TCP-INT: Lightweight INT in TCP Transport

Simon Wass (Intel) & Miao, Mao (Baidu)

Agenda

- Motivations
- TCP-INT High-Level Design
- Example use cases
- Implementation (Intel)
- Demo
- Congestion Control Enhancement
- Implementation (Baidu)
- Baidu Vision & Planned Usage
- Roadmap

Contributors

Intel

Theo Jepsen
Grzegorz Jereczek
Bimmy Pujari
JK Lee
Simon Wass

Baidu

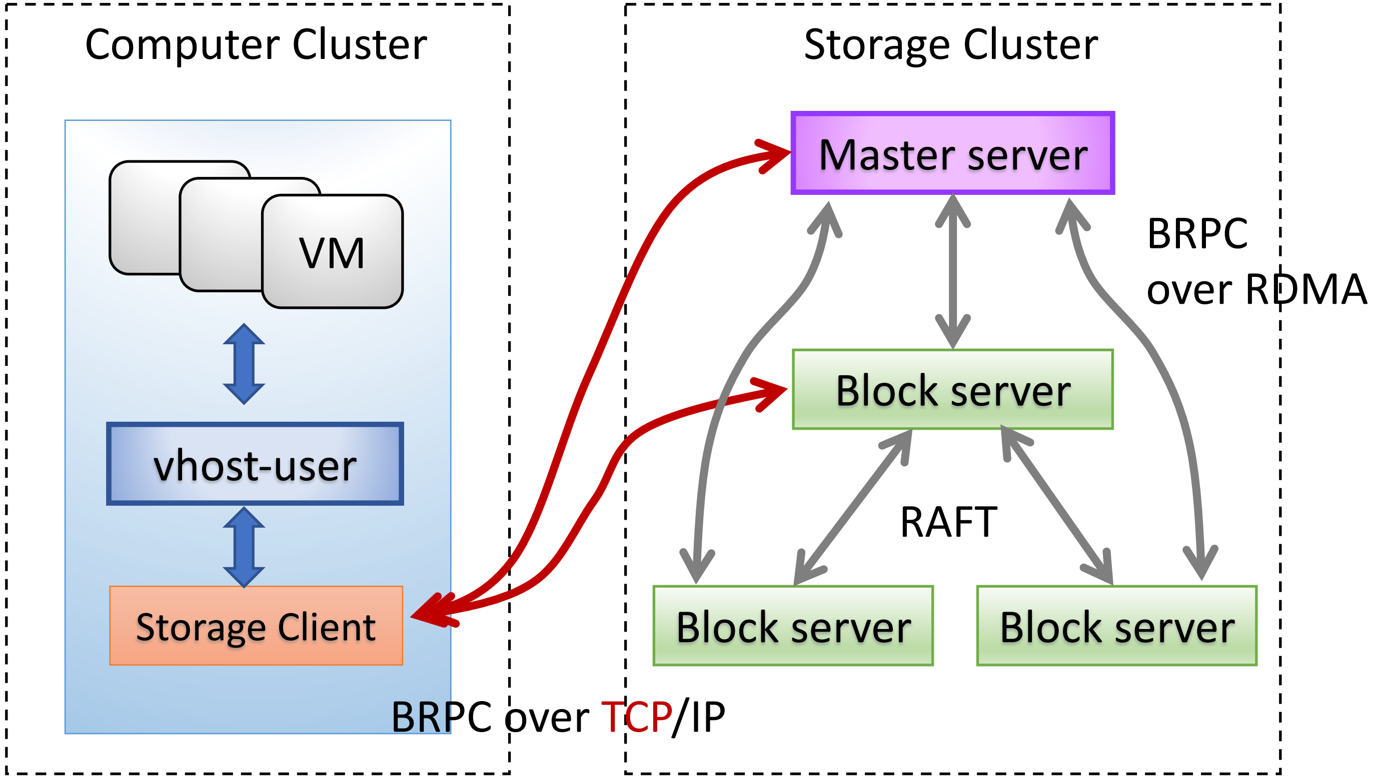
Miao, Mao
Cheng, Gang
Li, Zhaogeng
Xie, Pan

Motivations

- The separation between storage and computation requires a low latency, high throughput fabric to realize the benefit

- BaiduRPC over RDMA is used in the storage cluster

- Congestion & bottleneck exist between Storage client and master/block server due to the use of TCP

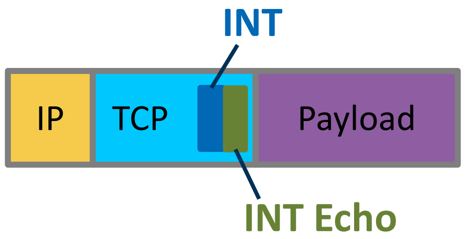


Motivations (2)

