Google

LINUX PLUMBERS CONFERENCE 2025

Congestion Signaling (CSIG) for Linux TCP Data Center Networking

Sagarika Sharma

Dec 2025

Agenda

01 **CSIG Introduction**

02 CSIG Protocol Details

03 Life of a CSIG Packet in Linux TCP Stack

04 Deployment: Monitoring, Safe Rollout, and Testing

05 CSIG Use Cases

Congestion Control

- TCP provides a reliable, ordered byte stream interface
- Implemented on an unordered, lossy datagram network with varying speed and reliability
- TCP abstraction implemented with
    - Transport Protocol: Loss Recovery, Flow Control, Congestion Control, Multipathing
    - Scheduling: Bandwidth Allocation, Traffic Shaping/Packing, Quality of Service (QoS)

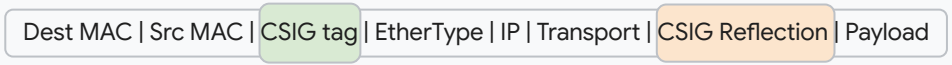
Congestion Signals

- Congestion control algorithms use congestion signals to estimate ideal data transmission rate
- Signals offer differing levels of information/precision
- Examples of Signals: Packet Loss, Round-Trip Time, Explicit Congestion Notification (ECN), In-band Network Telemetry (INT)
- Data center networks have growing requirements for network efficiency. Fine-grained congestion signals from the fabric allow for more accurate & faster reactions to bottlenecks.

Congestion Signaling (CSIG)

- In-band network telemetry protocol: Network devices can embed real-time congestion information directly into the headers of live data packets
- What information does CSIG provide end-hosts?
  - (1) Bottleneck signal: The type and value of the most significant congestion point along the packet's path (e.g. Minimum Available Bandwidth)
  - (2) Bottleneck location: Where bottleneck occurred within network fabric (e.g. ToR Uplink)
- CSIG Benefits: Space-efficient, Compute-efficient, Hardware-friendly, Transport and Encapsulation agnostic, Real-time, Extensible
- Successfully deployed and used at scale within Google's data centers

CSIG-tagged TCP packet Format



CSIG Standardization

- CSIG Standardization at Ultra Ethernet Consortium: Unanimous support for CSIG from multiple switch and NIC hardware vendors in UEC
- CSIG support is available in over a dozen switch and NIC ASICs in the market today
- CSIG Specification v0.5 has been standardized and published
- Congestion Signaling IETF draft
- CSIG at 2025 OCP Global Summit

Agenda

01 CSIG Introduction

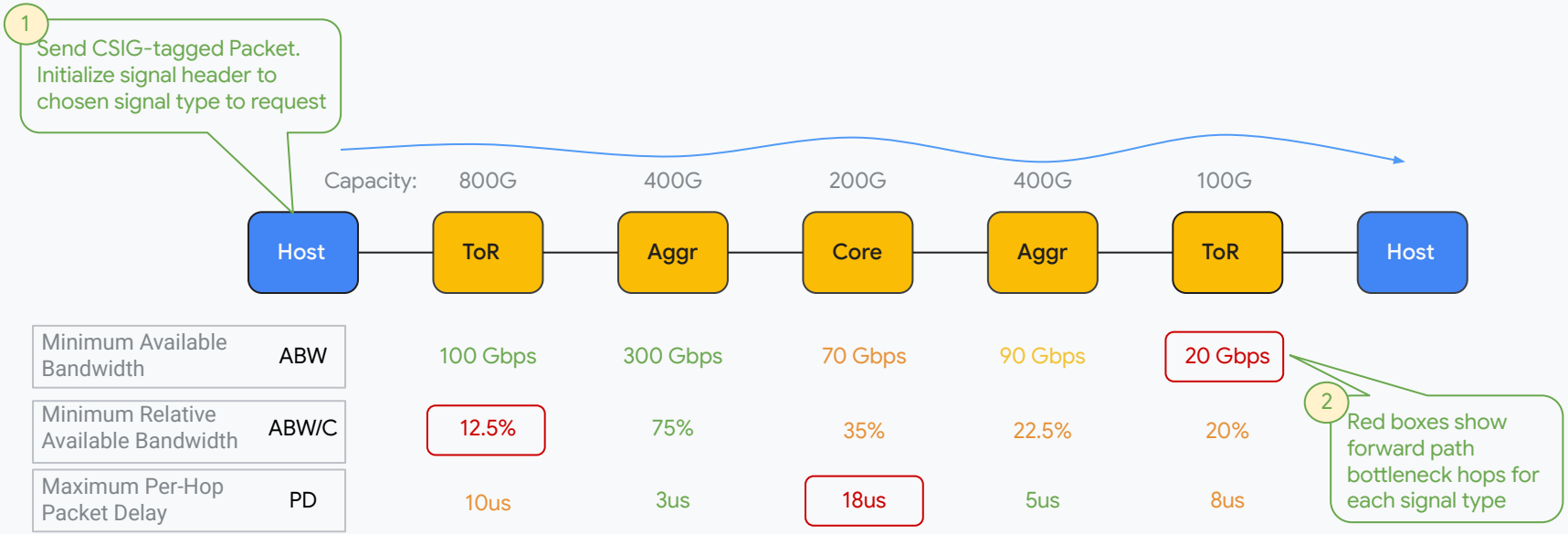
02 **CSIG Protocol Details**

03 Life of a CSIG Packet in Linux TCP Stack

04 Deployment: Monitoring, Safe Rollout, and Testing

05 CSIG Use Cases

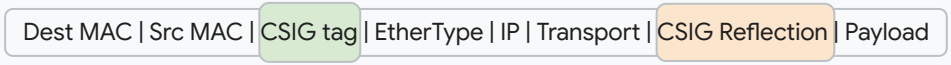
CSIG Protocol: Reporting the Path Bottleneck



