TECHNISCHE HOCHSCHULE DEGGENDORF

Datacenter Network Programming

In-Network Monitoring (Telemetry)

P4: Programmable Monitoring at Line Rate

In-band Network Telemetry (INT)

June 2016

Changhoon Kim, Parag Bhide, Ed Doe: *Barefoot Networks*
Hugh Holbrook: *Arista*
Anoop Ghanwani: *Dell*
Dan Daly: *Intel*
Mukesh Hira, Bruce Davie: *VMware*

Dapper: Data Plane Performance Diagnosis of TCP

Network-Wide Heavy Hitter Detection with Commodity Switches

Rob Harrison, Qizhe Cai, Arpit Gupta, and Jennifer Rexford
Princeton University

Sonata: Query-Driven Streaming Network Telemetr

Language-Directed Hardware Design for Network Performance Monitoring

Srinivas Narayana[1], Anirudh Sivaraman[1], Vikram Nathan[1], Prateesh Goyal[1], Venkat Arun[2], Mohammad Alizadeh[1], Vimalkumar Jeyakumar[3], Changhoon Kim
[1] *MIT CSAIL* [2] *IIT Guwahati* [3] *Cisco Tetration Analytics* [4] *Barefoot Networks*

SketchLearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference

LossRadar: Fast Detection of Lost Packets in Data Center Networks

Elastic Sketch: Adaptive and Fast Network-wide Measurements

Tong Yang Jie Jiang Peng Liu

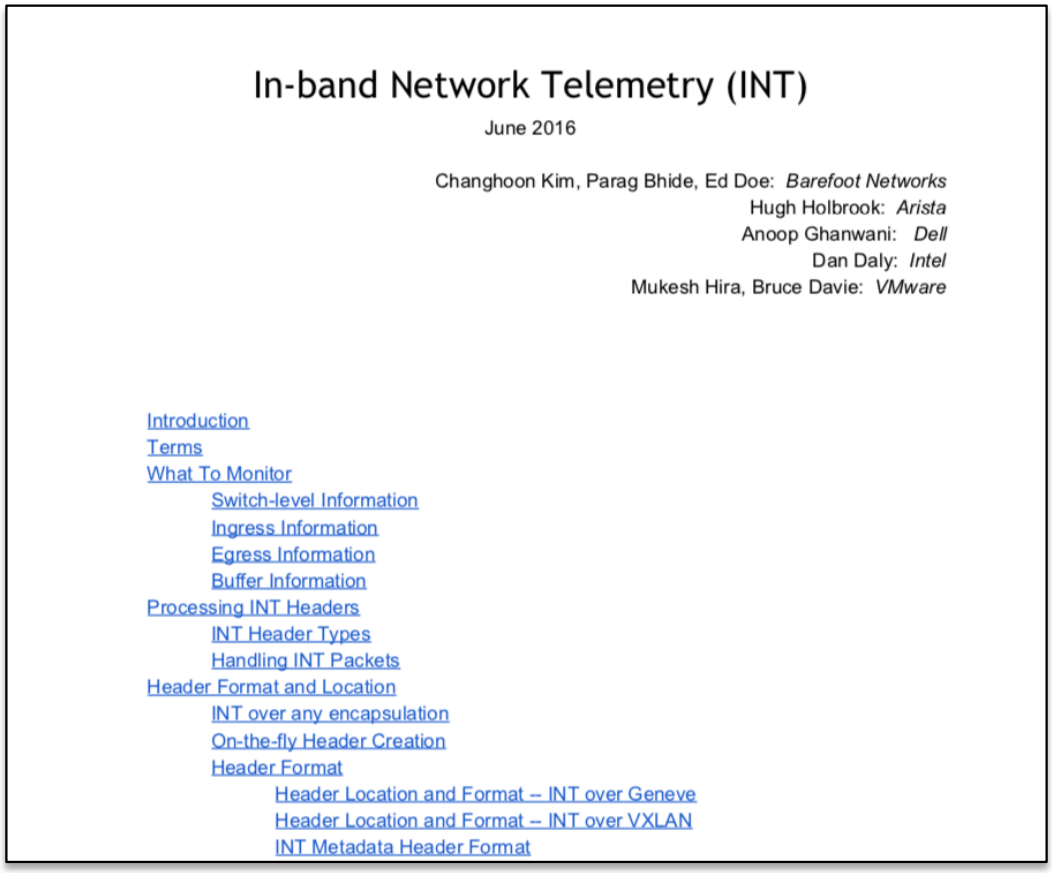
FlowRadar: A Better NetFlow for Data Centers

Yuliang Li* Rui Miao* Changhoon Kim† Minlan Yu*
**University of Southern California* †*Barefoot Networks*

One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon

Zaoxing Liu†, Antonis Manousis⋆, Gregory Vorsanger†, Vyas Sekar⋆, Vladimir Braverman†
† Johns Hopkins University ⋆ Carnegie Mellon University

P4: Programmable Monitoring at Line Rate



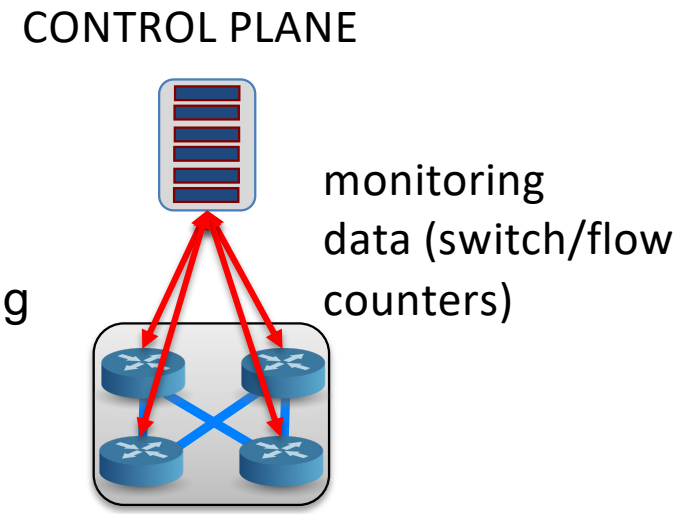
In-band Network Telemetry (INT)

June 2016

Changhoon Kim, Parag Bhide, Ed Doe: *Barefoot Networks*
Hugh Holbrook: *Arista*
Anoop Ghanwani: *Dell*
Dan Daly: *Intel*
Mukesh Hira, Bruce Davie: *VMware*

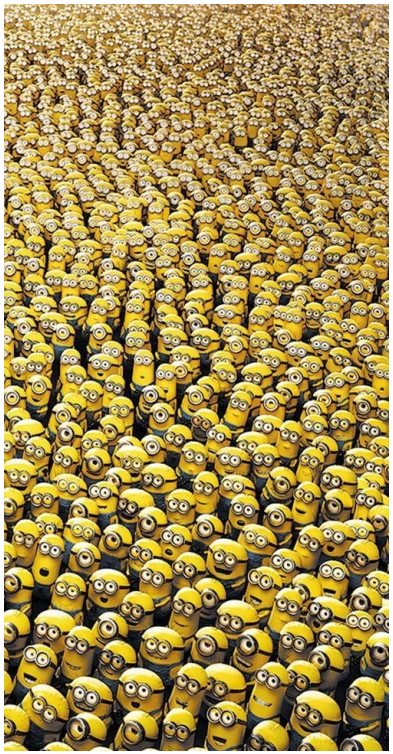
Network Monitoring

- **Current Network Monitoring approaches**
  - (SDN)-switch maintains counters
    - Per port, flow, traffic aggregate (flow rule)
  - Control Plane or Switch CPU polls counters
  - periodically or is informed periodically
  - Not fast enough
    - Network state changes rapidly
    - CPU stress too high for fine-granular monitoring
      - Sampling applied – Sflow/Netflow → loss of granularity
  - Do not provide end-to-end monitoring
    - Need to correlate per switch state → difficult

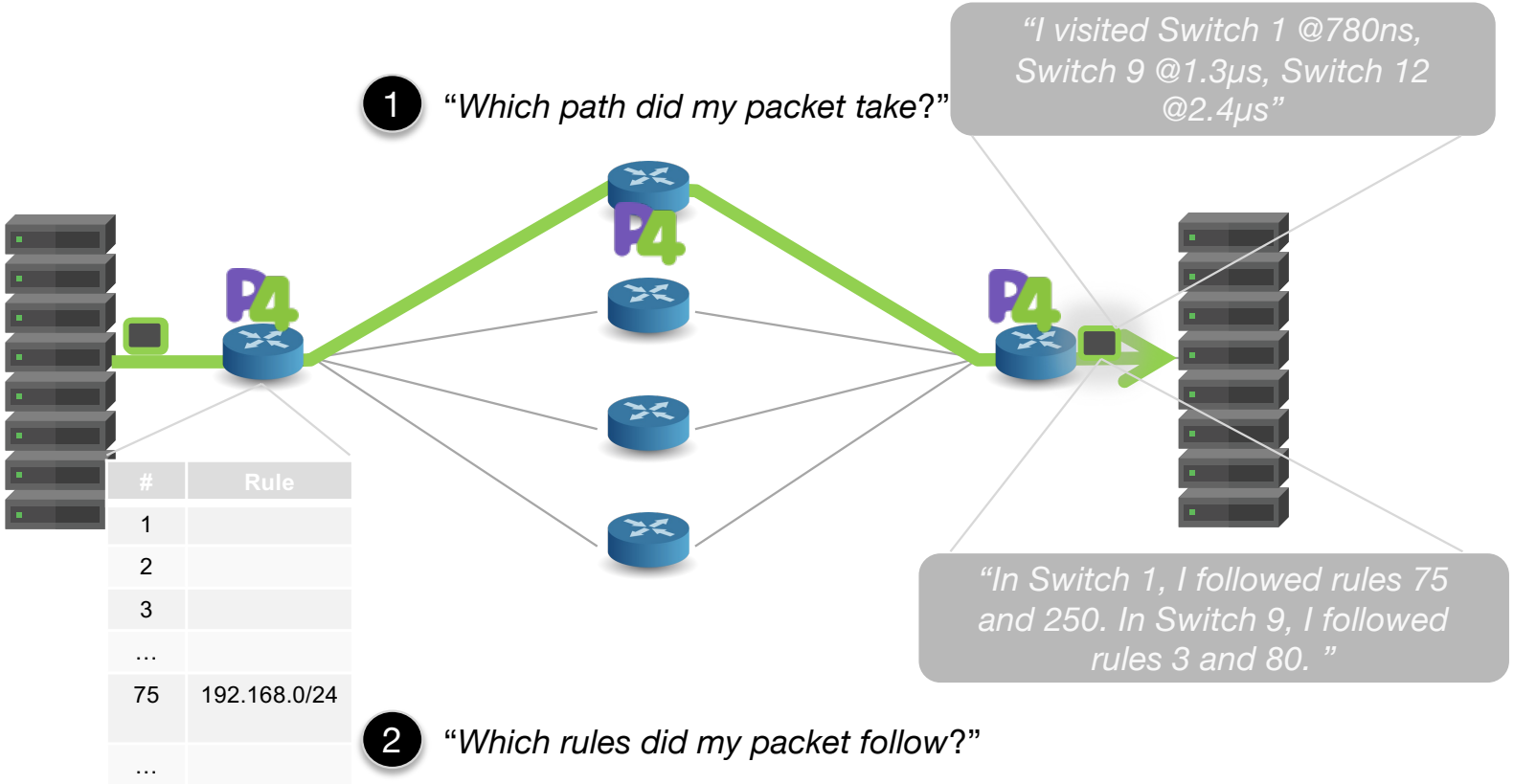


Million Minions Approach: Packets = Minions

- **In-band Network Telemetry**
  - Collect network state in the dataplane
    - Switch ID, Ingress, Egress port ID
    - Egress Link Utilization
    - Hop latency, Queue occupancy, Congestion Status
    - …
  - Re-use network packets that traverse the data plane to collect monitoring information
  - attach it in programmable way to packet headers
  - At line speed, in a programmable way



