A General Delta-based In-band Network Telemetry Framework with Extremely Low Bandwidth Overhead

SHENG Siyuan

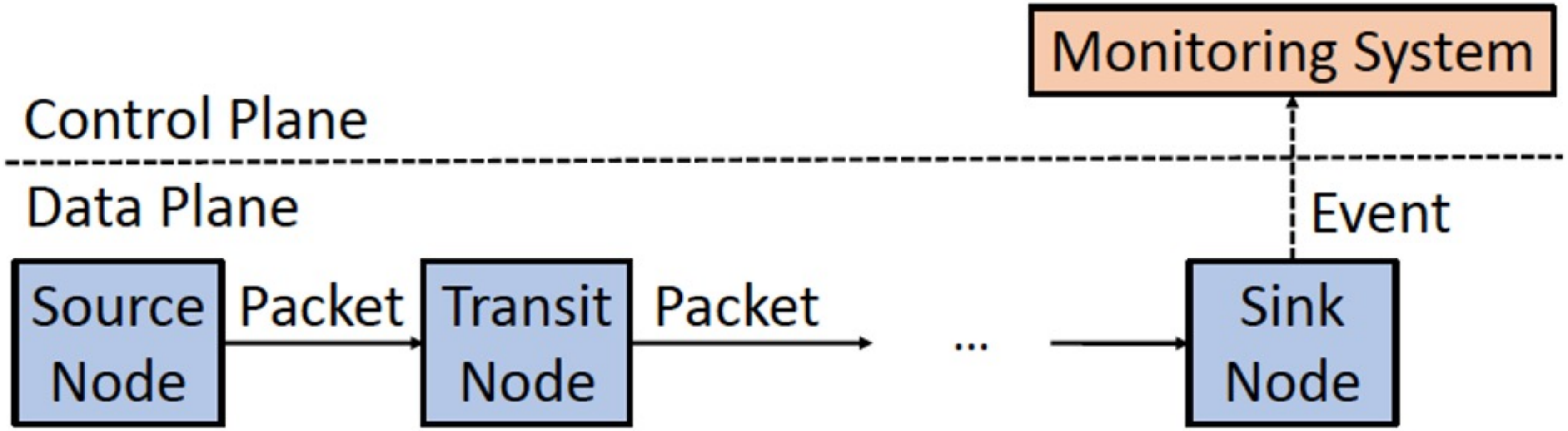
Supervised by Prof. Patrick P. C. Lee

The Chinese University of Hong Kong

April 2021

In-band Network Telemetry (INT)

- Source pushes control information and device-internal states
- Transit pushes states according to control information
- Sink extracts INT information and reports an event



INT framework

Limitations of INT

- Significant bandwidth overhead
  - Linearly grow with the length of forwarding path
  - Reduce effective bandwidth for network applications
  - Increase likelihood of IP-level fragmentation
- Example
  - 5-node fat-tree topology in data center
  - Trace device ID, ingress port, and egress port, of 4B each
  - 12B per-node states and 8B INT control information
  - 68B in total → at least 4.53% of 1,500B MTU in Ethernet

Existing Studies

- Sampling-based methods
  - Embed INT information to only a subset of sampled packets
  - Reduce bandwidth overhead yet with **slow convergence**
  - Cannot retrieve INT information unless collecting sufficient packets
- Other methods
  - Designed for specific telemetry tasks
- All existing methods suffer from **low generality**
  - Cannot support all families of common applications

Our Contributions

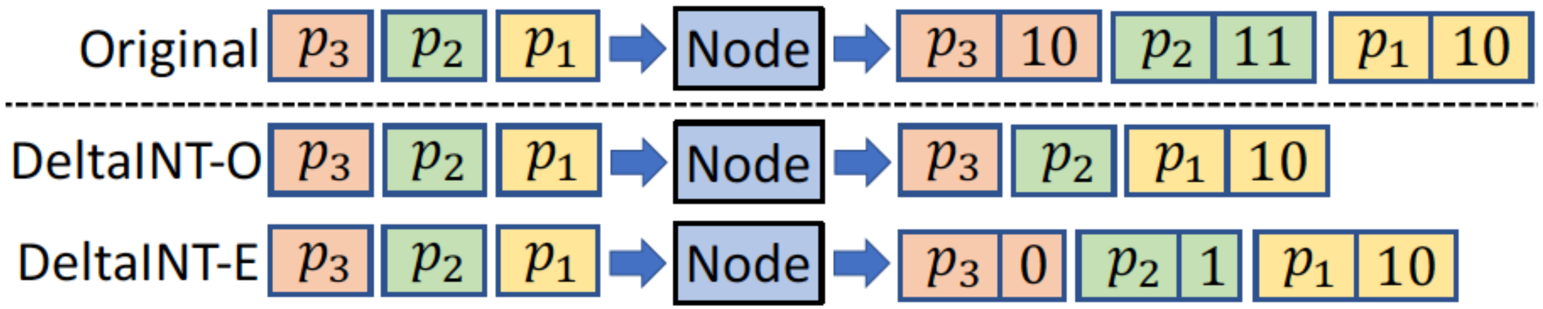
- **DeltaINT**, a general INT framework
  - Two variations: DeltaINT-O and DeltaINT-E
  - Extremely low bandwidth overhead
  - High generality and convergence
- Theoretical analysis on bandwidth mitigation guarantees
- Software simulation for various applications
  - For example, reducing up to 93% bandwidth cost in gray failure detection
- P4-based hardware implementation
- Open-source DeltaINT prototype

