Lecture 7: Programmable Forwarding
Nick McKeown

“Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN”
[Pat Bosshart et al. 2013]
Pat Bosshart  
At the time: TI (Texas Instruments)  
Architect of first LISP CPU and 1GHz DSP

George Varghese  
At the time: MSR  
Today: Professor at UCLA

+ Others from TI

+ Others from Stanford

At the time the paper was written (2012)…
- Fastest switch ASICs were fixed function, around 1Tb/s
- Lots of interest in “disaggregated” switches for large data-centers
Switch with fixed function pipeline

Fixed Parser

L2 Table

IPv4 Table

IPv6 Table

ACL Table

Fixed Header Processing Pipeline
Amalee Wilson
There’s a key phrase in the abstract, “contrary to concerns within the community,” and I’m curious about what those concerns are.
“Programmable switches run 10x slower, consume more power and cost more.”

Conventional wisdom in 2010
Packet Forwarding Speeds

Gb/s (per chip)

0 1 10 100 1000 10000 100000


Switch Chip

6.4Tb/s
Packet Forwarding Speeds

- Switch Chip: 80x Gb/s (per chip)
- CPU: 6.4Tb/s

 plot with years 1990-2020 on the x-axis and Gb/s (per chip) on the y-axis, showing an increase in packet forwarding speed over time with a significant jump to 6.4Tb/s in 2020.
Domain Specific Processors

- Computers
  - Java Compiler
  - OpenCL Compiler
  - Matlab Compiler
  - TensorFlow Compiler
  - CPU

- Graphics
  - GPU

- Signal Processing
  - DSP

- Machine Learning
  - TPU

- Networking
  - Language Compiler
  - ???
Domain Specific Processors

Computers

Java Compiler
CPU

Graphics

OpenCL Compiler
GPU

Signal Processing

Matlab Compiler
DSP

Machine Learning

TensorFlow Compiler
TPU

Networking

P4 Compiler
PISA aka “RMT”
Network systems tend to be designed "bottom-up"

“This is how I process packets…"

Switch OS

Driver

Fixed-function switch
What if they could be programmed “top-down”?

“This is precisely how you must process packets”
You said

Wantong Jiang:
At the end of the paper, the authors mention FPGA and claim that they are too expensive. This paper was published in 2013 and I wonder if it's still the case nowadays.

Firas Abuzaid:
The paper mentions that FPGAs are too expensive to be considered. Now that FPGAs have become more widely available, could they be used instead of RMTs?
The RMT design [2013]
Figure 3: Switch chip architecture.
(a) RMT model as a sequence of logical Match-Action stages.

(b) Flexible match table configuration.

(c) VLIW action architecture.

Figure 1: RMT model architecture.
Will Brand

What goes into designing the vocabulary of a RISC instruction set? Since I can't just try to prove the instructions are Turing-complete, and the instruction set doesn't have the kind of specification I might expect from a general-purpose language, I find it difficult to "trust" that Table 1 encapsulates a reasonable portion of the actions we might want to make possible…
PISA: Protocol Independent Switch Architecture

Programmer declares which headers are recognized

Programmer declares what tables are needed and how packets are processed

All stages are identical. A “compiler target”.
PISA: Protocol Independent Switch Architecture
PISA: Protocol Independent Switch Architecture
PISA: Protocol Independent Switch Architecture
P4 program example: Parsing Headers

```p4
def parser parse_ethernet {
  extract(ethernet);
  return select(latest.etherType) {
    0x8100 : parse_my_encap;
    0x800 : parse_ipv4;
    0x86DD : parse_ipv6;
  }
}
```
**P4 program example**

```
table ipv4_lpm {
  reads {
    ipv4.dstAddr :
  }
  actions {
    set_next_hop;
    drop;
  }
}

control ingress {
  apply(l2);
  apply(my_encap);
  if (valid(ipv4) {
    apply(ipv4_lpm);
  } else {
    apply(ipv6_lpm);
  }
  apply(acl);
}

action set_next_hop(nhop_ipv4_addr) {
  modify_field(metadata.nhop_ipv4_addr, nhop_ipv4_addr);
  modify_field(standard_metadata.egress_port, port);
  add_to_field(ipv4.ttl, -1);
}
```
How programmability is used

1. Reducing complexity
Reducing complexity

IPv4 and IPv6 routing
- Unicast Routing
- Routed Ports & SVI
- VRF
- Unicast RPF
- Strict and Loose
- Multicast
- PIM-SM/DM & PIM-Bidir

Ethernet switching
- VLAN Flooding
- MAC Learning & Aging
- STP state
- VLAN Translation