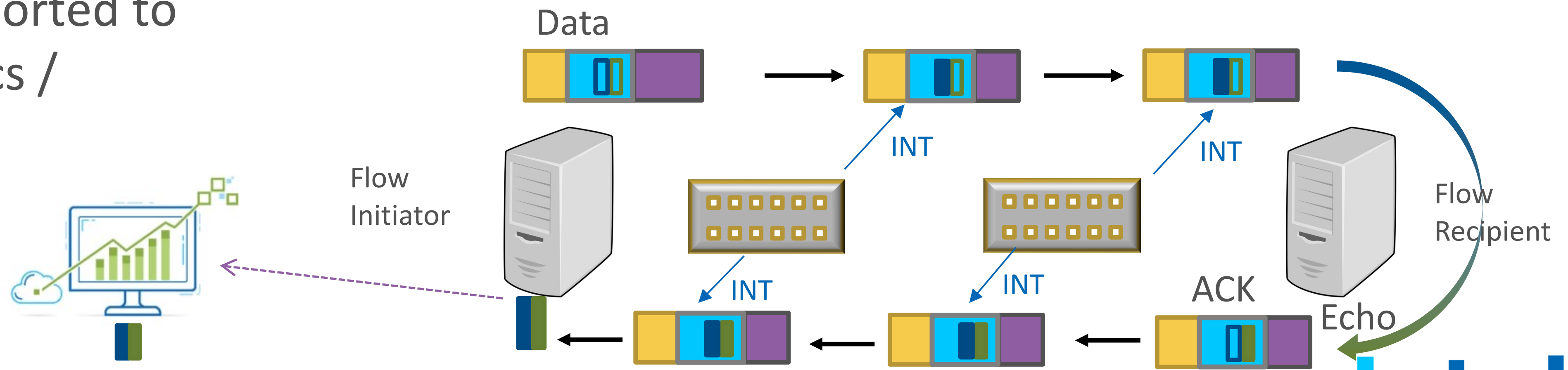
Feedback on INT

1. Flow Scale
2. Telemetry “for Control”
3. Direct mapping to application session/message

TCP-INT High-Level Design (3)

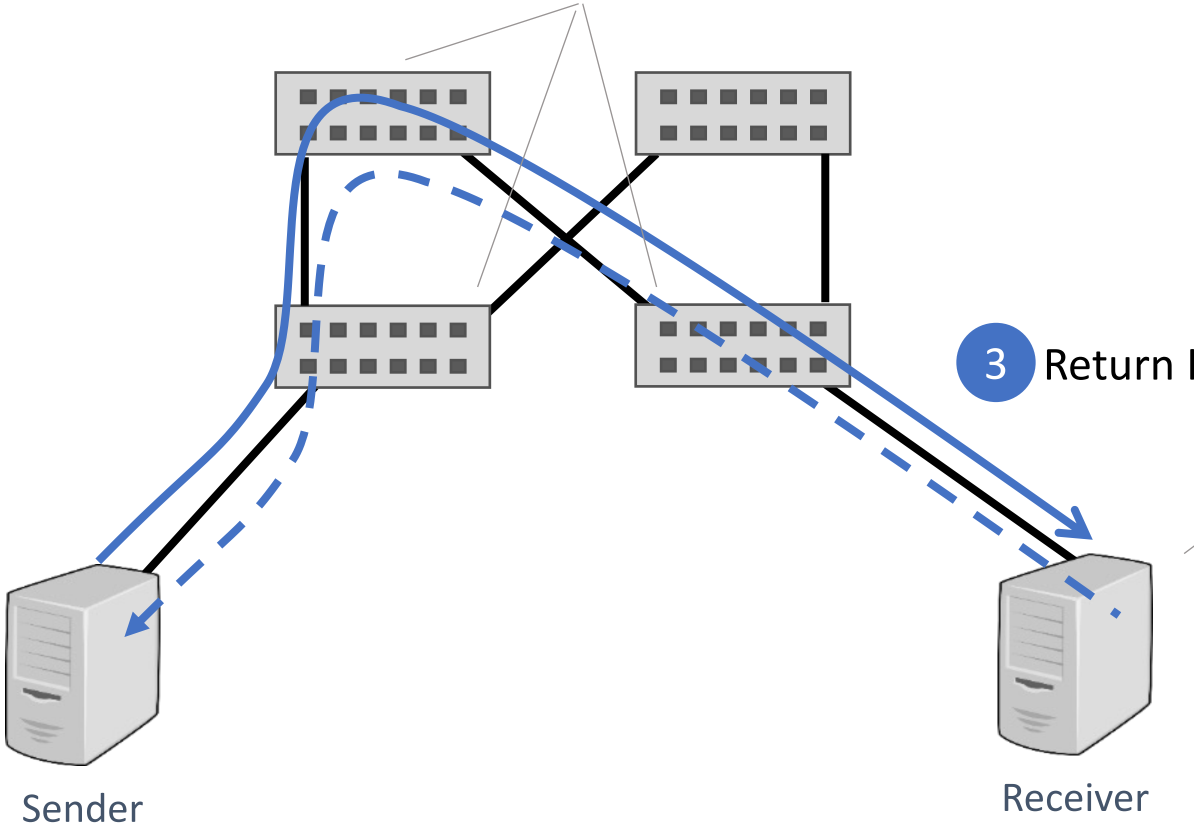


- In-band Network Telemetry (INT) embedded in the TCP header (as TCP Option)
- Correlates fabric telemetry (Q depth) with TCP states (congestion window)
- Lightweight: metrics aggregated (max or sum) over switch hops
- INT enriched with end-host metrics
- Consumed locally by application or exported to centralized metrics / visualization tools



