Systematic
Paris Region Systems & ICT Cluster

Agence Nationale de la Recherche
ANR

Monitoring L4S using INT and P4

Huu-Nghia NGUYEN

IMT Atlantique - Rennes, 23-03-2022

orange™

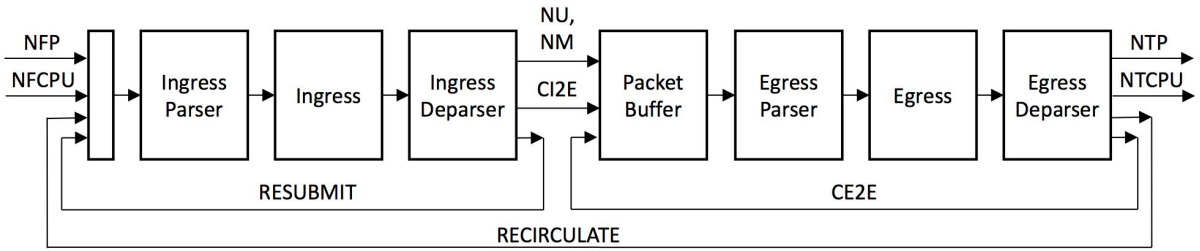
montimage

Loria
Laboratoire lorrain de recherche en informatique et ses applications

utt
université de technologie Troyes

Background

- Basic blocks and packet paths in $P4_{16}$ Portable Switch Architecture:



  - A P4 program needs to implement these 6 blocks
  - P4 L4S: an existing implementation of L4S using P4
- In-band Network Telemetry
  - *"in-band" feature offers the possibility to attach real-time network state to every packet at the line rate, which allows fine-grained monitoring*
  - 3 node types:
    - source node: add INT header into packet
    - transit node: embed metadata into the INT packet
    - sink node: remove INT header+data and restore original state of the packet
  - P4 INT: different existing implementations of INT using P4

Motivation

- Motivation:
  - introduce INT in P4 program of L4S to export monitoring metrics of L4S switches into outgoing packets
  - implement a high performance INT collector to collect the metrics from the packets
- Challenges:
  - monolithic P4 program => hard to write a portable and modular program
    - a new framework µP4* enables modular programming, but supports only Barefoot's Tofino
  - high packet rate at data plane => high INT report rate

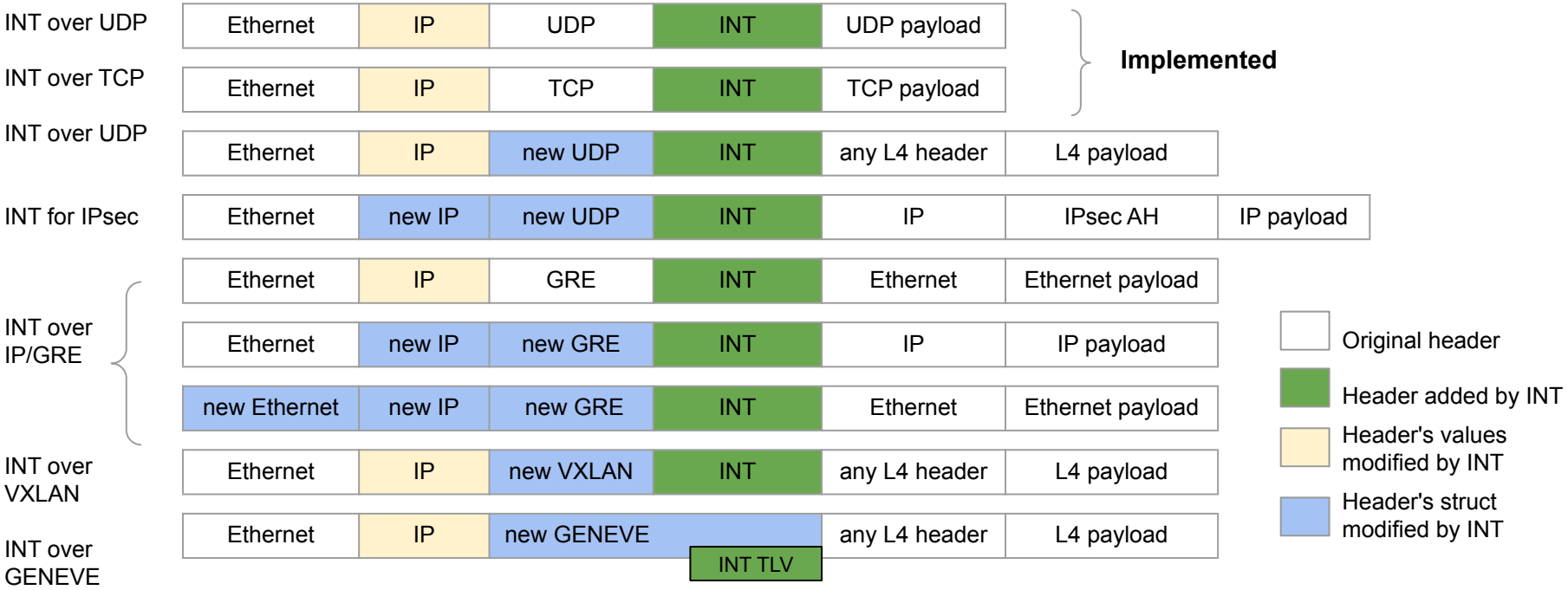
* H. Soni, M. Rifai, P. Kumar, R. Doenges, and N. Foster, "Composing Dataplane Programs with µP4," in *Proc. of SIGCOMM 2020*, 2020, pp. 329–343.

