Managing Networks in a Software-Defined Future

Jeff Gehlbach

Southern California Linux Expo 2015 (SCaLE13x)

February 23, 2015

Agenda

- Speaker vitals
- Elements of old-school networks
- Elements of software-defined networks
- A case study
- Conclusions
- Questions!

Jeff Gehlbach

Ten fingers, ten toes, some industry experience

- NASA NISN → Management of large IP networks
- Empire / Concord → Making and consulting on NMS
- BellSouth → Cranium formed into Bell shape
- OpenNMS Group → Making and consulting on *free* NMS!

Elements of old-school networks

Elements of old-school networks

Switches (oversimplified)

- Functions:
  - Switching L2 frames
  - Running STP
- Many physical ports, often modular
- High-throughput data plane
- Control plane driven by local config (!)

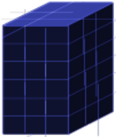
Elements of old-school networks

Routers (oversimplified)

- Functions:
  - Forwarding L3 packets
  - Running OSPF, BGP, *et al*
- Relatively few physical ports, often modular
- Medium- to high-throughput data plane
- Control plane driven by local config (!!)

Elements of old-school networks

Firewalls (oversimplified)



- Functions:
  - Forwarding L3 packets subject to a ruleset
  - Taking blame when anything breaks
- Relatively few physical ports, sometimes modular
- Low- to medium-throughput data plane
- Control plane driven by local config (!!!)

Elements of old-school networks

Inventory and configuration management

- Functions:
  - Making the right configs run on the right devices
  - Accounting for hardware elements in the network
  - Eating time and / or money
- Two separate problems really, each pretty hard
- Typically no understanding of configurations (control plane)
- Are all your nodes in your inventory?

Image: graemefazakerley / DeviantArt / CC BY-SA 3.0

Elements of old-school networks

Network management system (NMS)

- Functions – OSI FCAPS model:
    - **F**ault management*
    - **C**onfiguration management
    - **A**ccounting management
    - **P**erformance management*
    - **S**ecurity management
- OpenNMS adheres roughly to FCAPS
- Focus on fault (FM) and performance (PM)

Elements of old-school networks

FM and PM as implemented in OpenNMS

- **Provisioning** – how can we get nodes, interfaces, services into the system?
- **Service assurance** – how can we know whether important network entities are responsive?
- **Fault management** – how can a network element tell us it has a problem?
- **Performance management** – how can we quantify utilization of a network interface or a CPU?

Elements of old-school networks

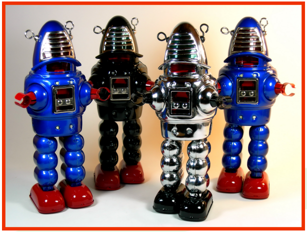
Simple Network Management Protocol (SNMP)

- Functions:
  - NMS-to-managed-node data queries (GET / GET-BULK)
  - Managed-node-to-NMS unsolicited messages (TRAP)
- Routers, switches, *et al* are where the action happens
- The NMS talks to the SNMP agent on the managed node
- Data gathered: interface traffic, BGP statistics, environmentals...
- Extensible via Management Information Base (MIB)

Image: tedeytan / Wikimedia Commons / CC SA-2.0 Generic

Elements of old-school networks

In summary



- Many sovereign nodes with local configs driving control plane
- When we’re lucky, traffic flows as intended
- Impossible to simulate accurately
- Clearly not designed by hackers

Image: D J Shin - My Toy Museum / Wikimedia Commons / CC BY-SA 3.0 Unported

Elements of software-defined networks

Elements of software-defined networks

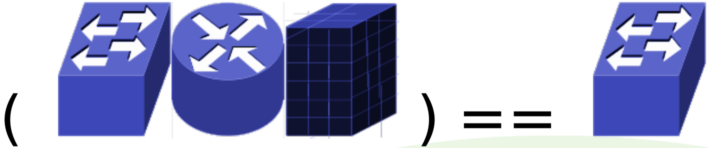
Separation of planes is key!

If you take away just one SDN fact:

SDN is about **separation** of **control plane** from **data plane**; and **programmability**.