TCP-INT High-Level Design

2 Switches update INT fields: sum latency and max qdepth, util

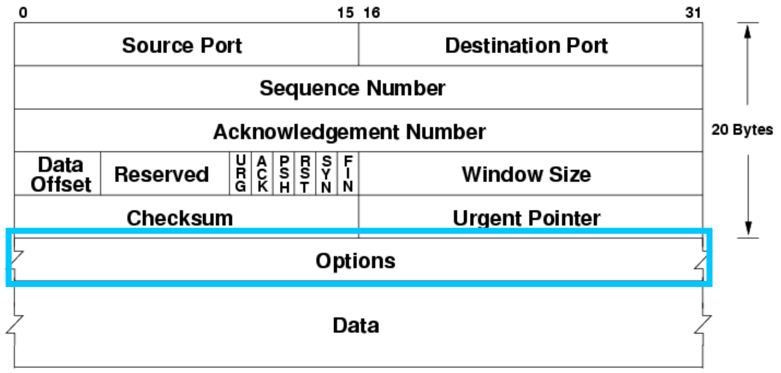


1 Kernel adds INT option to TCP header

3 Return INT header in ACK

TCP-INT High Level Design (2)

New TCP Option – TCP-INT



Used by the end-hosts to echo INTval and ID back to the senders, ignored by the switches

Updated by the switches

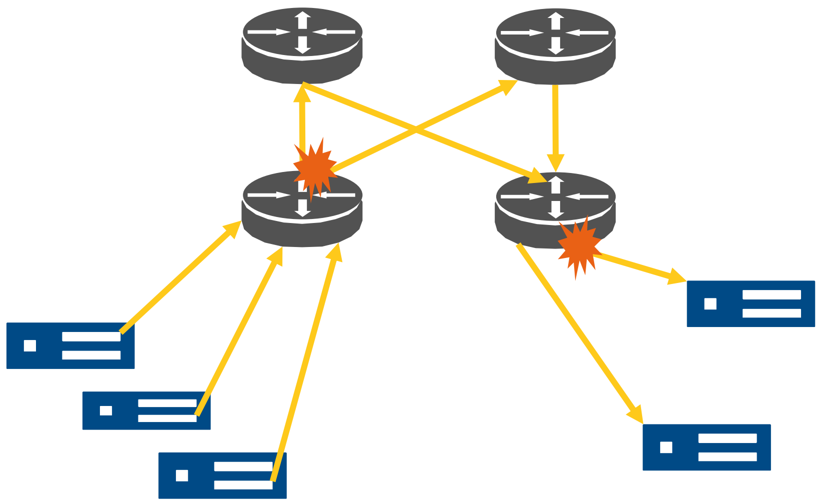
Option-kind (1B)	Option-length (1B)	**INTval** (1B) **(Scaled summation of utilization or queue depth)**	INTecr (1B)	**ID** (1B)	IDecr (1B)	**Lat** (3B)	LatEcr (3B)

0x72	0x0C	if qdepth < qdepth_threshold:
		Bit [7]: 0
		Bits [0-6]: **util** >> y
		(saturates if above max)
		else:
		Bit [7]: 1
		Bits [0-6]: **qdepth** >> x
		(saturates if above max)
			INTval Echo Reply	IP.TTL	IP.TTL Echo Reply	Lat += switchLat	Lat Echo Reply

Example Use Cases

Extend Host Linux TCP Toolset with TCP-INT Information

- Quick view into network state
- Compare different congestion control algorithms
- Develop new congestion control algorithms
- Debug QoS configuration in the fabric (e.g. WRED thresholds)



