

Lab2 网络传输机制

一、实验目的

同学们需要仔细阅读实验的工程文件，在此基础上完善一个基本的TCP协议栈，包括：

- 实现TCP socket的API
- 实现TCP建立和断开连接的状态转移逻辑
- 实现TCP接收/发送数据包的处理过程
- 实现应用层的短消息收发和大文件传送功能，对协议栈进行测试

希望在做完实验之后，同学们能够更深入地掌握TCP报文段的结构、TCP socket的实现、TCP三次握手和四次挥手的状态转移逻辑以及TCP收发数据包的处理流程，增强同学们在理论课上的理解。

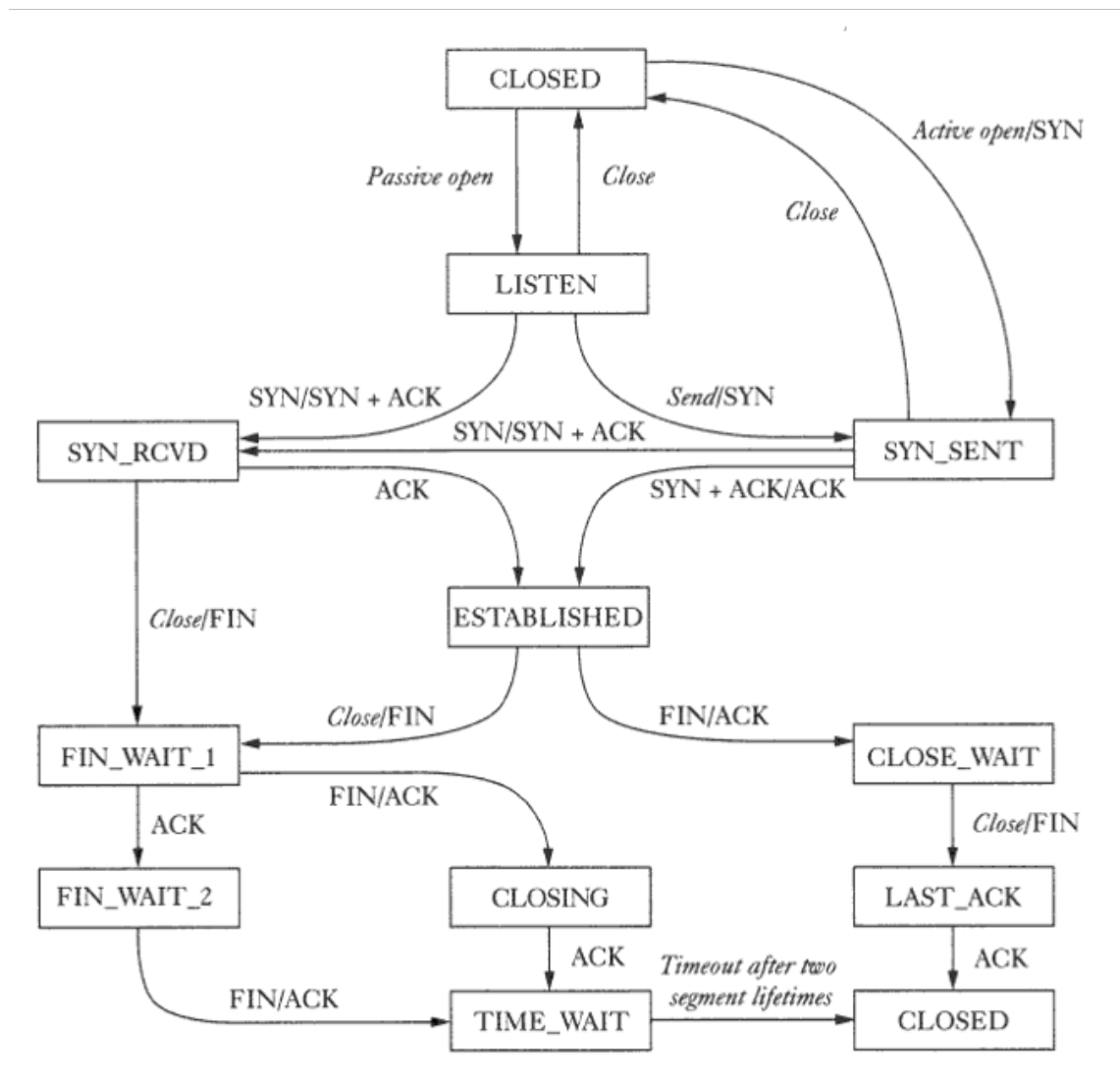
二、相关资料

1. 附件文件列表

- **main.c**：整个工程的入口函数，初始化tcp_stack，解析用户的命令行参数，执行tcp_app.c中实现的客户端/服务端应用程序
- **tcp_app.c**：实现应用层的客户端/服务端程序，包括短文件收发和大文件传送
- **tcp.c**：实现了tcp报文段结构和tcp控制块结构，是处理收到的TCP数据包的入口函数
- **tcp_socket.c**：实现TCP socket的API
- **tcp_in.c**：实现TCP接收数据包的相关函数
- **tcp_out.c**：实现了TCP发送数据包的相关函数
- **tcp_timer.c**：TCP定时器，用来实现TCP断开连接时的time_wait定时器（本次实验），以及发生丢包时的重传定时器（下次实验）
- **tcp_stack_conn.py**：python应用程序实现，调用操作系统的TCP协议栈，用来测试连接管理
- **tcp_stack_trans.py**：python应用程序实现，调用操作系统的TCP协议栈，用来测试短消息收发
- **tcp_topo.py**：创建Mininet拓扑，包含两个节点，链路不存在丢包

2. TCP状态转移图

可以根据这张TCP状态转移图的逻辑，编写time_in.c中的tcp_process函数。



三、实验内容

1. 连接管理

1.1. 编写源代码

在编写源代码之前，你需要了解工程中各个文件的分工，以及函数之间的调用关系。函数上面的注释会说明它所实现的功能，会对大家阅读和编写源码提供帮助。

- 实现tcp_socket.c中的相关函数

```

void free_tcp_sock(struct tcp_sock *tsk);
struct tcp_sock *tcp_sock_lookup_established(u32 saddr, u32 daddr, u16 sport, u16 dport);
