

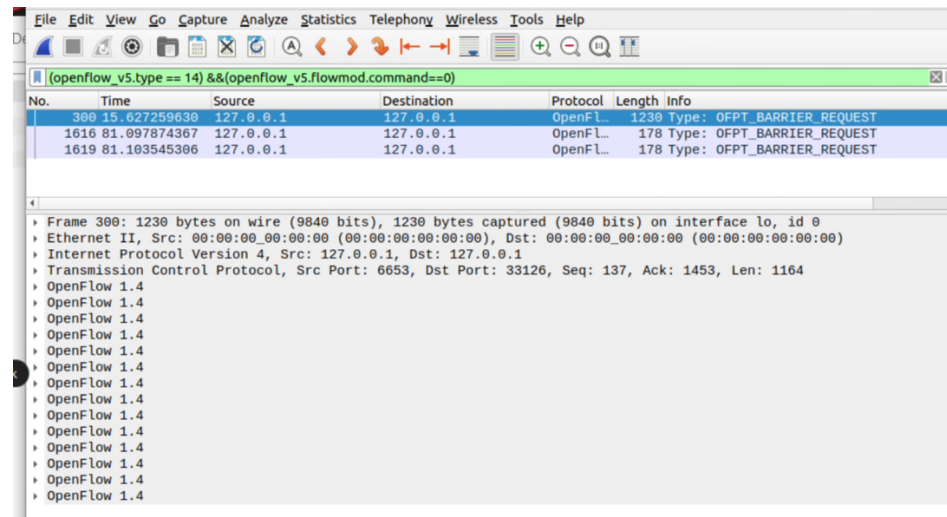
SDNFV Lab2

Part 1: Answer Questions

There are **6** distinct “OFPT_FLOW_MOD” headers during the experiment.

| Math fields | Actions | Timeout values |
|---|--------------------------|----------------|
| ETH_TYPE=0x0806 (ARP) | Out port=OFPP_CONTROLLER | 0 |
| ETH_TYPE=0x88cc (LLDP) | Out port=OFPP_CONTROLLER | 0 |
| ETH_TYPE=0x8942 (BDDP) | Out port=OFPP_CONTROLLER | 0 |
| ETH_TYPE=0x0800 (IPv4) | Out port=OFPP_CONTROLLER | 0 |
| IN_PORT=2 ETH_DST=EE:2C:D5:37:D7:CB ETH_SRC=52:7e:19:AA:EA:87 | Out port=1 | 10 |
| IN_PORT=1 ETH_DST=52:7e:19:AA:EA:87 ETH_SRC=EE:2C:D5:37:D7:CB | Out port=2 | 10 |

我使用 filter (openflow_v5.type == 14) && (openflow_v5.flowmod.command == 0) in wireshark.



Part 2: Install Flow Rules

Install **one** flow rule to forward ARP packets with below condition

Screenshot of mininet shell

```
mininet> h1 arping h2
ARPING 10.0.0.2
42 bytes from fa:4e:0b:3f:06:f8 (10.0.0.2): index=0 time=265.470 usec
42 bytes from fa:4e:0b:3f:06:f8 (10.0.0.2): index=1 time=4.410 usec
42 bytes from fa:4e:0b:3f:06:f8 (10.0.0.2): index=2 time=2.403 usec
42 bytes from fa:4e:0b:3f:06:f8 (10.0.0.2): index=3 time=5.417 usec
42 bytes from fa:4e:0b:3f:06:f8 (10.0.0.2): index=4 time=7.204 usec
```

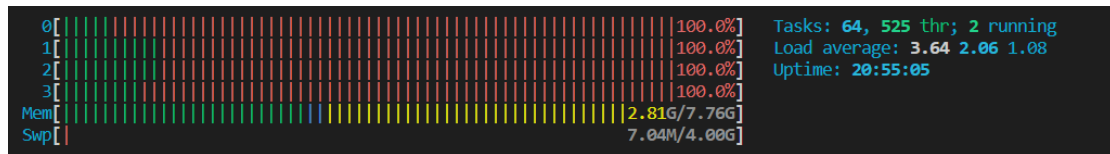
Install **two** flow rules to forward IPv4 packets