intel

Implementation

Intel Implementation Overview

End-hosts

Switch

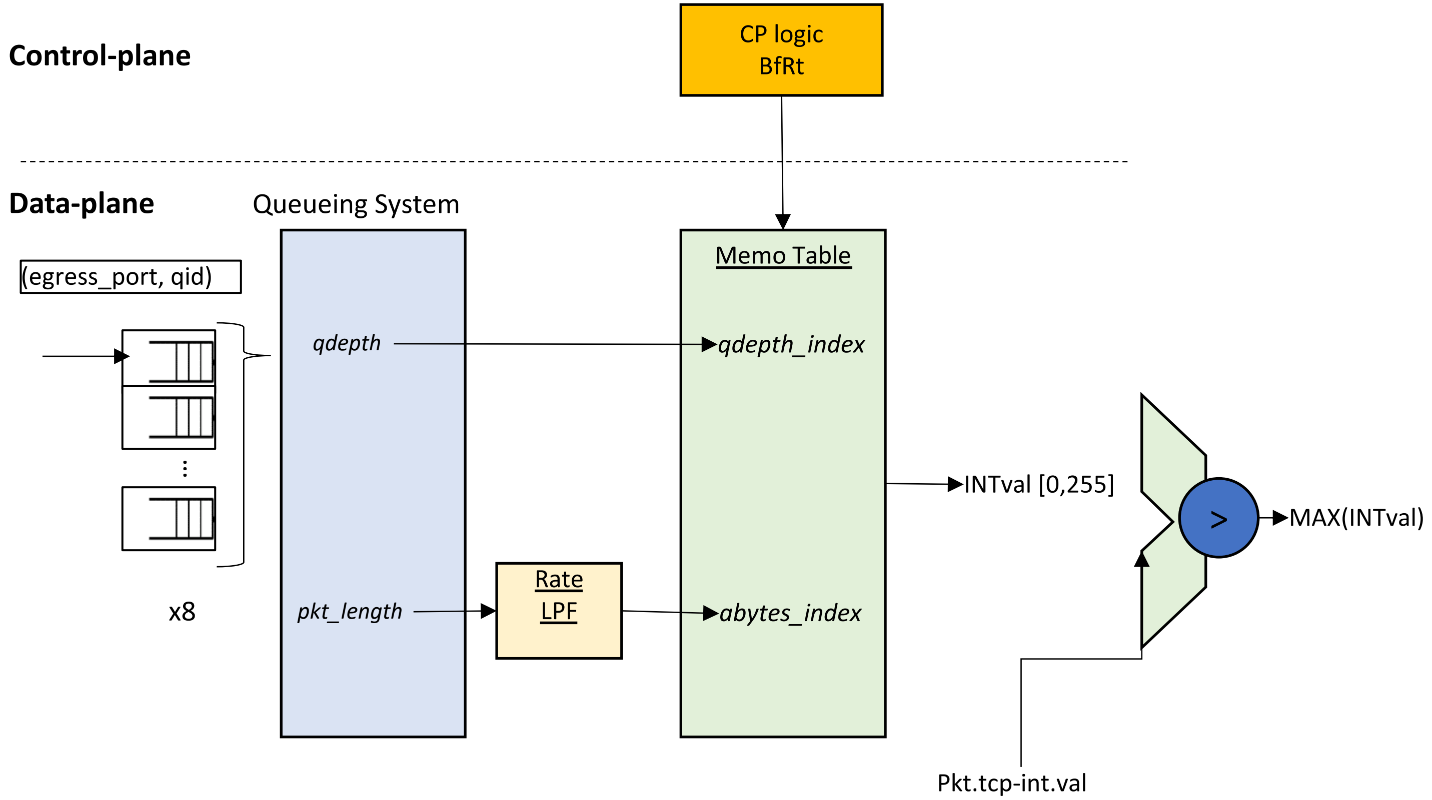
Implementation (2)

Host-side eBPF-based implementation

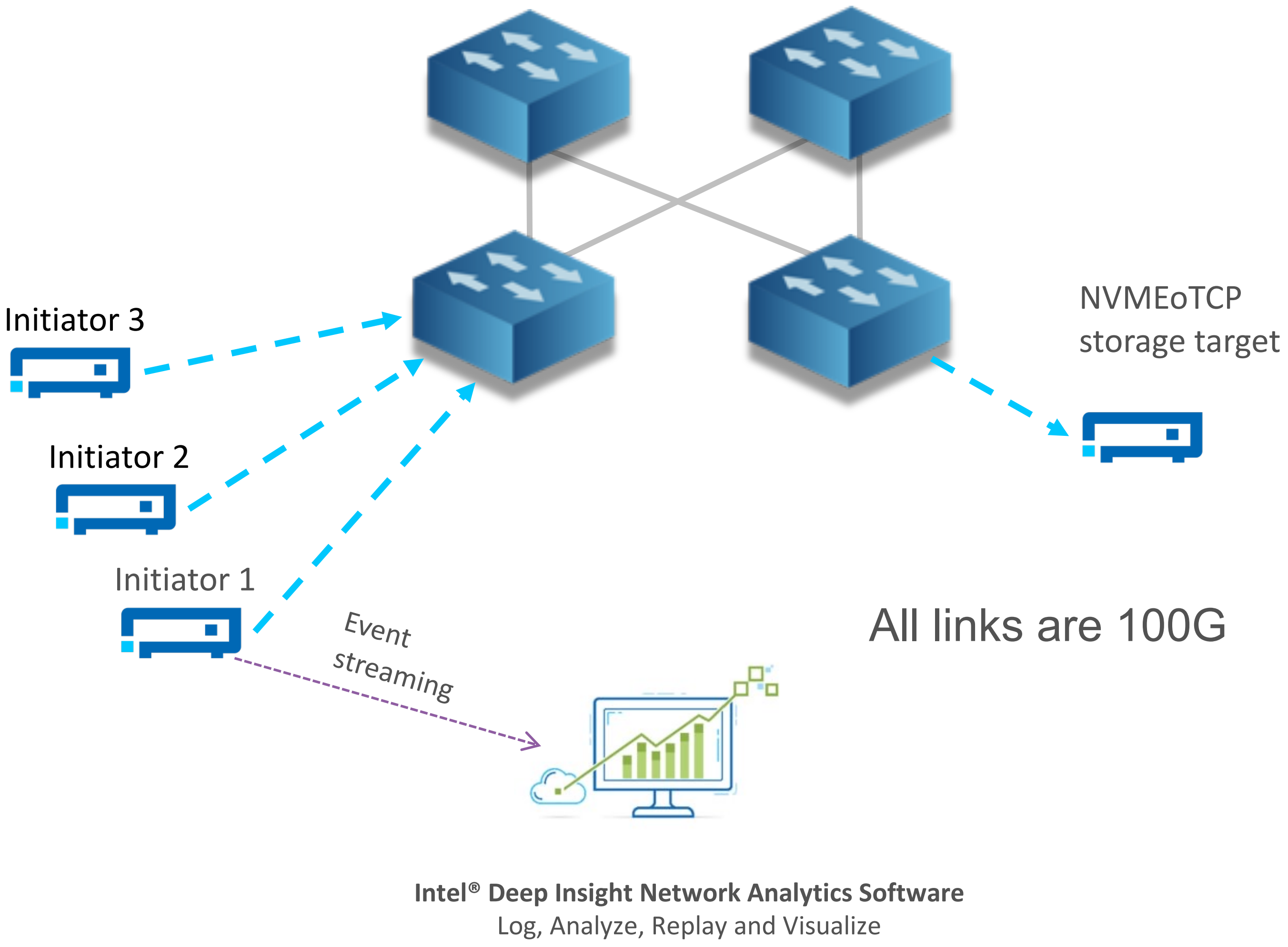
- eBPF callbacks for TCP options (kernel 5.10). TCP-INT eBPF code is called when:
  - new TCP connection is established -> enabling option callbacks for the lifetime of a flow
  - any unknown TCP option is received -> look for and handle TCP-INT
  - TCP adds options to a new outgoing segment -> initialize TCP-INT (INTval=0)
- SK buff local storage (kernel 5.2) used to keep TCP-INT state (echo TCP-INT b2s)
- Perf events and histograms for live data monitoring
  - Exportable via other user-space applications (E.g. gRPC client)
- Configuration maps for enable, disable, disable echo
- User-space application loads, controls, and polls data from eBPF program

