Writing Task 2

1.

The source MAC address.

2.

1674.

3.

IPv4: 20 bytes. IPv6: 40 bytes.

Run program and tests

First change the destination IP address in src/tests/ns1.c . In every test, ns1 regularly sends packets to a fixed IP address.

Run sudo make.

cd vnetUtils/examples, sudo bash ./makeVNet < test1.txt or test2.txt, test3.txt.

The three test networks are:

Then open a terminal with every ns hosts.

```
cd vnetUtils/helper;
sudo ./execNS ns* bash
cd ../../build
```

Then sudo ./ns1 or sudo ./router . We can also sudo ./ns1 in another ns to make it the packet sender.

Programming Task 3

I implemented these functions in arp.h/c:

```
int getMACaddress(struct in_addr *target_ip, uint8_t *mac_address, int last_id);

void sendARPrequest(struct in_addr *target_ip, int last_id);

void sendARPreply(struct in_addr *dest_ip, uint8_t *dest_mac_address, uint8_t
 *source_mac_address, int last_id);

void processARPrequest(const uint8_t *packet, int last_id);

void processARPreply(const uint8_t *packet);
```

I implemented these functions in ip.h/c:

```
int sendIPPacket(const struct in_addr src , const struct in_addr dest ,
int proto , const void *buf , int len, int TTL);

void processARPpacket(const uint8_t *packet, int last_id);
void processIPpacket(const uint8_t *packet, int deviceID);
```

I implemented these functions in rip.h/c:

```
void initDVTrie(void);
void insertDVTrie(struct in_addr ip, struct in_addr mask, int hops, int deviceID);
void sendRIPpacket(void);
int setRoutingTable(struct in_addr dest, struct in_addr mask, const char *device);
void processRIPpacket(const uint8_t *packet, int payloadLength, int deviceID);
int route(const struct in_addr ip);
TrieNode *getDVTrieRoot(void);
```

Writing Task 3

I implemented the ARP. If the caller of sendFrame() doesn't know the MAC address corresponded to the IP address, it broadcasts ARP packets to adjacent hosts.

Here's an example with kernel protocol on: (the wireshark monitors vnet2-1 on ns2)

```
10.000000000
                  0a:6c:f1:1a:e... Broadcast
                                                  ARP
                                                               ...Who has 10.100.1.2? Tell 10.100.1.1
                                                               ...10.100.1.2 is at 8e:04:74:a4:c5:31
20.000062172
                  8e:04:74:a4:c... 0a:6c:f1:1a:e7... ARP
                                                  ARP
32.389697424
                  8e:04:74:a4:c... Broadcast
                                                               ...Who has 10.100.1.1? Tell 10.100.1.2
                                                               ...10.100.1.1 is at 0a:6c:f1:1a:e7:90
42.389719415
                  0a:6c:f1:1a:e... 8e:04:74:a4:c5... ARP
52.389773878
                                                  ARP
                                                               ...Who has 10.100.2.2? Tell 10.100.1.2
                  8e:04:74:a4:c... Broadcast
62.389814937
                                                               ...Who has 10.100.4.2? Tell 10.100.1.2
                  8e:04:74:a4:c... Broadcast
73.029161012
                  0a:6c:f1:1a:e... 8e:04:74:a4:c5... ARP
                                                               ...10.100.1.1 is at 0a:6c:f1:1a:e7:90
```

With kernel protocol on, the ARP request receives 2 ARP replies.

After receiving an ARP reply, the host adds the info to its ARP cache.

Writing Task 4

I implemented an simplified version of RIP. I maintained a routing table on every host. I chose Trie to easily handle IPmasks and longest prefix match. Every host in network regularly (every 5 seconds) sends its info to adjacent hosts.

```
typedef struct TrieNode {
   struct in_addr ip;
   struct in_addr mask;

int hops;
   int attenuate_timer;

int deviceID;

struct TrieNode *ch[2];
} TrieNode;
```

hops marks the distance between this host and the target subnet. The info with minimum hops takes precedence.

