyanaraman. 2005. One More Bit Is Enough. In *Proceedings of the ACM SIGCOMM 2005 Conference* (Philadelphia, Pennsylvania, USA). Association for Computing Machinery, New York, NY, USA, 37–48. doi:10.1145/1080091.1080098

[58] Lisong Xu, Sangtae Ha, Injong Rhee, Vidhi Goel, and Lars Eggert. 2023. CUBIC for Fast and Long-Distance Networks. RFC 9438. doi:10.17487/RFC9438

[59] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. 2017. High-Resolution Measurement of Data Center Microbursts. In *Proceedings of the 2017 Internet Measurement Conference (IMC '17)*. Association for Computing Machinery, New York, NY, USA, 78–85. doi:10.1145/3131365.3131375

Appendices

Appendices consist of non-peer-reviewed supporting material.

A Format, Encoding and support

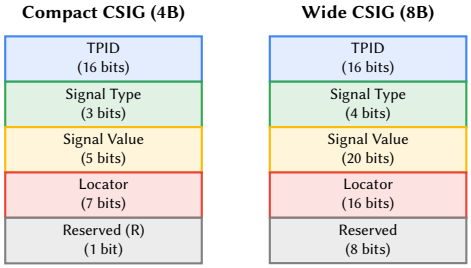

Figure 18: Compact vs. Wide CSIG packet formats

Fig. 18 shows the field-level comparison of the Compact and Wide CSIG tag formats. Compact CSIG tags employ fully configurable signal "buckets" to more efficiently represent a signal with a limited number of bits. This allows for greater granularity in specific value ranges of interest to an application. Wide CSIG tags use uniform signal quantization for higher accuracy and provide greater flexibility in defining signals and metadata with a larger bit width. In addition, its 16-bit locator field can be leveraged to carry finer-grained hop identifier. For example, it can carry the hop's uSID in SRv6 architectures, enabling deterministic path placement to avoid bottlenecks for transport protocols such as MRC [39]. Tab. 4 shows the encoding used in our deployments of ABW and ABW/C. Tab. 5 compares software-assist CSIG with hardware-native CSIG.

Signal Value	min(ABW/C)	min(ABW)
0x0	0%–1%	0–1 Gbps
0x1	1%–5%	1–5 Gbps
0x2	5%–10%	5–10 Gbps
0x3	10%–20%	10–20 Gbps
0x4	20%–40%	20–40 Gbps
0x5	40%–60%	40–80 Gbps
0x6	60%–90%	80–100 Gbps
0x7	90%–100%	>100 Gbps

Table 4: Non-linear range encoding for SW-assist CSIG signals.

Feature	SW-assist CSIG	HW-Native CSIG
Target Hardware	Brownfield / Legacy ASICs	Greenfield / New ASICs
Mechanism	Control plane HST agent	Native Pipeline Logic
Update Freq.	~100 μs	Real-time (~1 μs) / Per-packet
Tag Support	Compact (4B)	Compact (4B) & Wide (8B)
Signal Value	5-bit (Bucketized)	Up to 20-bit (Quantized)
Performance	Decoupled SW Signal Calc.	Line-rate Signal Calc.

Table 5: Comparison of SW-assist and HW-Native CSIG

B Congestion Control

B.1 NSCC CSIG Integration

max(Delay) Integration. We integrated CSIG into NSCC in UEC's reference simulator, *htsim* [53]. We extend data packets with a compact (4-byte) L2 tag carrying the `max(Delay)` signal, which receivers reflect back to senders via a 16-bit ACK header field. The sender's control loop replaces cumulative whole-path queuing delay with this bottleneck-hop `max(Delay)` signal.

NSCC balances load for a single flow using packet spraying. The CSIG-enabled NSCC sender averages *bottleneck-hop* delay samples from distinct paths with an EWMA, and so models the congestion level of a "pooled" path, leveraging CSIG's observability of congestion at a single bottleneck switch. Fig. 17 uses a 3-tier, 2:1 oversubscribed Clos fabric with 100 Gbps links, and the collective-like traffic uses a 2-tier, 4:1 oversubscribed Clos with 800 Gbps links.