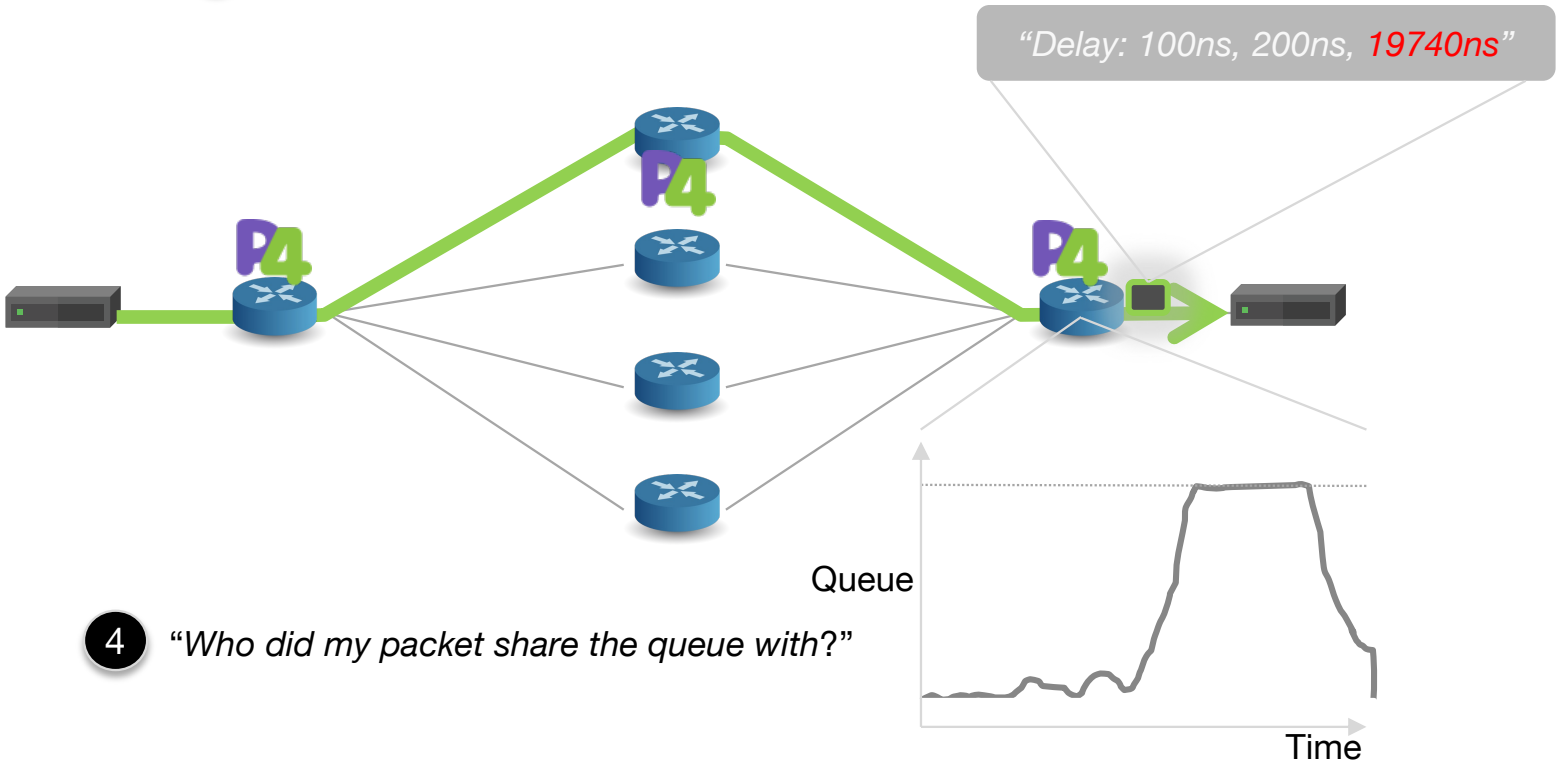
In-band Network Telemetry (INT) using P4



Based on: Nick McKeown (Stanford)

In-band Network Telemetry (INT) using P4

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

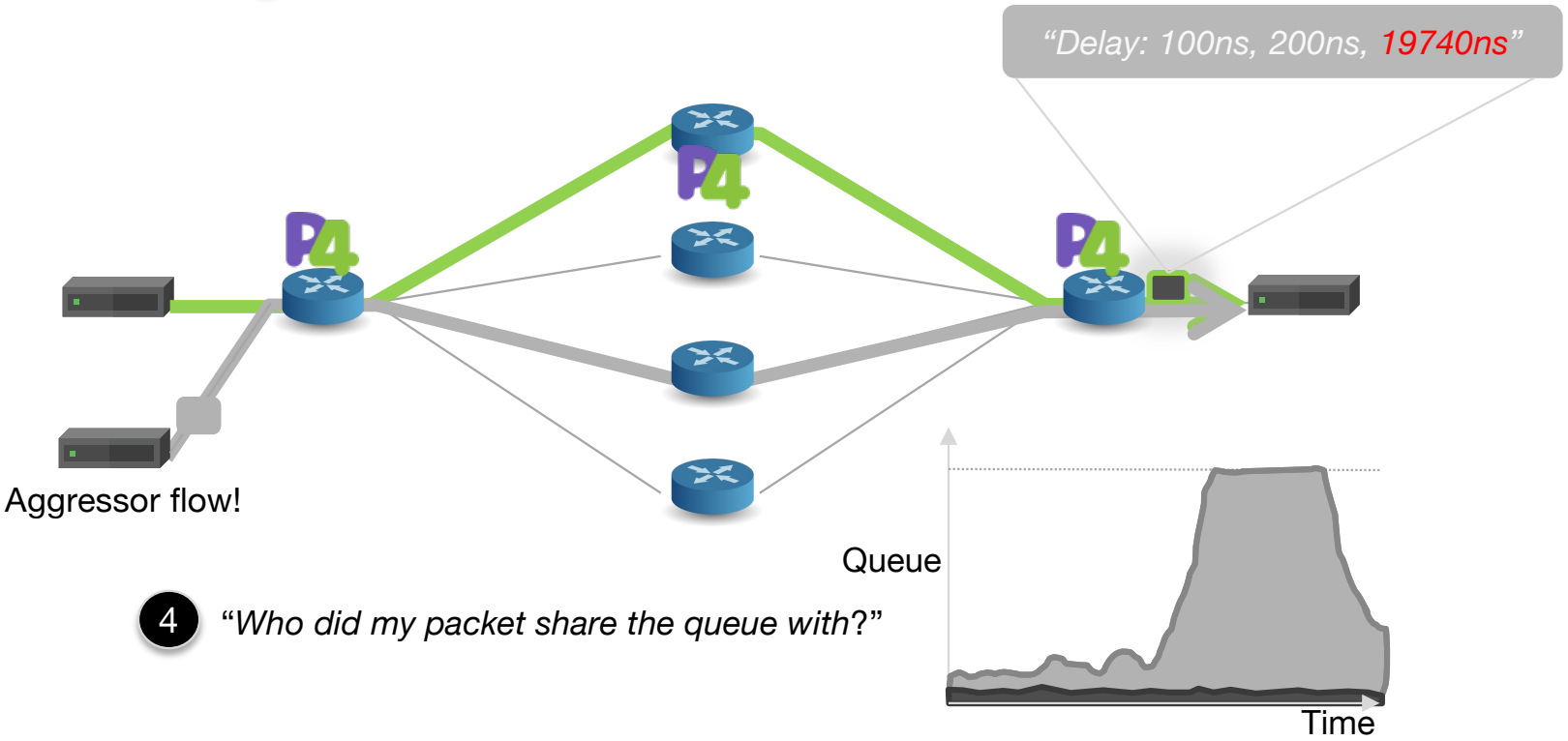


4 *"Who did my packet share the queue with?"*

Based on: Nick McKeown (Stanford)

In-band Network Telemetry (INT) using P4

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

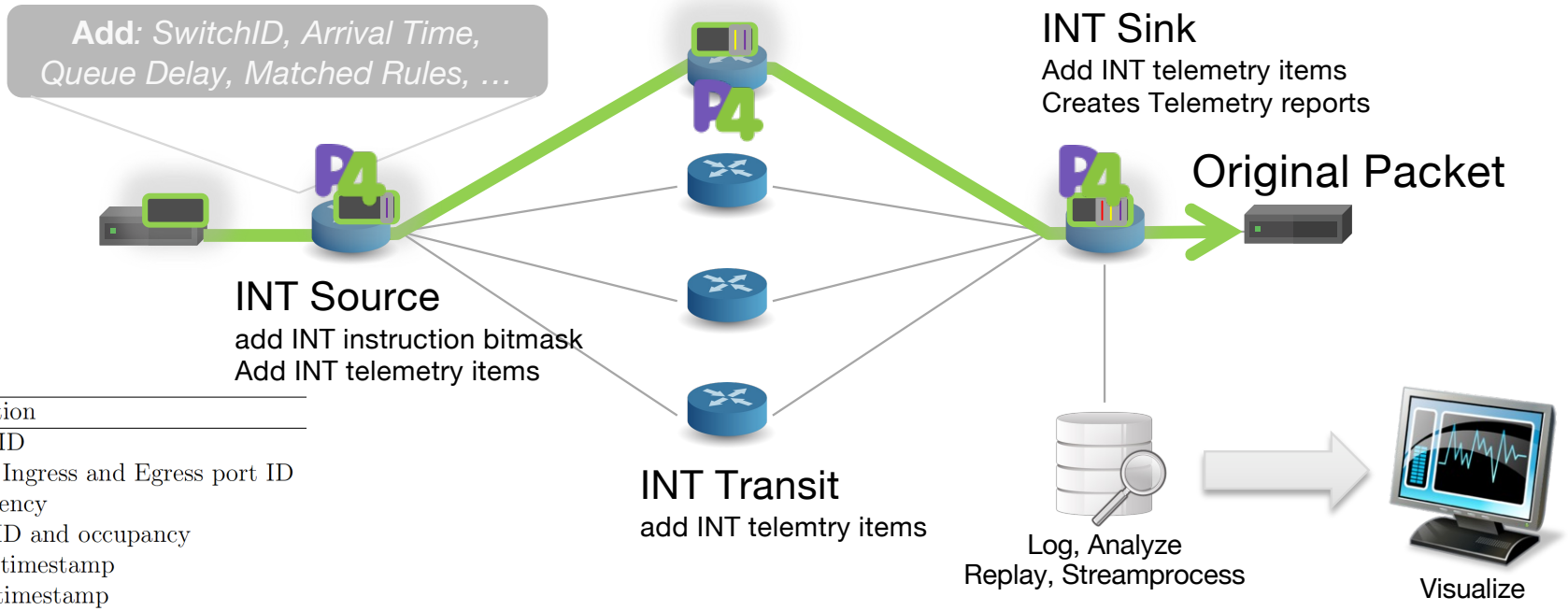


4 *"Who did my packet share the queue with?"*

Based on: Nick McKeown (Stanford)

