Software-Defined Networking: OpenFlow and Frenetic

Mohamed Ismail
Background
Problem: Programming Networks is Hard
Network Stack Pros

- Key to the success of the Internet

- Layers and layers of abstraction

- Independent innovation at each layer
  - Communication media
  - Ethernet standards
  - Transport layer protocols

- Follows end-to-end argument

(Source: Shenker, 2011)
Network Stack Cons

- Network switches and routers built and optimized for internet traffic

- Network components and internet protocols set in stone
  - Difficulty to switch from IPv4 to IPv6

- Difficult to perform research on Internet

Problem:
Network infrastructure has “ossified”

(Source: Shenker, 2011)
Functions of a switch/router

- Receive a packet and send to appropriate destination
- Prevent a packet from reaching a certain destination
Programming a switch/router

- Use a limited API to program the switch/router flow table
- Must program each network device separately
- Programming dependent on topology
- Does not scale

Problem: No generalized API for programming scalable networks
Data Plane vs. Control Plane

Data Plane
- Receive a packet
- Forward packet based on flow table
- Network stack abstractions are data plane abstractions

Control Plane
- Update flow table to specify where packets should go
- Update flow table to specify where packets should not go
- No abstractions for updating the control plane
Programming networks is hard because...

• Network stack is an abstraction for the data plane

• Network infrastructure has “ossified” due to the success of the internet

• Switch and router internals vary by manufacturer and there is no standard API for the control plane

• Without any abstractions for control plane, research and innovation in network programming is near impossible
  • Must compute configuration of each device
  • Can only work with given network-level protocol (i.e. IP)
OpenFlow
Authors

• Nick McKeown
  • ‘95 PhD UC Berkeley
  • Co-founded Nicira Networks, ONF
  • Faculty at Stanford

• Tom Anderson
  • ‘91 PhD Univ. of Wash.
  • UC Berkeley ‘91-’97
  • Faculty at Univ. of Wash.

• Hari Balakrishnan
  • ‘98 PhD UC Berkeley
  • Faculty at MIT

• Guru Parulkar
  • ‘87 PhD Univ. of Delware
  • Many network-related startups
  • Executive director of Clean Slate Internet Design Program

• Larry Peterson
  • ‘85 PhD Purdue University
  • GENI project chair
  • Faculty at Princeton

• Jennifer Rexford
  • ‘96 PhD Univ. of Mich.
  • AT&T Labs ‘96-’05
  • Broader Gateway Protocol
  • Faculty at Princeton

• Scott Shenker
  • ‘83 PhD Univ. of Chig.
  • XEROX Parc
  • Co-founder of Nicira Networks, ONF
  • Faculty at Berkeley

• Jonathan Turner
  • Faculty at Washington University in St. Louis
Goals

• Run experiments on campus networks

• Software-based approach

• Low cost
Goals and Challenges

• Run experiments on campus networks
  • Reluctance by admins to using experimental equipment on college network
  • Isolation: Control over network without disruptions to normal traffic
  • What functionality is needed for experiments?

• Software-based approach
  • Software-based solutions have low performance
  • Software-based solutions support low port density

• Low cost
  • Take advantage of existing infrastructure
  • Closed platforms from vendors
Take Aways

• OpenFlow allows network devices to decouple the data plane from the control plane
• Data plane processing done by network device
• Data plane abstraction is the network stack
• Control plane processing done by controller
• New control stack for OpenFlow devices provides standardized API and abstractions necessary to innovate in field of network management
Design

• Separate data plane from control plane

• Data plane
  • High performance forwarding

• Control plane
  • Flow table is programmable
  • Accessed through controller using OpenFlow Protocol
OpenFlow API

- Forward packets to given port (or ports)
- Forward packets to controller
  - Usage: Can analyze and process packets
- Drop the packet
  - Usage: Protect against attacks by removing suspicious packets
Flow Table Entry

- Packet header to define flow
- Action to be performed
- Statistics

<table>
<thead>
<tr>
<th>MAC src</th>
<th>MAC dst</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>TCP dport</th>
<th>...</th>
<th>Action</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10:20:..</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>port 1</td>
<td>250</td>
</tr>
<tr>
<td>+</td>
<td></td>
<td>+</td>
<td>5.6.7.8</td>
<td>+</td>
<td>+</td>
<td>port 2</td>
<td>300</td>
</tr>
<tr>
<td>+</td>
<td></td>
<td>+</td>
<td></td>
<td>25</td>
<td>+</td>
<td>drop</td>
<td>892</td>
</tr>
<tr>
<td>+</td>
<td></td>
<td>+</td>
<td>192.192</td>
<td>+</td>
<td>+</td>
<td>local</td>
<td>120</td>
</tr>
<tr>
<td>+</td>
<td></td>
<td>+</td>
<td></td>
<td></td>
<td>+</td>
<td>controller</td>
<td>11</td>
</tr>
</tbody>
</table>
Isolation

Two Options:

• Add another action to the OpenFlow API
  • Forward packets through normal pipeline

  OR

• Define separate VLANs
  • No overlap over production and experimental traffic
Discussion

• What is easy to accomplish with the OpenFlow solution?

