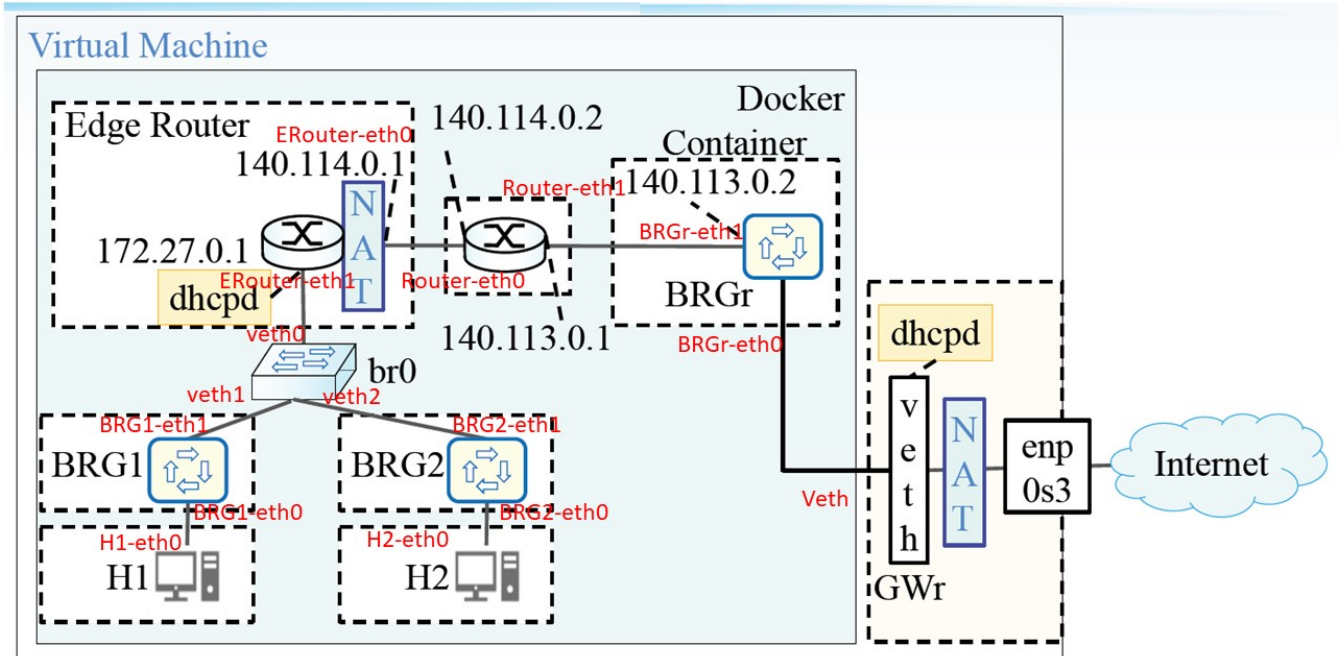


NS Project Report

Topology:



1. configuration commands:

A. BRG1: GRE over UDP (statically):

```
server_ip=$(ifconfig | sed -n '9p' | awk '{print $2;}')
ip fou add port 30000 ipproto 47
ip link add GRE1 type gretap remote 140.113.0.2 local $server_ip key 1 encap fou encap-sport 30000 encap-dport 50000
ip link set GRE1 up
ip link add br1 type bridge
brctl addif br1 BRG1-eth0
brctl addif br1 GRE1
ip link set br1 up
```

由上到下:

- 用 ifconfig 拿到自己 ip
- 設定 fou 的 port 和 ipproto 47 (gre)
- 設定通道 GRE1，remote 填 BRGr-eth1 ip、local 填自己 ip、key 填 1 (BRG2 那條填 2，區別用)、sport 填 i 設定的 port、dport 填 BRGr-eth1 那端的 port。
- 建立 GRE1
- 設定 bridge br1，並加入 BRG1-eth0 和通道 GRE1 兩個 interface，然後將其 set up。