Fast Ramp-Up for NSCC. We designed an FRU variant for NSCC, similar to FRU for Swift. Baseline NSCC's FastIncrease (FI) [12] rapidly increases cwnd based solely on whole-path queueing delay, ECN, or NACKs from trimmed packets, in a fashion proportional to the fabric's full unloaded BDP. However, it remains oblivious to the extent of spare bottleneck capacity. In contrast, NSCC-FRU leverages explicit `min(ABW/C)` feedback to ramp up more rapidly than FI, yet safely. Furthermore, under packet spraying across multiple paths, returning ACKs carry bottleneck capacity feedback from these multiple (possibly overlapping) paths. The NSCC-FRU sender processes these signals to adapt a single shared congestion window (*cwnd*).

We evaluated NSCC-FRU against baseline FI in htsim by simulating a large-scale incast (200–300 short 10KB flows uniformly drawn, 1 long 1 MB flow) on a 320-host, 3-tier, 100 Gbps Clos topology with a 10μs `ABW/C` measurement window. The vast majority of the last-hop link's capacity is abruptly released when the short flows end. As shown in Fig. 19, NSCC-FRU ramps the long flow's window to a full BDP faster than FI to capture this released capacity, reducing its mean flow completion time (FCT) by 10% (39μs, or ~3 RTTs), with peak reductions reaching 20%. NSCC-FRU can be combined with `max(Delay)` fairness optimizations by alternating signal types across packets. We leave the integration of these two techniques to future work.

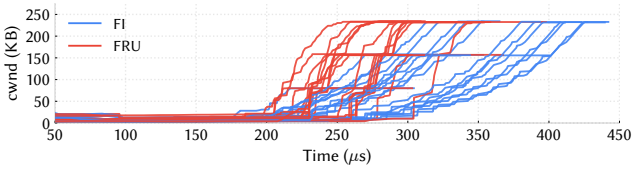

Figure 19: cwnd dynamics of the single long flow over multiple runs with either NSCC-FRU or baseline NSCC-FI, starting from steady-state sharing by all flows in incast.

B.2 Fast Ramp-Up for CUBIC CC

Algorithm 2 describes FRU for CUBIC, the default TCP CC for today's widely deployed operating systems [58]. It is similar to FRU for Swift (Algorithm 1). FRU is integrated into CUBIC by inserting a call to the *CUBICFRU()* function at the end of the *cubic_update()* function in Algorithm 1 of [23].

CUBIC CC grows the congestion window (*cwnd*) by a dynamically calculated amount per RTT during its congestion avoidance phase. This is implemented by incrementing *cwnd* by one packet for each *cnt* packets ACKed. FRU enables a *cwnd* growth rate faster than the baseline via an adaptive growth rate calculation on each ACK. Exactly as with Swift FRU, by pairing the bottleneck ABW/C reported in each ACK (*abwc*) with the flight size recorded at the corresponding data packet's transmission time (*tx_in_flight*), the algorithm estimates the flow's share of the bottleneck's capacity, *est_flow_capacity* (Lines 8–10, identical to the Swift FRU *est_flow_capacity* calculation). FRU identifies the gap between the current *cwnd* and this *fru_target* (Line 11) as *fru_delta* (Line 12). If

Algorithm 2 Fast Ramp-Up (FRU) for CUBIC CC

```
1:  Input: acknowledged (# packets ACKed in current ACK),
2:     tx_in_flight (flight size at time of packet transmission),
3:     abwc (bottleneck ABW/C),
4:     cwnd (congestion window size)
5:  Input/Output: cnt (packets ACKed per cwnd inc)
6:  function CUBICFRU(...)
7:     ▷ CSIG FRU Logic
8:     utilization = 1.0 − abwc            ▷ bottleneck utilization as a fraction
9:     f = 1.0 / utilization               ▷ safe multiplicative growth factor
10:    est_flow_capacity ← tx_in_flight × f
11:    fru_target ← max(new_fcwnd, est_flow_capacity)
12:    fru_delta ← (fru_target − cwnd)
13:    if fru_delta > 0 then
14:       fru_cnt ← cwnd / fru_delta
15:       fru_cnt ← max(fru_cnt, 1)
16:       cnt ← min(fru_cnt, cnt)
17:    end if
18: end function
```

