Lab 2c报告

王颖 1900013016

Transport-layer: TCP Protocol

> 1. Use the interfaces provided by ip.h to implement the following POSIX-compatible interfaces. You should follow RFC793 when working on this. You are also expected to keep compatibility with POSIX.1-2017 standard when implementing your socket interfaces, but that's just for your applications to run correctly, you will NOT lose credits if your interfaces behave slightly different. Note: You can use a file descriptor allocating algorithm slightly different from the standardized one to avoid conflicts with fds allocated by the system.

- See `incl/tcp.h`, `src/tcp.cpp` , `incl/socket.h`, `scr/socket.cpp`, `incl/timer.h`, and `src/timer.cpp`
- The following functions are implemented:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

#ifndef _TCP_H_
#define _TCP_H_

#define MAX_PORT_NUMBER (1 << 16)

struct Info_t{
    uint32_t srcIP;
    uint32_t dstIP;
    uint16_t srcPort;
    uint16_t dstPort;
};

struct tcpHdr_t{
    uint16_t srcPort;
    uint16_t dstPort;
    uint32_t seq;
    uint32_t ack;
    uint16_t flags;
    uint16_t window;
    uint32_t others;
};

#define FLAG_SYN (1 << 9)
#define FLAG_ACK (1 << 12)
#define FLAG_FIN (1 << 8)

bool isSYN(uint16_t flag);
bool isACK(uint16_t flag);
bool isFIN(uint16_t flag);
uint16_t setSYN(uint16_t flag);
uint16_t setACK(uint16_t flag);
```

```
uint16_t setFIN(uint16_t flag);

/**
* @brief Send an TCP packet to specified host.
*
* @param buf pointer to payload
* @param len Length of payload
* @param info Information about Source IP address, Source Port, Destination IP
address, Destination Port.
* @param seq SEQ number
* @param ack ACK number
* @param flags Value of flags in header
* @return 0 on success, -1 on error.
*/
int sendTCPPacket(const void* buf, int len, Info_t* info, uint32_t seq, uint32_t
ack, int16_t flags);

int recvTCPPacket(const void* buf, int len, Info_t info);

#endif
```

```
/**
* @file socket.h
* @brief POSIX-compatible socket library supporting TCP protocol on IPv4.
*/
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include "tcp.h"
#include <bits/stdc++.h>
using namespace std;

#ifndef _SOCKET_H_
#define _SOCKET_H_

#define ROLE_CLIENT     1
#define ROLE_SEVER      0

#define STATE_CLOSED        0
#define STATE_LISTEN        1
#define STATE_SYNRCVD       2
#define STATE_SYNSENT       3
#define STATE_ESTABLISHED   4
#define STATE_FINWAIT1      5
#define STATE_FINWAIT2      6
#define STATE_TIMEWAIT      7
#define STATE_CLOSING       8
#define STATE_CLOSEWAIT     9
#define STATE_LASTACK       10

#define SOCK_TYPE_STREAM    1

#define SOCK_BUFFER_SIZE (1 << 18)

struct tryConn_t {
    int synReceive;
    int ackReceive;
    int connState;
    uint32_t srcIP;
    uint16_t srcPort;
    uint32_t dstIP;
    uint16_t dstPort;
    tryConn_t () {
        synReceive = false;
        ackReceive = false;
        connState = STATE_LISTEN;
    }
};


struct socket_t {
    unsigned char* buffer;  // for receiving
    int readp;              //next read position
    int szep;               //pos of the last received byte

    int writep;             //next write position

    int domain;
    int type;
    int protocol;

    Info_t info; //port是主机序的

    int index;              //corresponding fd number
    int role; // CLIENT or SEVER
    int state;
    int backlog;

    bool connectReady;
    //std :: mutex connect_mutex;
    //std :: condition_variable connect_cv;

    set <tryConn_t*> connTrying_set;

    socket_t() {
        buffer = new unsigned char [SOCK_BUFFER_SIZE];
        readp = writep = szep = 0;
        domain = type = protocol = 0;
        index = -1, role = state = -1, backlog = 1;
        connectReady = false;
        connTrying_set.clear();
    }
    socket_t(const socket_t* x) {
        buffer = new unsigned char [SOCK_BUFFER_SIZE]; memcpy(buffer, x ->
buffer, SOCK_BUFFER_SIZE);
        readp = x -> readp, writep = x -> writep, szep = x -> szep;
        domain = x -> domain, type = x -> type, protocol = x -> protocol;
        info = x -> info;
        index = x -> index, role = x -> role, state = x -> state, backlog = x ->
state;
        connectReady = x -> connectReady;
        connTrying_set.clear();
    }
    ~socket_t() {
    }
};

map <int, socket_t*> *getSocketPool();