CSIG-tagged TCP packet Format: Dest MAC | Src MAC | CSIG tag | EtherType | IP | Transport | CSIG Reflection | Payload

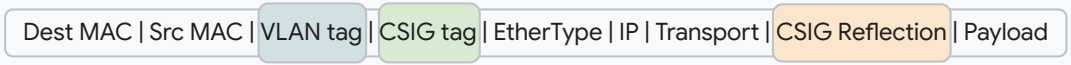
CSIG Protocol: CSIG L2 tag Format

CSIG-tagged TCP packet Format



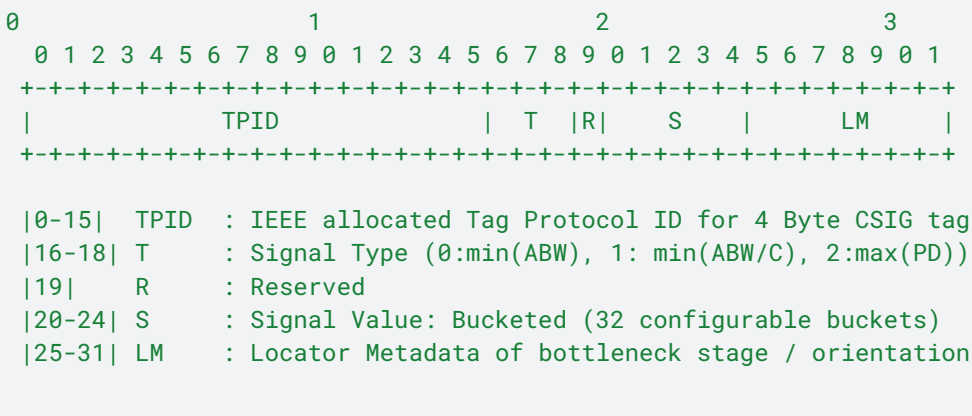
- CSIG tag is a fixed size tag at the layer 2 header (fixed offset) and structurally similar to VLAN tags
  - Updated by end hosts, network switches, and other transit devices
  - Compatible with PSP and IPSec encrypted packets by design
- Support VLAN tag + CSIG tag on a packet with CSIG tag as last tag in L2 header