Switch-Side TCP-INT

- Control planes determines the mapping between (qdepth, txRate) and INTval
- This increases flexibility and allows complex mapping functions that include division



Demo: End-to-End Performance Bottleneck Identification



FIO NVMe Storage Benchmark

- As number of initiators increases, storage application latency increases
- Identifying the root-cause is a big problem in large distributed systems

Demo: End-to-End Performance Bottleneck Identification

Enhancing CC Algorithms with TCP-INT

1. Initialize TCP-INT option

```
INTval = 0
Lat = 0
```

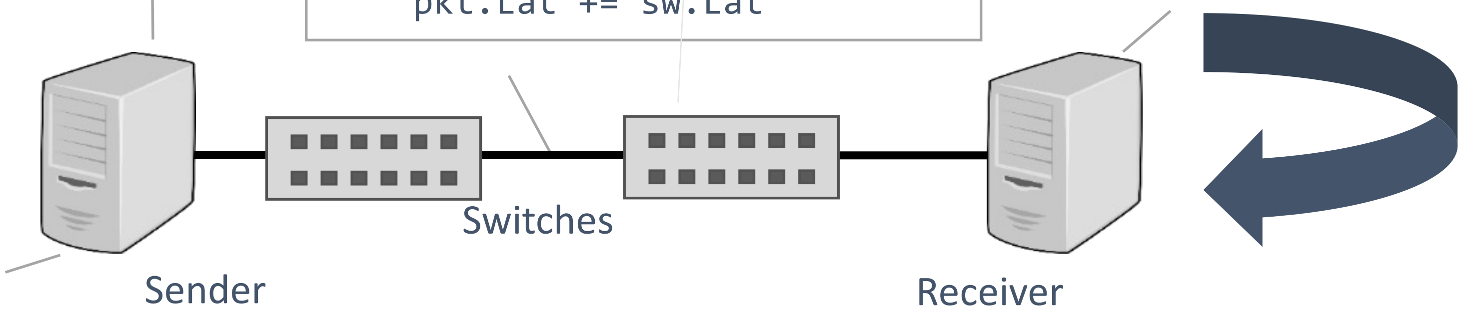
2. Calculate and update INTval + Lat

```
if sw.INTval > pkt.INTval:
    pkt.INTval = sw.INTval
    pkt.INTid = sw.INTid
pkt.Lat += sw.Lat
```

3. Echo received INTval & Lat back to the sender

4. **TCP-INTsink:**

- **congestion control**
  - **HPTCP**
  - **DCTCP++**
  - **AI-CC**
  - …
- **state monitoring and debugging**
- …



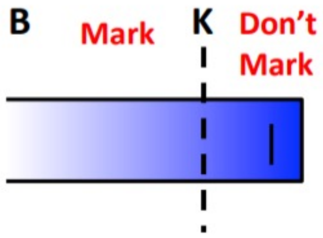
Goal: Simple implementations to allow customers add their secret sauce

ECN+INT Congestion Control

DCTCP: Algorithm

Switch side:

– Mark packets when **Queue Length > K.**



Sender side:

– Maintain running average of ***fraction*** of packets marked **(α)**.

$$\text{each RTT}: F = \frac{\#\ \text{of marked ACKs}}{\text{Total}\ \#\ \text{of ACKs}} \Rightarrow \alpha \leftarrow (1-g)\alpha + gF$$

➢ **Adaptive window decreases:** $W \leftarrow (1 - \frac{\alpha}{2})W$

– Note: decrease factor between 1 and 2.