/**
* @see [POSIX.1-2017:socket]
(http://pubs.opengroup.org/onlinepubs/9699919799/functions/socket.html )
*/
extern "C" {
    int __wrap_socket(int domain, int type, int protocol);
}
int __my_socket(int domain, int type, int protocol);

/**
* @see [POSIX.1-2017:bind]
(http://pubs.opengroup.org/onlinepubs/9699919799/functions/bind.html)
*/
extern "C" {
    int __wrap_bind(int socket, const struct sockaddr* address, socklen_t
address_len);
}
int __my_bind(int socket, const struct sockaddr* address, socklen_t
address_len);

/**
* @see [POSIX.1-2017:listen]
(http://pubs.opengroup.org/onlinepubs/9699919799/functions/listen.html)
*/
extern "C" {
    int __wrap_listen(int socket, int backlog);
}
int __my_listen(int socket, int backlog);

/**
* @see [POSIX.1-2017:connect]
(http://pubs.opengroup.org/onlinepubs/9699919799/functions/connect.html)
*/
extern "C" {
    int __wrap_connect(int socket, const struct sockaddr* address, socklen_t
address_len);
}
int __my_connect(int socket, const struct sockaddr* address, socklen_t
address_len);


/**
* @see [POSIX.1-2017:accept]
(http://pubs.opengroup.org/onlinepubs/9699919799/functions/accept.html)
*/
extern "C" {
    int __wrap_accept(int socket, struct sockaddr* address , socklen_t *
address_len );
}
int __my_accept(int socket, struct sockaddr* address , socklen_t * address_len
);
```

```
/**
* @see [POSIX.1-2017:read]
(http://pubs.opengroup.org/onlinepubs/9699919799/functions/read.html)
*/
extern "C"{
    ssize_t __wrap_read(int fildes, void* buf, size_t nbyte);
}
ssize_t __my_read(int fildes, void* buf, size_t nbyte);

/**
* @see [POSIX.1-2017:write]
(http://pubs.opengroup.org/onlinepubs/9699919799/functions/write.html)
*/
extern "C"{
ssize_t __wrap_write(int fildes, const void* buf, size_t nbyte);
}
ssize_t __my_write(int fildes, const void* buf, size_t nbyte);

/**
* @see [POSIX.1-2017:close]
(http://pubs.opengroup.org/onlinepubs/9699919799/functions/close.html)
*/
extern "C"{
int __wrap_close(int fildes);
}
int __my_close(int fildes);

/**
* @see [POSIX.1-2017:getaddrinfo]
(http://pubs.opengroup.org/onlinepubs/9699919799/functions/getaddrinfo.html)
*/
extern "C"{
int __wrap_getaddrinfo(const char* node, const char* service, const struct
addrinfo* hints, struct addrinfo** res);
}
int __my_getaddrinfo(const char* node, const char* service, const struct
addrinfo* hints, struct addrinfo** res);

