Using In-band Network Telemetry(INT) for Traffic Engineering
RnD Project Report

Vikas Kumar (Roll No. 163059007)

Under the guidance of
Prof Mythili Vutukuru

April 2018

Contents

1 Abstract

One of the use case of programmable switches is INT which means to collect the statistics from the switches and send to a centralized processing node. This node can analyze the statistics collected and can be used to do congestion aware re-routing and Traffic engineering and helps in getting deep insights into network. As part of this RnD project I have explored how INT can be used for Traffic engineering to dynamically re-route flows on the bottleneck switch. The results are very promising that with INT module enabled we are able to achieve 50% more throughput and 93% reduction in the average latency when a packet traverses from host to destination over a network topology consisting of 8 switches and 2 end hosts.

2 Introduction

In-band Network Telemetry is an abstraction which is used to collect statistics from switches to get insights into the network. INT can be used to collect switch internal states like queue size, hop latency and link utilization. These statistics collected from the switch can be later used to answer various questions related to packet which were previously not possible or limited in the fixed function switches.

1. How much time did the packet spent at each switch?
2. Where the packet faced huge hop latency and what was the reason for the large hop latency at that particular switch?
3. What was the queue size at each egress port of switch when the packet was forwarded through the switches?

3 IN-BAND NETWORK TELEMETRY

With the advent of high-performance forwarding switching chips like Barefoot Network's Tofino [3] and Cavium's Xpliant Ethernet switch family [10] which promise multi-Tb/s of packet processing, programmers can now define how the packets are to be processed all the way down to the wire. Programmers can now define the forwarding behaviour of the switches in a high level domain specific languages like P4 [1] and Domino [9] as per the need of the hour. P4 is target-independent and provides in field reconfigurability. In P4, the programmer declares how packets are to be processed, and a compiler generates a configuration for a protocol-independent switch chip or NIC. For example, the programmer might program the switch to be a top-of-rack switch, a firewall, or a load-balancer, and might add features to run automatic diagnostics and novel congestion-control algorithms. With INT, a network owner will be able to debug and root-cause various network incidents rapidly and intuitively.

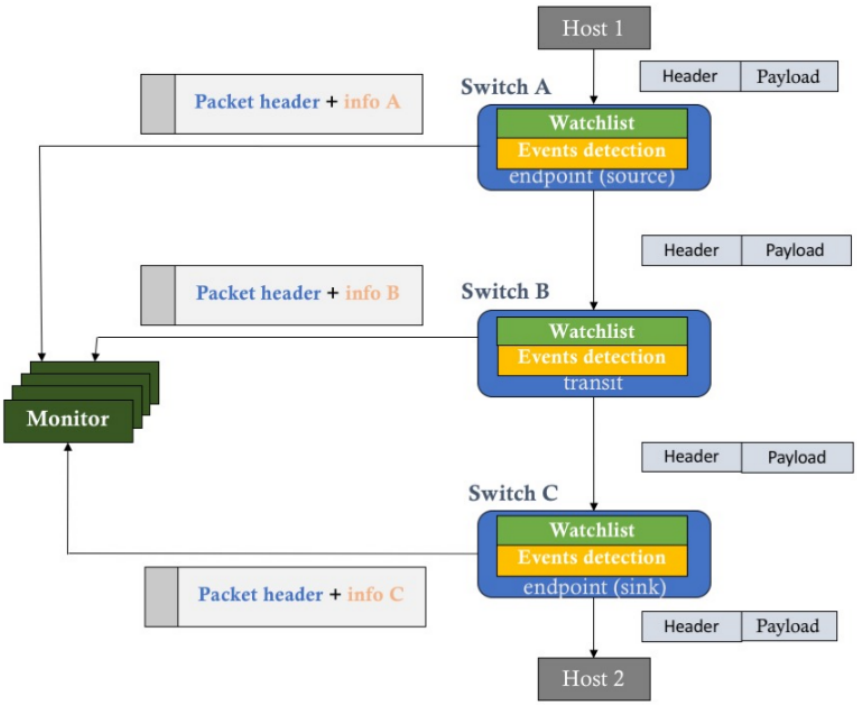

Figure 1: Postcard Telemetry mode[7]

P4 language consortium(p4.org) has formed an Applications Working Group [8] which provides specifications for the telemetry application. As per the telemetry report format specification [7], we can have two telemetry modes as specified in the next section.