Outline

- Background & Motivation
- Introduce INT in L4S program
  - INT over TCP/UDP
  - Additional metrics for L4S
  - Introduce INT in L4S
- Another INT collector - MMT-Probe
- Demo
- Conclusion

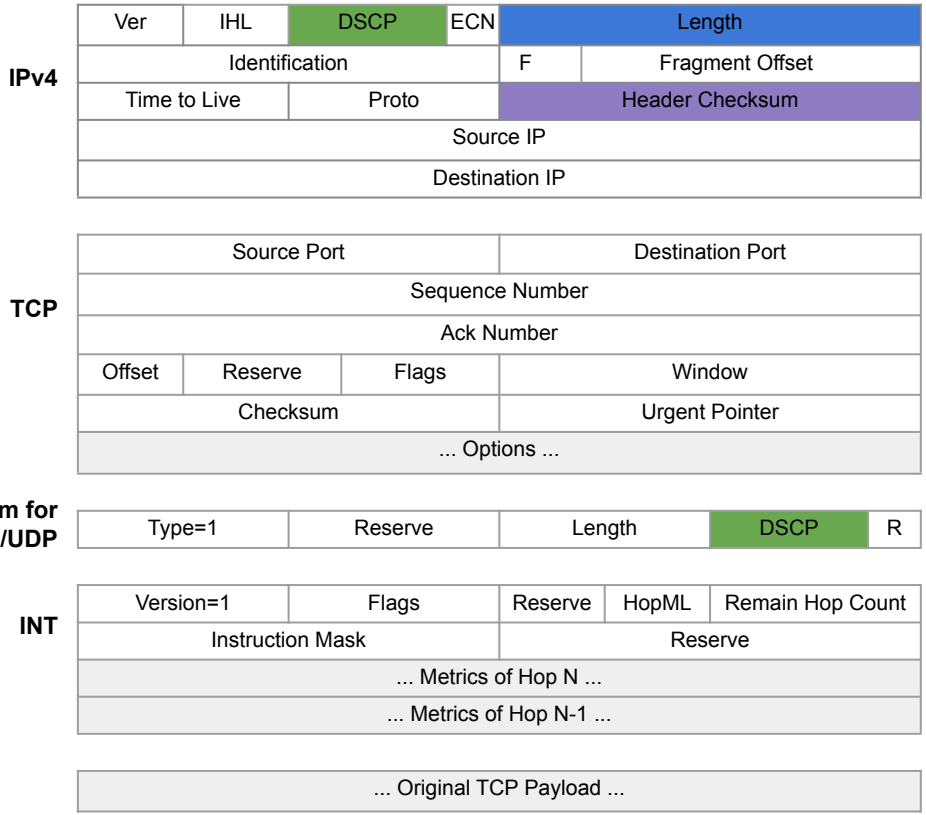
INT headers location in packets*

- Need to modify either fields or struct of precedent protocols to (1) mark the presence of INT and (2) update their length and checksum



* Synthesized from www.geant.org and p4.org

INT over TCP/UDP



- Modify IP:
  - DSCP=0x20
  - Update Length and Header Checksum
- Struct of Shim can be different for INT over other protocols
- Each switch adds INT data only if no exceed MTU
  - Source node:
    - add Shim+INT header (12 octets)
    - copy IP DSCP to Shim
    - set IP DSCP to 0x20
  - Sink node:
    - remove Shim+INT
    - restore IP DSCP
  - Any node:
    - add its metrics based on Instruction Mask
    - update IP Length and Header Checksum
    - activate M flag in INT header if exceed MTU

