Authors:

António Vidais (96162) antonio.vidais@tecnico.ulisboa.pt
Duarte Costa (96196) duarte.c.costa@tecnico.ulisboa.pt

Project Report

Implementing Telemetry and Dynamic Load Balancing Using P4 INT and OpenFlow

Master's in Electrical and Computer Engineering

Contents

1 Introduction

1.1 P4 Programming Language

The Programming Protocol-Independent Packet Processor (P4) is a specialized language designed for network devices, allowing users to define how the data plane devices like switches, NICs, routers and filters handle packets. In the past, network device funcionality were proprietary and took years to roll out, as vendors had full control. P4 disrupts this model by empowering application developers and network engineers to define specific network behaviours. [1]

1.2 PISA - Protocol Independent Switch Architecture

PISA is an architecture designed to support P4. It enables implementation of custom packet-processing pipelines on network devices. PISA separates the control plane from the data plane, allowing the data plane to be programmed while keeping independent from specific protocols such as IPv4 or Ethernet. In traditional switches forwarding and packet processing actions are hardcoded in the silicon, so adding new features requires new hardware. In contrast, PISA-based switches allow for customizable, protocol-independent, behaviour by having P4 programs written to define the behaviour of the switch. Generally, PISA enables greater flexibility in how switches operate and process packets. It forms an hardware platform for the P4 programming language.

1.3 Proposal of Project

The main objective of this project is to make use of the specifications on the INT (In-Band Network Telemetry) headers to acquire information about the packets that are being processed by each switch, and use it to divert the flows from saturated switches, to balance the load based on trigger conditions, such as latency or packet drops.

2 Implementation

2.1 Working Environment

To start the implementation of our solution, we made use of the Virtual Machine for P4 tutorials. This VM already contains in its environment the necessary P4 compiler and Runtime, as well as Mininet to create a virtual network that simulates our implementation.

2.2 Creating the Topology

With the intent of testing how well the load in each link was balanced, a topology was created with 6 hosts and 8 switches, comprising three possible paths between the two hosts on one side an four on the other. The topology was implemented as follows:

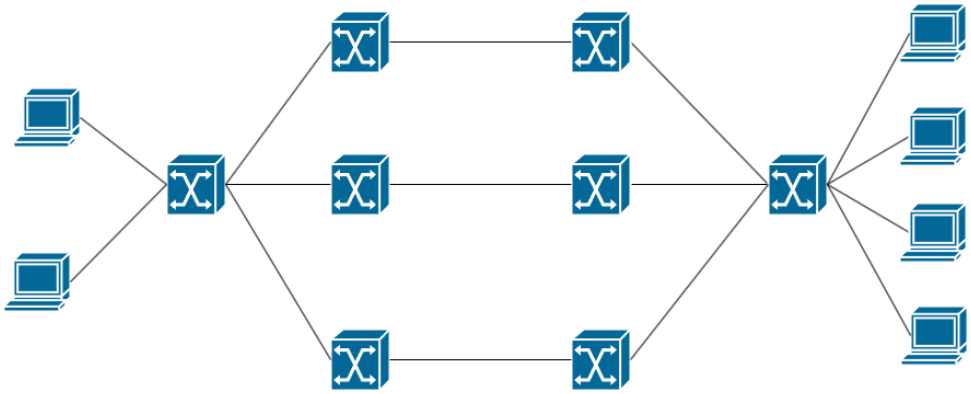

Figure 1: Topology of Simulated Network

The file *topology.json* was created, using the necessary structure to represent the schematics above in Mininet.

```
{
"hosts": {
    "h1": {
        "ip": "10.0.1.1/24",
        "mac": "08:00:00:00:01:11",
        "commands": [
            "route add default gw 10.0.1.10 dev eth0",
            "arp -i eth0 -s 10.0.1.10 08:00:00:00:01:00",
            "python3 receive.py logs/h1.txt > logs/h1_out.txt 2>&1 &"
        ]
    },
    "h2": {
        "ip": "10.0.2.2/24",
        "mac": "08:00:00:00:02:22",
        "commands": [
            "route add default gw 10.0.2.20 dev eth0",
            "arp -i eth0 -s 10.0.2.20 08:00:00:00:02:00",
            "python3 receive.py logs/h2.txt > logs/h2_out.txt 2>&1 &"
        ]
    },
    "h3": {
        "ip": "10.0.3.3/24",
        "mac": "08:00:00:00:03:33",
        "commands": [
            "route add default gw 10.0.3.30 dev eth0",
            "arp -i eth0 -s 10.0.3.30 08:00:00:00:03:00",
            "python3 receive.py logs/h3.txt > logs/h3_out.txt 2>&1 &"
        ]
    },
    "h4": {
        "ip": "10.0.4.4/24",
        "mac": "08:00:00:00:04:44",
        "commands": [
```

```
                "route add default gw 10.0.4.40 dev eth0",
                "arp -i eth0 -s 10.0.4.40 08:00:00:00:04:00",
                "python3 receive.py logs/h4.txt > logs/h4_out.txt 2>&1 &"
            ]
        },
        "h5": {
            "ip": "10.0.5.5/24",
            "mac": "08:00:00:00:05:55",
            "commands": [
                "route add default gw 10.0.5.50 dev eth0",
                "arp -i eth0 -s 10.0.5.50 08:00:00:00:05:00",
                "python3 receive.py logs/h5.txt > logs/h5_out.txt 2>&1 &"
            ]
        },
        "h6": {
            "ip": "10.0.6.6/24",
            "mac": "08:00:00:00:06:66",
            "commands": [
                "route add default gw 10.0.6.60 dev eth0",
                "arp -i eth0 -s 10.0.6.60 08:00:00:00:06:00",
                "python3 receive.py logs/h6.txt > logs/h6_out.txt 2>&1 &"
            ]
        }
    },
    "switches": {
        "s1": {},"s2": {},"s3": {},"s4": {},"s5": {},"s6": {},"s7": {},"s8": {}
    },
    "links": [
        ["h1","s1-p1"],
        ["h2", "s1-p2"],
        ["s1-p3", "s2-p1"],
        ["s1-p4","s3-p1"],
        ["s1-p5","s4-p1"],
        ["s2-p2","s5-p1"],
        ["s3-p2","s6-p1"],
        ["s4-p2","s7-p1"],
        ["s5-p2","s8-p1"],
        ["s6-p2","s8-p2"],
        ["s7-p2","s8-p3"],
        ["h3","s8-p4"],
        ["h4","s8-p5"],
        ["h5","s8-p6"],
        ["h6","s8-p7"]
    ]
}
```

2.3 Implementing In-Band Network Telemetry

Our implementation of INT was based on [2], using the INT-MD (eMbedData) headers over TCP to capture the telemetry information collected at each switch. This header is created at the source, and new information of each transit node gets appended, such as timestamps to calculate latency. The sink node is then responsible for parsing this information and send it to the controller, where it will process it and see if changes to the flows need to be made.
According to the specifications, this is how each header should be constructed:

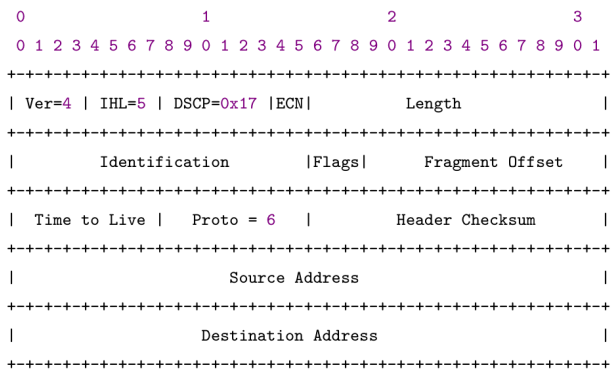

Figure 2: IP Header

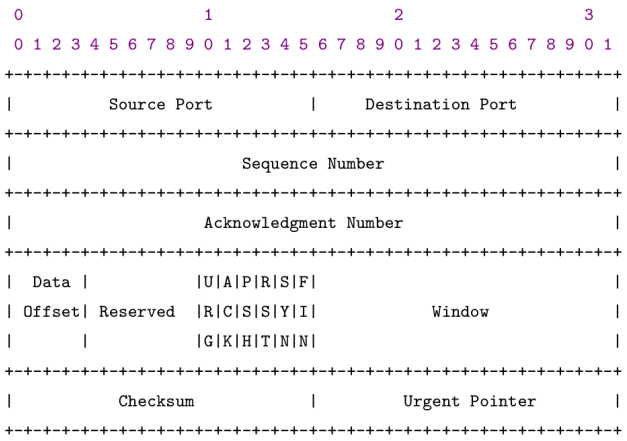

Figure 3: TCP Header

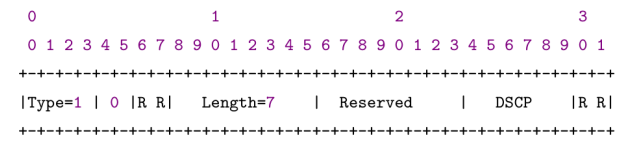

Figure 4: INT Shim Header for TCP/UDP, INT type is INT-MD (1) and NPT (Next Protocol Type) is zero