3.1 Telemetry Modes

Depending on the location from where the telemetry data is generated and how it is collected by the telemetry monitoring system we have two telemetry modes postcard mode and In-band telemetry mode.

3.1.1 Postcard Mode

In postcard mode, the telemetry data report is generated by the switches individually as shown in Figure 1. The distributed telemetry monitoring system collects the telemetry meta-data (such as switch ID, hop latency, queue depth) from all switches and processes it. The meta-data header precedes the original packet header in this mode and the original data packet traversing the packet remains unaffected.

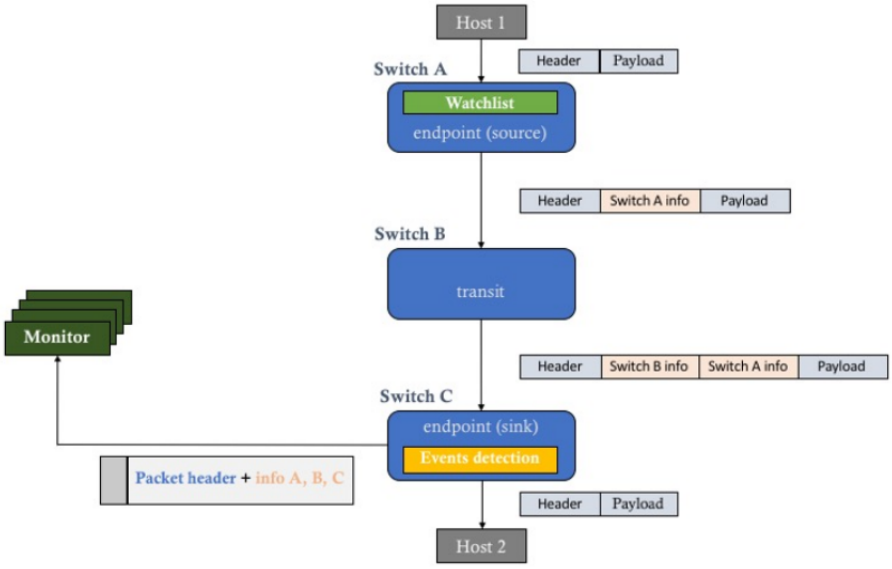

Figure 2: In-band Network Telemetry [7]

3.1.2 In-band Network Telemetry(INT)

In INT mode, the telemetry meta-data collected from the switches is appended to the original packet header between the Layer4 header and the application data as shown in Figure 2. The original packet header length increases as the packet traverses the next hops. As shown in Figure 2, when the packet enters the network the INT source hop (Switch A) inserts the telemetry instruction header which instructs the downstream switches to insert the desired telemetry meta-data. Each hop on the network path (transit switch) inserts its own telemetry meta-data and finally the sink switch (Switch C) clones the packet and sends the cloned packet to the telemetry monitoring system. The original packet is then forwarded to the destination after removing the telemetry meta-data headers.

3.2 INT Headers

There are two types of INT headers hop-by-hop headers which are processed by all the intermediate hops in the packets data path and INT destination headers which are processed only by the sink hop as described in the INT dataplane specification [6] . In the next part hop-by-hop are described in more detail as I have used them in this project.

3.2.1 INT Hop-by-Hop Meta-data Header Format

INT hop-by-hop meta-data headers is defined below as shown in Figure 3.