#endif
```

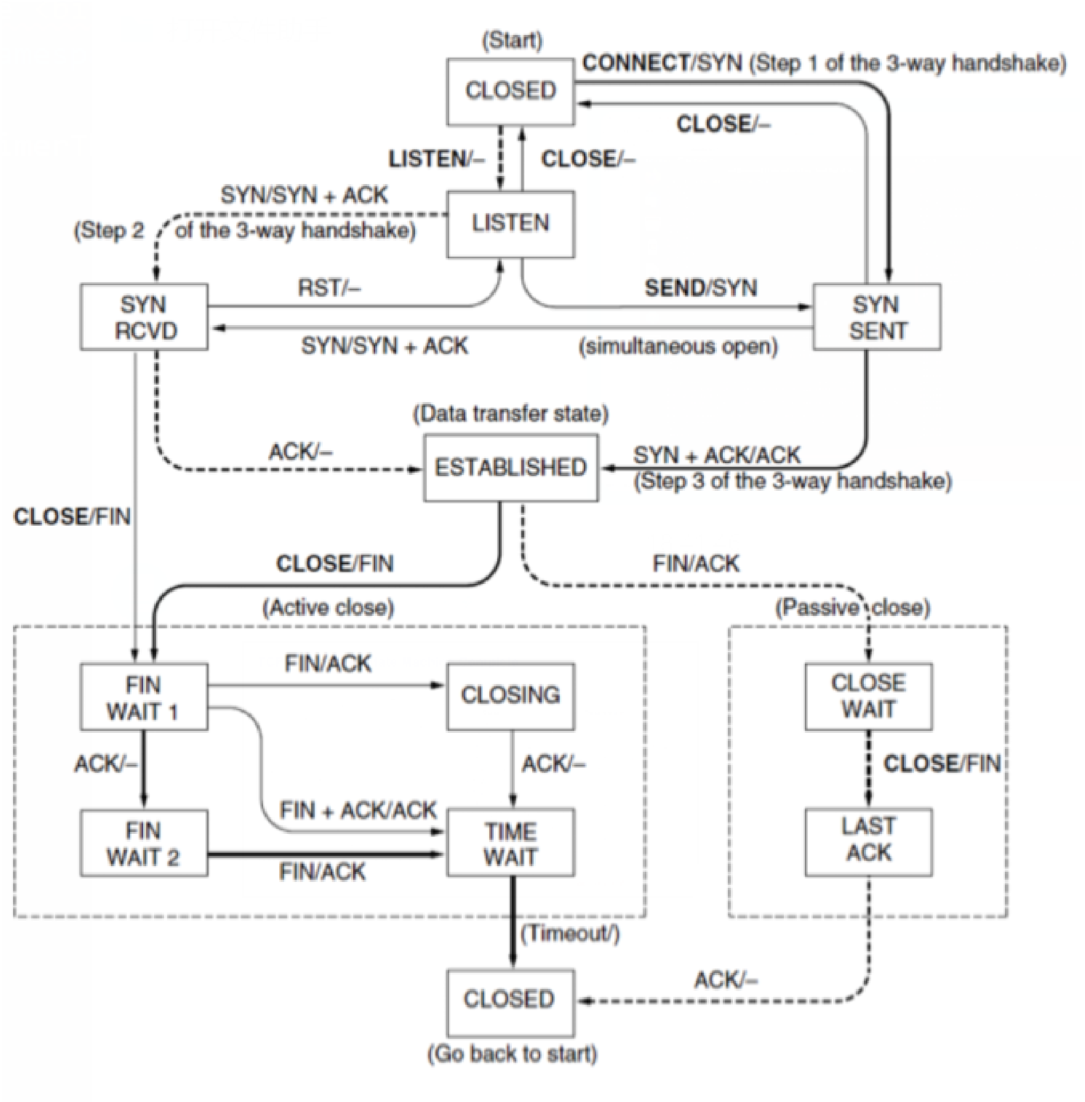
```
#include <bits/stdc++.h>
using namespace std;

void *TimerThread(void *vargp);
```

- Functions in `tcp.cpp` use interface provided by IP layer to send packets in TCP layer.
- Functions in `socket.cpp` rovide interfaces to the application layer. Specially,
    - connect(), write(), read() and close() are all trapped. Which means that
        - connect() will not return until the connection is established
        - write() will not return until the message is filled into opposite buffer
        - read() will not return until it read at least one byte or the opposite side close the connection
        - close() will not return until four waves and a wait-time finish

- TimerThread() in timer.cpp is used to wake up the suspended thread so that packets can be resent.

> 2. Describe how you correctly handled TCP state changes

- In `tcp.cpp/recvTCPPacket()`, every time the host receive a tcp packet. It will find the corresponding socket, check the socket status and SEQ, ACK, flags in the TCP header of the packet.
- If SYN, ACK, FIN flag is seen in TCP header, I might change the socket status according to the Schematic diagram.



- The difference of my implement and this diagram is that, in my implement, there will be no a packet with both FIN and ACK flags, so `FIN WAIT1` will not change to `TIME_WAIT` immediately.
- Most state changes are driven by TCP packets receiving. In particular,
  - the state transition from `ESTABLISHED` to `FIN WAIT1`, and the state change from `CLOSE WAIT` to `LAST ACK` occur in `socket.cpp/close()`.
  - the change from `CLOSED` to `LISTEN` happens in `socket.cppp/listen()`
  - the change from `CLOSED` to `SYN SENT` happens in `socket.cppp/connect()`
  - I did not implement RST flag and its associated state transitions

> 3. Use tcpdump/wireshark to capture the TCP packets generated by your implementation. Hexdump the content of any one packet here, and show meanings for each byte in the TCP header.

- The following IP header package was captured using tcpdump.
- It is the SYN/ACK that the server returns after the current machine sends a SYN via port `26437` to the server `10.100.3.2:10086`.

