Intro. to Computer Newtwork Proj4 Report

0416324 胡安鳳

1. 請嘗試使用 pingall 指令來觀察程式碼，在尚未做任何修改時，有哪些 Host 之間已經可以互通？為什麼？

Ans:

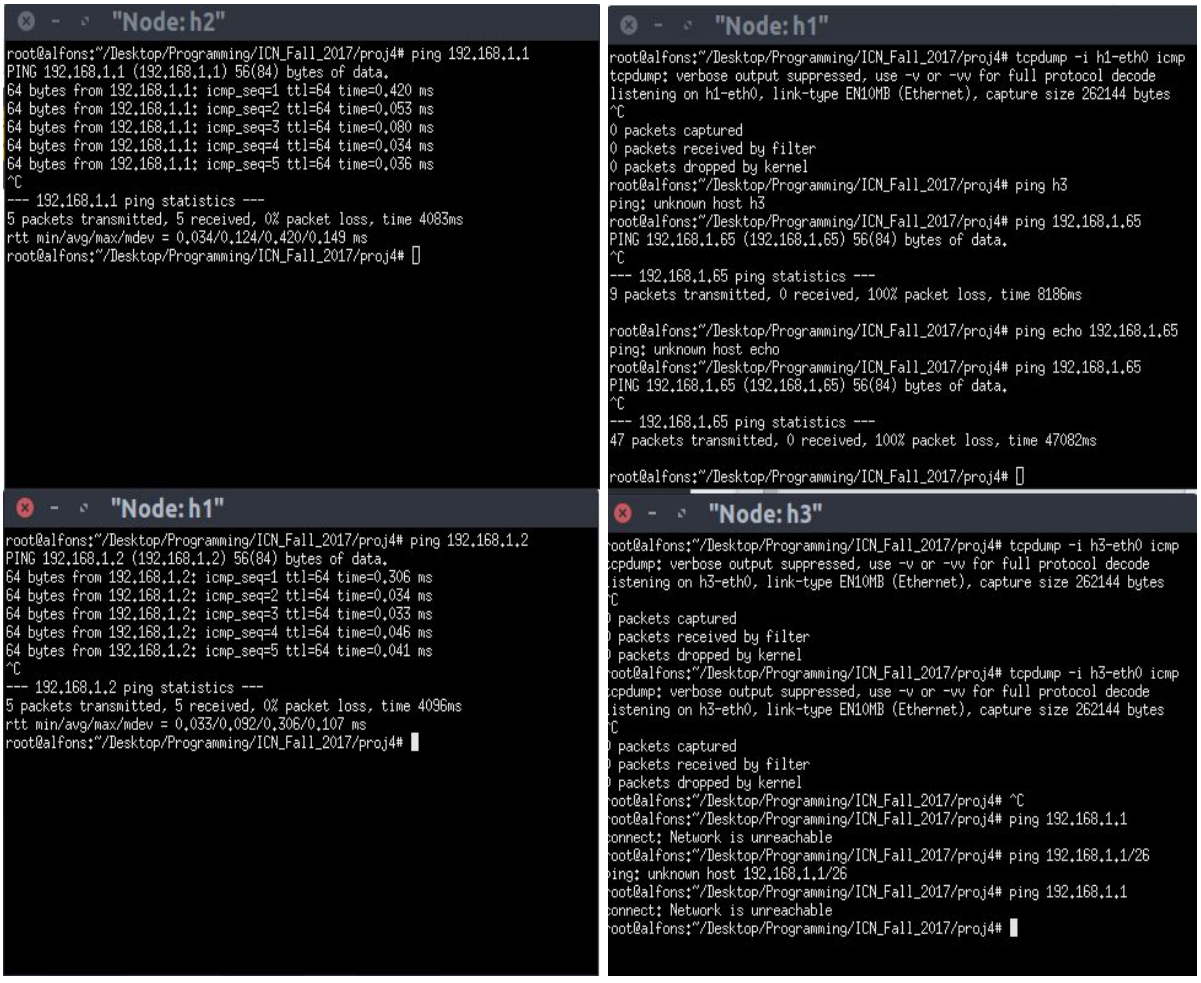
```
alfons@alfons ~/Desktop/Programming/ICN Fall 2017/proj4 master sudo python topo.py
mininet> h1 ping h2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=0.449 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.035 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=0.041 ms
^C
--- 192.168.1.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2036ms
rtt min/avg/max/mdev = 0.035/0.175/0.449/0.193 ms
mininet> h1 ping h3
PING 192.168.1.65 (192.168.1.65) 56(84) bytes of data.
^C
--- 192.168.1.65 ping statistics ---
11 packets transmitted, 0 received, 100% packet loss, time 10221ms

mininet> h1 ping h4
PING 192.168.1.66 (192.168.1.66) 56(84) bytes of data.
^C
--- 192.168.1.66 ping statistics ---
12 packets transmitted, 0 received, 100% packet loss, time 11245ms

mininet> h1 ping h5
PING 192.168.2.1 (192.168.2.1) 56(84) bytes of data.
From 192.168.1.62 icmp_seq=1 Destination Net Unreachable
From 192.168.1.62 icmp_seq=2 Destination Net Unreachable
From 192.168.1.62 icmp_seq=3 Destination Net Unreachable
From 192.168.1.62 icmp_seq=4 Destination Net Unreachable
^C
--- 192.168.2.1 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3079ms

mininet> h1 ping h6
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
From 192.168.1.62 icmp_seq=1 Destination Net Unreachable
From 192.168.1.62 icmp_seq=2 Destination Net Unreachable
From 192.168.1.62 icmp_seq=11 Destination Net Unreachable

^C
--- 192.168.2.2 ping statistics ---
13 packets transmitted, 0 received, +3 errors, 100% packet loss, time 12295ms
```

```
mininet> h2 ping h3
connect: Network is unreachable
mininet> h2 ping h4
connect: Network is unreachable
mininet> h2 ping h5
connect: Network is unreachable
mininet> h2 ping h6
connect: Network is unreachable
mininet> h3 ping h4
PING 192.168.1.66 (192.168.1.66) 56(84) bytes of data.
64 bytes from 192.168.1.66: icmp_seq=1 ttl=64 time=0.433 ms
64 bytes from 192.168.1.66: icmp_seq=2 ttl=64 time=0.037 ms
64 bytes from 192.168.1.66: icmp_seq=3 ttl=64 time=0.034 ms
64 bytes from 192.168.1.66: icmp_seq=4 ttl=64 time=0.039 ms
64 bytes from 192.168.1.66: icmp_seq=5 ttl=64 time=0.043 ms
64 bytes from 192.168.1.66: icmp_seq=6 ttl=64 time=0.043 ms
64 bytes from 192.168.1.66: icmp_seq=7 ttl=64 time=0.035 ms
^C
--- 192.168.1.66 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6143ms
rtt min/avg/max/mdev = 0.034/0.094/0.433/0.138 ms
mininet> h3 ping h5
connect: Network is unreachable
mininet> h3 ping h6
connect: Network is unreachable
mininet> h4 ping h5
connect: Network is unreachable
mininet> h4 ping h6
connect: Network is unreachable
mininet> h5 ping h6
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
64 bytes from 192.168.2.2: icmp_seq=1 ttl=64 time=0.435 ms
64 bytes from 192.168.2.2: icmp_seq=2 ttl=64 time=0.036 ms
64 bytes from 192.168.2.2: icmp_seq=3 ttl=64 time=0.035 ms
64 bytes from 192.168.2.2: icmp_seq=4 ttl=64 time=0.038 ms
64 bytes from 192.168.2.2: icmp_seq=5 ttl=64 time=0.051 ms
^C
--- 192.168.2.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4094ms
rtt min/avg/max/mdev = 0.035/0.119/0.435/0.158 ms
mininet>
```