802.1q CSIG-tagged TCP packet Format



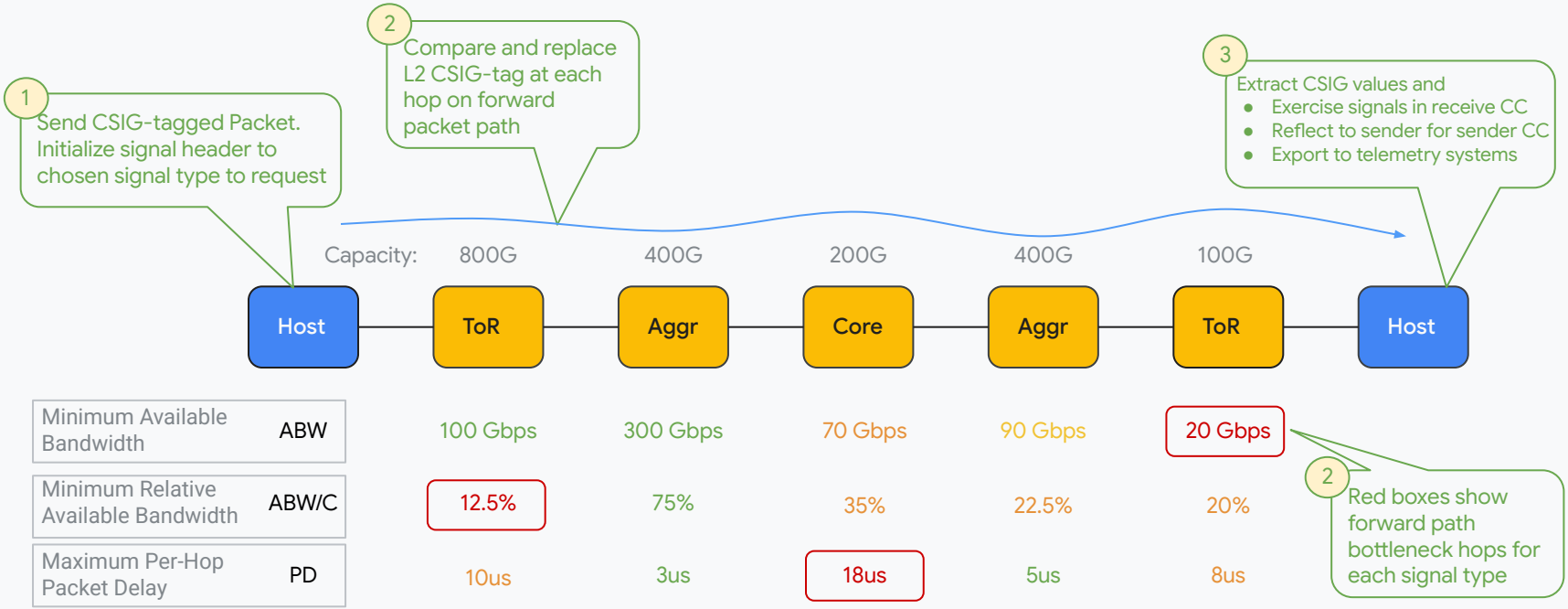
CSIG Protocol: CSIG L2 tag Format

L2 CSIG compact tag format



Note: There is also an expanded 8B CSIG tag. Presentation focuses on compact tag.

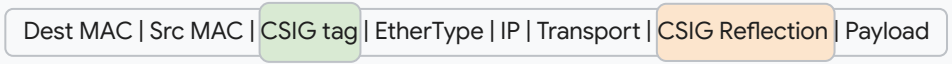
CSIG Protocol: Reporting the Path Bottleneck



CSIG-tagged TCP packet Format: Dest MAC | Src MAC | CSIG tag | EtherType | IP | Transport | CSIG Reflection | Payload

CSIG Protocol: CSIG L4 header Format

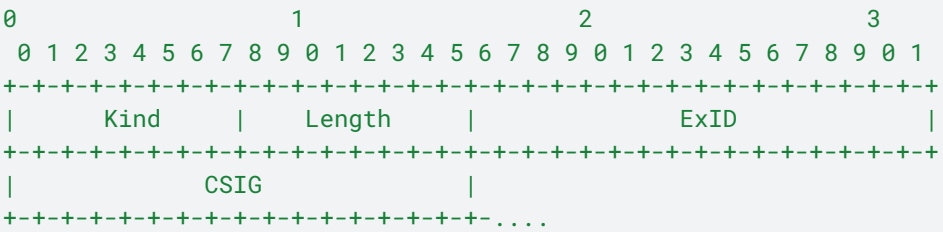
Example: CSIG-tagged TCP packet



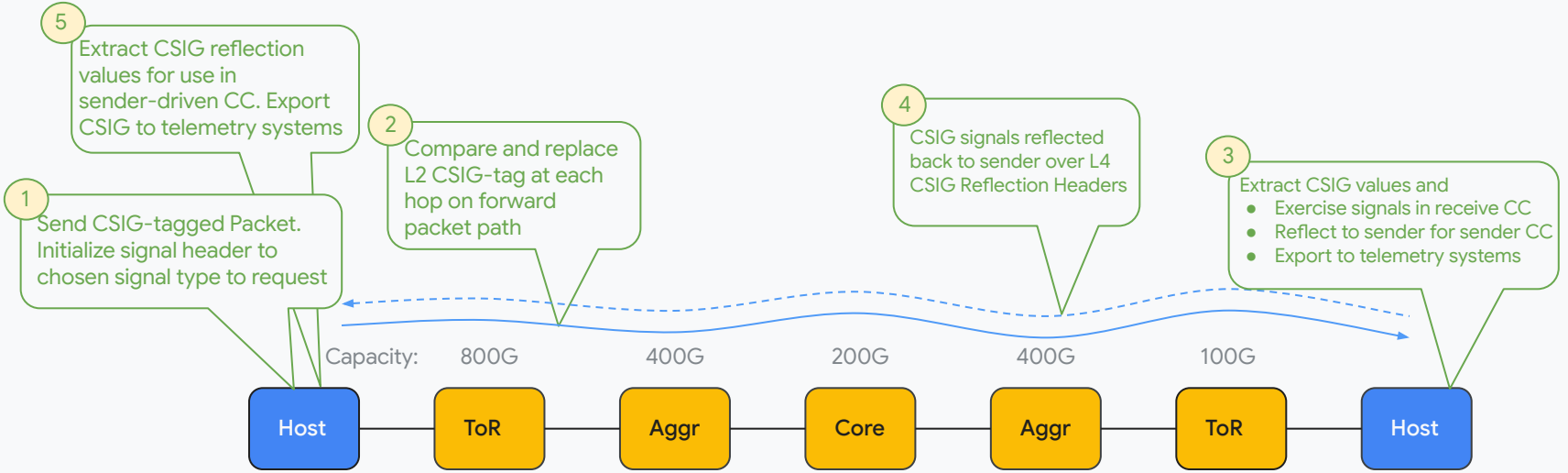
- The CSIG Reflection TCP Option is used by the receiving end-host to send the congestion information gathered from the L2 CSIG tags back to the original sending end-host
- The reflected information represents the **latest** CSIG data extracted from incoming data packets at the receiver. Not necessarily 1-to-1 mapping
- Feedback enables the original sender to make decisions for congestion control, traffic management, etc.