```
19:36:39.455428 IP 10.100.3.2.10086 > none44353-virtual-machine.26437: Flags
[S.], seq 0:1, ack 1, win 4096, length 1
    0x0000:  4500 0029 0000 0000 1e06 0000 0a64 0302  E..).........d..
    0x0010:  0a64 0101 2766 6745 0000 0000 0000 0001  .d..'fgE........
    0x0020:  5012 1000 0030 0000 0000 0000 00         P....0.......
```

- `4500 0029 0000 0000 1e06 0000 0a64 0302 0a64 0101` is IP header
- `2766 6745 0000 0000 0000 0001 5012 1000 0000 0000` is TCP header
  - `2766` is srcPort whose decimal value is 10086
  - `6745` is dstPort whose decimal value is 26437
  - `0000 0000` corresponding to SEQ number 0
  - `0000 0001` corresponding to ACK number 1
  - `5` indicates that the tcp header length is 5 * 32 bits = 5 * 4 bytes
  - `012` is reserved position and flags. Here, the ACK and SYN bits are 1.
  - `1000` is window size. This limits the TCP packets to 4096 bytes.
  - `0030` and `0000` are checksum and urgent pointer.
    - Checksum is set using xor sum.
    - I don't set urgent pointer.
- the following one byte is tcp payload `00`
- the last 4 bytes are IP checksum, I don't set them, so they are `00 0000 00`

I use cp4.txt to build the network ns1 —— ns2 —— ns3 —— ns4.

> 4. Show your implementaion provides reliable delivery (i.e., it can detect packet loss and retransmit the lost packets). You are encouraged to attach a screenshot of the wireshark packet trace here. Check section 4.2 to see how to emulate a lossy link.

- See https://wiki.linuxfoundation.org/networking/netem to build a lossy network with delay using netem.

ATTENTION: since I don't implement an adaptive resending mechanism, you need to modify the double value `rGAP` in `socket.cpp` to depending onthe RTT of the network you build. The default value of `rGAP` is 0.07

- In ns1's terminal, the following commands are executed to build a lossy link

