SDN
Phase 3: Getting the humans out of the way

Nick McKeown
Stanford University
With SDN we will:

1. Formally verify that our networks are behaving correctly.
2. Identify bugs, then systematically track down their root cause.
“closed and proprietary”
“proliferation of standards”
“barrier to entry”
“stranglehold by vendors”
“ossification”
“clean slate”
“4D, Ethane, etc”

Number of IETF RFCs
“closed and proprietary”
“proliferation of standards”
“barrier to entry”
“stranglehold by vendors”
“ossification”
“clean slate”
“4D, Ethane, etc”

Open-source
Disaggregation
SDN  NFV  Network  Programmable  forwarding  Telemetry

OpenFlow  Virtualization  2010  2020  2030
“closed and proprietary”
“proliferation of standards”
“barrier to entry”
“stranglehold by vendors”
“ossification”
“clean slate”
“4D, Ethane, etc”

Phase 1
Network owners take control of their software 2020

Now we take it for granted!

ONF has played a big role in this transformation:
ONOS, CORD, Trellis, SEBA, Stratum ...
Phase 1
Network owners take control of their software

Phase 2
Network owners take control of packet processing too
Switch with fixed function pipeline

Fixed Parser

L2 Table

IPv4 Table

IPv6 Table

ACL Table

Fixed Header Processing Pipeline
Switch OS

Driver

OSPF  BGP  New  etc.
Network systems were built “bottom-up”

“This is how I process packets ...”
Network systems starting to be built “top-down”

“This is precisely how you must process packets”
PISA: Protocol Independent Switch Architecture

Programmable Parser

Match+Action Stage

Programmable Memory
ALU

Programmable Match-Action Pipeline

Generalization of RMT [Sigcomm’13]
PISA: Protocol Independent Switch Architecture
Example P4 Program

Parser Program

```p4
parser parse_ethernet {
  extract(ethernet);
  return switch(ethernet.ethertype) {
    0x8100 : parse_vlan_tag;
    0x0800 : parse_ipv4;
    0x8847 : parse_mpls;
    default: ingress;
  }
}
```

Header and Data Declarations

```p4
header_type ethernet_t { ... }
header_type l2_metadata_t { ... }

header ethernet_t ethernet;  
header vlan_tag_t vlan_tag[2];
metadata l2_metadata_t l2_meta;
```

Tables and Control Flow

```p4
table port_table { ... }

control ingress {
  apply(port_table);
  if (l2_meta.vlan_tags == 0) {
    process_assign_vlan();
  }
}
```
Why I devoted 5 years to programmable forwarding...

Programmable switch chips can have the same power, performance and cost as fixed function switches.

Beautiful new ideas are now owned by the programmer, not the chip designer.

Which means more innovation.
How do we know if a programmable switch chip has the same power, performance and cost as a fixed function switch chip?
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>P4 Programmable “Tofino”</th>
<th>Fixed Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2/L3 Throughput</td>
<td>6.4Tb/s</td>
<td>6.4Tb/s</td>
</tr>
<tr>
<td>Number of 100G Ports</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Availability</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Max Forwarding Rate</td>
<td>5.1B packets per sec</td>
<td>4.2B packets per sec</td>
</tr>
<tr>
<td>Max 25G/10G Ports</td>
<td>256/258</td>
<td>128/130</td>
</tr>
<tr>
<td>Programmability</td>
<td>Yes (P4)</td>
<td>No</td>
</tr>
<tr>
<td>Typical System Power draw</td>
<td>4.2W per port</td>
<td>5.3W per port</td>
</tr>
<tr>
<td>Large Scale NAT</td>
<td>Yes (100k)</td>
<td>No</td>
</tr>
<tr>
<td>Large scale stateful ACL</td>
<td>Yes (100k)</td>
<td>No</td>
</tr>
<tr>
<td>Large Scale Tunnels</td>
<td>Yes (192k)</td>
<td>No</td>
</tr>
<tr>
<td>Packet Buffer</td>
<td>Unified</td>
<td>Segmented</td>
</tr>
<tr>
<td>Segment Rtg/Bare Metal</td>
<td>Yes/Yes</td>
<td>No/No</td>
</tr>
<tr>
<td>LAG/ECMP Hash Algorithm</td>
<td>Full entropy, programmable</td>
<td>Hash seed, reduced entropy</td>
</tr>
<tr>
<td>ECMP</td>
<td>256 way</td>
<td>128 way</td>
</tr>
<tr>
<td>Telemetry and Analytics</td>
<td>Line-rate per flow stats</td>
<td>Sflow (Sampled)</td>
</tr>
<tr>
<td>Latency</td>
<td>Under 400 ns</td>
<td>450 ns</td>
</tr>
</tbody>
</table>

Otherwise, both systems are identical:
- # of Ports
- CPU
- Power Supplies
SDN, Part 2: Programmable Forwarding

How it gets used
1. Reducing complexity
2. Adding new features to the network
3. Telemetry

P4.org
• Now part of ONF
• Lots of activities and workshops: get involved!
• P4-16 stable. Device independent: Switches, NICs, FPGAs, vSwitches
• P4Runtime part of Stratum, launched this week

A cast of many, led by: Nate Foster (Cornell), Amin Vahdat (Google), Jennifer Rexford (Princeton), Chang Kim (Barefoot)
The network (switch, router, NIC, firewall, 5G...) is now a programmable platform. Top down, including the control plane and the forwarding plane.