Instruction Mask

- 16 bits => max 16 metrics (8 standard INT metrics + 1 checksum complement => the remaining 7 bits are reserved for other metrics)

1. **Switch ID:** 32 bit - ID of the switch, assigned by admin
2. **In-egress Ports**: 16 bit ingress + 16 bit egress ports of the packet.
3. **Hop Latency**: 32 bit, microsecond - time taken for the packet within the switch
4. **Queue ID and Occupy**: 8 bit q ID + 24 bit q occupy while the packet being forwarded
5. **Ingress Time**: 64 bit, nanosecond - moment the packet shows up on ingress.
6. **Egress Time**: 64 bit, nanosecond - moment the packet starts egress processing. The clock in in-egress is set to 0 every time the switch starts*.
7. **L2 In-egress Ports**: 16 bit ingress + 16 bit egress Level 2 ports of the packet, not avail*
8. **TX Link Utilisation**: 32 bit - current utilisation rate of the egress port while the packet being starts egress processing, not avail*
9. **L4S Mark-Drop**: 16 bit nb marked + 16 bit nb dropped - total packets being marked/dropped since the last sent

➢ Other metrics to add: occupy of LL/classic queues, dominant IP of the packets in a queue, etc

* on BMv2 Simple Switch targets

Introduce P4-based INT into L4S

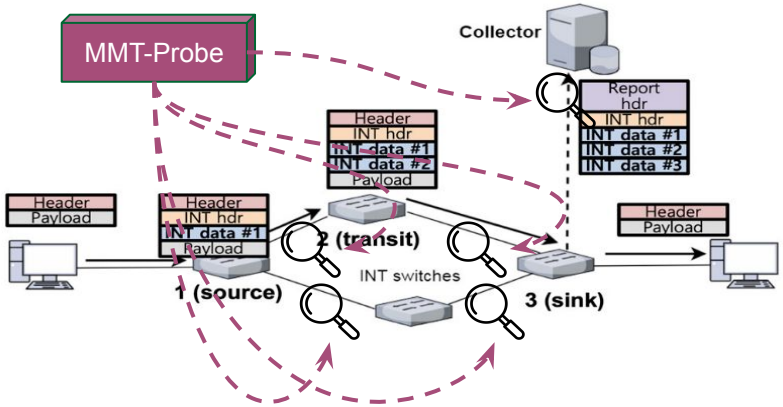
- P4-based INT, ~800 LoC P4, can be used as a "library" by appropriately calling its parser, control blocks in parser, ingress, egress and deparser blocks of the P4 program.
  - respect to INT v1.0
  - add 2 specific action for updating L4S metrics: int_l4s_drop and int_l4s_mark
- Configure INT nodes via control plane at runtime:
  - source node:
    - range of flows to be monitored via 4 tuples (src IP, src Port, dst IP, dst Port)
    - metrics to be collected at each node via instruction mask
    - max number of hops to be collected

  sink node:

    - egress port on which to remove INT headers
    - port to send INT report to
  - any node:
    - specify switch ID

Yet another INT collector: MMT-Probe

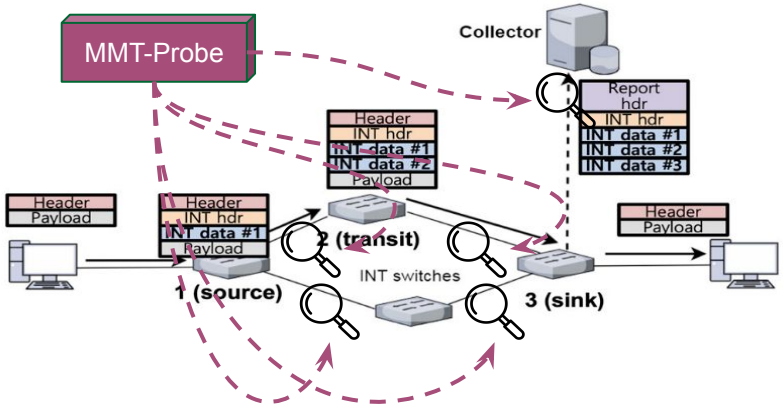
- Compatible with INT v1.0, can act as:
  - classic INT collector: analyse INT reports generated by sink nodes
  - inline INT collector: analyse INT metrics in INT packets in dataplane
- Selectable INT metrics to be analysed




```
# inband-network telemetry
event-report int {
   enable = true
   event  = "int.latency"
   delta-cond = {"int.latency", "int.hop_switch_ids"}
   attributes = {"ip.src", "ip.dst", "int.num_hop", "int.instruction_bits",
      "int.hop_switch_ids", "int.hop_latencies", "int.hop_queue_ids", "int.hop_queue_occups",
      "int.hop_ingress_times", "int.hop_egress_times", "int.hop_l4s_mark", "int.hop_l4s_drop"}
   output-channel = { redis , kafka , file, mongodb, socket }
}
```

