A quick look at SDN

Zheng Luo

November 21, 2016

Table of contents

Why SDN

Traditional network

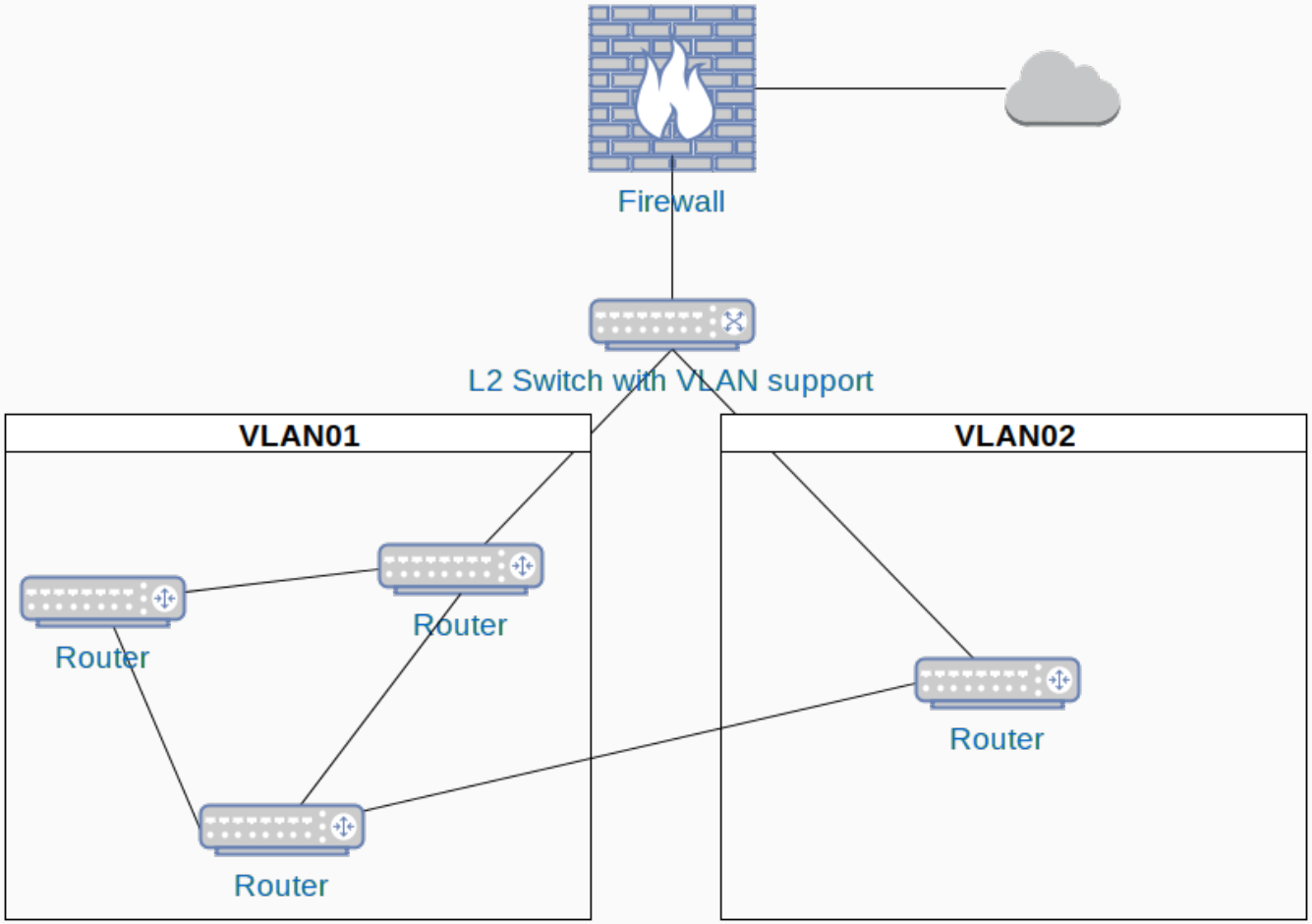

Figure 1: Traditional network

Problem 1: Flexibility

- L2 switch forwards by MAC Address Table(target MAC, etc.)
- Router forwards by routing table(target IP, etc.)
- Firewall filters different types of packets with builtin policies

Questions:

- Can router restrict the speed of video streaming?
- Can switch perform customlized forwarding policies?
- Can we change the builtin policy of firewall dynamicly?