B. BRGr: GRE over UDP (dynamically):

```

string cmd = "ip fou add port " + dst_port + " ipproto 47";
system(cmd.c_str());
cmd = "ip link add GRE" + to_string(tunnel_cnt) + " type gretap remote " + string(sourceIp) + " local 140.113.0.2 key " + to_string(tunnel_cnt) + " encap fou encap-sport " + dst_port + " encap-dport " + src_port;
system(cmd.c_str());
cmd = "ip link set GRE" + to_string(tunnel_cnt) + " up";
system(cmd.c_str());
if(tunnel_cnt == 1){
    system("ip link add br1 type bridge");
    system("brctl addif br1 BRGr-eth0");
}
cmd = "brctl addif br1 GRE" + to_string(tunnel_cnt);
system(cmd.c_str());
if(tunnel_cnt == 1){
    system("ip link set br1 up");
}
cout<<"\nCreate tunnel " << string(sourceIp) << ":" << src_port << " to " << "140.113.0.2:" << dst_port << endl;
tunnel_cnt++;

//update filter
string temp = filter;
filter = temp + " and not port " + dst_port;
const char* filter_exp = filter.c_str();
if (pcap_compile(descr, &fp, filter_exp, 0, net) == -1) {
    fprintf(stderr, "Couldn't parse filter %s: %s\n", filter_exp, pcap_geterr(descr));
    return;
}

if (pcap_setfilter(descr, &fp) == -1) {
    fprintf(stderr, "Couldn't install filter %s: %s\n", filter_exp, pcap_geterr(descr));
    return;
}

```

和上題 A 的指令類似，只是將 ip 和 port 的地方換成 parse 出來的結果。而因為對 BRG1 來說 BRGr 是 tunnel 的另一端，因此 encap-sport 和 encap-dport 那邊要相反。超過一條 tunnel 的話，bridge br1 不用重複建。而底下為 dynamic update filter 的部分，建過 tunnel 的 port 就不再 parse 封包。

C. Edge Router:

DHCP for BRG1, BRG2:

```
sudo docker exec -it EdgeRouter ip addr add 172.27.0.1/24 dev ERouter-eth1
```

設定 Edge Router interface 的 ip。

```

sudo docker cp dhcpd_edge.conf EdgeRouter:/dhcpd.conf
sudo docker exec -it EdgeRouter touch /var/lib/dhcp/dhcpd.leases
sudo docker exec -it EdgeRouter /usr/sbin/dhcpd 4 -pf /run/dhcp-server-dhcpd.pid -cf ./dhcpd.conf ERouter-eth1

```

將 dhcpd_edge.conf 複製進 Edge Router 並創立 leases 檔，再將 dhcp server run 在 ERouter-eth1 上。

```

subnet 172.27.0.0 netmask 255.255.255.0 {
    range 172.27.0.2 172.27.0.254;
    option routers 172.27.0.1;
    option subnet-mask 255.255.255.0;
}

```

dhcpd_edge.conf。

NAT rules for BRG1 (show NAT tables to justify your answer):

```
sudo docker exec -it EdgeRouter iptables -t nat -A POSTROUTING -o ERouter-eth0 -j MASQUERADE
```

設定 ERouter-eth0 為 nat 的 output interface。

```
root@4c8ecf339eda: /
yucheng@ubuntu:~/Desktop$ docker attach EdgeRouter
root@4c8ecf339eda:/# iptables -t nat -nvL
Chain PREROUTING (policy ACCEPT 21 packets, 3380 bytes)
pkts bytes target      prot opt in     out     source            destination
Chain INPUT (policy ACCEPT 14 packets, 2224 bytes)
pkts bytes target      prot opt in     out     source            destination
Chain OUTPUT (policy ACCEPT 3 packets, 125 bytes)
pkts bytes target      prot opt in     out     source            destination
Chain POSTROUTING (policy ACCEPT 3 packets, 125 bytes)
pkts bytes target      prot opt in     out     source            destination
  2    756 MASQUERADE  all  --  *      ERouter-eth0  0.0.0.0/0        0.0.0.0/0
root@4c8ecf339eda:/#
```