struct tcp_sock *tcp_sock_lookup_listen(u32 saddr, u16 sport);
int tcp_sock_connect(struct tcp_sock *tsk, struct sock_addr *skaddr);
int tcp_sock_listen(struct tcp_sock *tsk, int backlog);
struct tcp_sock *tcp_sock_accept(struct tcp_sock *tsk);
void tcp_sock_close(struct tcp_sock *tsk);
int tcp_sock_read(struct tcp_sock *tsk, char *buf, int len);
int tcp_sock_write(struct tcp_sock *tsk, char *buf, int len);
  
```

- 实现tcp_timer.c中的相关函数

```
void tcp_scan_timer_list();  
void tcp_set_timewait_timer(struct tcp_sock *tsk);
```

- 实现time_in.c中的相关函数

```
void tcp_process(struct tcp_sock *tsk, struct tcp_cb *cb, char *packet);
```

1.2. 编译测试

- 编译

```
# 编译  
make clean  
make  
sudo python3 tcp_topo.py
```

- 测试1：服务端和客户端都执行tcp_stack

```
# 主机h1 server执行命令如下：  
dump # 查看主机h1 h2的pid号  
h1 ./scripts/disable_tcp_rst.sh # 执行两个脚本禁止协议栈相应功能  
h1 ./scripts/disable_offloading.sh  
h1 ./tcp_stack server 10001  
  
# 主机h2 client执行命令如下：  
sudo mnexec -a <pid> ./scripts/disable_tcp_rst.sh # 这里的pid是h2的pid号  
sudo mnexec -a <pid> ./scripts/disable_offloading.sh  
sudo mnexec -a <pid> ./tcp_stack client 10.0.0.1 10001
```

