Rethinking the SDN Abstraction:
May the Flexibility, Scalability and Security be with Us

Chengchen Hu
XJTU
Oct. 15, 2016
Traditional Computer Networks

Data plane:
Packet streaming

Table lookup, forward, filter, buffer, mark
Traditional Computer Networks

Control plane: Distributed algorithms

Track topology changes, compute routes, install forwarding path
Legacy Computer Networks

Closed

Specialized Packet Forwarding Hardware
Operating System
App
App
App

Specialized Packet Forwarding Hardware
Operating System
App
App
App

Specialized Packet Forwarding Hardware
Operating System
App
App
App

Specialized Packet Forwarding Hardware
Operating System
App
App
App
The Ossified Network

Routing, management, mobility management, access control, VPNs, ...

Million of lines of source code  6000+ RFCs  Barrier to entry

Billions of gates  Bloated  Power Hungry

Many complex functions baked into the infrastructure

OSPF, BGP, VLAN, multicast, differentiated services, traffic engineering, NAT, firewalls, MPLS, ...

An industry with a “mainframe-mentality”, reluctant to change
Software Defined Network (SDN)
Software Defined Network (SDN)

2. At least one Network OS probably many Open- and closed-source

3. Well-defined open API
Key Features of SDN

- Separate Control and Data
- Abstraction
- Global view

Logically-centralized control

API (abstraction) to the data plane (e.g., OpenFlow)

Smart, slow

Dumb, fast

Switches
History

- **Y2006**: Openflow/SDN concept
- **Y2009**: OpenFlow V1.0
- **Y2012**: Google B4, VMWare/Nicira VN
- **Y2013-2014**: Spanish/China Mobile Telecom/AT&T/Huawei built SDN-based IPRAN for testing
- **Y2013**: Microsoft SWAN for Inter-DC scheduling

We are now considering

More Flexible, Dependable, Scalable, secure and more apps.
Legacy $ightarrow$ OpenFlow $ightarrow$ POF/P4

POF/P4 is flexible enough?

What’s next?
Rethink...

More Secure

- Less Conflicts
- Less Talk

Controller

Network Function

Easier Debugging

More Flexible
Case 0: programmable SDN Data Plane

- **ONetCard**
  - 2012 Aug
  - PCIe Card

- **ONetSwitch 45**
  - 2013 Aug
  - 4*10G, 4*GE, wifi

- **ONetSwitch 20**
  - 2013 Dec
  - 4*GE, with ZEDboard

- **ONetSwitch 30**
  - 2014 Dec.
  - wifi/storage, 5*GE

Sponsored by Xilinx

http://onetswitch.org
ONetSwitch: All programmable SDN Switch


Users

400+ ONetSwitches deployed in China, Europe, US
Case 1: Frequent protocols

Case 1: ARP Gateway
Physical Gateway usually doesn’t exist in SDN
Controller reply to server end hosts
Data plane queries controller repeatedly

Case 2: LLDP (Link Layer Discovery Protocol)
Get neighbor info. then keep watching
Data plane packet goes to controller repeatedly even topology is stable
Usually 2-5s per port per packet

Case 3: LACP (Link aggregation Control Protocol)
State maintenance
Repeat the same work again and again after topology is stable
Usually 8 packets to converge
Usually 1/30s per port per packet after convergence
Communication overhead reduced: 50%-100%
Controller CPU work load reduced: 80%-98%
ARP response time: from 10+ms to us

FOCUS: Function Offloading from a Controller to Utilize Switch Power.
Poster @ NSDI’16 and @ SDN/NFV workshop’16
Case 2: Table-miss

Repeated flowmod message: 68%-98% when flow table with 1k-50k entries. Eat either fast path memory or bandwidth between switch and controller. Becomes bottleneck between Slow path and fast path in switch. Small flows make the problem worse.