從圖可看 h1-h2 h3-h4 h5-h6 在做任何的設定之已互通，原因應是各屬於一個私有網路（內網）的關係，彼此可用mac address傳遞訊息。
h1 h2建立在s3 之下、h3 h4建立在s4之下、 h5 h6建立在s5之下。

2. 在未修改程式碼時， h1 的 ping echo 有被 h3 收到（透過 tcpdump 或 wireshark 觀察），但 h3 的回應卻沒回到 h1 ，請問為什麼會有這樣的現象發生？

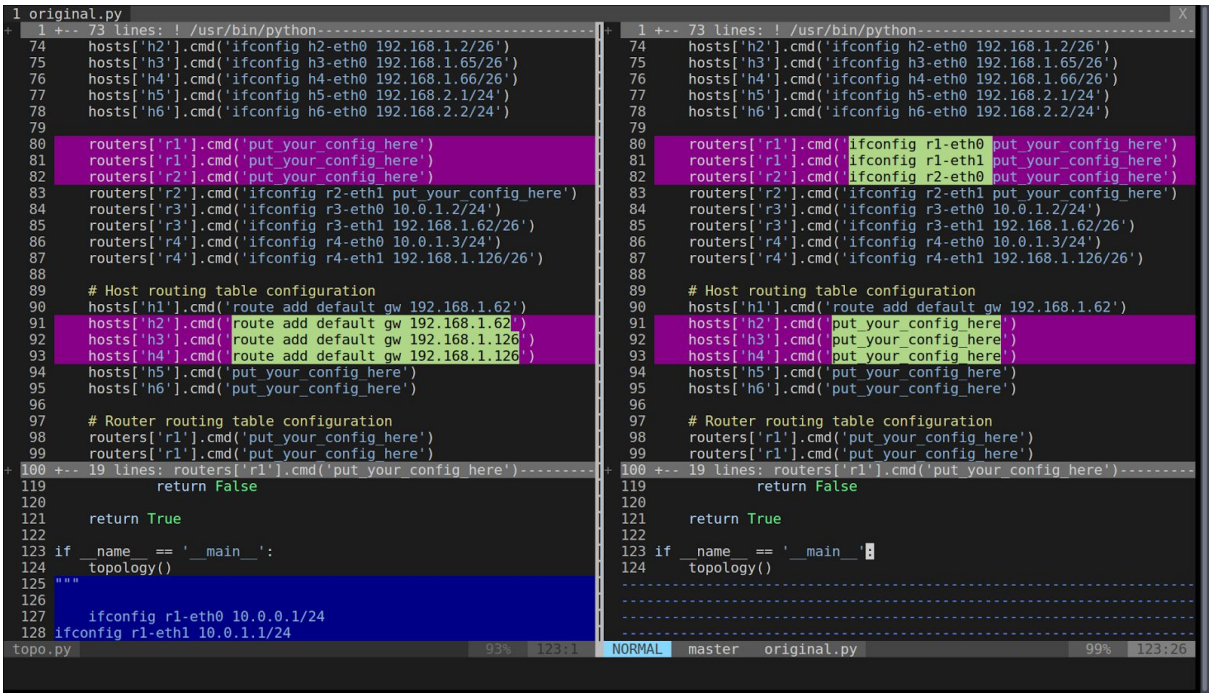
Ans:經由測試過後 發現原本相接的h1 h2可以互通 但是h1 h3無法互通



是router的interface IP沒有設定好以及對於h3的routing 對應的router r4沒有設定好的緣故，亦即，h3需要將自己的資料發給r4，但原本的code並沒有設定成功，以至於收到h1的資料後無法回覆，因此只能有單方面的傳送（h1無法收到h3的ack確認成功信息，因此無法判斷h3是否成功接收，而h3本身也沒有能力透過r4將資料送出去，因此會顯示network unreachable)因此，發送的訊息只能單方面由h1 送h3並且沒有從h3 到

