WRITING-TASK

Part 2

2. **`sendFrame()` requires the caller to provide the destination MAC address when sending IP packets, but users of IP layer won't provide the address to you. Explain how you addressed this problem when implementing IP protocol.**

   Use standard ARP protocol. Each host has a globol map for ip and MAC address. Any time the host would like to send a packet to an IP address whose MAC address unknown, the host will send an ARP request to each host in the same subset. And device with the specific IP address will send an ARP reply to the host.

3. **Describe your routing algorithm.**

   Self-Destruct Protocol

   SDP(self-destruct protocol) is a routing protocol. Instread of RIP using UDP, the SDP is based on link layer. It is designed for the reason that the routing table stores the *MAC address* of nexthop for the task function `setRoutingTable` in our lab.

   The ***format*** of SDP frame is shown below.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Length     |      Flag     |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               |
|                                         MAC address           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Route Address Prefix                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     pLen      |.  itemFlag    |            Distance           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- **Length**: the number of SDP item. The max number is 255.
- **Flag**: basic information of the frame. The defination of each bit will be explained latter.

  ```
  [rrrrvnui]
    r (Reserved)
    i Increment   : is an increment routing information.
    u Unfinished  : have more packet because of length limited
    n isNew       : is a new device added into network
    v Verify.     : is a verify frame
  ```

- **MAC address**: MAC address of the sender.

A SDP item:

- **Route Address Prefix**: Route address prefix such as 10.100.1.0.
- **pLen:** The length of prefix such as 24.
- **Distance**: Distance to the receiver.
- **itemFlag**: Information of the item.

  ```
  [rrrrrrrd]
    r (Reserved)
    d Deleted     : delete the item
  ```

Protocol

SDP uses the algorithm of distance vector.

- Once added into network, a new host will initial its own routing table according to its devices' IP address and subnet mask. For example, ns2 has the routing table like this

```
IP Prefix        Device     Nexthop MAC        Distance
matric
10.100.1.0/24    veth2-1    (mac of veth2-1)   0 (dev)     -1
10.100.2.0/24    veth2-3    (mac of veth2-3)   0 (dev)     -1
```

Then the new host will send its routing table to all its neighbours with flag **isNew** called **Request Frame**. (We will discuss `matric` later)

- Whenever receiving an SDP frame, the host will update its own routing table according to the distance (+1 for each hop) and added it into `updateSDPItems`. Then the host send `updateSDPItems` to its neighbours *except* the src called **Update Frame**. If the SDP frame has the flag *isNew*, the host will send its own routing table to the new host (src) called **Reply Frame**.

Matric:

matric of items in routing table with `via` *will always be -1.*

- Every 30~40 seconds, matric of each item in the routing table will +1.
- Every 30~40 seconds, every host will send its whole routing table to its neighbours to verify called **Verify Frame**.
- When receiving a verify frame, matric of items in routing table, whose `nexthop` is src, will be reset to 0. It means "You are my uplink, so I believe that the routing is still effective!"
- Whenever a host is deleted from the network, the matric will be added without update. Once the matric is greater than or equal to 2, the host will send a update frame to tell neighbours delete this item with itemFlag `Deleted`. As getting an updating item with `Deleted`, the metric will be set to 8 directly.
- But, but the host won't delete the item from routing table immediately. It will be reserved for about 1-2 period(s), or a routing frame will be added again from downlink.

4. **To implement routing properly, you need to detect other hosts/let other hosts know about you. In this lab, you are not required to detect hosts not running your protocol stack automatically/let them know about you, but you must not make them complain about strange incoming packets. Describe how your IP implementation achieved this goal.**

   Considering SDP is based on link layer, SDP uses `0x2333` as EtherType. Other host or router will get a frame with an unregistered EtherType, it will ignore the frame.

Part 3

2. **Describe how you correctly handled TCP state changes.**

   Using **DFA**. We have two threads for a TCP socket. One of them is mainly used to accept the segments. The other one is used to send segments. So, considering a TCP state change, mostly, *listening thread* handles the `recv` procedure, and *main thread* handles the `send` procedure. Also, listening thread will change `criticalSt` as critical state to promise the protocol **thread safe** if the main thread have things to do.

