Networking is on the verge of a paradigm shift towards *deep programmability*

Network programmability is attracting tremendous industry interest (and money)

Network programmability is getting traction in many academic communities

Why? It's really a story in 3 stages

>7.7k

# of citations of the original OpenFlow paper (*) in ~10 years

(*) https://dl.acm.org/citation.cfm?id=1355746
Stage 1

**The network management crisis**

These algorithms produce the forwarding state which drives IP traffic to its destination

Operators adapt their network forwarding behavior by configuring each network device individually

Given an existing network behavior induced by a low-level configuration C and a desired network behavior
Adapt C so that the network follows the new behavior

Configuring each element is often done manually, using arcane low-level, vendor-specific “languages”

A single mistyped line is enough to bring down the entire network
It’s not only about the problem of configuring…
the level of complexity in networks is staggering


Complexity + Low-level Management = **Problems**

For a little more than 90 minutes […],

Internet service for millions of users in the U.S. and around the world slowed to a crawl.

The cause was yet another BGP routing leak, a router misconfiguration directing Internet traffic from its intended path to somewhere else.

Someone in Google fat-thumbed a Border Gateway Protocol (BGP) advertisement and sent Japanese Internet traffic into a black hole. […]

The outage in Japan only lasted a couple of hours, but was so severe that […] the country’s Internal Affairs and Communications ministries want carriers to report on what went wrong.

NYSE network operators identified the culprit of the 3.5 hour outage, blaming the incident on a “network configuration issue.”

For a little more than 90 minutes […],

Internet service for millions of users in the U.S. and around the world slowed to a crawl.

The cause was yet another BGP routing leak, a router misconfiguration directing Internet traffic from its intended path to somewhere else.

Someone in Google fat-thumbed a Border Gateway Protocol (BGP) advertisement and sent Japanese Internet traffic into a black hole. […]

The outage in Japan only lasted a couple of hours, but was so severe that […] the country’s Internal Affairs and Communications ministries want carriers to report on what went wrong.

NYSE network operators identified the culprit of the 3.5 hour outage, blaming the incident on a “network configuration issue.”

For a little more than 90 minutes […],

Internet service for millions of users in the U.S. and around the world slowed to a crawl.

The cause was yet another BGP routing leak, a router misconfiguration directing Internet traffic from its intended path to somewhere else.

Someone in Google fat-thumbed a Border Gateway Protocol (BGP) advertisement and sent Japanese Internet traffic into a black hole. […]

The outage in Japan only lasted a couple of hours, but was so severe that […] the country’s Internal Affairs and Communications ministries want carriers to report on what went wrong.

NYSE network operators identified the culprit of the 3.5 hour outage, blaming the incident on a “network configuration issue.”
“Human factors are responsible for 50% to 80% of network outages”

http://bit.ly/2sBJ2jf

“Ironically, this means that data networks work better during week-ends…”

% of route leaks
Monday 10
Tuesday 5
Wednesday 10
Thursday 5
Friday 20
Saturday 10
Sunday 5

source: Job Snijders (NTT)

Internet advertisements rates suggest that
The Internet was more stable than normal on Sept 11

Information suggests that operators were watching the news instead of making changes to their infrastructure

Solving this problem is hard because network devices are completely locked down

“Cost per network outage can be as high as 750,000$”

Smart Management for Robust Carrier Network Health and Reduced TCO!, NANOG54, 2012
Stage 2

Software-Defined Networking

What is SDN and how does it help?

• SDN is a new approach to networking
  – Not about “architecture”; IP, TCP, etc.
  – But about design of network control (routing, TE,...)
• SDN is predicated around two simple concepts
  – Separates the control-plane from the data-plane
  – Provides open API to directly access the data-plane
• While SDN doesn’t do much, it enables a lot

Rethinking the “Division of Labor”

Traditional Computer Networks

Data plane:
Packet processing & delivery
Forward, filter, buffer, mark, rate-limit, and measure packets

Control plane:
Distributed algorithms, establish state in devices
Track topology changes, compute routes, install forwarding rules

Software Defined Networking (SDN)

Smart, slow
Logically-centralized control
API to the data plane (e.g., OpenFlow)

OpenFlow Networks

SDN advantages

• Simpler management
  – No need to “invert” control-plane operations
• Faster pace of innovation
  – Less dependence on vendors and standards
• Easier interoperability
  – Compatibility only in “wire” protocols
• Simpler, cheaper equipment
  – Minimal software
OpenFlow is an API to a switch flow table

- **Simple packet-handling rules**
  - Pattern: match packet header bits, i.e. flowspace
  - Actions: drop, forward, modify, send to controller
  - Priority: disambiguate overlapping patterns
  - Counters: #bytes and #packets