- 测试2：服务端执行tcp_stack_conn.py，客户端执行tcp_stack

```
# 主机h1 server执行命令如下：  
dump  
h1 ./scripts/disable_tcp_rst.sh  
h1 ./scripts/disable_offloading.sh  
h1 python3 tcp_stack_conn.py server 10001  
  
# 主机h2 client执行命令如下：  
sudo mnexec -a <pid> ./scripts/disable_tcp_rst.sh  
sudo mnexec -a <pid> ./scripts/disable_offloading.sh  
sudo mnexec -a <pid> ./tcp_stack client 10.0.0.1 10001
```

- 测试3：服务端执行tcp_stack，客户端执行tcp_stack_conn.py

主机h1 server执行命令如下:

```
dump
h1 ./scripts/disable_tcp_rst.sh
h1 ./scripts/disable_offloading.sh
h1 ./tcp_stack server 10001
```

主机h2 client执行命令如下:

```
sudo mnexec -a <pid> ./scripts/disable_tcp_rst.sh
sudo mnexec -a <pid> ./scripts/disable_offloading.sh
sudo mnexec -a <pid> python3 tcp_stack_conn.py client 10.0.0.1 10001
```

1.3. 实验预期结果

h2上显示连接成功之后, 然后自动断开连接, 可以看到server端和client端的状态转移过程符合TCP状态转移图。

```
liuyu@ubuntu22arm:~/02-tcp_stack_test$ sudo python3 tcp_topo.py
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=41044>
<Host h2: h2-eth0:10.0.0.2 pid=41046>
mininet> h1 ./scripts/disable_tcp_rst.sh
mininet> h1 ./scripts/disable_offloading.sh
Disabling h1-eth0 ...
mininet> h1 ./tcp_stack server 10001
DEBUG: find the following interfaces: h1-eth0.
Routing table of 1 entries has been loaded.
DEBUG: listening port 10001.
DEBUG: 0.0.0.0:10001 switch state, from CLOSED to LISTEN.
DEBUG: listen to port 10001.
DEBUG: 10.0.0.1:10001 switch state, from CLOSED to SYN_RECV.
DEBUG: Pass 10.0.0.1:10001 <=> 10.0.0.2:12345 from process to listen_queue
DEBUG: 10.0.0.1:10001 switch state, from SYN_RECV to ESTABLISHED.
DEBUG: accept a connection.
DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT.
DEBUG: close sock 10.0.0.1:10001 <=> 10.0.0.2:12345, state CLOSE_WAIT
DEBUG: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK.
DEBUG: Do free 10.0.0.1:10001 <=> 10.0.0.2:12345.
DEBUG: Current state is TCP_LISTEN but rcv not SYN

liuyu@ubuntu22arm:~/02-tcp_stack_test$ sudo mnexec -a 41046 ./scripts/disable_tcp_rst.sh
[sudo] password for liuyu:
liuyu@ubuntu22arm:~/02-tcp_stack_test$ sudo mnexec -a 41046 ./scripts/disable_offloading.sh
Disabling h2-eth0 ...
liuyu@ubuntu22arm:~/02-tcp_stack_test$ sudo mnexec -a 41046 ./tcp_stack client 10.0.0.1 10001
DEBUG: find the following interfaces: h2-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.2:12345 switch state, from CLOSED to SYN_SENT.
DEBUG: 10.0.0.2:12345 switch state, from SYN_SENT to ESTABLISHED.
DEBUG: close sock 10.0.0.2:12345 <=> 10.0.0.1:10001, state ESTABLISHED
DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to FIN_WAIT-2.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-2 to TIME_WAIT.
DEBUG: insert 10.0.0.2:12345 <=> 10.0.0.1:10001 to timewait, ref_cnt += 1
DEBUG: 10.0.0.2:12345 switch state, from TIME_WAIT to CLOSED.
DEBUG: Do free 10.0.0.2:12345 <=> 10.0.0.1:10001.
[]
```