h1的回應，同樣h3也無法pingh1 。

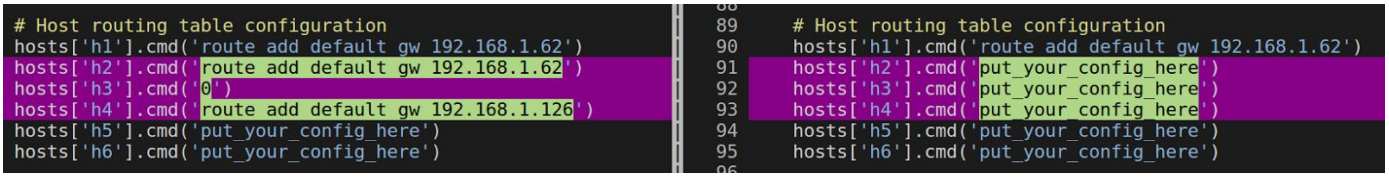
3. 若希望 h1 能得到 h3 的 ping reply，應該要在封包沿途經過的 router 設定什麼？除此之外還需要在 host 上設定什麼？為什麼？



首先我們要把封包經過的router(從作業圖中看來是r3 r4)各端口(interface)的IP設定好，（即上圖vimdiff的上半部r1 r2的部份），這樣遠方的資料送過來，不論是從內網的或是外網的才有地址可循(但原本已經設定好了，故不須再多做設定)。

接著我們還要把host要發給那一個router的ip設定好，這樣host才能找到正確的傳資料路徑，把資料發包給遠端

首先這是沒有設定好host3的routing規則的結果



```
ython topo.py
mininet> h1 ping h3
PING 192.168.1.65 (192.168.1.65) 56(84) bytes of data.
^C
--- 192.168.1.65 ping statistics ---
16 packets transmitted, 0 received, 100% packet loss, time 15360ms

mininet> h3 ping h1
connect: Network is unreachable
mininet> h3 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.64    0.0.0.0         255.255.255.192 U     0      0        0 h3-eth0
mininet>
```

接著我們把h3的routing規則設定好

```
hosts['h3'].cmd('route add default gw 192.168.1.126')
```

他的routing table出來了，的確有連到 192.168.1.126 亦即r4

```
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.1.126   0.0.0.0         UG    0      0        0 h3-eth0
192.168.1.64    0.0.0.0         255.255.255.192 U     0      0        0 h3-eth0
```

全部設定好之後h1 h3便能互通資料了！

```
sudo python topo.py
--- 192.168.1.65 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7129ms
rtt min/avg/max/mdev = 0.050/0.187/1.041/0.323 ms
mininet> h1 ping h3
PING 192.168.1.65 (192.168.1.65) 56(84) bytes of data.
64 bytes from 192.168.1.65: icmp_seq=1 ttl=62 time=0.563 ms
64 bytes from 192.168.1.65: icmp_seq=2 ttl=62 time=0.071 ms
64 bytes from 192.168.1.65: icmp_seq=3 ttl=62 time=0.088 ms
64 bytes from 192.168.1.65: icmp_seq=4 ttl=62 time=0.068 ms
64 bytes from 192.168.1.65: icmp_seq=5 ttl=62 time=0.071 ms
^C
--- 192.168.1.65 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4092ms
rtt min/avg/max/mdev = 0.068/0.172/0.563/0.195 ms
mininet> h3 ping h1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=62 time=0.086 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=62 time=0.112 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=62 time=0.107 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=62 time=0.098 ms
64 bytes from 192.168.1.1: icmp_seq=5 ttl=62 time=0.117 ms
64 bytes from 192.168.1.1: icmp_seq=6 ttl=62 time=0.113 ms
^V64 bytes from 192.168.1.1: icmp_seq=7 ttl=62 time=0.112 ms
^?^C
--- 192.168.1.1 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6139ms
rtt min/avg/max/mdev = 0.086/0.106/0.117/0.013 ms
mininet>
```

4. 延續上題，下一個目標是使 h1, h2 所在的子網域能與

h5, h6 所在的子網域互通，以及使 h3, h4 所在的子網域與 h5, h6 互通，請問你需要分別在各 host 與各 router 設定什麼？為什麼？

首先我們可以看到原本設定好的

```
routers['r3'].cmd('route add -net 192.168.1.64/26 gw 10.0.1.3')
routers['r3'].cmd('put_your_config_here') #
routers['r4'].cmd('route add -net 192.168.1.0/26 gw 10.0.1.2')
routers['r4'].cmd('put_your_config_here') #
```

這個表示 r3在收到destination為192.168.1.64/26 亦即longest prefix到192.168.1.64 & 0xFFFFFF2 = 192.168.1.01XXXXXX的IP就會往 10.0.0.3的遠方router去routing 故192.168.1.65 192.168.1.66的h3 h4才能在這個子網路底下收到要的資料。