Four Families of Applications

- Per-packet-per-node monitoring
  - Collect per-node states for each packet (e.g., fine-grained monitoring and gray failure detection)
- Per-packet aggregation
  - Aggregate per-node states for each packet (e.g., congestion control)
- Static per-flow aggregation
  - Collect static per-node states for each flow (e.g., path tracing)
- Dynamic per-flow aggregation
  - Aggregate per-node states for each flow (e.g., latency measurement)

Our Solution

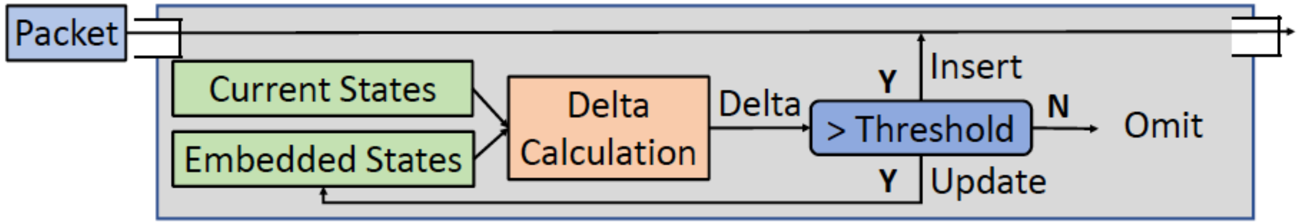
- Key observation
  - **Delta**, the change between *current state* and *embedded state*
  - Delta is often **negligible** at most time in typical applications
    - For example, relatively stable hop latency and static device IDs
- Motivating example



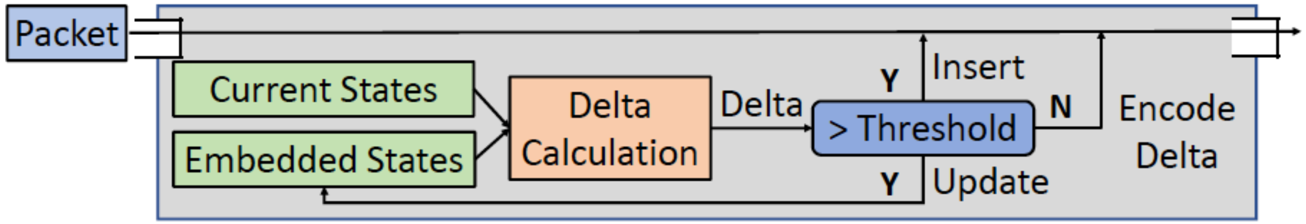
Per-node Architecture in DeltaINT

- Per-node architecture
  - Calculate the delta between current states and embedded states
  - Only if the delta exceeds a threshold, we insert current states into a packet and update the embedded state