2. 短消息收发

- 参照tcp_stack_trans.py, 修改tcp_apps.c, 使之能够收发短消息。tcp_app.c实现的功能和tcp_stack_trans.py保持一致, 需要注意的细节是, Client端发送的字符串循环不能少于5次, Server端返回字符串给Client时需要加上前缀“server echoes: ”, 冒号后面有空格。
- 编译测试流程和实验内容1的步骤相同
- 实验预期结果中, Client端会回显出一段字符串“server echo:”, 如下所示:

```
liuyu@ubuntu22arm:~/02-tcp_stack_test$ sudo python3 tcp_topo.py
[sudo] password for liuyu:
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=44358>
<Host h2: h2-eth0:10.0.0.2 pid=44360>
mininet> h1 ./scripts/disable_tcp_rst.sh
mininet> h1 ./scripts/disable_offloading.sh
Disabling h1-eth0 ...
mininet> h1 ./tcp_stack server 10001
DEBUG: find the following interfaces: h1-eth0.
Routing table of 1 entries has been loaded.
DEBUG: listening port 10001.
DEBUG: 0.0.0.0:10001 switch state, from CLOSED to LISTEN.
DEBUG: listen to port 10001.
DEBUG: 10.0.0.1:10001 switch state, from CLOSED to SYN_RECV.
DEBUG: Pass 10.0.0.1:10001 <=> 10.0.0.2:12345 from process to listen_queue
DEBUG: 10.0.0.1:10001 switch state, from SYN_RECV to ESTABLISHED.
DEBUG: accept a connection.
DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT.
DEBUG: tcp_sock_read return 0, finish transmission.
DEBUG: close this connection.
DEBUG: close sock 10.0.0.1:10001 <=> 10.0.0.2:12345, state CLOSE_WAIT
DEBUG: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK.
DEBUG: Do free 10.0.0.1:10001 <=> 10.0.0.2:12345.
DEBUG: Current state is TCP_LISTEN but rcv not SYN

liuyu@ubuntu22arm:~/02-tcp_stack_test$ sudo mnexec -a 44360 ./scripts/disable_tcp_rst.sh
liuyu@ubuntu22arm:~/02-tcp_stack_test$ sudo mnexec -a 44360 ./scripts/disable_offloading.sh
Disabling h2-eth0 ...
liuyu@ubuntu22arm:~/02-tcp_stack_test$ sudo mnexec -a 44360 ./tcp_stack client 10.0.0.1 10001
DEBUG: find the following interfaces: h2-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.2:12345 switch state, from CLOSED to SYN_SENT.
DEBUG: 10.0.0.2:12345 switch state, from SYN_SENT to ESTABLISHED.
server echoes: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 23456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 3456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 56789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 6789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 89abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
server echoes: 9abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
DEBUG: close sock 10.0.0.2:12345 <=> 10.0.0.1:10001, state ESTABLISHED
DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to FIN_WAIT-2.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-2 to TIME_WAIT.
DEBUG: insert 10.0.0.2:12345 <=> 10.0.0.1:10001 to timewait, ref_cnt += 1
DEBUG: 10.0.0.2:12345 switch state, from TIME_WAIT to CLOSED.
DEBUG: Do free 10.0.0.2:12345 <=> 10.0.0.1:10001.
```

3. 大文件传送

- 执行create_randfile.sh脚本 `./create_randfile.sh`, 生成待传输数据文件client-input.dat。
- 修改tcp_apps.c以及tcp_stack_trans.py, 使之能够收发文件。具体的逻辑是: Client将client-input.dat中的内容传输给Server, Server将收到的内容存入server-output.dat中。
- 编译测试流程和实验内容1的步骤相同。