Yet another INT collector: MMT-Probe

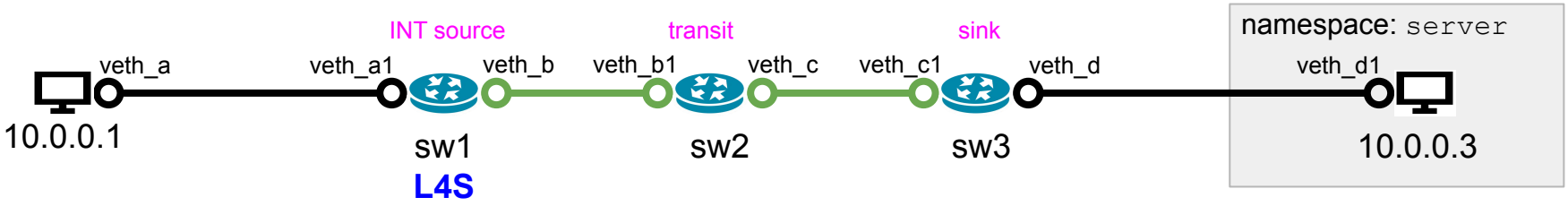
- Compatible with INT v1.0, can act as:
  - classic INT collector: analyse INT reports generated by sink nodes
  - inline INT collector: analyse INT metrics in INT packets in dataplane
- Selectable INT metrics to be analysed