Source: https://web.stanford.edu/class/ee384m/Handouts/handout16.pdf

ECN+INT Congestion Control

Switch Side

- INTval conveys absolute qdepth

Sender Side

- Introduce α_{int}:

  $\alpha_{int} = qdepth_{smoothed} / qdepth_{target}$

  $qdepth_{smoothed} \leftarrow (1 - g)*qdepth_{smoothed} + g*qdepth$

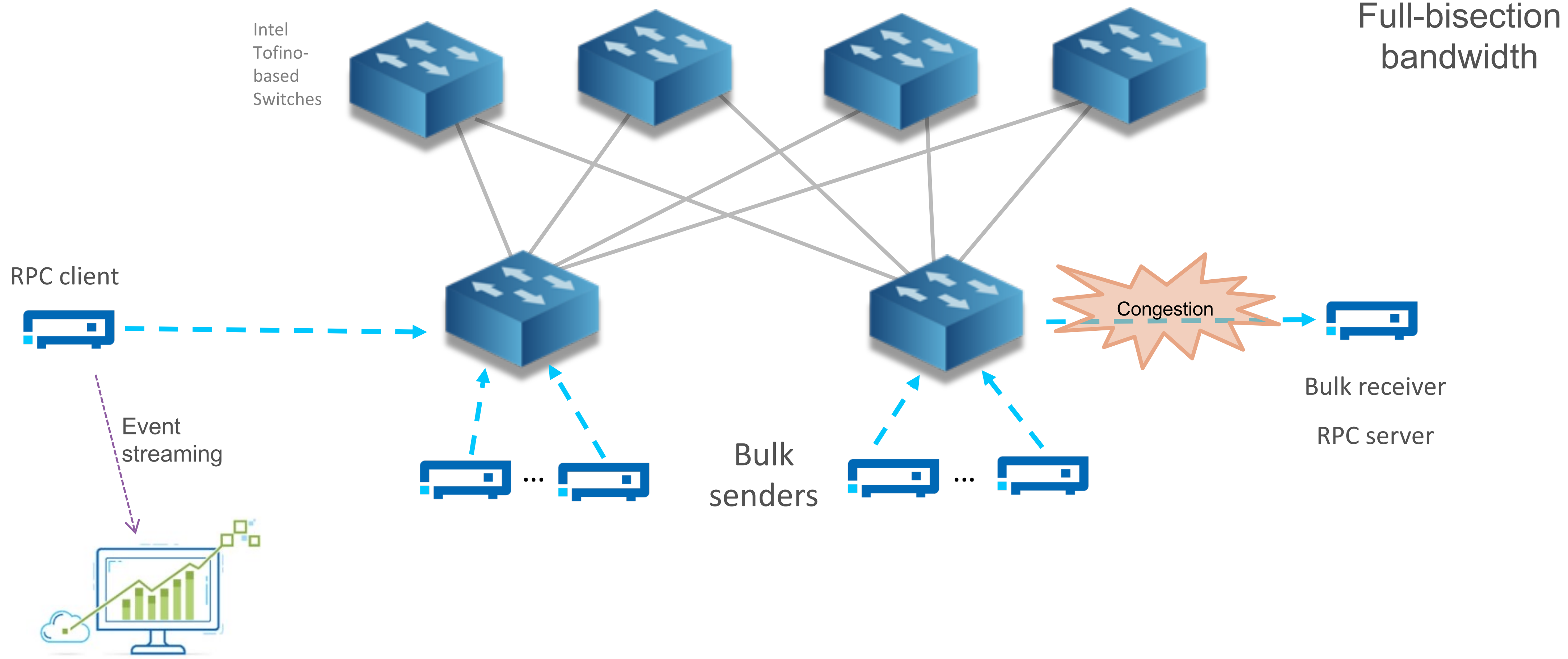
  g – smoothing factor

- Combine α_{DCTCP} and α_{int}:

  $\alpha = \max(\alpha_{DCTCP}, \alpha_{int})$

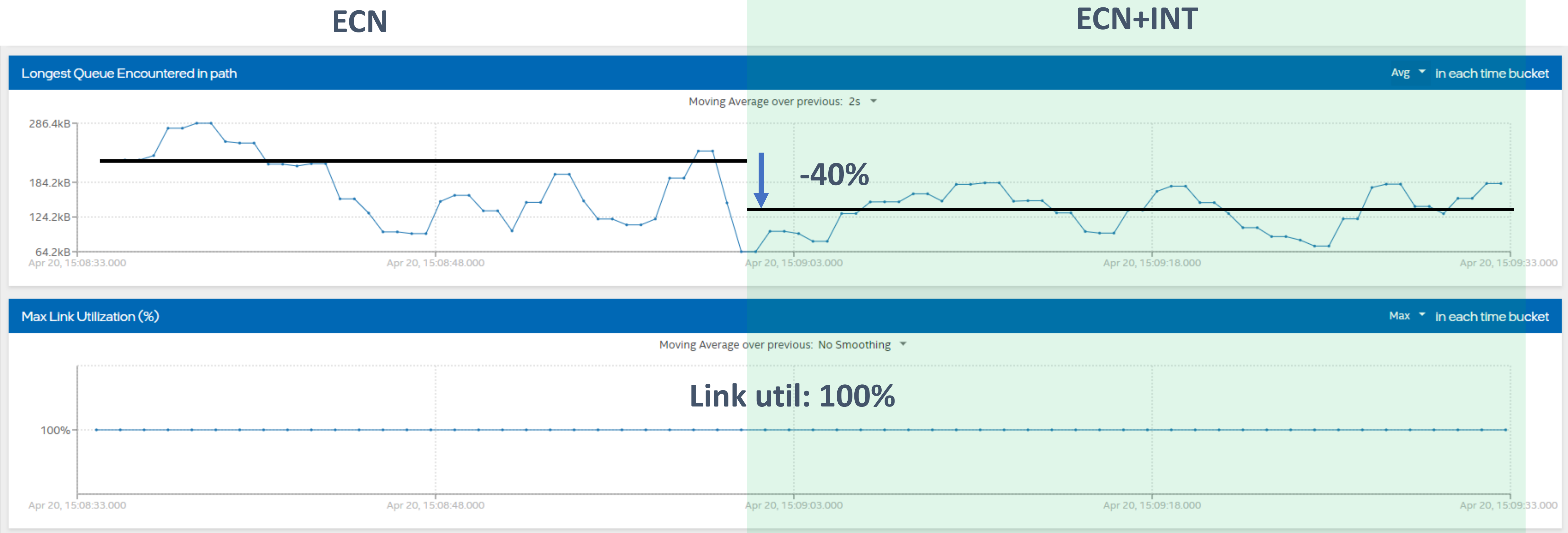
This approach ensures interoperability with switches that do not support TCP-INT