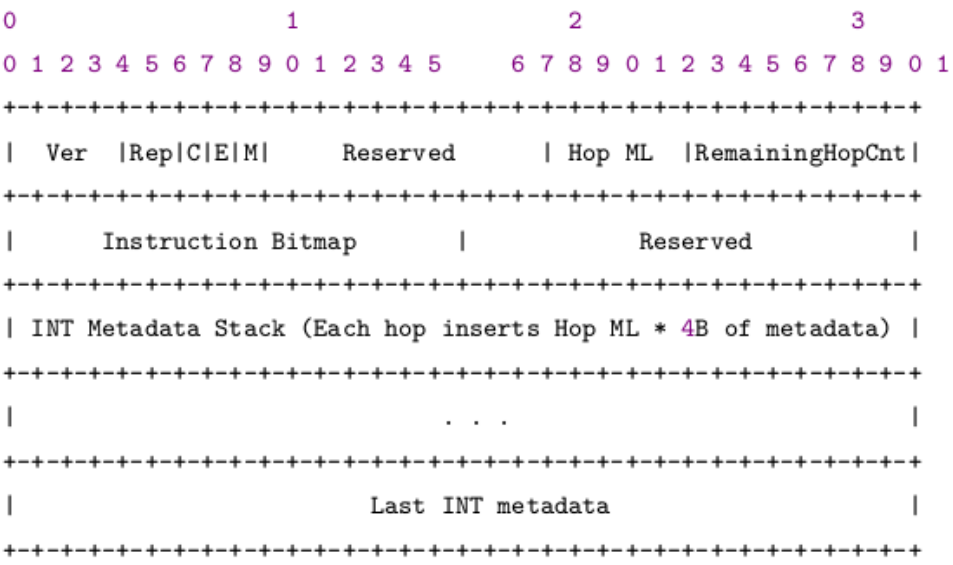

Figure 3: INT Meta-data header and Meta-data Stack [6]

- INT meta-data header is 8 bytes long followed by a stack of INT meta-data.The total length of the meta-data stack is variable as different packets may traverse different paths and hence different number of INT hops.
- INT meta-data headers fields are described below
  - Ver (4b): INT meta-data header version. Should be 1 for this version.
  - Rep (2b): Replication requested. Support for this request is optional. If this value is non-zero, the device may replicate the INT packet.
  - C (1b): Copy.
  - E (1b): Max Hop Count exceeded.
    - This flag must be set if a device cannot prepend its own meta-data due to the Remaining Hop Count reaching zero.
    - E bit must be set to 0 by INT source
  - M (1b): MTU exceeded
    - This flag must be set if a device cannot add all of the requested meta-data because doing so will cause the packet length to exceed egress link MTU.
  - R: Reserved bits. These must be set to 0 by the INT source hop.
  - Remaining Hop Count (8b): The remaining number of hops that are allowed to add their meta-data to the packet. This is set by INT source.

        * When a packet is received with the Remaining Hop Count equal to 0, the device must ignore the INT instruction, pushing no new meta-data onto the stack, and the device must set the E bit.
    - INT instructions are encoded as a bitmap in the 16-bit INT Instruction field.
        * bit0 (MSB): Switch ID
        * bit1: Ingress Port ID (16 bits) + Egress Port ID (16 bits)
        * bit2: Hop latency
        * bit3: Queue ID (8 bits) + Queue occupancy (24 bits)
        * bit4: Ingress timestamp
        * bit5: Egress timestamp
        * bit7: Egress port Tx utilization
        * The remaining bits are reserved.
    - INT transit switch adds its own meta-data values as specified in the instruction bitmap immediately after the INT meta-data header.

- Intermediate devices can set the following fields:
    - C, E, M, Remaining Hop Count

- The INT Source must set the following fields:
    - Ver, Rep, C, M, Per-hop meta-data Length, Remaining Hop Count, and Instruction Bitmap.
    - INT Source must set all reserved bits to zero.

4 Problem Statement

Given a network topology of n nodes connected to each other linearly and with m bottleneck switches such that m¡n, identify and dynamically re-route the flows causing congestion at bottleneck switches.

5 Design and implementation of INT

I have implemented INT on a specific network topology of 8 switches and 2 end hosts connected to each other in linear topology as shown in Figure 4. The network topology is emulated using Mininet [2] and the INT monitoring engine runs on the base system in which the Mininet is running. In this topology the Switch 1 is configured as INT source and Switch2- Switch7 are configured as INT transit and Switch8 is INT sink which send the cloned INT meta-data packet to the INT monitoring system. The three types of switches are burned with different p4 programs to specify the behaviour of how to process the incoming packet. A sample reference code is given in [6] and I have built on top of it.