In-band Network Telemetry (INT) using P4

- **Control Plane decides which Telemtry items to collect per flow**
  - Populates INT source with table entries to set instruction bits in headers



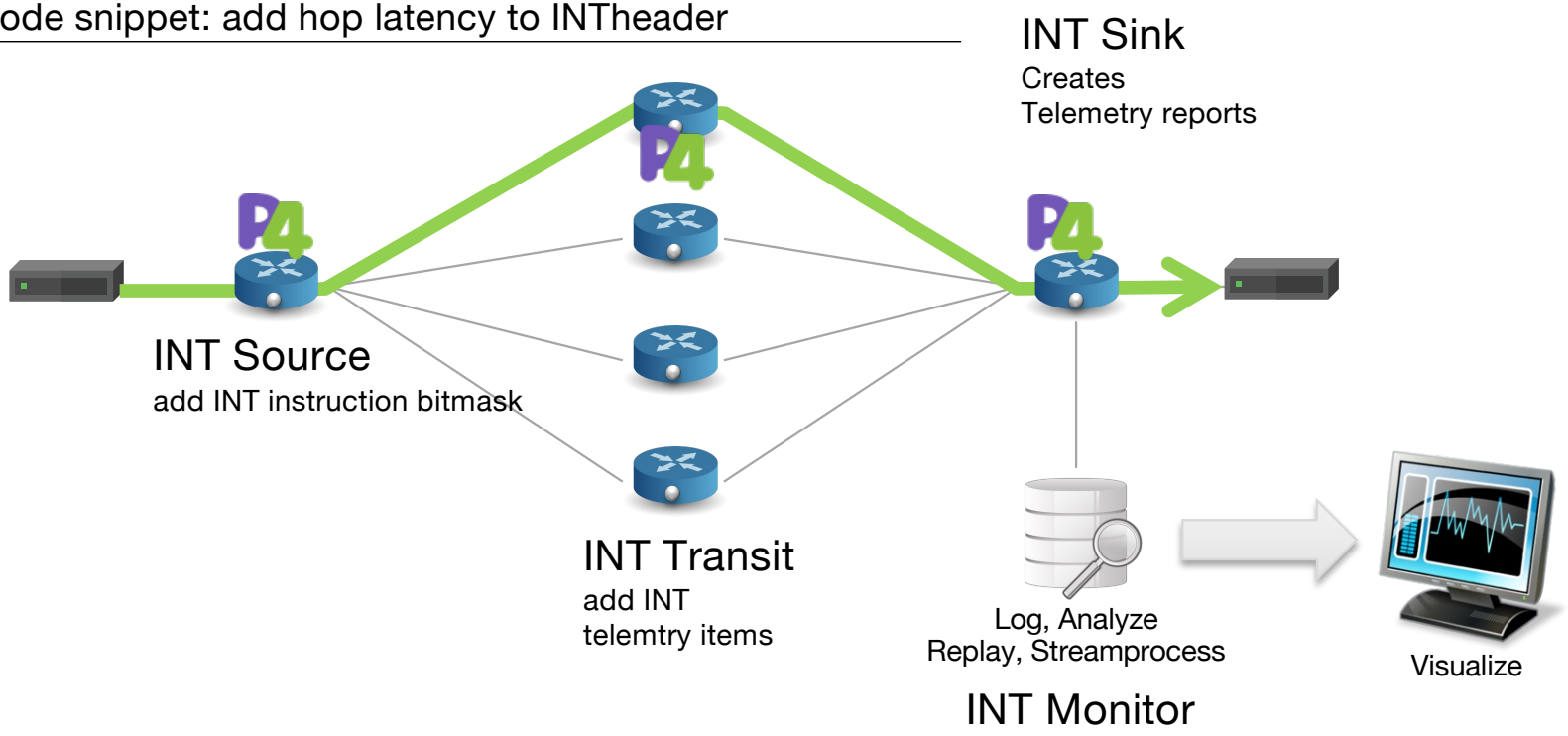
Bit (MSB)	Instruction
0	Switch ID
1	Level 1 Ingress and Egress port ID
2	Hop latency
3	Queue ID and occupancy
4	Ingress timestamp
5	Egress timestamp
6	Level 2 Ingress and Egress port ID
7	Egress port Tx utilization

Based on: Nick McKeown (Stanford)

In-band Network Telemetry (INT) using P4

```
action add_int_hop_latency () {
    hdr.int_metadata.push_front(1);
    hdr.int_metadata[0].data = (bit<32>)(meta.fwd_metadata.eg_timestamp -
     ↪  meta.fwd_metadata.ig_timestamp);
}
```

P4 code snippet: add hop latency to INTheader

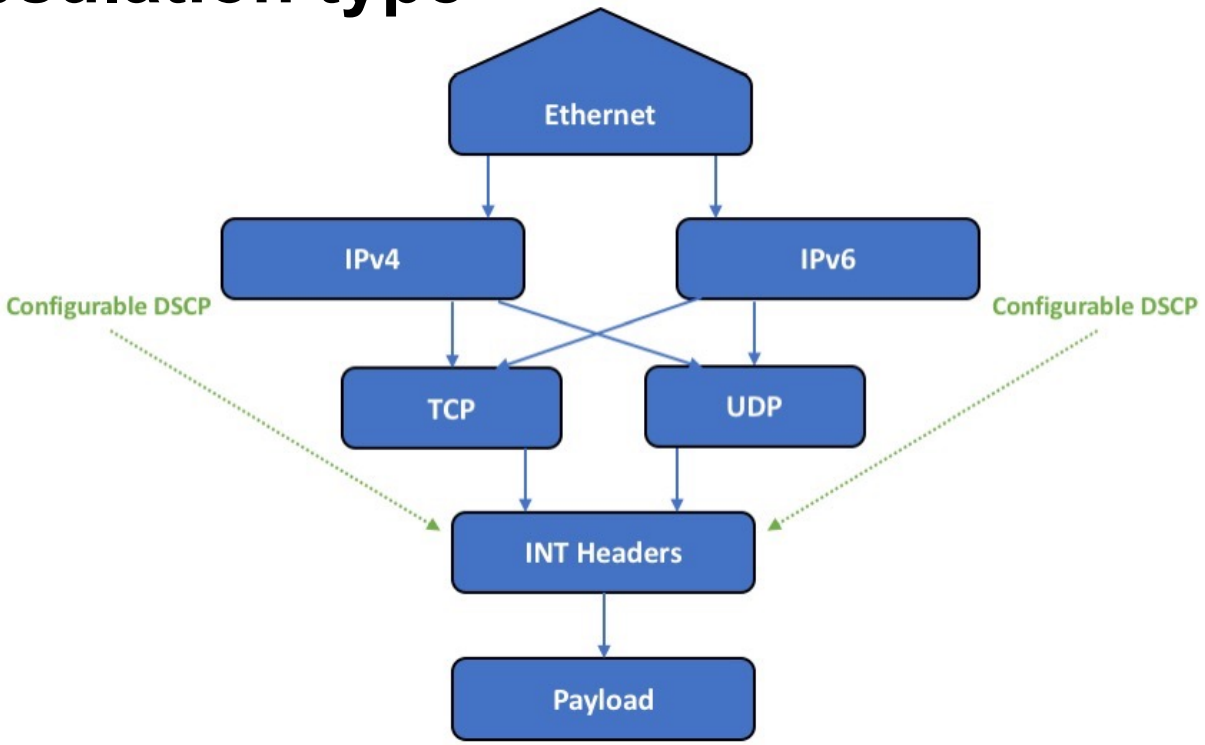


INT Header over encapsulation

- **INT header can be inserted as an option or payload of any encapsulation type**
  - INT over TCP/UDP (introduces a shim header after TCP/UDP header and carry INT Headers between the shim header and TCP/UDP payload)
  - INT over VXLAN (as VXLAN payload, per GPE extension)
  - INT over Geneve (as Geneve option)
  - INT over NSH (as NSH payload)
  - INT over GRE (as shim header between GRE and encapsulated payload)
- **Need to be supported by all devices on the path**

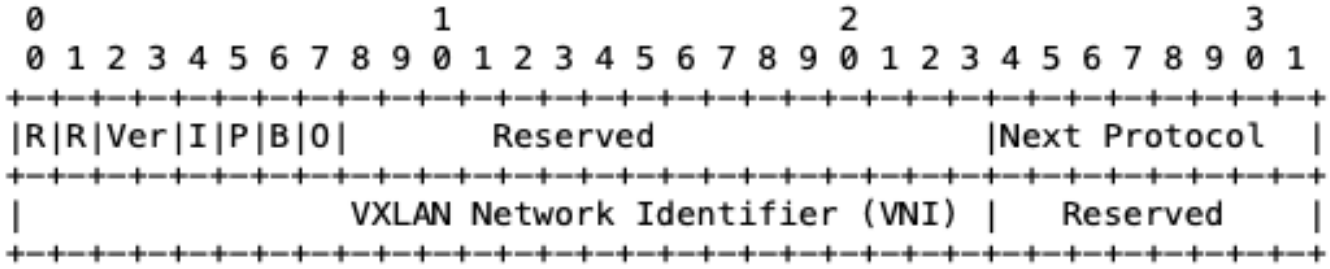
INT Header over encapsulation

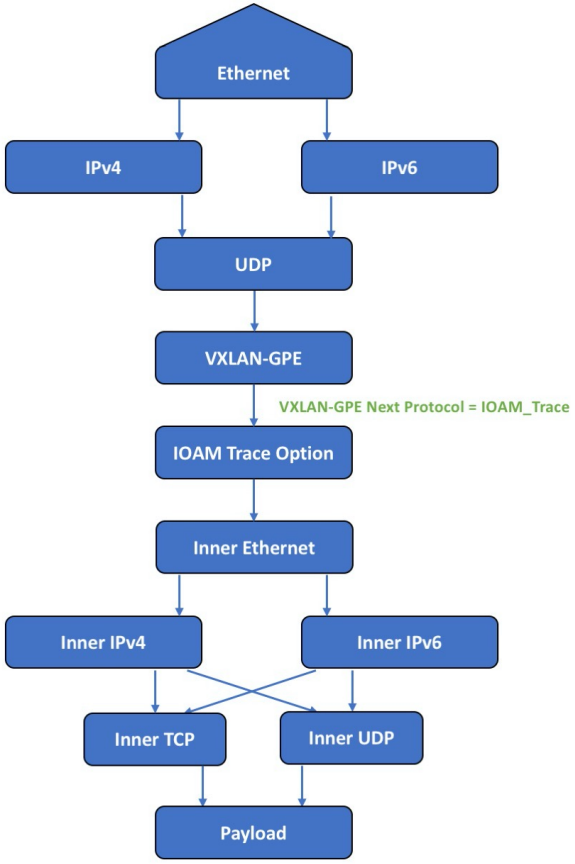
- **INT header can be inserted as an option or payload of any encapsulation type**