To deal with broken hosts, I used the attenuate_timer . If the current info is out-dated, it will be replaced by another info (likely with a larger hops). Each time a worse info arrives, attenuate_timer++ .

Here's the whole replacing policy: (tmp is the old info)

```
if (hops + 1 < tmp->hops || tmp->attenuate_timer >= 5) {
    tmp->ip = ip;
    tmp->mask = mask;
    tmp->hops = hops + 1;
    tmp->attenuate_timer = 0;
    tmp->deviceID = deviceID;
} else if (hops + 1 == tmp->hops) {
    tmp->deviceID = deviceID;
    tmp->attenuate_timer = 0;
} else if (hops + 1 > tmp->hops && tmp->hops > 1) {
    // tmp->hops = 0 means this is local IP, or this is manually added(with highest priority)
```

```
// tmp->hops = 1 means this is direct link, don't change
tmp->attenuate_timer++;
}
```

I used protocol ID OXFE to mark these RIP packets. And for simplicity, I compressed each info to 12 bytes.

```
.10.000494239
                     10.100.1.2
                                     10.100.1.1
                                                       IPv4
                                                                    ... Unknown (254)
  ... 10.940328297
                                                       IPv4
                                                                    ... Unknown (254)
                     10.100.1.1
                                     10.100.1.2
                                     10.100.1.1
                                                       IPv4
  ... 15.000727824
                     10.100.1.2
                                                                   ... Unknown (254)
  ... 15.940560864
                     10.100.1.1
                                     10.100.1.2
                                                      TPv4
                                                                   ... Unknown (254)
                                  10.100.1.1
  ... 20.000905306
                     10.100.1.2
                                                      IPv4
                                                                   ... Unknown (254)
   20 040065272
                     10 100 1 1
                                     10 100 1 7
                                                      TDv/
                                                                     Unknown (254)
Frame 10: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface veth2-1, id 0
Ethernet II, Src: 8e:04:74:a4:c5:31 (8e:04:74:a4:c5:31), Dst: 0a:6c:f1:1a:e7:90 (0a:6c:f1:1a:e7:90)
Internet Protocol Version 4, Src: 10.100.1.2, Dst: 10.100.1.1
 0100 .... = Version: 4
 .... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
 Total Length: 56
 Identification: 0x0000 (0)
Flags: 0x00
 ...0 0000 0000 0000 = Fragment Offset: 0
 Time to Live: 64
Protocol: Unknown (254)
0000 0a 6c f1 1a e7 90 8e 04 74 a4 c5 31 08 00 45 00 0010 00 38 00 00 00 00 40 fe 00 00 0a 64 01 02 0a 64
                                                              · l · · · · · · · t · · 1 · · E ·
                                                             ·8····d···d
                                                             · · · d · · · · · · · · d
0020 01 01 0a 64 01 02 ff ff ff 00 01 00 00 00 0a 64
0030 02 01 ff ff ff 00 01 00 00 00 0a 64 04 01 ff ff
                                                             ....d....
0040 ff 00 01 00 00 00
```