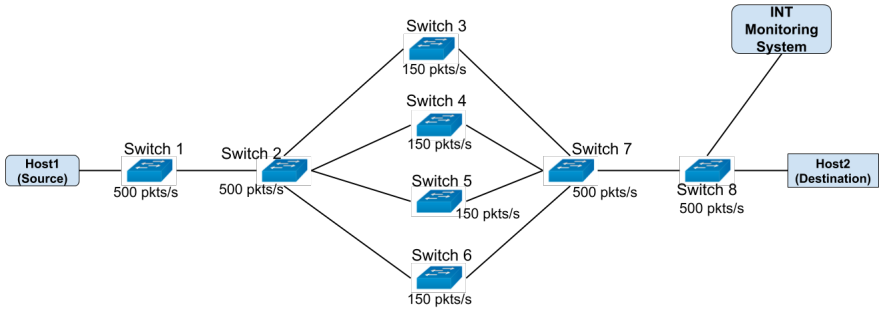

Figure 4: INT network topology

The packets traverse the network from Host1 connected with Switch1 to Host2. when the UDP packet from Host1 enters the incoming interface of Switch1, it adds the INT meta-data header and initializes the instruction bitmap which tells the INT transit switches what meta-data to append to the INT headerstack. When the packet comes to the INT transit switch say Switch2 then it parses the INT packet differently by identifying it from DSCP bit set by the Switch1 as shown in Figure 5. If the DSCP bit matches 1 then it means it is an INT packet so it adds its own meta-data to the INT meta-data stack otherwise it processes the packet normally by forwarding it to one of the egress ports.When the INT packet reaches Switch8 then it clones the packet and sends the cloned copy of packet to the INT monitoring engine and forwards the original UDP packet to Host2 after removing the INT meta-data headers which were appended by the other switches on the network path.

INT module Switch3, Switch4, Switch5 and Switch6 are rate limited as compared to other switches in the network topology shown in Figure 4. Initially the flow from Host1 to Host2 is configured to take the path Switch1 - Switch2 - Switch3 - switch7 - Switch8. The INT module runs on the INT monitoring system which collects the statistics collected from switches (such as queue depth, hop latency, switch ID) and identifies the flow causing congestion at Switch3 since it is rate limited and initially all flows pass through it. When the packet passes through the switches then at Switch1 we calculate the 4-tuple hash and send it as part of one of the header fields of INT meta-data. This can be done even at controller but since real switches can calculate hashes much faster so we do it in Switch1 itself. The INT module then samples every 30 packets and if the same hash is repeated in 1/4th of the packets and the avg packet size of the sampled packet with same hash is more than 1000B then we have identified the flow causing congestion. Then we identify the corresponding entry for the flow in Switch2 control plane and modify it to re-route the flow between other

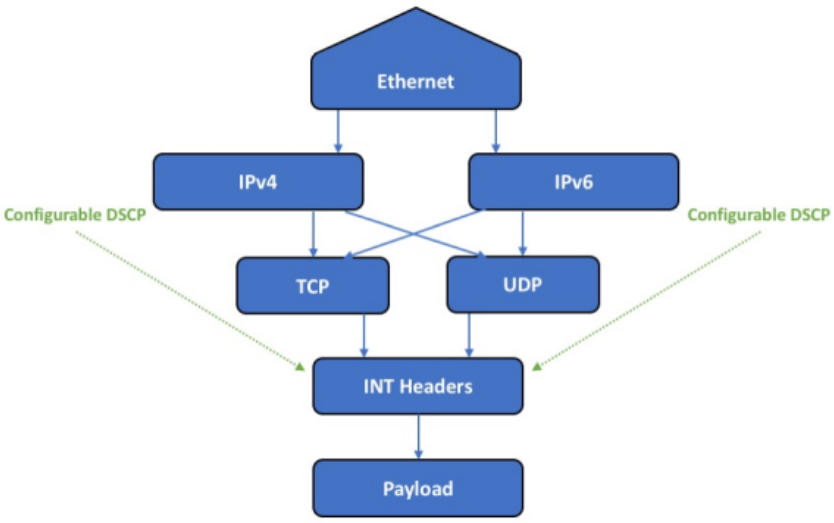

Figure 5: INT Packet Parser Flowchart [7]

links Switch2-Switch4, Switch2-Switch5, switch2-Switch6 using round robin algorithm for Traffic engineering at Switch3.