fru_delta is positive (Line 13) it calculates the number of packets ACKed per FRU *cwnd* increment as *fru_cnt* (Line 14). It imposes a floor of 1 for *fru_cnt* to ensure that cwnd grows by at most $2x$ each round trip (Line 15). It then ensures the *cwnd* increase rate is at least as large as the FRU-enabled rate by ensuring *cnt* is no larger than the *fru_cnt* (Line 16). For example, when a flow with *cwnd* = 10 and *tx_in_flight* = 10 receives an ACK carrying an *abwc* value of 33%, it will estimate a capacity and target of $10*(1/(1-.33)) \approx 15$ packets; FRU calculates an *fru_delta* of $15 - 10 = 5$ and an *fru_cnt* of $10/5 = 2$. Over the next round trip the flow receives ACKs for *tx_in_flight* = 10 packets, and increments *cwnd* by 1 packet for each *cnt* = 2 packets ACKed, so that *cwnd* grows by 5 packets, driving *cwnd* to the target of 15 over a single RTT.

B.3 HPCC CSIG Integration

To evaluate CSIG as a drop-in INT replacement for HPCC, we integrate CSIG into HPCC in the htsim simulator [53] by adapting the HPCC sender to add compact (4-byte) CSIG tags to all outbound data packets. The maximum bottleneck normalized queue depth (`max(nQD)`) and the minimum bottleneck normalized available bandwidth (`min(ABW/C)`) signal types' raw signal values are represented using linear, equal-width bucketizations over the 32 signal values available in a compact tag. We further adapt the HPCC receiver to reflect the received CSIG signal type and value in the HPCC ACK using a 16-bit reflection header.

We compare the performance of two CSIG variants of HPCC with that of baseline, INT-based HPCC. In the first, on successive data packets, the sender alternates collecting `max(nQD)` and `min(ABW/C)` on data packets; unlike for INT, these may be measured at distinct bottlenecks. The second computes HPCC's U metric locally to report a single `max(Fused)` signal, ensuring all data is captured at the same switch. All simulations whose results appear in Fig. 20 and Fig. 21 are of the W4 Hadoop workload [47] running on a *320-server 3-tier Clos with 100 Gbps links*.

Fig. 20(a) confirms that both variants achieve 95p FCT slowdowns near-equivalent to baseline INT-based HPCC. CSIG thus effectively supports transports that consume in-band signals without the overhead of full INT. Moreover, as shown in Fig. 20(b), HPCC with CSIG feedback on every packet largely outperforms baseline HPCC using INT probes only once per RTT, illustrating the performance benefit of CSIG's support for per-packet feedback.

Fig. 21 shows the results of htsim simulations in which we vary the frequency at which packets collect INT-based congestion feedback from per-data-packet ("Per-Pkt") to one probe packet per multiple of the RTT ("Probe (N) RTT"). As discussed in §2, these results clearly show that HPCC responds more effectively to congestion with shorter intervals between switch-based feedback, with per-packet feedback achieving the least slowdown vs. ideal flow completion time.

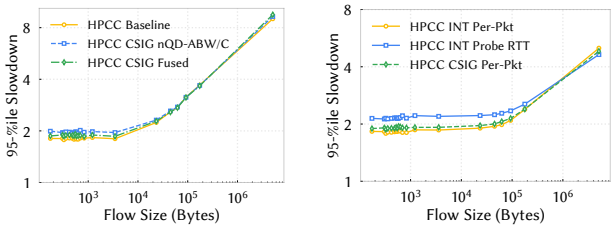

Figure 20: FCT Slowdown (vs. ideal) for HPCC-CSIG variants; 12 μs RTT.

