INT:
In-band Network Telemetry

Jeongkeun “JK” Lee
Sr. Principal Engineer, Intel, Connectivity Group
jk.lee@intel.com
Apr 2022

**Courtesy Changhoon Kim at Stanford for a few slides*

“If you can’t measure it, you can’t improve it”

-- Peter Drucker

Agenda

- Network telemetry for debugging
  - Network = distributed system with high-speed HW/SW components
  - Network dataplane debugging ~= HW or OS kernel debugging
- INT intro and brief history
- INT variations and systems aspect
- INT use case
- Industry direction: 1) scalable telemetry 2) for control

Logging and debugging

- Kubernetes error log
  - Jul 23 12:33:43 kubelet: E0723 16:33:42.826691 2740 pod_workers.go:186] Error syncing pod 2771d596-8e96-11e8-a08e-080027ced1d7 ('nginx-678c57c5cc-jcs5d_default(2771d596-8e96-11e8-a08e-080027ced1d7)'), skipping: failed to 'StartContainer' for 'nginx' **with** ImagePullBackOff: 'Back-off pulling image 'nginx:0.10'' ... Jul 23 12:34:24 minikube kubelet: E0723 16:34:24.472171 2740 remote_image.go:108] PullImage 'nginx:0.10' **from** image service **failed**: rpc **error**: code = Unknown **desc** = **Error** response **from** daemon: manifest **for** nginx:0.10 **not found**

- Stack trace

Timestamp	Event	Details
08:27:15:979	METHOD_EXIT	[81]|01p300000000k4d|DoctorInfoService.getInstance(Id)
08:27:15:979	SOQL_EXECUTE...	[38]|Aggregations:0|SELECT Id FROM Account WHERE (Affiliate__c
08:27:15:992	SOQL_EXECUTE...	[38]|Rows:0
08:27:15:992	METHOD_ENTRY	[3]|01p4C000000A6Hn|CurrencyExchange.CurrencyExchange()
08:27:15:992	SOQL_EXECUTE...	[15]|Aggregations:0|SELECT Name, Symbol__c, Pro_Upgrade_Pref
08:27:16:004	SOQL_EXECUTE...	[15]|Rows:3
08:27:16:005	METHOD_EXIT	[3]|CurrencyExchange
08:27:16:005	METHOD_ENTRY	[57]|01p4C000000A6Hn|CurrencyExchange.parseCurrencyType(Id)
08:27:16:005	METHOD_EXIT	[57]|01p4C000000A6Hn|CurrencyExchange.parseCurrencyType(Id)

Network debugging ~= kernel debugging?

- Kernel does not support debug mode, no stack trace
- Hence printk() into syslog

```
ynamic_debug/control
ction flags format
nhancednfs-1.4/default/net/sunrpc/svc_rdma.c:323 [svcxprt_rdma]svc_rdma_cleanup =_ "SVCRDMA Module Removed, deregister RPC RDMA transport\012"
nhancednfs-1.4/default/net/sunrpc/svc_rdma.c:341 [svcxprt_rdma]svc_rdma_init =_ "\011max_inline       : %d\012"
nhancednfs-1.4/default/net/sunrpc/svc_rdma.c:340 [svcxprt_rdma]svc_rdma_init =_ "\011sq_depth         : %d\012"
nhancednfs-1.4/default/net/sunrpc/svc_rdma.c:338 [svcxprt_rdma]svc_rdma_init =_ "\011max_requests     : %d\012"
```

What constitute useful log and debug trace?

- Timestamp
- Description of the event
- Event driven (to scale)
- Runtime control of logging level

(Have you heard of) Network logging?

- Maybe this
- Or this
- Or this

```
[15/Apr/2011:09:40:17 -0700] "GET /global.asa HTTP/1.0" 404 315 "-" "-"
[15/Apr/2011:09:40:17 -0700] "GET /~root HTTP/1.0" 404 310 "-" "-"
[15/Apr/2011:09:40:18 -0700] "GET /~apache HTTP/1.0" 404 312 "-" "-"
```

Web server log

```
*Dec 19 03:17:00.035: %LINEPROTO-5-UPDOWN: Line protocol on Interface IPv6-mpls, changed state to up
*Dec 19 03:17:00.167: %PARSER-4-BADCFG: Unexpected end of configuration file.

*Dec 19 03:17:00.167: %SYS-5-CONFIG_I: Configured from memory by console
*Dec 19 03:17:00.247: %LINK-5-CHANGED: Interface FastEthernet0/0, changed state to administratively down
```

Router link status

```
*Mar  1 00:06:07.863: BGP: 192.168.1.1 sending OPEN, version 4, my as: 200, holdtime 180 seconds
*Mar  1 00:06:07.871: BGP: 192.168.1.1 send message type 1, length (incl. header) 45
*Mar  1 00:06:07.947: BGP: 192.168.1.1 rcv message type 1, length (excl. header) 26
```

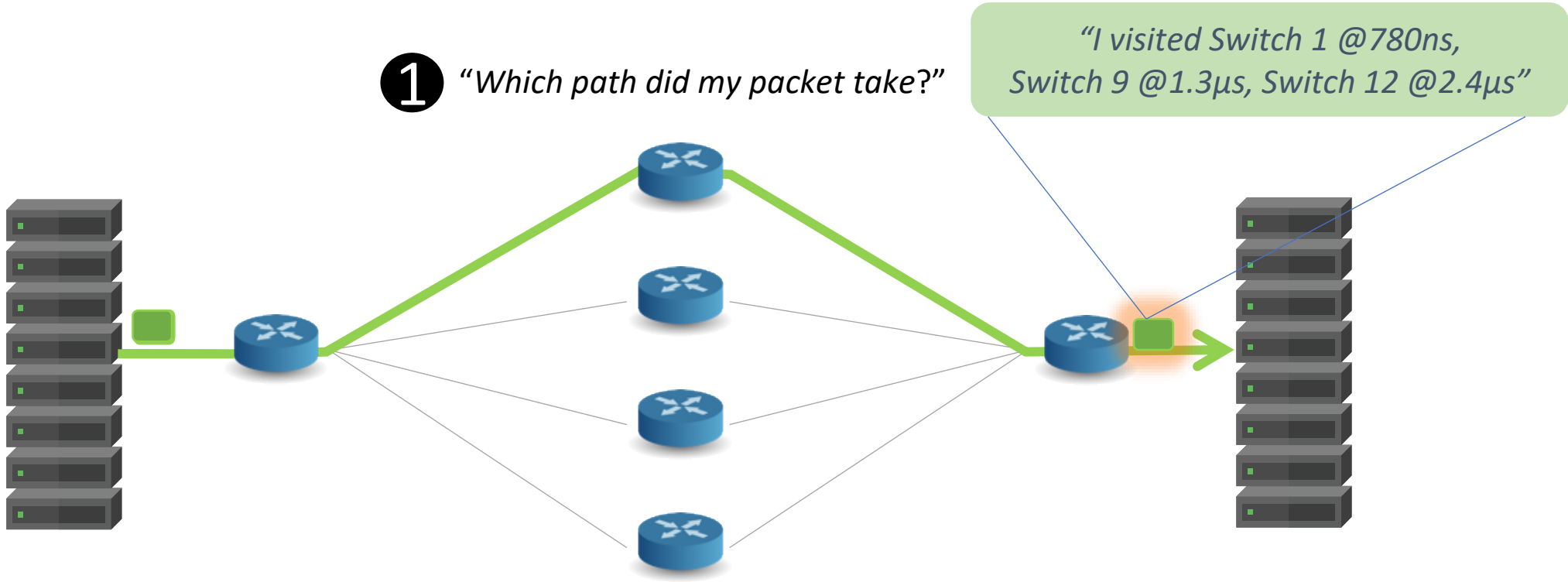
Debugging network dataplane?

- Ping/trace
  - No guarantee to reflect path/congestion/drop events of real data pkts experience
- Network telemetry
  - Per-port/queue stats, polling from control plane
  - Per-flow statistics (NetFlow)
  - Per-packet monitoring (pcap trace)

- Packet monitoring is the finest-grained, lowest-level telemetry
  - Crucial for network troubleshooting and debugging
  - Similar to printk() in Linux ➔ event-based logging/filtering is a must
- Existing methodologies of pkt monitoring
  - Packet sampling (sFlow, e.g., 1 out of 1000 pkts)
  - Mirroring/tapping with NPB (Network Packet Broker)