6 Experimental Setup

The experimental setup has 1 Virtual Machine with the below configurtation

- Processor : 4 VCPUs
- RAM : 4 GB
- OS : Ubuntu 16.04 LTS

The VM has a mininet based setup to emulate the network topology of Figure 4. The VM itself acts as INT monitoring system attached to one of the interfaces of Switch8 which collects the INT meta-data packets from it. All the switches run simple_switch program [5] which is a software switch based on bmv2 Behavioural Model [4]. The behaviour of the switches is specified in a high level domian specific language [1] and then compiled and burnt to the target switches Switch1 - Switch8.

The queue rate of Switch3, Switch4, Switch5, Switch6 is configured as 1/4th of other switches. Initially, every incoming flow at Switch2 from Host1 is forwarded to Switch3. This means that there will be a large queue built up on Switch3. Hence the packets stop dropping and throughput decreases but using

the INT meta-data collected at INT monitoring engine we can identify and re-route the long flows to other switches (Switch4, Switch5, Switch6) from Switch2 using the contol plane of Switch2.

7 Evaluation

For evaluation I have generated the UDP short and long flows at Host1 with destination of Host2. The short flow and long flow are also called as mouse flows and elephant flows in this report. By varying the percentage mix of short and long flows, I have checked if the flows are correctly re-routed or not based on the queue depth observed at the various switches. The experiments performed are elaborated in the below sections. The INT module here refers to the Traffic engineering done by the INT monitoring system to re-route the large flows dynamically at Switch2. We have two types of flows in the network traffic going from Host1 to Host2 long flows and short flows. Long flows last for 10 seconds and contains 1370 Bytes of payload data in the packet. Short flows last for 0.1 seconds and contains 10 Bytes of payload data in the packet.The mix of traffic generated by Host1 is as mentioned below.

- Case A : 50-50 % Elephant and Mouse flow without INT module
- Case B : 50-50 % Elephant and Mouse flow with INT module
- Case C : 75-25 % Elephant and Mouse flow without INT module
- Case D : 75-25 % Elephant and Mouse flow with INT module

For Case A and Case B, the queue depth at all switches is set to 1000 and the queue rate at Switch1, Switch2, Switch7, Switch8 is configured to be 500 packets/sec and that for Switch3, Switch4, Switch5, Switch6 is configured to process 150 packets/sec. Similarly for Case C and Case D, the queue depth at all switches is set to 1000 and the queue rate at Switch1, Switch2, Switch7, Switch8 is configured to be 500 packets/sec and that for Switch3, Switch4, Switch5, Switch6 is configured to process 120 packets/sec.

7.1 Throughput with and without INT module

The throughput is defined as the number of UDP packets received per second at Host2. The Graph in Figure 6 compares the throughput for Case A and Case B where Case A and Case B refers to the traffic flow when the ratio of elephant to mouse flow is 50:50 and 75:25 respectively. As it is evident from the graph that the throughout increased by 50% when INT module is running.