DeltaINT-O



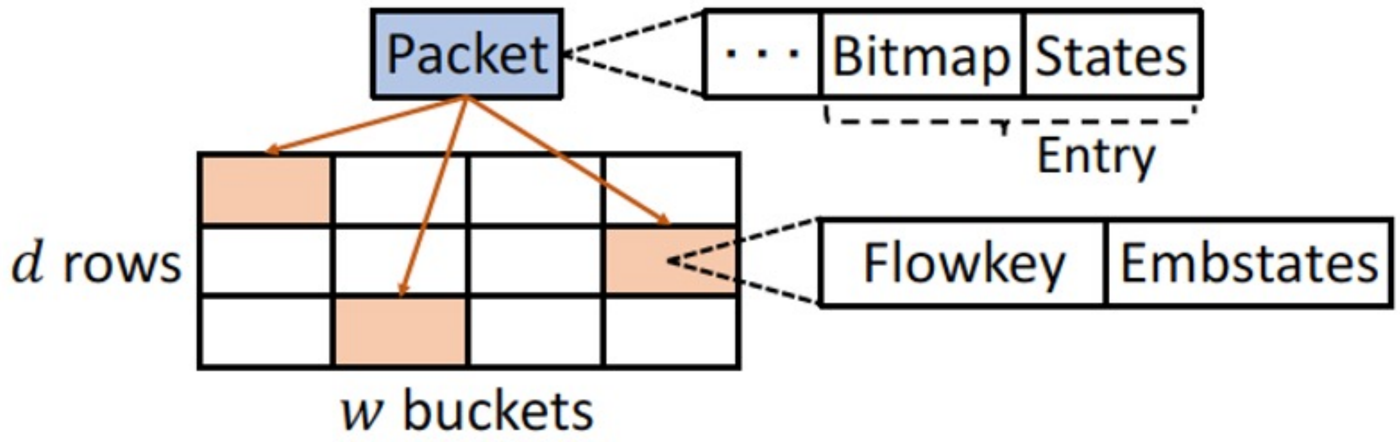
DeltaINT-E



- *How to maintain embedded states efficiently in data plane?*

Sketching in DeltaINT

- Sketch-based technique
  - Store approximate information with limited memory and computations
  - Track embedded states in the data plane with limited resources
- Per-node sketch data structure
  - Each bucket stores a flowkey and the embedded states
  - Each entry of a packet includes a bitmap and the states being embedded



Primitives in DeltaINT

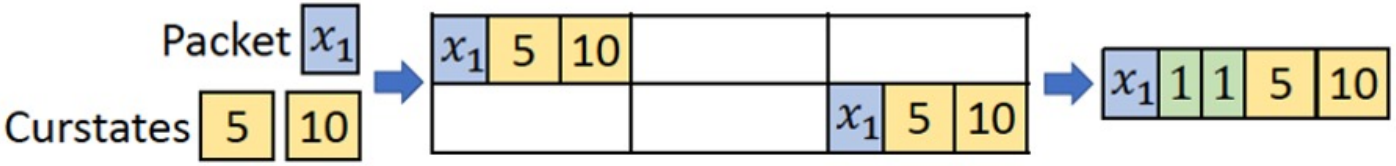
- Four primitives to form DeltaINT workflow
  - StateLoad
    - Hash flowkey and load embedded states from the first bucket matching flowkey
  - DeltaCalc
    - Calculate the delta and compare with the predefined threshold
  - StateUpdate
    - Update flowkey and relevant embedded states in the hashed buckets
  - MetadataInsert
    - Insert a bitmap and the states with non-negligible deltas into the packet
    - Encode negligible deltas into the packet by Huffman coding if DeltaINT-E is used
- Fit DeltaINT into applications with slight changes to primitives

Delta Encoding in DeltaINT-E

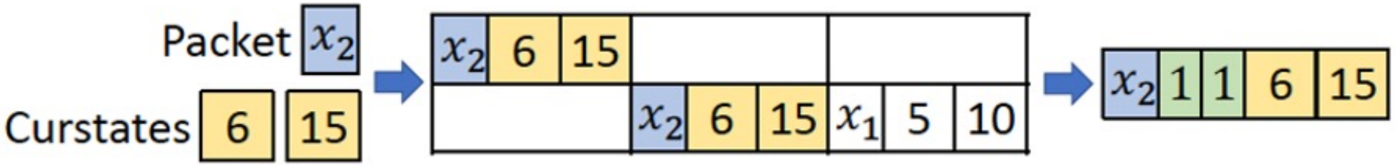
- Assumption on probability distribution of delta values
  - Delta = 0 with the largest probability
  - Each non-zero delta <= ϕ with an equal remaining probability
- Based on Huffman coding
  - A single bit ‘0’ to represent zero delta
  - One bit ‘1’ followed by $\lceil \log_2(2\phi) \rceil$ bits to represent each non-zero delta
  - For example, bit ‘0’, bits ‘10’, and bits ‘11’ for deltas of 0, -1, and 1 if ϕ = 1
  - Note that if ϕ = 0, DeltaINT-E omits negligible deltas as in DeltaINT-O