<table>
<thead>
<tr>
<th>Action</th>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>drop</td>
<td>10. src = 1.2.<em>.</em>, dest=3.4.5.*</td>
<td>05. src = 3.4.<em>.</em></td>
</tr>
<tr>
<td>forward(2)</td>
<td>src=10.1.2.3, dest=3.4.<em>.</em></td>
<td>send to controller</td>
</tr>
</tbody>
</table>

OpenFlow switches can emulate different kinds of boxes

- **Router**
  - Match: longest destination IP prefix
  - Action: forward out a link

- **Firewall**
  - Match: IP addresses and TCP/UDP port numbers
  - Action: permit or deny

- **Switch**
  - Match: destination MAC address
  - Action: forward or flood

- **NAT**
  - Match: IP address and port
  - Action: rewrite address and port
Controller: Programmability

while (true):
  read event e:
  if e == switch up:
    - update topology
    - populates switch table
  …

Example OpenFlow Applications

- Dynamic access control
- Seamless mobility/migration
- Server load balancing
- Network virtualization
- Using multiple wireless access points
- Energy-efficient networking
- Adaptive traffic monitoring
- Denial-of-Service attack detection

E.g.: Dynamic Access Control

- Inspect first packet of a connection
- Consult the access control policy
- Install rules to block or route traffic

E.g.: Seamless Mobility/Migration

- See host send traffic at new location
- Modify rules to reroute the traffic

E.g.: Server Load Balancing

- Pre-install load-balancing policy
- Split traffic based on source IP

Heterogeneous Switches

- Number of packet-handling rules
- Range of matches and actions
- Multi-stage pipeline of packet processing
- Offload some control-plane functionality (?)
**Controller Delay and Overhead**

- Controller is much slower than the switch
- Processing packets leads to delay and overhead
- Need to keep most packets in the “fast path”

**Distributed Controller**

- For scalability and reliability
- Partition and replicate state

**Testing and Debugging**

- OpenFlow makes programming possible
  - Network-wide view at controller
  - Direct control over data plane
- Plenty of room for bugs
  - Still a complex, distributed system
- Need for testing techniques
  - Controller applications
  - Controller and switches
  - Rules installed in the switches

**Programming Abstractions**

- OpenFlow is a low-level API
  - Thin veneer on the underlying hardware
- Makes network programming possible, not easy!

**Example: Simple Repeater**

```python
def switch_join(switch):
    # Repeat Port 1 to Port 2
    p1 = {in_port:1}
    a1 = [forward(2)]
    install(switch, p1, DEFAULT, a1)
    # Repeat Port 2 to Port 1
    p2 = {in_port:2}
    a2 = [forward(1)]
    install(switch, p2, DEFAULT, a2)
```

When a switch joins the network, install two forwarding rules.

**Example: Web Traffic Monitor**

```python
def switch_join(switch):
    # Web traffic from Internet
    p = {inport:2,tp_src:80}
    install(switch, p, DEFAULT, [])
    query_stats(switch, p)

    def stats_in(switch, xid, pattern, packets, bytes):
        print bytes
        sleep(30)
        query_stats(switch, pattern)
```

When a switch joins the network, install one monitoring rule.

**Composition: Repeater + Monitor**

```python
def switch_join(switch):
    pat1 = {inport:1}
    pat2 = {inport:2}
    pat2web = {in_port:2, tp_src:80}
    install(switch, pat1, DEFAULT, None, [forward(2)])
    install(switch, pat2web, HIGH, None, [forward(1)])
    install(switch, pat2, DEFAULT, None, [forward(1)])
    query_stats(switch, pat2web)

    def stats_in(switch, xid, pattern, packets, bytes):
        print bytes
        sleep(30)
        query_stats(switch, pattern)
```

**Asynchrony: Switch-Controller Delays**

- Common OpenFlow programming idiom
  - First packet of a flow goes to the controller
  - Controller installs rules to handle remaining packets
- What if more packets arrive before rules installed?
  - Multiple packets of a flow reach the controller
- What if rules along a path installed out of order?
  - Packets reach intermediate switch before rules do

**Must think about all possible event orderings.**
**Better: Increase the level of abstraction**

- **Separate reading from writing**
  - Reading: specify queries on network state
  - Writing: specify forwarding policies
- **Compose multiple tasks**
  - Write each task once, and combine with others
- **Prevent race conditions**
  - Automatically apply forwarding policy to extra packets
- **See http://frenetic-lang.org/**