7.2 Average Latency with and without INT module

Average latency is defined as the the total time taken by a packet to traverse though the switches in case of both short as well as long flows. The average

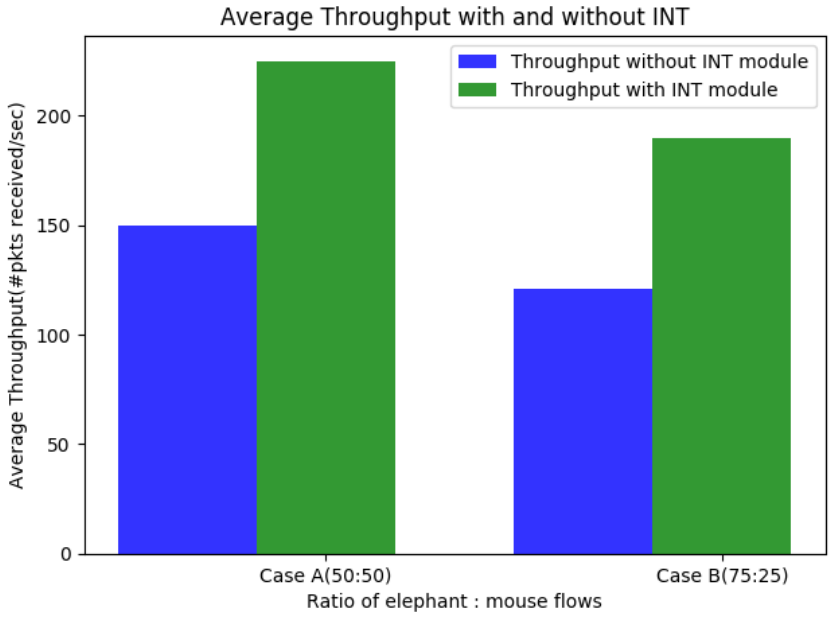

Figure 6: Average Throughput for Mouse and Elephant flow.

latency for long flow is calculated as sum of all hop latencies for all packets of long flow divided by number of long flow packets. As shown in Figure 7 the average latency is reduced by 93% when the INT module is enabled as compared to the case when the switches are rate limited but INT module is not enabled.

7.3 Queue Depth at Bottleneck Switch with and without INT module

Since Switch3 is rate limited the queue builds up on this switch and the throughput is less when the INT module is not enabled because the packets stats dropping once the queue is full.

As it is observed from Figure 8 without INT module enabled the queue depth reaches to maximum 1000 in Switch3 but when the INT module is enabled in Figure 9 the queue depth is under control at Switch3 and never crosses beyond 450 as the large flows are diverted to other switches using INT module. Similar results are found even when the traffic ratio changes to 75:35 for elephant:mouse flows respectively as shown in Figure 10 and 11.

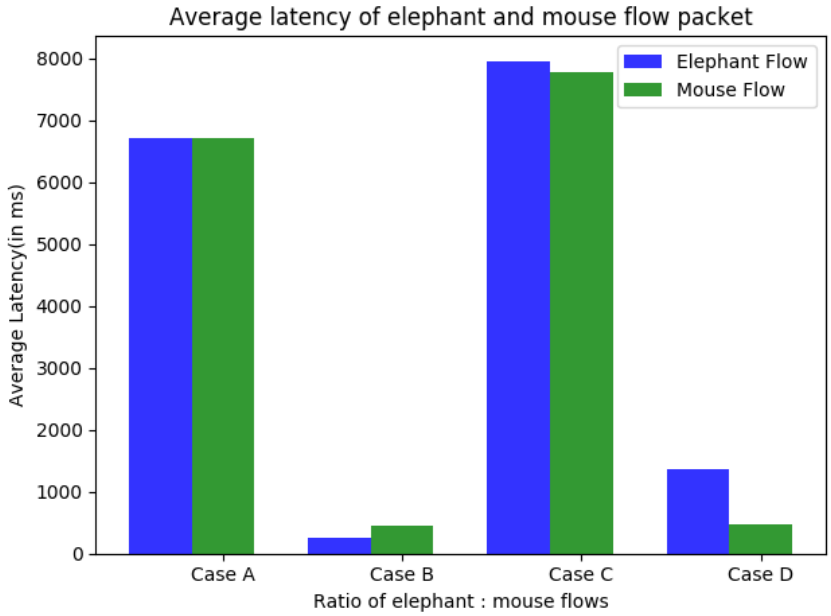

Figure 7: Average packet latency for Mouse and Elephant flow.

