lab2c report

code layout

```
$tree
.
├── checkpoints
│   ├── capture_echo.pcapng   # capture file when running echo test
│   ├── echo_client
│   ├── echo_client.c
│   ├── echo_server
│   ├── echo_server.c
│   ├── Makefile
│   ├── perf_client
│   ├── perf_client.c
│   ├── perf_server
│   ├── perf_server.c
│   ├── unp.c
│   └── unp.h
├── helper
│   ├── addNS
│   ├── addVethPair
│   ├── bypassKernel
│   ├── connectNS
│   ├── delNS
│   ├── delVeth
│   ├── enableForward
│   ├── execNS
│   ├── giveAddr
│   ├── setGRoute
│   ├── setNS
│   ├── setRoute
│   ├── undoBypass
│   ├── vdo
│   └── vecho
├── readme.pdf
├── run.sh     # run echo test and perf test
├── src
│   ├── arp.cpp
```

```
|   ├── arp.hpp
|   ├── build
|   ├── CMakeLists.txt
|   ├── device.cpp
|   ├── device.hpp
|   ├── general.cpp
|   ├── general.hpp
|   ├── iplayer.cpp
|   ├── iplayer.hpp
|   ├── packetio.cpp
|   ├── packetio.hpp
|   ├── sock.cpp
|   ├── sock.hpp
|   ├── tcplayer.cpp
|   ├── tcplayer.hpp
|   ├── tcp_timer.cpp
|   └── tcp_timer.hpp
└── test
    ├── init.sh
    ├── parameter.hpp
    ├── router       # as a router running in ns2 and ns3 and do nothing
    ├── router.cpp
    ├── test1b           # tset1a and test1b as a extra test
    ├── test1b.cpp
    ├── tset1a
    └── tset1a.cpp
```

First in `src/build` use `cmake .. && make` to get `libsrc.so`.

`test1b` is a program using linux network stack.

```
$g++ -std=c++17 -o test1b test1b.cpp -lgtest -lpthread
```

To get `tset1a` can refer to `test/.vscode/task.json`.

writing task 3

```
enum class tcp_status   // src/tcplayer.hpp
{
    CLOSED, // no closing for simultaneous close
    LISTEN,
    SYN_SENT,
    SYN_RCVD,
    ESTABLISHED,
    FIN_WAIT_1,
    FIN_WAIT_2,
    TIME_WAIT,
    CLOSE_WAIT,
    LAST_ACK,
    OTHER_ABORT, // no responds of other side over a long time, wait to be closed by user
    NO_USE,      // channel has been closed on both sides, wait closed by user
    ORPHAN,      // has been closed by user, can delete from sockmap directly
};
```

Here is all tcp status I defined. Besides general tcp status in rfc 793, I add three new status, `other_abort`, `no_use`, `orphan` respectively. If retransmission time reaches limitation and no ack received, I think other side has crashed and change status into `other_abort`. if a tcp connection channel has been closed by both sides and socket has not been closed(e.g., shutdown function), socket will enter `no_use` status after `time_wait` status. Any socket will be cleared if and only if its status is `orphan`. For example, if upper layer close a `no_use` socket, this socket will enter `orphan` status and be cleared later.

Any socket created by `socket` function is `closed` status and will enter `orhpan` status at last. To provide thread safety, each socket has a read-write lock and once we need to check some socket's information, first thing is to acquire its lock. For example, by `connect` function, a socket will enter `syn_sent` status from `closed` status, if another thread calls `connect` function simultaneously, only first thread will see `closed` status of socket and subsequent call will fail because of atomic status transformation.

To support message passing between threads, I define four condition variable in tcp_socket(corresponds to four situation where thread may be blocked).

```
class tcp_socket
{
public:
    shared_mutex rw_mtx;
    condition_variable_any cond_connect;     // for connect function
    condition_variable_any cond_accept;      // for accept function
    condition_variable_any cond_send;        // for write function
    condition_variable_any cond_recv;        // for read function
    ****
};
```

For example, thread that send syn packet will wait on `cond_connect` variable, once another thread received a syn-ack packet, we will find corresponding `syn_sent` socket and notify corresponding thread, socket status will enter `established`. In a corner case, another thread may close a `syn_sent` socket, we also need to notify corresponding thread and `connect` function will return -1.