We promise that listening thread will not be blocked any time. But, listening thread will be blocked if `criticalSt != st` which means main thread does not finish its work for **thread safe**.

3. **Describe how you implemented in-order data transferring.**

   As sender, the potocol send only one segment until it receive an ACK related.

   As receiver, the algorithm is similiar with Go-Back-N of sliding window. If we receive a packet with sequence number greater than wanted (`rcv_nxt`), the protocol will send a **duplicate ACK** to tell other side retransmit the packet with correct sequence number. If the sequence number less than `rcv_nxt`, the protocol will send an **ACK** acknowledged actually.

4. **corner cases**

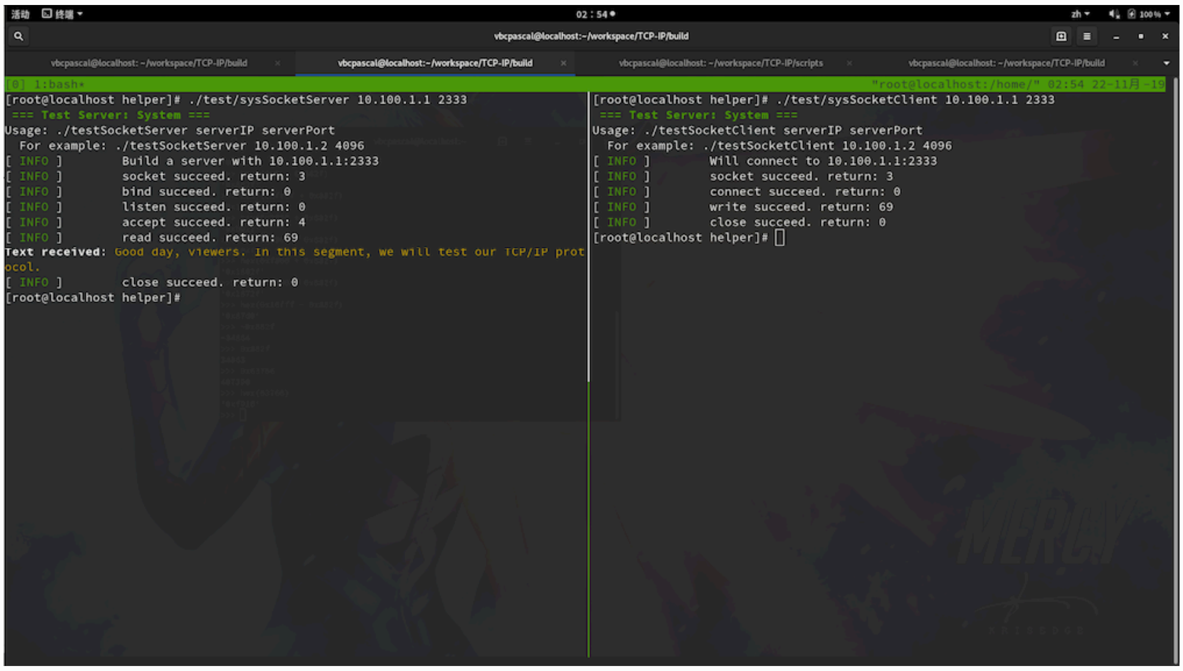
   It can work. But it does not have enough robustness actually. For example, real Linux machine will **always** send segments with `PSH` and `ACK`, even with `FIN`. This ma y lead to erroneous results.

Part 4

2. **Describe your evaluating methodology, result and usage of your evaluating method.**

   Build a connect between **client and server**, which are implemented in `tests/testSocketClient.cpp` and `tests/testSocketServer.cpp`. Each of them are be compiled into two versions, wrapped ones and system ones. In fact, it is more convenient that using `TEST_API` flag to generate them. Then, rry to connet them with each other. It can work correctly as following pictures shows.