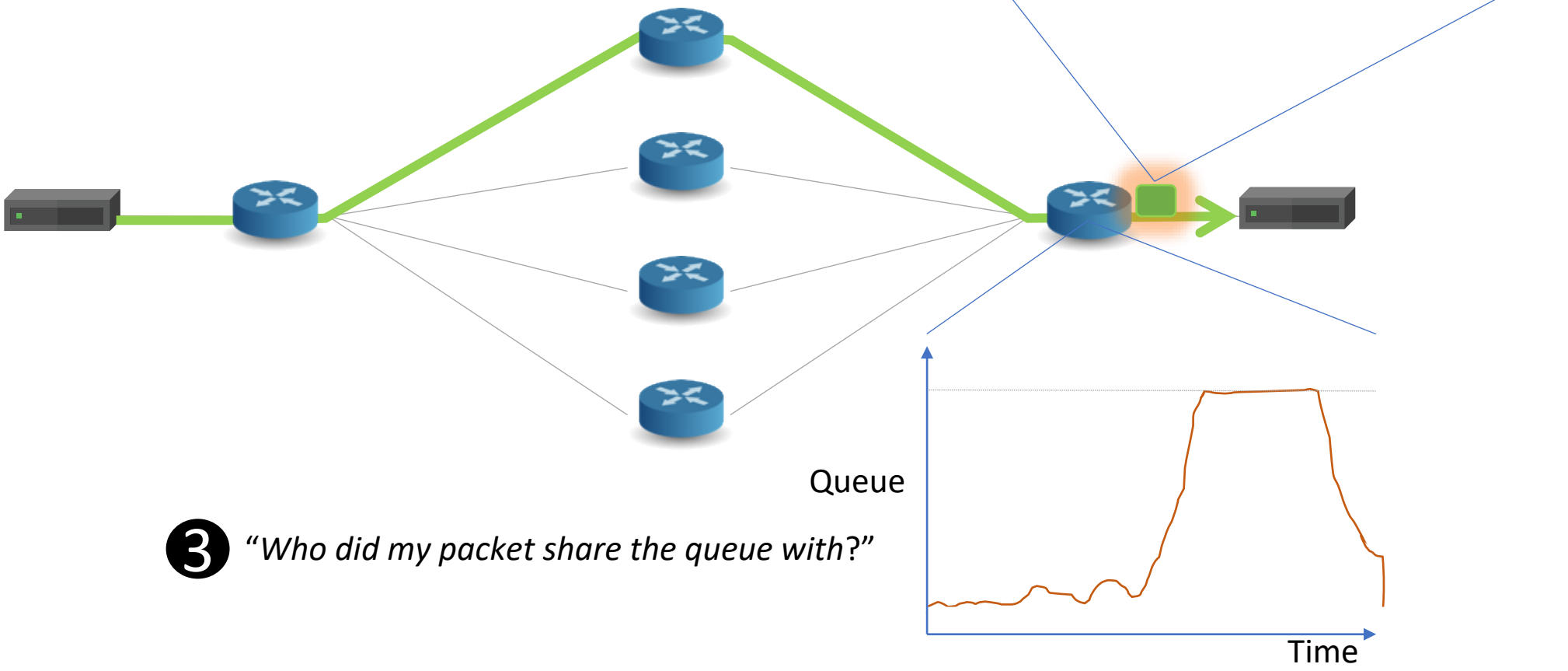
INT: In-band Network Telemetry
a.k.a
Data-plane Telemetry

❶ *"Which path did my packet take?"*



❷ *"How long did my packet queue at each switch?"*

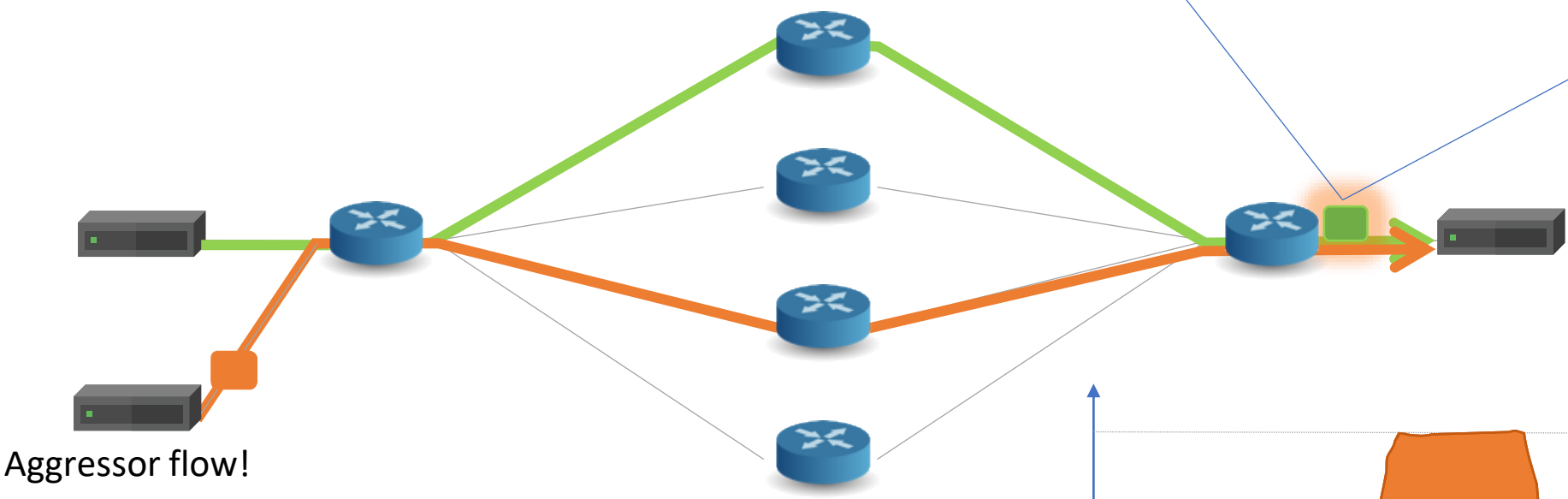
"Delay: 100ns, 200ns, 19740ns"



❸ *"Who did my packet share the queue with?"*

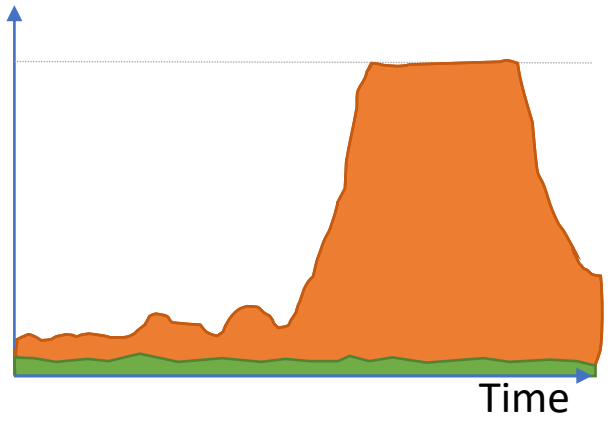
❷ *"How long did my packet queue at each switch?"*

"Delay: 100ns, 200ns, 19740ns"



Aggressor flow!

❸ *"Who did my packet share the queue with?"*



INT is the true network telemetry and more

- Packet monitoring with rich dataplane info for trouble shooting
  - timestamps, in/out ports, queueing info, drop reasons, …
- In-dataplane **event detection**
  - Packet-level visibility but with much reduced monitoring traffic

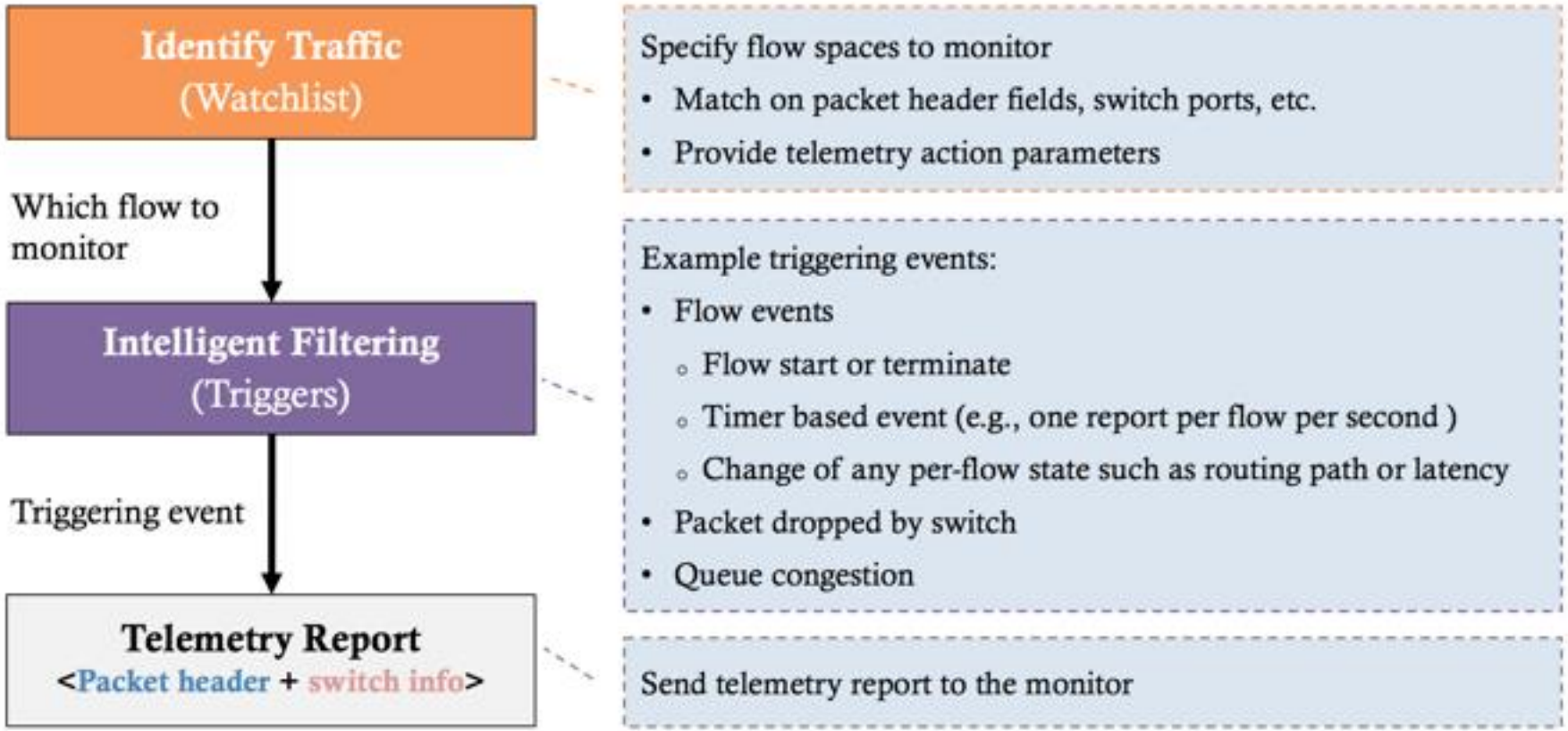
- INT is a debug trace and also a perf indicator
  - Enable in-band closed loop control such as Alibaba HPCC (High-Precision Congestion Control)