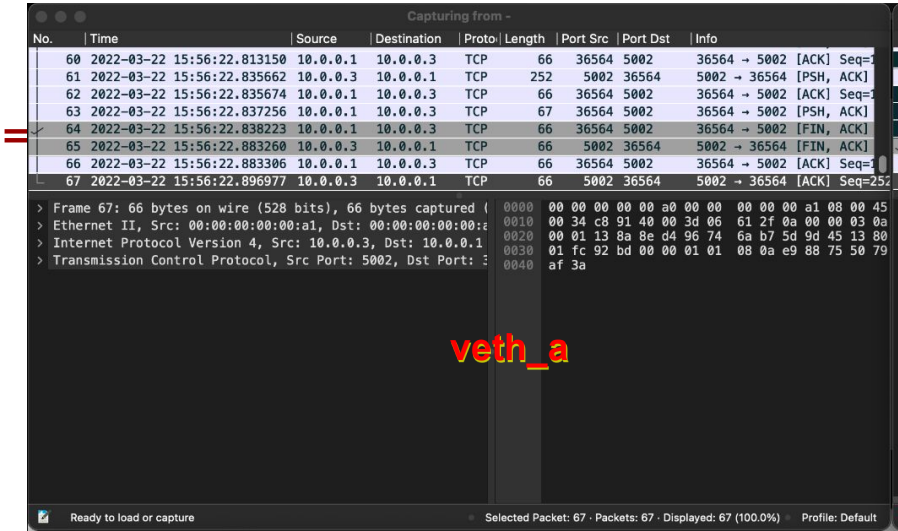


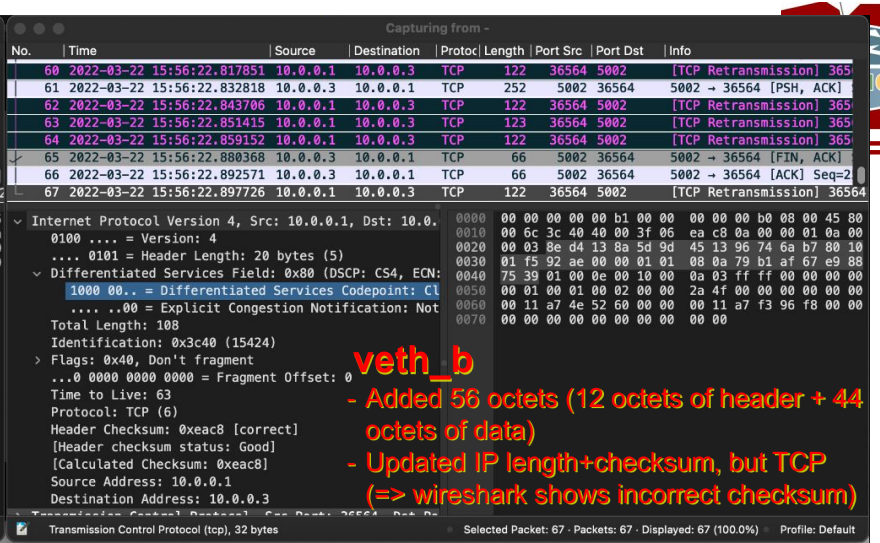

```
1000,3,"veth_c1_out.pcap",1647876773.690899,"int",3261,"10.0.0.1","10.0.0.3",2,65535,2,1,1074,2187,0,0,0,0,125752412000,125691057000,125753486000,125693244000,0,0,0,0
1000,3,"veth_c1_out.pcap",1647876774.087810,"int",2578,"10.0.0.1","10.0.0.3",2,65535,2,1,1461,1117,0,0,0,0,126148240000,126050763000,126149701000,126051880000,0,0,0,0
1000,3,"veth_c1_out.pcap",1647876774.444078,"int",5143,"10.0.0.1","10.0.0.3",2,65535,2,1,2570,2573,0,0,0,0,126503390000,126440914000,126505960000,126443487000,0,0,0,0
1000,3,"veth_c1_out.pcap",1647876774.786432,"int",2156,"10.0.0.1","10.0.0.3",2,65535,2,1,1070,1086,0,0,0,0,126847561000,126786903000,126848631000,126787989000,0,0,0,0
1000,3,"veth_c1_out.pcap",1647876775.140458,"int",2562,"10.0.0.1","10.0.0.3",2,65535,2,1,1064,1498,0,0,0,0,127201856000,127091423000,127202920000,127092921000,0,0,0,0
1000,3,"veth_c1_out.pcap",1647876775.239042,"int",3314,"10.0.0.1","10.0.0.3",2,65535,2,1,2126,1188,0,0,0,0,127298878000,127191980000,127301004000,127193168000,0,0,0,0
1000,3,"veth_c1_out.pcap",1647876775.291111,"int",53645,"10.0.0.1","10.0.0.3",2,65535,2,1,7026,46619,0,0,0,0,127303391000,127194745000,127310417000,127241364000,0,0,0,0
1000,3,"veth_c1_out.pcap",1647876775.344886,"int",5039,"10.0.0.1","10.0.0.3",2,65535,2,1,2145,2894,0,0,0,0,127405117000,127297106000,127407262000,127300000000,0,0,0,0
1000,3,"veth_c1_out.pcap",1647876775.393547,"int",45820,"10.0.0.1","10.0.0.3",2,65535,2,1,1063,44757,0,0,0,0,127454780000,127350945000,127455843000,127395702000,0,0,0,0
1000,3,"veth_c1_out.pcap",1647876775.443969,"int",52532,"10.0.0.1","10.0.0.3",2,65535,2,1,1699,50833,0,0,0,0,127504628000,127394342000,127506327000,127445175000,0,0,0,0
1000,3,"veth_c1_out.pcap",1647876775.592961,"int",3533,"10.0.0.1","10.0.0.3",2,65535,2,1,1725,1808,0,0,0,0,127653007000,127590934000,127654732000,127592742000,0,0,0,0
```