Upper layer also can use `bind` and `listen` function to transfer socket status from `closed` into `listen`. If we receive a syn packet and find a `listen` socket in corresponding port, syn-ack packet will be sent and a new socket with `syn_rcvd` status will be created, or a rst packet will be sent. But before receiving a ack packet for this syn-ack packet and transferring its status into `established`, We can't add this socket into accept-queue of `listen` socket(so a `accept` call will get a `fd` descriptor from accept-queue). With the problem is that when get a ack packet for `syn_rcvd` socket, corresponding `listen` socket may have been closed by user ! In this situation, a rst packet will be sent and transfer socket status into `orphan`. Or we can add this socket into accept-queue of corresponding `listen` socket and notify one thread that waits on `cond_accept` variable. By the way, a `close` for `listen` socket will cause a status transformation into `orphan`(if accept-queue is not empty, recursive `close` for those sockets will be called).

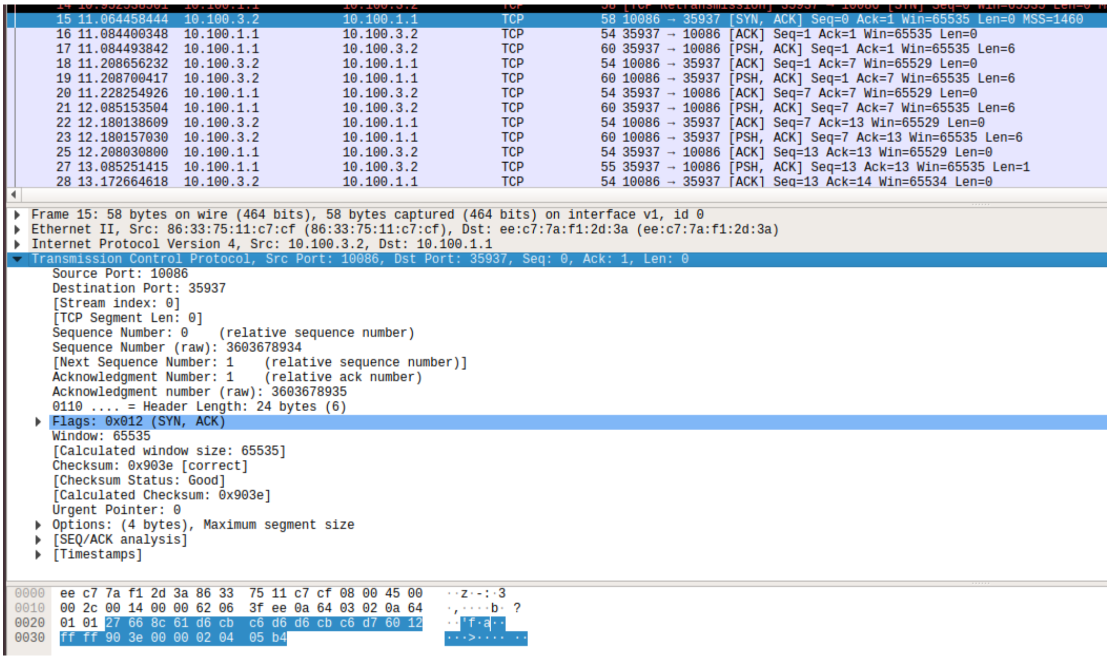
After entering `established` status, both sides can receive and send data normally. If retransmissions over a long time are not acknowledged by other side. I will think other side has crashed and transfer status into `other_abort`, sending data on this kind of socket will get 0 as return value to hint at a occurred exception. Once this socket is closed by user, it will be transferred into `orphan` status and cleared later. At some time we may received a ordered fin packet, which cause a transformation from `established` into `close_wait` or from `fin_wait_2` into `time_wait`. What happened if a out-of-order fin packet ?(e.g., some packet before it lost), We will buffer this fin packet as usual, but not change socket status. Of course, we will also send a ack packet to hint at next sequence number needed. Status transformation into `close_wait` or `time_wait`happens until I confirmed that all packets other side sent has been received. And a next sending ack packet will transfer status of other side into `fin_wait_2` or `orphan` or `no_use`(depend on if this socket have been closed by user). If this ack packet lost, another fin packet will be sent by other side and a new ack packet will be sent, not a big deal.

A `close` call acting on `established` or `close_wait` socket will elicit a transformation into `fin_wait_1` or `last_ack`, but immediacy can't be guaranteed(much data left in sending buffer because of window size limitation is possible. Only all data before has been sent at least once a fin packet will be sent and elicit a status transformation)

After two minutes, a `time_wait` socket will transfer into `orphan` socket.

