CSC/ECE 573 INTERNET PROTOCOLS

Project Report

In-band Network State Monitoring using P4

Submitted On: 3-Dec-2018

Submitted By: Team No - 1

Pranjal Sharma (psharma9@ncsu.edu)

Prithvi Sharan (psharan@ncsu.edu)

Sai Jayesh Bondu (sjbondu@ncsu.edu)

Luv Khurana (lkhuran@ncsu.edu)

Work Distribution

Component	Component Weightage	Pranjal	Prithvi	Sai Jayesh	Luv
Background research	0.1	25	25	25	25
High-level design	0.2	25	25	25	25
Algorithm development	0.2	25	25	25	25
Coding and debugging	0.4	25	25	25	25
Report writing	0.1	25	25	25	25
Per student aggregate contribution		25	25	25	25

INTRODUCTION

In this project, we have implemented In-Band Network State Monitoring (INSM), which is a scalable and efficient network monitoring solution in which crucial network state gets embedded into a data packet as it transits network elements. This was achieved by programming the data plane of the network using P4 language.

The popular Software-Defined Networking paradigm involves decoupling of control plane from the data plane of a device. It involves implementing the control plane in a logically centralized controller. The controller has complete view of the network topology and can, therefore, make smarter decisions based on this network-wide information. This has entailed an explosion of innovation in the control plane. Innovation in the data plane, however, has been slow to come, rendering the data plane largely rigid in terms of its capabilities. Today, with modern and dynamic environments such as virtualized data centers and cloud, this can prove to be a significant bottleneck. In other words, it is possible to do much better when it comes to the flexibility of data plane. To address this, a high-level language called P4 has been developed by some of the pioneers of the SDN movement. [1]

P4 is an open source programming language that lets end users dictate how networking devices operates by programming network forwarding planes. It controls silicon processor chips in network forwarding devices such as switches, routers and network interface cards. Whereas currently network functions are built "bottom-up" with fixed-function switches having one way of forwarding packets, programmable networks can be controlled "top-down" to install any functionality the user wants.

INSM is one of the many applications of P4 and data plane programming. It is a network monitoring solution that provides deep insights into the current network state. In INSM, the source (end-point) adds instructions in the packet listing the type of network state to be collected from the network elements. Network elements then add the requested network state as packet traverses the network. A program written in 'P4' is used to express the kind of packet header parsing and modifications required for INSM.

With INSM, one can track the path taken by a packet (sequence of switches traversed) as well as the queue depth at each switch. To achieve this, the program written in P4 appends the Device ID and the queue length to the IPv4 options part of each packets header. At the destination, the sequence of Switch IDs and the queue depths can be retrieved for identifying potential bottlenecks.

While INSM can be effectively used for real-time troubleshooting of complex data center networks, it is capable of much more. The collected granular network state can be used to drive intelligent decisions in the network (self-healing networks) such as for dynamic link utilization and automatic fast reroute. There exist specifications taking this approach to network monitoring such as INT from P4 Language Consortium [2]. Furthermore, Barefoot Networks has even gone on to develop an enterprise-grade solution called 'Deep Insight' that uses metadata collected using INT to offer Machine Learning based Analytics [3]

