INTCollector: A High-performance Collector for In-band Network Telemetry

- Master Thesis Defense -

Nguyen Van Tu

Supervisor: Prof. James Won-Ki Hong

Dept. of CSE, DPNM Lab., POSTECH, Korea
tunguyen@postech.ac.kr

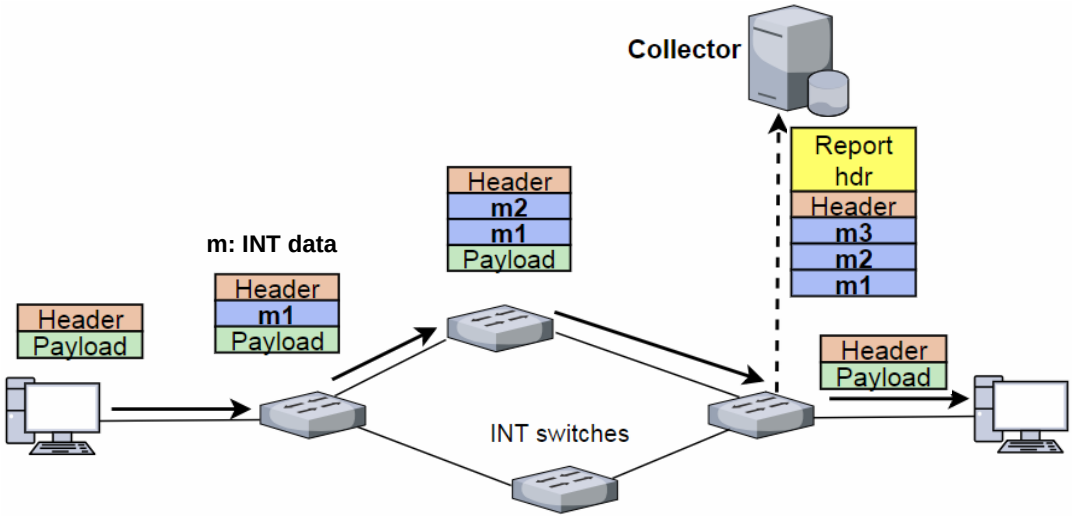
2018-6-15

Outline

- Introduction
- Related work
- Design
- Implementation
- Evaluation
- Conclusion

Introduction

- **Network monitoring is important**
  - Know the network state
  - Help control network (e.g., traffic steering)
- **In-band Network Telemetry (INT)**
  - Real-time, fine-grained, end-to-end monitoring
- **INT problem**
  - INT Report traffic is high
    - 10Gbps link, packet size of 1500-byte: INT report rate of **0.83 Mpps** (1 report for each network packet)



Goal

- **Design and implement a high-performance collector for In-band Network Telemetry: INTCollector**
  - **Design a mechanism to extract important network information (event) from INT raw data**
    - Store all INT reports is costly in storage space and CPU usage
    - Filter the events to reduce the amount of metric values need to store, while still ensure to capture all important network information
  - **Define INT metrics of network information**
  - **In-kernel processing**
    - Process INT reports directly in the kernel space to further improve performance

Related work

❖ IntMon Collector

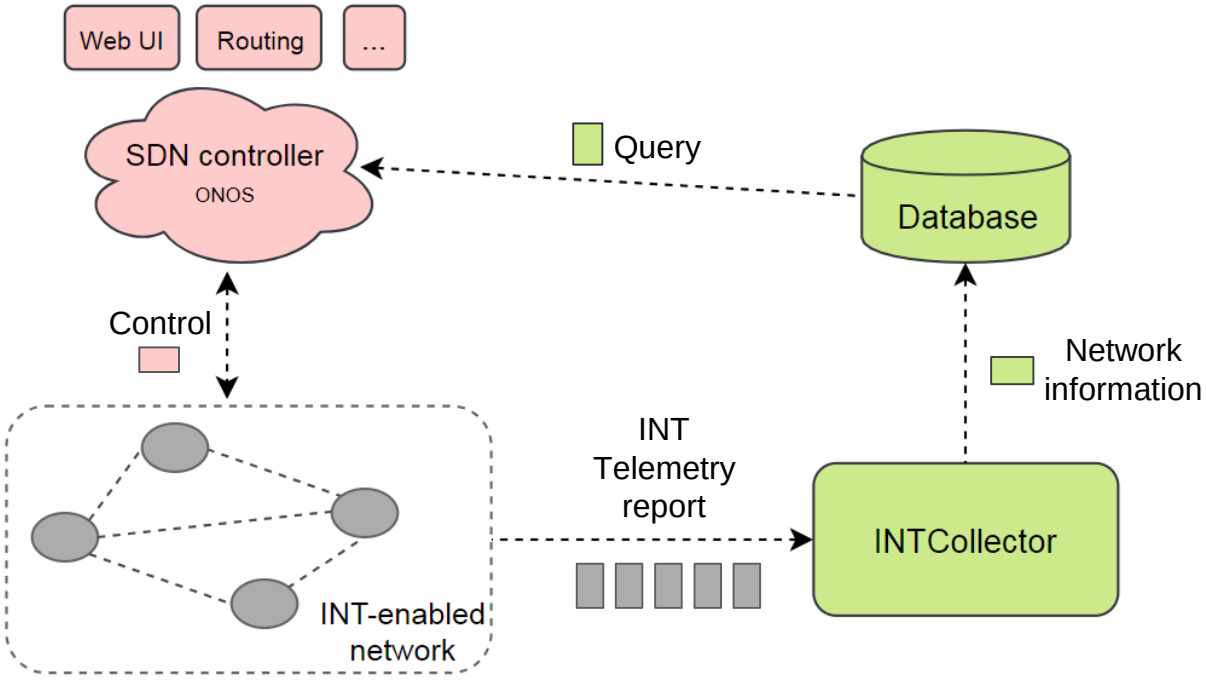
- Our previous work [N.V.Tu et.al., APNOMS 2017]
- IntMon Collector only store immediate data
  - ***Cannot query history data later***
  - ***Run as ONOS application, high overhead***