Systems aspect of INT service

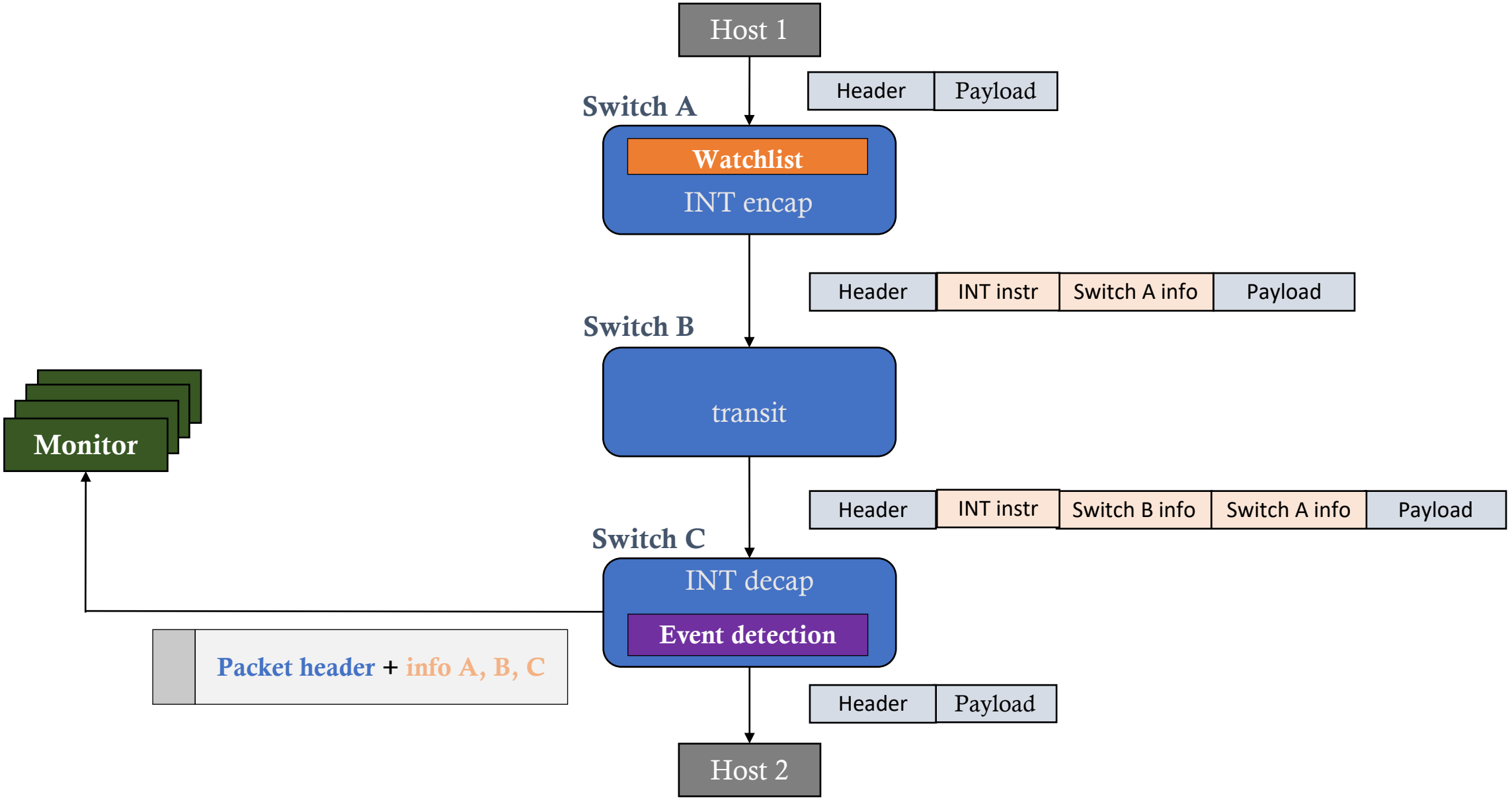
- INT has three functional components
  - config: determine which flows to monitor, which metadata to collect
  - report: produce telemetry reports as instructed by the config w/ change detection
  - consume: analyze the data in a remote monitor (e.g., Intel DeepInsight), or consume in-band for closed-loop control (e.g., HPCC)

- There are many ways to deploy INT functions
  - Devices: [endhost, switch, collector] X functions: [config, report, consume]