Answer:

Hard. Because they are implemented as hardware. Those come with many functions often have high price.

Problem 2: distributed algorithm/locality

Many distributed algorithms run between neighbors.

Pros:

- Resilient to network change

Cons(like many other distributed algorithms):

- Complexity
- Convergence speed
- Optimality

Extract the complexity

- - L2 switch *forwards* by MAC Address Table with *matching*
  - and *updates* MAC Address table with its own algorithm
- - Router *forwards* by routing table *matching*
  - and *updates* routing table by routing algorithm
- - Firewall *filters*(= forward to no port) by rule *matching*
  - and *updates* rule by configuration

Here we divide the job of network devices into two categories:

Data plane Simple *matching* and *forward*

Control plane Controls the rules that data plane use

Traditional network couples the two planes together. Thus control plane is distributed and often built with hardware.

Initiative

Data plane Often common among devices.

Control plane Quite different among devices.

- Make specification about data plane
- Make control plane programmable

What is SDN

SDN Architecture

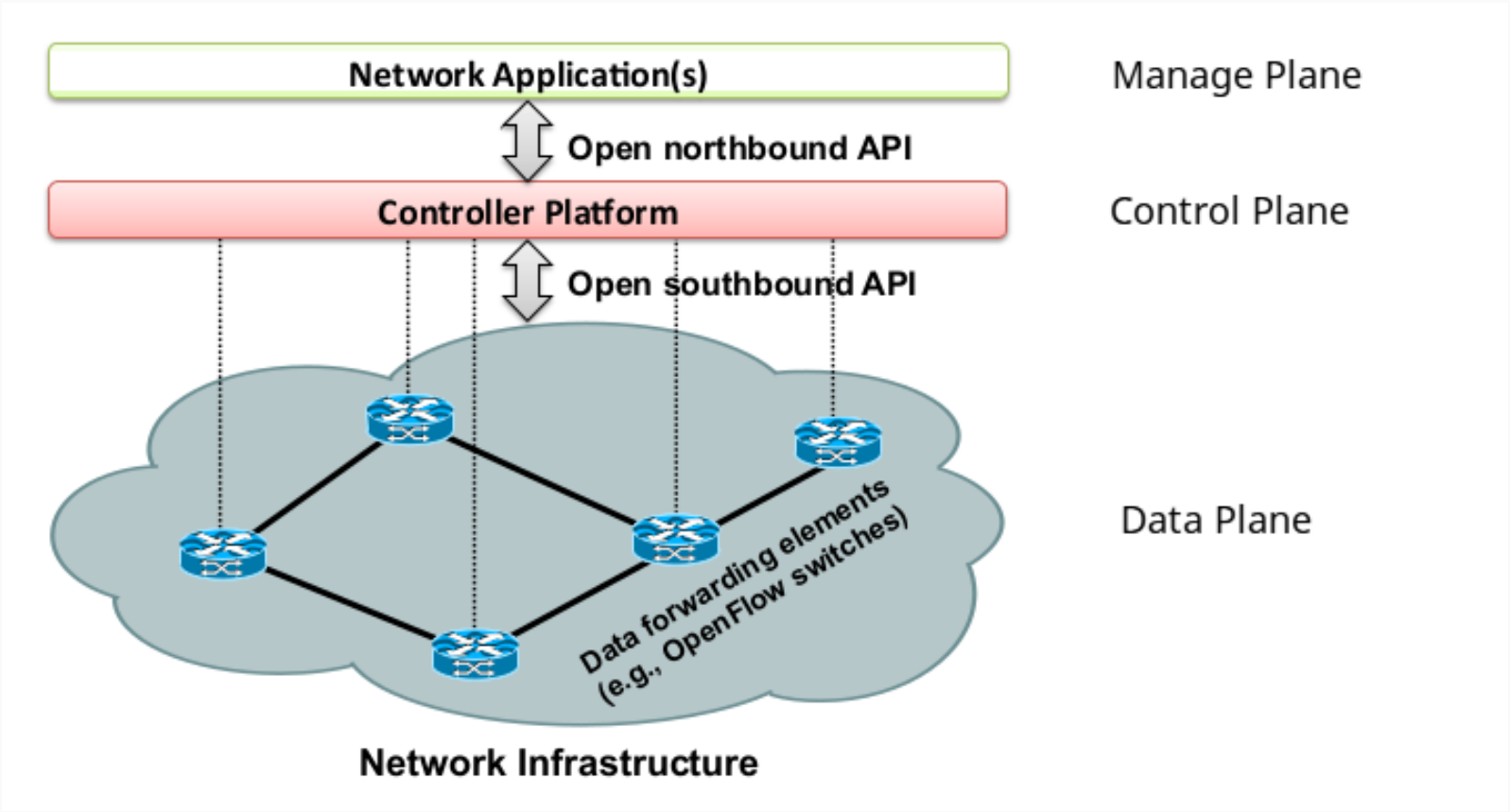

Figure 2: Architecture of SDN

SDN vs traditional network

	SDN	**traditional network**
Control & data plane	Decoupled	Coupled
Control Algorithm	Centralized	Distributed
Programmability	Good	Poor
Programming interface	Open	Private