❖ Prometheus INT Exporter

- From ONOS P4 Brigade [Serkant et.al]
  - https://github.com/serkantul/prometheus_int_exporter
- Send all INT data to an intermediate gateway, Prometheus database periodically collects latest data from the gateway
  - ***Can lose network information***
  - ***High CPU usage, overhead on gateway***

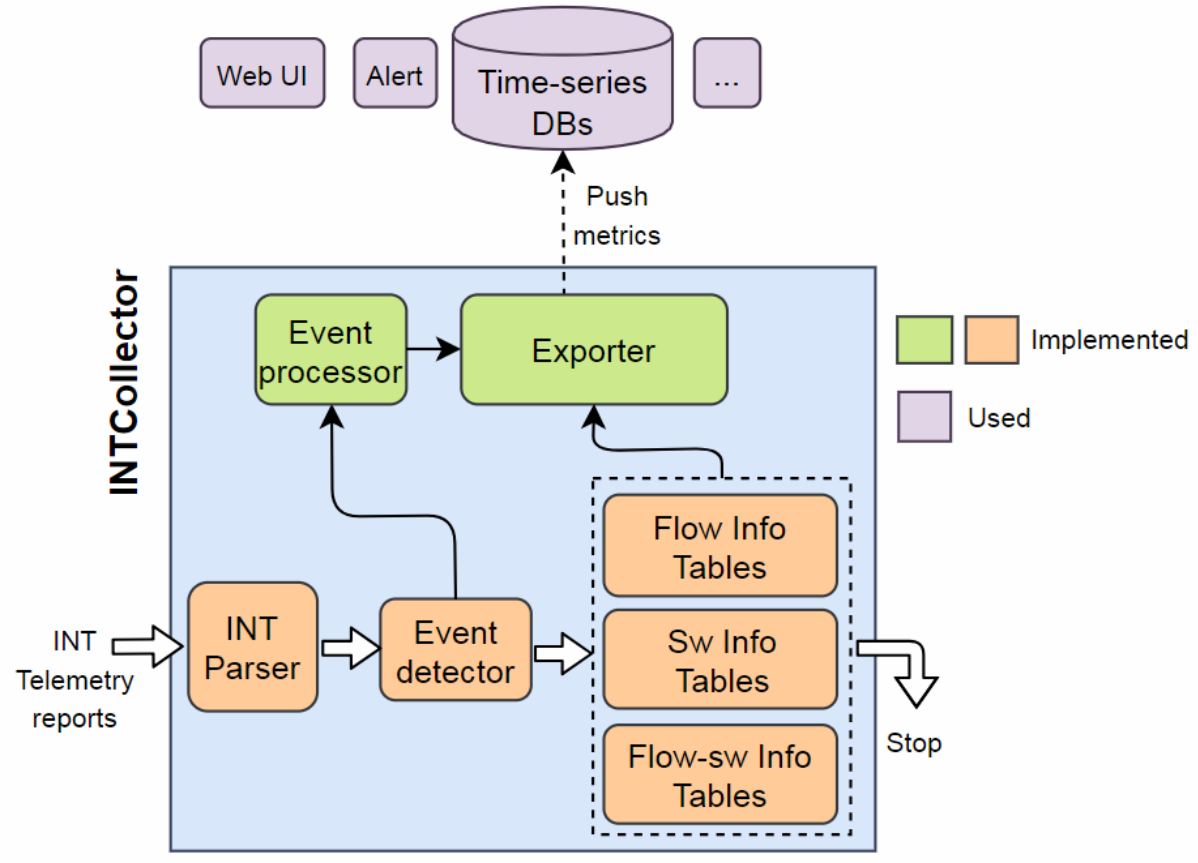
Design - Architecture

- **INTCollector in SDN system**



Design - Architecture

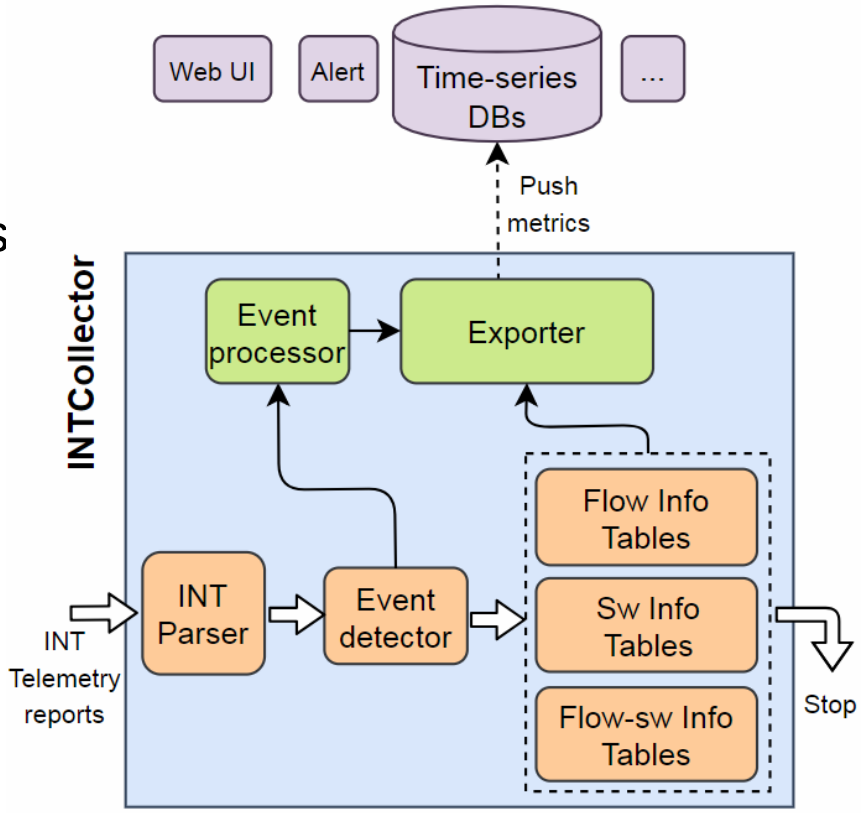
❖ INTCollector architecture



Design - Processing Flow

INTCollector has two processing flows

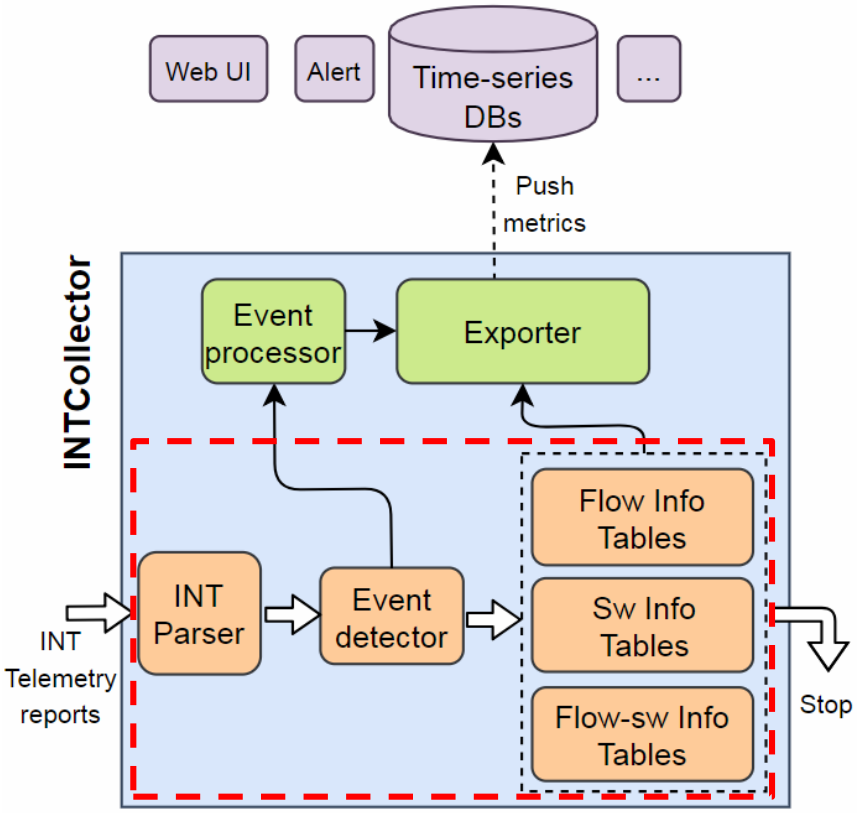