That's the profile that I deal with tcp status changes.

checkpoint 7



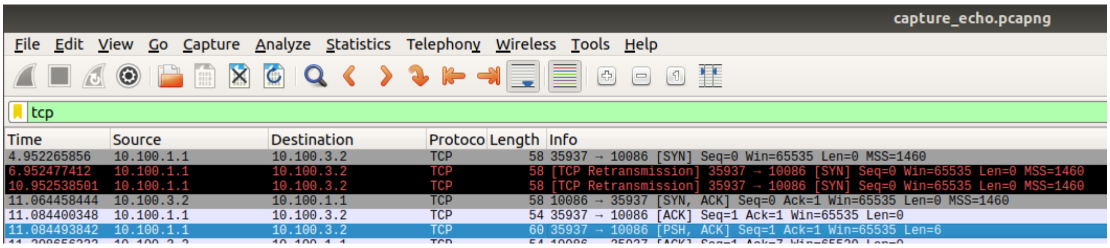
```
14 10.932538501  10.100.1.1     10.100.3.2     TCP     58 [TCP Retransmission] 35937 → 10086 [SYN] Seq=0 Win=65535 Len=0 M
15 11.064458444  10.100.3.2     10.100.1.1     TCP     58 10086 → 35937 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
16 11.084400348  10.100.1.1     10.100.3.2     TCP     54 35937 → 10086 [ACK] Seq=1 Ack=1 Win=65535 Len=0
17 11.084493842  10.100.1.1     10.100.3.2     TCP     60 35937 → 10086 [PSH, ACK] Seq=1 Ack=1 Win=65535 Len=6
18 11.208656232  10.100.3.2     10.100.1.1     TCP     54 10086 → 35937 [ACK] Seq=1 Ack=7 Win=65529 Len=0
19 11.208700417  10.100.3.2     10.100.1.1     TCP     60 10086 → 35937 [PSH, ACK] Seq=1 Ack=7 Win=65535 Len=6
20 11.228254926  10.100.1.1     10.100.3.2     TCP     54 35937 → 10086 [ACK] Seq=7 Ack=7 Win=65529 Len=0
21 12.085153504  10.100.1.1     10.100.3.2     TCP     60 35937 → 10086 [PSH, ACK] Seq=7 Ack=7 Win=65535 Len=6
22 12.180138609  10.100.3.2     10.100.1.1     TCP     54 10086 → 35937 [ACK] Seq=7 Ack=13 Win=65529 Len=0
23 12.180157030  10.100.3.2     10.100.1.1     TCP     60 10086 → 35937 [PSH, ACK] Seq=7 Ack=13 Win=65535 Len=6
25 12.208030800  10.100.1.1     10.100.3.2     TCP     54 35937 → 10086 [ACK] Seq=13 Ack=13 Win=65529 Len=0
27 13.085251415  10.100.1.1     10.100.3.2     TCP     55 35937 → 10086 [PSH, ACK] Seq=13 Ack=13 Win=65535 Len=1
28 13.172664618  10.100.3.2     10.100.1.1     TCP     54 10086 → 35937 [ACK] Seq=13 Ack=14 Win=65534 Len=0

▸ Frame 15: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface v1, id 0
▸ Ethernet II, Src: 86:33:75:11:c7:cf (86:33:75:11:c7:cf), Dst: ee:c7:7a:f1:2d:3a (ee:c7:7a:f1:2d:3a)
▸ Internet Protocol Version 4, Src: 10.100.3.2, Dst: 10.100.1.1
▾ Transmission Control Protocol, Src Port: 10086, Dst Port: 35937, Seq: 0, Ack: 1, Len: 0
     Source Port: 10086
     Destination Port: 35937
     [Stream index: 0]
     [TCP Segment Len: 0]
     Sequence Number: 0    (relative sequence number)
     Sequence Number (raw): 3603678934
     [Next Sequence Number: 1    (relative sequence number)]
     Acknowledgment Number: 1    (relative ack number)
     Acknowledgment number (raw): 3603678935
     0110 .... = Header Length: 24 bytes (6)
   ▸ Flags: 0x012 (SYN, ACK)
     Window: 65535
     [Calculated window size: 65535]
     Checksum: 0x903e [correct]
     [Checksum Status: Good]
     [Calculated Checksum: 0x903e]
     Urgent Pointer: 0
   ▸ Options: (4 bytes), Maximum segment size
   ▸ [SEQ/ACK analysis]
   ▸ [Timestamps]

0000  ee c7 7a f1 2d 3a 86 33  75 11 c7 cf 08 00 45 00   ··z·-:·3
0010  00 2c 00 14 00 00 62 06  3f ee 0a 64 03 02 0a 64   ·,····b· ?
0020  01 01 27 66 8c 61 d6 cb  c6 d6 d6 cb c6 d7 60 12   ··'f·a··
0030  ff ff 90 3e 00 00 02 04  05 b4                     ···>···· ··
```

Above screenshot is from `checkpoints/capture_echo.pcap` , I will show each Byte's meaning in the TCP header of frame 15.