OpenFlow

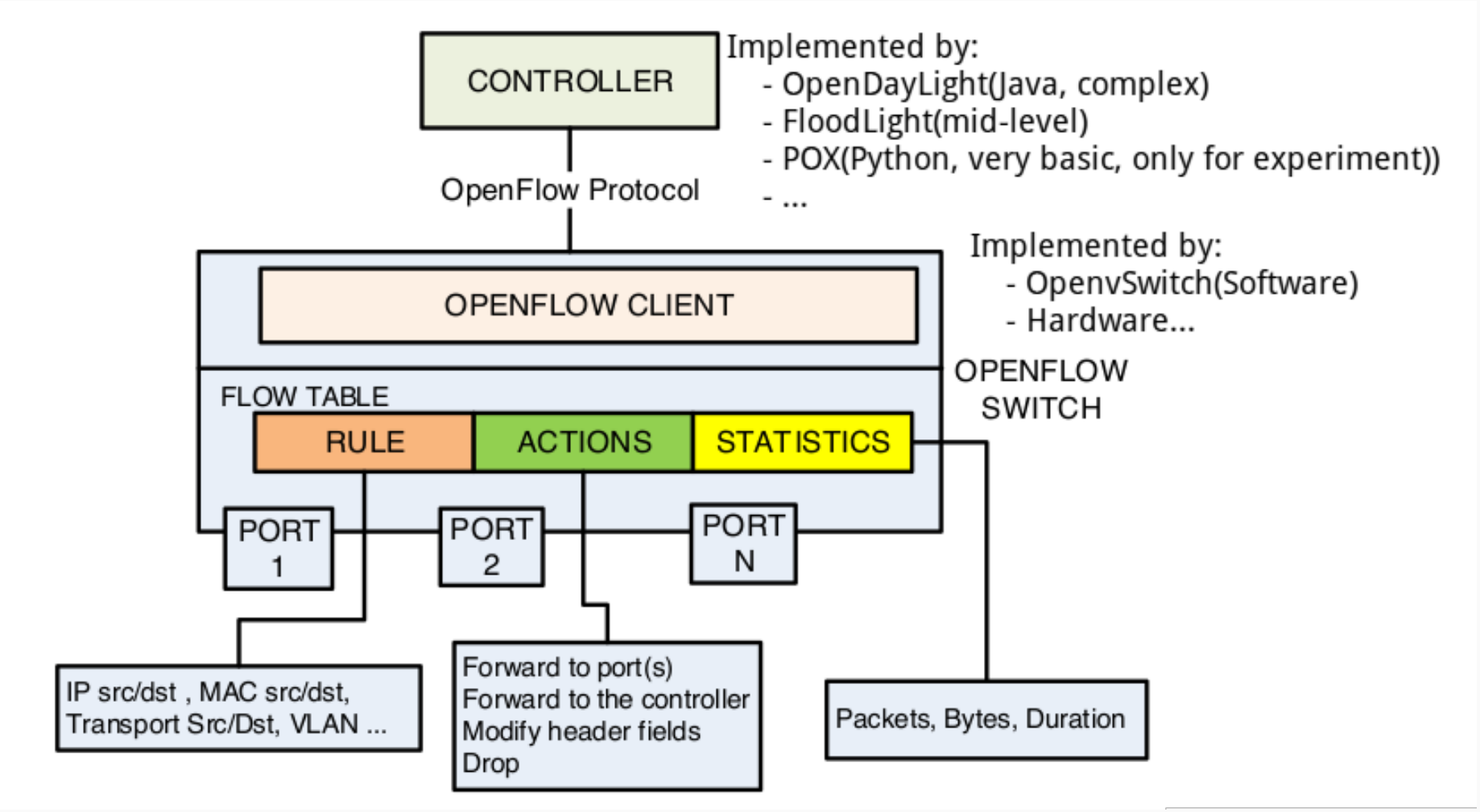

Figure 3: Architecture of OpenFlow

Flow table on Switch(Simplified)

```
When receive packet:
    while True:
        flow_table = 0
        for flow_entry in flow_table with priority desc:
            if match(packet, flow_entry.match):
                run(flow_entry.action)
                # may modify packet
                # or send to another table(flow_table=X)
                # or output, drop, ... and finish
                update(flow_entry.counter)
                update(flow_table.counter)
                break
        if no flow_entry in flow_table matches:
            run(flow_table.table_miss_entry)
            # Usually send_to_controller/drop/next_table
```

Matching

```
field = extract_from(packet, match.oxm_type)
if match.oxm_hasmask:
    field &= match.mask
return field == match.oxm_value
```

Actions

Output `port_no` Output to a normal port/in port/all ports/ or send to controller/specific table

Group `group_id` Process through group

Drop Drop this packet

Set-Queue `queue_id` Send to queue

Meter `meter_id` Direct to meters

Set-Field `field_type value` Rewrite specific field

Data & Control Plane messaging

Switch can tell controller:

- Hello(during startup)
- Packet-in(when `output CONTROLLER`)
- Flow-removed

Controller can tell switch:

- Modify-state
- Read-state
- Send-packet

* Both lists are incomplete

How to implement QoS in OpenFlow

Queues

OpenFlow 1.2+ support both minimum and maximum rates for a given queue.

Definition of Queue(OpenvSwitch):

```
ovs-vsctl set port eth0 qos=@newqos -- --id=@newqos \
 create qos type=linux-htb other-config:max-rate=5000000 \
 queues:0=@newqueue -- --id=@newqueue create queue \
 other-config:min-rate=3000000 \
 other-config:max-rate=3000000

# ---> Queue --[min: 3Mbps, max: 3Mbps]-> linux-htb
#  --[max: 5Mbps]--> eth0
```

Meters

Unlike queues, meters could be installed/removed/modified like a flow entry in meter table.

Packet could be forwarded to group by `Meter meter_id` action

```
When packet comes into meter m with rate r:
    for band in meter.bands:
        if r > band.rate:
            apply(band.type) # Usually DROP
            band.counter++
```

Collecting information

- Table level
- Flow level
- Port level
- Queue level
- Meter level

Thanks for listening