- **Fast path**
  - To process INT telemetry reports
  - Need to run fast
- **Normal path**
  - To process event notification from fast path, and send INT metrics to the database
  - Need to be done efficiently, but not as urgent as fast path



Design - Processing Flow

❖ Fast path

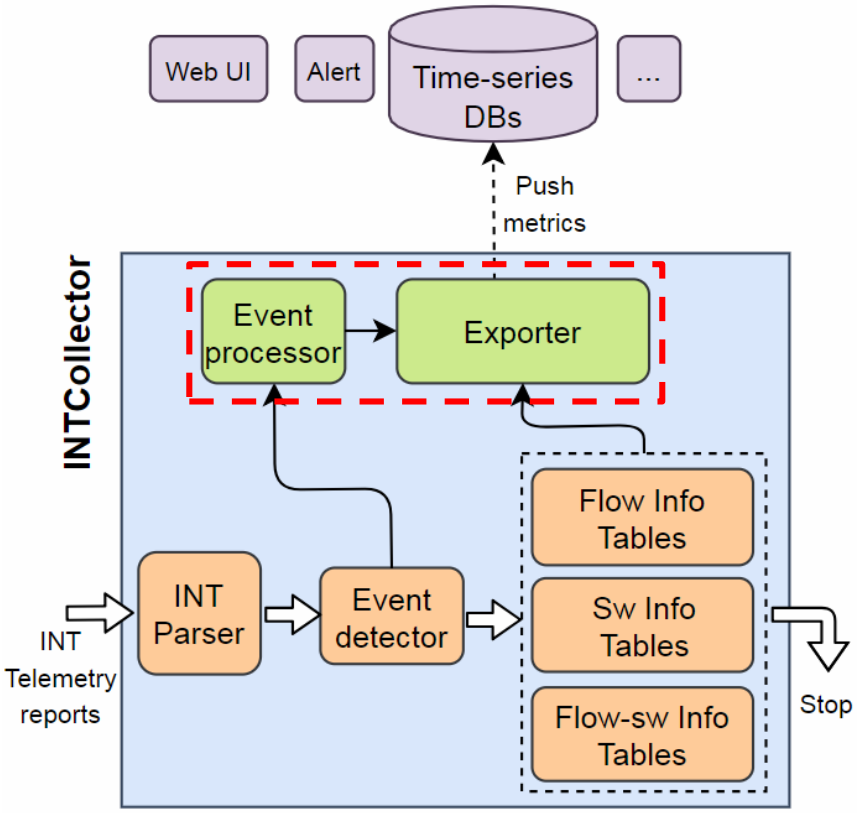
- **INT parser**: Deserialize the packet reports
  - Follow the INT report specification
- **Event detector**: detect network events
  - Send the event to *Event processor* (if any)
  - Event definition and detection mechanism (later)
- **Tables** store latest metric values
  - Metrics definition (later)