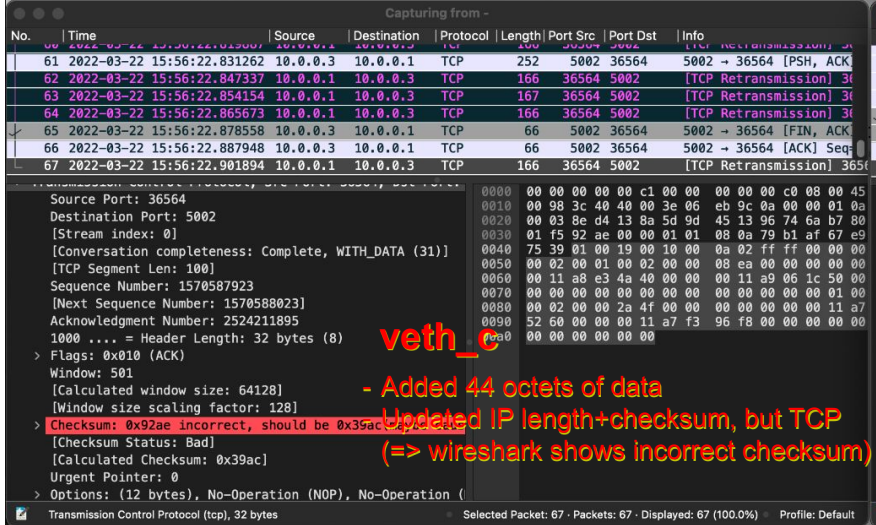
Demo

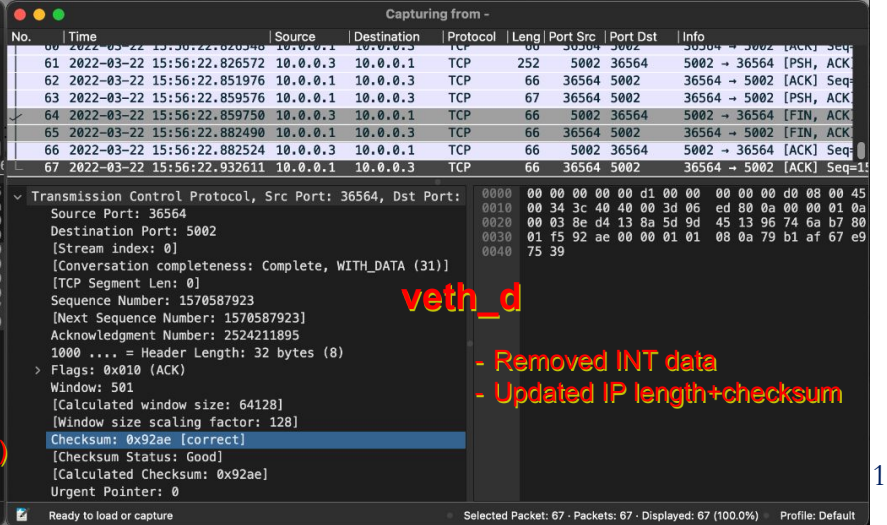
- A VM running Ubuntu 18.04.6 LTS
- 4 pair of virtual NICs type veth
- 3 BMv2 switches, L4S in the source node
- 2 simulated hosts at veth_a and veth_d1 that is in server namespace to avoid direct contact




Capturing from -
No. Time Source Destination Proto Length Port Src Port Dst Info
60 2022-03-22 15:56:22.813150 10.0.0.1 10.0.0.3 TCP 66 36564 5002 36564 → 5002 [ACK] Seq=1
61 2022-03-22 15:56:22.835662 10.0.0.3 10.0.0.1 TCP 252 5002 36564 5002 → 36564 [PSH, ACK]
62 2022-03-22 15:56:22.835674 10.0.0.1 10.0.0.3 TCP 66 36564 5002 36564 → 5002 [ACK] Seq=1
63 2022-03-22 15:56:22.837256 10.0.0.1 10.0.0.3 TCP 67 36564 5002 36564 → 5002 [PSH, ACK]
64 2022-03-22 15:56:22.838223 10.0.0.1 10.0.0.3 TCP 66 36564 5002 36564 → 5002 [FIN, ACK]
65 2022-03-22 15:56:22.883260 10.0.0.3 10.0.0.1 TCP 66 5002 36564 5002 → 36564 [FIN, ACK]
66 2022-03-22 15:56:22.883306 10.0.0.1 10.0.0.3 TCP 66 36564 5002 36564 → 5002 [ACK] Seq=1
67 2022-03-22 15:56:22.896977 10.0.0.3 10.0.0.1 TCP 66 5002 36564 5002 → 36564 [ACK] Seq=252
Frame 67: 66 bytes on wire (528 bits), 66 bytes captured (
Ethernet II, Src: 00:00:00:00:00:a1, Dst: 00:00:00:00:00:a
Internet Protocol Version 4, Src: 10.0.0.3, Dst: 10.0.0.1
Transmission Control Protocol, Src Port: 5002, Dst Port:
0000 00 00 00 00 00 a0 00 00 00 00 00 a1 08 00 45
0010 00 34 c8 91 40 00 3d 06 61 2f 0a 00 00 03 0a
0020 00 01 13 8a 8e d4 96 74 6a b7 5d 9d 45 13 80
0030 01 fc 92 bd 00 00 01 01 08 0a e9 88 75 50 79
0040 af 3a
veth_a
Ready to load or capture
Selected Packet: 67 · Packets: 67 · Displayed: 67 (100.0%)
Profile: Default