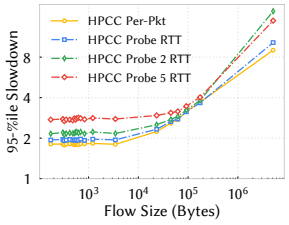

Figure 21: Effect of feedback frequency on HPCC flow slowdown (vs. ideal); 50 μs RTT.

B.4 FRU: Simulations and Dynamics

Fig. 15 presents results of simulations in which 4 foreground incast RPC flows compete with 37 background RPC flows that each burst in a synchronized 0.5 ms on/10 ms off pattern. Foreground RPC operations are 1 MB in size, and each foreground flow originates load at 90 Gbps, with Poisson operation arrivals. Background RPC operations are 8 KB in size, and during "on" periods, each background flow originates load at 11 Gbps. The topology includes two racks, in which all foreground and background sources are co-located on distinct hosts in one rack, and send to a single host in the other rack via aggregation blocks. All fabric links run at 400 Gbps. Swift uses a 25 μs base target delay. The simulations use compact CSIG tags with ABW/C encodings detailed in Tab. 4.

Fig. 22 shows the evolution of senders' aggregate fcwnds over time in the above scenario, at a moment when the background flows have paused and FRU has the opportunity to claim their relinquished bandwidth, plotting baseline Swift and Swift/FRU with 1, 10, and 100 μs ABW/C intervals. The shaded regions show the periods during which FRU remains active for each interval. During the periods when FRU is active, the steeper slope of aggregate fcwnd *vs.* that for baseline Swift illustrates how FRU claims spare capacity significantly faster than baseline Swift's additive increase. As the ABW/C interval lengthens, the signal value tracks changes in bottleneck available bandwidth more slowly, and FRU in turn responds more slowly upon onset and end of spare capacity. The shallower slopes of the increases in aggregate fcwnd as the ABW/C

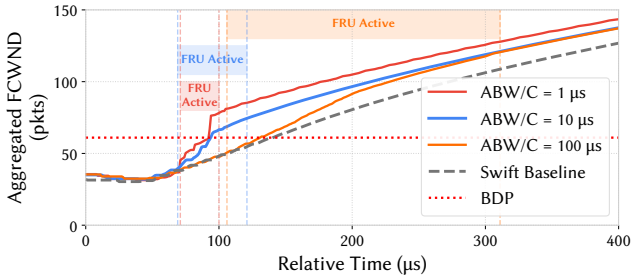

Figure 22: Aggregate cwnd growth for baseline Swift and FRU for 1, 10, and 100 μs ABW/C intervals.

interval lengthens reflect this dynamic, as does the increasing duration that FRU is active.

For all three intervals, FRU exceeds the bottleneck's BDP faster than the baseline, and thus fully reclaims the released spare capacity. A 1 μs interval reacts rapidly to transients, causing FRU to ramp up in large increments (sometimes significant enough to cause a multiplicative decrease) but deactivate quickly after the BDP is surpassed. Conversely, a 100 μs interval ramps up in more conservative increments but is slower to deactivate due to signal staleness. A 10 μs interval strikes a balance, reaching the BDP promptly without overly large increments while deactivating soon after.

C Efficient HST with interrupt-driven DMA

This section presents details for an efficient, low-overhead implementation for the user-space HST Agent. As described in §4.2 and illustrated in Fig. 5, the HST Agent can achieve μs-scale operations by directly accessing ASIC registers. To determine when a read or write operation completes, the agent performs active busy-polling on completion flags or registers via MMIO. While this active busy-polling approach is simpler, it requires the HST thread to continuously spin for completions, effectively pegging a logical core at near-100% utilization.

We discuss an optimization to alleviate this CPU overhead: an interrupt-driven DMA approach. Instead of issuing individual register accesses over the PCIe bus, the HST Agent configures a chain of descriptors in host memory that specifies a list of target register/table addresses and operations. The ASIC's internal DMA engine then autonomously traverses this descriptor chain, reads the telemetry counters or writes new datapath signals, and copies the results in bulk directly to a coherent DMA buffer in host RAM. During the DMA engine's execution, the HST Agent thread yields the CPU and sleeps in the kernel. Upon completing the entire descriptor chain, the ASIC asserts a PCIe Message Signaled Interrupt (MSI), which triggers the kernel's interrupt service routine (ISR) to wake up the HST Agent thread. The thread then processes the results and explicitly triggers the next DMA cycle.