NAT tables in Edge Router ◦

D. GWr:

DHCP for hosts:

```
sudo ip link add BRGr-eth0 type veth peer name Veth
```

建立 veth pair ◦ BRGr-eth0 放入 BRGr 裡，Veth 留在本機端。

```
sudo ip link set dev Veth up
```

set up Veth ◦

```
sudo /usr/sbin/dhcpd 4 -pf /run/dhcp-server-dhcpd.pid -cf ./dhcpd.conf Veth
```

run server program on Veth ◦

```
subnet 20.0.0.0 netmask 255.0.0.0 {
    range 20.0.0.2 20.0.0.100;
    option routers 20.0.0.1;
    option subnet-mask 255.0.0.0;
    option domain-name-servers 8.8.8.8;
}
```

dhcpd.conf ◦

NAT rules for hosts (show NAT tables to justify your answer):

```
sudo iptables -t nat -A POSTROUTING -s 20.0.0.0/8 -j MASQUERADE
```

設定來自 20.0.0.0/8 的 ip 要經過 nat 到外網。

```
yucheng@ubuntu: ~/Desktop
yucheng@ubuntu:~/Desktop$ sudo iptables -t nat -nvL
Chain PREROUTING (policy ACCEPT 32 packets, 5825 bytes)
pkts bytes target      prot opt in     out     source            destination
Chain INPUT (policy ACCEPT 9 packets, 2705 bytes)
pkts bytes target      prot opt in     out     source            destination
Chain OUTPUT (policy ACCEPT 126 packets, 11533 bytes)
pkts bytes target      prot opt in     out     source            destination
Chain POSTROUTING (policy ACCEPT 128 packets, 13088 bytes)
pkts bytes target      prot opt in     out     source            destination
  9    629 MASQUERADE  all  --  *      *        20.0.0.0/8        0.0.0.0/0
Chain DOCKER (0 references)
pkts bytes target      prot opt in     out     source            destination
yucheng@ubuntu:~/Desktop$
```

NAT tables in host ◦

2. Show interfaces list on node BRGr and BRG1, 2:

BRGr:

```
yucheng@ubuntu:~$ docker attach BRGr
root@ef695e0d4345:/# ifconfig
BRGr-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    ether 8e:2c:36:a1:02:69 txqueuelen 1000 (Ethernet)
    RX packets 70 bytes 8350 (8.3 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 14 bytes 1964 (1.9 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

BRGr-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 140.113.0.2 netmask 255.255.255.0 broadcast 0.0.0.0
    ether c2:36:9c:e9:8d:5c txqueuelen 1000 (Ethernet)
    RX packets 73 bytes 11615 (11.6 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 87 bytes 12556 (12.5 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

GRE1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1450
    ether 56:10:c8:66:82:dc txqueuelen 1000 (Ethernet)
    RX packets 6 bytes 928 (928.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 38 bytes 3766 (3.7 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

GRE2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1450
    ether da:51:60:60:35:29 txqueuelen 1000 (Ethernet)
    RX packets 6 bytes 928 (928.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 35 bytes 3530 (3.5 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

br1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1450
    ether 56:10:c8:66:82:dc txqueuelen 1000 (Ethernet)
    RX packets 33 bytes 3526 (3.5 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 2 bytes 108 (108.0 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
    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
```

BRG1:

```
yucheng@ubuntu:~$ docker attach BRG1
root@9052eee0da36:/# ifconfig
BRG1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    ether b6:a2:7f:80:d4:2b txqueuelen 1000 (Ethernet)
    RX packets 6 bytes 928 (928.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 16 bytes 2110 (2.1 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

BRG1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.27.0.2 netmask 255.255.255.0 broadcast 172.27.0.255
    ether 22:f1:82:25:8d:65 txqueuelen 1000 (Ethernet)
    RX packets 95 bytes 12028 (12.0 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 17 bytes 2434 (2.4 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

GRE1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1450
    ether 06:5c:a1:1b:a5:ab txqueuelen 1000 (Ethernet)
    RX packets 14 bytes 2002 (2.0 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 8 bytes 924 (924.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

br1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1450
    ether 06:5c:a1:1b:a5:ab txqueuelen 1000 (Ethernet)
    RX packets 11 bytes 1646 (1.6 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 2 bytes 108 (108.0 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
    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

root@9052eee0da36:/#
```