CoSwitch: A Cooperative Switching Design for Software Defined Data Center Networking
@HotData, 2014 (best paper award)
Co-Work with IBM Research Lab
Taming the Flow Table Overflow in OpenFlow Switch. Poster at SIGCOMM '16.
Case 3: Rules Conflicts

- blue and orange policies are only to connect A and D → random overwriting
- blue and orange policies critically specify the current path → unsolvable
- blue/orange policy is to connect A and with B/C in path → A to B to C to D

Hard to eliminate the conflicts without the high-level intents

Modular SDN Compiler Design with Intermediate Representation.
poster @ SIGCOMM ’16.
### Diagram 1

- **{A, DstIP 10.0.0.2}, FORWARD**: towards B \(\cap\) forbid A
- **{A, DstIP 10.0.0.2}, FORWARD**: towards C \(\cap\) forbid A
- **{B, DstIP 10.0.0.2}, FORWARD**: towards D \(\cap\) forbid B
- **{C, DstIP 10.0.0.2}, FORWARD**: towards D \(\cap\) forbid C
- **{D, DstIP 10.0.0.2}, FORWARD**, fixed_forward 2
- **{D, DstIP 10.0.0.2}, FORWARD**, fixed_forward 2

### Diagram 2

- **avg. #insts before**
- **avg. #insts after**
- **max #insts before**
- **max #insts after**

### Diagram 3

- **elimination time (s)**
- **decouple**
- **elimination**

### Diagram 4

- **#hops**
- **before**
- **after**

### Data Table

<table>
<thead>
<tr>
<th>Platform</th>
<th>Stanford</th>
<th>FT(8)</th>
<th>FT(16)</th>
<th>FT(32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#hops</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>
Case 4: Rule update

- Flowtable update bottleneck
- 10s to 100s of rule edits per second
- Full refresh of 5K entries takes minutes

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;1, 2&gt;</td>
<td>3</td>
</tr>
<tr>
<td>* , 2</td>
<td>2</td>
</tr>
<tr>
<td>* , *</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;1, 2&gt;</td>
<td>5</td>
</tr>
<tr>
<td>&lt;2, *&gt;</td>
<td>4</td>
</tr>
<tr>
<td>&lt;1, *&gt;</td>
<td>3</td>
</tr>
<tr>
<td>* , 2</td>
<td>3</td>
</tr>
<tr>
<td>&lt;3, *&gt;</td>
<td>2</td>
</tr>
<tr>
<td>* , *</td>
<td>1</td>
</tr>
</tbody>
</table>

Old

New

Priority Updates
3 rule adds + 2 priority updates
- Unmodified fields
- Modified fields
Try to minimize the update

(a) An example of rule update.  (b) Update with priority.  (c) Update with dependency graph.

CACTI: CAche Counting

Diagram showing the Fast Path and Slow Path of CACTI, with Flow ID Reg., Flow Identification, Counter Register Update Logic, Shadow Counter Reg., DMA, Update process, Update Queue, and Counter Array.
Case 6: L7 Abstraction

We propose a new specification form DCCFG, which can be formulated as a six-tuple, $\Gamma = (N, \Sigma, C, R, S, E)$, where $N$, $\Sigma$, $C$, $R$, $S$, $E$ are the finite set of non-terminals, terminals, counters, production rules, start non-terminal, and extraction tokens, respectively. The non-terminals are the symbols where the terminals can be derived. The terminals can be a single character or a RegEx. The production rules can be described as $<guard>: <non-terminal> \rightarrow <body><action>$.

1. $S \rightarrow T L V$
2. $T \rightarrow <key, 0, "[A-Z]">$
3. $L \rightarrow "[0-9]" [len=getnum()]
4. $[len>0]V \rightarrow <value, 0, "[A-Za-z]*" > [len=reduce()]
5. $[len=0]V \rightarrow \epsilon$

Figure 3: A TLV specification $\Gamma$ in DCCFG.
Case 7: Vulnerabilities

- Flow table attack
- Secure Channel attack
- Session Hijacking
- Policy bypass
- ....

On Denial of Service Attacks in Software Defined Networks @ IEEE Network
SDNShield: Reconciliating Configurable Application Permissions for SDN App Markets @ DSN 2016
Mind the Gap: Monitoring the Control-Data Plane Consistency in Software Defined Networks @ CoNext 2016
Open Questions

- How to make data plane programable?
- Match-Action abstraction?
- Forwarding—Switch, Control—Controller?
- Fully Centralization?
- Anything between high level intents and low level rules?
- How to co-design Fast Path & Slow Path in Switch?
Thank you