可以看到r3 r4的routing 已經趨近完善但是r1的完全還沒配置成功

```
mininet> r1 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 r1-eth0
10.0.1.0        0.0.0.0         255.255.255.0   U     0      0        0 r1-eth1
mininet>  r4 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.1.0        0.0.0.0         255.255.255.0   U     0      0        0 r4-eth0
192.168.1.0     10.0.1.2        255.255.255.192 UG    0      0        0 r4-eth0
192.168.1.64    0.0.0.0         255.255.255.192 U     0      0        0 r4-eth1
mininet> r3 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.1.0        0.0.0.0         255.255.255.0   U     0      0        0 r3-eth0
192.168.1.0     0.0.0.0         255.255.255.192 U     0      0        0 r3-eth1
192.168.1.64    10.0.1.3        255.255.255.192 UG    0      0        0 r3-eth0
mininet>
```

用同樣的道理 我們再把r3到 h5 h6所會經過的router設定一次吧，並不需要設定到r2因為routing table就只是在相鄰的路由器進行數據交換罷了

r3看資料如何送，設定r4和r1對應的，例如192.168.2./24的子網路變可以往r1去

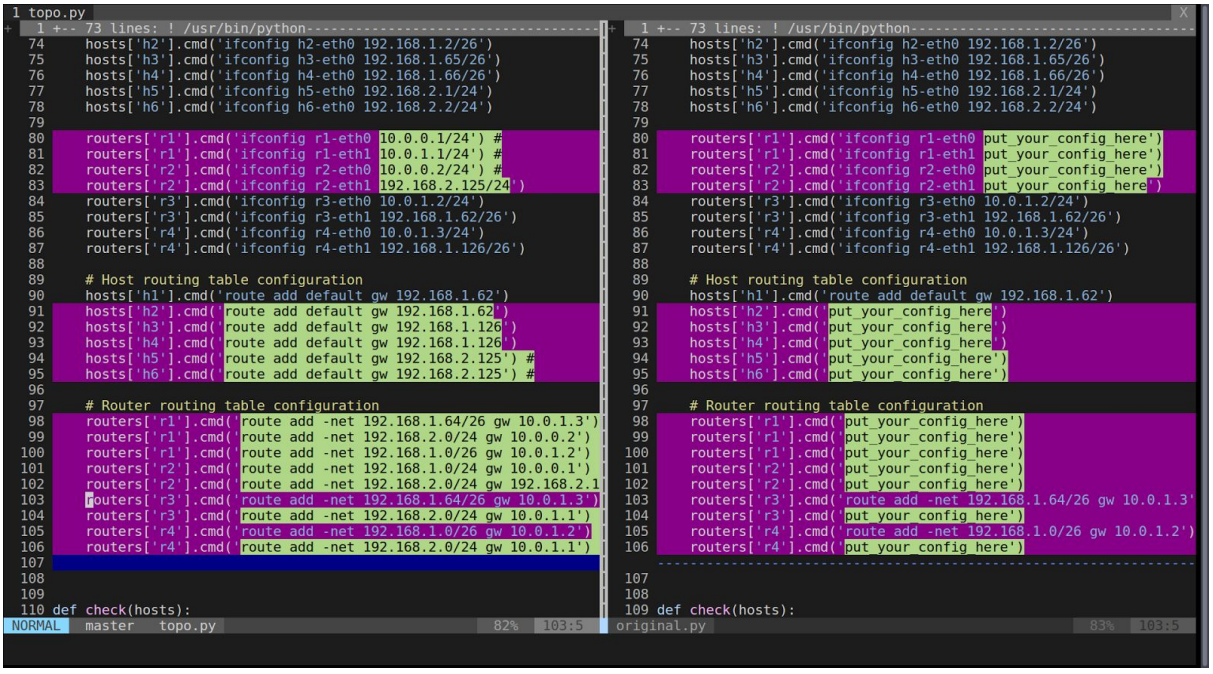
同理可證r4以及其他的設定

```
routers['r3'].cmd('route add -net 192.168.1.64/26 gw 10.0.1.3')
routers['r3'].cmd('route add -net 192.168.2.0/24 gw 10.0.1.1') #
routers['r4'].cmd('route add -net 192.168.1.0/26 gw 10.0.1.2')
routers['r4'].cmd('route add -net 192.168.2.0/24 gw 10.0.1.1') #
```

```
mininet> r1 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 r1-eth0
10.0.1.0        0.0.0.0         255.255.255.0   U     0      0        0 r1-eth1
mininet>  r4 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.1.0        0.0.0.0         255.255.255.0   U     0      0        0 r4-eth0
192.168.1.0     10.0.1.2        255.255.255.192 UG    0      0        0 r4-eth0
192.168.1.64    0.0.0.0         255.255.255.192 U     0      0        0 r4-eth1
mininet> r3 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.1.0        0.0.0.0         255.255.255.0   U     0      0        0 r3-eth0
192.168.1.0     0.0.0.0         255.255.255.192 U     0      0        0 r3-eth1
192.168.1.64    10.0.1.3        255.255.255.192 UG    0      0        0 r3-eth0
mininet>
```