DESIGN

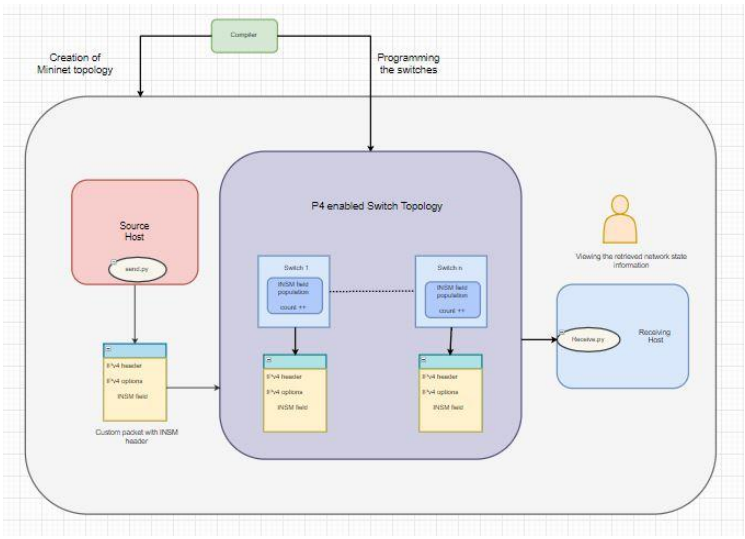

Figure: INSM Flow Chart

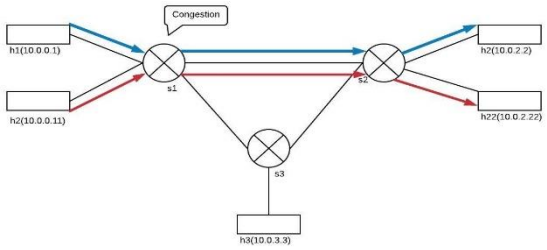

Figure: Mininet Topology

System overview

1. The link between S1 and S2 has been hard coded to a bandwidth of 512 Kbps to ensure that effects of contention is observable.
2. We demonstrate path taken by a packet by appending the switch ID of every switch that the packet traverses.
3. We send custom traffic from h1 (send.py) to h2 (receive.py).
4. We send iperf traffic from h11 to h22 at a simulate contention for link utilization.
5. All the tests were run on mininet topologies comprising of BMv2 switches. BMv2 [4] is the name of a P4-capable software based switch. The target architecture for BMv2 switches is called V1Model.
6. P4 codes specify how a packet is to be processed in the pipeline by rules that are pre defined by the control plane, when a packet matches a P4 rule, the control plane supplies with some parameters (essentially packet headers), that P4 can now modify.
7. The control plane has been implemented using static rules, that are pushed on every switch using Python.
8. The system appends network state information on customized header fields that are sent as a part of the user generated traffic program (send.py).

Figure: V1Model Processing Pipeline

Parser : Every packet arriving would enter the pipeline at this block. Programmers declare headers that should be recognized and their positioning in the packet.

Checksum Verification - These are used to check the validity of the packet (via checksum).

Ingress Match Action - Packet fields that are to be operated on before the decision of exit port is made are worked on here.

Egress Match Action - Packet fields that are to be operated on after the decision of exit port is made are worked on here.

Checksum Update - Responsible for calculation of checksum after header modification.

Deparser - Defines how the packet is serialized on the wire.

Runtime information

1. Any text can be passed as an argument in send.py. It simply travels as an L5 payload.
2. Timer values can also be varied within reasonable limits (any value under 60 secs is fine).
3. At h22, the sequence of switch ID's represent the path, which is then ensued by the queue length at every switch.

IMPLEMENTATION

Data plane of the switches was programmed using P4. P4 is a domain-specific programming language that is target independent, protocol independent and makes the switches field reconfigurable.
All the other components of the INSM application, like topology setup, P4 code deployment on P4-capable switches, control plane setup, were written in Python.

INSM code is segregated into core and utils directories:

INSM-P4/src/core/ - Contains the P4 code, INSM sender and receiver code, topology json, text files with control plane rules

INSM-P4/src/utils/ - Contains the auxiliary code for enabling P4 programmability. Most of this code has been adapted from Open Source community's work on P4. [5]

Description of important source-code files:
1. INSM-P4/src/core/insm.p4 - Contains the P4 code to program the data plane of the switches. This code contains logic for implementing basic IPv4 forwarding, as well as INSM-specific operations.
2. INSM-P4/src/core/send.py - Contains the logic to send datagrams to a given host. These datagrams have placeholders for carrying INSM-defined network state.
3. INSM-P4/src/core/receive.py - Contains the logic to listen for and display datagrams sent by INSM sender. Embedded network state can be gleaned from these datagrams.
4. INSM-P4/src/core/Makefile - Contains rules to build, run, stop, clean the application. Note that the P4 compiler gets triggered using this file.
5. INSM-P4/src/utils/run_insm.py - a) Creates the mininet topology b) programs the switches with control plane and data plane rules c) programs the hosts with static arp entries and default routes

6. INSM-P4/src/utils/p4runtime_switch.py - To program the switch at run-time, run_insm.py uses p4runtime_switch.py
7. INSM-P4/src/utils/p4_mininet.py - Contains the definitions of P4Switch (BMv2) and P4Host.
8. INSM-P4/src/utils/netstat.py - Contains logic to ensure that grpc port to be used on BMv2 switch, for configuration, is not already in use.