BRG2:

```
yucheng@ubuntu:~$ docker attach BRG2
root@815a6fc2a584:/# ifconfig
BRG2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    ether 7a:d3:dc:26:9a:e5 txqueuelen 1000 (Ethernet)
    RX packets 6 bytes 928 (928.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 12 bytes 1342 (1.3 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

BRG2-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.27.0.3 netmask 255.255.255.0 broadcast 172.27.0.255
    ether aa:5d:7b:f9:02:d8 txqueuelen 1000 (Ethernet)
    RX packets 91 bytes 11410 (11.4 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 16 bytes 2392 (2.3 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

GRE2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1450
    ether 6a:92:94:c1:e3:60 txqueuelen 1000 (Ethernet)
    RX packets 10 bytes 1234 (1.2 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 8 bytes 924 (924.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

br1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1450
    ether 6a:92:94:c1:e3:60 txqueuelen 1000 (Ethernet)
    RX packets 7 bytes 934 (934.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 2 bytes 108 (108.0 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
    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

root@815a6fc2a584:/#
```

h1 ping google DNS server 8.8.8.8:

```
yucheng@ubuntu:~/Desktop$ docker attach H1
root@d9cdbc8a79f6:/# dhclient H1-eth0
mv: cannot move '/etc/resolv.conf.dhclient-new.24' to '/etc/resolv.conf': Device or resource busy
root@d9cdbc8a79f6:/# ifconfig
H1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 20.0.0.2 netmask 255.0.0.0 broadcast 20.255.255.255
    ether 56:66:5d:d8:2a:3d txqueuelen 1000 (Ethernet)
    RX packets 7 bytes 938 (938.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 4 bytes 788 (788.0 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
    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

root@d9cdbc8a79f6:/# ping 8.8.8.8 -c1
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=127 time=3.68 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 3.688/3.688/3.688/0.000 ms
root@d9cdbc8a79f6:/#
```

3. Capture packets and take screenshots on node:
BRG1 input/output:


```

root@fea78469bbff:/# tcpdump -i BRG1-eth0 -v
tcpdump: listening on BRG1-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
05:44:24.068085 IP (tos 0x0, ttl 64, id 55129, offset 0, flags [DF], proto ICMP (1), length 64)
  20.0.0.2 > 8.8.8.8: ICMP echo request, id 36, seq 1, length 64
05:44:24.072000 IP (tos 0x0, ttl 127, id 4100, offset 0, flags [none], proto ICMP (1), length 64)
  8.8.8.8 > 20.0.0.2: ICMP echo reply, id 36, seq 1, length 64
05:44:29.088317 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 20.0.0.1 tell 20.0.0.1, length 28
05:44:29.088364 ARP, Ethernet (len 6), IPv4 (len 4), Reply 20.0.0.2 is-at b2:bb:43:ab:46:01 (oui Unknown), length 28
05:44:29.088410 ARP, Ethernet (len 6), IPv4 (len 4), Reply 20.0.0.1 is-at 4a:db:99:bc:d7:3c (oui Unknown), length 28
^C
6 packets captured
6 packets received by filter
0 packets dropped by kernel
root@fea78469bbff:/#

root@fea78469bbff:/# tcpdump -i BRG1-eth1 -v
tcpdump: listening on BRG1-eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
05:44:24.068125 IP (tos 0x0, ttl 64, id 54287, offset 0, flags [DF], proto UDP (17), length 106)
  172.27.0.2.30000 > 140.113.0.2.50000: UDP, length 106
05:44:24.068401 IP (tos 0x0, ttl 64, id 18883, offset 0, flags [DF], proto UDP (17), length 66)
  172.27.0.2.33171 > 192.168.171.2.domain: 45440+ PTR? 8.8.8.8.in-addr.arpa. (38)
05:44:24.068483 IP (tos 0x0, ttl 64, id 18886, offset 0, flags [DF], proto UDP (17), length 70)
  172.27.0.2.38865 > 192.168.171.2.domain: 26772+ PTR? 2.0.113.140.in-addr.arpa. (42)
05:44:24.068489 IP (tos 0xc0, ttl 64, id 13310, offset 0, flags [none], proto ICMP (1), length 94)
  172.27.0.1 > 172.27.0.2: ICMP net 192.168.171.2 unreachable, length 74
  IP (tos 0x0, ttl 64, id 18885, offset 0, flags [DF], proto UDP (17), length 66)
  172.27.0.2.33171 > 192.168.171.2.domain: 45440+ PTR? 8.8.8.8.in-addr.arpa. (38)
05:44:24.068507 IP (tos 0xc0, ttl 64, id 13311, offset 0, flags [none], proto ICMP (1), length 98)
  172.27.0.1 > 172.27.0.2: ICMP net 192.168.171.2 unreachable, length 78
  IP (tos 0x0, ttl 64, id 18886, offset 0, flags [DF], proto UDP (17), length 70)
  172.27.0.2.38865 > 192.168.171.2.domain: 26772+ PTR? 2.0.113.140.in-addr.arpa. (42)
05:44:24.072644 IP (tos 0x0, ttl 125, id 60632, offset 0, flags [DF], proto UDP (17), length 106)
  140.113.0.2.50000 > 172.27.0.2.30000: UDP, length 106
  