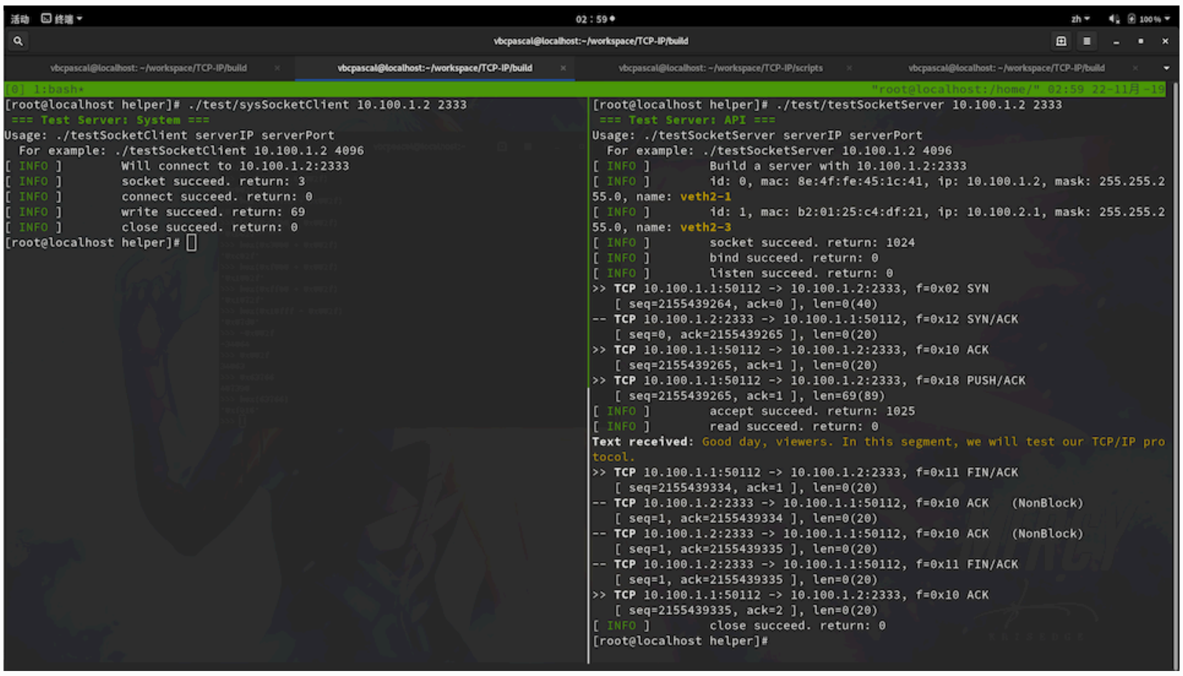
   Server: System, Client: System


活动
终端
02 : 54
vbcpascal@localhost:~/workspace/TCP-IP/build
vbcpascal@localhost: ~/workspace/TCP-IP/scripts
zh
100%
[0] 1:bash*
"root@localhost:/home/" 02:54 22-11月-19

```
[root@localhost helper]# ./test/sysSocketServer 10.100.1.1 2333
 === Test Server: System ===
Usage: ./testSocketServer serverIP serverPort
  For example: ./testSocketServer 10.100.1.2 4096
[ INFO ]        Build a server with 10.100.1.1:2333
[ INFO ]        socket succeed. return: 3
[ INFO ]        bind succeed. return: 0
[ INFO ]        listen succeed. return: 0
[ INFO ]        accept succeed. return: 4
[ INFO ]        read succeed. return: 69
Text received: Good day, viewers. In this segment, we will test our TCP/IP prot
ocol.
[ INFO ]        close succeed. return: 0
[root@localhost helper]#
```

```
[root@localhost helper]# ./test/sysSocketClient 10.100.1.1 2333
 === Test Server: System ===
Usage: ./testSocketClient serverIP serverPort
  For example: ./testSocketClient 10.100.1.2 4096
[ INFO ]        Will connect to 10.100.1.1:2333
[ INFO ]        socket succeed. return: 3
[ INFO ]        connect succeed. return: 0
[ INFO ]        write succeed. return: 69
[ INFO ]        close succeed. return: 0
[root@localhost helper]# 
```