Checkpoint 3

In this network, ns1 regularly sends IPv4 packets to ns6, with protocol ID 0xFD: (the wireshark monitors vnet6-5 on ns6)

```
ip.version==4 && ip.proto==0xfd
                                                            IPv4
    13.859065039
                        10.100.1.1
                                         10.100.6.2
  ... 14.899008247
                        10.100.1.1
                                         10.100.6.2
                                                            IPv4
                                                                           ... Unknown (253)
  ... 15.939053339
                        10.100.1.1
                                         10.100.6.2
                                                            IPv4
                                                                           ... Unknown
                                                                                     (253)
                                                            IPv4
  ... 16.979147172
                        10.100.1.1
                                         10.100.6.2
                                                                          ... Unknown (253)
  ... 16.979165247
                        10.100.1.1
                                         10.100.6.2
                                                            TPv4
                                                                          ... Unknown (253)
  ... 18.019343881
                        10.100.1.1
                                         10.100.6.2
                                                            IPv4
                                                                          ... Unknown (253)
  ... 19.059126840
                                                            IPv4
                        10.100.1.1
                                         10.100.6.2
                                                                          ... Unknown (253)
  ... 20.099162477
                        10.100.1.1
                                         10.100.6.2
                                                            IPv4
                                                                          ... Unknown (253)
  ... 21.138995144
                        10.100.1.1
                                         10.100.6.2
                                                            IPv4
                                                                           ... Unknown (253)
```

```
Frame 12: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface veth6-5, id 0
Ethernet II, Src: ba:8a:31:09:c5:6b (ba:8a:31:09:c5:6b), Dst: 2e:2a:7f:59:cb:ef (2e:2a:7f:59:cb:ef)
Internet Protocol Version 4, Src: 10.100.1.1, Dst: 10.100.6.2
Data (12 bytes)
```

Data: 48656c6c6f20576f726c6421

[Length: 12]

The first 12 bytes mark the source MAC address and the dest MAC address:

```
0000
                  59
                             ba 8a
                                     31 09
                                            c5 6b
                                                    08
                                                           45
                                                               00
       2e
          2a
                      cb
                         ef
                                                       00
0010
       00
          20
              00
                  00
                     00
                         00
                             3e
                                fd
                                     00
                                         00
                                            0a
                                                64
                                                    01
                                                       01 0a
                                                               64
          02
              48
                  65
                     6c
                        6c
                             6f
                                20
                                     57
                                         6f
                                            72
                                                6c 64 21 00
0020
       06
0030
          00
              00
                  00
                     00
                         00
                             00
                                00
                                         00
                                             00
                                                00
       00
                                     00
0000
          2a
              7f
                 59
                     cb ef
                            ba 8a
                                     31
                                        09
                                            c5
                                                   08
                                                       00 45
                                                              00
       2e
                                               6bl
                            3e fd
0010
          20
              00
                  00
                     00
                         00
                                     00
                                        00
                                            0a
                                               64
                                                   01
       00
                                                       01
                                                          0a
                                                              64
          02
                            6f
                                20
                                        6f
                                            72
       06
              48
                  65
                     6c
                         6c
                                     57
                                               6c 64 21 00 00
0030
       00 00 00 00 00 00 00 00
                                     00
                                        00
                                           00
                                               00
```

The 13th and 14th bytes are 0x0800, which shows this is an IPv4 packet.

```
08
0000
       2e
          2a
              7f
                  59
                      cb ef ba 8a
                                      31
                                         09
                                             c5
                                                 6b
                                                        00
                                                            45
                                                                00
                             3e
                                 fd
          20
              00
                  00
                      00
                         00
                                      00
                                                 64
                                                     01
                                                        01
0010
       00
                                         00
                                             0a
                                                            0a
                                                                64
          02
                             6f
                                 20
                                      57
                                         6†
                                             72
                                                        21
0020
              48
                  65
                      6c
                         6c
                                                 6c
                                                     64
                                                            00
       06
0030
       00 00 00 00 00 00 00 00
                                      00
                                         00
                                             00
                                                 00
```

The 15th byte is 0x45, 0100 0101 under binary. 0100 shows the IP version is 4, 0101 shows the header length is $5\times 4=20$ bytes.

```
59
                            ba 8a
                                            c5
                                                              00
0000
             7f
                     cb ef
                                     31 09
                                                6b
                                                   08
                                                       00
0010
          20
              00
                 00
                     00
                         00
                            3e
                               fd
                                     00
                                        00
                                            0a
                                                64
                                                   01
                                                       01
                                                              64
       00
                                            72
0020
       06
          02
              48
                 65
                     6c
                         6c 6f
                               20
                                     57
                                        6f
                                                6c
                                                  64 21 00
                                                              00
0030
       00 00
              00
                 00
                     00 00 00 00
                                     00 00
                                           00
                                               00
```