Update Example of DeltaINT-O

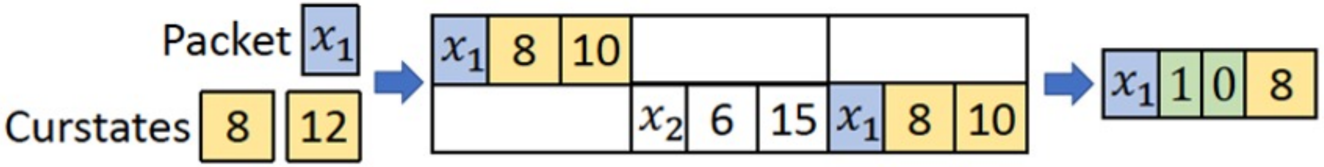
- Receive the first packet of x_1



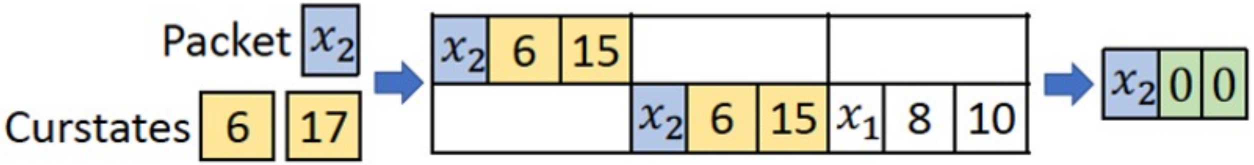
- Receive the first packet of x_2



- Receive the second packet of x_1

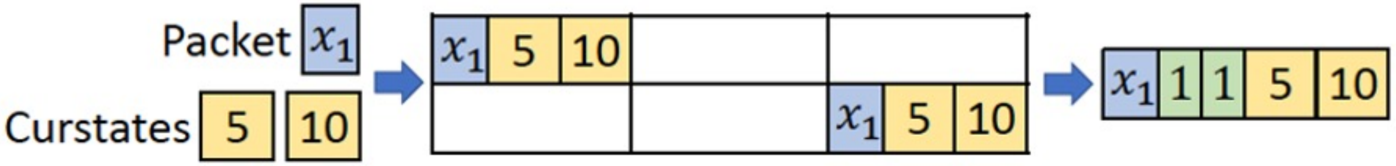


- Receive the second packet of x_2

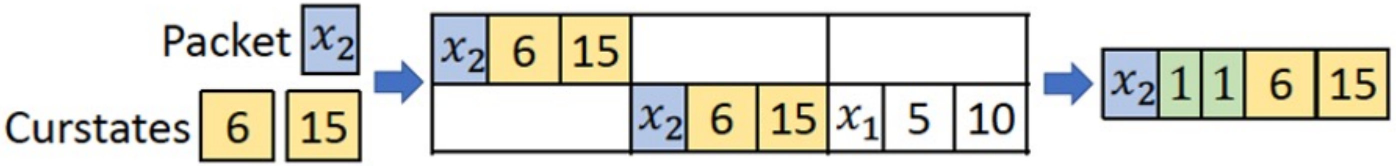


Update Example of DeltaINT-E

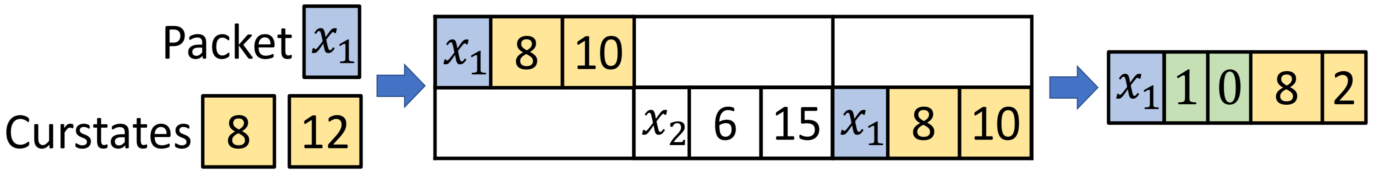
- Receive the first packet of x_1



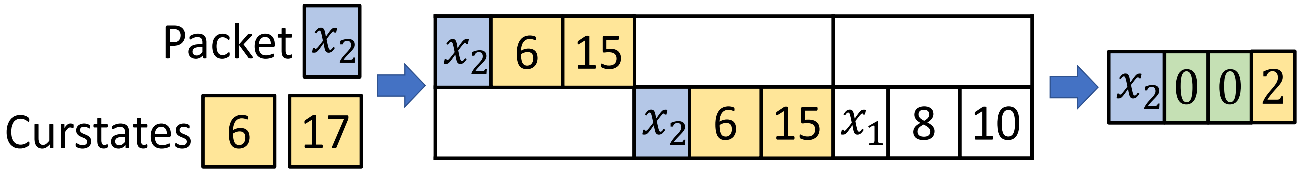
- Receive the first packet of x_2



- Receive the second packet of x_1



- Receive the second packet of x_2

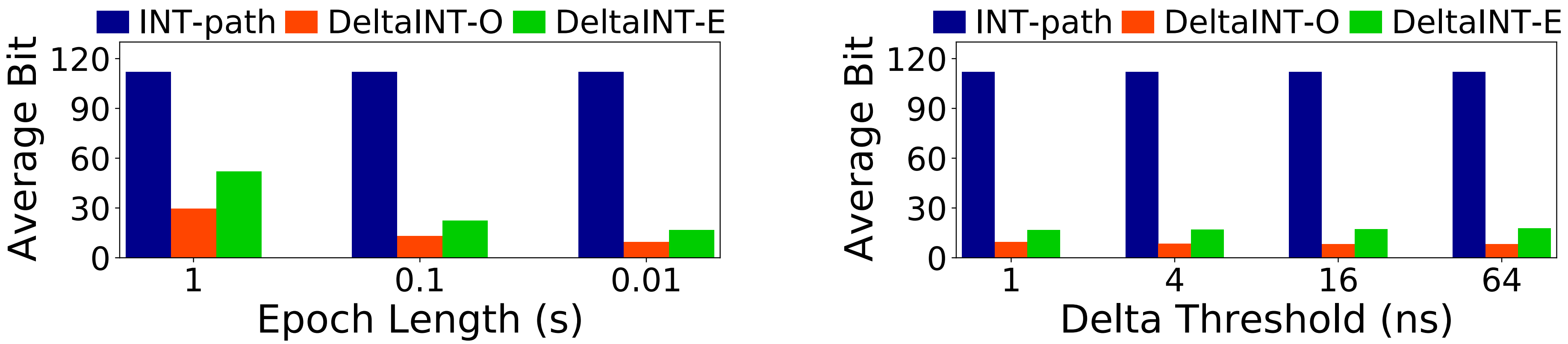


Evaluation

- Methodology
  - For software simulation, we use both bmv2 and NS3
  - For hardware implementation, we compile P4 in Barefoot Tofino switch
  - For sketch in the data plane, we keep 1MB memory and 1 hash function
- Experiments
  - Gray failure detection
  - Congestion control