CSIG Protocol: CSIG L4 header Format

Example: CSIG Reflection Header as TCP Experimental Option



CSIG Protocol: Reporting the Path Bottleneck



Minimum Available Bandwidth	ABW	100 Gbps	300 Gbps	70 Gbps	90 Gbps	20 Gbps
Minimum Relative Available Bandwidth	ABW/C	12.5%	75%	35%	22.5%	20%
Maximum Per-Hop Packet Delay	PD	10us	3us	18us	5us	8us

2 Red boxes show forward path bottleneck hops for each signal type

CSIG-tagged TCP packet Format: Dest MAC | Src MAC | CSIG tag | EtherType | IP | Transport | CSIG Reflection | Payload

CSIG Protocol: Feature Negotiation during TCP Connection Establishment

- End-host CSIG capabilities may differ. Host-wide CSIG support is controlled via a sysctl parameter
- A new “CSIG-permitted” TCP Option in SYN/SYNACK packets negotiates CSIG capabilities during the handshake
- CSIG tagging and reflection are enabled per TCP connection only upon mutual agreement by both end-hosts

Agenda

01 CSIG Introduction

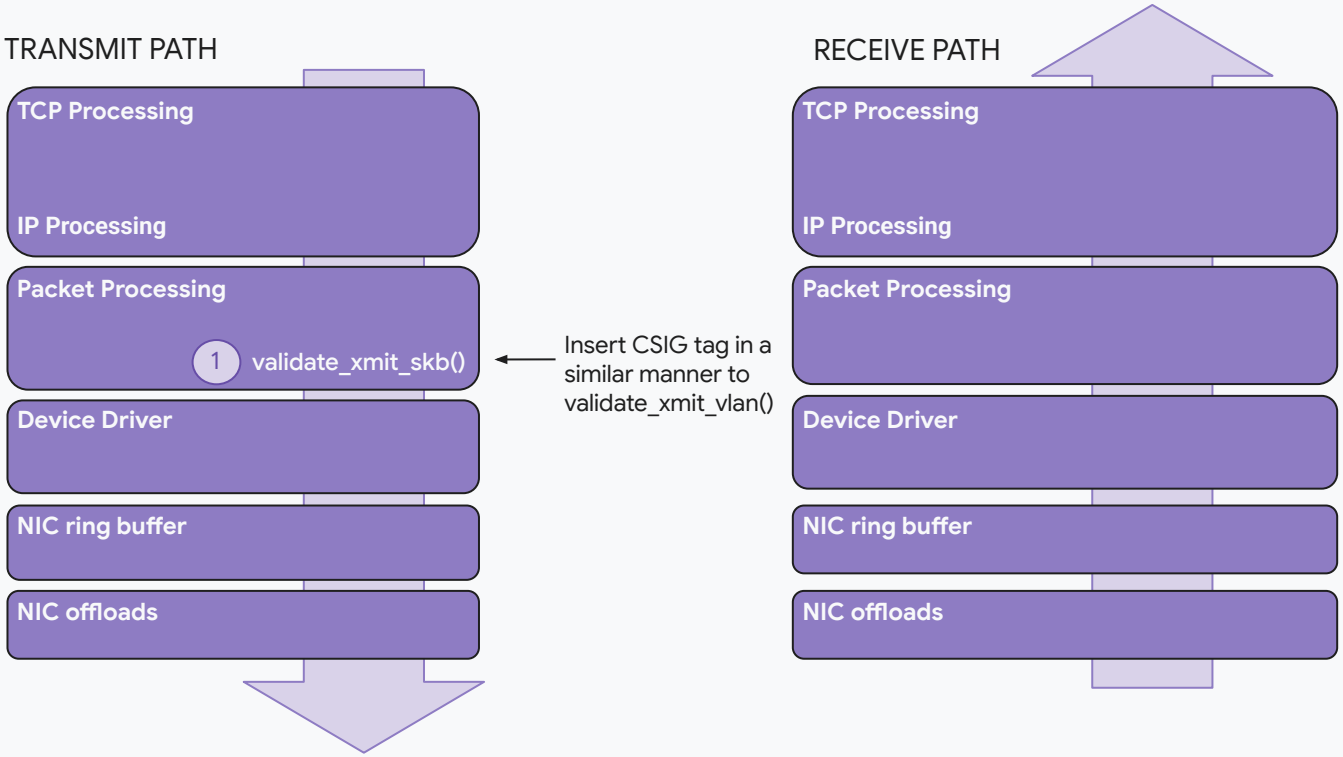
02 CSIG Protocol Details

03 Life of a CSIG Packet in Linux TCP Stack

04 Deployment: Monitoring, Safe Rollout, and Testing

05 CSIG Use Cases

Life of a CSIG-tagged Packet in Linux TCP: TX and RX



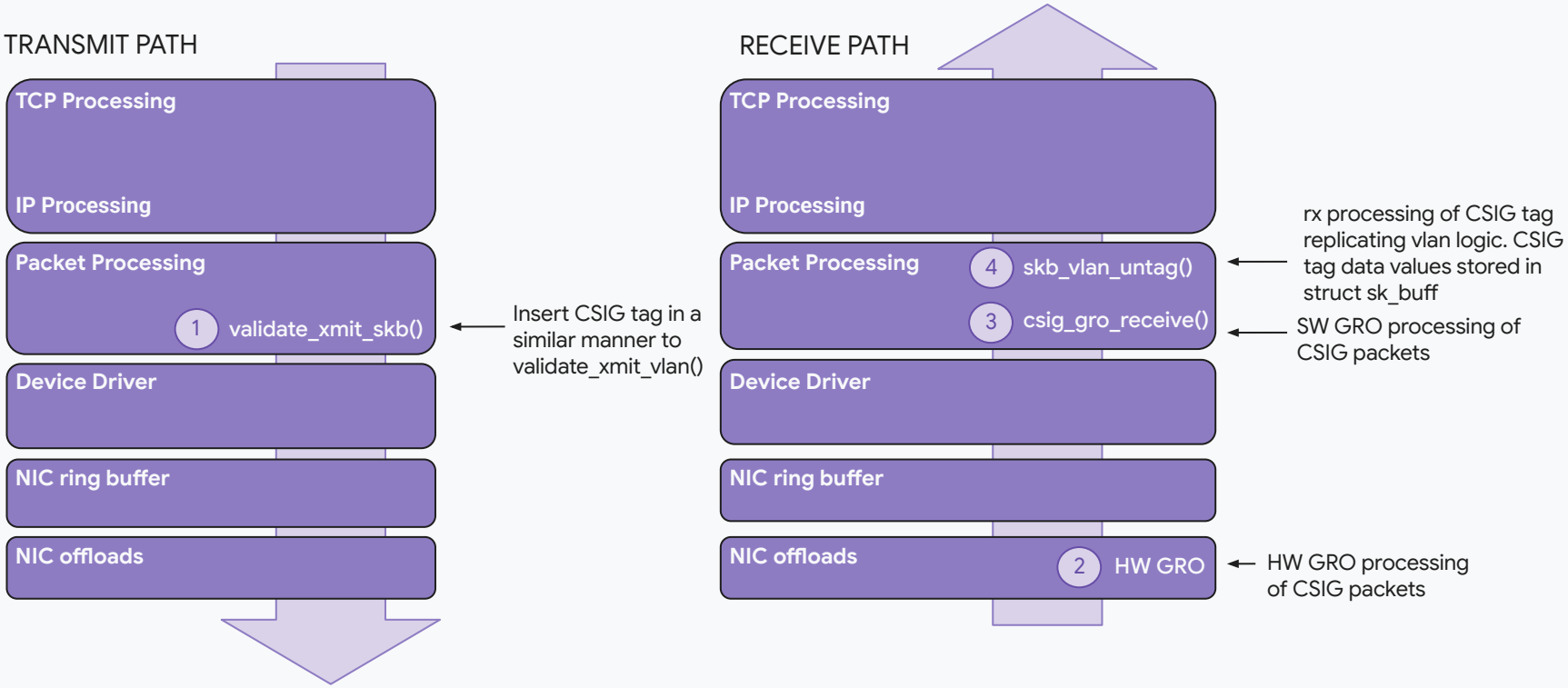
Life of a CSIG Packet: Insertion of CSIG tag

```
static struct sk_buff *validate_xmit_skb(struct sk_buff *skb, struct
net_device *dev, bool *again)
{
       netdev_features_t features;

       skb = validate_xmit_unreadable_skb(skb, dev);
       if (unlikely(!skb))
              goto out_null;

       features = netif_skb_features(skb);
       skb = validate_xmit_vlan(skb, features);
       if (unlikely(!skb))
              goto out_null;
...
}

static struct sk_buff *validate_xmit_vlan(struct sk_buff *skb,
                                  netdev_features_t features)
{
