

Project 2 - OpenFlow Protocol Observation and Flow Rule Installation

112109045 何逸君

Part1

1 . How many OpenFlow headers with type “OFPT_FLOW_MOD” and command “OFPPC_ADD” are there among all the packets?

A: 10

2 . What are the match fields and the corresponding actions in each “OFPT_FLOW_MOD” message?

3 . What are the Idle Timeout values for all flow rules on s1 in GUI?

Match fields Timeout Values	Action	Timeout Values
ETH_TYPE = IPv4 (0x0800)	Output port = OFPP_CONTROLLER	0
ETH_TYPE = BDDP (0x8942)(Unknown)	Output port = OFPP_CONTROLLER	0
ETH_TYPE = APR (0x0806)	Output port = OFPP_CONTROLLER	0
ETH_TYPE = 802.1 Link Layer Discovery Protocol ,LLDP (0x88c)	Output port = OFPP_CONTROLLER	0
IN_PORT = 1	Output port = 2	0
ETH_DST = 72:66:09:c2:f1:ea	Output port = 2	0
ETH_SRC = 92:cc:64:fc:1a:27	Output port = 2	0
IN_PORT = 2	Output port = 1	0
ETH_DST = 92:cc:64:fc:1a:27	Output port = 1	0
ETH_SRC = 72:66:09:c2:f1:ea	Output port = 1	0

Part2

[one flow rule] Screenshot of the flow rules I installed:

Flows for Device of:0000000000000001 (4 Total)

Search

All Fields

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	405	40000	0	ETH_TYPE:arp	imm[OUTPUT CONTROLLER] cleared:true	*core
Added	0	405	40000	0	ETH_TYPE:icmp	imm[OUTPUT CONTROLLER] cleared:true	*core
Added	0	405	40000	0	ETH_TYPE:icmp	imm[OUTPUT CONTROLLER] cleared:true	*core
Added	106	76	50000	0	ETH_TYPE:arp	imm[OUTPUT FLOOD] cleared:false	*rest

```
mininet> h1 arp ping h2
ARPING 10.0.0.2
42 bytes from 6a:4e:ab:a4:f6:cc (10.0.0.2): index=0 time=12.879 usec
42 bytes from 6a:4e:ab:a4:f6:cc (10.0.0.2): index=1 time=4.552 usec
42 bytes from 6a:4e:ab:a4:f6:cc (10.0.0.2): index=2 time=11.250 usec
42 bytes from 6a:4e:ab:a4:f6:cc (10.0.0.2): index=3 time=6.682 usec
42 bytes from 6a:4e:ab:a4:f6:cc (10.0.0.2): index=4 time=7.084 usec
42 bytes from 6a:4e:ab:a4:f6:cc (10.0.0.2): index=5 time=8.766 usec
42 bytes from 6a:4e:ab:a4:f6:cc (10.0.0.2): index=6 time=9.206 usec
42 bytes from 6a:4e:ab:a4:f6:cc (10.0.0.2): index=7 time=4.841 usec
42 bytes from 6a:4e:ab:a4:f6:cc (10.0.0.2): index=8 time=10.950 usec
42 bytes from 6a:4e:ab:a4:f6:cc (10.0.0.2): index=9 time=9.063 usec
42 bytes from 6a:4e:ab:a4:f6:cc (10.0.0.2): index=10 time=12.587 usec
```

[two flow rules] Screenshot of the flow rules I installed:

Flows for Device of:0000000000000001 (6 Total)

Search

All Fields

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	100	40000	0	ETH_TYPE:icmp	imm[OUTPUT CONTROLLER] cleared:true	*core
Added	0	100	40000	0	ETH_TYPE:icmp	imm[OUTPUT CONTROLLER] cleared:true	*core
Added	2	100	5	0	ETH_TYPE:ip4	imm[OUTPUT CONTROLLER] cleared:true	*core
Added	4	100	40000	0	ETH_TYPE:arp	imm[OUTPUT CONTROLLER] cleared:true	*core
Added	10	10	10	0	RLPORT2 ETH_DST:12:34:56:78:9A:BC ETH_SRC:82:C3:A5:59:85:B3	imm[OUTPUT 1] cleared:false	*bad
Added	10	10	10	0	RLPORT1 ETH_DST:82:C3:A5:59:85:B3 ETH_SRC:12:34:56:78:9A:BC	imm[OUTPUT 2] cleared:false	*bad

```
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=0.735 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.046 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.089 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.075 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.105 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.539 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.044 ms
```