可以看到路由資訊更趨完善。

照著同樣的概念全部打完，用vimdiff來看一下新增了哪些東西



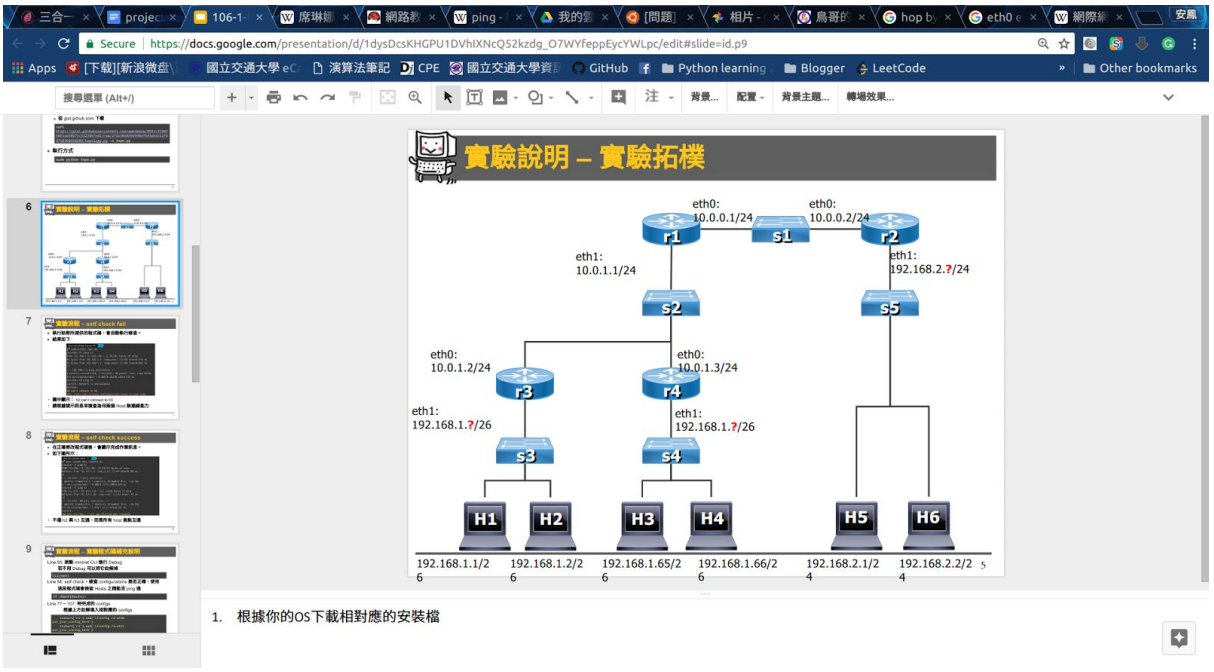
```
# Router routing table configuration
routers['r1'].cmd('route add -net 192.168.1.64/26 gw 10.0.1.3') #r1-r4
routers['r1'].cmd('route add -net 192.168.2.0/24 gw 10.0.0.2') #r1-r2
routers['r1'].cmd('route add -net 192.168.1.0/26 gw 10.0.1.2') #r1-r3
routers['r2'].cmd('route add -net 192.168.1.0/24 gw 10.0.0.1') #r2-r1 (prefix25 is ok) 64is 01000000 and 1 is 00000001
routers['r2'].cmd('route add -net 192.168.2.0/24 gw 192.168.2.125') #r2-h5+h6
routers['r3'].cmd('route add -net 192.168.1.64/26 gw 10.0.1.3') #r3-r4
routers['r3'].cmd('route add -net 192.168.2.0/24 gw 10.0.1.1') #r3-r1
routers['r4'].cmd('route add -net 192.168.1.0/26 gw 10.0.1.2') #r4-r3
routers['r4'].cmd('route add -net 192.168.2.0/24 gw 10.0.1.1') #r4-r1
```

也就是要把剩下未完善的路由規則寫完，而對於eth接口的子網路可以自己定義prefix後面的IP 。

因為把這些路由規則打完後，才能讓各個節點的封包互通有無，形成一個完整的網路拓樸結構。

在模仿r3 r4 的方式定義子網路之中的可用IP元件之一

```
routers['r2'].cmd('ifconfig r2-eth1 192.168.2.125/24')
routers['r3'].cmd('ifconfig r3-eth0 10.0.1.2/24')
routers['r3'].cmd('ifconfig r3-eth1 192.168.1.62/26')
routers['r4'].cmd('ifconfig r4-eth0 10.0.1.3/24')
routers['r4'].cmd('ifconfig r4-eth1 192.168.1.126/26')

# Host routing table configuration
hosts['h1'].cmd('route add default gw 192.168.1.62')
hosts['h2'].cmd('route add default gw 192.168.1.62')
hosts['h3'].cmd('route add default gw 192.168.1.126')
hosts['h4'].cmd('route add default gw 192.168.1.126')
hosts['h5'].cmd('route add default gw 192.168.2.125') #
hosts['h6'].cmd('route add default gw 192.168.2.125') #

# Router routing table configuration
routers['r1'].cmd('route add -net 192.168.1.64/26 gw 10.0.1.3') #r1-r4
routers['r1'].cmd('route add -net 192.168.2.0/24 gw 10.0.0.2') #r1-r2
routers['r1'].cmd('route add -net 192.168.1.0/26 gw 10.0.1.2') #r1-r3
routers['r2'].cmd('route add -net 192.168.1.0/24 gw 10.0.0.1') #r2-r1 (prefix25 i
routers['r2'].cmd('route add -net 192.168.2.0/24 gw 192.168.2.125') #r2-h5+h6
routers['r3'].cmd('route add -net 192.168.1.64/26 gw 10.0.1.3') #r3-r4
routers['r3'].cmd('route add -net 192.168.2.0/24 gw 10.0.1.1') #r3-r1
routers['r4'].cmd('route add -net 192.168.1.0/26 gw 10.0.1.2') #r4-r3
routers['r4'].cmd('route add -net 192.168.2.0/24 gw 10.0.1.1') #r4-r1
```