       if (skb_vlan_tag_present(skb) &&
           !vlan_hw_offload_capable(features, skb->vlan_proto))
              skb = __vlan_hwaccel_push_inside(skb);
       return skb;
}
```

Rather than inserting CSIG tag in device driver, imitate VLAN tag insertion in validate_xmit_skb()

Delay insertion of CSIG tag until last device in the transmit path in case of a stack of virtual devices

Life of a CSIG-tagged Packet in Linux TCP: TX and RX



Life of a CSIG Packet: Generic Receive Offload (GRO)

- CSIG support must be added to GRO, a critical network performance optimization
- L2 CSIG tag values can vary even among packets originating from the same TSO-generated segment (although unlikely)
- GRO should coalesce packets of the same flow even if their L2 CSIG tags differ
    - Preserve the L2 CSIG tag value from one of the coalesced packets, preferably that of the last packet
- The CSIG reflection TCP Option shouldn’t pose a challenge for coalescing for SW GRO

Life of a CSIG Packet: Generic Receive Offload (GRO)

```
int skb_gro_receive(struct sk_buff *p, struct sk_buff *skb)
{
        struct skb_shared_info *pinfo, *skbinfo = skb_shinfo(skb);
        unsigned int offset = skb_gro_offset(skb);
        unsigned int headlen = skb_headlen(skb);
        unsigned int len = skb_gro_len(skb);
        unsigned int delta_truesize;
        unsigned int new_truesize;
        struct sk_buff *lp;
        int segs;
...
done:
        NAPI_GRO_CB(p)->count += segs;
        p->data_len += len;
        p->truesize += delta_truesize;
        p->len += len;
        if (lp != p) {
                lp->data_len += len;
                lp->truesize += delta_truesize;
                lp->len += len;
        }
        skb_copy_latest_csig(p, skb);
        NAPI_GRO_CB(skb)->same_flow = 1;
        return 0;
}
```