Load balancing
- LAG
- ECMP & WCMP
- Resilient Hashing
- Flowlet Switching

Fast Failover
- LAG & ECMP

Tunneling
- IPv4 and IPv6 Routing & Switching
  - IP-in-IP (6in4, 4in4)
  - VXLAN, NVGRE, GENEVE & GRE
  - Segment Routing, ILA

MPLS
- LER and LSR
- IPv4/v6 routing (L3VPN)
- L2 switching (FoMPLS, VPLS)
- MPLS over UDP/GRE

ACL
- MAC ACL, IPv4/v6 ACL, RACL
- QoS ACL, System ACL, PBR
- Port Range lookups in ACLs

QOS
- QoS Classification & marking
- Drop profiles/WRED
- RoCE v2 & FCoE
- CoPP (Control plane policing)

Security Features
- Storm Control, IP Source Guard

Monitoring & Telemetry
- Ingress Mirroring and Egress Mirroring
- Negative Mirroring
- Sflow
- INT

Counters
- Route Table Entry Counters
- VLAN/Bridge Domain Counters
- Port/Interface Counters

Protocol Offload
- BFD, OAM

Multi-chip Fabric Support
- Forwarding, QOS
Reducing complexity

Compiler
My switch.p4

Switch OS
Driver

Programmable Switch
How programmability is used

2 Adding new features
Protocol complexity 20 years ago

- Ethernet
- IPv4
- IPX

Diagram:

- Ethernet
  - ethtype
    - IPv4
    - IPX
Datacenter switch today

switch.p4
Example new features

1. New encapsulations and tunnels
2. New ways to tag packets for special treatment
3. New approaches to routing: e.g. source routing in DCs
4. New approaches to congestion control
5. New ways to process packets: e.g. ticker-symbols
Example new features

1. Layer-4 Load Balancer\(^1\)
   - Replace 100 servers or 10 dedicated boxes with one programmable switch
   - Track and maintain mapping for 5-10 million HTTP flows

2. Fast stateless firewall
   - Add/delete and track 100s of thousands of new connections per second

3. Cache for Key-value store\(^2\)
   - Memcache in-network cache for 100 servers
   - 1-2 billion operations per second

How programmability is used

3 Network telemetry
1. "Which path did my packet take?"

   "I visited Switch 1 @780ns, Switch 9 @1.3\mu s, Switch 12 @2.4\mu s"

2. "Which rules did my packet follow?"

   "In Switch 1, I followed rules 75 and 250. In Switch 9, I followed rules 3 and 80. "

<table>
<thead>
<tr>
<th>#</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>192.168.0/24</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
3. “How long did my packet queue at each switch?”

4. “Who did my packet share the queue with?”

“Delay: 100ns, 200ns, 19740ns”
3. "How long did my packet queue at each switch?"

4. "Who did my packet share the queue with?"

"Delay: 100ns, 200ns, 19740ns"
These seem like pretty important questions

1. “Which path did my packet take?”
2. “Which rules did my packet follow?”
3. “How long did it queue at each switch?”
4. “Who did it share the queues with?”

A programmable device can potentially answer all four questions. At line rate.
INT: In-band Network Telemetry

Add: SwitchID, Arrival Time, Queue Delay, Matched Rules, …

Original Packet

Log, Analyze Replay

Visualize
Example using INT

Anomaly Records

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Switch Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>July 25, 2017 - 18:17:51.513 UTC</td>
<td></td>
</tr>
</tbody>
</table>

Queue Occupancy Over Time (bytes)

17 Affected Flows

<table>
<thead>
<tr>
<th>Flow</th>
<th>kB In Queue</th>
<th>% of Queue Buildup</th>
<th>Packet Drops</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.32.2.2:46380 -&gt; 10.36.1.2:5101 TCP</td>
<td>3282</td>
<td>29</td>
<td>0</td>
</tr>
<tr>
<td>10.32.2.2:46374 -&gt; 10.36.1.2:5101 TCP</td>
<td>3073.5</td>
<td>27</td>
<td>25</td>
</tr>
<tr>
<td>10.32.2.2:46386 -&gt; 10.36.1.2:5101 TCP</td>
<td>2092.5</td>
<td>18</td>
<td>27</td>
</tr>
<tr>
<td>10.32.2.2:46388 -&gt; 10.36.1.2:5101 TCP</td>
<td>1456.5</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>10.32.2.2:46390 -&gt; 10.36.1.2:5101 TCP</td>
<td>1227</td>
<td>11</td>
<td>36</td>
</tr>
<tr>
<td>10.32.2.2:46372 -&gt; 10.36.1.2:5101 TCP</td>
<td>45</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10.32.2.2:46392 -&gt; 10.36.1.2:5101 TCP</td>
<td>37.5</td>
<td>0</td>
<td>39</td>
</tr>
<tr>
<td>10.35.1.2:34256 -&gt; 10.36.1.2:5102 TCP</td>
<td>34.5</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
End.