RESULTS AND DISCUSSION

Test Traffic

Sending 30 IPv4 packets – (with ISNM field) from h1 to h2

Sending Iperf UDP traffic from h11 to h22

Screen Captures

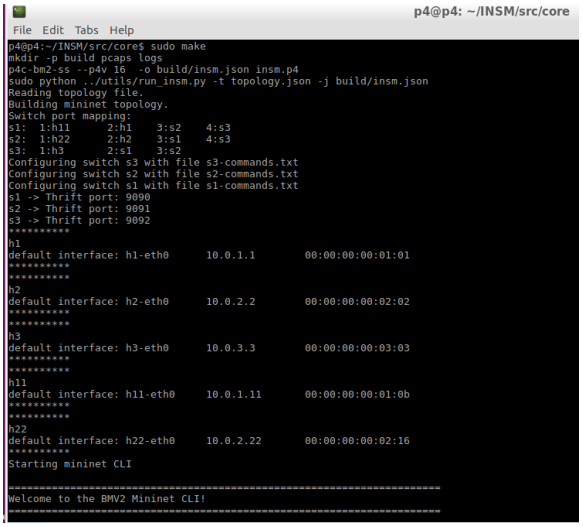

Figure: Mininet startup

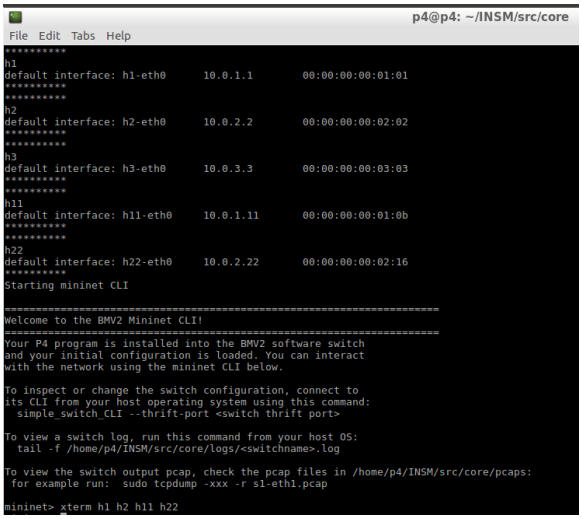

Figure: Host terminals being opened

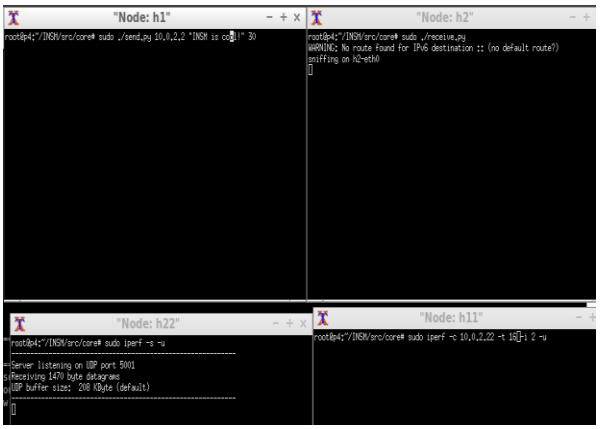

Figure: h2 and h22 start listening for traffic

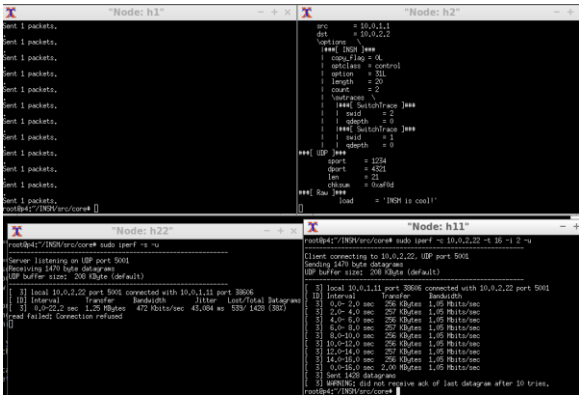

Figure: h1 and h11 start sending traffic

Per-Packet Queue Depth Record

Packet Number	S1 – queue depth	S2 – queue depth
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	49	0
8	34	0
9	44	0
10	52	0
11	59	0
12	21	0
13	4	0
14	0	0
15	0	0
16	0	0
17	0	0
18	0	0
19	0	0
20	0	0
21	0	0
22	0	0
23	0	0
24	0	0
25	0	0
26	0	0
27	0	0
28	0	0
29	0	0
30	0	0

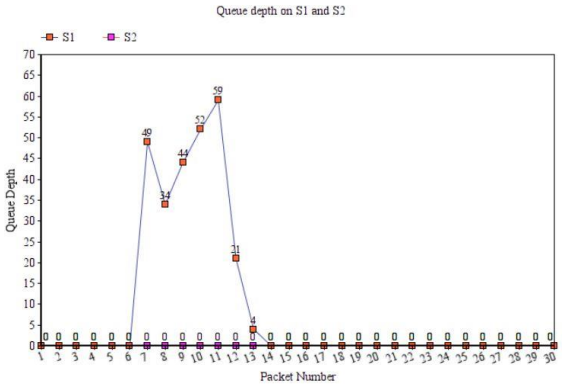

Figure: Per-Packet Queue Depth on S1 and S2

Inference

As shown in Fig. x, the packets received at h2 contain INSM field carrying queue depth information. We can observe that the queue depth obtained at S1 is temporarily higher, as both h1 and h11's traffic contend for the same output link. Queue depth at S2 is 0, as each of the destination (h2 and h22) reside at different output ports. This results in packet loss due to buffer overflow.

Note: S1's queue depth increases and then drops back to 0. The non-zero value is only seen when there is contention due to iperf traffic (which is strategically short-lived as compared to the custom traffic)

REFERENCES

1. https://www.sigcomm.org/sites/default/files/ccr/papers/2014/July/0000000-0000004.pdf
2. https://p4.org/assets/INT-current-spec.pdf
3. https://www.barefootnetworks.com/s/app/pdf/DI-UG42-003ea-ProdBrief.pdf
4. https://github.com/p4lang/behavioral-model
5. https://github.com/p4lang
6. https://ieeexplore.ieee.org/document/7502472
7. https://www.semanticscholar.org/paper/In-band-Network-Telemetry-via-Programmable-Kim-Sivaraman/a3f19dc8520e2f42673be7cbd8d80cd96e3ec0c1
8. https://www.networkworld.com/article/3163496/cloud-computing/what-p4-programming-is-and-why-it-s-such-a-big-deal-for-software-defined-networking.html