Subject - Link Aggregation Control Protocol Simulation

Abstract

鏈路聚合(Link Aggregation Control Protocol) 為IEEE 802.3ad協定中的一部分，透過此功能同一個裝置(例如:網路儲存裝置NAS)上的兩個獨立連接埠(網路孔)可邏輯上相結合，視為同一條實體線路。不僅增加整體的使用頻寬，提升裝置間的網路傳輸速度

Motivation

在另一門課CCNA 中學到如何在 packet tracer 中設定 EtherChannel、LACP、Port Aggregation Control Protocol(PAgP)，而最近上課有教到 SDN相關知識，因此覺得可以將兩者結合，用 SDN controller 來實現 LACP。

Results

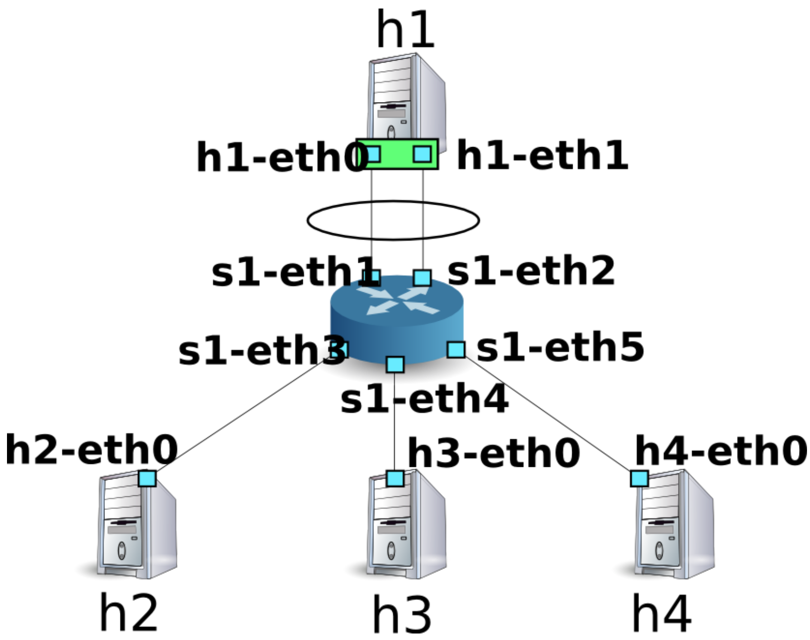
實驗環境:

Python 3.8
Ubuntu 20.04
Virtualbox
Mininet 2.3.1b1 (master branch)
Ryu 4.34 (pip install)

執行指令

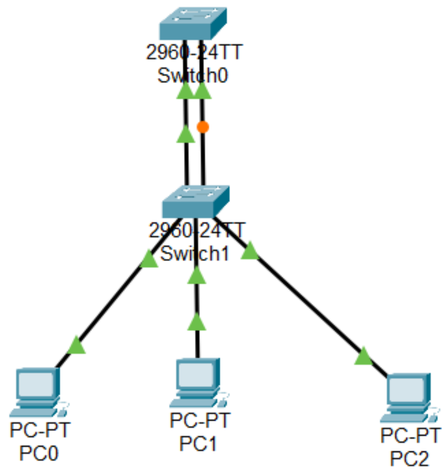
```
(在視窗A) ryu-manager controller.py --verbose
(在視窗B) sudo mn --custom topo.py --topo=mytopo --
controller=remote,ip=127.0.0.1
```

目前拓樸圖



[Link Aggregation — Ryubook 1.0 documentation]

在這個拓樸中，如果沒有使用LACP 的話， 就會在 spanning tree protocol 的作用下關掉 switch 的某個 port 來避免 self-loop，被 blocked 的 link 只有當另一條無法正常作用時，才會啟用。
啟用 LACP 之後，兩條 link都能作用，不只可以達到 Load Balance 的效果，當 h2、h3、h4 同時從 h1 下載時，也可以提升 throughput 和 communication speed



⬆️ 未啟用 LACP 的樣子

⤋ 如果不對 bounding driver 進行任何設定，bond0 預設會使用 round robin 的方式來選擇使用哪條 link 進行傳輸。

```
root@nscap2:/media/sf_Final_Project# cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v5.15.0-60-generic

Bonding Mode: load balancing (round-robin)
MII Status: up
MII Polling Interval (ms): 0
Up Delay (ms): 0
Down Delay (ms): 0
Peer Notification Delay (ms): 0

Slave Interface: h1-eth0
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 00:00:00:00:00:11
Slave queue ID: 0

Slave Interface: h1-eth1
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 00:00:00:00:00:12
Slave queue ID: 0
```

