Hands-on presentation

SIGCOMM 2015 - P4 Tutorial

3 different P4 exercises

- implementing EasyRoute, a custom source routing protocol
- realizing TCP flowlet switching
- observing bufferbloat with Inband Network Telemetry (INT)

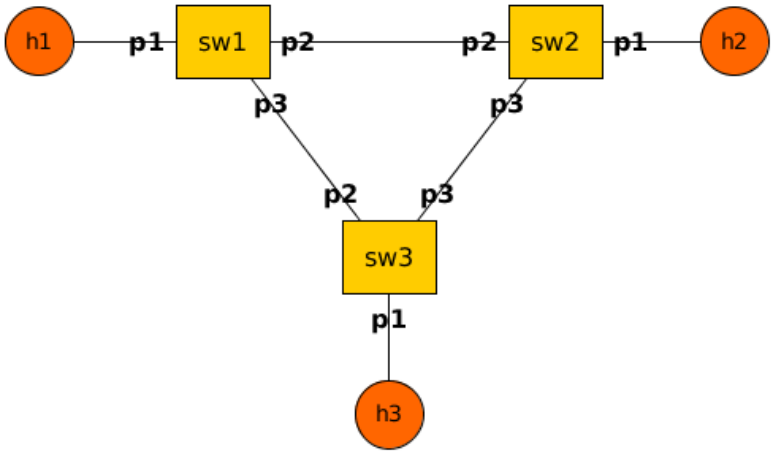
Pb1 - EasyRoute (source routing)

- very simple header

Preamble 8B of 0s	Num hops 4B	Port_1 1B	Port_2 1B	...	Port_N 1B	Payload

- preamble lets you identify EasyRoute packets
- do not have to worry about encapsulation / decapsulation at end hosts
- at each hop:
  - use the 1st port number as the outgoing port
  - decrement the number of hops
  - pop the head of the list

Pb1 - EasyRoute (source routing)



- let’s send ‘Hello’ from h1 to h3
- when it leaves h1:

```
00000000 00000000 | 00000002 | 03 | 01 | Hello
```

- when it leaves sw1 (on port 3):

```
00000000 00000000 | 00000001 | 01 | Hello
```

- when it leaves sw3 (on port 1):

```
00000000 00000000 | 00000000 | Hello
```

Pb2 - TCP Flowlet Switching

- leverage the burstiness of long TCP flows to achieve more accurate load balancing
- we start with regular ECMP, then add a flowlet_id to the list of hash fields used to select the next hop
- the flowlet_id is incremented everytime we observed a gap > 50ms between packets

Pb2 - TCP Flowlet Switching

- crc16(5-tuple) -> flow_idx
- register1[flow_idx] -> last_timestamp
  register2[flow_idx] -> flowlet_id
- if now - last_timestamp > 50ms: flowlet_id++
- crc16(5-tuple + flowlet_id) -> ecmp_nhop

Pb3 - Observing bufferbloat with INT

- more complex that Pb1 & Pb2 because of some current P4 “specificities” (see writeup)
- an oversized buffer, one long-lived TCP flow, injecting a TCP option in data packets to report queue latency and observe sawtooths

kind 8 bits - 0x40	length (in bytes) 8 bits - 0x08	switch_id 16 bits	q_latency (micros) 32 bits

Debugging your P4

- use the p4-validate tool to check that your program is correct:
  - `p4-validate p4src/source_routing.p4`
- look at the pcap files (one for each port)
- look at the switch logs in /tmp/
- ask us :)