INT Header over encapsulation

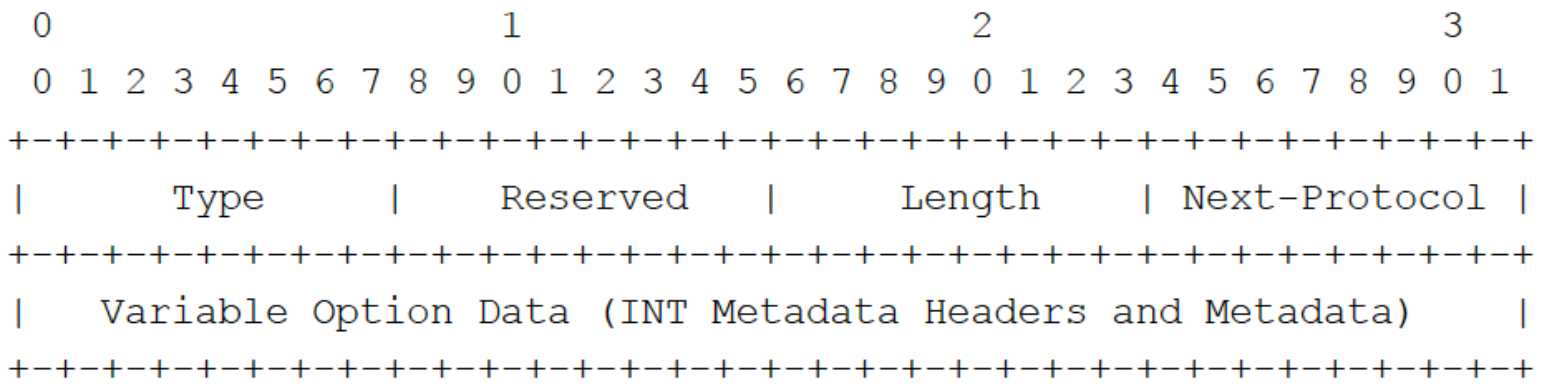
- **INT header can be inserted as an option or payload of any encapsulation type**
  - INT over VXLAN (as VXLAN payload, per GPE extension)
    - Next Protocol = 0x01 → IPv4
    - Next Protocol = 0x02 → IPv6
    - Next Protocol = 0x03 → Ethernet
    - Next Protocol = 0x04 → Network Service Header (NSH)
    - Next Protocol = 0x05 → IOAM_TRACE (INT)





INT Header embedding

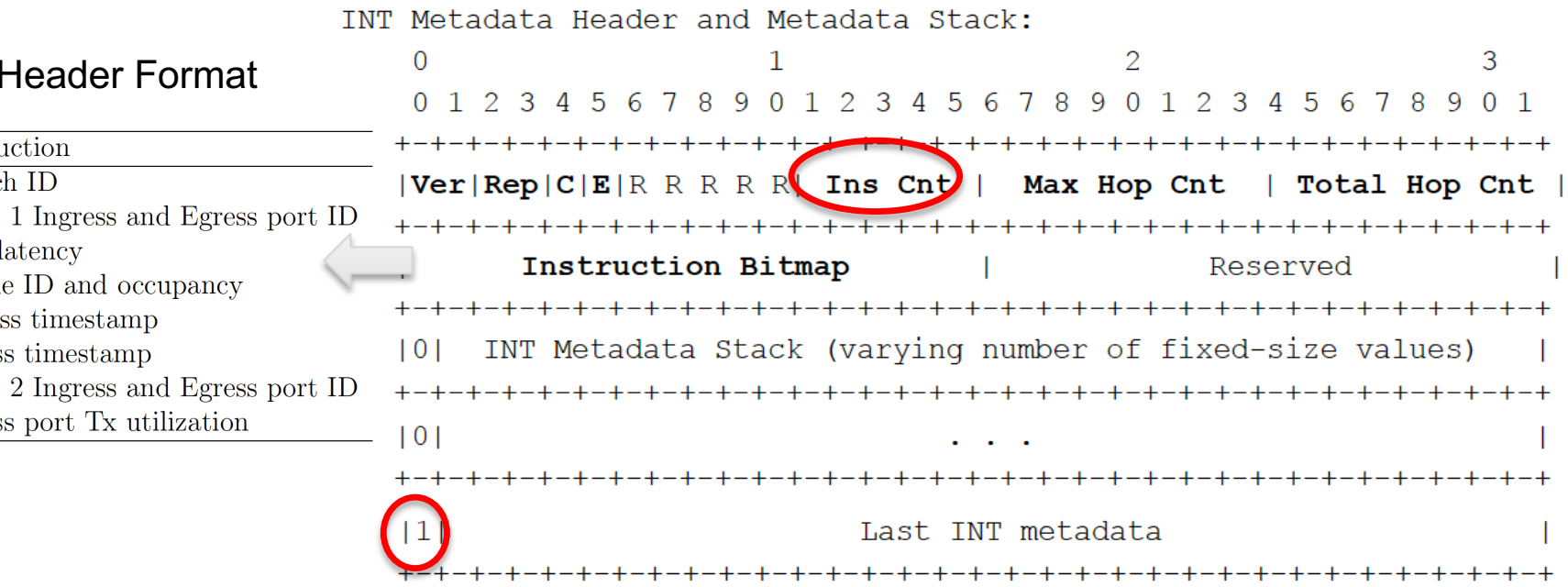
- **INT Metadata has variable length**
  - INT shim header following the GPE
    - Type: INT header is following
    - Length: total length of variable INT option data and shim header in 4 byte words



INT Metadata header and Metadata stack

INT Metadata Header Format

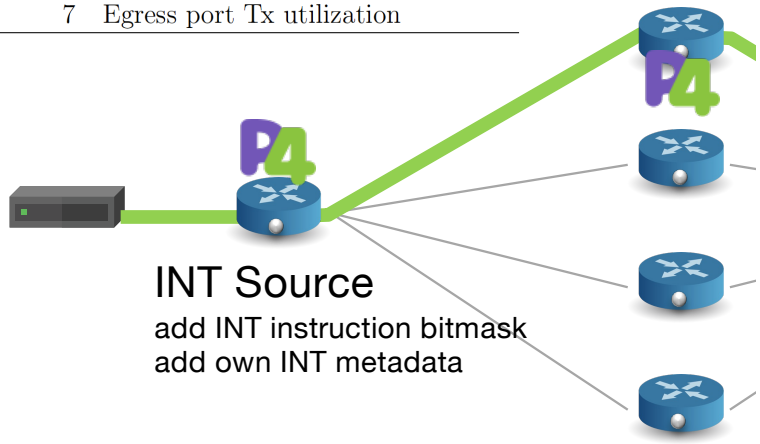
Bit (MSB)	Instruction
0	Switch ID
1	Level 1 Ingress and Egress port ID
2	Hop latency
3	Queue ID and occupancy
4	Ingress timestamp
5	Egress timestamp
6	Level 2 Ingress and Egress port ID
7	Egress port Tx utilization



- **Each INT devices pushes its own metadata at front of stack**

INT Metadata header and Meta

Bit (MSB)	Instruction
0	Switch ID
1	Level 1 Ingress and Egress port ID
2	Hop latency
3	Queue ID and occupancy
4	Ingress timestamp
5	Egress timestamp
6	Level 2 Ingress and Egress port ID
7	Egress port Tx utilization