- First two bytes means source port and 3th, 4th Bytes means destination port number.
- 5th, 6th, 7th, 8th bytes is sequence number of this packet, and 9th, 10th, 11th, 12th is next sequence number sender want to receive.
- Next 4 bits are header length taken 4 bytes as a unit. and following 12 bits are flag bits, besides reserved bits, including urgent bit(urgent pointer field is valid ?), acknowledgment bit(acknowledgment number field is valid ?), push bit(these data need to be sent to upper layer immediately), reset bit(reset this connection), syn bit(request a new connection), fin bit(close this connection).
- 15th, 16th bits are window size of sender, receiver can't send data once whose size exceeds this window size limitation.
- 17th, 18th is checksum of tcp packet, tcp pseudo header and tcp payload will be included in checksum calculation.
- 19th, 20th is urgent pointer field, payload before this value should be sent to upper layer immediately.
- the following bytes of header are tcp option. 21th byte means option kind, here is mss size, 22th is option length. 23th, 24th are size of mss.

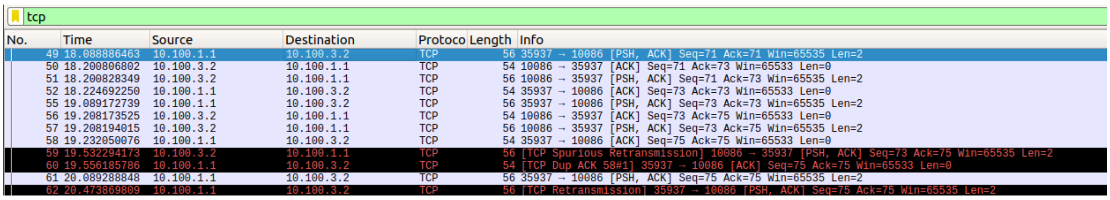
checkpoint 8

`checkpoints/capture_echo.pcap` is got in dev v1 with 20% packet loss rate in dev v3 (checking run.sh script will be clear). here I paste some screenshot from `checkpoints/capture_echo.pcap` to show reliable delivery.


capture_echo.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp

Time	Source	Destination	Protoco	Length	Info
4.952265856	10.100.1.1	10.100.3.2	TCP	58	35937 → 10086 [SYN] Seq=0 Win=65535 Len=0 MSS=1460
6.952477412	10.100.1.1	10.100.3.2	TCP	58	[TCP Retransmission] 35937 → 10086 [SYN] Seq=0 Win=65535 Len=0 MSS=1460
10.952538501	10.100.1.1	10.100.3.2	TCP	58	[TCP Retransmission] 35937 → 10086 [SYN] Seq=0 Win=65535 Len=0 MSS=1460
11.064458444	10.100.3.2	10.100.1.1	TCP	58	10086 → 35937 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
11.084400348	10.100.1.1	10.100.3.2	TCP	54	35937 → 10086 [ACK] Seq=1 Ack=1 Win=65535 Len=0
11.084493842	10.100.1.1	10.100.3.2	TCP	60	35937 → 10086 [PSH, ACK] Seq=1 Ack=1 Win=65535 Len=6

The packet delivery route has not been found first, So first two syn packets are lost, But third syn packet build a connection successfully.