Part3

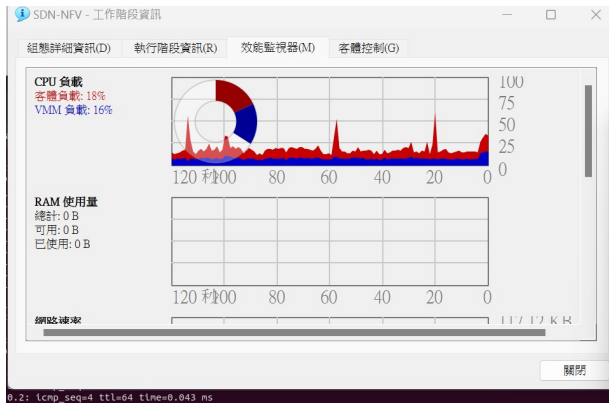
Screenshot of CPU's utilization - Before:

```

top - 01:34:13 up 1:00, 2 users, load average: 0.61, 0.71, 0.71
Tasks: 216 total, 2 running, 214 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.7 us, 1.0 sy, 0.0 ni, 97.9 id, 0.2 wa, 0.0 hi, 0.2 st, 0.0 st
Mem Mem : 7933.7 total, 190.9 free, 3092.1 used, 4650.6 buff/cache
Mem Swap: 2048.0 total, 2047.5 free, 0.5 used, 4487.1 avail Mem

  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
 2443 sylvia    20   0 5220112 804140 28416 S  2.7  9.9   2:39.15 java
1465 sylvia    20   0 4546364 339088 136496 S  1.3  4.2  4:19.97 gnome-s+
   15 root      20   0      0      0      0 R   0.3  0.0   0:03.65 rcu_pre+
 557 systemd+ 20   0 14828   6912  6144 S  0.3  0.1   0:05.65 systemd+
2152 sylvia    20   0 4738700   1.1g 19584 S  0.3 13.6   2:30.27 java
3050 root      20   0      0      0      0 I   0.3  0.0   0:04.36 kworker+
3171 sylvia    20   0 3516892 383716 147900 S  0.3  4.7   1:20.25 firefox
3802 root      20   0      0      0      0 I   0.3  0.0   0:05.17 kworker+
   1 root      20   0 166688   11828  8244 S  0.0  0.1   0:02.29 systemd
   2 root      20   0      0      0      0 S   0.0  0.0   0:00.00 kthreadd
   3 root      0 -20      0      0      0 I   0.0  0.0   0:00.00 rcu_gp
   4 root      0 -20      0      0      0 I   0.0  0.0   0:00.00 rcu_par+
   5 root      0 -20      0      0      0 I   0.0  0.0   0:00.00 club_fla