- **Bonding Mode: load balancing**
  表示流量會以輪循的方式分配到每個連接介面，達到均衡負載的效果。
- **MII Status: up**
  MII (Media Independent Interface) 狀態為「up」，表示 bond0 介面目前是啟動且運作中的。
- **Up Delay (ms): 0**
  新啟動的連接介面被包含在 Bond 中的延遲時間。這裡設定為 0 毫秒，表示沒有延遲，即新連接立即生效。
- **Down Delay (ms): 0**
  被移除的連接介面從 Bond 中移除的延遲時間。這裡設定為 0 毫秒，表示沒有延遲，即失效連接立即被移除。

在 /etc/modprobe.d/bonding.conf 添加下面設定之後，bonding driver 就會更改分散流量的方式

```
alias bond0 bonding
options bonding mode=4
```

之後在 h1 的 xterm 中執行下方shell script
script 的作用: 首先建立一個邏輯介面 bond0，並且設定其 mac address，之後先暫時關閉原有的兩個介面 h1-eth0、h1-eth1，重新設定介面的 mac address，並把他們的 master 設為 bond0 以方便管理

```bash
#!/bin/bash

modprobe bonding

# Create bond0 interface and set MAC address
sudo ip link add bond0 type bond
sudo ip link set bond0 address 02:01:02:03:04:08

# Set physical interfaces down, set MAC address, and add to bond0
sudo ip link set h1-eth0 down
sudo ip link set h1-eth0 address 00:00:00:00:00:11
sudo ip link set h1-eth0 master bond0
sudo ip link set h1-eth1 down
sudo ip link set h1-eth1 address 00:00:00:00:00:12
sudo ip link set h1-eth1 master bond0

# Assign IP address to bond0 and delete from h1-eth0
sudo ip addr add 10.0.0.1/8 dev bond0
sudo ip addr del 10.0.0.1/8 dev h1-eth0

# Bring bond0 interface up
sudo ip link set bond0 up
```

⬇️ 更改 bond driver 設定檔後，bond0 用 MAC address 進行 hashing 後的結果來決定要選用哪條 link 來進行傳輸

```
root@nscap2:/media/sf_Final_Project# cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v5.15.0-60-generic

Bonding Mode: IEEE 802.3ad Dynamic link aggregation
Transmit Hash Policy: layer2 (0)
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0
Peer Notification Delay (ms): 0

802.3ad info
LACP active: on
LACP rate: slow
Min links: 0
Aggregator selection policy (ad_select): stable
System priority: 65535
System MAC address: 02:01:02:03:04:08
Active Aggregator Info:
        Aggregator ID: 1
        Number of ports: 2
        Actor Key: 15
        Partner Key: 15
        Partner Mac Address: c2:57:12:bc:51:49
```

- **Transmit Hash Policy: layer2 (0)**
  Layer 2 表示根據第二層（MAC地址）資訊來分配出站流量。
- **MII Polling Interval (ms): 100**
  MII 輪詢間隔（毫秒）：100 毫秒，表示每 100 毫秒會檢查一次連接狀態。
- **LACP rate: slow**
  慢速，表示 LACP control frame 每 30 秒發送一次。傳送間隔的 3 倍時間(90 s)若是無任何通訊發生時，則該界面自該群組移除，不再使用於封包的傳送。
- **Min links: 0**
  表示即使沒有活動的連接介面，Bond 仍然保持活躍

⬇️ 未設定 link aggregation 前的網路設定

```
root@nscap2:/media/sf_Final_Project# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::47b:7fff:fe9b:fdcc  prefixlen 64  scopeid 0x20<link>
        ether 06:7b:7f:9b:fd:cc  txqueuelen 1000  (Ethernet)
        RX packets 55  bytes 5408 (5.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9  bytes 726 (726.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

h1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::78b3:b9ff:fecc:da15  prefixlen 64  scopeid 0x20<link>
        ether 7a:b3:b9:cc:da:15  txqueuelen 1000  (Ethernet)
        RX packets 53  bytes 5232 (5.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9  bytes 726 (726.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

⬇️ 完成設定後顯示的網路介面: 邏輯界面 bond0 為 MASTER，實體界面 h1-eth0 和 h1-eth1 為 SLAVE。而且可以看到 bond0、h1-eth0 和 h1-eth1 的 MAC 位址全部都是相同的。