Screenshot of mininet shell

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.820 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.188 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.077 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2137ms
rtt min/avg/max/mdev = 0.077/0.361/0.820/0.327 ms
mininet>
mininet> h2 ping h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.035 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.039 ms
^C
--- 10.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 4010ms
```

Part 3: Create Topology with Broadcast Storm

Screenshot of CPU's utilization



Screenshot of flows for switch 1

Flows for Device of:0000000000000001 (4 Total)

| STATE | PACKETS | DURATION | FLOW PRIORITY | TABLE NAME | SELECTOR | TREATMENT | APP NAME |
|-------|------------|----------|---------------|------------|--------------|--------------------------------------|----------|
| Added | 0 | 155 | 40000 | 0 | ETH_TYPEarp | imm[OUTPUT:CONTROLLER], cleared:true | *core |
| Added | 104 | 155 | 40000 | 0 | ETH_TYPElldp | imm[OUTPUT:CONTROLLER], cleared:true | *core |
| Added | 104 | 155 | 40000 | 0 | ETH_TYPEbddp | imm[OUTPUT:CONTROLLER], cleared:true | *core |
| Added | 67,892,047 | 155 | 50000 | 0 | ETH_TYPEarp | imm[OUTPUT:FLOOD], cleared:false | *rest |

Screenshot of flows for switch 2

Flows for Device of:0000000000000002 (4 Total)

| STATE | PACKETS | DURATION | FLOW PRIORITY | TABLE NAME | SELECTOR | TREATMENT | APP NAME |
|-------|------------|----------|---------------|------------|--------------|--------------------------------------|----------|
| Added | 0 | 155 | 40000 | 0 | ETH_TYPEarp | imm[OUTPUT:CONTROLLER], cleared:true | *core |
| Added | 102 | 155 | 40000 | 0 | ETH_TYPEbddp | imm[OUTPUT:CONTROLLER], cleared:true | *core |
| Added | 102 | 155 | 40000 | 0 | ETH_TYPElldp | imm[OUTPUT:CONTROLLER], cleared:true | *core |
| Added | 67,891,641 | 155 | 50000 | 0 | ETH_TYPEarp | imm[OUTPUT:FLOOD], cleared:false | *rest |

Screenshot of flows for switch 3

Flows for Device of:0000000000000003 (4 Total)

| STATE | PACKETS | DURATION | FLOW PRIORITY | TABLE NAME | SELECTOR | TREATMENT | APP NAME |
|-------|------------|----------|---------------|------------|--------------|--------------------------------------|----------|
| Added | 0 | 155 | 40000 | 0 | ETH_TYPEarp | imm[OUTPUT:CONTROLLER], cleared:true | *core |
| Added | 103 | 155 | 40000 | 0 | ETH_TYPElldp | imm[OUTPUT:CONTROLLER], cleared:true | *core |
| Added | 103 | 155 | 40000 | 0 | ETH_TYPEbddp | imm[OUTPUT:CONTROLLER], cleared:true | *core |
| Added | 67,891,443 | 155 | 50000 | 0 | ETH_TYPEarp | imm[OUTPUT:FLOOD], cleared:false | *rest |

My observation:

當我在 mininet shell 輸入 `h1 arping h2` 時，有成千上萬的 ARP 回覆終端機。另外我使用 `htop` 來監控 CPU 的使用率，核心的使用率都接近 100%。我也注意到，即使我中斷了 `arping` 指令，我在每個 switch 加入的 flow rule 的計數器還是在增加。

原因

1. 我的拓樸有一個循環。
2. 我的流量規則的匹配情況和動作是匹配 ARP 封包並送到除入口 port 以外的每個 port。

假設我們現在輸入 `h1 arping h2`。

s1 會廣播 ARP 請求，s2 收到 ARP 請求後，會廣播請求，並將 h2 的 ARP 回覆轉寄給 h1。現在 s3 交換器收到兩個 ARP 請求，一個來自 s1，另一個來自 s2 (我假設 s2 先收到封包)，s3 交換器會分別廣播這兩個 ARP 請求。這種情況會在每個交換器中持續發生，這就是原因。