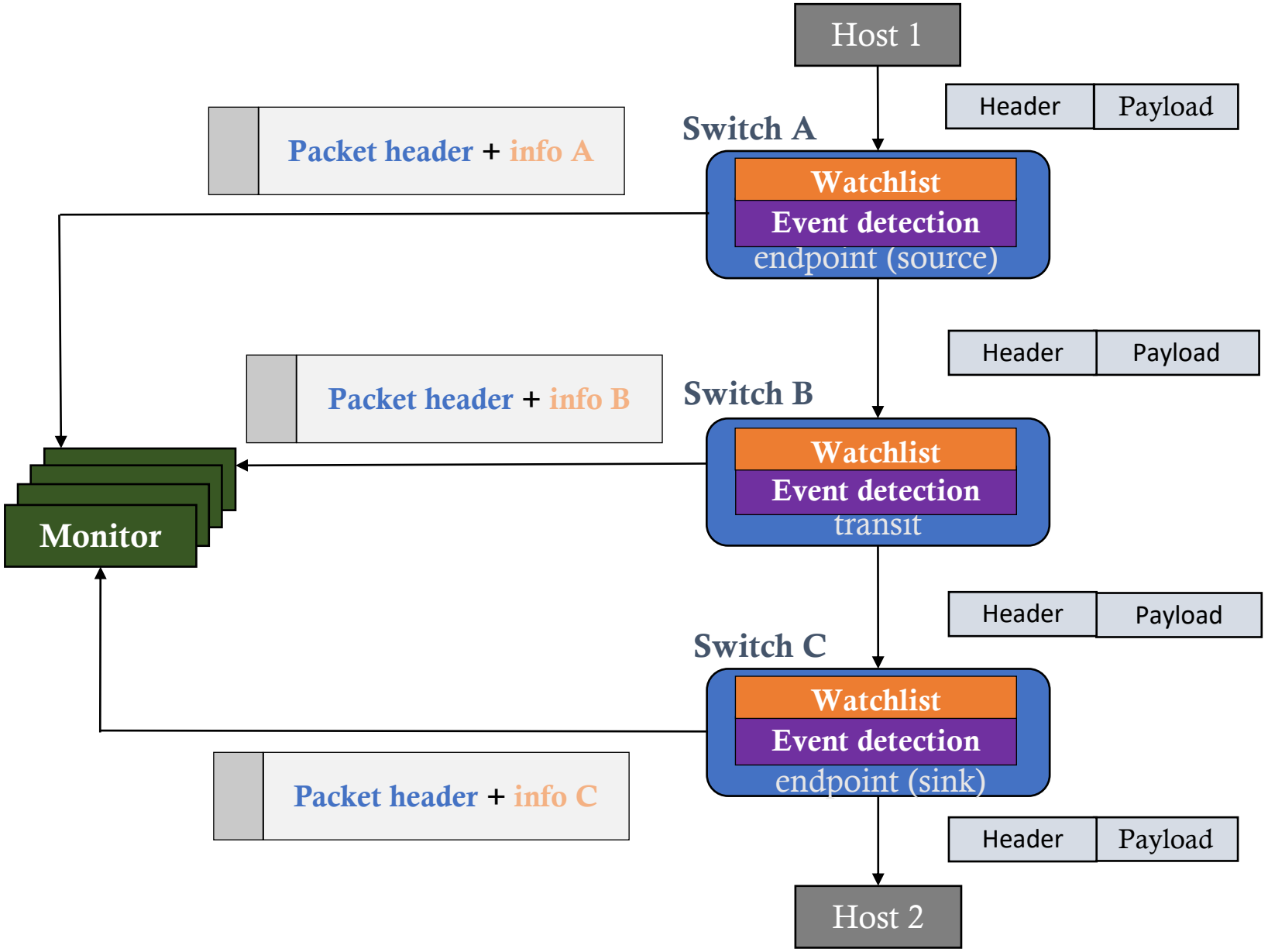
INT logical functions



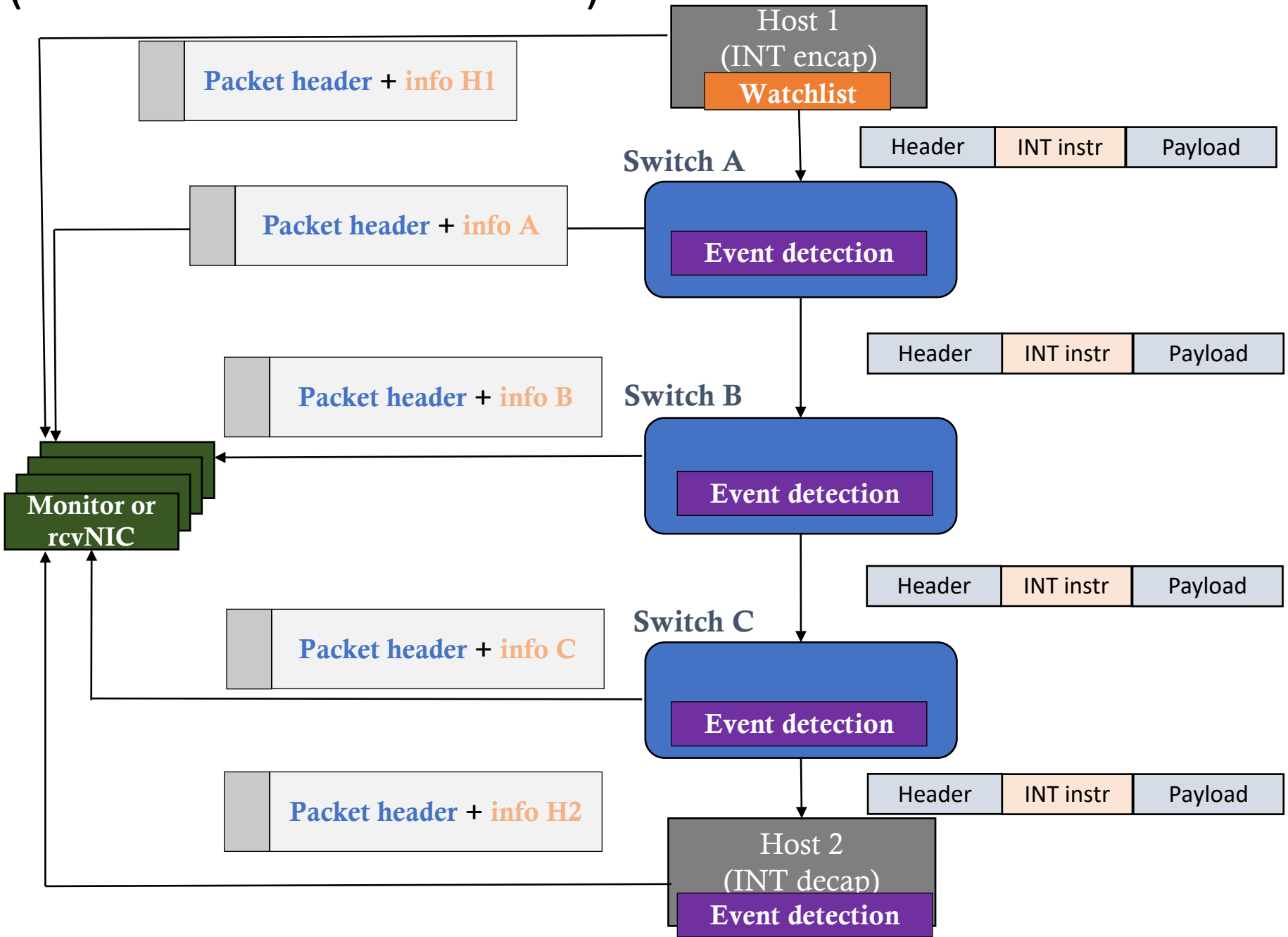
Classic INT, aka INT-MD (eMbed Data)



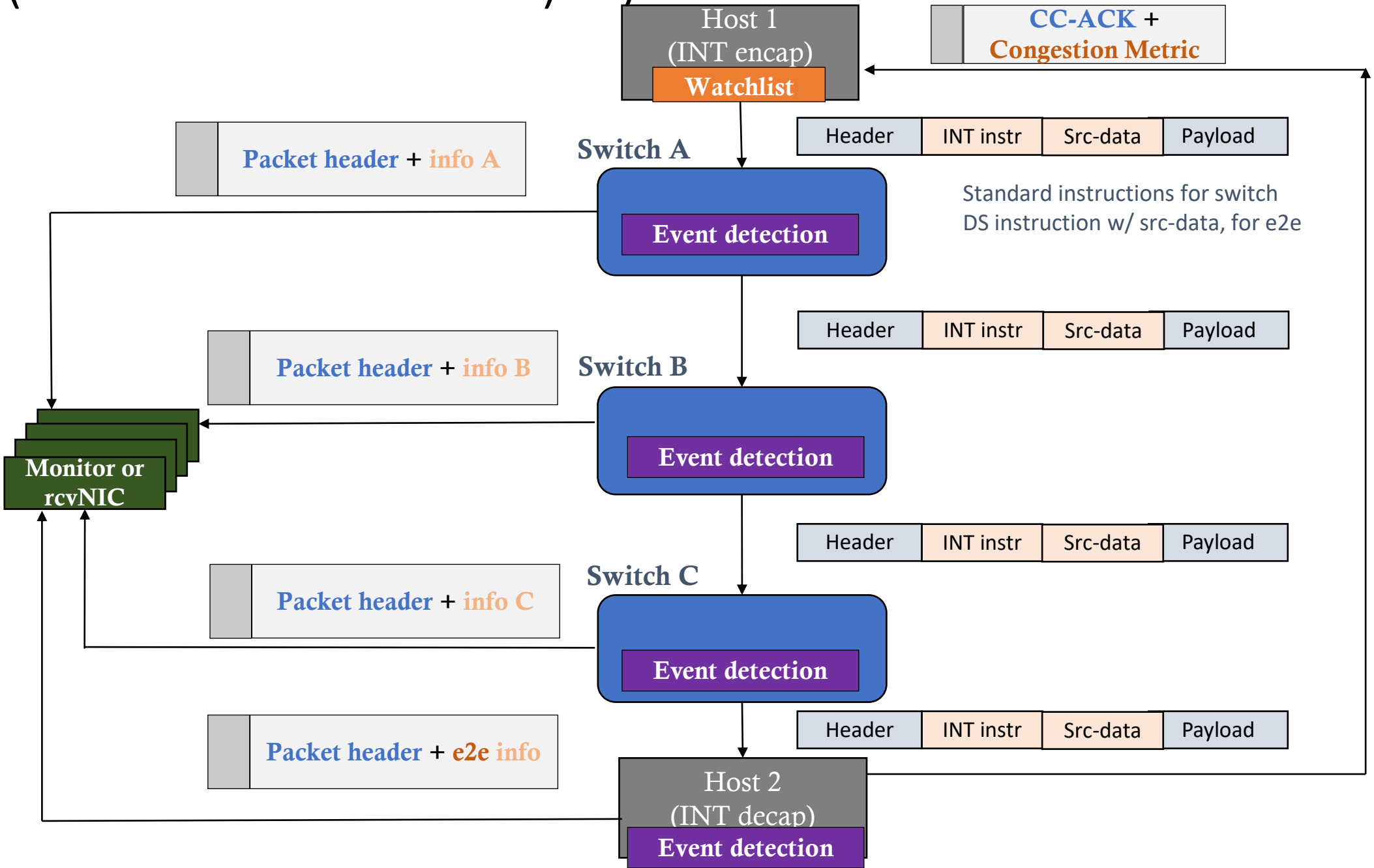
INT-XD (eXport Data) … aka Postcards



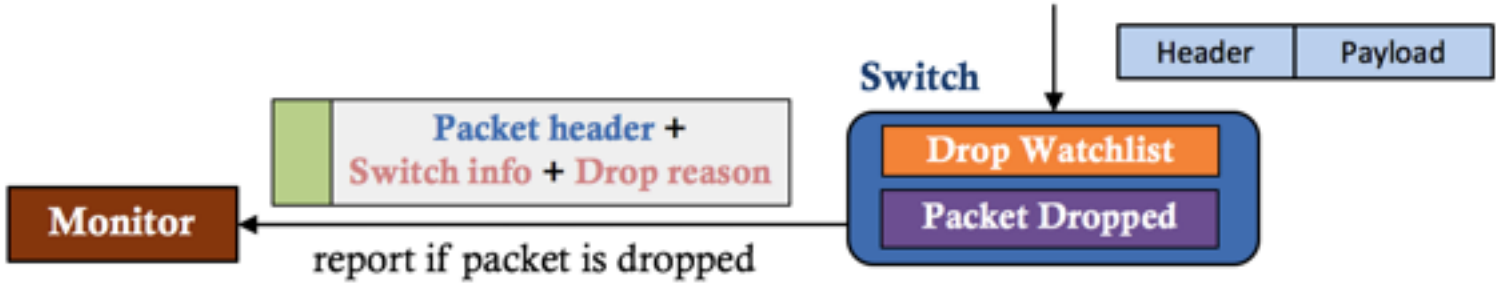
INT-MX (eMbed instruction)