而最左邊的？為0（老師寫的）

但是因為要和中間的？因為他們longest prefix同 但是為了區分 65 66 因此老師寫64 剛好64 = 01000000 卡到第26位的1能和前面的

1 2進行二進位的區分因為他們沒64那個1

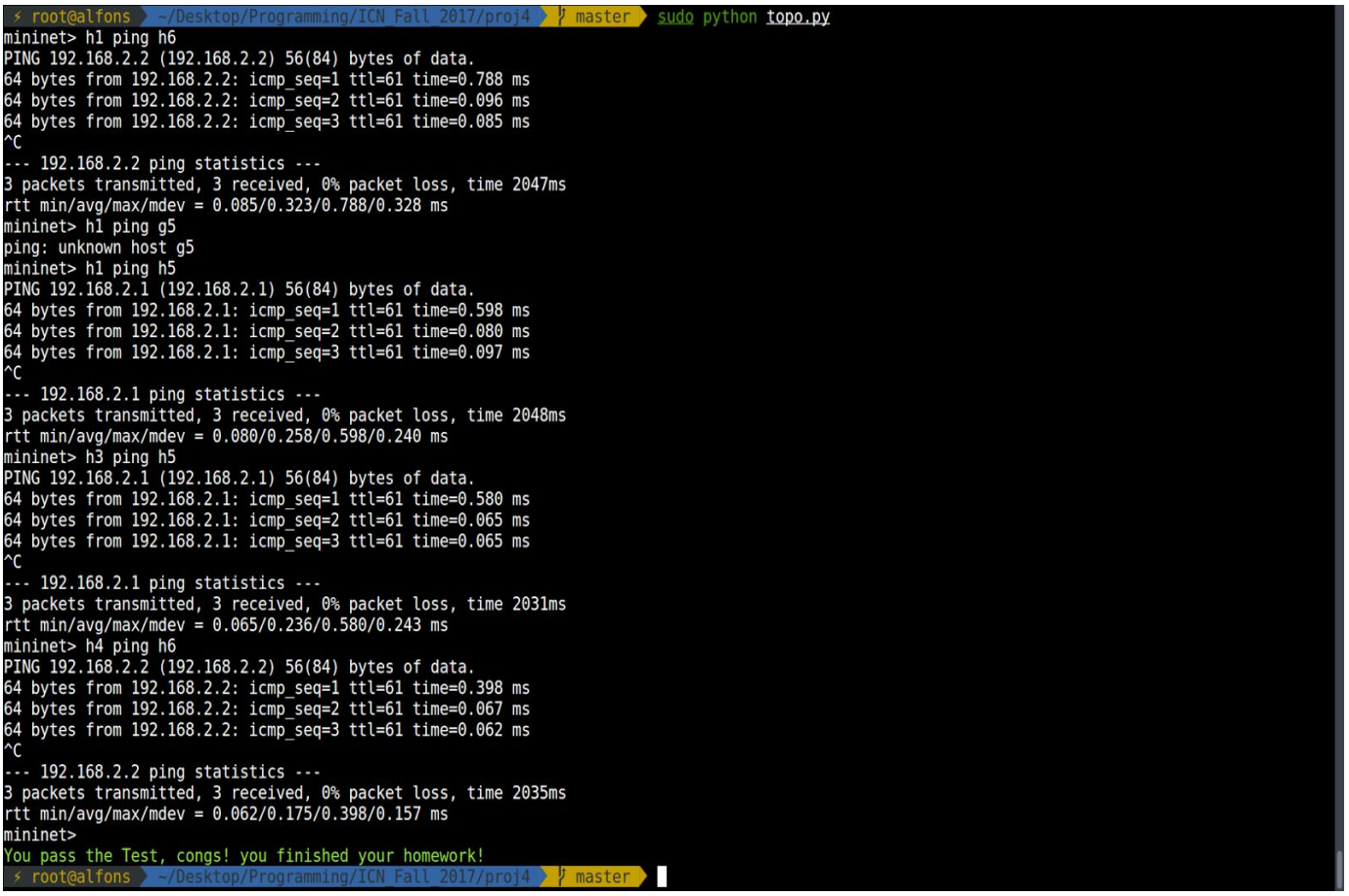
最後192.168.2.？最右側的 因為前面已經不同了 不需要在細分的26bit 用24bit就好 因此可以選一個2^(32-24)但不包含0 255的數值作為r2-eth1子網路的ip即可

要注意後面能用的ip只有longest prefix match之後剩下的bit數L 以及不可以用 2^L 和 0作為interface eth接口的IP 0是作為子網路的位址而

因此若將192.168.2.125改為192.168.2.0 或是192.168.2.255 前者為子網路開始，後者為廣播用IP都無法通行 只能改為(0,255)區間的數值

(結果一使用0 結果二使用255 結果三使用137)

```
alfons@alfons ~/Desktop/Programming/ICN_Fall_2017/proj4 master ● sudo python topo.py
mininet> exit
h2 can't connect to h6
Hmm, looks some network configurations wrong in your code.
alfons@alfons ~/Desktop/Programming/ICN_Fall_2017/proj4 master ● sudo python topo.py
mininet> exit
h2 can't connect to h6
Hmm, looks some network configurations wrong in your code.
alfons@alfons ~/Desktop/Programming/ICN_Fall_2017/proj4 master ● sudo python topo.py
mininet> exit
You pass the Test, congs! you finished your homework!
alfons@alfons ~/Desktop/Programming/ICN_Fall_2017/proj4 master ●
```