```
tc qdisc add dev veth1-2 root netem
tc qdisc change dev veth1-2 root netem loss 5%
```

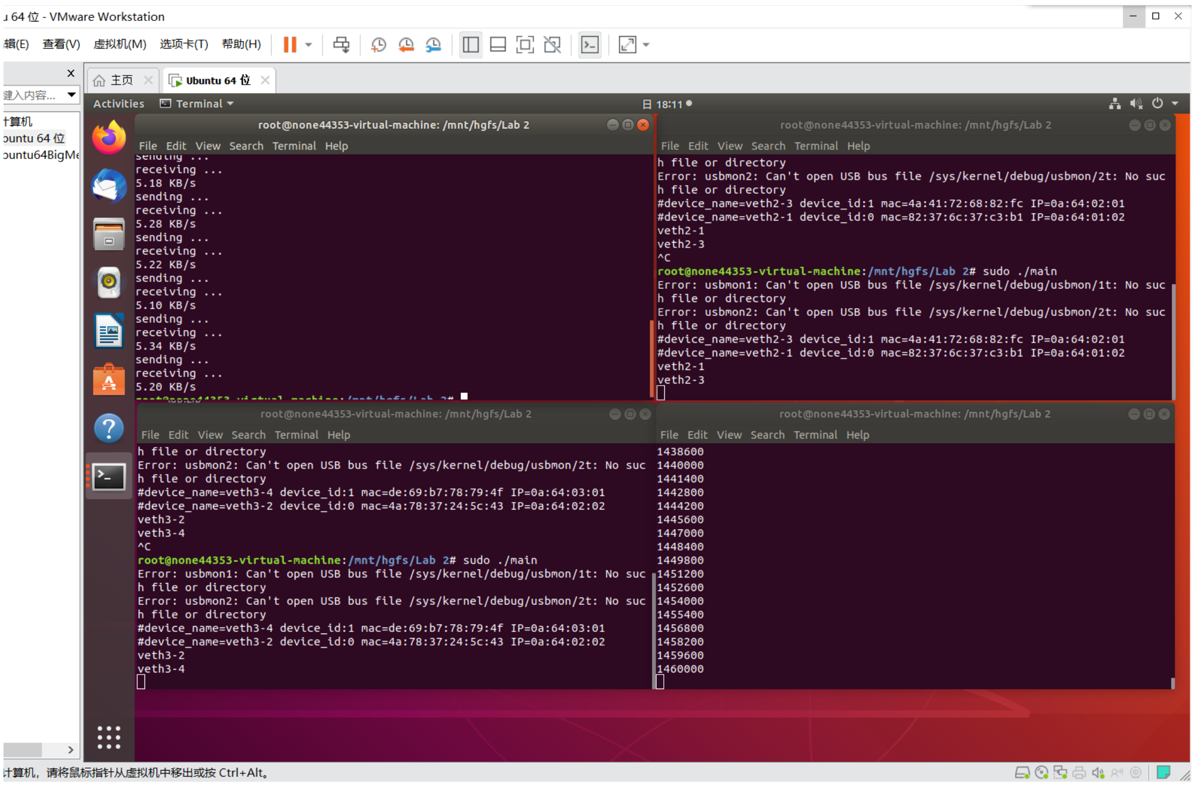
- In ns3's terminal, the following commands are executed to build link with delay
  - ATTENTION: it's a one-way delay

```
tc qdisc add dev veth3-4 root netem
tc qdisc change dev veth3-4 root netem delay 200ms
```

- In this case, I run perf-test.

```
- ns1 terminal
    sudo ./perfclient 10.100.3.2
- ns2 terminal
    sudo ./main
- ns3 terminal
    sudo ./main
- ns4 terminal
    sudo ./perfserver
```

- Below is the result.


64 位 - VMware Workstation
辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
主页
Ubuntu 64 位
Activities
Terminal
18:11
计算机，请将鼠标指针从虚拟机中移出或按 Ctrl+Alt。

```
root@none44353-virtual-machine: /mnt/hgfs/Lab 2
receiving ...
5.18 KB/s
sending ...
receiving ...
5.28 KB/s
sending ...
receiving ...
5.22 KB/s
sending ...
receiving ...
5.10 KB/s
sending ...
receiving ...
5.34 KB/s
sending ...
receiving ...
5.20 KB/s
```

```
root@none44353-virtual-machine: /mnt/hgfs/Lab 2
h file or directory
Error: usbmon2: Can't open USB bus file /sys/kernel/debug/usbmon/2t: No suc
h file or directory
#device_name=veth2-3 device_id:1 mac=4a:41:72:68:82:fc IP=0a:64:02:01
#device_name=veth2-1 device_id:0 mac=82:37:6c:37:c3:b1 IP=0a:64:01:02
veth2-1
veth2-3
^C
root@none44353-virtual-machine:/mnt/hgfs/Lab 2# sudo ./main
Error: usbmon1: Can't open USB bus file /sys/kernel/debug/usbmon/1t: No suc
h file or directory
Error: usbmon2: Can't open USB bus file /sys/kernel/debug/usbmon/2t: No suc
h file or directory
#device_name=veth2-3 device_id:1 mac=4a:41:72:68:82:fc IP=0a:64:02:01
#device_name=veth2-1 device_id:0 mac=82:37:6c:37:c3:b1 IP=0a:64:01:02
veth2-1
veth2-3
```