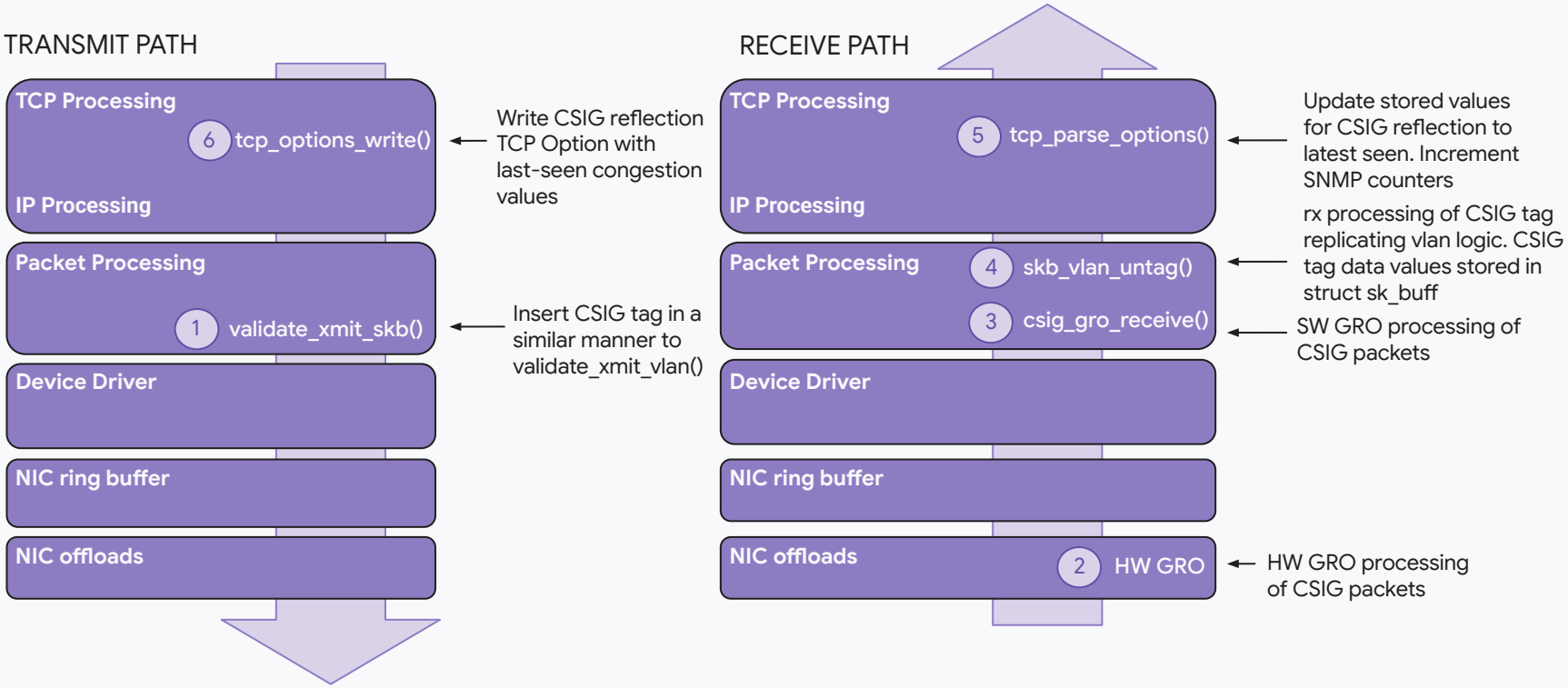
Given head skb and packet to be aggregated, “overwrite” CSIG tag of aggregated packet if CSIG tag present

Life of a CSIG Packet: RX Processing of CSIG tag

```
struct sk_buff *skb_vlan_untag(struct sk_buff *skb)
{
...
       if (unlikely(!pskb_may_pull(skb, VLAN_HLEN + sizeof(unsigned short))))
              goto err_free;

       vhdr = (struct vlan_hdr *)skb->data;
       vlan_tci = ntohs(vhdr->h_vlan_TCI);
       __vlan_hwaccel_put_tag(skb, skb->protocol, vlan_tci);

       skb_pull_rcsum(skb, VLAN_HLEN);
       ≈(skb, vhdr);

       skb = skb_reorder_vlan_header(skb);
       if (unlikely(!skb))
              goto err_free;

       skb_reset_network_header(skb);
       if (!skb_transport_header_was_set(skb))
              skb_reset_transport_header(skb);
       skb_reset_mac_len(skb);

       return skb;
...
}
```

Generalize VLAN tag rx processing in __netif_receive_skb_core

Life of a CSIG-tagged Packet in Linux TCP: TX and RX



Life of a CSIG Packet: Reflection Header

- RX Path: Storing Last-Seen L2 CSIG Value (Receiver)
  - The most recent valid CSIG value from the incoming data stream is cached within a field in `struct tcp_sock`
- TX Path: Reflecting Stored Value in TCP Option (Receiver -> Sender)
  - On next TCP segment from receiver to sender, encode the latest csig values from `struct tcp_sock` into CSIG Reflection TCP Option in `tcp_options_write()`
- RX Path: Processing Reflected CSIG Option (Original Sender)
  - Sender processes CSIG reflection TCP Option in `tcp_parse_options()`