  - Path tracing
  - Latency measurement
  - Fine-grained monitoring
  - Hardware resource usage

Gray Failure Detection

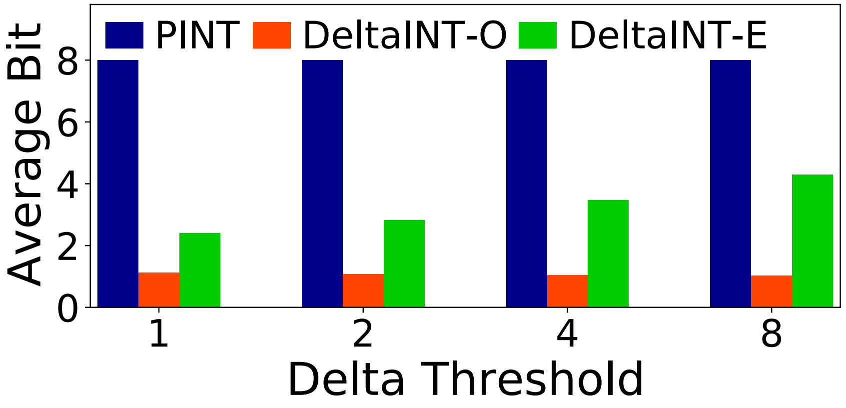
- Tracked states
  - 8-bit device ID, 8-bit ingress port, 8-bit egress port, and 32-bit latency
- Bandwidth usage
  - DeltaINT-O (8.1 bits) mitigates **93%** bandwidth usage of INT-Path (112 bits)
    - DeltaINT-E (16.8 bits) also significantly reduces INT bandwidth overhead
  - Reason: DeltaINT only embeds critical states with non-negligible deltas



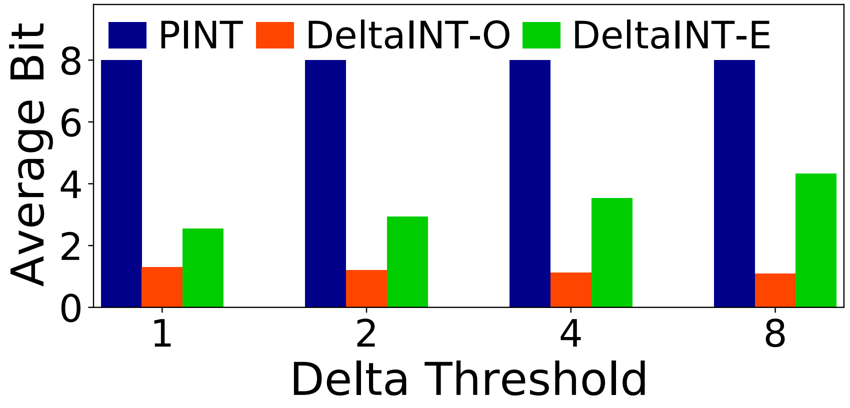
(a) Different epoch lengths (b) Different thresholds

Congestion Control

- Tracked state: 8-bit link utilization
- Bandwidth usage
  - DeltaINT-O (≈1 bit) and DeltaINT-E ($2\sim4$ bits) are better than PINT (8 bits)
  - Reason
    - DeltaINT-O only needs a 1-bit bitmap for negligible delta
    - DeltaINT-E needs delta encoding yet with limited extra bandwidth overhead