• What is still hard to do with OpenFlow?
Controllers

- Must communicate using OpenFlow protocol
- Individual controllers for multiple switches or single controller for all switches
- Use with Network OS
  - NOX
- Should provide some permissions to prevent mixing of traffic or unauthorized flow table changes
- Implementation details left unspecified
Control Stack

- OpenFlow is only a means to achieve the decoupling needed for Software-Defined Networking
- Network OS provides common control functionality that can be used by multiple applications

(Sources: Casado, 2011; Shenker, 2011)
Discussion

• What functionality should the Network OS have?

• What layers or abstractions are missing from the control stack?
Google B4

• Provides connectivity among Google datacenters

• Use SDN and OpenFlow

• Centralized traffic engineering application
  • Resource contention
  • Multipath forwarding/tunneling to leverage network capacity according to application priority
  • Dynamically relocate bandwidth

• Many links run at near 100% utilization for extended periods of time

(Source: Jain, 2013)
Open Network Foundation

• Promote adoption of Software-Defined Networking through open standards such as OpenFlow

• Partners:
Open Network Foundation

- Promote adoption of Software-Defined Networking through open standards such as OpenFlow

- Partners:
Open Network Foundation

• Promote adoption of Software-Defined Networking through open standards such as OpenFlow

• Partners:
Open Network Foundation

- Promote adoption of Software-Defined Networking through open standards such as OpenFlow

- Partners:
Open Network Foundation

- Promote adoption of Software-Defined Networking through open standards such as OpenFlow

- Partners:
Take Aways

- OpenFlow allows network devices to decouple the data plane from the control plane
- Data plane processing done by network device
- Data plane abstraction is the network stack
- Control plane processing done by controller
- New control stack for OpenFlow devices provides standardized API and abstractions necessary to innovate in field of network management
Frenetic
Authors

- Nate Foster
  - ‘09 PhD Upenn
  - Faculty at Cornell

- Rob Harrison
  - ‘11 Masters Princeton
  - Westpoint

- Matthew L. Meola
  - ?

- Michael J. Freedman
  - PhD NYU
  - CoralCDN
  - Faculty at Princeton

- Jennifer Rexford
  - ‘96 PhD Univ. of Mich.
  - AT&T Labs ‘96-’05
  - Broader Gateway Protocol
  - Faculty at Princeton

- David Walker
  - ‘01 PhD Cornell (Morrisett)
  - Faculty at Princeton
Problems

• OpenFlow is a “machine language”
  • Directly reflects underlying hardware
  • High level policy may require multiple low-level rules

• Network programs are not isolated from each other
  • No equivalent of virtual memory space
  • Composition of programs is a manual process and error prone

• Controller does not see all traffic, so some information may be hidden
  • Delay in programming switches and routers
  • Must take care of additional corner cases

Hard to effectively program OpenFlow tables using NOX
Take Aways

• OpenFlow is the “machine language” of network programming
  • Difficult to program correctly and efficiently
  • Not enough layers of abstraction for programmers

• Frenetic addresses issues with compositibility, low-level interaction, and providing a unified view through the Frenetic run-time system and Frenetic programming language
Approach

- Add a layer of abstraction
  - Run-time system converts between high-level program to correct low-level network rules

- Frenetic programming language based on functional reactive programming (FRP)
  - “See every packet” abstraction
  - Composition
  - Rich pattern algebra

(Source: Foster, 2010)
Example w/o Frenetic

def repeater(switch):
    p1 = {IN_PORT:1}
    p2 = {IN_PORT:2}
    a1 = [output(2)]
    a2 = [output(1)]
    install(switch, p1, a1, DEFAULT)
    install(switch, p2, a2, DEFAULT)

def monitor(switch):
    p = {IN_PORT:2, TP_SRC:80}
    install(switch, p, [], DEFAULT)
    query_stats(switch, p)

def repeater_monitor(switch):
    p1 = {IN_PORT:1}
    p2 = {IN_PORT:2}
    p2web = {IN_PORT:2, TP_SRC:80}
    a1 = [output(2)]
    a2 = [output(1)]
    install(switch, p1, a1, DEFAULT)
    install(switch, p2, a2, DEFAULT)
    install(switch, p2web, a2, HIGH)
    query_stats(switch, p2web)
Example w/ Frenetic

```python
def monitor_sf():
    return(Filter(inport_p(2) & srcport_p(80)) | o |
    GroupByTime(30) | o |
    SumSizes())

rules = [Rule(inport_p(1), [output(2)]),
          Rule(inport_p(2), [output(1)])]

def repeater_monitor():
    register_static(rules)
    stats = Apply(Packets(), monitor_sf())
    print_stream(stats)
```
Discussion

• Are there any issues with OpenFlow that Frenetic could not address?

• How does Frenetic reinforce the idea that innovation in this field will come through abstractions and layering?

• Does Frenetic or OpenFlow help address the issue of “ossification” of the internet?
Take Aways

- OpenFlow is the “machine language” of network programming
  - Difficult to program correctly and efficiently
  - Not enough layers of abstraction for programmers

- Frenetic addresses issues with compositibility, low-level interaction, and providing a unified view through the Frenetic run-time system and Frenetic programming language
References


• Open Network Foundation. http://opennetworking.org