更嚴重的是 ARP 沒有 TTL 欄位，這表示封包會永遠存在於網路上。這也是我

在每台交換器上安裝的流量規則計數器，即使我中斷了 `arping` 指令，計數器還是不斷增加的原因。

Part 4: Trace Reactive Forwarding

Summarize ReactiveForwarding Behavior:

1. 啟動 (In ONOS shell type: `app activate fwd`) [Source code line233](#)
 - 執行 `ReactiveForwarding.activate`
 - 啟動 `ReactivePacketProcessor` (該 `Processor` 負責處理封包)。
 - 執行 `readComponentConfiguration` 讀取設定檔。
2. `ReactivePacketProcessor.process` (該 `process` 負責處理封包) [Source code line466](#)
 - 若不知道 `destination host` 是誰 -> Invoke `ReactiveForwarding.flood`
 - 知道 `destination host` 是誰，但沒有合適的 `Flow Rule` -> Invoke `ReactiveForwarding.installRule`
3. `ReactiveForwarding.flood` ([Source code line583](#))
 - Invoke `ReactiveForwarding.packetOut` 送出封包 (`PortNumber` 為 `Flood`)
4. `ReactiveForwarding.installRule` ([Source code line600](#))
 - 先對 `Switch` 下 `Flow Rule` (`selector: IN_PORT, ETH_DST and ETH_SRC`)。
(**Control Plane**)
 - 在呼叫 `ReactiveForwarding.packetOut` 送出封包。(Data Plane)

Observe what happens in data and control planes

- From the time when `h1` pings `h2` until `h2` receives the first ICMP request
- Write down each operation made by data and control planes
- Please refer to the ONOS ReactiveForwarding application

當 `h1` ping `h2` 時:

1. `h1` 會先發送 `ARP request` 詢問 `h2` 的 `MAC Address`。
 - `h1` 送出 `ARP request` 到 `Switch`。(Data Plane)
 - 此時根據 `Switch` 上一條預設的 `Flow Rule` 會將該 `ARP request` 包裝進 `packet-in Message` 送至 `Controller`。(Control Plane)
 - `Controller` 使用 `packet-out Message` 包裝 `ARP request` 送至 `Switch`。(Control Plane)
 - 由於 `Controller` 還不知道 `Destination Host` 位於 `Switch` 的哪一個 `port` 上，因此這次會透過 `Flood` 的方式送出 `ARP request`。(Data Plane)
 - Note: 此時 `Switch` 知道 `h1` 連接 `port 1`，`h2` 連接 `port 2`。
2. 接著 `h2` 會回覆 `ARP reply` 告訴 `h1` 它的 `MAC Address`。

- h2 送出 ARP reply 到 Switch 。 (Data Plane)
 - 此時根據 Switch 上一條預設的 Flow Rule 會將該 ARP request 包裝進 packet-in Message 送至 Controller 。 (Control Plane)
 - Controller 透過 packet-out Message 包裝 ARP reply 送到 Switch 。 (Control Plane)
 - Switch 已經知道 port 1 連接著 h1 ， 因此會直接將 ARP reply 透過 port 1 送至 h1 。 (Data Plane)
3. 再來 h1 會送 ICMP echo request 給 h2 。
- h1 先送 ICMP echo request 給 Switch 。 (Data Plane)
 - 此時 Switch 上沒有合適的 Flow Rule 進行 Forwarding ， 因此透過 packet-in Message 包裝 ICMP echo request 送給 Controller 。 (Control Plane)
 - Controller 先在 Switch 上安裝 Flow Rule ， 再使用 packet-out Message 包裝 ICMP echo request 送到 Switch 。 (Control Plane)
 - Switch 會將 ICMP echo request 送至 h2 。 (Data Plane)
4. 最後 h2 會收到來自 h1 的 ICMP echo request 。

What I have learned or solved

1. OpenFlow header type, "OFPT_FLOW_MOD", 是用來新增/修改/刪除 Flow Rule。
 - 可以搭配不同的 Command 使用 (OFPFC_ADD, OFPFC_MODIFY and OFPFC_DELETE)
2. Action *ALL* and *FLOOD* 的差異
 - FLOOD: Flood the packet along the minimum spanning tree, not including the incoming interface.
 - ALL: Flood the packet along the minimum spanning tree, not including the incoming interface.
3. 如何透過 curl 和 GUI 對 Switch 安裝/刪除 Flow Rule。
4. 了解 Broadcast Storm 發生的原因，以及可行的解決方法。
5. Trace Reactive Forwarding 了解該程式的運作。