The 16th byte is 0x00. It's the TOS byte.

```
      0000
      2e
      2a
      7f
      59
      cb
      ef
      ba
      8a
      31
      09
      c5
      6b
      08
      00
      45
      00

      0010
      00
      20
      00
      00
      00
      3e
      fd
      00
      00
      0a
      64
      01
      01
      0a
      64

      0020
      06
      02
      48
      65
      6c
      6c
      6f
      20
      57
      6f
      72
      6c
      64
      21
      00
      00

      0030
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
      00
```

The 17th and 18th bytes are 0x0020, showing the total packet length is 32 bytes.

```
2e 2a 7f 59 cb ef ba 8a
0000
                                31 09 c5 6b 08 00 45
                                                      00
      00 20
            00 00 00 00 3e fd
0010
                                00 00 0a 64 01
                                               01 0a
                                                      64
0020
      06 02 48 65 6c 6c 6f 20
                                57 6f 72 6c 64 21 00 00
      00 00 00 00 00 00 00 00
                                00 00 00 00
0030
```

The 19th, 20th, 21st, 22nd bytes are all 0x00, they are identification code and flags, manually set to 0 without fragmentation.

```
0000
      2e 2a 7f 59 cb ef ba 8a
                                31 09 c5 6b 08 00 45
                                                      00
0010
      00 20
            00 00 00 00 3e fd
                                00 00 0a 64 01 01 0a 64
      06 02 48 65 6c 6c 6f 20
                                57 6f 72 6c 64 21 00 00
0020
      00 00 00 00 00 00 00 00
0030
                                00 00
                                      00 00
0000
      2e 2a 7f 59 cb ef ba 8a
                                31 09 c5
                                         6b 08 00 45
                                                      00
                  00 00
0010
      00 20 00 00
                        3e fd
                                00 00
                                      0a 64 01 01 0a 64
0020
      06 02 48 65 6c 6c 6f 20
                                57 6f
                                      72 6c 64 21 00 00
0030
      00 00 00 00 00 00 00 00
                                00 00 00 00
```

The 23rd byte is 0x3e, showing this packet's TTL is 62.

```
      0000
      2e 2a 7f 59 cb ef ba 8a
      31 09 c5 6b 08 00 45 00

      0010
      00 20 00 00 00 00 3e fd
      00 00 0a 64 01 01 0a 64

      0020
      06 02 48 65 6c 6c 6f 20
      57 6f 72 6c 64 21 00 00

      0030
      00 00 00 00 00 00 00 00 00 00
      00 00 00 00
```

The 24th byte is 0xfd, the protocol ID.

```
      0000
      2e 2a 7f 59 cb ef ba 8a
      31 09 c5 6b 08 00 45 00

      0010
      00 20 00 00 00 00 3e fd
      00 00 0a 64 01 01 0a 64

      0020
      06 02 48 65 6c 6c 6f 20
      57 6f 72 6c 64 21 00 00

      0030
      00 00 00 00 00 00 00 00 00 00
      00 00 00 00
```

The 25th and 26th bytes are 0x0000, disabling the header checksum.

```
      0000
      2e 2a 7f 59 cb ef ba 8a
      31 09 c5 6b 08 00 45 00

      0010
      00 20 00 00 00 00 3e fd
      00 00 0a 64 01 01 0a 64

      0020
      06 02 48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00 00

      0030
      00 00 00 00 00 00 00 00 00 00 00
```