5.在 r1 開啟兩個 Wireshark，並監聽於 r1-eth0 與 r1-eth1。

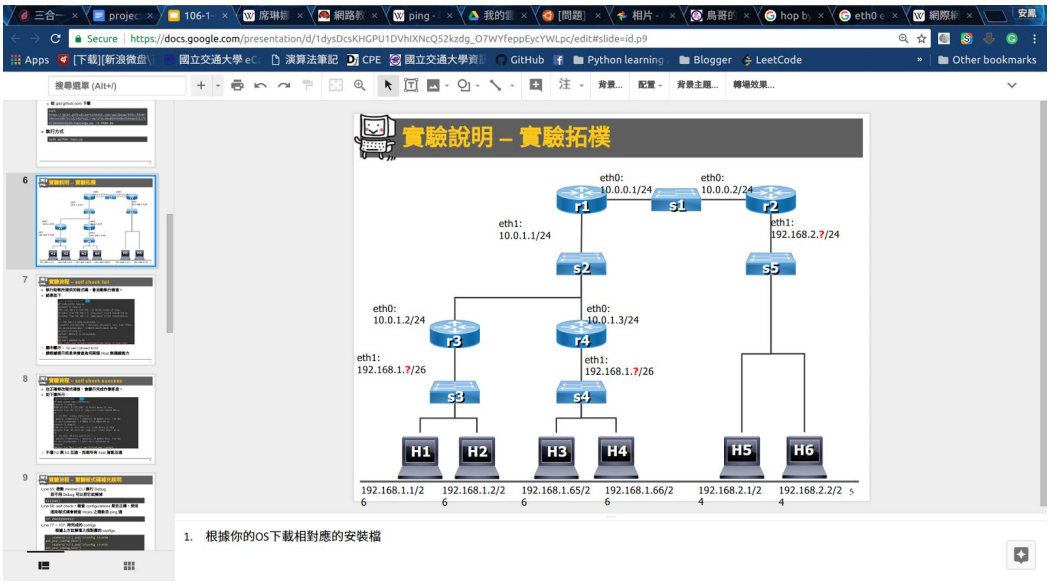
在 mininet CLI 使用 h1 traceroute h6，觀察由 h1 發出的 udp 封包，經過 r1 以後，為何 r1 的兩個 interface 觀察到的封包之 IP header checksum 不同。

header checksum作為檢查 IP header 之用，跟其他協定(例如：TCP、UDP等等)的 header 無關，因為其他協定的 header 有自己的 checksum 欄位。

另外，由於 checksum 是依據 IP header 中各欄位所計算出來的，因此在 IP datagram 經過 router 而 TTL 減 1 後，router 會重新計算 checksum 的值並繼續傳遞 IP datagram

6. 延續上題，不要關閉 wireshark，此時再用 h1 ping h6，觀察封包經過 r1 時，Layer 2 與 Layer 3 的 header 有何差異。

在hop by hop routing中 相鄰的router會利用MAC Address來溝通，而MAC Address和ARP協定相輔相成，由wireshark中我們可以看到



r1-eth0的mac addr

35 11.520277754 f2:68:fb:57:7b:68 5e:8f:0b:1c:0a:b0 ARP 42

10.0.1.1 is at f2:68:fb:57:7b:68

r1-eth1的mac addr

44 109.798767116 22:2a:99:df:78:9a 66:69:47:23:e9:6c ARP 42

10.0.0.1 is at 22:2a:99:df:78:9a

左方為sec 右方為dst

甲 傳過來r1之前eth1端口處

1.第二層ARP封包

```
51 116.224408882 5e:8f:0b:1c:0a:b0    f2:68:fb:57:7b:68    ARP          42 10.0.1.2 is at 5e:8f:0b:1c:0a:b0
```

2.第三層ICMP封包

```
52 117.152125487 192.168.1.1          192.168.2.2          ICMP         98 Echo (ping) request  id=0x53f8, seq=7/1792, ttl=63 (reply in 53)
```

乙 傳過來r1之後eth0端口處

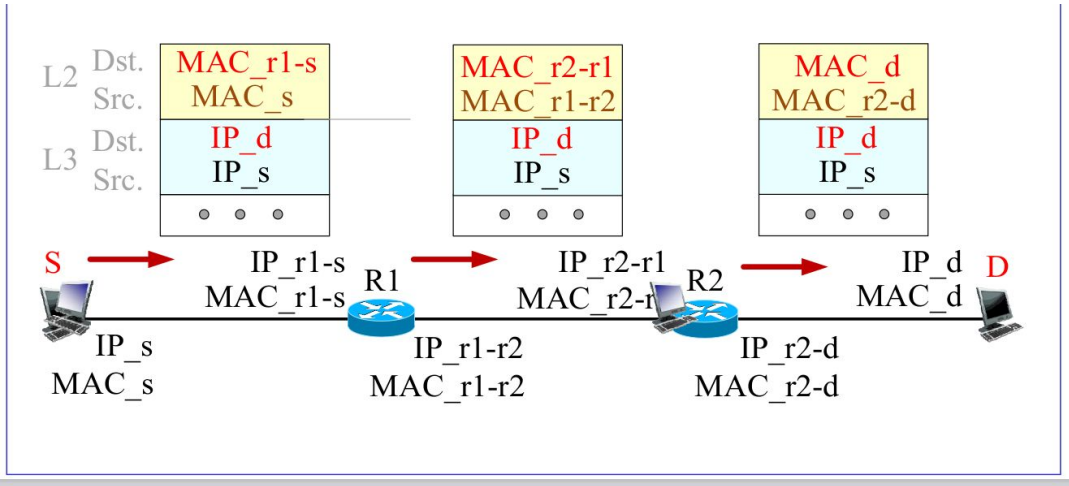
1.第二層ARP封包

```
44 109.798767116 22:2a:99:df:78:9a    66:69:47:23:e9:6c    ARP          42 10.0.0.1 is at 22:2a:99:df:78:9a
```