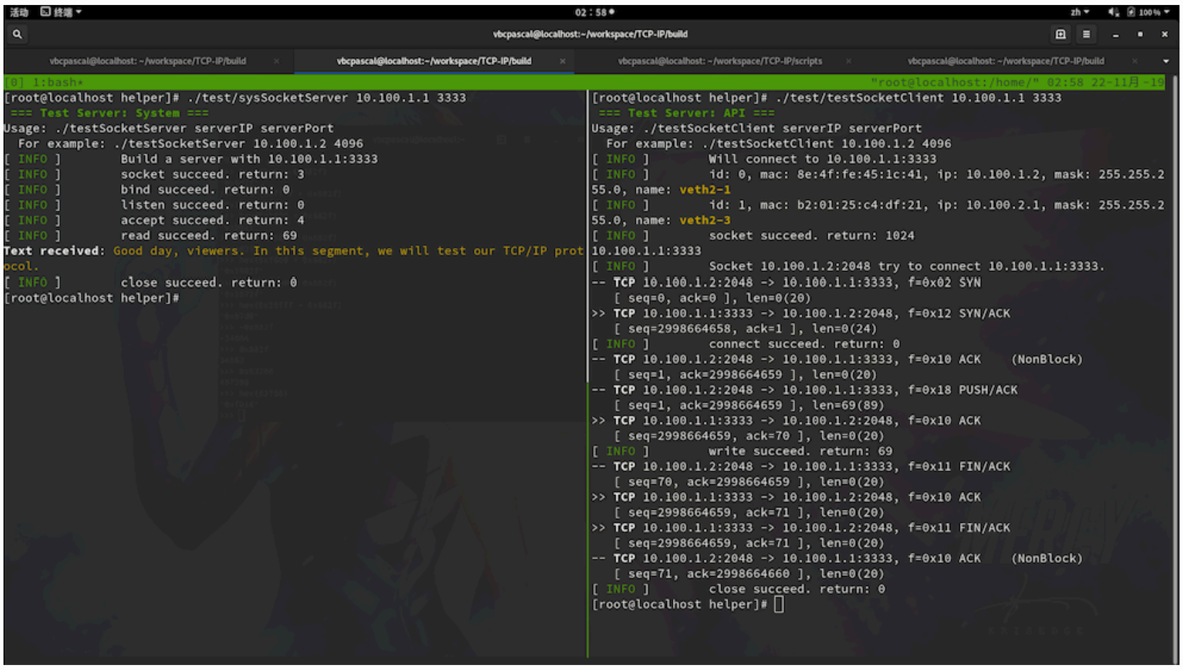
   Server: API, Client: System


vbcpascal@localhost:~/workspace/TCP-IP/build
vbcpascal@localhost: ~/workspace/TCP-IP/scripts

```
[0] 1:bash*                                                    "root@localhost:/home/" 02:59 22-11月-19
[root@localhost helper]# ./test/sysSocketClient 10.100.1.2 2333
 === Test Server: System ===
Usage: ./testSocketClient serverIP serverPort
  For example: ./testSocketClient 10.100.1.2 4096
[ INFO ]        Will connect to 10.100.1.2:2333
[ INFO ]        socket succeed. return: 3
[ INFO ]        connect succeed. return: 0
[ INFO ]        write succeed. return: 69
[ INFO ]        close succeed. return: 0
[root@localhost helper]#
```

```
[root@localhost helper]# ./test/testSocketServer 10.100.1.2 2333
 === Test Server: API ===
Usage: ./testSocketServer serverIP serverPort
  For example: ./testSocketServer 10.100.1.2 4096
[ INFO ]        Build a server with 10.100.1.2:2333
[ INFO ]        id: 0, mac: 8e:4f:fe:45:1c:41, ip: 10.100.1.2, mask: 255.255.2
55.0, name: veth2-1
[ INFO ]        id: 1, mac: b2:01:25:c4:df:21, ip: 10.100.2.1, mask: 255.255.2
55.0, name: veth2-3
[ INFO ]        socket succeed. return: 1024
[ INFO ]        bind succeed. return: 0
[ INFO ]        listen succeed. return: 0
>> TCP 10.100.1.1:50112 -> 10.100.1.2:2333, f=0x02 SYN
   [ seq=2155439264, ack=0 ], len=0(40)
-- TCP 10.100.1.2:2333 -> 10.100.1.1:50112, f=0x12 SYN/ACK
   [ seq=0, ack=2155439265 ], len=0(20)
>> TCP 10.100.1.1:50112 -> 10.100.1.2:2333, f=0x10 ACK
   [ seq=2155439265, ack=1 ], len=0(20)
>> TCP 10.100.1.1:50112 -> 10.100.1.2:2333, f=0x18 PUSH/ACK
   [ seq=2155439265, ack=1 ], len=69(89)
[ INFO ]        accept succeed. return: 1025
[ INFO ]        read succeed. return: 0
Text received: Good day, viewers. In this segment, we will test our TCP/IP pro
tocol.
>> TCP 10.100.1.1:50112 -> 10.100.1.2:2333, f=0x11 FIN/ACK
   [ seq=2155439334, ack=1 ], len=0(20)
-- TCP 10.100.1.2:2333 -> 10.100.1.1:50112, f=0x10 ACK    (NonBlock)
   [ seq=1, ack=2155439334 ], len=0(20)
-- TCP 10.100.1.2:2333 -> 10.100.1.1:50112, f=0x10 ACK    (NonBlock)
   [ seq=1, ack=2155439335 ], len=0(20)
-- TCP 10.100.1.2:2333 -> 10.100.1.1:50112, f=0x11 FIN/ACK
   [ seq=1, ack=2155439335 ], len=0(20)
>> TCP 10.100.1.1:50112 -> 10.100.1.2:2333, f=0x10 ACK
   [ seq=2155439335, ack=2 ], len=0(20)
[ INFO ]        close succeed. return: 0
[root@localhost helper]#
```