```
root@none44353-virtual-machine: /mnt/hgfs/Lab 2
h file or directory
Error: usbmon2: Can't open USB bus file /sys/kernel/debug/usbmon/2t: No suc
h file or directory
#device_name=veth3-4 device_id:1 mac=de:69:b7:78:79:4f IP=0a:64:03:01
#device_name=veth3-2 device_id:0 mac=4a:78:37:24:5c:43 IP=0a:64:02:02
veth3-2
veth3-4
^C
root@none44353-virtual-machine:/mnt/hgfs/Lab 2# sudo ./main
Error: usbmon1: Can't open USB bus file /sys/kernel/debug/usbmon/1t: No suc
h file or directory
Error: usbmon2: Can't open USB bus file /sys/kernel/debug/usbmon/2t: No suc
h file or directory
#device_name=veth3-4 device_id:1 mac=de:69:b7:78:79:4f IP=0a:64:03:01
#device_name=veth3-2 device_id:0 mac=4a:78:37:24:5c:43 IP=0a:64:02:02
veth3-2
veth3-4
```

```
root@none44353-virtual-machine: /mnt/hgfs/Lab 2
1438600
1440000
1441400
1442800
1444200
1445600
1447000
1448400
1449800
1451200
1452600
1454000
1455400
1456800
1458200
1459600
1460000
```

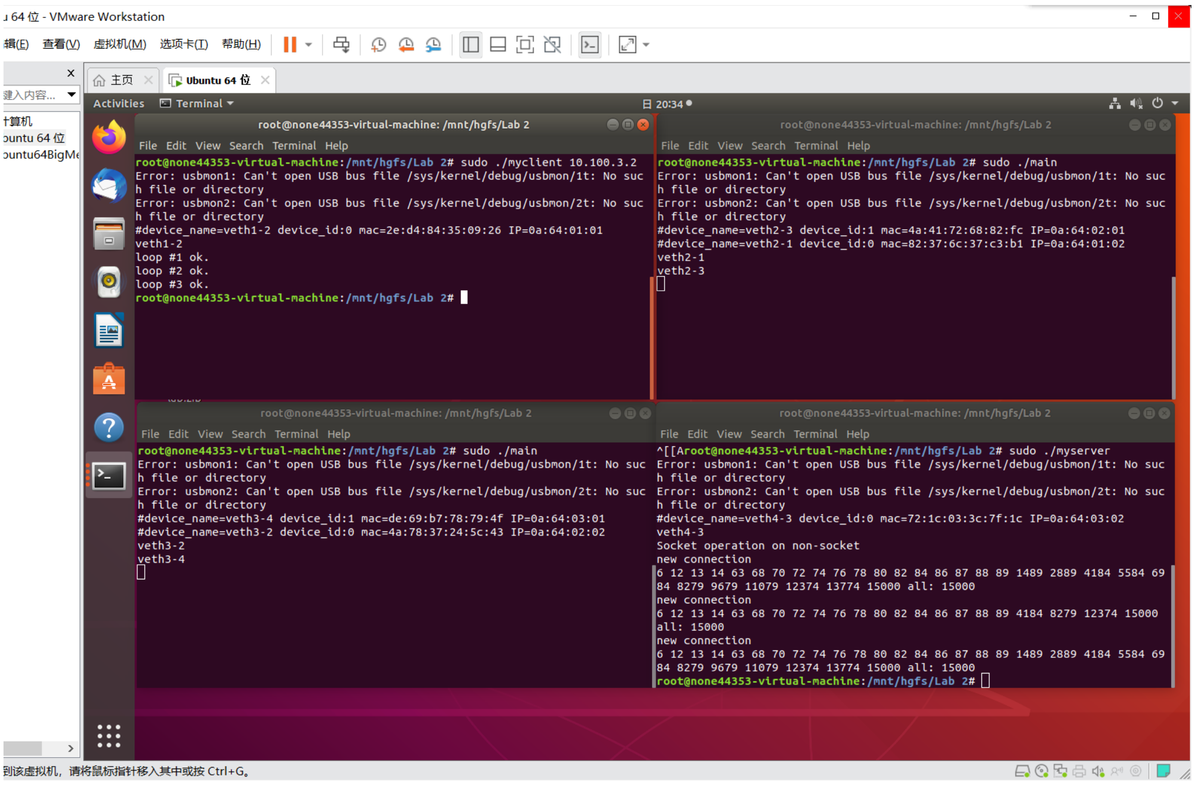
> 5. Create a virtual network with the following topology and run echo_server at ns4 and echo_client at ns1. The source code is under the folder called "checkpoints". Paste the output of them here. Note that you are not allowed to make changes to the source code (i.e., the *.h and *.c files). Check out section 4 to see how to hijack the library functions such as listen().

- After building the network in cp4.txt, I run the following commands