```

從 h1 送出、BRG1-eth0 收到的 icmp request，src ip 是 h1 的 ip 20.0.0.2，dst ip 是 8.8.8.8；從 BRG1-eth1 送出的 src ip 是 BRG1-eth1 的 ip 172.27.0.2、port 30000，dst ip 是 BRGr-eth1 的 ip 140.113.0.2、port 50000。因為這裡走的是一個 gre fou tunnel，它的 src、dst 就分別是通道兩端的端口。而回來的 icmp reply 也是同理。

Edge Router input/output:

```

root@784b8587b561:/# tcpdump -i ERouter-eth1 -v
tcpdump: listening on ERouter-eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
03:52:05.954969 IP (tos 0x0, ttl 64, id 13661, offset 0, flags [DF], proto UDP (17), length 134)
  172.27.0.2.30000 > 140.113.0.2.50000: UDP, length 106
03:52:05.959333 IP (tos 0x0, ttl 125, id 29879, offset 0, flags [DF], proto UDP (17), length 134)
  140.113.0.2.50000 > 172.27.0.2.30000: UDP, length 106

root@784b8587b561:/# tcpdump -i ERouter-eth0 -v
tcpdump: listening on ERouter-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
03:52:05.954993 IP (tos 0x0, ttl 63, id 13661, offset 0, flags [DF], proto UDP (17), length 134)
  140.114.0.1.30000 > 140.113.0.2.50000: UDP, length 106
03:52:05.959326 IP (tos 0x0, ttl 126, id 29879, offset 0, flags [DF], proto UDP (17), length 134)
  140.113.0.2.50000 > 140.114.0.1.30000: UDP, length 106
  
```

而 Edge Router 負責實際轉送 tunnel 封包。ERouter-eth1 的 src、dst 和上圖 BRG1-eth1 一樣；而 ERouter-eth0 的 dst 一樣，但 src ip 部分變成了 ERouter-eth0 自己本身的 ip 140.114.0.1。這裡要經過一個 nat，由自己的 ip:port 對應 dst 的 ip:port。而反向也是同理。