Phase 1
Network owners take control of their software

Phase 2
Network owners take control of packet processing too
Extrapolating to 2030

1. NICs, Switches, vSwitches, stacks will have been programmable for 10 years.

2. We will think of a network as a programmable platform. Behavior described at the top. Then partitioned, compiled and run across elements.

3. Every large network will work slightly differently, programmed and tailored locally.
4. We will no longer think in terms of protocols. Instead, we will think in terms of software. All functions and “protocols” will have migrated up and out of hardware into software.

5. Networking students will learn how to program a network top-down, as a distributed computing platform. Protocols will be described in quaint historical terms.

6. “Routing” and “Congestion control” will be programs, partitioned across the end-to-end system by a compiler.
If we want to get the humans out of the way, what else do we need?
Three pieces

1. The ability to observe packets, network state and code, in real-time.
2. The ability to generate new control and forwarding behaviors, on the fly, to correct errors.
3. The ability to verify newly generated code and deploy it quickly.
Observing packets

Per-packet telemetry is already starting to happen
Today, basic information is hard to find

1. “Which path did my packet take?”
   “I visited Switch 1 @780ns, Switch 9 @1.3µs, Switch 12 @2.4µs”

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?”

“Delay: 100ns, 200ns, 19740ns”

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

Aggressor flow!
Today, basic information is hard to find

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?”

With P4 + INT we can answer all four questions for the first time. At full line rate. Without generating additional packets.
INT: In-band Network Telemetry

SwitchID, Arrival Time, Queue Delay, Matched Rules, ...

Original Packet

Log, Analyze Replay and Visualize

+ SONATA [Sigcomm ‘18], Sketches [Sigcomm ‘12] ...
Viewing Microbursts (to the nanosecond)

Anomaly Records

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

1. The ability to observe packets, network state and code, in real-time.
2. The ability to generate new control and forwarding behaviors, on the fly, to correct errors.
3. The ability to verify newly generated code and deploy it quickly.
Header Space Analysis

$T_1(h, p)$

$T_2(h, p)$

$T_3(h, p)$

$T_4(h, p)$

A

B

HSA [NSDI ‘12]
Example: Can A talk to B?
Three pieces

1. The ability to observe packets, network state and code, in real-time.
2. The ability to generate new control and forwarding behaviors, on the fly, to correct errors.
3. The ability to verify newly generated code and deploy it quickly.
Software Defined Network (SDN)

Abstract Network View

Network Virtualization

Global Network View

Network OS

Packet Forwarding

Packet Forwarding

Packet Forwarding

Packet Forwarding

Packet Forwarding

Packet Forwarding

Packet Forwarding
Software Defined Network (SDN)

Control: Generate and Verify control code

Observe
- Control code: Measure and Validate

Observe
- State: Measure and Validate

Observe
- Packets: Measure and Validate

Network Virtualization

Abstract Network View

Global Network View

Network OS

Packet Forwarding

Partition, Generate, Verify, Download
Getting humans out of the way

SDN with Verifiable Closed-Loop Control

Network owners and operators will use fine-grain measurement and formal verification to automate network control at scale.

Joint work with: Nate Foster (Cornell), Guru Parulkar (ONF), Larry Peterson (ONF), Jennifer Rexford (Princeton)
ONF Open-source Software Today

Trellis

ONOS Control Plane

P4Runtime Contract

Stratum OS

Stratum OS

Stratum OS

P4 switch

P4 switch

P4 switch

P4-OVS

P4-OVS

NIC

NIC

P4 NIC

P4 NIC

P4 N
Verifiable Closed-Loop Control

First production & research tools exist (INT/DeepInsight, SONATA)

Early research tools (p4v)

Control code
Contract
dataplane code

Fine-grained Per-packet Measurement
Phase 1
Network owners take control of their software

2010

Phase 2
Network owners take control of packet processing too

Phase 3
Networks managed by verifiable closed loop control

2030
With SDN we will:

1. Formally verify that our networks are behaving correctly.
2. Identify bugs, then systematically track down their root cause.
3. Measure and validate correctness, then generate and verify code fix.
   Download to correct the bug.
4. Goto beach....?