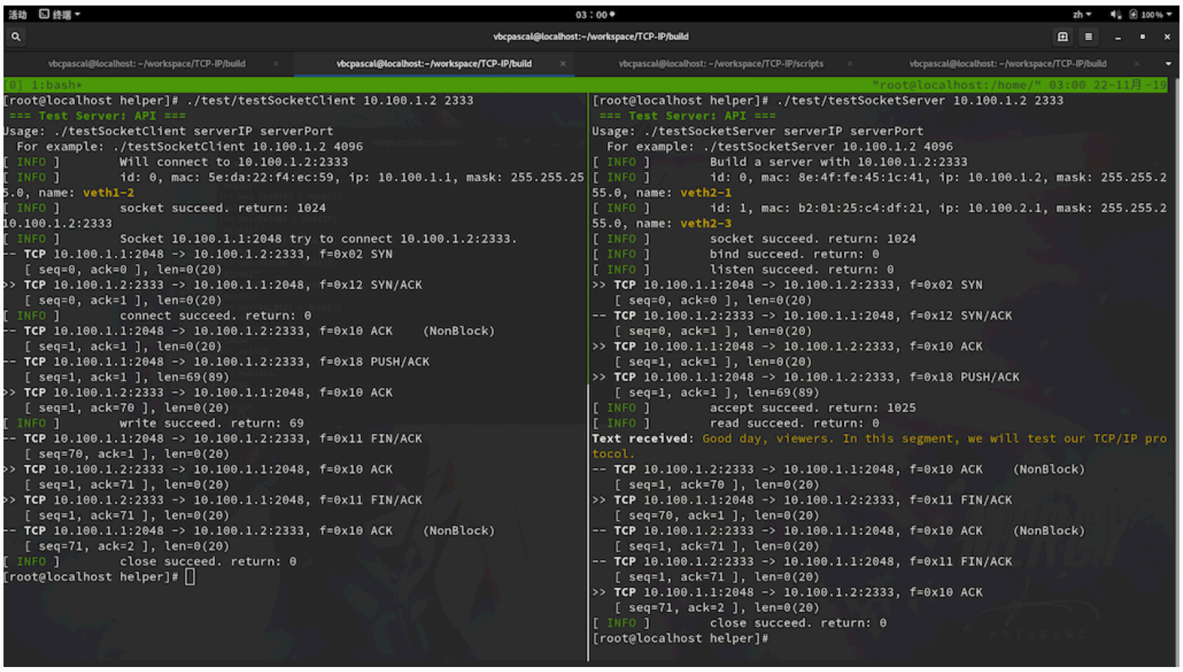
Client: System, Server: API


vbcpascal@localhost:~/workspace/TCP-IP/build
vbcpascal@localhost: ~/workspace/TCP-IP/scripts

```
[0] 1:bash*                                                    "root@localhost:/home/" 02:58 22-11月-19
[root@localhost helper]# ./test/sysSocketServer 10.100.1.1 3333
 === Test Server: System ===
Usage: ./testSocketServer serverIP serverPort
  For example: ./testSocketServer 10.100.1.2 4096
[ INFO ]        Build a server with 10.100.1.1:3333
[ INFO ]        socket succeed. return: 3
[ INFO ]        bind succeed. return: 0
[ INFO ]        listen succeed. return: 0
[ INFO ]        accept succeed. return: 4
[ INFO ]        read succeed. return: 69
Text received: Good day, viewers. In this segment, we will test our TCP/IP prot
ocol.
[ INFO ]        close succeed. return: 0
[root@localhost helper]#
```

```
[root@localhost helper]# ./test/testSocketClient 10.100.1.1 3333
 === Test Server: API ===
Usage: ./testSocketClient serverIP serverPort
  For example: ./testSocketClient 10.100.1.2 4096
[ INFO ]        Will connect to 10.100.1.1:3333
[ INFO ]        id: 0, mac: 8e:4f:fe:45:1c:41, ip: 10.100.1.2, mask: 255.255.2
55.0, name: veth2-1