```



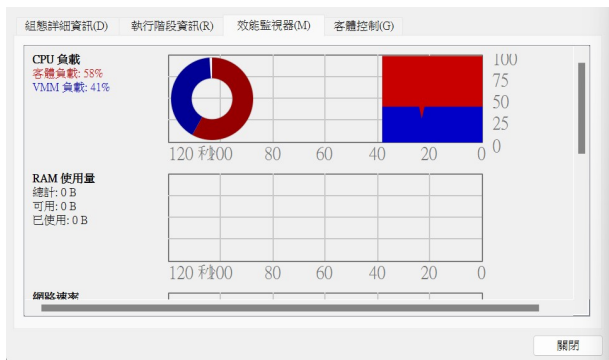
Screenshot of CPU's utilization - After: (top 中CPU的load average超過1, 幾乎是原來的兩倍)

```

top - 02:00:00 up 1:26, 2 users, load average: 1.20, 1.12, 1.13
Tasks: 218 total, 2 running, 216 sleeping, 0 stopped, 0 zombie
%Cpu(s): 7.6 us, 9.6 sy, 0.0 ni, 81.6 id, 0.0 wa, 0.0 hi, 1.1 st, 0.0 st
Mem Mem : 7933.7 total, 150.2 free, 3134.2 used, 4644.2 buff/cache
Mem Swap: 2048.0 total, 2047.5 free, 0.5 used, 4444.7 avail Mem

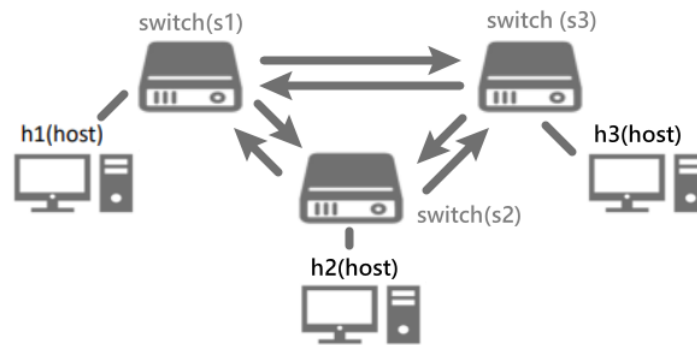
  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
1465 sylvia    20   0 4553392 350292 136492 R 29.9  4.3   7:31.40 /usr/bin/gnome-shell
3171 sylvia    20   0 3495688 372852 148028 S  6.0  4.6   2:38.04 /snap/firefox/3290/usr/lib/ft+
2443 sylvia    20   0 5220112 810528 28416 S  3.3 10.0   3:41.18 /tmp/onos-2.7.0-jdk/bin/java +
2000 sylvia    20   0 604480 57940  41796 S  2.3  0.7   0:30.97 /usr/libexec/gnome-terminal+
3838 sylvia    20   0 2434948 115996 78804 S  2.0  1.4   0:37.30 /snap/firefox/3290/usr/lib/ft+
1949 sylvia    20   0 219796 79532  64560 S  1.7  1.0   0:33.22 /usr/bin/Xwayland :0 -rootles+
3818 sylvia    20   0 2429972 108288 79224 S  1.7  1.3   0:08.04 /snap/firefox/3290/usr/lib/ft+
4994 root      20   0      0      0      0 I   0.7  0.0   0:00.93 [kworker/u412-events_power_eff+
156 root      20   0      0      0      0 S   0.3  0.0   0:01.63 [jbd2/sda3-0]
1653 sylvia    20   0 315600 12208  7168 S  0.3  0.2   0:09.48 /usr/bin/ibus-daemon --panel +
3050 root      20   0      0      0      0 I   0.3  0.0   0:07.01 [kworker/i:0-events]
3804 sylvia    20   0 2451840 125492 85320 S  0.3  1.5   0:37.79 /snap/firefox/3290/usr/lib/ft+
5322 root      20   0      0      0      0 I   0.3  0.0   0:00.29 [kworker/u413-events_unbound]
5437 sylvia    20   0 13552  4352  3584 R  0.3  0.1   0:00.50 top
   1 root      20   0 166688   11828  8244 S  0.0  0.1   0:02.33 /sbin/init splash
   2 root      20   0      0      0      0 S   0.0  0.0   0:00.00 [kthreads]
   3 root      0 -20      0      0      0 I   0.0  0.0   0:00.00 [rcu_gp]

```



Q: Why the broadcast storm occur?

A: 我先創造出以下的topology, 接著安裝相對應的flow rules, 接著在mininet中執行h1 arping h2。從示意圖中可以看到h1會先送出一個arp broadcast封包, switch (s1) 收到從h1來的封包後會broadcast到switch (s2)與switch (s3), switch (s2)收到從switch (s1)來的封包會broadcast到switch (s3)與h2, 而switch (s3)收到封包後會在broadcast到switch (s2) 與 switch (s1), 如此不斷持續broadcast下去, 最終就會導致broadcast storm.



Part4

Activate only “org.onosproject.fwd” and other initially activated APPs.

Use Mininet default topology and let h1 ping h2.

Q: What happens in control and data planes?

Data plane:

1. h1將ping packets forward到h2時, packets 先從h1 forward到 s1
2. s1收到packets 時, 因為最初沒有適用的flow rule, 因此將packets事件訊號先傳給controller
3. packets forward到controller, 由controller做處理與轉發
4. 根據Reactive Forwarding app的ONOS controller產生的flow rule, controller將packets forward到相對應的switch以forward packets到目標host

Control plane:

1. Reactive Forwarding app的ONOS controller接收到packets事件訊號。controller會先讀取並檢查packets(ex: DST, SRC..etc), 並決定處理方式
2. controller根據packets內容, 創建flow rule以將packets forward到h2, 而後將其安裝到發送packets事件訊號的switch中, 以flow rule命令switch根據符合目標特徵的packet進行什麼action
3. switch安裝好flow rule後, SDN switch會根據指令, 將後續符合類似條件且來自h1的packets forward到h2。
4. 當h2收到ping的packets時, 他會產生一個預計回覆訊號給h1
5. 當packets到達連接h2的switch時, switch會查看table找出匹配的flow rule, 並根據其中內容, 將回覆訊號回給h1

6. 當h1收到回覆, 即會顯示ping連接成功
-

What I've learned or solved ?

- 熟悉**flow rule**: 透過part 1/2 兩小題練習flow rule的寫法以及須注意的事項。由於未注意到整體架構, 先把mininet&flow rule重置才做part2第二小題, 導致packets傳不出去, 而更懂得ping連接的要素。
- 熟悉**wireshark**: 透過part 1/4 兩題練習解讀wireshark每個欄位的大致意義與架構。