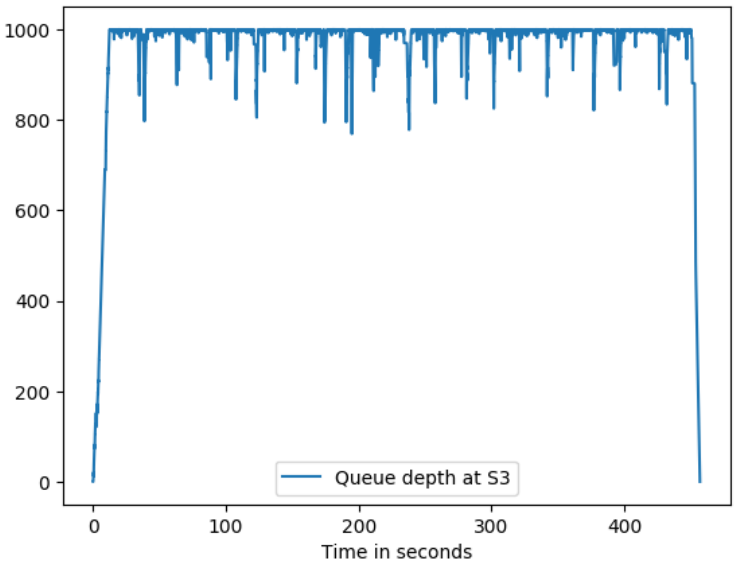

Figure 8: Queue depth at switch3 without INT in Case A

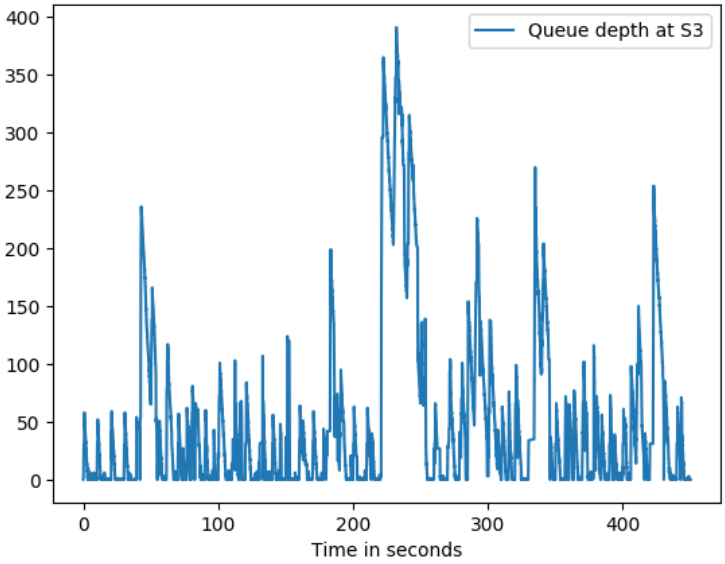

Figure 9: Queue depth at switch3 with INT in Case B

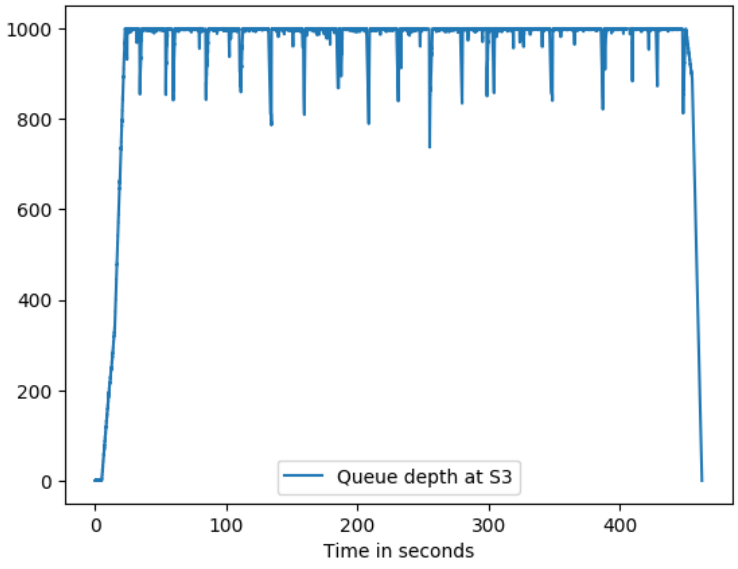

Figure 10: Queue depth at switch3 without INT in Case C

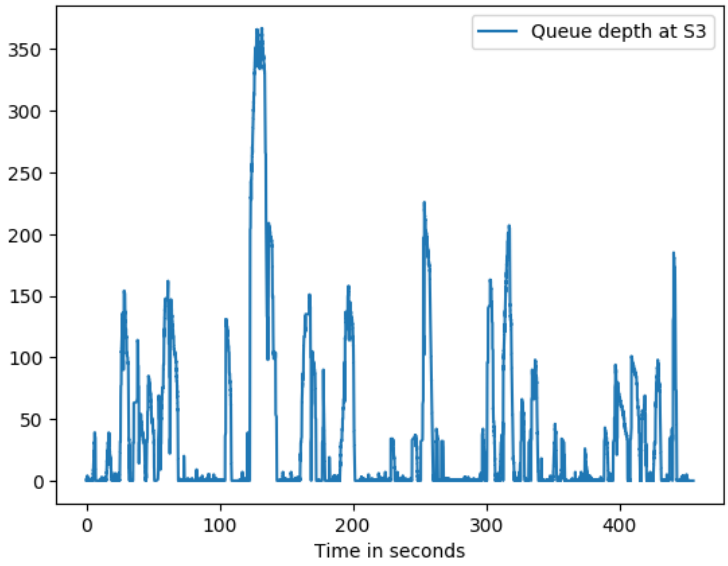

Figure 11: Queue depth at switch3 with INT in Case D

8 Conclusion

I have developed an INT module in P4 which collects switch statistics from the data path of the packet as part of this RnD project which helps in Traffic Engineering to identify re-route the elephant flows dynamically to control congestion at the bottleneck switch. The results shows that the large elephant flows are re-routed correctly with a 93% less latency and 50% more throughput when the INT module is enabled. The INT module is able to handle skewed workload as experimented by varying the ratio of short as well as long flows.

9 Future Work

Since this work is very specific to the given network topology as part of future work we can work to develop an more generalized INT solution. Also as the INT meta-data collected from switches grows linearly with the number of switches in the data path of the packet due to which the MTU size exceeds and other switches cannot add more INT meta-data to packet beyond a point so we can work to solve this problem. Implementing INT over TCP is an another work which can be done as part of future work.