Design - Processing Flow

❖ Normal path

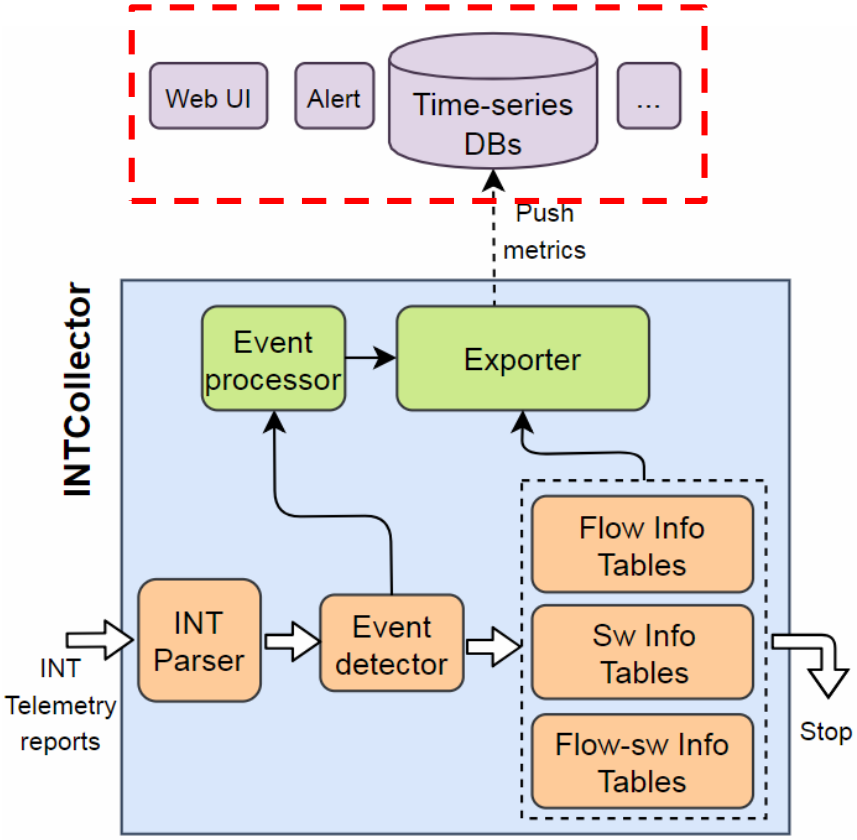
- **Event processor**
  - Define new metrics from event
  - Update metric values and send to the Exporter
- **Exporter**
  - Push metric values to the database
  - Metric values from Event processor or Info tables



Design - Processing Flow

❖ Normal path

- **Database**
  - Store metrics received from exporter for later queries (e.g., queries from SDN controller)
  - Extensions: Web UI, Alert, etc.



Design - Metric

- **Why metrics?**
  - INT raw format is design for doing INT function in the data plane
  - Need to convert to suitable metrics
- **INT metric key: a tuple of (IDs, measurement)**
  - **IDs**: combination of one or several characteristics (of flow or switch) that does not change with time
    - E.g., combination of *sw_id=4*, *queue_id=1*
  - **Measurement:** The measurement we want to know
    - E.g., hop latency, queue congestion
- **A metric value: the value of the measurement of one metric key at one time point**
  - E.g, *hop latency* of (*sw_id=4, queue_id=1*) is ***1ms*** *at time=10s*

Design - Metric

❖ Metric design for INT

- Metrics can be drawn from INT (**time=timestamp**)