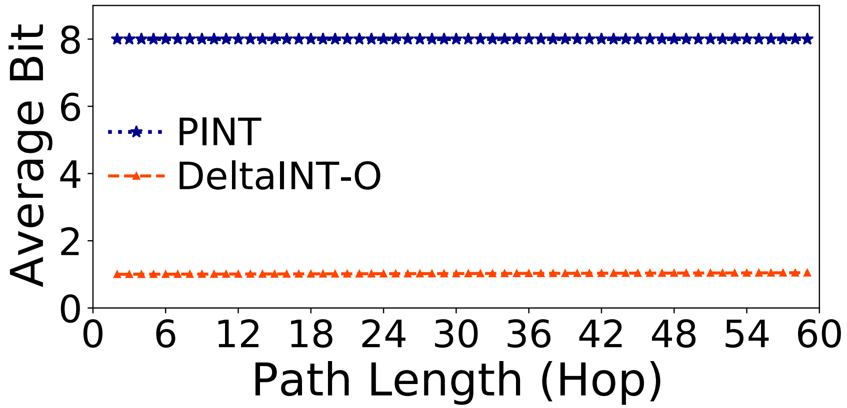
(a) Web search workload



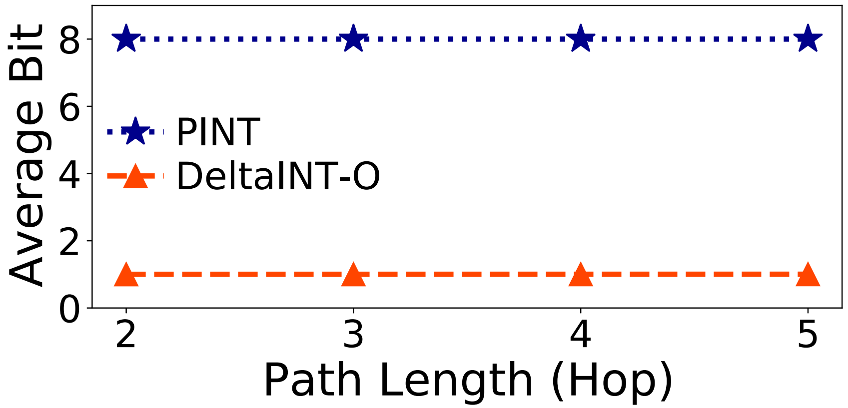
(b) Hadoop workload

Path Tracing

- Tracked state: 8-bit device ID (threshold = 0)
- Bandwidth usage
  - DeltaINT-O (≈1 bit) is better than PINT (8 bits)
  - Reason: DeltaINT-O only needs a 1-bit bitmap for non-first packets of each flow due to static device ID with negligible delta



