Software-Defined Networking Architecture
Brighten Godfrey
CS 538 February 27 2017
The Problem

Networks are complicated

- Just like any computer system
- Worse: it’s distributed
- Even worse: no clean programming APIs, only “knobs and dials”
The Problem

Networks are complicated

- Just like any computer system
- Worse: it’s distributed
- Even worse: no clean programming APIs, only “knobs and dials”

Network equipment is proprietary

- Integrated solutions (software, configuration, protocol implementations, hardware) from major vendors

Result: Hard to innovate and modify networks
hostname bgpd4
password zebra

router bgp 8000
  bgp router-id 10.1.4.2

! for the link between A and B
  neighbor 10.1.2.3 remote-as 8000
  neighbor 10.1.2.3 update-source 10.0.0.0
  network 10.0.0.0/7

! for the link between A and C
  neighbor 10.1.3.3 remote-as 7000
  neighbor 10.1.3.3 ebgp-multihop
  neighbor 10.1.3.3 next-hop-self
  neighbor 10.1.3.3 route-map PP out

! for link between A and D
  neighbor 10.1.4.3 remote-as 8000
  neighbor 10.1.4.3 ebgp-multihop
  neighbor 10.1.4.3 next-hop-self
  neighbor 10.1.4.3 route-map TagD in

! route update filtering
  ip community-list 1 permit 8000:1000

!
Traditional network

- device software
- protocols
- device software
- protocols
- device software
- protocols
- device software
- protocols
- device software
- protocols
- device software
- protocols
Here that the PC be validl The PV replies with the PCx where the form as R$A_l$ The client then sends a request to the PV of and have each generated a publicmprivate key pair $e$ in Figure rl Before the process beginsi the client and PV questsl

and handshake with an appropriate PV for future rek PV the server does not trusti it can fall back to a qWH third partiesi can exist in parallell If the client uses a not unprecedentedx certificate authorities and the root is that servers need to trust a third partyl But this is delay for each new server or domainl The disadvantage The advantage is that a client can avoid paying an RTT web sites or content distribution networksl

a single RTTl This will be highly e RTTs eas in TCPfi and subsequent connections require the server directlyl Thusi the first connection takes two

PC is the duration for which the client requests provenance certificate (PC)

K

priv

pv

is the time the client sent the data hash

m

c

sub

m

c

t

c

is the length of time

c

d

c

d

c

and the

c

d

c

app

app

software abstractions

“Network OS”

Logically centralized controller

Data plane API

---

Here that the PC be validl The PV replies with the PCx where the form as R$A_l$ The client then sends a request to the PV of and have each generated a publicmprivate key pair $e$ in Figure rl Before the process beginsi the client and PV questsl

and handshake with an appropriate PV for future rek PV the server does not trusti it can fall back to a qWH third partiesi can exist in parallell If the client uses a not unprecedentedx certificate authorities and the root is that servers need to trust a third partyl But this is delay for each new server or domainl The disadvantage The advantage is that a client can avoid paying an RTT web sites or content distribution networksl

a single RTTl This will be highly e RTTs eas in TCPfi and subsequent connections require the server directlyl Thusi the first connection takes two

PC is the duration for which the client requests provenance certificate (PC)

K

priv

pv

is the time the client sent the data hash

m

c

sub

m

c

t

c

is the length of time

c

d

c

d

c

and the

c

d

c

app

app

software abstractions

“Network OS”

Logically centralized controller

Data plane API
On user authentication, statically setup VLAN tagging rules at the user’s first hop switch

```python
def setup_user_vlan(dp, user, port, host):
    vlanid = user_to_vlan_function(user)

    # For packets from the user, add a VLAN tag
    attr_out[IN_PORT] = port
    attr_out[DL_SRC] = nox.reverse_resolve(host).mac
    action_out = [(nox.OUTPUT, (0, nox.FLOOD)),
                  (nox.ADD_VLAN, (vlanid))]
    install_datapath_flow(dp, attr_out, action_out)

    # For packets to the user with the VLAN tag, remove it
    attr_in[DL_DST] = nox.reverse_resolve(host).mac
    attr_in[DL_VLAN] = vlanid
    action_in = [(nox.OUTPUT, (0, nox.FLOOD)), (nox.DEL_VLAN)]
    install_datapath_flow(dp, attr_in, action_in)

nox.register_for_user_authentication(setup_user_vlan)
```