Key	Flow info
<5-tuple>	*- Flow path* *- Flow latency*

Key	Flow per-hop Info
<5-tuple + sw_id>	*- Flow per-hop latency*

Key	Egress info
<sw_id, egress_id>	*- link utilization*

Key	Queue info
<sw_id, queue_id>	*- Queue occupancy* *- Queue congestion*

5-tuple: src-dst IPs, src-dst ports, protocol

Design - Event Detection

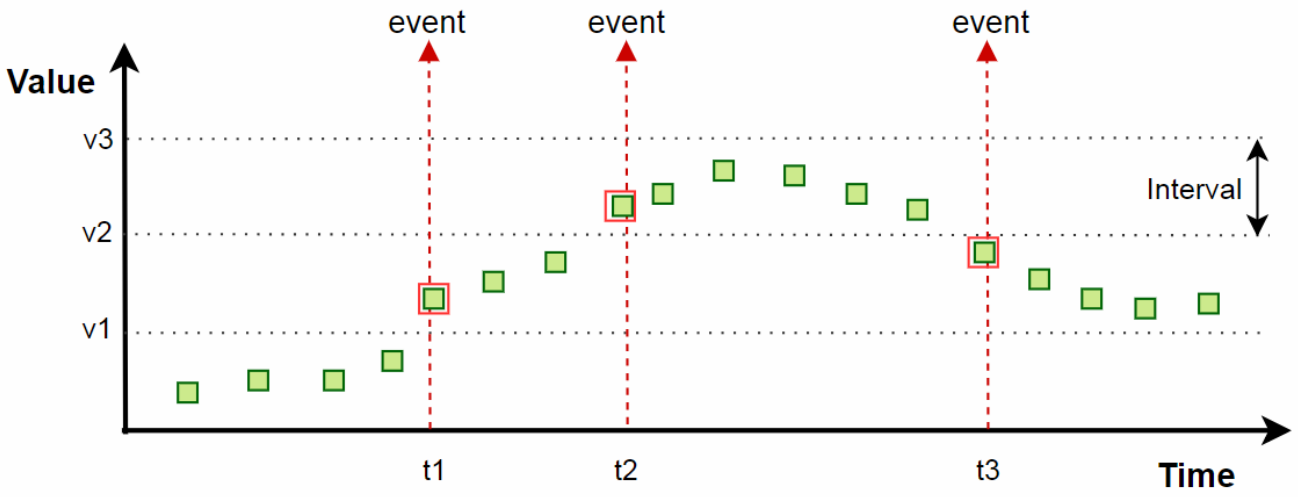
❖ Event definition

- A piece of information that contains new metric key or significant change of some existing metric value
- INTCollector event happens when
  - New metric key (**IDs, measurement)**
    - E.g., new flow, new switch ID
  - **Significant change** of metric value of an **existing (IDs, measurement)**
    - E.g., hop latency changes significantly, which may indicate network congestion

Design - Event Detection

❖ How to define significant change?

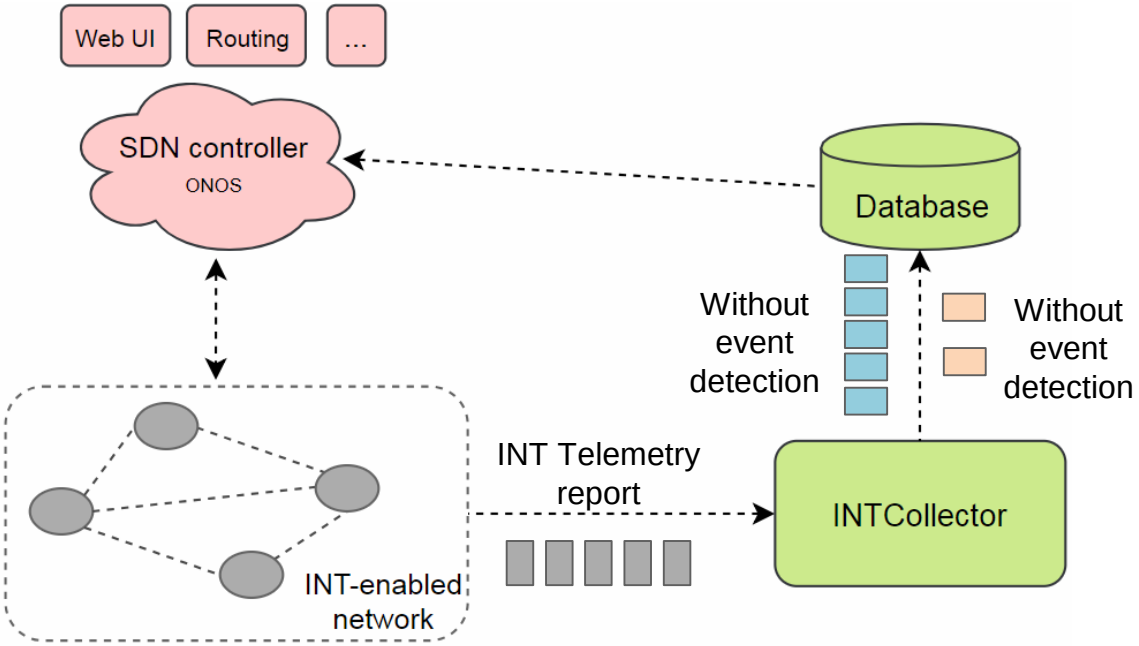
- Value space of the measurement is partition into “interval”
  - E.g.: 0 - 100; 100 - 200; 200 - 300; etc.
- **No event** if current value in the **same** interval with last value
  - E.g., if value changes from 54 to 74
- **New event** if current value in **different** interval with last value
  - **New event** if value changes from 54 to 124



Design - Event Detection

❖ Advantages

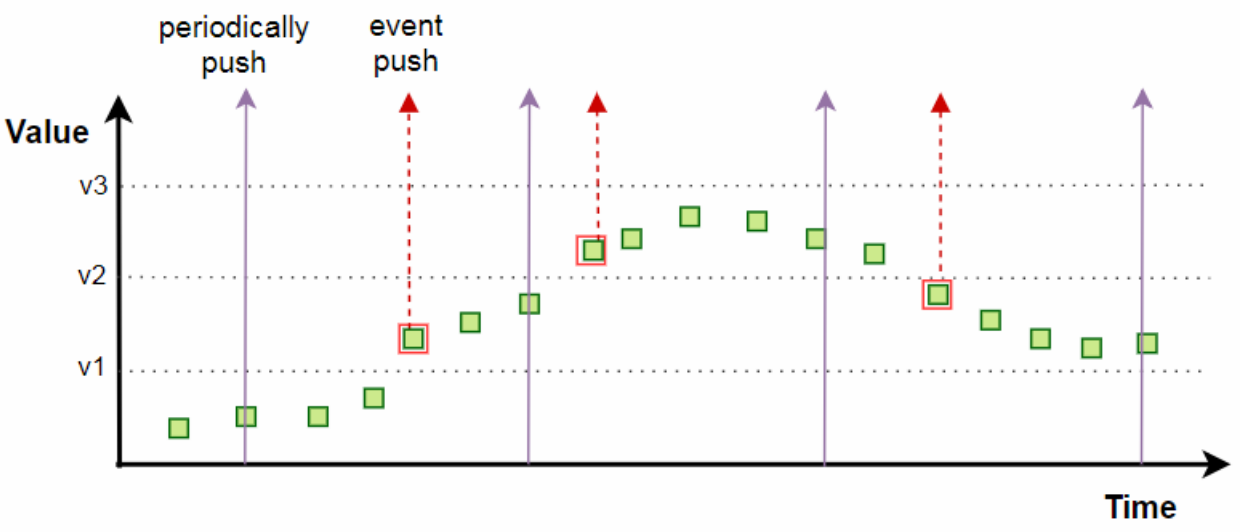
- Reduce the amount of data to push to the server (with accuracy trade-off)
- Reduce CPU usage in both collector and database
- When there is no event, server can know that the value is still in the old interval