INT Source

add INT instruction bitmask
add own INT metadata

INT Tran

add INT
telemtry item

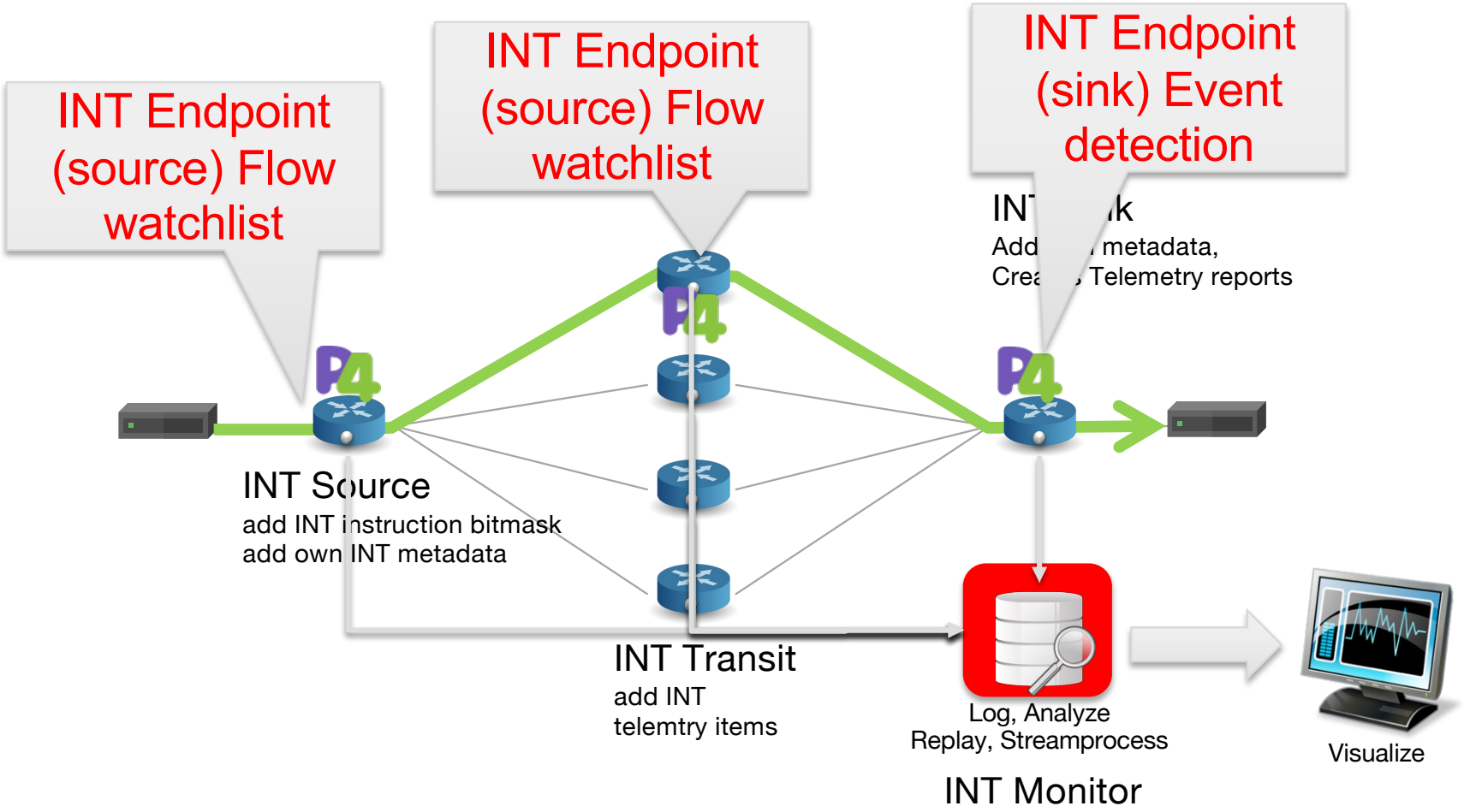
```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|R|R|R|R|1|1|R|R|           Reserved            | NextProto=0x5 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                VXLAN Network Identifier (VNI) |   Reserved    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type=1    |   Reserved    |  Length=9     | NextProto=0x5 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver|Rep|C|E|    R    | InsCnt=2|  MaxHopCnt=16 | TotalHopCnt=3 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0|           Reserved            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|                         sw id of hop3                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|                     queue occupancy of hop3                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|                         sw id of hop2                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|                     queue occupancy of hop2                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|                         sw id of hop1                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1|                     queue occupancy of hop1                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type=2    |   Reserved    |   Length=6    | NextProto=0x3 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver|Rep|C|E|    R    | InsCnt=1|  MaxHopCnt=16 | TotalHopCnt=3 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0|           Reserved            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|                   queue occupancy of hop3 (sw1)               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|                   queue occupancy of hop2 (sw2)               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1|                   queue occupancy of hop1 (sw3)               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Encapsulated Ethernet Payload               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