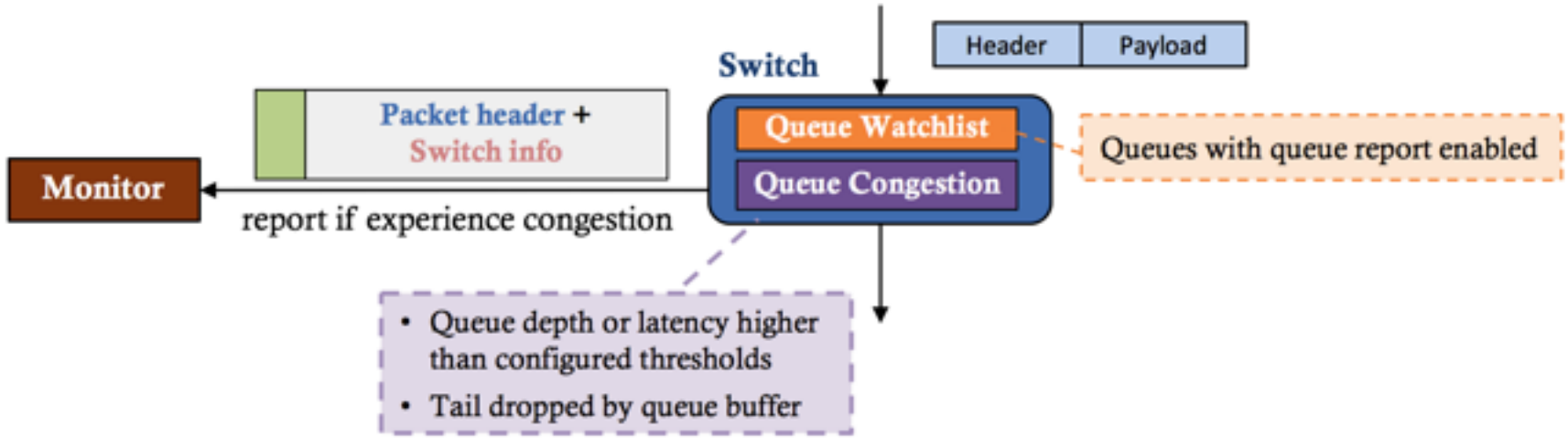


INT-MX (eMbed instruction) w/ src-data



Drop report & Queue report (switch local events)



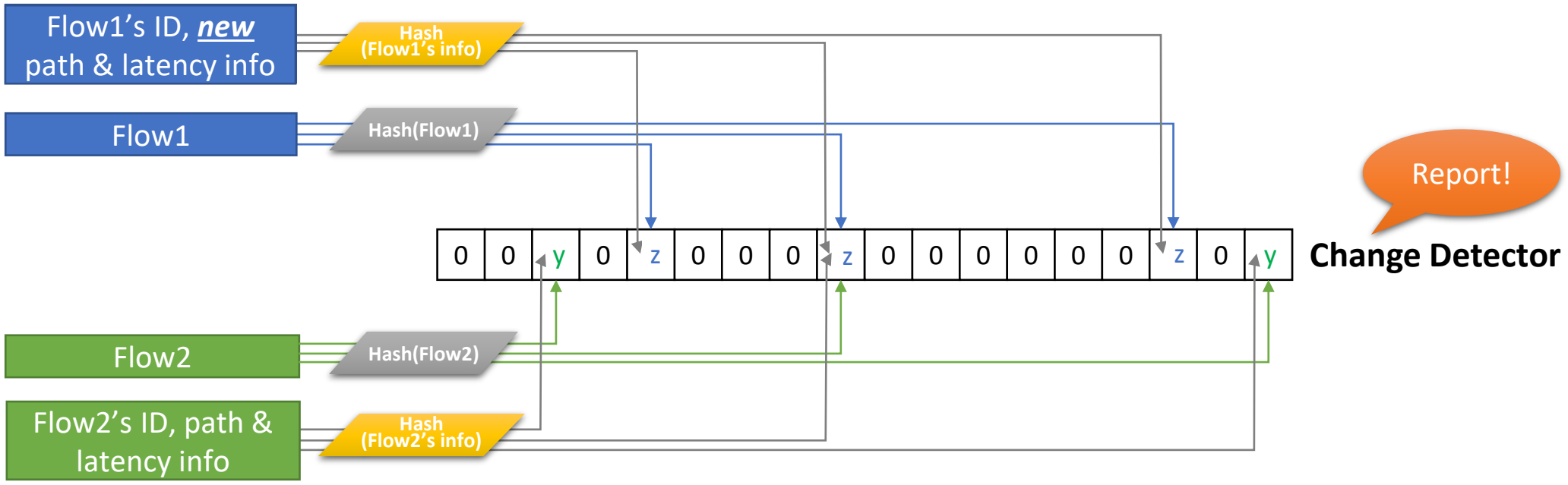


Challenge of Flow Reporting: speed vs scale

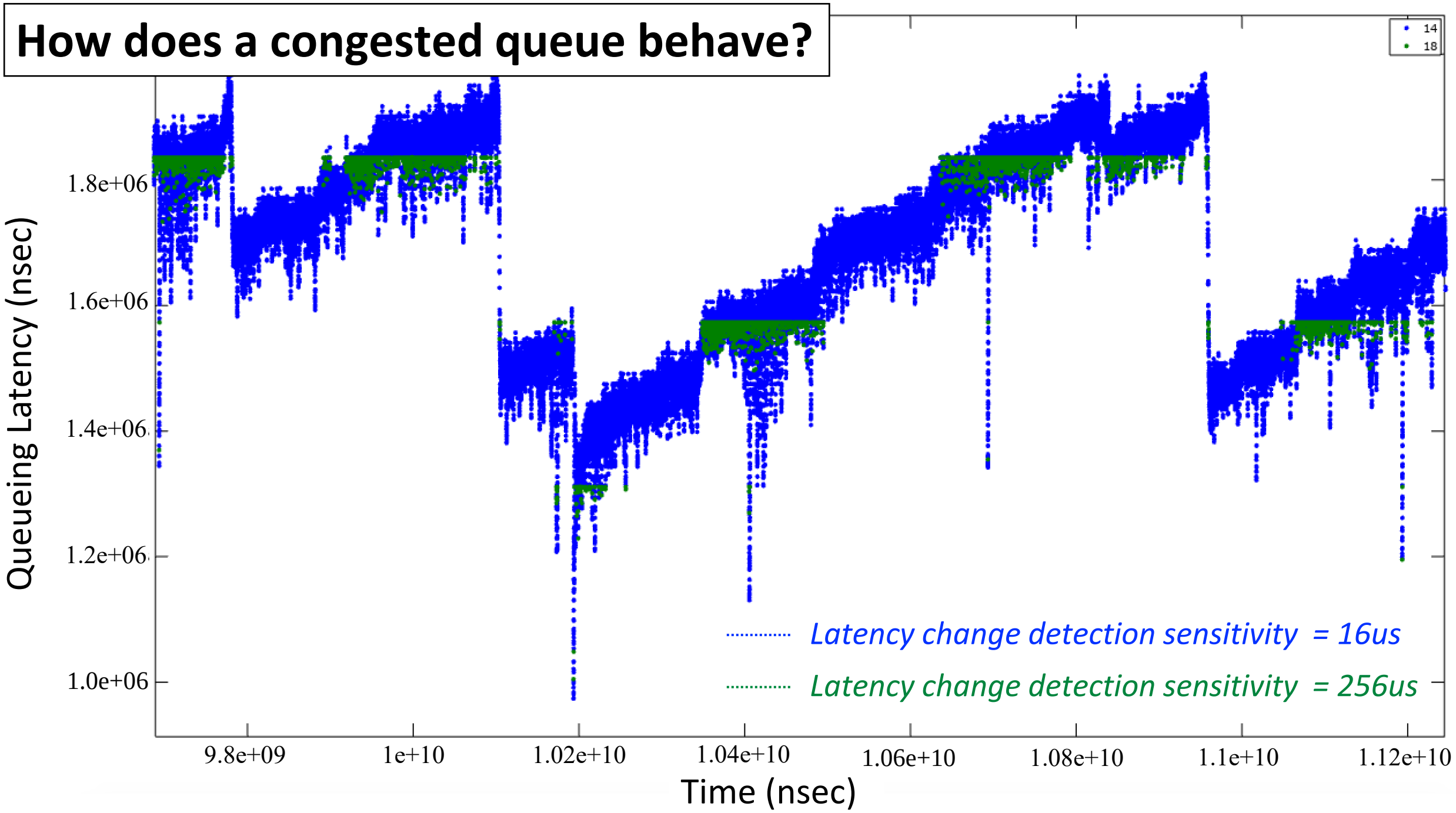
- Very high speed
  - 12.8Tbps or 6Bpps per switching chip (ASIC)
  - Even with four parallel pipelines (cores), one pipeline needs to handle 1.5Bpps; process one packet every 670 psecs
  - DRAM read or write takes tens of nsecs; SRAM is the only viable option
- Limited amount of on-chip SRAM
  - 10s~100s of MBs, shared with tables for various other features
- Scalable flow state tracking is critical

Flow Change Detector

- Inspect every packet, and report only relevant ones
  - Keep the digest of flow's info (ID, path, latency, etc.) in the hash table
  - Report when at least one of the cell values differ
  - Low false negatives, reasonably low false positives when # of flows is smaller than the soft capacity of change detector



How does a congested queue behave?



INT pointers and standard efforts

Early work

- Millions of Little Minions: Using packets for low latency network programming and visibility, Sigcomm 2014.
- In-band Network Telemetry via Programmable Data Planes, Sigcomm 2015 Demo.