Design - Exporter

❖ Exporter push data to the database

- Sends data to server **periodically** or **when event happens**
  - Why periodically? If there is no event, don't know whether the metric is still existed (alive) or not, don't know the latest value
  - Periodically pushing data helps check live status, update latest value



Design - Database

❖ Database requirement

- Store history INT metric data sent from Exporter
  - Good performance
  - Well support for time-series data
  - Support pushing mechanism
- Choose InfluxDB - an open source time-series database
  - Prometheus is chosen by ONOS P4 Brigade

InfluxDB	Prometheus
- Time-series databases - Can handle high-traffic (800k sample/s for Prometheus, 500k sample/s for InfluxDB) - Rich extensions (UI, alert, etc.)	
 support event push and custom timestamp	 limited event push and no custom timestamp
 Complex queries, higher storage	 Flexible, simpler queries, lower storage

Implementation

- **Version specific**
  - Use INT / Telemetry report specification v0.5
- **Normal path**
  - Implemented in Python 2.7
- **Database**
  - Full support for InfluxDB
  - Partial support for Prometheus
    - No event push, may miss network events

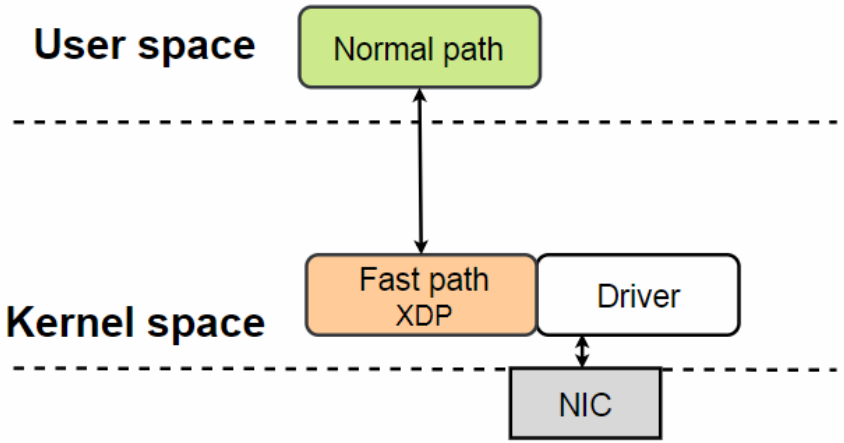
Implementation

- **Fast path**
  - Implemented in C
  - In-kernel processing acceleration with XDP
- **eXpress Data Path (XDP)**
  - New function in Linux kernel for fast packet processing
  - Process packet at lowest level of kernel networking stack
    - Avoid kernel stack processing
    - Avoid kernel-user space switching

Implementation

❖ INTCollector with XDP

- XDP allows the INTCollector fast path run as an XDP program
- Require Linux kernel with XDP support (v4.8 or newer)



Evaluation

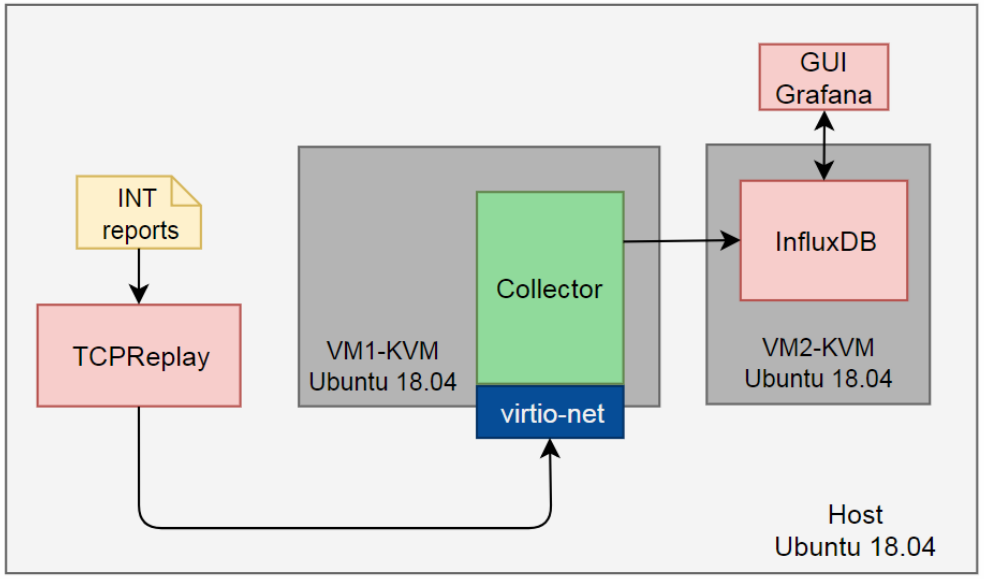
❖ What to evaluate?