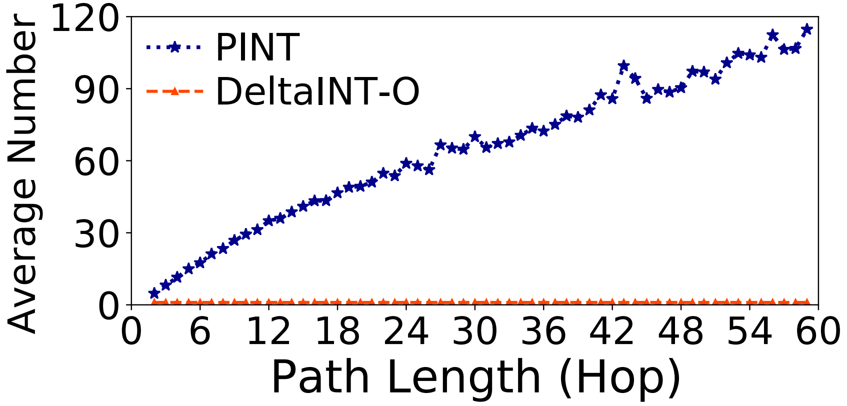
(a) Kentucky Datalink

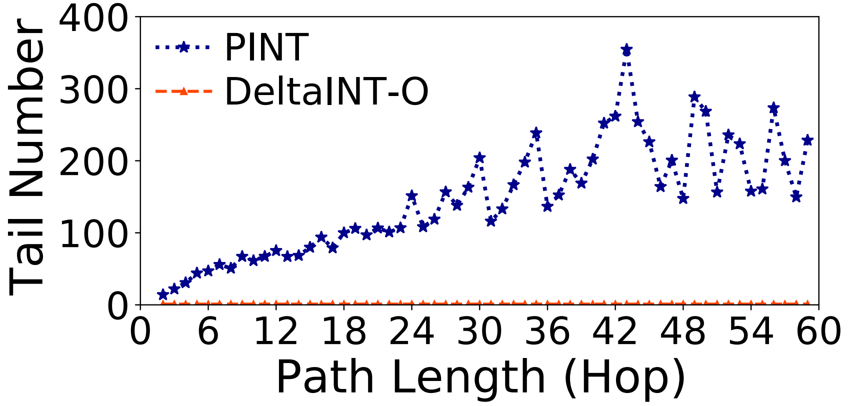


(b) Fat Tree

Path Tracing

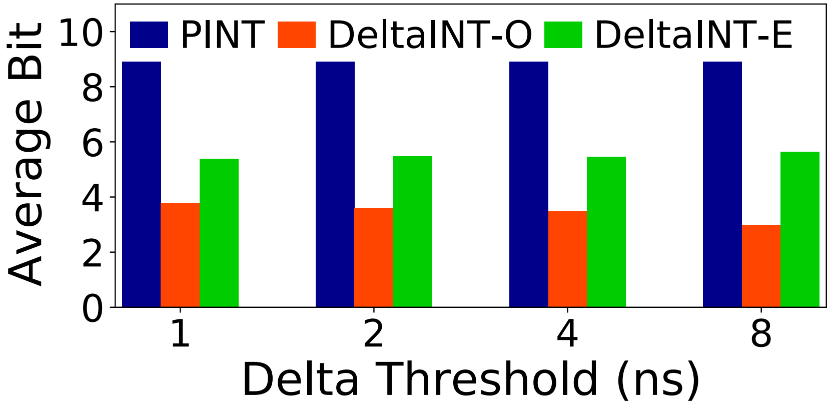
- Convergence
  - Average number: DeltaINT-O (1) vs. PINT (120)
  - Tail (99th percentile) number: DeltaINT-O (1) vs. PINT (350)
  - Reason
    - DeltaINT-O only embeds per-node device ID in the first packet of each flow
    - PINT needs sufficient sampled packets to retrieve per-flow device IDs



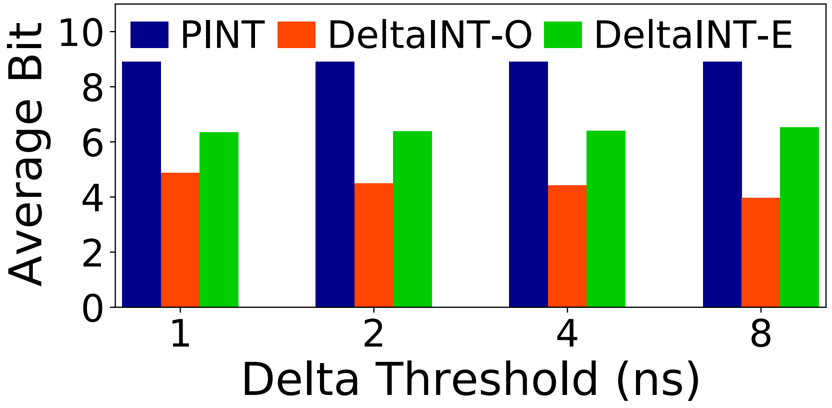


Latency Measurement

- Tracked state: 8-bit latency
- Bandwidth usage
  - (a) DeltaINT-O (3 bits), DeltaINT-E (5.4 bits), and PINT (8.9 bits)
  - (b) DeltaINT-O (4 bits), DeltaINT-E (6.3 bits), and PINT (8.9 bits)
  - Reason: DeltaINT only embeds critical latency with non-negligible delta



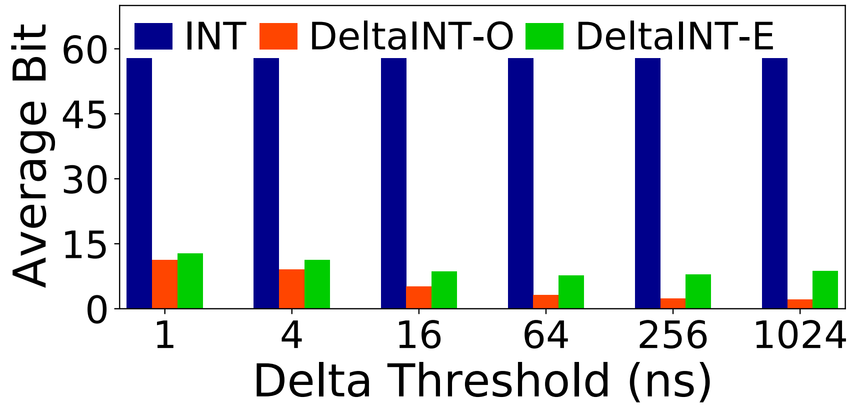
(a) Web search workload



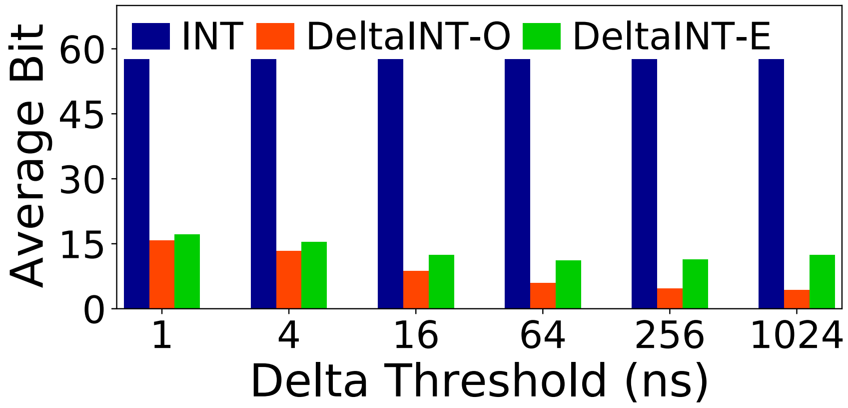
(b) Hadoop workload

Fine-grained monitoring

- Tracked state: 32-bit latency
- Bandwidth usage in web search workload
  - DeltaINT-O decreases from 11.2 bits to 2.1 bits
  - DeltaINT-E decreases from 12.8 bits to 7.7 bits, and increases to 8.7 bits
  - Original INT uses 57.8 bits