2.第三層ICMP封包

```
45 110.726622223 192.168.1.1          192.168.2.2          ICMP         98 Echo (ping) request  id=0x53f8, seq=7/1792, ttl=62 (reply in 46)
```

可以發現第二層 亦即連結層的封包src dst會改變，因為第二層連結層的路由器乃利用MAC Address和周圍的路由器交換資料，而第三層始終維持最起點與最末點的ip地址，這個就是hop by hop routing的概念

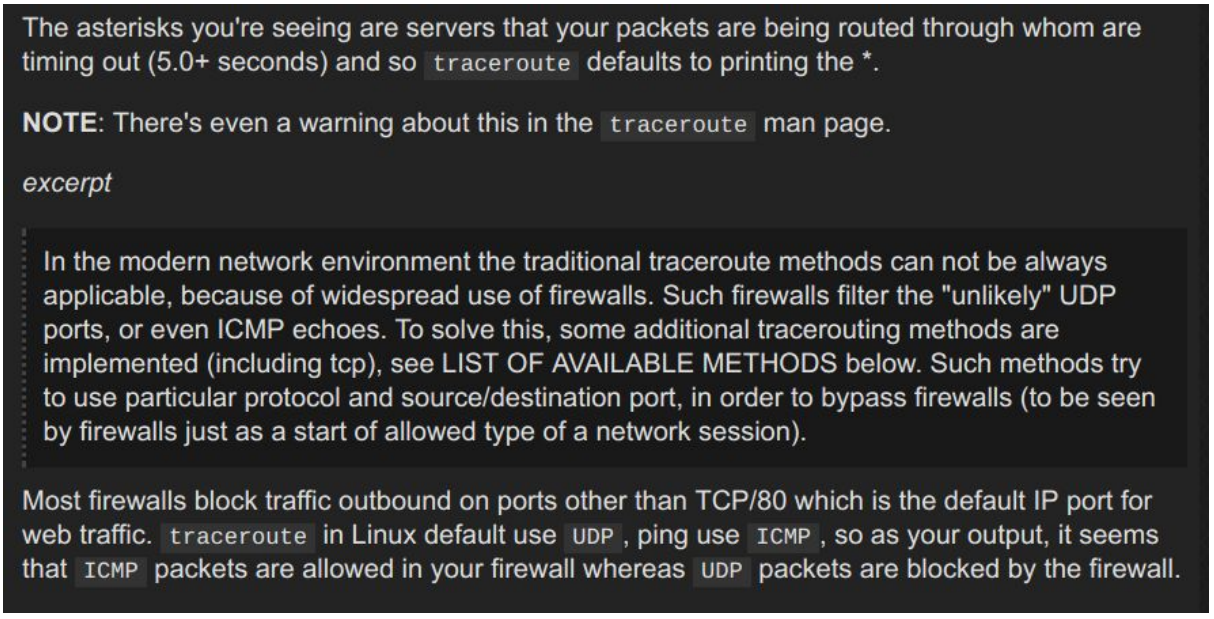


7.(與程式碼無關）在現實網路環境中，我們在使用 traceroute 時，偶爾會碰到有些節點內容不會顯示出來，而是以星號（*）來表示。請透過搜尋引擎學習 traceroute 的運作方式，並說明為什麼有時會有

星號出現？

程式是利用增加存活時間（TTL）值來實現其功能的。每當封包經過一個路由器，其存活時間就會減1。當其存活時間是0時，主機便取消封包，並傳送一個ICMP TTL封包給原封包的發出者。

而


The asterisks you're seeing are servers that your packets are being routed through whom are timing out (5.0+ seconds) and so `traceroute` defaults to printing the *.

NOTE: There's even a warning about this in the `traceroute` man page.

excerpt

> In the modern network environment the traditional traceroute methods can not be always applicable, because of widespread use of firewalls. Such firewalls filter the "unlikely" UDP ports, or even ICMP echoes. To solve this, some additional tracerouting methods are implemented (including tcp), see LIST OF AVAILABLE METHODS below. Such methods try to use particular protocol and source/destination port, in order to bypass firewalls (to be seen by firewalls just as a start of allowed type of a network session).

Most firewalls block traffic outbound on ports other than TCP/80 which is the default IP port for web traffic. `traceroute` in Linux default use `UDP`, ping use `ICMP`, so as your output, it seems that `ICMP` packets are allowed in your firewall whereas `UDP` packets are blocked by the firewall.

由stackoverflow可以知道，因為網路中許多路由器都有防火牆，因此有星號是因為路由器不會回應，不表示路由器掛掉，而是他們並不回應隨便路上撿來的封包，以確保安全 。

＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝我是心得分隔線＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

在編譯器 微處理機與機器學習的凌虐之下，還是遲交了幾十分鐘，然而這次的作業算是所有作業中最好玩的一個，讓我深深了解連結層與網路層的運作原理，也發現python程式碼的巧妙之處，在從圖片與python程式碼加上各種google觀念的協助，也得以完成這頗具挑戰卻又好玩的作業，程式碼的最後類似於judge的評測方式相當有趣，也為學期劃下完美的句點。