THIS ASSUMED AN OLDER VERSION OF THE STANDARD

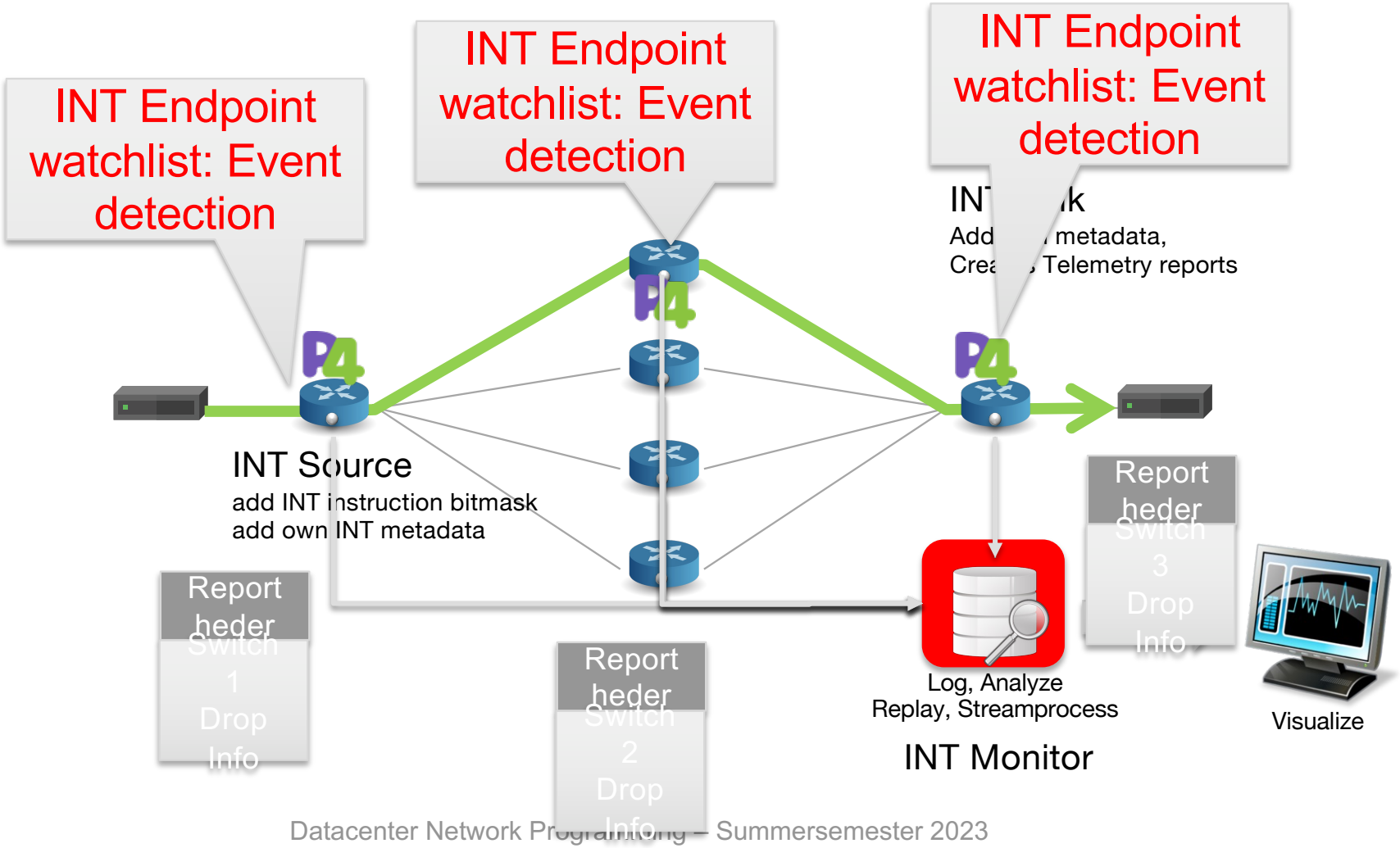
INT report types

- **Local Flow Report – generated from flow events**
  - Sent from source or sink for host-to-host data flows matching the whitelist
- **Drop Report - generated from drop events**
  - Every INT switch sends to monitor collector
- **Queue Congestion Report – generated from queue related events**
  - Sent for packets where queue depth or latency is exceeded
- **INT reports – sent by INT sinks, two reports generated**
  - Local flow report for traffic arriving on port
  - INT report for data received from the source

INT FLOW event – watchlist and event detection

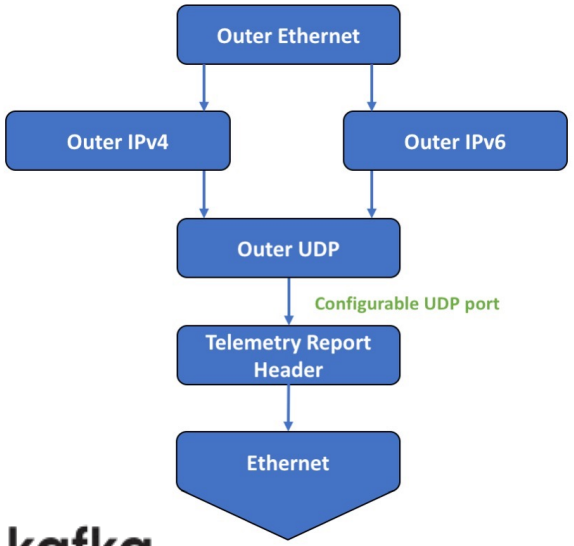


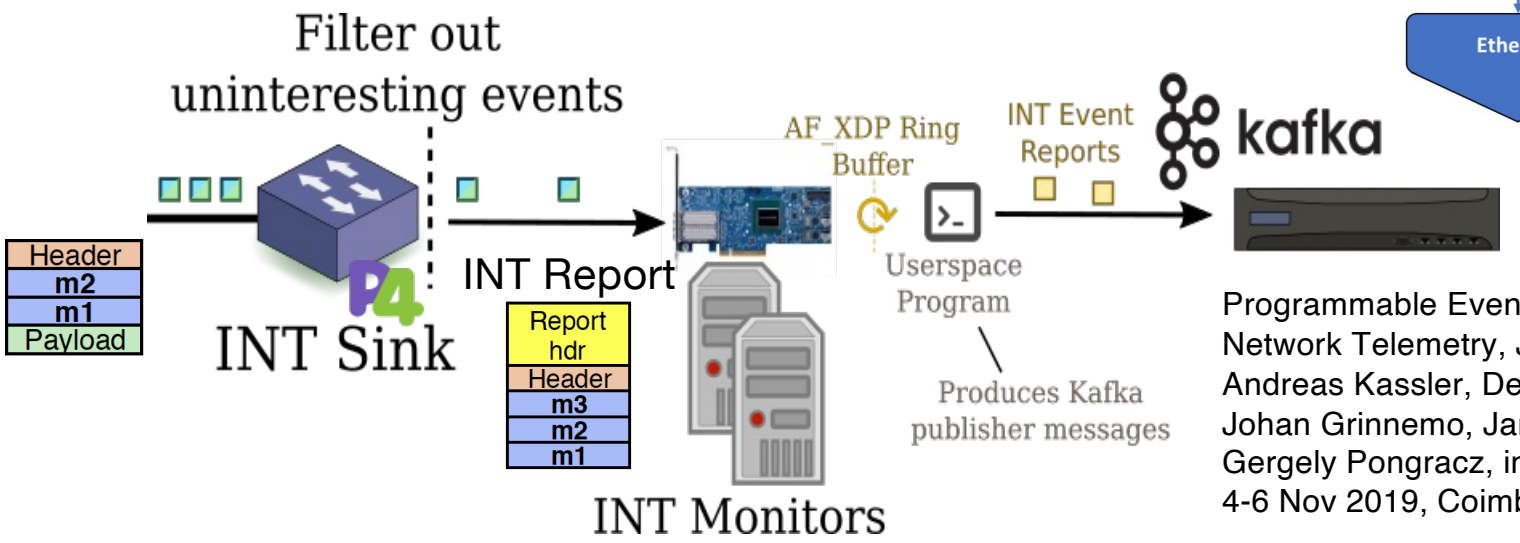
INT FLOW event – watchlist and event detection



INT Monitor

- **INT Report Packet format:**
  - Several formats defined
  - Efficient header parsing required to
    - retrieve INT metadata
    - identify the flow



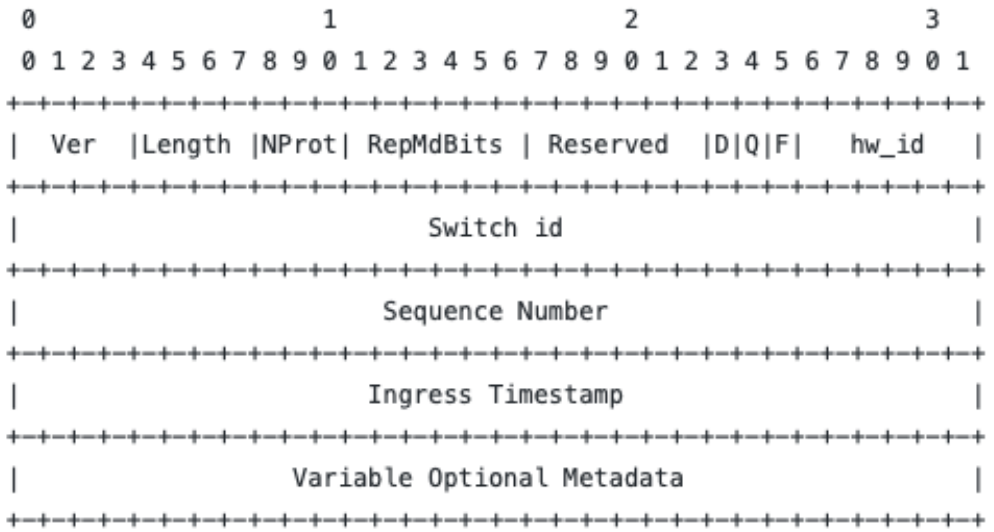


Programmable Event Detection for In-Band Network Telemetry, Jonathan Vestin, Andreas Kassler, Deval Bhamare, Karl-Johan Grinnemo, Jan-Olof Andersson, Gergely Pongracz, in IEEE CloudNet2019, 4-6 Nov 2019, Coimbra, Portugal

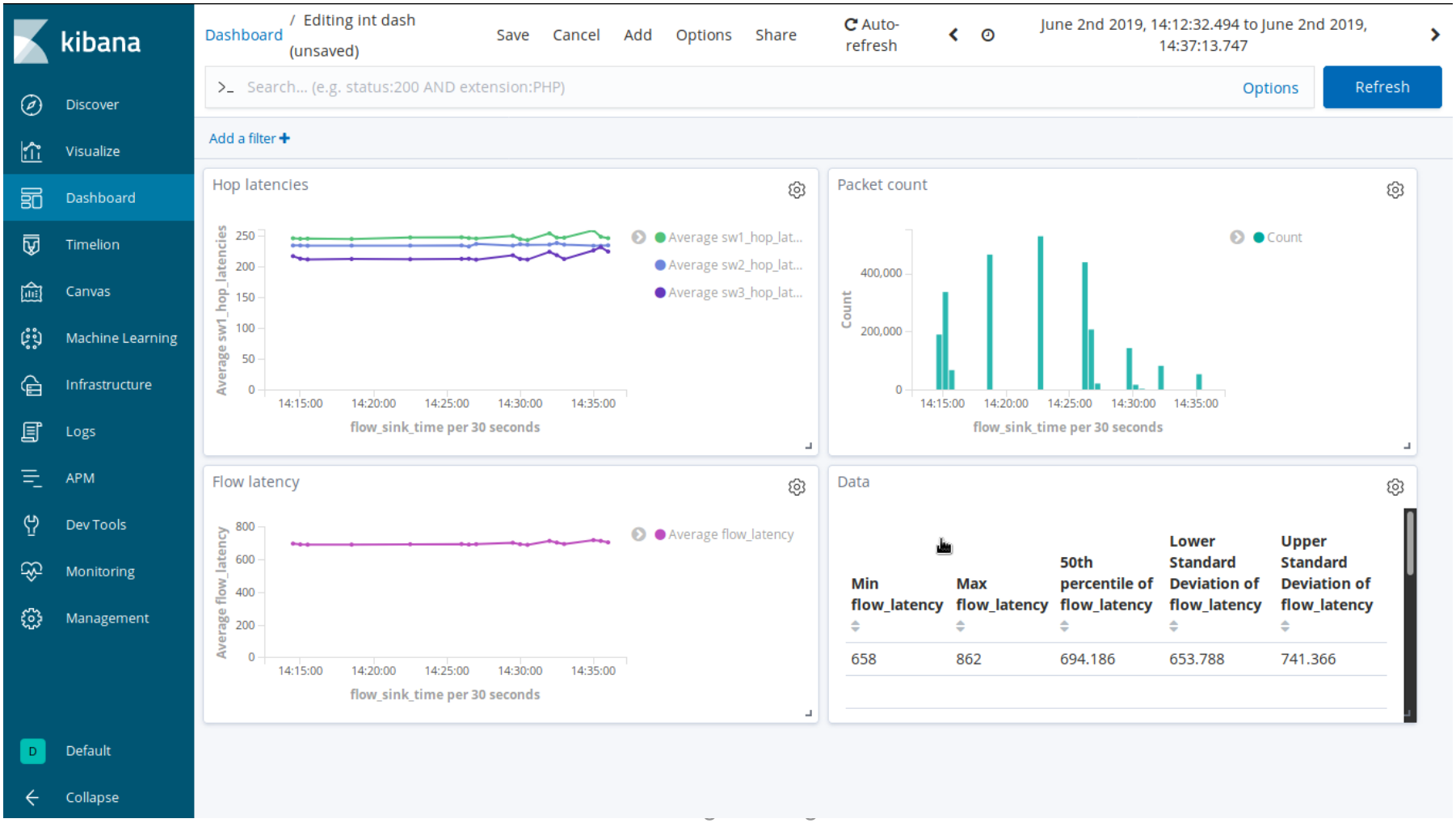
INT Monitor report HEader

- **RepMdBits: Report Metadata Bits**
- **Bitmap that indicates which optional metadata is present in the telemetry report header.**
  - bit 0 (MSB): Ingress port id (16 bits) + Egress port id (16 bits)
  - bit 1: Hop latency
  - bit 2: Queue ID (8 bits) + Queu
  - bit 3: Egress Timestamp
  - bit 4: Queue ID (8 bits) + Drop
  - bit 5: Egress port tx utilization

Drop reason, see https://github.com/p4lang/switch/blob/master/p4src/includes/drop_reason_codes.h



Can push to Kafka and Big Data Analytics toolchain



In-band Network Telemetry (INT) using P4

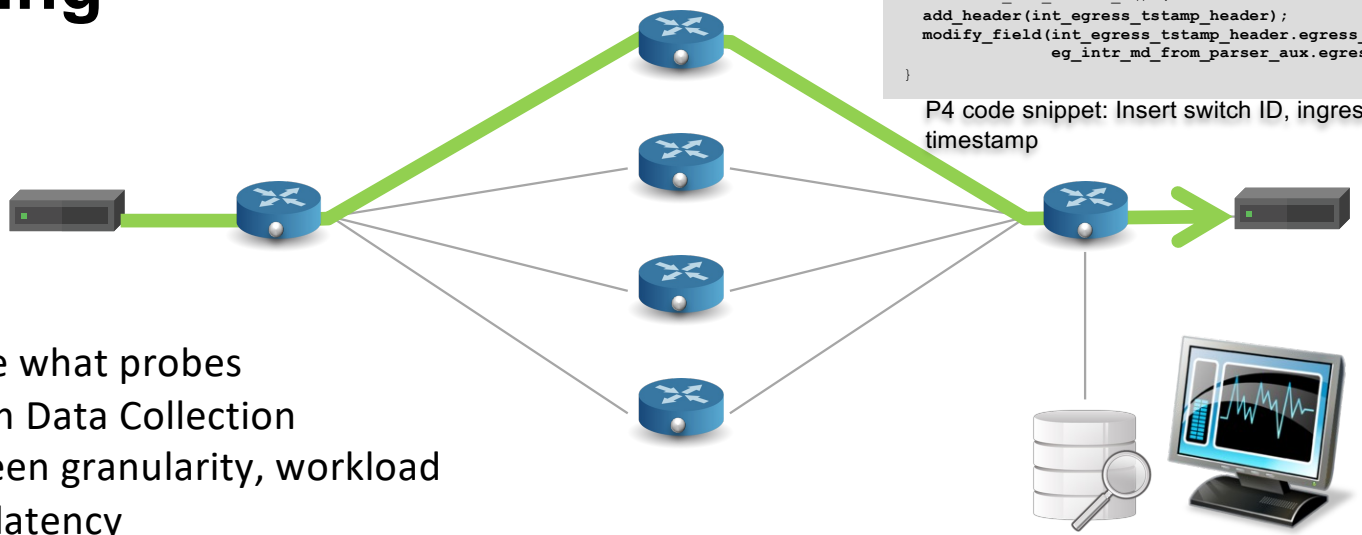
- **Challenge is in Where and How to add what Telemetry items for network wide customized monitoring**