[ INFO ]        id: 1, mac: b2:01:25:c4:df:21, ip: 10.100.2.1, mask: 255.255.2
55.0, name: veth2-3
[ INFO ]        socket succeed. return: 1024
10.100.1.1:3333
[ INFO ]        Socket 10.100.1.2:2048 try to connect 10.100.1.1:3333.
-- TCP 10.100.1.2:2048 -> 10.100.1.1:3333, f=0x02 SYN
   [ seq=0, ack=0 ], len=0(20)
>> TCP 10.100.1.1:3333 -> 10.100.1.2:2048, f=0x12 SYN/ACK
   [ seq=2998664658, ack=1 ], len=0(24)
[ INFO ]        connect succeed. return: 0
-- TCP 10.100.1.2:2048 -> 10.100.1.1:3333, f=0x10 ACK    (NonBlock)
   [ seq=1, ack=2998664659 ], len=0(20)
-- TCP 10.100.1.2:2048 -> 10.100.1.1:3333, f=0x18 PUSH/ACK
   [ seq=1, ack=2998664659 ], len=69(89)
>> TCP 10.100.1.1:3333 -> 10.100.1.2:2048, f=0x10 ACK
   [ seq=2998664659, ack=70 ], len=0(20)
[ INFO ]        write succeed. return: 69
-- TCP 10.100.1.2:2048 -> 10.100.1.1:3333, f=0x11 FIN/ACK
   [ seq=70, ack=2998664659 ], len=0(20)
>> TCP 10.100.1.1:3333 -> 10.100.1.2:2048, f=0x10 ACK
   [ seq=2998664659, ack=71 ], len=0(20)
>> TCP 10.100.1.1:3333 -> 10.100.1.2:2048, f=0x11 FIN/ACK
   [ seq=2998664659, ack=71 ], len=0(20)
-- TCP 10.100.1.2:2048 -> 10.100.1.1:3333, f=0x10 ACK    (NonBlock)
   [ seq=71, ack=2998664660 ], len=0(20)
[ INFO ]        close succeed. return: 0
[root@localhost helper]#
```