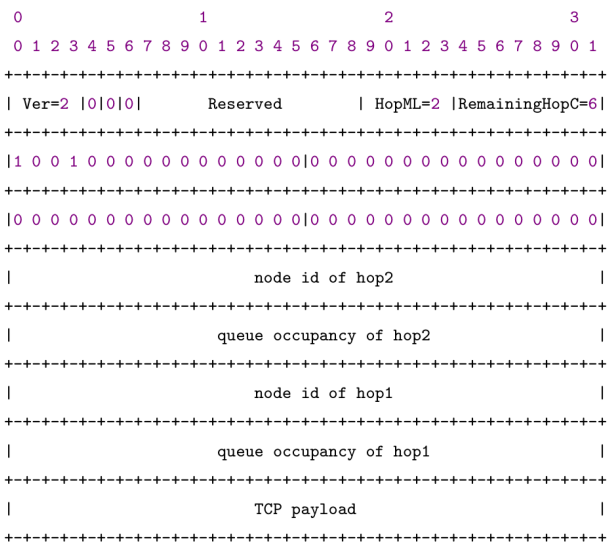

Figure 5: INT-MD Metadata Header and Metadata Stack, followed by TCP payload

2.3.1 INT Source

To enable In-band Network Telemetry (INT) monitoring, an INT source must be configured for each flow that requires tracking. A flow is defined by its source IP, destination IP, source TCP/UDP port, and destination TCP/UDP port. Once an INT source is set up for a specific flow, the node will add an INT shim header along with the initial INT node metadata headers, facilitating detailed telemetry for that flow.

```
 // Configure parameters of INT source node
// max_hop - how many INT nodes can add their INT node metadata
// hope_metadata_len - how INT metadata words are added by a single INT node
// ins_cnt - how many INT headers must be added by a single INT node
// ins_mask - instruction_mask defining which information (INT headers types) must added to
action configure_source(bit<8> max_hop, bit<5> hop_metadata_len, bit<5> ins_cnt, bit<16> ins
    hdr.int_shim.setValid();
    hdr.int_shim.int_type = INT_TYPE_HOP_BY_HOP;
    hdr.int_shim.len = (bit<8>)INT_ALL_HEADER_LEN_BYTES>>2;

    hdr.int_header.setValid();
    hdr.int_header.ver = INT_VERSION;
    hdr.int_header.rep = 0;
    hdr.int_header.c = 0;
    hdr.int_header.e = 0;
    hdr.int_header.rsvd1 = 0;
    hdr.int_header.rsvd2 = 0;
    hdr.int_header.hop_metadata_len = hop_metadata_len;
    hdr.int_header.remaining_hop_cnt = max_hop;  //will be decreased immediately by 1 within
    hdr.int_header.instruction_mask = ins_mask;

    hdr_seq_num_register.read(hdr.int_header.seq, 0);
```

```
        hdr_seq_num_register.write(0, hdr.int_header.seq + 1);

        hdr.int_shim.dscp = hdr.ipv4.dscp;

        hdr.ipv4.dscp = IPv4_DSCP_INT;   // indicates that INT header in the packet
        hdr.ipv4.totalLen = hdr.ipv4.totalLen + INT_ALL_HEADER_LEN_BYTES;  // adding size of INT

        hdr.udp.len = hdr.udp.len + INT_ALL_HEADER_LEN_BYTES;
    }
```

2.3.2 INT Transit

The *int_transit.p4* contains the code to be loaded to the INT Transit nodes, responsible for adding the telemetry information into the header. This

2.3.3 INT Sink

The INT Sink nodes correspond to the terminal switch in the flow and is responsible for removing the INT headers in the packets and reporting the information to the INT collector

2.3.4 INT Report

The code in *int_report.p4* is responsible for processing the INT information collected and sending it to

2.4 Receiving the INT information

2.5 Managing Flows Using an OpenFlow Controller

3 Conclusions

By working on this project, we were able to better understand how P4 can be used to customize the way the packages are dealt with in the switches, and how it opens doors to versatile applications, with us focusing on telemetry. However, this language can also show some complexity to work with, which proved difficult for us to implement all functionalities to the extent we were looking for.

References

[1] P4 Language Consortium, *P4: Programming protocol-independent packet processors*, Accessed: 2024-10-07, 2024. [Online]. Available: https://p4.org/.

[2] The P4.org Applications Working Group, *In-band network telemetry (int) dataplane specification v2.1*, Accessed: 2024-10-18. [Online]. Available: https://p4.org/p4-spec/docs/INT_v2_1.pdf.