Agenda

01 CSIG Introduction

02 CSIG Protocol Details

03 Life of a CSIG Packet in Linux TCP Stack

04 **Deployment: Monitoring, Safe Rollout, and Testing**

05 CSIG Use Cases

Telemetry

- CSIG Packet Counters (TX/RX)
  - Exposed via standard kernel SNMP counters e.g. TCPRxCsig, TCPTxCSIG
- Per-TCP Connection Congestion Information (from reflection headers)
  - The most valuable telemetry comes from the CSIG data reflected back in TCP Options. Extend `struct tcp_info` to export the latest reflected CSIG data
- Integrate into packet sampling

Auto-Fallback for Reliability

```
/* A write timeout has occurred. Process the after effects. */
static int tcp_write_timeout(struct sock *sk)
{
...
        if ((1 << sk->sk_state) & (TCPF_SYN_SENT | TCPF_SYN_RECV)) {
...
        } else {
                if (retransmits_timed_out(sk, READ_ONCE(net->ipv4.sysctl_tcp_retries1), 0)) {
                        /* Black hole detection */
                        tcp_mtu_probing(icsk, sk);

                        __dst_negative_advice(sk);
                }
                tcp_csig_fallback(sk);

                retry_until = READ_ONCE(net->ipv4.sysctl_tcp_retries2);
                if (sock_flag(sk, SOCK_DEAD)) {
                        const bool alive = icsk->icsk_rto < tcp_rto_max(sk);

                        retry_until = tcp_orphan_retries(sk, alive);
                        do_reset = alive ||
                                !retransmits_timed_out(sk, retry_until, 0);

                        if (tcp_out_of_resources(sk, do_reset))
                                return 1;
                }
        }
}
```

Defense against events causing CSIG-tagged packets to be blackholed: “fallback” behavior necessary

Disable CSIG tagging & reflection across a TCP connection on Nth consecutive timeout retransmit

Packetdrill: Effective Testing of CSIG

- **Packetdrill:** open-source scripting tool for unit testing kernel networking stack
- Many upstream TCP tests are packetdrill tests
- Tests are written in time-stamped script format. Scripts dictate system calls to make (strace-like), network packets to inject, and outgoing packets to expect (tcpdump-like)
- USENIX ATC ‘13 packetdrill presentation

```
 0 socket(..., SOCK_STREAM, IPPROTO_TCP) = 4
+0...0 connect(4, ..., ...) = 0

+0 > S 0:0(0) <mss 1460,sackOK,TS val 0 ecr 0,nop,wscale 8>
+0 < S. 0:0(0) ack 1 win 32792 <mss 1000,sackOK,nop,nop,nop,wscale 7>
+0 > . 1:1(0) ack 1
```

Example: packetdrill client packet

Packetdrill: Effective Testing of CSIG

- Why Packetdrill for CSIG specifically?
    - Check correctness of CSIG edge cases e.g. mimic CSIG-tagged packet drops to validate Auto-Fallback, GRO handling of CSIG L2 tags
- Add support for L2 tags with VLAN/CSIG and CSIG reflection TCP option to packetdrill to test CSIG outlier issues

Agenda

01 CSIG Introduction

02 CSIG Protocol Details

03 Life of a CSIG Packet in Linux TCP Stack

04 Deployment: Monitoring, Testing, and Safe Rollout

05 **CSIG Use Cases**

Congestion Signaling (CSIG) Use Cases

Congestion Control
Reduce transfer latency

Ramp-up quickly and safely to available bandwidth in the absence of congestion

Respond precisely to the bottleneck hop in the presence of congestion

Traffic management
Maximize goodput

Multipathing and Load Balancing

- Select paths with the most available bandwidth
- Use locator to distinguish between incast and core congestion

Traffic Engineering

- Aggregate CSIG values spatially and temporally across flows, augment Traffic Matrix
- Provision routes for better burst absorption, route around bottlenecks

Network debuggability and operations
Better network provisioning

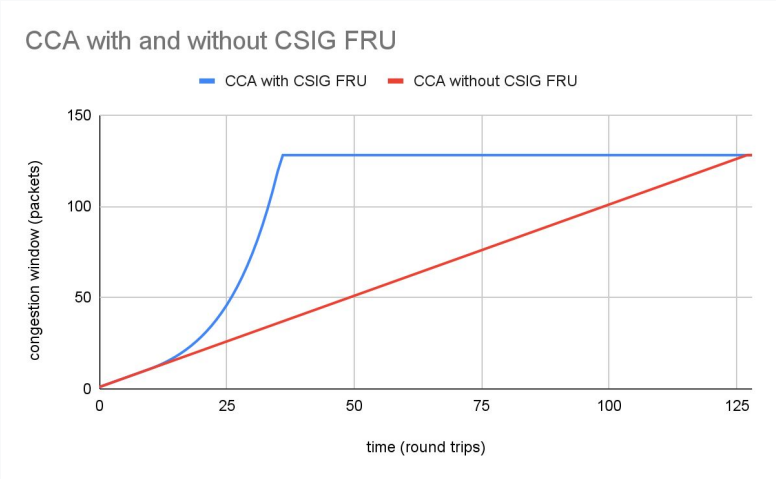
Debug and resolve bottlenecks in the network, as experienced by flows