Elements of software-defined networks

Data plane



- Functions:
  - Moving frames or packets around
  - According to rules gotten from controller (“control plane”)
- Comparatively generic hardware
- Sometimes virtual
- Called “switches” regardless of role

Elements of software-defined networks

Control plane

- Functions:
  - Control behavior of switches (“data plane”)
    - According to centrally-managed rules (eases config)
    - Across registered nodes (eases inventory)
  - Expose inventory, configuration, *etc.* via open APIs
  - Scripting hooks for network programmability
- Controller is just a general-purpose computer
- May have a bridge or flower tattooed on it
- May be virtual

Elements of software-defined networks

In summary

- Relatively dumb switches
- Switch inventory, configurations centrally managed
- Programmability enables awesome wackiness, agility
- When we’re skilled, traffic flows as intended
- Might even be unit-testable
- This is how hackers would build a network!

Image: Andreas Trepte / Wikimedia Commons / CC SA 2.5 Generic

A case study

Controller: Project Floodlight

- Implements OpenFlow 1.0 – 1.4
- Apache-licensed
- Maintained by Big Switch Networks
- projectfloodlight.org

Switches: Open vSwitch / Fedora 21

- Implements OpenFlow 1.3
- Apache-licensed
- Distributed maintainership
- Kernelspace implementation in Linux, FreeBSD
- Userspace implementation in NetBSD
- openvswitch.org

Provisioning / Inventory

SDN for provisioning

Provisioning: SDN controller as a source of truth

- Option 1: Push inventory from controller to OpenNMS API
    - Leans on SDN controller's internal programmability
- Option 2: Pull inventory from controller's API
    - Leans on SDN controller's API

SDN for provisioning

Floodlight → OpenNMS

Option 1: Push-mode

SDN for provisioning

Floodlight → OpenNMS

- Floodlight features pluggable notification managers

```java
public interface INotificationManager {
    /**
     * Post a notification. Depending on the underline implementation, it
     * may write the notes to log file or send an SNMP notification/trap.
     *
     * @param notes   string message to be sent to receiver
     */
    public void postNotification(String notes);
}
```

```java
    private static class NotificationSwitchListener implements IOFSwitchListener {
        // ...
        @Override
        public void switchAdded(DatapathId switchId) {
            notifier.postNotification("Switch " +  switchId + " connected.");
        }
        // ...
    }
```

SDN for provisioning

Floodlight → OpenNMS

- Default implementation just squawks to syslog
  - Write a new one that POSTs to OpenNMS requisition ReST endpoint
  - Or just watch logs from outside, do the POST from there
- Doesn’t seem the cleanest approach, but should be effective
- Floodlight / other SDN controller hackers, comments?

SDN for provisioning

Floodlight → OpenNMS

Option 2: Pull-mode

SDN for provisioning

OpenNMS ← Floodlight

- Query Floodlight's `core/controller/switches` endpoint

```
// Output of http://mal:8080/wm/core/controller/switches/json
[
  {
    // Switch "wash"
    "inetAddress": "/10.0.0.138:45261",
    "connectedSince": 1424451598399,
    "switchDPID": "00:00:26:09:6a:ae:e3:49"
  },
  {
    // Switch "zoe"
    "inetAddress": "/10.0.0.57:35907",
    "connectedSince": 1424453016500,
    "switchDPID": "00:00:d2:0b:68:3a:d2:49"
  }
]
```

SDN for provisioning

OpenNMS ← Floodlight

- Query Floodlight's `core/switch/<DPID>` endpoint

```
// Output of http://mal:8080/wm/core/switch/00:00:d2:0b:68:3a:d2:49/desc/json
// This is "zoe"
{
  "desc": {
    "version": "OF_13",
    "manufacturerDescription": "Nicira, Inc.",
    "hardwareDescription": "Open vSwitch",
    "softwareDescription": "2.3.1-git3282e51",
    "serialNumber": "None",
    "datapathDescription": "None"
  }
}
```

- A bit short on details, but that's on Open vSwitch
- Anybody with Nexus, Arista, *etc.* gear see better data?