```
- ns1 terminal
    sudo ./myclient 10.100.3.2
- ns2 terminal
    sudo ./main
- ns3 terminal
    sudo ./main
- ns4 terminal
    sudo ./myclient
```

(See makefile, echo_client.cpp and echo_server.cpp are compiled into ELF myclient and myserver.)

Without loss and delay, the result is shown below:



```
root@none44353-virtual-machine:/mnt/hgfs/Lab 2# sudo ./myclient 10.100.3.2
Error: usbmon1: Can't open USB bus file /sys/kernel/debug/usbmon/1t: No suc
h file or directory
Error: usbmon2: Can't open USB bus file /sys/kernel/debug/usbmon/2t: No suc
h file or directory
#device_name=veth1-2 device_id:0 mac=2e:d4:84:35:09:26 IP=0a:64:01:01
veth1-2
loop #1 ok.
loop #2 ok.
loop #3 ok.
root@none44353-virtual-machine:/mnt/hgfs/Lab 2#
```

```
root@none44353-virtual-machine:/mnt/hgfs/Lab 2# sudo ./main
Error: usbmon1: Can't open USB bus file /sys/kernel/debug/usbmon/1t: No suc
h file or directory
Error: usbmon2: Can't open USB bus file /sys/kernel/debug/usbmon/2t: No suc
h file or directory
#device_name=veth2-3 device_id:1 mac=4a:41:72:68:82:fc IP=0a:64:02:01
#device_name=veth2-1 device_id:0 mac=82:37:6c:37:c3:b1 IP=0a:64:01:02
veth2-1
veth2-3
```