- Performance of **INTCollector** vs. other collectors
  - Collector from ONOS P4 Brigade (**Prometheus INT Exporter**)
  - Collector from IntMon (**IntMon Collector**)
- How INT characteristics affect INTCollector
  - Number of hops in flow path
  - Number of metric values
  - Activated INT fields
  - Frequency of network event
- INTCollector with InfluxDB and Prometheus (thesis)

Evaluation

❖ Setup

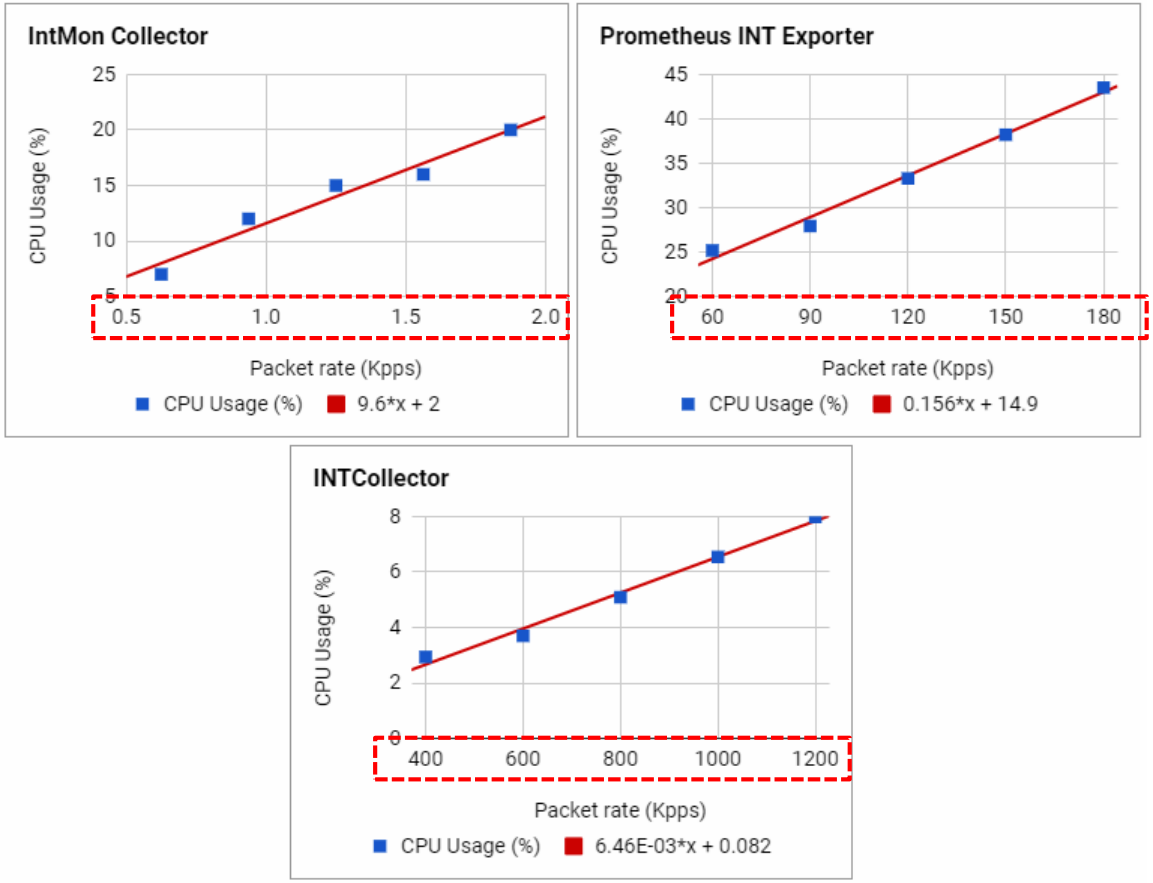
- Host: CPU I5 3570, 12 GB DDR3 RAM
- Each VM: KVM, 1 vCPU, 2GB RAM
- Measure **avg total CPU** usage in VM1 in 3 minutes, use Linux *mpstat* tool
- **INT reports**: pcap file that store INT telemetry report packets



Evaluation

❖ Performance of INTCollector vs. others

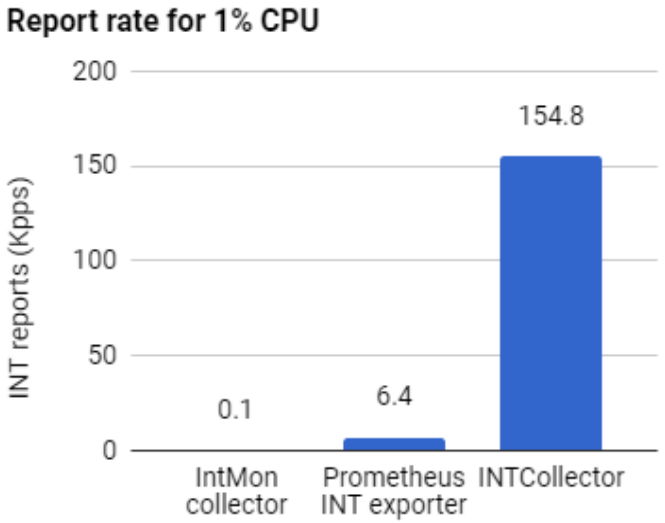
- Same INT reports: 1 flow, 6 hops, all 9 INT fields
- CPU usage increases linearly with packet rate



Evaluation

❖ CPU efficiency: 1% CPU can process how many INT reports?