P4.Org Apps WG leads INT specs (since 2015) with VMware, Cisco, Arista, …

- Currently v2.1 (https://p4.org/specs/)
- de-facto open standard in the industry and research community

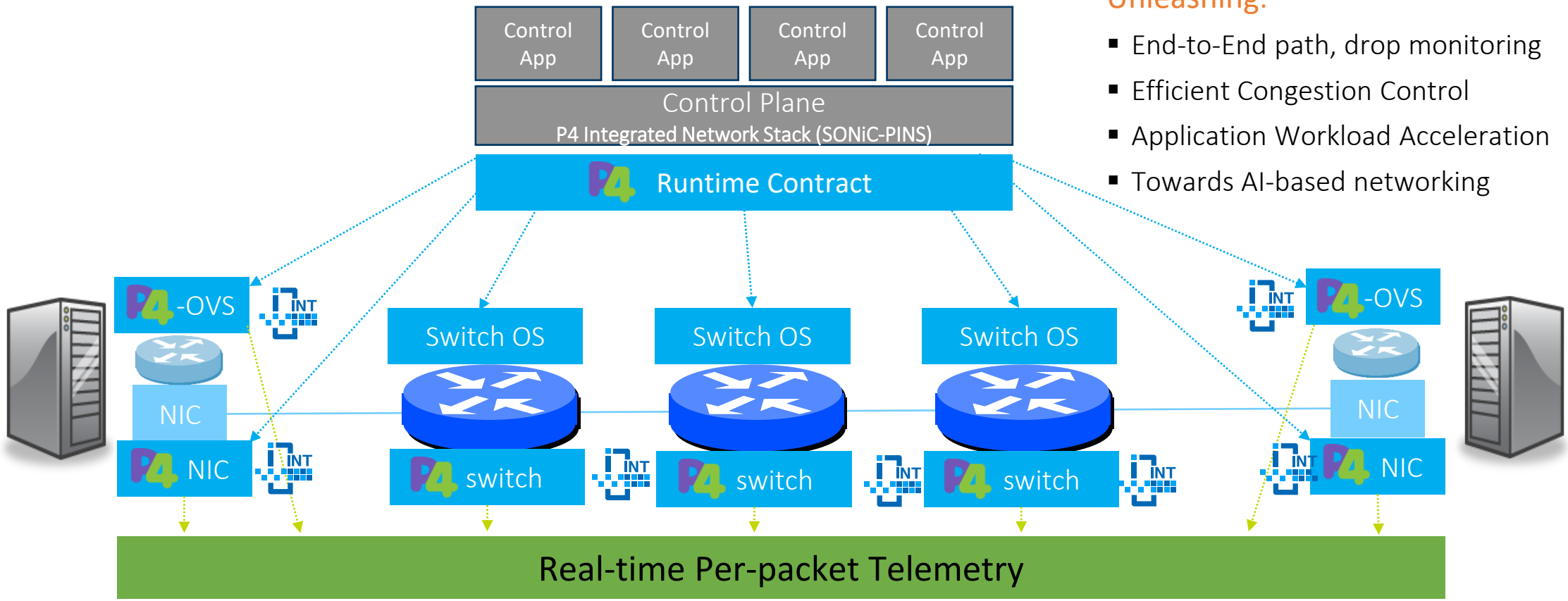
IETF followed, started IOAM in 2016. RFC standard expected soon.

IFA (In-band Flow Analyzer) draft

- template-based, header format not part of the draft

INT use cases

INT Vision: Telemetry-based control and mgmt

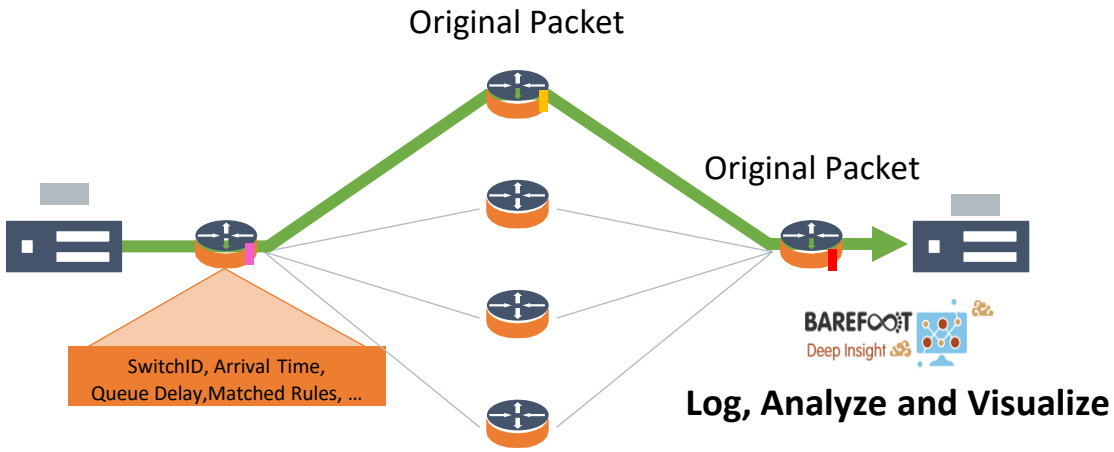


Unleashing:

- End-to-End path, drop monitoring
- Efficient Congestion Control
- Application Workload Acceleration
- Towards AI-based networking

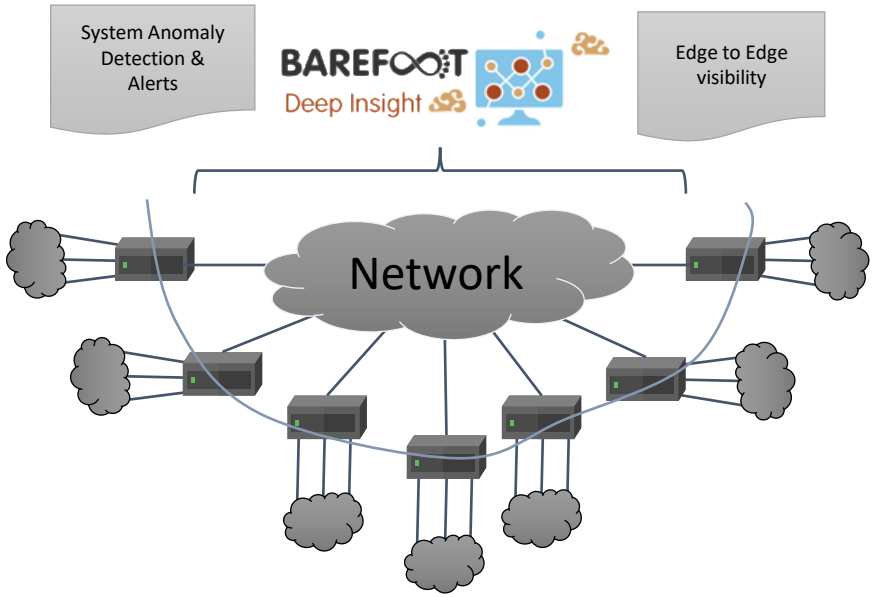
INT deployment options

Switch INT



- Rich telemetry about network

Host Based INT

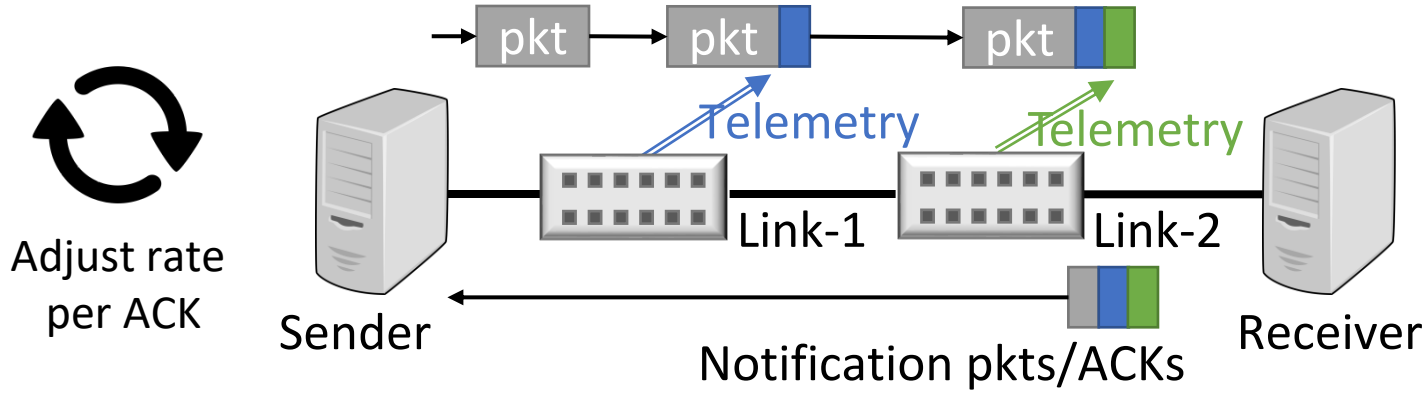


- Kernel Module
- VNF/VM

HPCC: High Precision Congestion Control (SIGCOMM'19, Alibaba/Harvard/MIT/…)

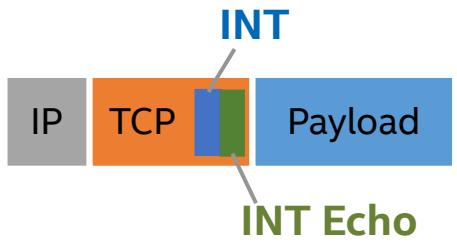
https://datatracker.ietf.org/doc/draft-miao-iccrg-hpccplus/