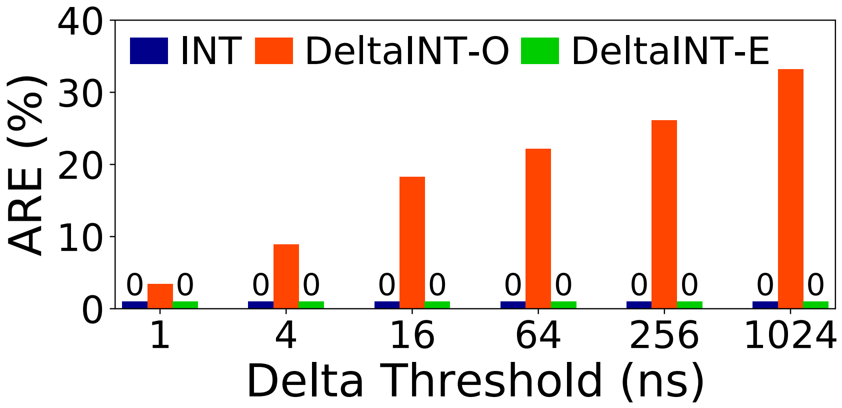
(a) Web search workload



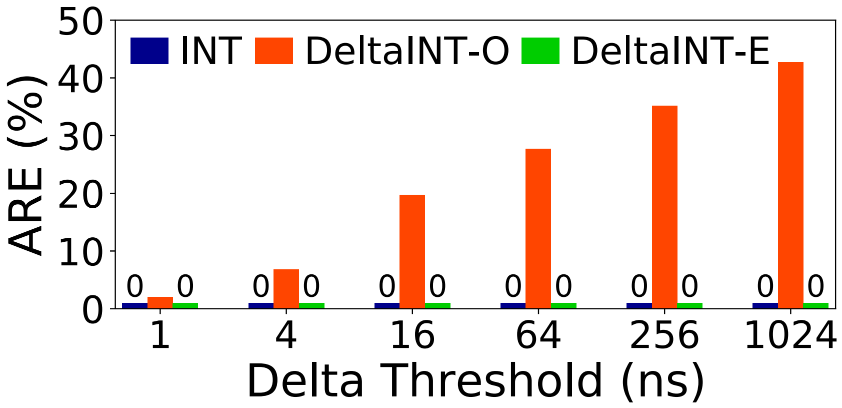
(b) Hadoop workload

Fine-grained monitoring

- Measurement accuracy
  - Average relative error between reported latency and actual latency
  - Results
    - Both original INT and DeltaINT-E can achieve full accuracy
    - DeltaINT-O suffers from large average relative error
  - Reason: DeltaINT-O omits negligible deltas while DeltaINT-E encodes them



(a) Web search workload



(b) Hadoop workload

Hardware Resource Usage

- Hardware resource usage
  - Percentages in brackets are fractions of total resource usage
  - DO and DE incur slightly more SRAM, stages, and stateful ALUs
    - DO and DE need to track embedded states in the data plane
  - INT incurs more PHV sizes and actions
    - INT has larger bandwidth overhead and hence more information to process and transmit

	SRAM (KB)	No. stages	No. actions	No. ALUs	PHV size (bytes)
DO-F1	336 KB (2.19%)	3 (25%)	11 (nil)	4 (8.33%)	110 (14%)
DO-F2	288 KB (1.88%)	5 (42%)	10 (nil)	3 (6.25%)	96 (13%)
DO-F3	304 KB (1.98%)	2 (16%)	5 (nil)	1 (2.08%)	91 (12%)
DO-F4	336 KB (2.19%)	3 (25%)	7 (nil)	2 (4.17%)	101 (13%)
DE-F1	384 KB (2.50%)	4 (33%)	12 (nil)	4 (8.33%)	116 (15%)
DE-F2	320 KB (2.08%)	5 (42%)	11 (nil)	3 (6.25%)	96 (13%)
DE-F3	304 KB (1.98%)	2 (16%)	5 (nil)	1 (2.08%)	91 (12%)
DE-F4	368 KB (2.39%)	3 (25%)	8 (nil)	2 (4.17%)	115 (15%)
INT	176 KB (1.15%)	4 (33%)	42 (nil)	0 (0%)	231 (30%)

Conclusion

- DeltaINT, a novel INT framework to achieve extremely low bandwidth overhead
  - Generality
  - Convergence
- Evaluation on various applications
  - Both variations incur less bandwidth usage than state-of-the-art methods
- Source code:
  - **http://adslab.cse.cuhk.edu.hk/software/deltaint**

Thank You!
Q & A