```
/* INT: add switch id */
action int_set_header_0() {
  add_header(int_switch_id_header);
  modify_field(int_switch_id_header.switch_id,
               global_config_metadata.switch_id);
}

/* INT: add ingress timestamp */
action int_set_header_1() {
  add_header(int_ingress_tstamp_header);
  modify_field(int_ingress_tstamp_header.ingress_tstamp,
i2e_metadata.ingress_tstamp);
}

/* INT: add egress timestamp */
action int_set_header_2() {
  add_header(int_egress_tstamp_header);
  modify_field(int_egress_tstamp_header.egress_tstamp,
               eg_intr_md_from_parser_aux.egress_global_tstamp);
}
```

P4 code snippet: Insert switch ID, ingress and egress timestamp

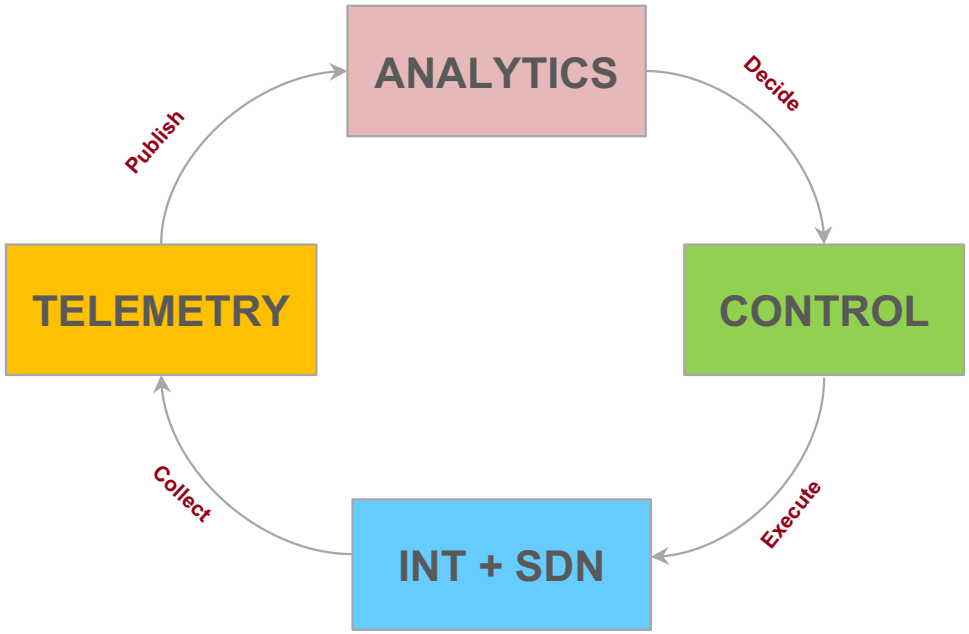


Problem:

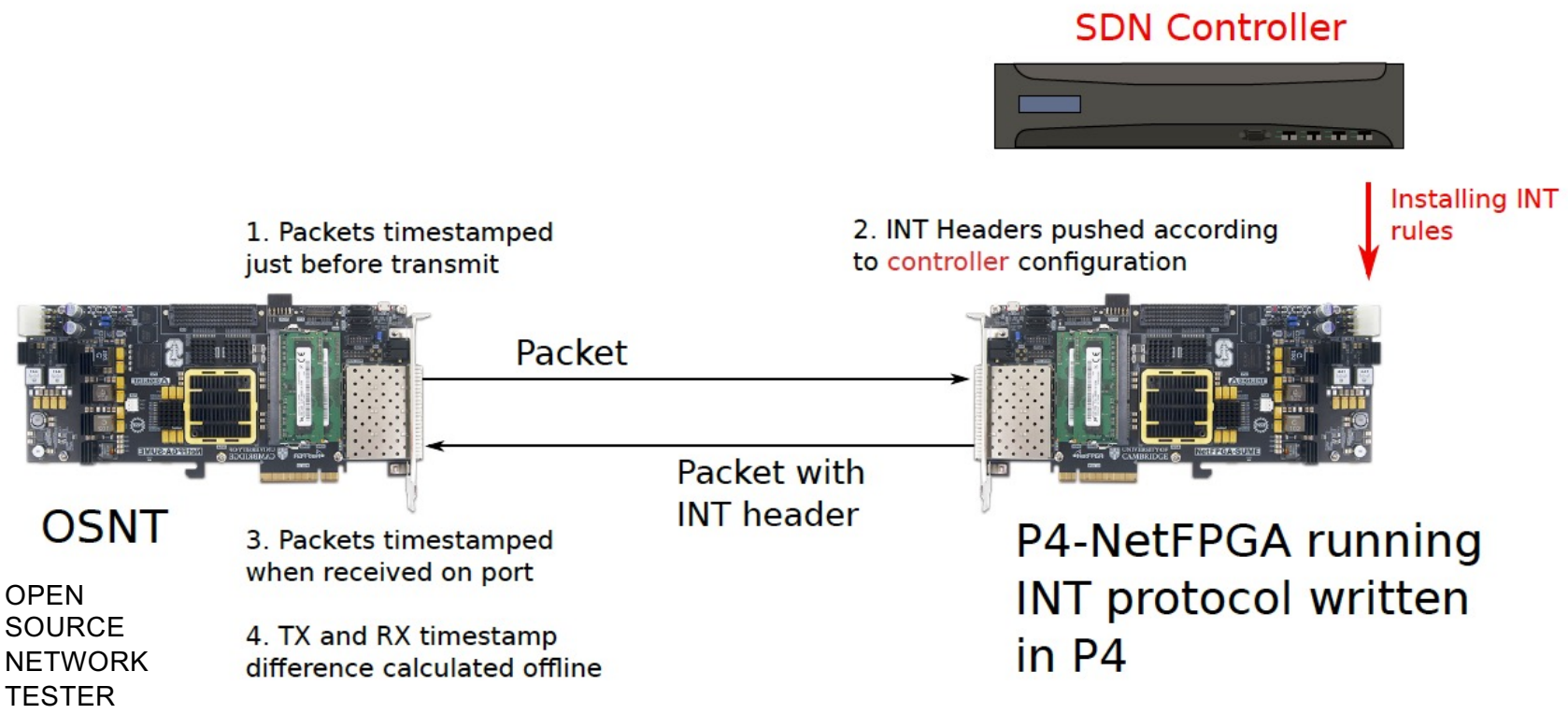
- When to create what probes
- When to inform Data Collection
- Tradeoff between granularity, workload and overhead/latency

INT Analytics Framework

- **Goal**
  - Enable programmable observability and closed loop control based on analytics
    - Fault prediction, resource utilization prediction, anomaly detection, SLA violation detection
  - Activate smart probes and observe traffic for a target subscriber for a given time and notify events

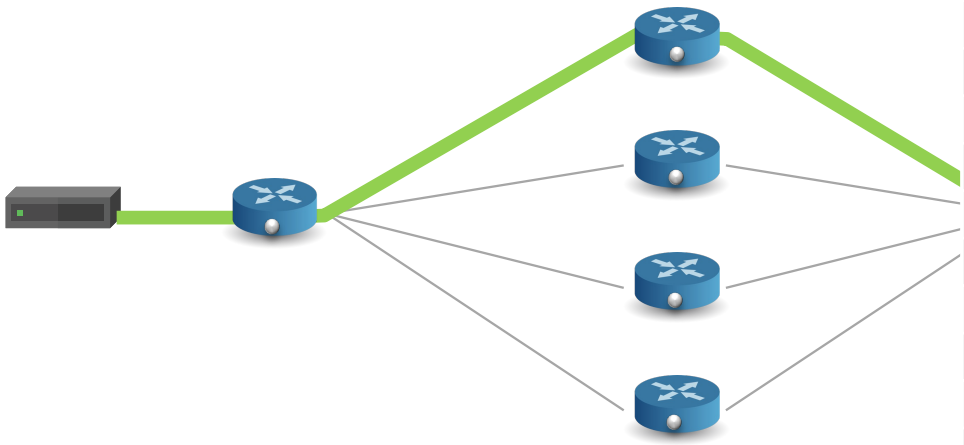


INT OVERHEAD using P4



INT OVERHEAD using P4

- **INT Overhead benchmark results**
  - NetFPGA: scales linearly with INT items to push
  - Edge more overhead due to more complex P4 code

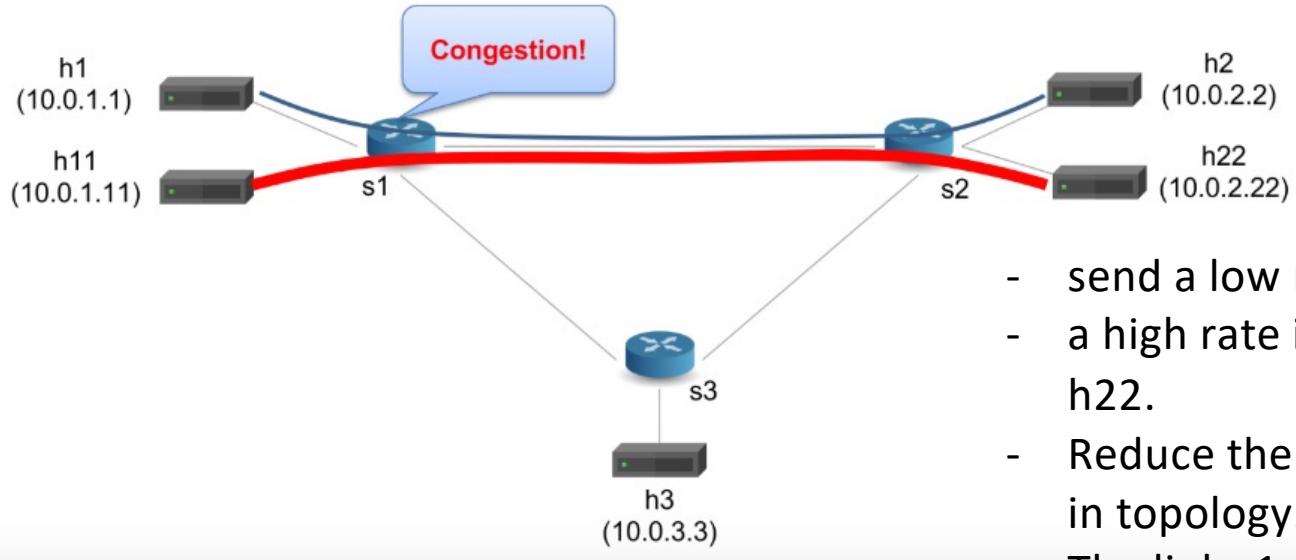


Header fields	Core (µs)	Edge (µs)
1	5.22	5.45
2	5.73	5.96
3	6.05	6.29
4	6.41	6.64
5	6.73	6.96
6	7.06	7.29
7	7.38	7.61
8	7.71	7.94
9	8.10	8.32
10	8.44	8.66
11	8.79	9.01
12	9.13	9.35
13	9.48	9.70
14	9.83	10.04
15	10.17	10.39
16	10.52	10.73

INT summary

- **INT provides programmable real-time network state collection in the data plane**
  - Scalable to large networks
  - Scalable to any link speed
  - Supports and encapsulation, network and application
- **Having knowledge of real-time network state enables**
  - New monitoring and troubleshooting methods
  - Routing aware of network state
  - Congestion aware load-balancing (HULA)
- **New Development: Probabilistic INT → PINT**

INT Programming Assignment - MRI



Multi-Hop Route Inspection (MRI)

- send a low rate traffic from h1 to h2
- a high rate iperf traffic from h11 to h22.
- Reduce the BW of link s1-s2 to 512kbps in topology.json.
- The link s1-s2 is common between the flows and is a bottleneck
- Insert the telemetry items such as Queue Length using P4
- Capture packets at h2, and observe high queue size for that link.

mRI header structure

```
struct headers {
    ethernet_t          ethernet;
    ipv4_t              ipv4;
    ipv4_option_t       ipv4_option;
    mri_t               mri;
    switch_t[MAX_HOPS]  swtraces;
}
```

```
header mri_t {
    bit<16>  count;
}

header switch_t {
    switchID_t  swid;
    qdepth_t    qdepth;
}
```

MRI egress