From NOX [Gude, Koponen, Pettit, Pfaff, Casado, McKeown, Shenker, CCR 2008]
Example

From NOX [Gude, Koponen, Pettit, Pfaff, Casado, McKeown, Shenker, CCR 2008]

# On user authentication, statically setup VLAN tagging
# rules at the user’s first hop switch

```python
def setup_user_vlan(dp, user, port, host):
    vlanid = user_to_vlan_function(user)

    # For packets from the user, add a VLAN tag
    attr_out[IN_PORT] = port
    attr_out[DL_SRC] = nox.reverse_resolve(host).mac
    action_out = [(nox.OUTPUT, (0, nox.FLOOD)),
                  (nox.ADD_VLAN, (vlanid))]
    install_datapath_flow(dp, attr_out, action_out)

    # For packets to the user with the VLAN tag, remove it
    attr_in[DL_DST] = nox.reverse_resolve(host).mac
    attr_in[DL_VLAN] = vlanid
    action_in = [(nox.OUTPUT, (0, nox.FLOOD)), (nox.DEL_VLAN)]
    install_datapath_flow(dp, attr_in, action_in)

nox.register_for_user_authentication(setup_user_vlan)
```
Example

From NOX [Gude, Koponen, Pettit, Pfaff, Casado, McKeown, Shenker, CCR 2008]

```python
# On user authentication, statically setup VLAN tagging
# rules at the user’s first hop switch

def setup_user_vlan(dp, user, port, host):
    vlanid = user_to_vlan_function(user)

    # For packets from the user, add a VLAN tag
    attr_out[IN_PORT] = port
    attr_out[DL_SRC] = nox.reverse_resolve(host).mac
    action_out = [(nox.OUTPUT, (0, nox.FLOOD)),
                  (nox.ADD_VLAN, (vlanid))]
    install_datapath_flow(dp, attr_out, action_out)

    # For packets to the user with the VLAN tag, remove it
    attr_in[DL_DST] = nox.reverse_resolve(host).mac
    attr_in[DL_VLAN] = vlanid
    action_in = [(nox.OUTPUT, (0, nox.FLOOD)), (nox.DEL_VLAN)]
    install_datapath_flow(dp, attr_in, action_in)

    nox.register_for_user_authentication(setup_user_vlan)
```
# On user authentication, statically setup VLAN tagging
# rules at the user’s first hop switch

def setup_user_vlan(dp, user, port, host):
    vlanid = user_to_vlan_function(user)

    # For packets from the user, add a VLAN tag
    attr_out[IN_PORT] = port
    attr_out[DL_SRC] = nox.reverse_resolve(host).mac
    action_out = [(nox.OUTPUT, (0, nox.FLOOD)),
                  (nox.ADD_VLAN, (vlanid))]
    install_datapath_flow(dp, attr_out, action_out)

    # For packets to the user with the VLAN tag, remove it
    attr_in[DL_DST] = nox.reverse_resolve(host).mac
    attr_in[DL_VLAN] = vlanid
    action_in = [(nox.OUTPUT, (0, nox.FLOOD)), (nox.DEL_VLAN)]
    install_datapath_flow(dp, attr_in, action_in)

nox.register_for_user_authentication(setup_user_vlan)
# On user authentication, statically setup VLAN tagging
# rules at the user’s first hop switch

def setup_user_vlan(dp, user, port, host):
    vlanid = user_to_vlan_function(user)