---

**OpenFlow is not all roses**

- **The protocol is too complex**
  - (12 fields in OF 1.0 to 41 in 1.5)
  - switches must support complicated parsers and pipelines
- **The specification itself keeps getting more complex**
  - extra features make the software agent more complicated
- **Consequences**
  - Switches vendor end up implementing parts of the spec. which breaks the abstraction of one API to rule-them-all

---

**Stage 3**

**Deep Network Programability**

Enters… **Protocol Independent Switch Architecture and P4**

Protocol Independent Switch Architecture (PISA) for high-speed programmable packet forwarding

Enters… **Protocol Independent Switch Architecture and P4**

A slightly more accurate architecture
En ters… Protocol Independent Switch Architecture and P4

P4 is a domain-specific language which describes how a PISA architecture should process packets

https://p4.org

PISA + P4 is strictly more general OpenFlow

This course will introduce you to the emerging area of network programmability

Learn the principles of network programmability at the control-plane and at the data-plane level

Get fluent in P4 programming: the go-to language for programming data planes

Get insights into hard, research-level problems and how programmability can help solving them
The course is gonna be divided in two 7-weeks blocks

<table>
<thead>
<tr>
<th>Lectures/Exercises</th>
<th>Group project</th>
</tr>
</thead>
<tbody>
<tr>
<td>~7 weeks</td>
<td>&gt;= 7 weeks</td>
</tr>
<tr>
<td>how to program in P4</td>
<td>in teams of 2—3</td>
</tr>
</tbody>
</table>

There will be 2h of lectures & 2h of exercises

Thu 8—10 Lecture (for 7 weeks)
Thu 10—12 Practical exercises with P4
Exercises are not graded but will help at the exam
For now, both will take place in LFW B 3

For the project, we’ll ask you to develop your own network application

You can choose your application
we’ll provide feedback and a list of default choices
We’ll provide feedback and assist you throughout during the lecture slot and/or online
Grade will depend on the code, report and presentation presentations during the last week of the lecture

You’ll have the opportunity to port your application on real hardware (not mandatory… if you’re motivated :-))

Barefoot Tofino Wedge 100BF-32X 3.2 Tbps

Your final grade

<table>
<thead>
<tr>
<th>Exam</th>
<th>Group project</th>
</tr>
</thead>
<tbody>
<tr>
<td>50% oral</td>
<td>50% code, report, and presentation</td>
</tr>
</tbody>
</table>

Examples

Design a P4 application for solving problem <X>
Optimize program <X>
Is program <X> correct?
… important to do the exercises
Your dream team for the semester

Edgar Roland Thomas Maria

Our website: https://adv-net.ethz.ch/
check it out regularly

We'll use Slack (chat client) to discuss about the course, exercises, and projects

Register today using your real name
> https://adv-net18.slack.com/signup

It depends…

You shouldn't take the course if…
- you hate programming
- you don't want to work during the semester
- you expect 10+ years of exam history

Besides that, if you like networking… go for it!

All of the assignments (and the course) will be new, meaning you will act as guinea pigs…

We'll try to take your feedback into account… so shoot!
Let’s look at one example.

IP forwarding in a traditional router

How can we do this in P4?

A P4 program consists of three basic parts

Parser | Match-Action Pipeline | Deparser

Programmer declares the headers that should be recognized and their order in the packet.

Programmer defines the tables and the exact processing algorithm.

Programmer declares how the output packet will look on the wire.
The parser uses a state machine to map packets into headers and metadata:

Parser Match-Action Pipeline Deparser

The parser has three predefined states: start, accept and reject:

Packet Headers and metadata

Control

Similar to functions in C:
- declare variables
- create tables
- describe control flow
- ...
Basic building blocks of P4 programs

- Control flow: similar to C but without loops
- Actions: similar to functions in C
- Tables: match a key and return an action

Controls can apply changes to packets

```c
control MyIngress(inout headers hdr, 
                  inout metadata meta, 
                  inout standard_metadata_t std_meta) {
  bit<9> port;
  apply {
    port = 1
    std_meta.egress_spec = port;
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = 0x2;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
  }
}
```

Actions allow to re-use code

```c
control MyIngress(inout headers hdr, 
                  inout metadata meta, 
                  inout standard_metadata_t std_meta) {
  action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
    std_meta.egress_spec = port;
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = dstAddr;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
  }
  apply {
    ipv4_forward(0x123, 1);
  }
}
```

Control Plane

- Table name
- Field(s) to match
- Possible actions
- Max. # entries in table
- Default action

Field(s) to match
- Match type
- Possible actions
Example: IP forwarding table

```
1.2.3.4  1.2.3.5  1.2.3.254
LAN 1

1.2.3.0/24  5.6.7.0/24
...
LAN 2

01:01:01:01:01:01
02:02:02:02:02:02
```
The Deparser assembles the headers back into a well-formed packet.