Capturing from -
No. Time Source Destination Protoc Length Port Src Port Dst Info
60 2022-03-22 15:56:22.817851 10.0.0.1 10.0.0.3 TCP 122 36564 5002 [TCP Retransmission] 365
61 2022-03-22 15:56:22.832818 10.0.0.3 10.0.0.1 TCP 252 5002 36564 5002 → 36564 [PSH, ACK]
62 2022-03-22 15:56:22.843706 10.0.0.1 10.0.0.3 TCP 122 36564 5002 [TCP Retransmission] 365
63 2022-03-22 15:56:22.851415 10.0.0.1 10.0.0.3 TCP 123 36564 5002 [TCP Retransmission] 365
64 2022-03-22 15:56:22.859152 10.0.0.1 10.0.0.3 TCP 122 36564 5002 [TCP Retransmission] 365
65 2022-03-22 15:56:22.880368 10.0.0.3 10.0.0.1 TCP 66 5002 36564 5002 → 36564 [FIN, ACK]
66 2022-03-22 15:56:22.892571 10.0.0.3 10.0.0.1 TCP 66 5002 36564 5002 → 36564 [ACK] Seq=2
67 2022-03-22 15:56:22.897726 10.0.0.1 10.0.0.3 TCP 122 36564 5002 [TCP Retransmission] 36564
Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x80 (DSCP: CS4, ECN:
1000 00.. = Differentiated Services Codepoint: Cl
.... ..00 = Explicit Congestion Notification: Not
Total Length: 108
Identification: 0x3c40 (15424)
Flags: 0x40, Don't fragment
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 63
Protocol: TCP (6)
Header Checksum: 0xeac8 [correct]
[Header checksum status: Good]
[Calculated Checksum: 0xeac8]
Source Address: 10.0.0.1
Destination Address: 10.0.0.3
0000 00 00 00 00 00 b1 00 00 00 00 00 b0 08 00 45 80
0010 00 6c 3c 40 40 00 3f 06 ea c8 0a 00 00 01 0a 00
0020 00 03 8e d4 13 8a 5d 9d 45 13 96 74 6a b7 80 10
0030 01 f5 92 ae 00 00 01 01 08 0a 79 b1 af 67 e9 88
0040 75 39 01 00 0e 00 10 00 0a 03 ff ff 00 00 00 00
0050 00 01 00 01 00 02 00 00 2a 4f 00 00 00 00 00 00
0060 00 11 a7 4e 52 60 00 00 00 11 a7 f3 96 f8 00 00
0070 00 00 00 00 00 00 00 00 00 00
veth_b
- Added 56 octets (12 octets of header + 44 octets of data)
- Updated IP length+checksum, but TCP (=> wireshark shows incorrect checksum)
Transmission Control Protocol (tcp), 32 bytes
Selected Packet: 67 · Packets: 67 · Displayed: 67 (100.0%)
Profile: Default


Capturing from -
No. Time Source Destination Protocol Length Port Src Port Dst Info
61 2022-03-22 15:56:22.831262 10.0.0.3 10.0.0.1 TCP 252 5002 36564 5002 → 36564 [PSH, ACK]
62 2022-03-22 15:56:22.847337 10.0.0.1 10.0.0.3 TCP 166 36564 5002 [TCP Retransmission]
63 2022-03-22 15:56:22.854154 10.0.0.1 10.0.0.3 TCP 167 36564 5002 [TCP Retransmission]
64 2022-03-22 15:56:22.865673 10.0.0.1 10.0.0.3 TCP 166 36564 5002 [TCP Retransmission]
65 2022-03-22 15:56:22.878558 10.0.0.3 10.0.0.1 TCP 66 5002 36564 5002 → 36564 [FIN, ACK]
66 2022-03-22 15:56:22.887948 10.0.0.3 10.0.0.1 TCP 66 5002 36564 5002 → 36564 [ACK] Seq=
67 2022-03-22 15:56:22.901894 10.0.0.1 10.0.0.3 TCP 166 36564 5002 [TCP Retransmission]
Source Port: 36564
Destination Port: 5002
[Stream index: 0]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 100]
Sequence Number: 1570587923
[Next Sequence Number: 1570588023]
Acknowledgment Number: 2524211895
1000 .... = Header Length: 32 bytes (8)
Flags: 0x010 (ACK)
Window: 501
[Calculated window size: 64128]
[Window size scaling factor: 128]
Checksum: 0x92ae incorrect, should be 0x39ac
[Checksum Status: Bad]
[Calculated Checksum: 0x39ac]
Urgent Pointer: 0
Options: (12 bytes), No-Operation (NOP), No-Operation (
0000 00 00 00 00 00 c1 00 00 00 00 00 c0 08 00 45
0010 00 98 3c 40 40 00 3e 06 eb 9c 0a 00 00 01 0a
0020 00 03 8e d4 13 8a 5d 9d 45 13 96 74 6a b7 80
0030 01 f5 92 ae 00 00 01 01 08 0a 79 b1 af 67 e9
0040 75 39 01 00 19 00 10 00 0a 02 ff ff 00 00 00
0050 00 02 00 01 00 02 00 00 08 ea 00 00 00 00 00
0060 00 11 a8 e3 4a 40 00 00 00 11 a9 06 1c 50 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 01 00
0080 00 02 00 00 2a 4f 00 00 00 00 00 00 00 11 a7
0090 52 60 00 00 00 11 a7 f3 96 f8 00 00 00 00 00
00a0 00 00 00 00 00 00
veth_c
- Added 44 octets of data
- Updated IP length+checksum, but TCP (=> wireshark shows incorrect checksum)
Transmission Control Protocol (tcp), 32 bytes
Selected Packet: 67 · Packets: 67 · Displayed: 67 (100.0%)
Profile: Default