In Tomahawk 6 testbed deployments, we find that this optimization reduces HST's CPU usage to under 20% of a logical core, and also reduces cycle times by up to 10× for counters stored contiguously in DMA-able memory blocks as compared to reading them individually. While we did not deploy this optimization in production for SW-assist platforms to simplify engineering, it remains a viable option to improve efficiency in CPU-constrained switch deployments or for extending HST to capture a larger set of counter statistics.

D Robust Deployment

Deploying a new Layer 2 protocol across a hyperscale fleet presents challenges fundamentally different from standard software rollouts. Unlike transport algorithms that can be updated solely at end-hosts or localized feature upgrades on switches, CSIG required carefully orchestrated updates across the entire networking ecosystem: NIC firmware, host networking stacks, switch stacks, and the SDN control plane. Over a multi-year period, we orchestrated this rollout across over a dozen distinct switch platforms and a large fraction of host platforms fleetwide. Our deployment highlights the friction between architectural evolution and the requirement for continuous, high-availability operation in a brownfield network.

D.1 Safe Expansion without blackholing

The primary risk in introducing a new packet header in a heterogeneous environment is traffic blackholing, wherein legacy switches or hosts that do not recognize the CSIG TPID drop these packets. We had two goals that occasionally conflicted with one another: first and foremost, avoid blackholing traffic, and second be able to iterate end-to-end on transport control loops and gain incremental telemetry coverage without waiting for CSIG support to be rolled out to every host and switch in the fleet. Two techniques allowed us to achieve these dual goals: "strip-and-forward" capability in switches, and being able to achieve islands of CSIG-safe domains with SDN.

To break the dependency of needing universal switch support for SW-assist CSIG before enabling the feature on hosts, we first implemented a *strip-and-forward* default behavior. Switches were first upgraded solely to recognize the CSIG TPID and strip the tag before forwarding. This was a simpler change that was faster to deploy than the full SW-assist CSIG pipeline. It ensured that hosts could send out CSIG-tagged packets without concerns about packets being dropped, avoiding the need for host↔switch negotiation. We also avoided complex per-hop negotiation protocols such as LLDP in favor of a centralized SDN-based port configuration via Orion [18] for safe rollouts: the key enabler to such safe expansion was a per-egress-port config knob which controlled whether to update or strip CSIG tags. This knob allowed us to define "CSIG-safe domains"—starting with intra-rack traffic and incrementally expanding to cluster-wide scopes. By configuring boundary ports to strip tags, we could fully enable CSIG end-to-end within a domain while insulating the rest of the network—particularly those with software version stragglers—from packets with the new format. This isolation was crucial to unblock production experiments and for iterating on control loops such as Fast Ramp-Up without waiting for the long tail of fleet upgrades. These production pilots, in fact, revealed that our early versions of FRU caused unacceptable fabric RTT regressions, allowing us to pivot to Algorithm 1 before expanding its rollout.

The transition from stripping tags to active end-to-end signaling required enabling CSIG's match/action pipeline and HST (§4.3). A major challenge during this transition was managing version stragglers in a heterogeneous environment where software-upgrade lifecycles vary. We identified two classes of switch version stragglers: those that stripped tags and those running versions old

enough that drop CSIG packets. While we could tolerate the former—accepting a temporary gap in CSIG coverage—the latter required forced upgrades before the associated domain was safe to enable CSIG packets in. Furthermore, since we continue to introduce new switch platforms into our deployments, these platforms need to support CSIG recognition from day one of their introduction to avoid traffic disruption.

Using this phased approach, SW-assist CSIG was deployed via multiple software releases. At the time of writing, 100% of our data center switches support "strip-and-forward," and over 80% have been upgraded to enable the full CSIG match-action logic.