    # For packets from the user, add a VLAN tag
    attr_out[IN_PORT] = port
    attr_out[DL_SRC] = nox.reverse_resolve(host).mac
    action_out = [(nox.OUTPUT, (0, nox.FLOOD)),
                  (nox.ADD_VLAN, (vlanid))]
    install_datapath_flow(dp, attr_out, action_out)

    # For packets to the user with the VLAN tag, remove it
    attr_in[DL_DST] = nox.reverse_resolve(host).mac
    attr_in[DL_VLAN] = vlanid
    action_in = [(nox.OUTPUT, (0, nox.FLOOD)), (nox.DEL_VLAN)]
    install_datapath_flow(dp, attr_in, action_in)

nox.register_for_user_authentication(setup_user_vlan)
4.2.2 Obtaining a Provenance Certificate

Once the client has a current PC, it can contact a server using the Lifesaver protocol. The obvious implementation of the above exchange is to use a Request Certificate (RC) because while it doesn't prove provenance, it does make connections much more efficient and robust to distributed denial of service attacks. For example, if the server observes the same RC, it can be confident the request comes from the same client identifier even if the client identifier changes.

Lifesaver leverages cryptographic proof to verify the provenance of client requests without requiring an RTT delay on every connection. First, the client handshake with an appropriate PV for future requests. The server does not trust the third parties that exist in parallel. If the client uses a DNS server, it can fall back to a third party, but this is not unprecedented. Certificate authorities and the root can exist in parallel.

The advantage of this design over TCP is that the PV can prove provenance via TCP's sequence number, which is the application-level data such as the duration for which the client requests the server, the source address of the client, and the length of time between the request and the server. The disadvantage is that servers need to trust a third party, but this is not unprecedented. Certificate authorities and the root can exist in parallel.

The client then sends a request to the PV of the server. The obvious implementation of the above exchange is to use a Request Certificate (RC) because while it doesn't prove provenance, it does make connections much more efficient and robust to distributed denial of service attacks. For example, if the server observes the same RC, it can be confident the request comes from the same client identifier even if the client identifier changes.
Label switching / MPLS (1997)

- Set up explicit paths for classes of traffic

Active Networks (1999)

- Packet header carries (pointer to) program code
Logically Centralized Control

Routing Control Platform (2005)

- [Caesar, Caldwell, Beamster, Rexford, Shaikh, van der Merwe, NSDI 2005]
- Centralized computation of BGP routes, pushed to border routers via iBGP
Logically Centralized Control

Routing Control Platform (2005)

4D architecture (2005)

- A Clean Slate 4D Approach to Network Control and Management [Greenberg, Hjalmtysson, Maltz, Myers, Rexford, Xie, Yan, Zhan, Zhang, CCR Oct 2005]
- Logically centralized “decision plane” separated from data plane
Logically Centralized Control

Routing Control Platform (2005)

4D architecture (2005)

Ethane (2007)

- [Casado, Freedman, Pettit, Luo, McKeown, Shenker, SIGCOMM 2007]
- Centralized controller enforces enterprise network Ethernet forwarding policy using existing hardware
Routing Control Platform (2005)

4D architecture (2005)

Ethane (2007)

- [Casado, Freedman, Pettit, L
SIGCOMM 2007]

- Centralized controller enforces Ethernet forwarding policy

---

**Figure 4: A sample policy file using Pol-Eth**

```
# Groups —
desktops = [“griffin”, “roo”];
laptops = [“glaptop”, “rlaptop”];
phones = [“gphone”, “rphone”];
server = [“http_server”, “nfs_server”];
private = [“desktops”, “laptops”];
computers = [“private”, “server”];
students = [“bob”, “bill”, “pete”];
profs = [“plum”];
group = [“students”, “profs”];
waps = [“wap1”, “wap2”];

# Rules —
[(hsrc=in(“server”))∧(hdst=in(“private”))]: deny;
# Do not allow phones and private computers to communicate
[(hsrc=in(“phones”))∧(hdst=in(“computers”))]: deny;
[(hsrc=in(“computers”))∧(hdst=in(“phones”))]: deny;
# NAT-like protection for laptops
[(hsrc=in(“laptops”))]: outbound-only;
# No restrictions on desktops communicating with each other
[(hsrc=in(“desktops”))∧(hdst=in(“desktops”))]: allow;
# For wireless, non-group members can use http through
# a proxy. Group members have unrestricted access.
[(apsrc=in(“waps”))∧(user=in(“group”))]: allow;
[(apsrc=in(“waps”))∧(protocol=“http”): waypoints(“http-proxy”);
[(apsrc=in(“waps”))]: deny;
]: allow; # Default-on: by default allow flows
```
Routing Control Platform (2005)