```
root@nscap2:/media/sf_Final_Project# ifconfig
bond0: flags=5187<UP,BROADCAST,RUNNING,MASTER,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 0.0.0.0
        inet6 fe80::1:2ff:fe03:408  prefixlen 64  scopeid 0x20<link>
        ether 02:01:02:03:04:08  txqueuelen 1000  (Ethernet)
        RX packets 11  bytes 1114 (1.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9  bytes 922 (922.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

h1-eth0: flags=6211<UP,BROADCAST,RUNNING,SLAVE,MULTICAST>  mtu 1500
        ether 02:01:02:03:04:08  txqueuelen 1000  (Ethernet)
        RX packets 70  bytes 6666 (6.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18  bytes 1540 (1.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

h1-eth1: flags=6211<UP,BROADCAST,RUNNING,SLAVE,MULTICAST>  mtu 1500
        ether 02:01:02:03:04:08  txqueuelen 1000  (Ethernet)
        RX packets 73  bytes 6916 (6.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 13  bytes 1114 (1.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

程式片段解釋

```
self._lacp.add(
    dpid=str_to_dpid('0000000000000001'),
    ports=[1, 2],
    lacp_update_time=3,
    lacp_timeout_time=10)
```

在 s1 上的 port1、port2 設定支援 lacp 的功能。

```
@set_ev_cls(lacplib.EventSlaveStateChanged, MAIN_DISPATCHER)
def _slave_state_changed_handler(self, ev):
    datapath = ev.datapath
    dpid = datapath.id
    port_no = ev.port
    enabled = ev.enabled
    self.logger.info("slave state changed port: %d enabled: %s",
                     port_no, enabled)
    if dpid in self.mac_to_port:
        for mac in self.mac_to_port[dpid]:
            match = datapath.ofproto_parser.OFPMatch(eth_dst=mac)
            self.del_flow(datapath, match)
        del self.mac_to_port[dpid]
    self.mac_to_port.setdefault(dpid, {})
```

這個函式用以處理 port state 發生改變的情形 (從 enable 變為 disable，或是從 disable 變為 enable)。當 datapath ID 有出現在 mac table 中，會呼叫 del_flow，並把 mac table 改為預設值。

```
def del_flow(self, datapath, match):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    mod = parser.OFPFlowMod(datapath=datapath,
                            command=ofproto.OFPFC_DELETE,
                            out_port=ofproto.OFPP_ANY,
                            out_group=ofproto.OFPG_ANY,
                            match=match)
    datapath.send_msg(mod)
```

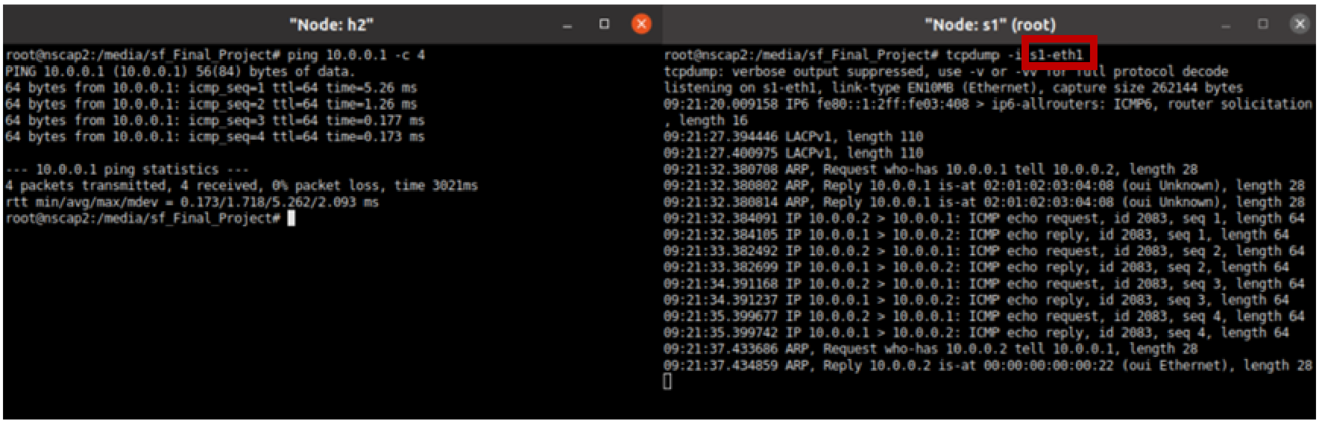
假如某個 port state 發生改變，則會呼叫此函數刪除 flow entries。