Client: API, Server: API


活动
终端
03 : 00
zh
100 %
vbcpascal@localhost:~/workspace/TCP-IP/build
vbcpascal@localhost: ~/workspace/TCP-IP/build
vbcpascal@localhost:~/workspace/TCP-IP/build
vbcpascal@localhost: ~/workspace/TCP-IP/scripts
vbcpascal@localhost: ~/workspace/TCP-IP/build
[0] 1:bash*
"root@localhost:/home/" 03:00 22-11月-19

```
[root@localhost helper]# ./test/testSocketClient 10.100.1.2 2333
 === Test Server: API ===
Usage: ./testSocketClient serverIP serverPort
  For example: ./testSocketClient 10.100.1.2 4096
[ INFO ]        Will connect to 10.100.1.2:2333
[ INFO ]        id: 0, mac: 5e:da:22:f4:ec:59, ip: 10.100.1.1, mask: 255.255.25
5.0, name: veth1-2
[ INFO ]        socket succeed. return: 1024
10.100.1.2:2333
[ INFO ]        Socket 10.100.1.1:2048 try to connect 10.100.1.2:2333.
-- TCP 10.100.1.1:2048 -> 10.100.1.2:2333, f=0x02 SYN
   [ seq=0, ack=0 ], len=0(20)
>> TCP 10.100.1.2:2333 -> 10.100.1.1:2048, f=0x12 SYN/ACK
   [ seq=0, ack=1 ], len=0(20)
[ INFO ]        connect succeed. return: 0
-- TCP 10.100.1.1:2048 -> 10.100.1.2:2333, f=0x10 ACK    (NonBlock)
   [ seq=1, ack=1 ], len=0(20)
-- TCP 10.100.1.1:2048 -> 10.100.1.2:2333, f=0x18 PUSH/ACK
   [ seq=1, ack=1 ], len=69(89)
>> TCP 10.100.1.2:2333 -> 10.100.1.1:2048, f=0x10 ACK
   [ seq=1, ack=70 ], len=0(20)
[ INFO ]        write succeed. return: 69
-- TCP 10.100.1.1:2048 -> 10.100.1.2:2333, f=0x11 FIN/ACK
   [ seq=70, ack=1 ], len=0(20)
>> TCP 10.100.1.2:2333 -> 10.100.1.1:2048, f=0x10 ACK
   [ seq=1, ack=71 ], len=0(20)
>> TCP 10.100.1.2:2333 -> 10.100.1.1:2048, f=0x11 FIN/ACK
   [ seq=1, ack=71 ], len=0(20)
-- TCP 10.100.1.1:2048 -> 10.100.1.2:2333, f=0x10 ACK    (NonBlock)
   [ seq=71, ack=2 ], len=0(20)
[ INFO ]        close succeed. return: 0
[root@localhost helper]# 
```

```
[root@localhost helper]# ./test/testSocketServer 10.100.1.2 2333
 === Test Server: API ===
Usage: ./testSocketServer serverIP serverPort
  For example: ./testSocketServer 10.100.1.2 4096
[ INFO ]        Build a server with 10.100.1.2:2333
[ INFO ]        id: 0, mac: 8e:4f:fe:45:1c:41, ip: 10.100.1.2, mask: 255.255.2
55.0, name: veth2-1
[ INFO ]        id: 1, mac: b2:01:25:c4:df:21, ip: 10.100.2.1, mask: 255.255.2
55.0, name: veth2-3
[ INFO ]        socket succeed. return: 1024
[ INFO ]        bind succeed. return: 0
[ INFO ]        listen succeed. return: 0
>> TCP 10.100.1.1:2048 -> 10.100.1.2:2333, f=0x02 SYN
   [ seq=0, ack=0 ], len=0(20)
-- TCP 10.100.1.2:2333 -> 10.100.1.1:2048, f=0x12 SYN/ACK
   [ seq=0, ack=1 ], len=0(20)
>> TCP 10.100.1.1:2048 -> 10.100.1.2:2333, f=0x10 ACK
   [ seq=1, ack=1 ], len=0(20)
>> TCP 10.100.1.1:2048 -> 10.100.1.2:2333, f=0x18 PUSH/ACK
   [ seq=1, ack=1 ], len=69(89)
[ INFO ]        accept succeed. return: 1025
[ INFO ]        read succeed. return: 0
Text received: Good day, viewers. In this segment, we will test our TCP/IP pro
tocol.
-- TCP 10.100.1.2:2333 -> 10.100.1.1:2048, f=0x10 ACK    (NonBlock)
   [ seq=1, ack=70 ], len=0(20)
>> TCP 10.100.1.1:2048 -> 10.100.1.2:2333, f=0x11 FIN/ACK
   [ seq=70, ack=1 ], len=0(20)
-- TCP 10.100.1.2:2333 -> 10.100.1.1:2048, f=0x10 ACK    (NonBlock)
   [ seq=1, ack=71 ], len=0(20)
-- TCP 10.100.1.2:2333 -> 10.100.1.1:2048, f=0x11 FIN/ACK
   [ seq=1, ack=71 ], len=0(20)
>> TCP 10.100.1.1:2048 -> 10.100.1.2:2333, f=0x10 ACK
   [ seq=71, ack=2 ], len=0(20)
[ INFO ]        close succeed. return: 0
[root@localhost helper]#
```