4D architecture (2005)

Ethane (2007)

OpenFlow (2008)

- [McKeown, Anderson, Balakrishnan, Parulkar, Peterson, Rexford, Shenker, Turner, CCR 2008]
- Thin, standardized interface to data plane
- General-purpose programmability at controller
Evolution of SDN:

Routing Control Platform (2005)

4D architecture (2005)

Ethane (2007)

OpenFlow (2008)

NOX (2008)

- [Gude, Koponen, Pettit, Pfaff, Casado, McKeown, Shenker, CCR 2008]
- First OF controller: centralized network view provided to multiple control apps as a database
- Behind the scenes, handles state collection & distribution
Evolution of SDN

Industry explosion (~2010+)
Opportunities

Open data plane interface

- Hardware: Easier for operators to change hardware, and for vendors to enter market
- Software: Can more directly access device behavior

Centralized controller

- Direct programmatic control of network

Software abstractions on the controller

- Solve dist. sys. problems once, then just write algorithms
- Libraries/languages to help programmers write net apps
- Systems to write high level policy instead of programming
Opportunities

Open data plane interface

- Hardware: easier for operators to change hardware, and for vendors to enter market
- Software: can finally directly access device behavior

Centralized controller

- Direct programmatic control of network

Software abstractions on the controller

- Solve distributed systems problems only once, then just write algorithms
- Libraries/languages to help programmers

All active areas of current research!
Challenges for SDN

Performance and scalability

Distributed system challenges still present

- Resilience of “logically centralized” controller
- Imperfect knowledge of network state
- Consistency issues between controllers
Challenges for SDN

Reaching agreement on data plane protocol

• OpenFlow? NFV functions? Whitebox switching? Programmable data planes?

Devising the right control abstractions

• Programming OpenFlow: far too low level
• But what are the right high-level abstractions to cover important use cases?
Q: When do you control the net?

When does the SDN controller send instructions to switches?

• ...in the OpenFlow paper?
• ...other options?
Q: When do you control the net?

When does the SDN controller send instructions to switches?

- ...in the OpenFlow paper? **Reactive** (when packet arrives needing forwarding rule)
- ...other options? **Proactive** (in advance of need)
Q: How does SDN affect reliability?

More bugs in the network, or fewer?
Separate interfaces:

- Host-network (external-to-internal data plane)
- Operator-network
- Packet-switch (internal data plane)
Q: “Host-Network and Packet-Switch interfaces were identical” in the Internet. How is this a simplification?

Q: Does OF meet the net’s goals of:

- Simplified hardware
- Vendor-neutral hardware
- “Future-proof” hardware
- Flexible software
Q: Drivers of early deployment?

What drove early deployment of OpenFlow & SDN?

Access control in enterprises? Net research?

- Good ideas, are already valuable
- But not the “killer apps” for initial large-scale deployment
The first “Killer Apps” for SDN

Inter-datacenter traffic engineering

• Drive utilization to near 100% when possible
• Protect critical traffic from congestion

Cloud virtualization

• Create separate virtual networks for tenants
• Allow flexible placement and movement of VMs

Key characteristics of the above use cases

• Special-purpose deployments with less diverse hardware
• Existing solutions aren’t just inconvenient, they don’t work!
Next up

Wednesday: SDN in the WAN

Monday: SDN in the virtualized data center