ECN+INT CC Demo Topology



Intel® Deep Insight Network Analytics Software
Log, Analyze, Replay and Visualize

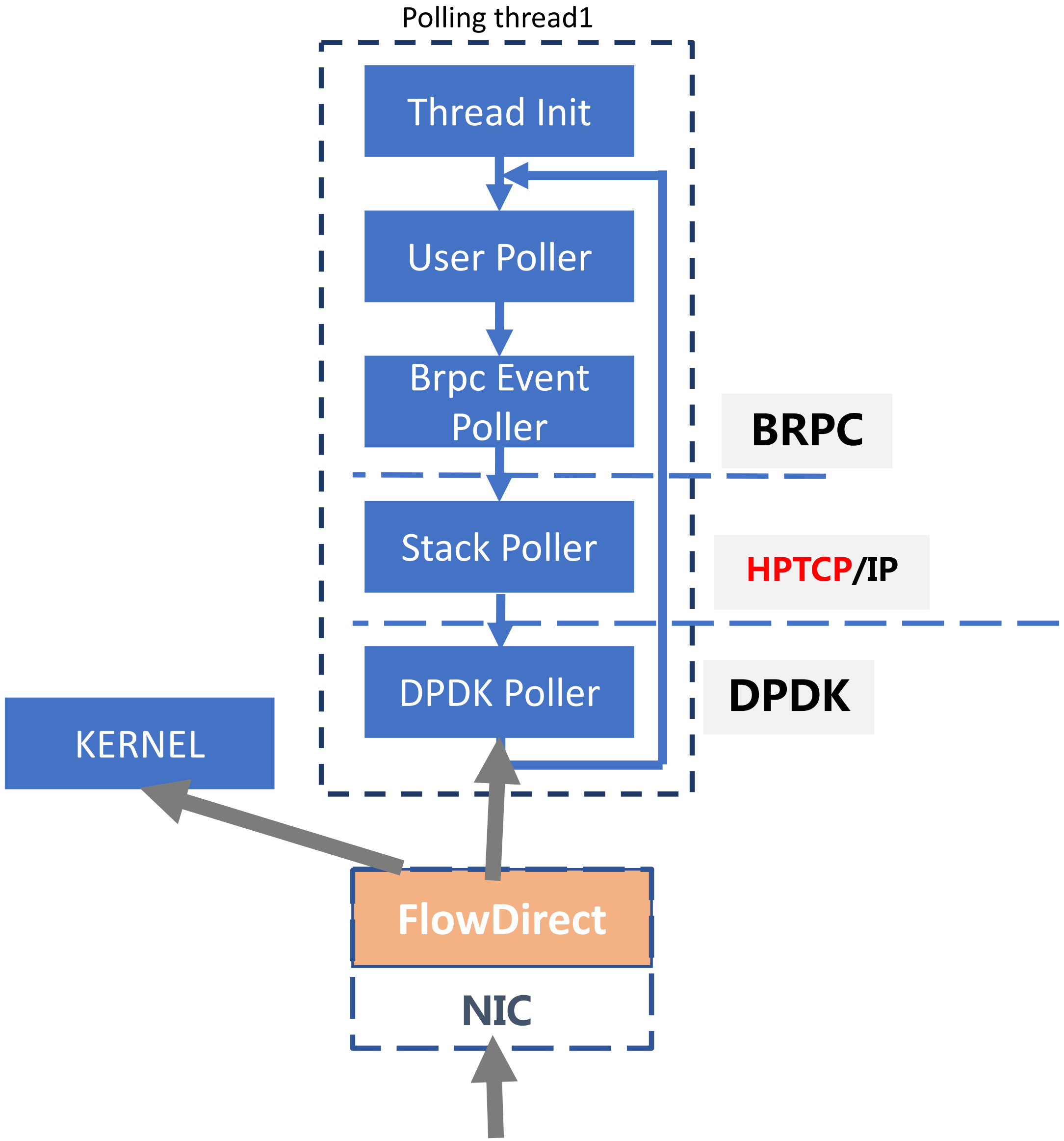
Early Performance Data



ECN		ECN+INT
157 us average latency	-43%	67 us average latency
172 us p99 tail latency	-23%	126 us p99 tail latency
6K reqs/s	2.5x*	15K reqs/s

*For workloads and configurations visit www.intel.com/PerformanceIndex. Results may vary.