- New commodity ASICs have in-band telemetry ability
- Use in-band telemetry as precise feedback for congestion control

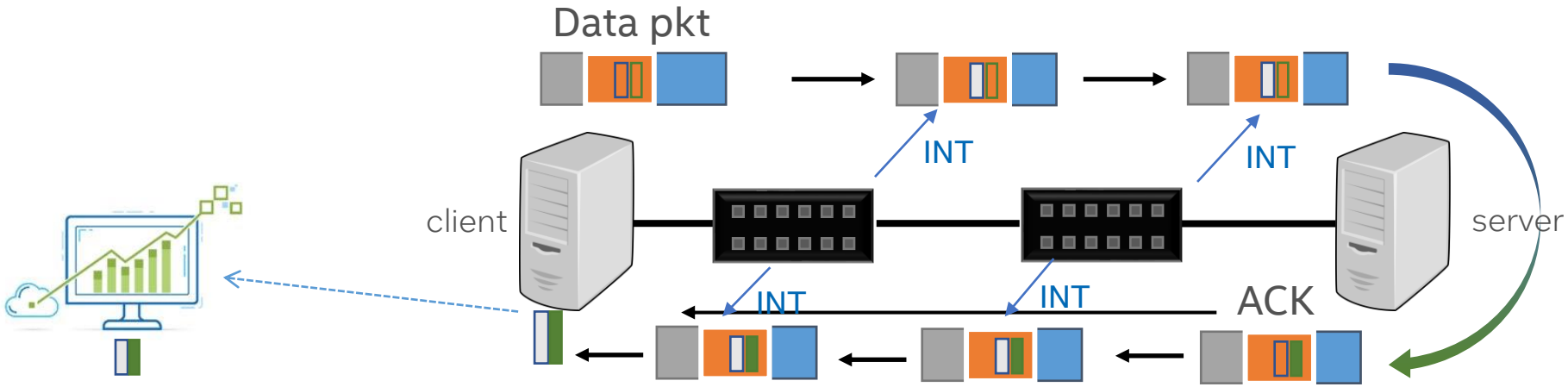


- Enables precise window increase/decrease control

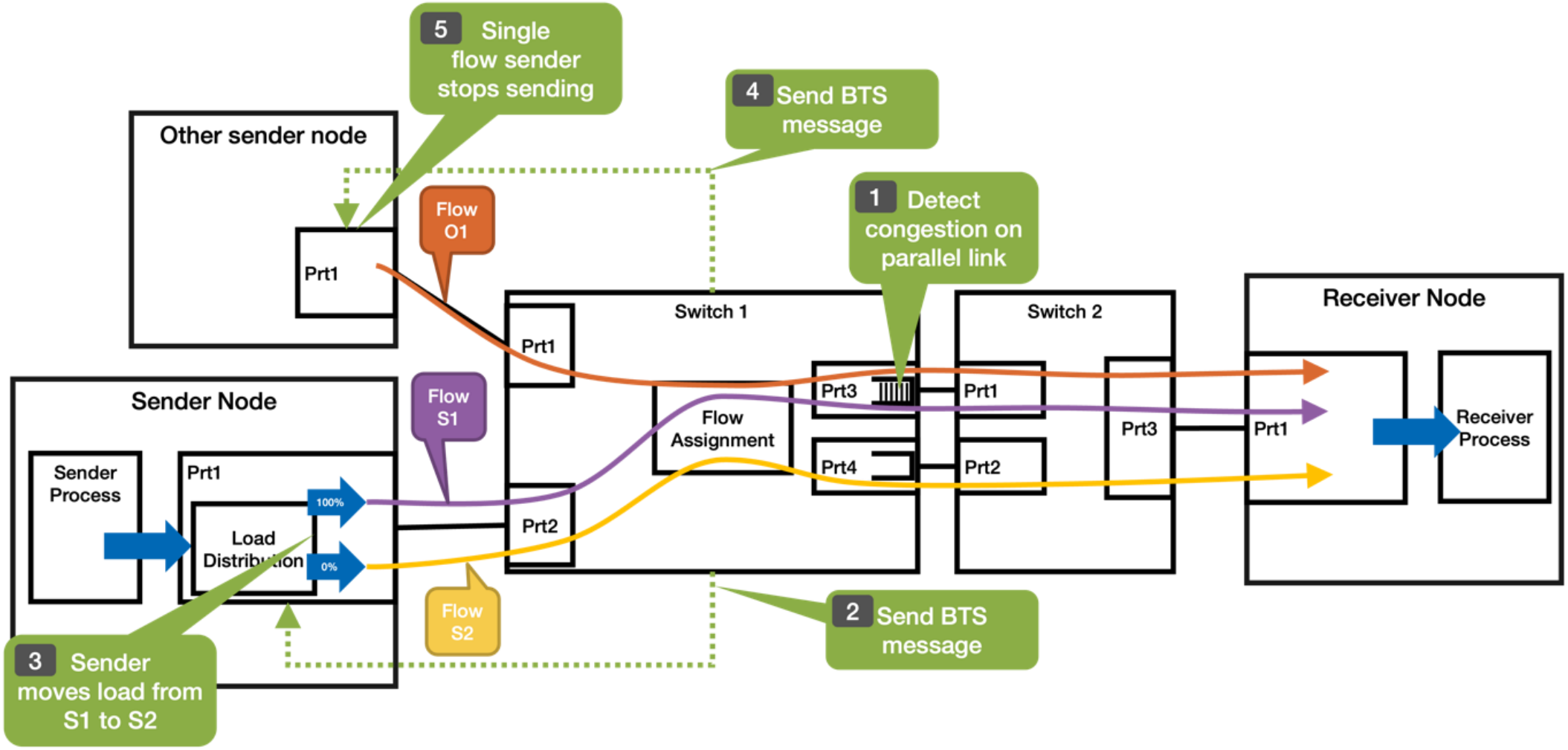
TCP-INT (soon to be open-sourced)



- INT stack reduced and embedded in the TCP header (as TCP Option)
  - Works with NIC HW offloading of TCP checksum and segmentation
- Correlates fabric telemetry (Q depth) with TCP states (congestion window)
- Lightweight: metrics aggregated (max or sum) over switch hops