Advantage of Link Aggregation

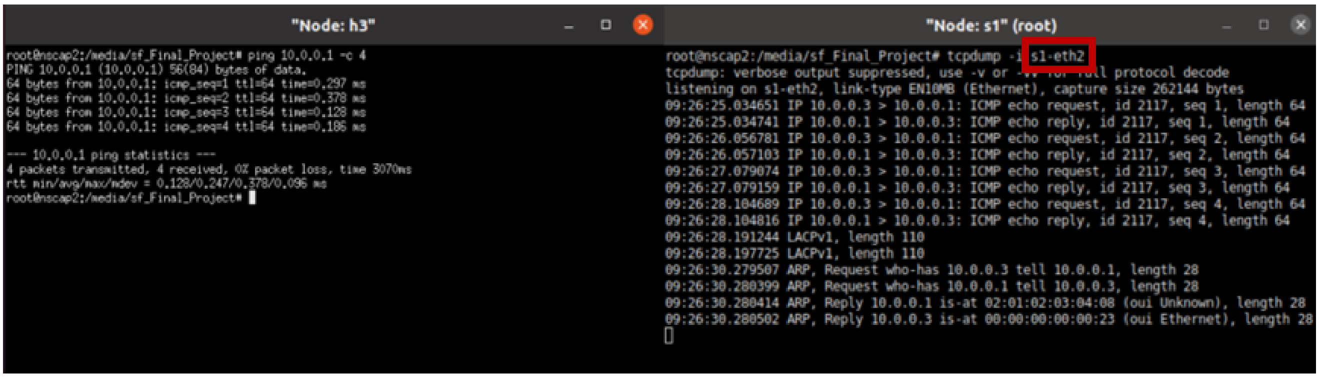
Load Balance

下面用三種情境檢查，經過了哪個 Interface

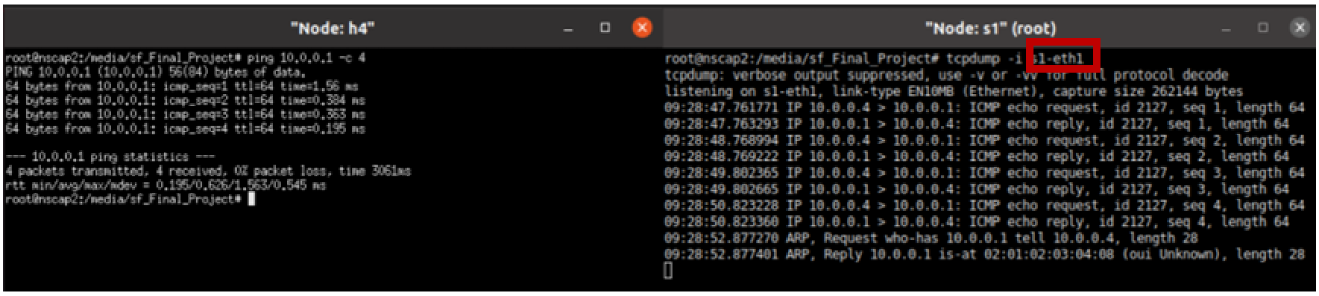
h2 ping h1 (10.0.0.1) 是經過 s1-eth1



h3 ping h1 (10.0.0.1) 是經過 s1-eth2



h4 ping h1 是經過 s1-eth1



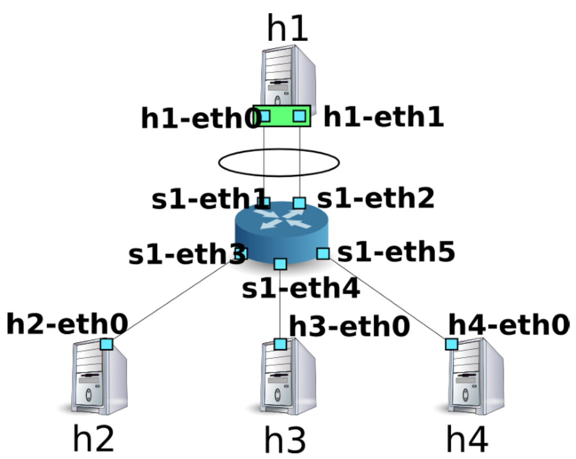
Destination host	Port used
h2	1
h3	2
h4	1