No.	Time	Source	Destination	Protoco	Length	Info
49	18.088886463	10.100.1.1	10.100.3.2	TCP	56	35937 → 10086 [PSH, ACK] Seq=71 Ack=71 Win=65535 Len=2
50	18.200806802	10.100.3.2	10.100.1.1	TCP	54	10086 → 35937 [ACK] Seq=71 Ack=73 Win=65533 Len=0
51	18.200828349	10.100.3.2	10.100.1.1	TCP	56	10086 → 35937 [PSH, ACK] Seq=71 Ack=73 Win=65535 Len=2
52	18.224692250	10.100.1.1	10.100.3.2	TCP	54	35937 → 10086 [ACK] Seq=73 Ack=73 Win=65533 Len=0
55	19.089172739	10.100.1.1	10.100.3.2	TCP	56	35937 → 10086 [PSH, ACK] Seq=73 Ack=73 Win=65535 Len=2
56	19.208173525	10.100.3.2	10.100.1.1	TCP	54	10086 → 35937 [ACK] Seq=73 Ack=75 Win=65533 Len=0
57	19.208194015	10.100.3.2	10.100.1.1	TCP	56	10086 → 35937 [PSH, ACK] Seq=73 Ack=75 Win=65535 Len=2
58	19.232050076	10.100.1.1	10.100.3.2	TCP	54	35937 → 10086 [ACK] Seq=75 Ack=75 Win=65533 Len=0
59	19.532294173	10.100.3.2	10.100.1.1	TCP	56	[TCP Spurious Retransmission] 10086 → 35937 [PSH, ACK] Seq=73 Ack=75 Win=65535 Len=2
60	19.556185786	10.100.1.1	10.100.3.2	TCP	54	[TCP Dup ACK 58#1] 35937 → 10086 [ACK] Seq=75 Ack=75 Win=65533 Len=0
61	20.089288848	10.100.1.1	10.100.3.2	TCP	56	35937 → 10086 [PSH, ACK] Seq=75 Ack=75 Win=65535 Len=2
62	20.473869809	10.100.1.1	10.100.3.2	TCP	56	[TCP Retransmission] 35937 → 10086 [PSH, ACK] Seq=75 Ack=75 Win=65535 Len=2

Packet 58 as a ack for packet 57 is lost in dev v3. So server in ns4 retransmit packet 59 and wireshark considers this as a spurious retransmission.

checkpoint 9

Running `./run.sh echo` simply, it will initiate running environment and get the following output

```
$./run.sh echo
# omit output of makefile
new connection
6 12 13 14 63 68 70 72 74 76 78 80 82 84 86 87 88 89 1549 4184 5644
8279 9739 12374 13834 15000 loop #1 ok.
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 loop #2 ok.
all: 15000
new connection
6 12 13 14 63 68 70 72 74 76 78 80 82 84 86 87 88 89 1549 4184 5644
8279 9739 12374 13834 15000 loop #3 ok.
all: 15000
```

checkpoint 10

Running `./run.sh perf` simply, it will initiate running environment and get the following output.

```
$./run.sh perf
# omit output of makefile
sending ...
new connection
receiving ...
212.30 KB/s
sending ...
receiving ...
202.91 KB/s
sending ...
receiving ...
202.82 KB/s
sending ...
receiving ...
139.36 KB/s
sending ...
receiving ...
139.31 KB/s
sending ...
receiving ...
138.88 KB/s
sending ...
receiving ...
139.31 KB/s
sending ...
receiving ...
168.32 KB/s
sending ...
receiving ...
150.23 KB/s
sending ...
receiving ...
150.34 KB/s
all: 1460000
```

Writing Task 4

I have written a test program to communicate between Linux socket and my tcp socket. Here is the test method.

```
$cat ./init.sh    # in test folder
#! /bin/sh

export LD_LIBRARY_PATH=../src/build
cd ../helper
./addNS ns1
./addNS ns2
./connectNS ns1 ns2 v1 v2 10.100.1
./execNS ns1 tc qdisc add dev v1 root netem delay 1s reorder 10% loss 10%
./execNS ns2 tc qdisc add dev v2 root netem delay 1s reorder 10% loss 10%
./execNS ns1 ./bypassKernel
```

We add reorder and loss into each device and only close kernel network stack of ns1 where we will run `test1a` based on my own tcp/ip stack. And run `test1b` in ns2 which uses kernel network stack.

```
$ LD_LIBRARY_PATH=..src/build ip netns exec ns1 ./tset1a
$ LD_LIBRARY_PATH=..src/build ip netns exec ns2 ./test1b  # another terminal
# test will run for 6 minutes because of terrible link state.
```

`Test1a` as a server waits connection from `test1b`, both sides will send and receive 65536 * 2 + 8 bytes. Last 8 bytes are hash value of first 65536 * 2 bytes using std::hash<string>.Use google test and only both sides receive expected size of data and hash value is correct the test will pass.

expected output:

```
$ip netns exec ns2 ./test1b
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from simpleRecevingTest
[ RUN      ] simpleRecevingTest.connectReceiving
[       OK ] simpleRecevingTest.connectReceiving (289236 ms)
[----------] 1 test from simpleRecevingTest (289236 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (289236 ms total)
[  PASSED  ] 1 test.

$ip netns exec ns1 ./tset1a          # another terminal
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
```

```
[----------] 1 test from listenWritingTest
[ RUN      ] listenWritingTest.listenTest
[       OK ] listenWritingTest.listenTest (302404 ms)
[----------] 1 test from listenWritingTest (302404 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (302404 ms total)
[  PASSED  ] 1 test.
```