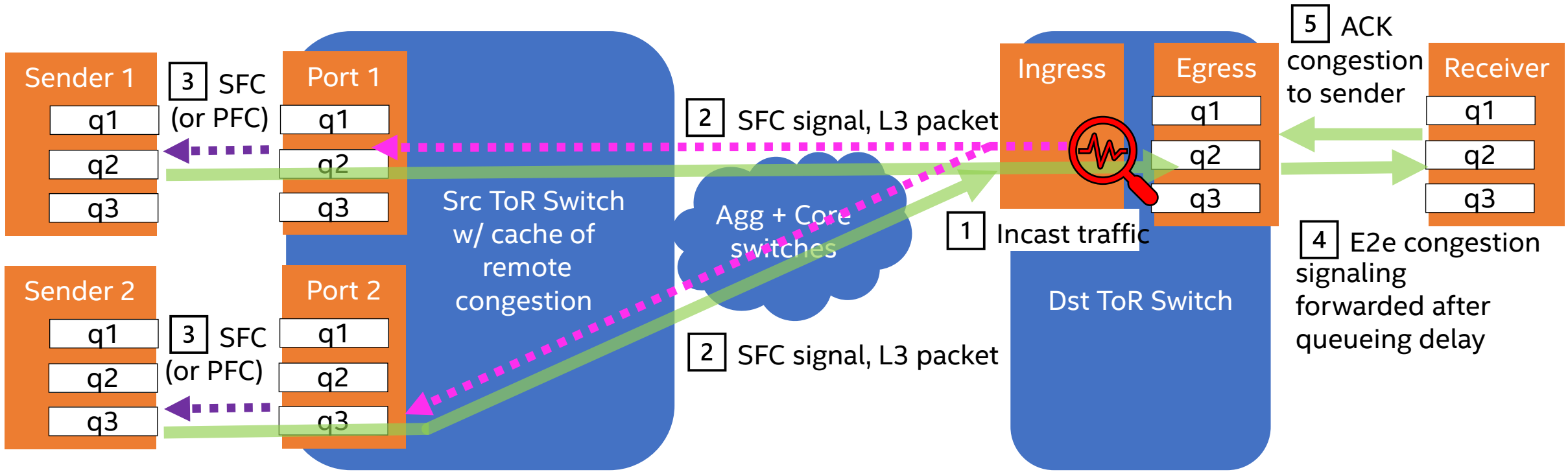


Back-To-Sender telemetry



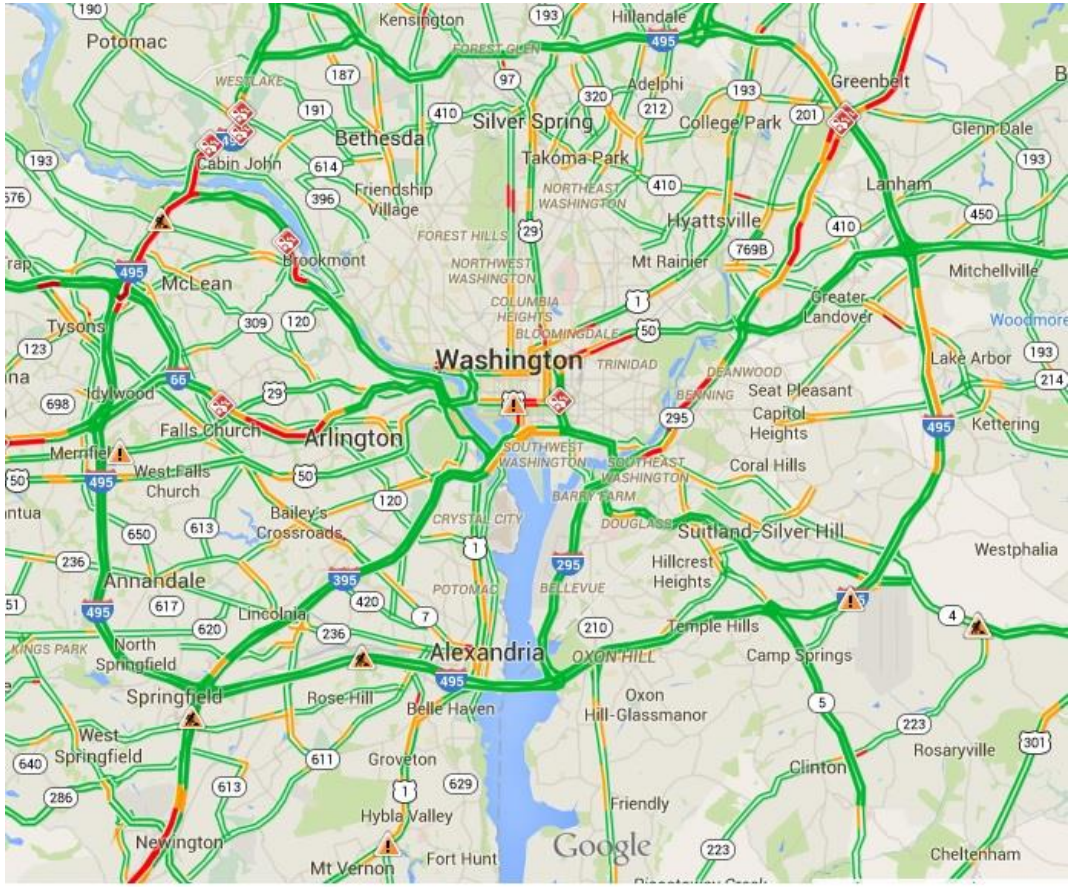
One use case of BTS signaling: Source Flow Control @ IEEE

- Backward signaling of congestion + 1st hop flow control by standard PFC (Priority Flow Control)
- Signaling + flow control all in congestion-free sub-RTT
- Standardization process started in IEEE 802.1
  - Motion for PAR/CSD approved: link to the material.



Telemetry for control → towards “Maps” service

- Precise telemetry
  - For senders to acquire available bandwidth asap while controlling congestion
  - E.g., INT (Inband Network Telemetry)
- Fast ‘back to sender’ signaling
  - Realtime telemetry signal, decoupled from on-going congestion or failure event
  - Enable VoQ queueing at network ingress
- Fine-grained / adaptive load-balancing
  - For non-blocking, full-bisection fabric
  - Fast reaction to congestion or failures



Conclusions

- Measurement is crucial to network operations
  - Measure, model, control
  - Detect, diagnose, fix
- Network measurement is challenging
  - Very high speed vs. limited h/w technology
  - Large volume of measurement data with multi dimensions
- Programmable data planes open exciting new possibilities
  - “Data-plane Telemetry” -- such as INT -- for fine-grained ground-truth information
  - Input to true AI-driven networking

Industry direction

- Dataplane telemetry not deployed “at scale”, still at early stage
- Telemetry solution must scale better, in terms of #reports overhead, #flows to monitor
- Telemetry “for immediate control” → direct improvement of tail latency, BW utilization and TCO
  - TCO (total cost of ownership)
  - Control by end-host or SDN controller
  - Direct correlation/integration with transport or application info is critical Example: TCP-INT
- Switch driven pkt telemetry evolving into
  - 1) transport/host-centric telemetry and
  - 2) device telemetry (data + ctrl plane co-optimization)

NetSeer

•To what extent can these systems protect against or attribute the cause and underlying actor involved in a DDoS attack or similar countermeasure to disrupt network performance? Certainly we can determine the type of incident and isolate it, but can we also distinguish between a forced malfunction and an organic one?

•Can you speak to your experience with academic research in industry? I was struck by the fact that this paper was written by Alibaba and noted that many of our papers were written by industry practitioners (e.g. Google as well). Can you speak to the differences you have noted between University vs. Industrial research? Would love to discuss your experience.

•Could NetSeer easily be adapted for per-flow per-hop information collection, like that done by LightGuardian? Could similar reductions in data volume be achieved for more general information than NPAs?

•NetSeer defines several preconditions on middleboxes: inter-device drop awareness, event-based anomaly detection, and reliable report. These are of course doable in simulation, but in practice, are these preconditions reasonable to expect?

•Is event data sufficient to inspect network problems? Are there other data needed?

•I would be interested to see how sensitive Figure 3 is to the architecture of the network. I am a bit surprised that more packet drops are caused by pipeline rejection as opposed to congestion. What implications does this have for our assumptions made for congestion control?

•How well can different network telemetry schemes interoperate? Is this something that's commonly studied? It seems like the approaches in these two papers are tailored towards different measurement needs (holistic traffic measurement versus anomalous event detection) and though each is optimized to scale relatively well, I could imagine that the cost of multiple measurement schemes could add up. How do network operators typically navigate that tradeoff?

•One technique that NetSeer does to accomplish batching is to instead of build packet in PDP memory, it takes advantage of the circular event batching packets (CEBPs) to accumulate the events to be batched. What are other common scenarios that utilize CEBPs? In addition, even though the paper says there are little overhead, in what scenarios would storing directly in PDP make sense vs sending/receiving circular packets?

LightGuardian

•Has further experimentation been conducted to assess the ability of LightGuardian to satisfy the same constraints of full-visibility, low overhead, and robustness at scale? Given that the prototype developed consisted of 10 switches and 8 end-hosts, how can we know the viability of this system in hyper-scale networks?

•What is the state-of-the-art for automatically adjusting the system parameters according to the current traffic patterns?

•It seems that authors of LightGuardian are not embedded in a cloud company, unlike the authors of NetSeer - how quickly do you expect LightGuardian to be taken up by industry, if at all? What would be some barriers?

•LightGuardian takes about 0.07% additional bandwidth - are there cases where this would be considered non-negligible, and would prevent the adoption of the method?

•In the future work, the authors suggest a system in which system parameters can be automatically adjusted according to traffic characteristics. It seems like there are many parameters in play here--probability thresholds, sketchlet sizes, the "k" variable in k+chance selection; the authors do not specify which parameters they might toggle. Do you have any ideas about which parameters might be most relevant for adjusting according to traffic characteristics?

•This system requires switch and routers to implement the algorithm so that the data can be collected and analyzed. Is this a limitation to make it work? For flow-level statistics, some data makes sense only after the last packet is completed transmitting. How can the final data transmitted to end hosts?

•I think it would be interesting to look into how something like fountain codes (LT codes might do [https://en.wikipedia.org/wiki/Luby_transform_code]) could be used to improve sketch reconstruction in the face of dropped sketchlets. The encoding/decoding would be done on a per-device basis. Transmission and reconstruction isn't too latency sensitive so I would feel that the encoding/decoding time wouldn't be prohibitive. However, this might introduce some complexity since the sketchlet is sent with some metadata.

LightGuardian

•The SuMax paper gave a pretty specific set of metrics that this iteration was focusing on reporting - estimating latency, detecting packet drops, detecting abnormal jitters, and tracing forwarding path. I was wondering how universal this set of metrics is, and what applications this knowledge best serves. Are there many networking applications that are taking advantage of the flexibility of this system to focus on other important metrics?

•The authors claim that most measurements can be gathered using the sum and max operators. To what extent are measures of variance needed for these network measurements? I could imagine that some measure of variance can be gathered over time as measurements come in, but it also seems like there's some uncertainty in the measurements depending on the number of sketchlets that were able to be gathered. I wonder how we could use their existing framework to better support estimations of uncertainty in these metrics.

•The paper says it takes 4 seconds to aggregate all sketchlets which seems like a long time compared to total RTT for flows especially in data centers - so how useful could LightGaurd really be in provided real time analysis on network traffic (like for congestion control as they suggest in the conclusion)?

•NetSeer seems to be explicitly designed for a cloud environment, while LightGuardian explicitly focuses on making its sketches simple enough to be implementable on generic programmable switches, to avoid being locked to a cloud/datacenter setting. Do any non-cloud networks (e.g. ISPs) implement mechanisms similar to LightGuardian today to collect flow telemetry? Also, in general, how helpful is flow telemetry in a network with a large number of short flows?

•Both works we read today set up a testbed to evaluate the efficacy of their network monitoring solutions. But how do we determine whether the testbed is "large enough"? (e.g, LightGuardian used 16 hosts, 20 switches, whereas Flow Event Telemetry used 10 switches and 8 servers).

•Which network settings yield themselves better to LightGuardian's robustness principle and which are better suited for NetSeer's accuracy guarantee?

•The paper mentions that SuMax (sum and max) should be able to estimate all packet attributes. However, one of the other measurements mentioned in the paper was the ability to store packet trace information. SuMax sketch would not able to do achieve tracing functionalities, but is mentioned as "could be easily implemented in the switch pipeline". How would the switching pipeline implement tracebacks, which could prove to be very useful in debugging packet route paths, etc?