Capturing from -
No. Time Source Destination Protocol Leng Port Src Port Dst Info
61 2022-03-22 15:56:22.826572 10.0.0.3 10.0.0.1 TCP 252 5002 36564 5002 → 36564 [PSH, ACK]
62 2022-03-22 15:56:22.851976 10.0.0.1 10.0.0.3 TCP 66 36564 5002 36564 → 5002 [ACK] Seq=
63 2022-03-22 15:56:22.859576 10.0.0.1 10.0.0.3 TCP 67 36564 5002 36564 → 5002 [PSH, ACK]
64 2022-03-22 15:56:22.859750 10.0.0.3 10.0.0.1 TCP 66 5002 36564 5002 → 36564 [FIN, ACK]
65 2022-03-22 15:56:22.882490 10.0.0.1 10.0.0.3 TCP 66 36564 5002 36564 → 5002 [FIN, ACK]
66 2022-03-22 15:56:22.882524 10.0.0.3 10.0.0.1 TCP 66 5002 36564 5002 → 36564 [ACK] Seq=
67 2022-03-22 15:56:22.932611 10.0.0.1 10.0.0.3 TCP 66 36564 5002 36564 → 5002 [ACK] Seq=1
Transmission Control Protocol, Src Port: 36564, Dst Port:
Source Port: 36564
Destination Port: 5002
[Stream index: 0]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 0]
Sequence Number: 1570587923
[Next Sequence Number: 1570587923]
Acknowledgment Number: 2524211895
1000 .... = Header Length: 32 bytes (8)
Flags: 0x010 (ACK)
Window: 501
[Calculated window size: 64128]
[Window size scaling factor: 128]
Checksum: 0x92ae [correct]
[Checksum Status: Good]
[Calculated Checksum: 0x92ae]
Urgent Pointer: 0
0000 00 00 00 00 00 d1 00 00 00 00 00 d0 08 00 45
0010 00 34 3c 40 40 00 3d 06 ed 80 0a 00 00 01 0a
0020 00 03 8e d4 13 8a 5d 9d 45 13 96 74 6a b7 80
0030 01 f5 92 ae 00 00 01 01 08 0a 79 b1 af 67 e9
0040 75 39
veth_d
- Removed INT data
- Updated IP length+checksum
Ready to load or capture
Selected Packet: 67 · Packets: 67 · Displayed: 67 (100.0%)
Profile: Default

Preliminary Test

- 8 standard metrics' values conform to the tests
- Issues:
  - hop_l4s_drop is less than nb real dropped packets => possibly BMv2 (or sender's NIC?) drops packets before they are able to be processed L4S.
    - => need a metric for nb dropped packets by hardware?
  - how to profile LL/classic queues in P4 L4S? do we need to modify BMv2 program?

Conclusion

- Conclusion
  - a P4 "library" to do INT that can be introduced into L4S P4 program
    - selectable metrics to be collected via Instruction Mask at runtime
  - an INT collector to analyse INT reports or even INT packets at data plane
    - selectable metrics to be analysed
- Future works:
  - Use packets to carry stat info => loss stat info when drop packets => need to generate INT reports on a specific port in high-precision monitoring profile
  - Test more INT+L4S and MMT ...
  - Implement other metrics