The following 8 bytes mark the source IP address 10.100.1.1 and the dest IP address 10.100.6.2.

```
7f 59 cb ef ba 8a
0000
          2a
                                   31 09 c5 6b 08 00 45
                                                           00
      2e
0010
      00
          20
             00 00 00 00
                          3e fd
                                   00
                                      00
                                          0a
                                             64
                                                01
                                                    01
                                                       0a
                                                           64
                              20
                                      6f
                                          72
                                             6c 64
                                                    21 00 00
      06 02
             48
                 65
                    6c
                       6c 6f
                                   57
0020
0030
      00 00 00 00 00 00 00 00
                                   00 00
                                          00
                                             00
0000
       2e 2a
             7f
                59 cb ef ba 8a
                                   31 09
                                          c5
                                             6b
                                                08
                                                    00
                                                       45
                                                          00
          20
                                             64
0010
       00
              00
                 00
                    00
                        00
                           3e
                              fd
                                   00
                                      00
                                          0a
                                                01
                                                    01
                                                          64
       06 02
              48 65 6c 6c 6f
                              20
                                   57
                                      6f
                                          72
                                             6c
                                                64
                                                    21 00
0020
                                                          00
                                         00
                                             00
0030
       00
          00 00 00 00 00 00 00
                                   00 00
```

The following 12 bytes are the payload Hello World!

```
      0000
      2e 2a 7f 59 cb ef ba 8a
      31 09 c5 6b 08 00 45 00
      .*·Y···· 1··k··E·

      0010
      00 20 00 00 00 00 3e fd
      00 00 0a 64 01 01 0a 64

      0020
      06 02 48 65 6c 6c 6f 20
      57 6f 72 6c 64 21 00 00

      0030
      00 00 00 00 00 00 00 00 00 00
      00 00 00 00
```

The rest bytes are ethernet padding.

```
0000
      2e 2a 7f 59 cb ef ba 8a
                                  31 09 c5
                                             6b 08
                                                   00 45
                                                          00
0010
      00
          20
             00
                00
                    00 00
                          3e fd
                                  00 00
                                         0a
                                             64 01
                                                   01
                                                       0a
                                                          64
             48
                65
                    6c 6c
0020
      06
          02
                           6f
                              20
                                   57
                                      6f
                                         72
                                             6c
                                                64
                                                   21
                                                          00
0030
      00 00
             00
                00 00 00 00
                                   00
                                      00
                                         00
                                             00
                              00
```

Checkpoint 4

ns1 regularly sends packets to ns4.

In the beginning, ns4 can receive the packets.

```
Πv
     1题 4 输出 调试控制台 终端 端口
                                                                                                                                                                                                                                                                                                                             bash lab-netstack-premium
  devices: 0, veth4-3, (null)
0xffffff
                                                                                                                                                                                                                                                                                                                                  addr: 10.100.3.2
mask: 255.255.255.0
mask: 255.255.255.0
devices: 1, any, Pseudo-device that captures on all interfaces devices: 2, bluetooth-monitor, Bluetooth Linux Monitor devices: 3, nflog, Linux netfilter log (NFLOG) interface devices: 4, nfqueue, Linux netfilter queue (NFQUEUE) interface devices: 5, dbus-system, D-Bus system bus devices: 6, dbus-session, D-Bus session bus devices: 7, lo, (null) src = 10.100.1.1 dest = 10.100.3.2 packet arrived, TTL = 62 src = 10.100.1.1 dest = 10.100.3.2
                                                                                                                                                                                                                                                                                                                                 ns2 helper
                                                                                                                                                                                                                                                                                                                                  ns3 helper
                                                                                                                                                                                                                                                                                                                              ns4 helper
                                                                                                                                                                                                                                                                                                                                 ns5 helper
                                                                                                                                                                                                                                                                                                                                  ns6 helper
                                                                                                                                                                                                                                                                                                                                   wireshark helper
  dest = 10.100.3.2
packet arrived, TTL = 62
src = 10.100.1.1
  dest = 10.100.3.2
packet arrived, TTL = 62
src = 10.100.1.1
 dest = 10.100.3.2
packet arrived, TTL = 62
src = 10.100.1.1
dest = 10.100.3.2
packet arrived, TTL = 62
```

After disconnecting ns2, ns4 stops receiving packets.