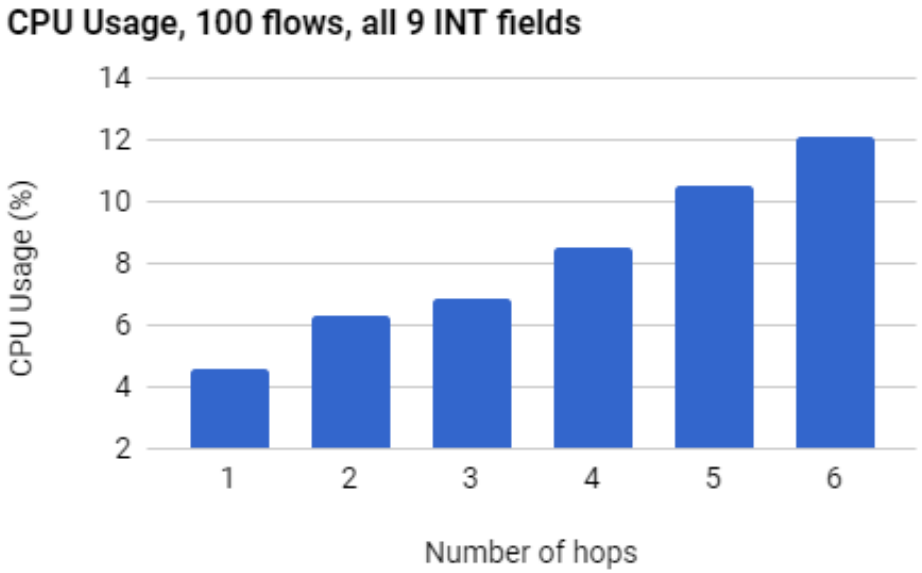
- **IntMon Collector**: 0.1 Kpps
  - Run as an ONOS application
- **Prometheus INT exporter**: 6.4 Kpps (**64x** IntMon Collector)
  - Run separated with ONOS
  - Additional work on a gateway
- **INTCollector**: 154.8 Kpps (**24x** Prometheus INT exporter)
  - Event detection to filter network events
  - XDP acceleration for INTCollector fast path



Evaluation

❖ CPU usage vs. INT characteristics

- CPU usage vs. number of hops
  - Report rate is fixed at 1 Mpps
  - CPU usage **increases gradually** with the number of hops
- **Same result** when we increase the number of metric values, event rate, INT fields: CPU usage **increases gradually**



Evaluation

❖ Insights

- INTCollector is faster than related work
- CPU usage gradually increases when the amount of information in INT reports increases
- Virtio-net and tcpreplay are the bottleneck points

Conclusion

- **In-band Network Telemetry**
  - Real-time, fine-grained, end-to-end monitoring
  - Requires high-performance collector
- **INTCollector: High performance collector for INT**
  - Event detection mechanism
    - Filter network events, reduce CPU usage, reduce storage cost
  - Define INT network metrics
  - Further performance optimization
    - Fast path: in-kernel processing with XDP
- **Future work**
  - Update latest INT/Telemetry report specification
  - Evaluate with hardware NICs.

Thank you!

Appendix

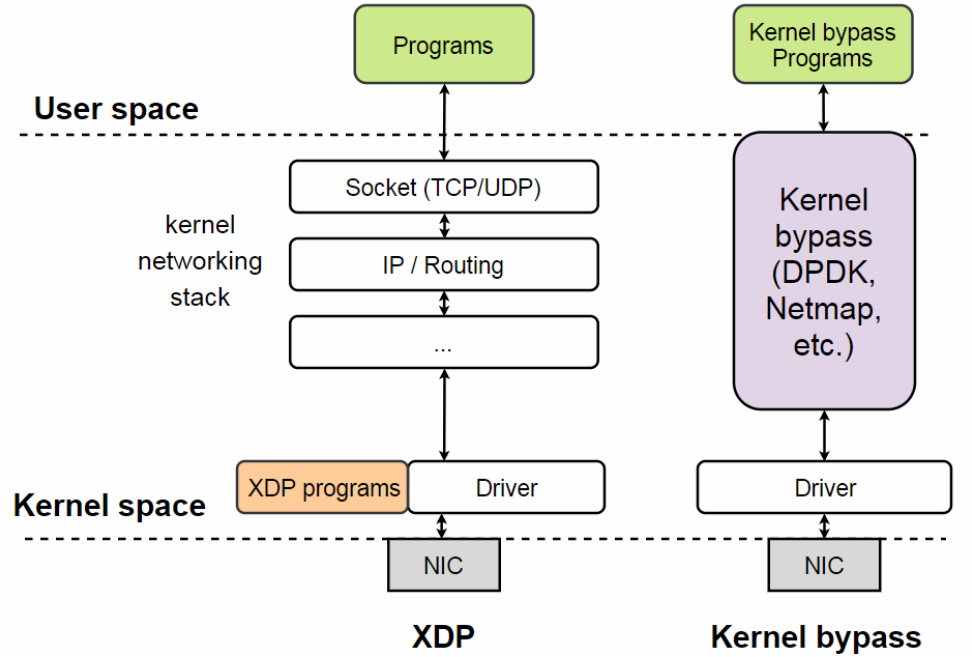
❖ Metric design for INT

- Current INT specification has nine fields:
  - Switch ID, Ingress-egress port IDs, Queue ID
  - Timestamp
  - Hop latency
  - Queue occupancy
  - Queue congestion
  - Link utilization

Appendix

❖ XDP vs. DPDK

- XDP does not require dedicated core for packet polling
- XDP does not require allocated large pages
- XDP works with kernel networking stack (not bypass)



Appendix

❖ INTCollector with InfluxDB and Prometheus

- CPU usage when using InfluxDB and Prometheus: roughly the same