可以注意到不會只使用特定 port 轉送封包，而藉此達到 load balance 的效果

Recovering from fault automatically

⬇️ 原本 h3 可以正常 ping 到 h1，s1 會學到要從 h3 送往 h1 時，要從 port 2 走到 h1-eth1

```
root@nscap2:/media/sf_Final_Project# ping 10.0.0.1 -c 4 | ts
 五  26 22:07:03 PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
 五  26 22:07:03 64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.234 ms
 五  26 22:07:04 64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.408 ms
 五  26 22:07:05 64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.071 ms
 五  26 22:07:06 64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.060 ms
 五  26 22:07:06
 五  26 22:07:06 --- 10.0.0.1 ping statistics ---
 五  26 22:07:06 4 packets transmitted, 4 received, 0% packet loss, time 3053ms
 五  26 22:07:06 rtt min/avg/max/mdev = 0.060/0.193/0.408/0.141 ms
```



⬇️ 輸入指令 'ip link set h1-eth1 nomaster' 之後，h1-eth1 不再是 bond0 的其中一個 SLAVE，但 controller 會以為h1-eth1 還可以正常運作，繼續丟往那個介面，因此 h3 會暫時無法 ping 到 h1 (直到 90秒後 exchange timeout 到了之後會自動恢復)

"Node: h1"

```
root@nscap2:/media/sf_Final_Project# ./setting.sh
root@nscap2:/media/sf_Final_Project# ts
^C
root@nscap2:/media/sf_Final_Project# date
公曆 20廿四年 五月 廿六日 週日 廿二時二分四秒
root@nscap2:/media/sf_Final_Project# date +"%H:%M:%S"
22:05:54
root@nscap2:/media/sf_Final_Project# date +"%H:%M:%S"
22:05:58
root@nscap2:/media/sf_Final_Project# ip link set h1-eth1 nomaster
root@nscap2:/media/sf_Final_Project# date +"%H:%M:%S"
22:08:17
root@nscap2:/media/sf_Final_Project#
```

```
[LACP][INFO] SW=0000000000000001 PORT=2 LACP exchange timeout has occurred.
EVENT lacplib->SimpleSwitchLacp13 EventSlaveStateChanged
slave state changed port: 2 enabled: False
```

⬇️大約 22:08:17 設定完後，一開始確實發現 h3 ping h1 並不通(約17 秒後)，但之後就可以通了 (100 秒後)

```
root@nscap2:/media/sf_Final_Project# ping 10.0.0.1 -c 4 | ts
 五  26 22:08:34 PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
 五  26 22:08:34
 五  26 22:08:34 --- 10.0.0.1 ping statistics ---
 五  26 22:08:34 4 packets transmitted, 0 received, 100% packet loss, time 3050m
s
 五  26 22:08:34
root@nscap2:/media/sf_Final_Project# ping 10.0.0.1 -c 4 | ts
 五  26 22:09:57 PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
 五  26 22:09:57 64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=5.40 ms
 五  26 22:09:58 64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.190 ms
 五  26 22:09:59 64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.116 ms
 五  26 22:10:00 64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.078 ms
 五  26 22:10:00
 五  26 22:10:00 --- 10.0.0.1 ping statistics ---
 五  26 22:10:00 4 packets transmitted, 4 received, 0% packet loss, time 3058ms
 五  26 22:10:00 rtt min/avg/max/mdev = 0.078/1.445/5.398/2.282 ms
```

在 final project 中，我使用 Ryu controller 模擬了鏈路聚合控制協議 (LACP)，並展示了其在"負載均衡" 和 "從錯誤中自動恢復" 兩大優點。

"負載均衡" 這樣可以避免單一路徑的壅塞，讓網路運作更流暢，效率更高。這對於處理大量資料流量的網路環境來說非常重要。

此外，當鏈路發生故障時，LACP 可以自動偵測並迅速重新配置路徑，確保網路連接穩定不中斷，提高了網路的可靠性。