```
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t standard_metadata) {
    action add_swtrace(switchID_t swid) {
        hdr.mri.count = hdr.mri.count + 1;
        hdr.swtraces.push_front(1);
        // According to the P4_16 spec, pushed elements are invalid, so we need
        // to call setValid(). Older bmv2 versions would mark the new header(s)
        // valid automatically (P4_14 behavior), but starting with version 1.11,
        // bmv2 conforms with the P4_16 spec.
        hdr.swtraces[0].setValid();
        hdr.swtraces[0].swid = swid;
        hdr.swtraces[0].qdepth = (qdepth_t)standard_metadata.deq_qdepth;

        hdr.ipv4.ihl = hdr.ipv4.ihl + 2;
        hdr.ipv4_option.optionLength = hdr.ipv4_option.optionLength + 8;
        hdr.ipv4.totalLen = hdr.ipv4.totalLen + 8;
    }
```

MRI Parsing states

```
state parse_mri {
    packet.extract(hdr.mri);
    meta.parser_metadata.remaining = hdr.mri.count;
    transition select(meta.parser_metadata.remaining) {
        0 : accept;
        default: parse_swtrace;
    }
}
```

SWTRACE


```
state parse_swtrace {
    packet.extract(hdr.swtraces.next);
    meta.parser_metadata.remaining = meta.parser_metadata.remaining  - 1;
    transition select(meta.parser_metadata.remaining) {
        0 : accept;
        default: parse_swtrace;
    }
}
```

REMAINING = 2-1 = 1

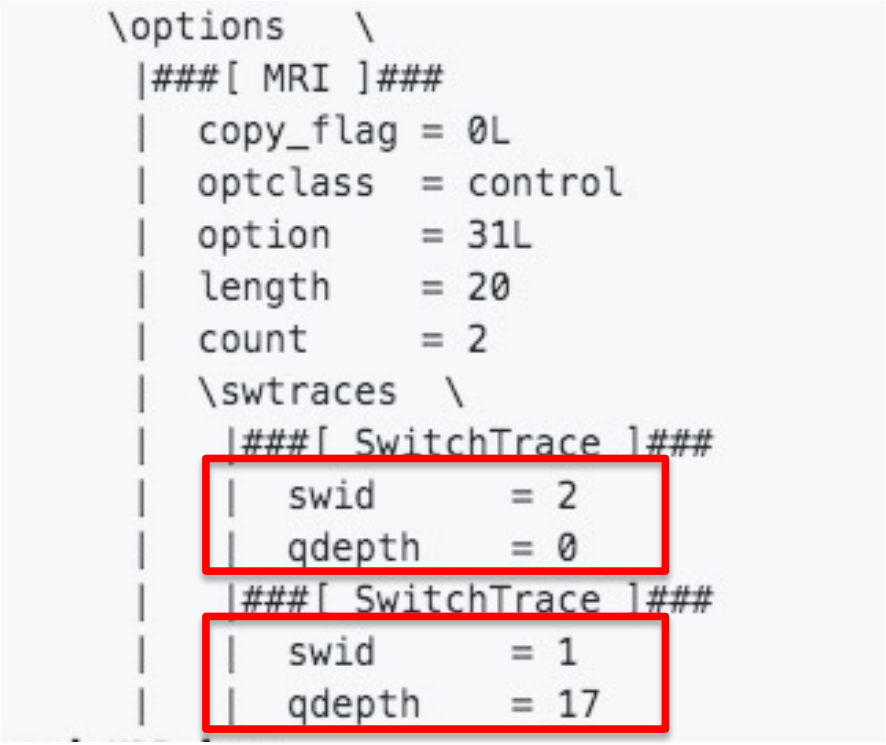
REMAINING = 1-1 = 0

MRI egress

```
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t standard_metadata) {
    action add_swtrace(switchID_t swid) {
        hdr.mri.count = hdr.mri.count + 1;
        hdr.swtraces.push_front(1);
        // According to the P4_16 spec, pushed elements are invalid, so we need
        // to call setValid(). Older bmv2 versions would mark the new header(s)
        // valid automatically (P4_14 behavior), but starting with version 1.11,
        // bmv2 conforms with the P4_16 spec.
        hdr.swtraces[0].setValid();
        hdr.swtraces[0].swid = swid;
        hdr.swtraces[0].qdepth = (qdepth_t)standard_metadata.deq_qdepth;
        hdr.ipv4.ihl = hdr.ipv4.ihl + 2;
        hdr.ipv4_option.optionLength = hdr.ipv4_option.optionLength + 8;
        hdr.ipv4.totalLen = hdr.ipv4.totalLen + 8;
    }
```

```
typedef bit<32> switchID_t;
typedef bit<32> qdepth_t;
```

INT Assignment - Results



MRI timestamps

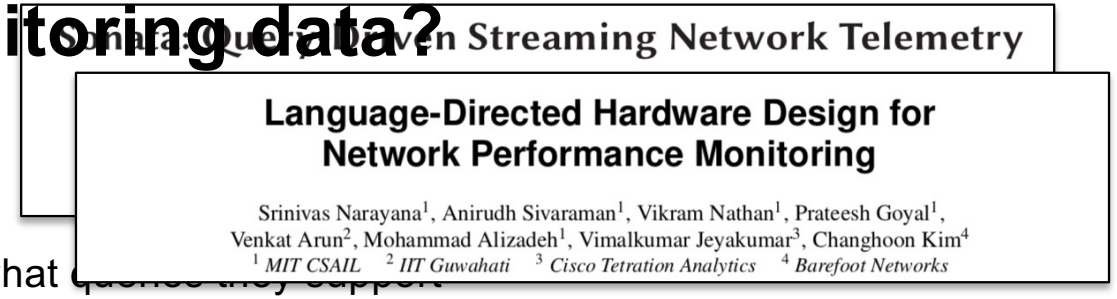
```
hdr.ioam_payload[0].tx_nsec =
             (bit<32>)meta.intrinsic_metadata.current_global_tstamp;
hdr.gpe_ioam.len = hdr.gpe_ioam.len + 4;

hdr.ipv4.totalLen = hdr.ipv4.totalLen + 32;
hdr.udp.len = hdr.udp.len + 32;
```

```
[0] switch_id | bos : 0x0 | switch_id : 0xcafe |
[0] ingress_ts | bos : 0x0 | value : 0x7661ee71 |
[0] egress_ts | bos : 0x0 | value : 0x7661fe71 |
[1] switch_id | bos : 0x0 | switch_id : 0xcafe |
[1] ingress_ts | bos : 0x0 | value : 0x7661ca93 |
[1] egress_ts | bos : 0x0 | value : 0x7661da93 |
[2] switch_id | bos : 0x0 | switch_id : 0xcafe |
[2] ingress_ts | bos : 0x0 | value : 0x7661a076 |
[2] egress_ts | bos : 0x1 | value : 0x0x7661ba93 |
```

Flexible Monitoring Queries

- **How to query for monitoring data?**
  - Marple (Sigcomm 2017)
  - Sonata (Sigcomm 2018)
  - Express monitoring query
    - Marple and Sonata differ in what queries they support
  - Compiler creates
    - data plane code and
    - control plane interaction




```
result = filter(pktstream, qid == 10 and
                switch == S and t_out - t_in > 0.1ms)
```

Identifies packetstream which experiences queuing delay larger than 0.1 ms

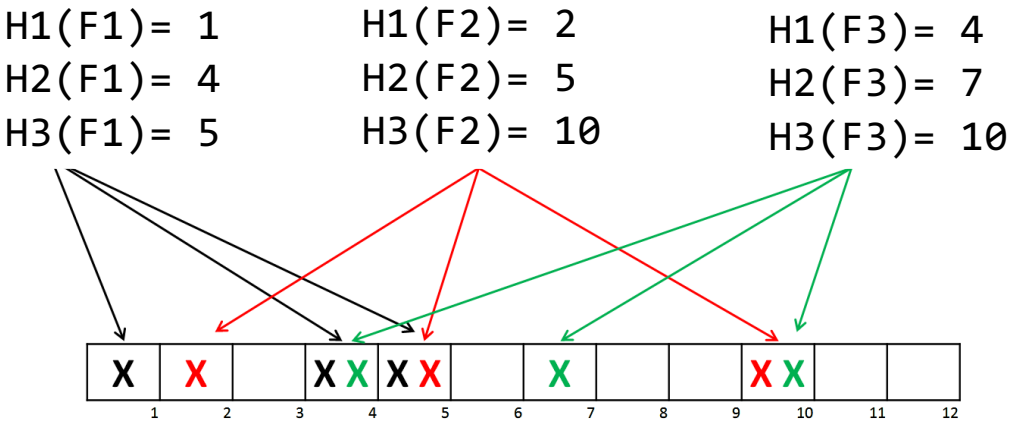
Flexible Flow Monitoring

- **How to monitor all flow**
  - Limited switch resources
  - Cannot have flow counters per flo
- **Main idea**
  - Streaming data structures such as Bl
  - Pull related flow information to control

LossRadar: Fast Detection of Lost Packets in Data Center Networks

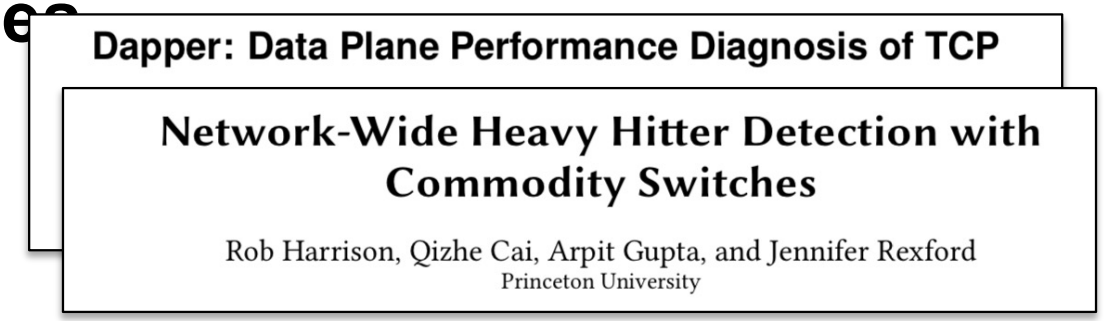
FlowRadar: A Better NetFlow for Data Centers

Yuliang Li* Rui Miao* Changhoon Kim† Minlan Yu*
*University of Southern California †Barefoot Networks



Diagnose Specific Flow Properties

- **Specific Flow Properties**
- **Dapper (SOSR 2017)**
  - TCP performance issues
    - Sender limited
    - Receiver limited
    - Network limited
  - Parses TCP headers in P4
- **Network wide Heavy Hitters (SOSR 2018)**
  - Port scan, Super spreaders, DDoS
  - Relies on efficient data structures in switch, e.g. HyperLogLog sketch
  - Centralized controller, communication overhead



Improve Sketch based Monitoring Primitives

- **Sketch based monitori**
  - E.g. Count-min sketch
  - Low overhead Streaming algorithm trading off accuracy/resources
  - Used for many monitoring
  - approaches
- **Improving upon them**
  - SketchLearn (SigComm 2018), htt
  - Elastic Sketch (SigComm 2018), compress and merge sketches
  - UnivMon (SigComm 2016), estimation algorithms run in control plane, and use the statistics from the data plane to compute application-level metrics