After reconnecting ns2, ns4 can receive the packets.



Checkpoint 5

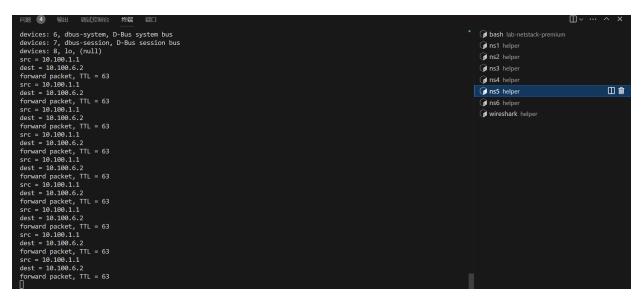
The distances are:

$$1-2:1, 2-3:1,\ldots$$

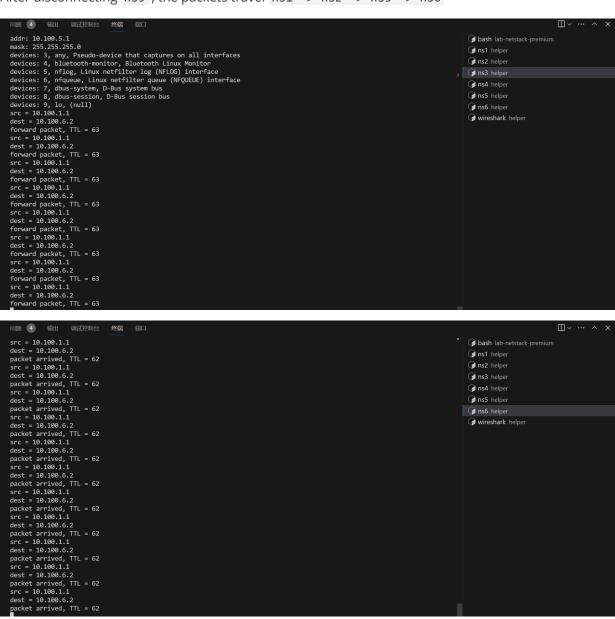
Disconnecting ns5 doesn't change distance between any pair of hosts.

For example, to measure the distance between ns1 and ns6, ns1 regularly sends packets to ns6.

In the beginning, the packets travel ns1 -> ns2 -> ns5 -> ns6.



After disconnecting ns5 , the packets travel ns1 -> ns2 -> ns3 -> ns6



Thus the distance between ns1 and ns6 is constantly 64 + 1 - TTL = 3.

I ran another test which involves change of distance:

To measure dis(1,3) , ns1 regularly sends packets to ns3.

In the beginning, the packets travel ns1 -> ns2 -> ns3, so the distance is 2.

```
devices: 4, nflog, Linux netfilter log (NFLOG) interface
devices: 5, nfqueue, Linux netfilter queue (NFQUEUE) interface
devices: 6, dbus-system, D-Bus system bus
devices: 8, 10, (null)
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 63
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 63
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 63
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 63
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 63
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 63
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 63
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 63
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 63
src = 10.100.1.1
dest = 10.100.1.2
packet arrived, TTL = 63
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 63
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 63
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 63
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 63
```

After disconnecting ns2, the packets travel ns1 -> ns4 -> ns5 -> ns6 -> ns3, so the distance is 4.

```
dest = 10.100.2.2
packet arrived, TTL = 63
src = 10.100.2.2
packet arrived, TTL = 63
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 63
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 63
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 63
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 63
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 63
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 61
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 61
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 61
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 61
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 61
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 61
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 61
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 61
src = 10.100.1.1
dest = 10.100.2.2
packet arrived, TTL = 61
```

(In my implementation, the recovery needs some iterations, so there will be wrongly forwarded packets for a short period of time after disconnection)

Checkpoint 6

During a routing procedure, only the IP mask with the longest prefix will match the current IP. While traversing on the Trie, whenever a node has a child, there's a longer prefix match.