```
root@none44353-virtual-machine:/mnt/hgfs/Lab 2# sudo ./main
Error: usbmon1: Can't open USB bus file /sys/kernel/debug/usbmon/1t: No suc
h file or directory
Error: usbmon2: Can't open USB bus file /sys/kernel/debug/usbmon/2t: No suc
h file or directory
#device_name=veth3-4 device_id:1 mac=de:69:b7:78:79:4f IP=0a:64:03:01
#device_name=veth3-2 device_id:0 mac=4a:78:37:24:5c:43 IP=0a:64:02:02
veth3-2
veth3-4
```

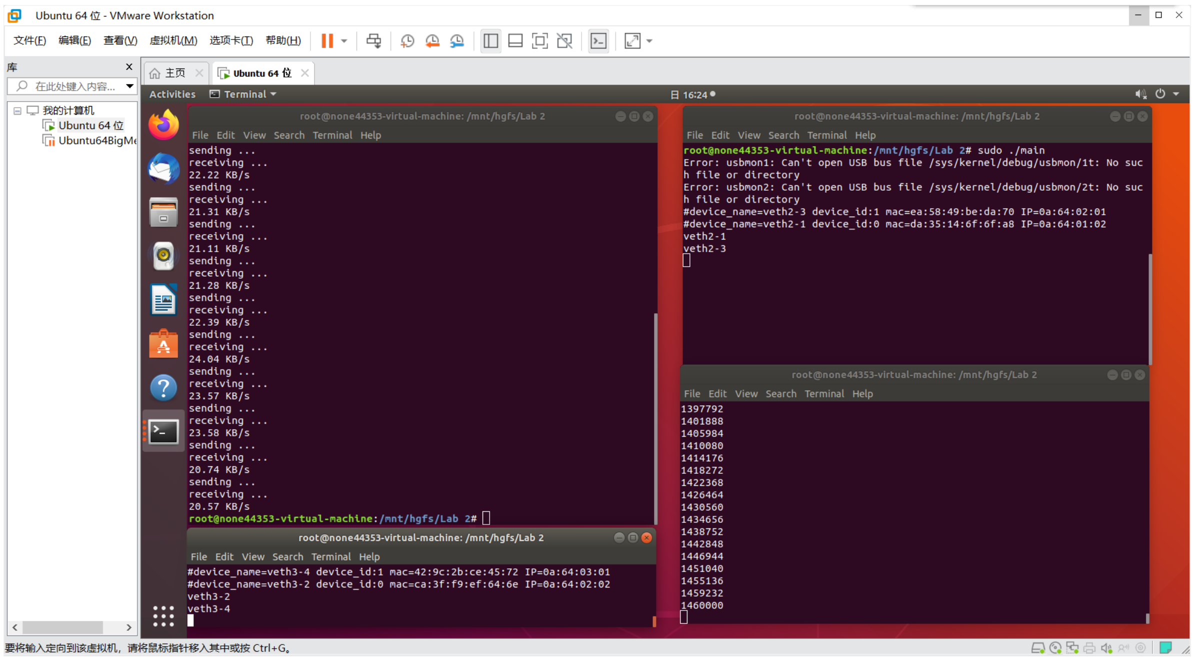
```
^[[Aroot@none44353-virtual-machine:/mnt/hgfs/Lab 2# sudo ./myserver
Error: usbmon1: Can't open USB bus file /sys/kernel/debug/usbmon/1t: No suc
h file or directory
Error: usbmon2: Can't open USB bus file /sys/kernel/debug/usbmon/2t: No suc
h file or directory
#device_name=veth4-3 device_id:0 mac=72:1c:03:3c:7f:1c IP=0a:64:03:02
veth4-3
Socket operation on non-socket
new connection
6 12 13 14 63 68 70 72 74 76 78 80 82 84 86 87 88 89 1489 2889 4184 5584 69
84 8279 9679 11079 12374 13774 15000 all: 15000
new connection
6 12 13 14 63 68 70 72 74 76 78 80 82 84 86 87 88 89 4184 8279 12374 15000
all: 15000
new connection
6 12 13 14 63 68 70 72 74 76 78 80 82 84 86 87 88 89 1489 2889 4184 5584 69
84 8279 9679 11079 12374 13774 15000 all: 15000
root@none44353-virtual-machine:/mnt/hgfs/Lab 2#
```

6. Create a virtual network with the following topology and run perf_server at ns4 and perf_client at ns1. Paste the output of them here. Again, you are not allowed to make changes to the source code

- After building the network in cp4.txt, I run the following commands

```
- ns1 terminal
    sudo ./perfclient 10.100.3.2
- ns2 terminal
    sudo ./main
- ns3 terminal
    sudo ./main
- ns4 terminal
    sudo ./perfclient
```

Without loss and delay, the result is shown below:



```
sending ...
receiving ...
22.22 KB/s
sending ...
receiving ...
21.31 KB/s
sending ...
receiving ...
21.11 KB/s
sending ...
receiving ...
21.28 KB/s
sending ...
receiving ...
22.39 KB/s
sending ...
receiving ...
24.04 KB/s
sending ...
receiving ...
23.57 KB/s
sending ...
receiving ...
23.58 KB/s
sending ...
receiving ...
20.74 KB/s
sending ...
receiving ...
20.57 KB/s
root@none44353-virtual-machine:/mnt/hgfs/Lab 2#
```

```
#device_name=veth3-4 device_id:1 mac=42:9c:2b:ce:45:72 IP=0a:64:03:01
#device_name=veth3-2 device_id:0 mac=ca:3f:f9:ef:64:6e IP=0a:64:02:02
veth3-2
veth3-4
```

```
root@none44353-virtual-machine:/mnt/hgfs/Lab 2# sudo ./main
Error: usbmon1: Can't open USB bus file /sys/kernel/debug/usbmon/1t: No suc
h file or directory
Error: usbmon2: Can't open USB bus file /sys/kernel/debug/usbmon/2t: No suc
h file or directory
#device_name=veth2-3 device_id:1 mac=ea:58:49:be:da:70 IP=0a:64:02:01
#device_name=veth2-1 device_id:0 mac=da:35:14:6f:6f:a8 IP=0a:64:01:02
veth2-1
veth2-3
```

```
1397792
1401888
1405984
1410080
1414176
1418272
1422368
1426464
1430560
1434656
1438752
1442848
1446944
1451040
1455136
1459232
1460000
```

This complete part C.