BRGr input/output:

```
root@fec38a34a6ab:/# tcpdump -i BRGr-eth1 -v
tcpdump: listening on BRGr-eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
05:56:53.357058 IP (tos 0x0, ttl 62, id 21817, offset 0, flags [DF], proto UDP (17), length 134)
  140.114.0.1.30000 > 140.113.0.2.50000: UDP, length 106
05:56:53.361410 IP (tos 0x0, ttl 127, id 60141, offset 0, flags [DF], proto UDP (17), length 134)
  140.113.0.2.50000 > 140.114.0.1.30000: UDP, length 106
05:56:58.399912 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 140.113.0.1 tell 140.113.0.2, length 28
05:56:58.400207 IP (tos 0x0, ttl 64, id 60722, offset 0, flags [DF], proto UDP (17), length 84)
  140.113.0.2.50000 > 140.114.0.1.30000: UDP, length 106

root@fec38a34a6ab:/# tcpdump -i BRGr-eth0 -v
tcpdump: listening on BRGr-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
05:56:53.357082 IP (tos 0x0, ttl 64, id 39733, offset 0, flags [DF], proto ICMP (1), length 84)
  20.0.0.2 > 8.8.8.8: ICMP echo request, id 37, seq 1, length 64
05:56:53.357709 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 20.0.0.1 tell 140.113.0.2, length 28
05:56:53.357722 ARP, Ethernet (len 6), IPv4 (len 4), Reply 20.0.0.1 is-at 4a:db:99:bc:d7:3c (oui Unknown), length 28
05:56:53.361395 IP (tos 0x0, ttl 127, id 4240, offset 0, flags [none], proto ICMP (1), length 84)
  20.0.0.2 > 8.8.8.8: ICMP echo request, id 37, seq 1, length 64
05:56:58.400207 IP (tos 0x0, ttl 64, id 60722, offset 0, flags [DF], proto UDP (17), length 84)
  140.113.0.2.50000 > 140.114.0.1.30000: UDP, length 106
```

BRGr-eth1 的 src、dst 和上圖 ERouter-eth0 一樣；而 BRGr-eth0 是

對應 BRG1-eth0 的 tunnel 的另一端，它會將 tunnel 的 header 解

開，還原出原本的 src、dst。因此它的 src、dst 和 BRG1-eth0 一

樣。反向同理。

GWr input/output:

```
yucheng@ubuntu:~$ sudo tcpdump -i Veth -v
tcpdump: listening on Veth, link-type EN10MB (Ethernet), capture size 262144 bytes
14:15:45.399639 IP (tos 0x0, ttl 64, id 30936, offset 0, flags [DF], proto ICMP (1), length 84)
  20.0.0.2 > dns.google: ICMP echo request, id 69, seq 1, length 64
14:15:45.403728 IP (tos 0x0, ttl 127, id 165, offset 0, flags [none], proto ICMP (1), length 84)
  dns.google > 20.0.0.2: ICMP echo reply, id 69, seq 1, length 64
14:15:50.478437 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 20.0.0.2 tell ubuntu, length 28
14:15:50.478671 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has ubuntu tell 20.0.0.2, length 28
14:15:50.478685 ARP, Ethernet (len 6), IPv4 (len 4), Reply ubuntu is-at 0e:2d:af:d4:1a:b7 (oui Unknown), length 28
14:15:50.478761 ARP, Ethernet (len 6), IPv4 (len 4), Reply 20.0.0.2 is-at 2e:9e:84:fd:61:d2 (oui Unknown), length 28
6 packets captured
6 packets received by filter
0 packets dropped by kernel

yucheng@ubuntu:~$ sudo tcpdump ip -i yuyucheng@ubuntu:~$ sudo tcpdump ip -i ens33
tcpdump: listening on ens33, link-type EN10MB (Ethernet), capture size 262144 bytes
14:15:45.399652 IP (tos 0x0, ttl 63, id 30936, offset 0, flags [DF], proto ICMP (1), length 84)
  20.0.0.2 > dns.google: ICMP echo request, id 69, seq 1, length 64
14:15:45.403728 IP (tos 0x0, ttl 127, id 165, offset 0, flags [none], proto ICMP (1), length 84)
  dns.google > 20.0.0.2: ICMP echo reply, id 69, seq 1, length 64
14:15:50.478437 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 20.0.0.2 tell ubuntu, length 28
14:15:50.478671 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has ubuntu tell 20.0.0.2, length 28
14:15:50.478685 ARP, Ethernet (len 6), IPv4 (len 4), Reply ubuntu is-at 0e:2d:af:d4:1a:b7 (oui Unknown), length 28
14:15:50.478761 ARP, Ethernet (len 6), IPv4 (len 4), Reply 20.0.0.2 is-at 2e:9e:84:fd:61:d2 (oui Unknown), length 28
6 packets captured
6 packets received by filter
0 packets dropped by kernel
```