References

[1] Pat Bosshart, Dan Daly, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. Programming Protocol-Independent Packet Processors. 2013.

[2] Bob Lantz, Brandon Heller, and N McKeown. A network in a laptop: rapid prototyping for software-defined networks. *... Workshop on Hot Topics in Networks*, pages 1–6, 2010.

[3] Barefoot Networks. Toffino switch. https://www.barefootnetworks.com/technology/.

[4] p4lang github. Behavioural Model. https://github.com/p4lang/behavioral-model.

[5] p4lang github. Behavioural Model Target $simple_switch.https://github.com/p4lang/behavioral-model/tree/master/targets/simple_switch$.

[6] p4.org Applications Working Group. INT Dataplane Specification. https://github.com/p4lang/p4-applications/blob/master/docs/INT.pdf.

[7] p4.org Applications Working Group. Telemetry Report Format Specification. https://github.com/p4lang/p4-applications/blob/master/docs/$telemetry_report.pdf$.

[8] p4.org Working Group. p4-applications. https://github.com/p4lang/p4-applications.

[9] Anirudh Sivaraman, Mihai Budiu, Alvin Cheung, Changhoon Kim, Steve Licking, George Varghese, Hari Balakrishnan, Mohammad Alizadeh, and Nick McKeown. Packet Transactions: High-level Programming for Line-Rate Switches. 2015.

[10] Cavium Ethernet switch Family. Xpliant switch. https://cavium.com/xpliant-ethernet-switch-product-family.html.