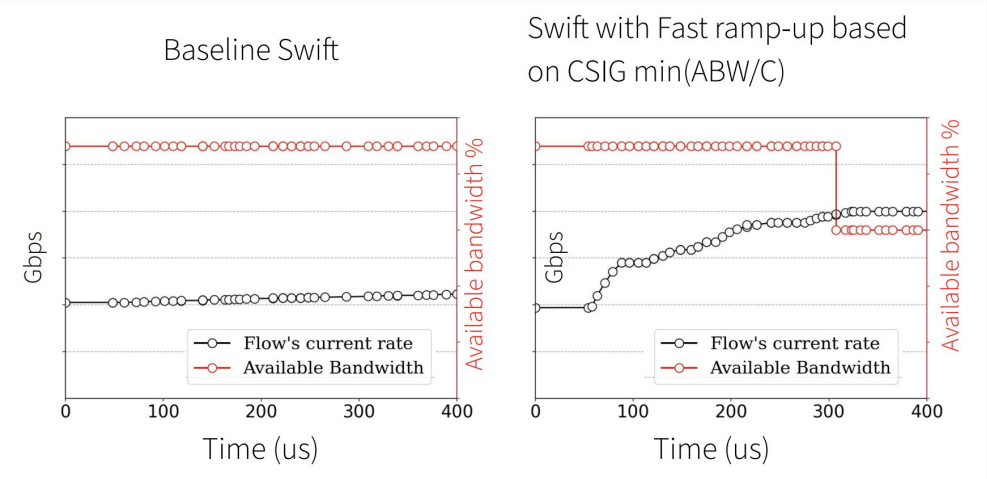
Mesh probers reveal bottleneck trends between end-point pairs

Timely provisioning and repair processes based on aggregated CSIG data

CSIG: Using Congestion signals for CCA

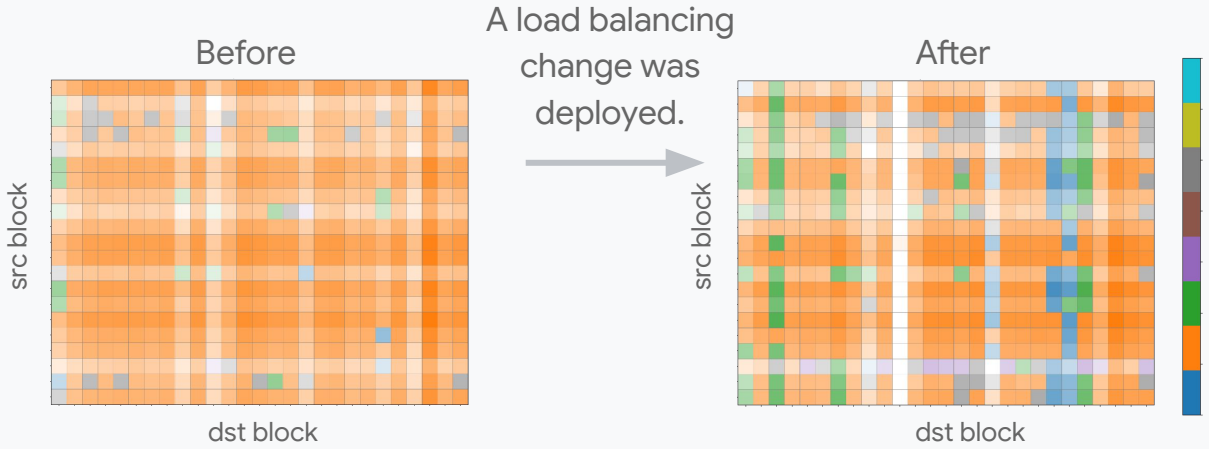
- Existing congestion control algorithms can use CSIG values to address blind spots in end-to-end signals such as packet loss, delay, and delivery rates
- CSIG FRU (Fast Ramp Up): have the CC algorithm increase cwnd more rapidly when CSIG ABW/C utilization signal indicates high available bandwidth
- Simplified Diagrams:





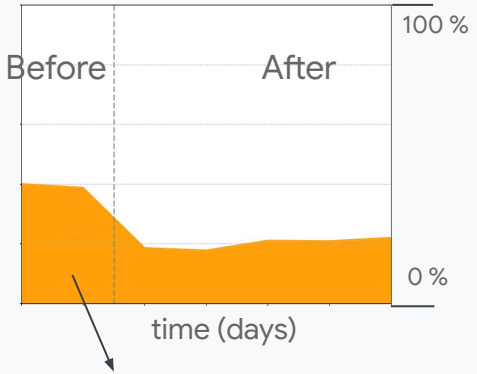
CSIG: Using Congestion signals to Validate Network Improvements

Real-world case study from Google showing how CSIG validated a load-balancing change in a production cluster



CSIG data revealed ToR uplinks (orange) were the most frequent bottlenecks for communication-pairs in the cluster

CSIG data *proves* that ToR uplink bottleneck is alleviated and that the bottleneck has moved to other tiers of the network fabric.



Fraction of traffic for which ToR uplink is the bottleneck

Next Steps

Thank you!

Contributor List: Abhiram Ravi, Brian Vazquez, Eric Dumazet, Grace Hwang, Kevin Yang, Nandita Dukkipati, Neal Cardwell, Shijith Chempeth, Volkan Mutlu, Willem de Bruijn, Yong Xia, Yuchung Cheng

Questions