GWr 會將 icmp request 封包穿過 nat 送到外網。Veth 收到的封包

src 是 H1 的 20.0.0.2，dst 是 8.8.8.8；穿過 nat 後，由本機的

ubuntu ip 送到 dns.google。反向同理。

4. How BRGr determines the GRE interface to tunnel the response packets back to BRG1:

比較 H1 和 H2 的 ping reply:

<pre> root@fec38a34a6ab:/# tcpdump -i BRGr-eth1 -v tcpdump: listening on BRGr-eth1, link-type EN10MB (Ethernet), capture size 262144 bytes 05:56:53.357058 IP (tos 0x0, ttl 62, id 21817, offset 0, flags [DF], proto UDP (17), length 134) 140.114.0.1.30000 > 140.113.0.2.50000: UDP, length 106 05:56:53.361410 IP (tos 0x0, ttl 127, id 60141, offset 0, flags [DF], proto UDP (17), length 134) 140.113.0.2.50000 > 140.114.0.1.30000: UDP, length 106 8.8.8.8 -> H1 05:56:53.359912 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 140.113.0.1 tell 140.113.0.2, length 28 05:56:58.400207 IP (tos 0x0, ttl 64, id 60722, offset 0, flags [DF], proto UDP (17), length 78) 140.114.0.1.20000 > 140.113.0.2.40000: UDP, length 106 10:13:20.307146 IP (tos 0x0, ttl 62, id 48626, offset 0, flags [DF], proto UDP (17), length 134) 140.113.0.2.40000 > 140.114.0.1.20000: UDP, length 106 8.8.8.8 -> H2 10:13:25.534552 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 140.113.0.1 tell 140.113.0.2, length 28 10:13:25.535024 IP (tos 0x0, ttl 64, id 65004, offset 0, flags [DF], proto UDP (17), length 78) </pre>	<pre> root@fec38a34a6ab:/# tcpdump -i BRGr-eth0 -v tcpdump: listening on BRGr-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes 05:56:53.357082 IP (tos 0x0, ttl 64, id 39733, offset 0, flags [DF], proto ICMP (1), length 84) 20.0.0.2 > 8.8.8.8: ICMP echo request, id 37, seq 1, length 64 05:56:53.357709 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 20.0.0.1 tell 140.113.0.2, length 28 05:56:53.357722 ARP, Ethernet (len 6), IPv4 (len 4), Reply 20.0.0.1 is-at 4a:db:99:bc:d7:3c (oui Unknown), length 28 05:56:53.361395 IP (tos 0x0, ttl 127, id 4240, offset 0, flags [none], proto ICMP (1), length 84) 8.8.8.8 > 20.0.0.2: ICMP echo reply, id 37, seq 1, length 64 root@fec38a34a6ab:/# tcpdump -i BRGr-eth0 -v tcpdump: listening on BRGr-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes 10:13:20.307183 IP (tos 0x0, ttl 64, id 25119, offset 0, flags [DF], proto ICMP (1), length 84) 20.0.0.3 > 8.8.8.8: ICMP echo request, id 37, seq 1, length 64 10:13:20.307665 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 20.0.0.1 tell 140.113.0.2, length 28 10:13:20.307674 ARP, Ethernet (len 6), IPv4 (len 4), Reply 20.0.0.1 is-at 4a:db:99:bc:d7:3c (oui Unknown), length 28 10:13:20.311796 IP (tos 0x0, ttl 127, id 7482, offset 0, flags [none], proto ICMP (1), length 84) 8.8.8.8 > 20.0.0.3: ICMP echo reply, id 37, seq 1, length 64 </pre>
---	--

由上圖可以發現，雖然由於 nat 的關係，兩者的 src ip 和 dst ip 是一樣的，但其 port 不一樣。而 BRGr 就是透過這點來分辨要走哪個 GRE interface。若是 50000 => 30000，則走 GRE1，經過 tunnel 送到 BRG1 回到 H1；若是 40000=>20000，則走 GRE2，經過 tunnel 送到 BRG2 回到 H2。指令上用 key 可以區別兩個通道。