SDN for provisioning

OpenNMS ← Floodlight

- Query Floodlight's `/core/switch/<DPID>/port-desc` endpoint

```
// Output of http://mal:8080/wm/core/switch/00:00:d2:0b:68:3a:d2:49/port-desc/json
// This is switch "zoe"
{
  "version": "OF_13",
  "portDesc": [
    {
      "portNumber": "1",
      "hardwareAddress": "06:4e:04:ca:b5:70",
      "name": "eth1",
      "config": "1",
      "state": "1", // ...
      "currSpeed": "1000000",
      "maxSpeed": "10000000"
    },
    {
      "portNumber": "local",
      "hardwareAddress": "d2:0b:68:3a:d2:49",
      "name": "br-int", // ...
    }
  ]
}
```

SDN for provisioning

OpenNMS ← Floodlight

- Build a requisition (PRIS source plugin)
  - Foreign-ID = DPID

```xml
<?xml version="1.0"?>
<model-import foreign-source="floodlight-switches">
  <node node-label="wash" foreign-id="00:00:26:09:6a:ae:e3:49">
    <interface ip-addr="10.0.0.138" descr="" status="1" snmp-primary="P" />
  </node>
  <node node-label="zoe" foreign-id="00:00:d2:0b:68:3a:d2:49">
    <interface ip-addr="10.0.0.57" descr="" status="1" snmp-primary="P" />
  </node>
</model-import>
```

Service assurance

SDN for network management

Service assurance (“are the switches up?”)

- Controller exposes presence / absence of switches
- Most other measures best done through synthetic transactions directly to switches
- Seems not much will change soon in this facet

Fault

SDN for network management

Fault management ("ZOMG a switch broke!")

- Controller able to send unsolicited messages to an NMS
- Similar in function to SNMP traps
- Examples
  - "Switch 00:00:00:00:de:ad:be:ef joined the controller"
  - "Switch 00:00:00:00:ca:fe:ca:fe left without saying goodbye"
- Not yet well-developed in main Floodlight code base
  - Downstream OEMs may provide their own `NotificationManagers`
- OpenNMS can reparent data onto the correct node (switch) via its Event Translator facility

Performance

SDN for network management

Performance management (“How busy is that switch interface?”)

- Floodlight exposes interface-level metrics and other stuff via ReST
- OpenNMS can collect performance data directly via ReST using XMLCollector with JSON handler
- Data trivially reparented onto the correct node (switch)

SDN for network management

Performance management (“How busy is that switch interface?”)

- Query Floodlight’s `/core/switch/<DPID>/port` endpoint

```
// Output of http://mal:8080/wm/core/switch/00:00:d2:0b:68:3a:d2:49/port/json
// This is switch "zoe"
{
  "version": "OF_13",
  "port": [
    {
      "portNumber": "1",
      "receivePackets": "5213610",
      "transmitPackets": "2947725",
      "receiveBytes": "2855576667",
      "transmitBytes": "2354303692",
      "receiveDropped": "0",
      "transmitDropped": "0",
      "receiveErrors": "0",
      "transmitErrors": "0",
      "receiveFrameErrors": "0",
      "receiveOverrunErrors": "0",
      "receiveCRCErrors": "0",
      "collisions": "0", // ...
    }, // ...
  ]
}
```

SDN for network management

Performance management (“How busy is that switch interface?”)

But...

SDN for network management

Performance management (“How busy is that switch interface?”)

- Scalability of ReST / JSON-based collection to huge networks is unproven
- Most SDN switches on the market also support SNMP
- Every NMS in the world groks SNMP already
  - Prediction: Gradual transition from SNMP to controller API
  - Consistency across controller APIs is key

Conclusions

Conclusions

How OpenNMS is coping

- It’s still early days for SDN on the ground
  - Standards landscape frequently changing
  - Most deployments we see are hybrid
- We’ve had some practice with similar movements
- Work on SDN full time? Let’s chat over a beer.

Is SNMP finally dead?

Predicted since late 1990s or earlier

- Not yet. Sorry.
- Problems? Sure.
  - Painful to implement
  - SMI struggles to model really complex relationships
  - Stateless nature increasingly problematic with larger data sets
- Still useful, though
- Entrenchment + utility = durability

Questions!

`jeffg@opennms.org`

IRC: Freenode `jeffg` / `#opennms`

Twitter: `@jeffgdotorg`