Baidu Implementation



- Congestion window update
  - Baidu implements HPTCP Algorithm
  - HPTCP is based on HPCC algorithm for RDMA networks, ported to TCP
  - Uval is calculated from INTVal field in TCP-INT

  HPCC:https://dl.acm.org/doi/pdf/10.1145/3341302.3342085

- Uval calculaction
  - If INTval[7]=1, INTval[6:0]=qdepth, then:

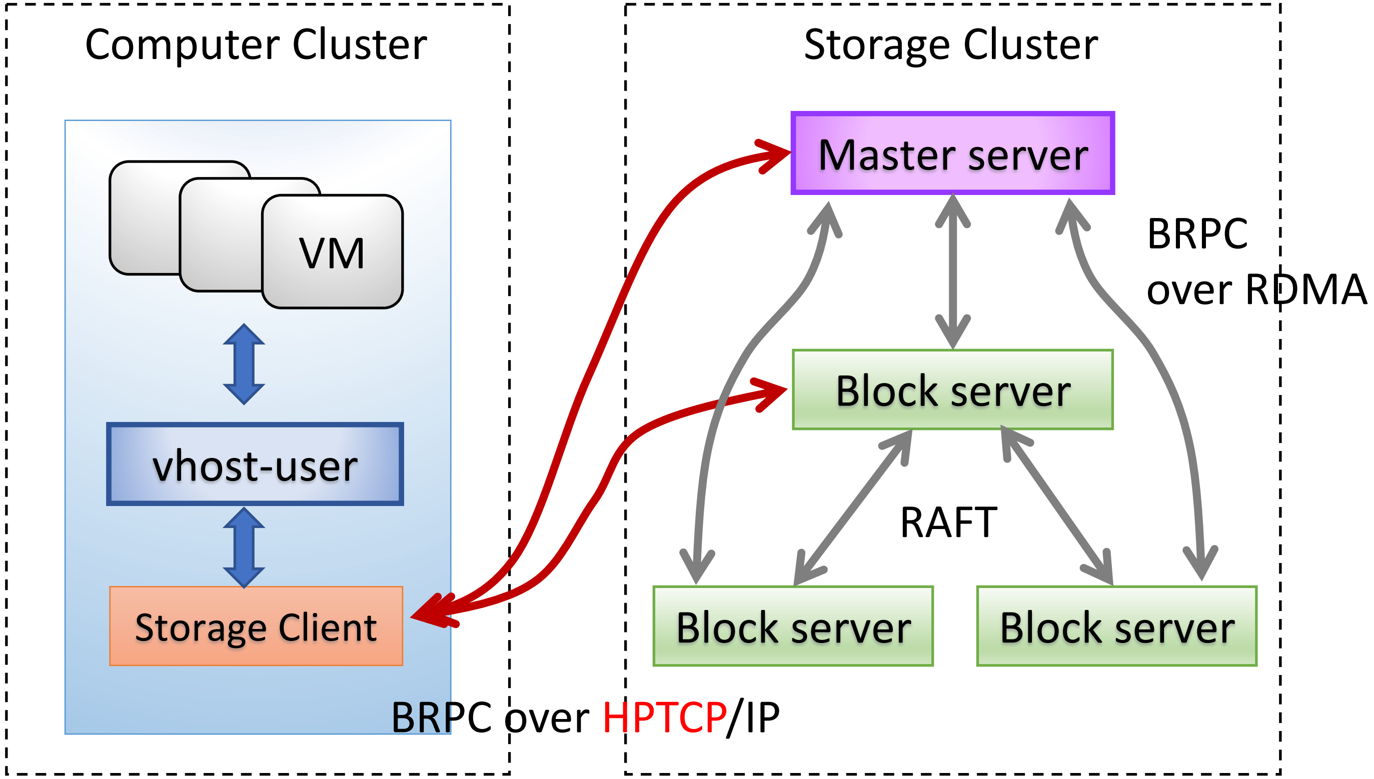
  $$Uval = \frac{qdepth}{B * RTT} + \frac{txRate}{B} = \frac{qdepth}{B * RTT} + 1$$

  - If INTval[7]=0, INTval[6:0]= Uval << bitshift

- Fast retransmit: snd_cwnd = snd_cwnd / 2
- RTO: snd_cwnd = min(snd_cwnd/2, Winit)

Baidu Vision & Planned Usage

- Intend to be used in Storage Client
- Comparison of various technical solutions is in progress
  - SWIFT
  - HPTCP
  - …
- Aim to compete with the storage networks from top cloud vendors



Roadmap (Intel)

- Open-Source TCP-INT host-side code
- Delivering network telemetry directly to applications
- Scale out testing with 100s of nodes
- Continue research into congestion control improvement and network mgmt. utilizing TCP-INT telemetry

Baidu Roadmap

- Continue to test HPTCP in Baidu's testbed
- Keep optimizing HPTCP congestion control algorithm
- Compare the performance of HPTCP with SWIFT and RDMA schemes
- Deploy the new transport stack in Storage client to improve the entire storage networks

Notices and Disclaimers

- Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.
- Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.
- Your costs and results may vary.
- Intel technologies may require enabled hardware, software or service activation.
- Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.
-

Thank You