D.2 Non-disruptive Upgrades

Switches: Non-stop Forwarding. We had to ensure zero packet loss while activating SW-assist CSIG support on live switches. This was critical for ToR switches where draining traffic for software upgrades is not viable. This requirement applied to both CSIG packets and non-CSIG packets, and therefore needed a nuanced approach for how the data plane objects were upgraded: First, we performed runtime updates to the TCAM policies to add the necessary qualifiers and actions required for CSIG compare-and-replace operations. We designed these updates to re-carve TCAM slices *in-place* without deleting and re-initializing the TCAM banks, preventing the temporary forwarding disruption that would have otherwise occurred during the updates. Second, we implemented in-place upgrades of next-hop table entries to handle CSIG tags. This ensured that the existing forwarding rule-set was unimpacted during the update (preventing packet loss), and eliminated the need to duplicate entries for CSIG and non-CSIG packet streams. Third, for subsequent upgrades, we ensured that the CSIG pipeline elements persist in HW even as the switch software is upgraded. Although telemetry is briefly stale for packets while the HST agent restarts (typically a few seconds), real-time accuracy resumes immediately following the upgrade.

Host: Hitless Upgrades. On the host side, preventing packet drops or connection resets during the rollout was also paramount. We could not reset existing connections —particularly long-lived ones— simply to start marking CSIG tags. Explicit negotiation for remote-host CSIG support largely addressed this issue by ensuring that only new connections were eligible for CSIG-tagging post transparent upgrades to CSIG-supported versions. However, this was complicated by the fact that switches might strip CSIG tags despite successful end-host negotiation. Therefore, to handle this and ensure backward compatibility, we designed the Rx and Tx packet processing logic to dynamically support headers with or without L2 CSIG tags or L4 CSIG reflection headers within the same flow. This approach ensured no disruption to connections as we expanded the host rollout.

D.3 Defense-in-depth for high availability

Despite rigorous pre-production testing, the scale of trillions of concurrent flows exposes edge cases that are impossible to predict. Mindful of the deployment challenges faced by ECN, where rigid middlebox policies famously caused widespread packet blackholing [35], we implemented multiple layers of protection to prevent CSIG from breaking connections and causing outages.

First, the transports leverage *Protective Re-route* [56] if they suspect packet drops. Before any CSIG fallback logic intervenes, PRR attempts to self-heal by re-pathing. In one instance, CSIG tagging was unintentionally enabled for traffic transiting an aggregation-block that lacked support for CSIG tag recognition. Upon detecting packet loss on these specific paths, the mechanism successfully diverted traffic to the dozens of parallel, CSIG-compatible transit blocks. Remarkably, this automated mitigation was so effective that there was no noticeable degradation in application performance during this period.

Second, the transport layer provides *Auto-disable fallback*, serving as the next line of defense and a critical safety requirement. If the transport stack detects pathological patterns—such as consistent packet loss suggesting a blackhole that PRR could not circumvent—it automatically ceases CSIG tagging for the connection and also reverts the connection to standard congestion control (e.g., Swift baseline). This ensures that even if a configuration error slips past routing defenses, individual connections remain robust. We prioritize connection stability over CSIG telemetry, accepting that unrelated transient retransmits may occasionally trigger false positives.

Third, *Manual Host-Side Rollback* serves as our last-resort "big red button" to turn off CSIG tagging. We opted to prefer host-side rollbacks over switch-side rollbacks because host flag flips to disable CSIG are hitless to connections, whereas switch rollbacks are typically disruptive and may require traffic draining.

Finally, *State Verification* serves as a critical diagnostic and auditing tool. Complementing these active mitigations, we run periodic verification checks that sweep switch hardware state to ensure CSIG TPID and Match/Action tables are correctly configured on all ports. If any misconfiguration is detected, high-priority alerts are fired immediately to aid debugging. This mechanism was instrumental during one incident where an SDN-triggered port speed change inadvertently erased port TPID configurations on the switches, resulting in CSIG blackholing. While active mitigations protected traffic, it was this state verification that flagged the underlying discrepancy, allowing us to identify and fix the root cause before it could aggravate into a wider failure.