- 实验预期结果是Server收到的内容server- output.dat和Client发送的内容client-input.dat完全一致，可以通过md5sum命令比较两个文件是否完全相同，如下所示：

```
liuyu@ubuntu22arm:~/02-tcp_stack_test$ ./create_randfile.sh
3+0 records in
3+0 records out
3000000 bytes (3.0 MB, 2.9 MiB) copied, 0.146455 s, 20.5 MB/s
liuyu@ubuntu22arm:~/02-tcp_stack_test$ sudo python3 tcp_topo.py
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=47907>
<Host h2: h2-eth0:10.0.0.2 pid=47909>
mininet> h1 ./scripts/disable_tcp_rst.sh
mininet> h1 ./scripts/disable_offloading.sh
Disabling h1-eth0 ...
mininet> h1 ./tcp_stack server 10001
DEBUG: find the following interfaces: h1-eth0.
Routing table of 1 entries has been loaded.
DEBUG: open file server-output.dat
DEBUG: listening port 10001.
DEBUG: 0.0.0.0:10001 switch state, from CLOSED to LISTEN.
DEBUG: listen to port 10001.
DEBUG: 10.0.0.1:10001 switch state, from CLOSED to SYN_RECV.
DEBUG: Pass 10.0.0.1:10001 <=> 10.0.0.2:12345 from process to listen_queue
DEBUG: 10.0.0.1:10001 switch state, from SYN_RECV to ESTABLISHED.
DEBUG: accept a connection.
DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT.
DEBUG: peer closed.
used time: 4 s
DEBUG: close this connection.
DEBUG: close sock 10.0.0.1:10001 <=> 10.0.0.2:12345, state CLOSE_WAIT
DEBUG: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK.
DEBUG: Do free 10.0.0.1:10001 <=> 10.0.0.2:12345.
DEBUG: Current state is TCP_LISTEN but rcv not SYN
DEBUG: sent 3839192 Bytes
DEBUG: sent 3849216 Bytes
DEBUG: sent 3859240 Bytes
DEBUG: sent 3869264 Bytes
DEBUG: sent 3879288 Bytes
DEBUG: sent 3889312 Bytes
DEBUG: sent 3899336 Bytes
DEBUG: sent 3909360 Bytes
DEBUG: sent 3919384 Bytes
DEBUG: sent 3929408 Bytes
DEBUG: sent 3939432 Bytes
DEBUG: sent 3949456 Bytes
DEBUG: sent 3959480 Bytes
DEBUG: sent 3969504 Bytes
DEBUG: sent 3979528 Bytes
DEBUG: sent 3989552 Bytes
DEBUG: sent 3999576 Bytes
DEBUG: sent 4009600 Bytes
DEBUG: sent 4019624 Bytes
DEBUG: sent 4029648 Bytes
DEBUG: sent 4039672 Bytes
DEBUG: sent 4049696 Bytes
DEBUG: sent 4052632 Bytes
DEBUG: the file has been sent completely.
DEBUG: close sock 10.0.0.2:12345 <=> 10.0.0.1:10001, state ESTABLISHED
DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to FIN_WAIT-2.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-2 to TIME_WAIT.
DEBUG: insert 10.0.0.2:12345 <=> 10.0.0.1:10001 to timewait, ref_cnt += 1
DEBUG: 10.0.0.2:12345 switch state, from TIME_WAIT to CLOSED.
DEBUG: Do free 10.0.0.2:12345 <=> 10.0.0.1:10001.
liuyu@ubuntu22arm:~/02-tcp_stack_test$ md5sum client-input.dat server-output.dat > compare.md5
liuyu@ubuntu22arm:~/02-tcp_stack_test$ md5sum -c compare.md5
client-input.dat: OK
server-output.dat: OK
liuyu@ubuntu22arm:~/02-tcp_stack_test$
```

四、实验注意事项

- 实验工程文件中对于网络层和链路层相关功能的实现，都被封装到了libnet.so动态库中，是同学们后续的实验内容。
- 同学们在OJ平台上的提交的工程压缩包，需要确保没有修改Makefile文件，否则可能造成编译不通过。
- 本次实验中，同学们需要在OJ上通过两个测试，分别是短消息收发的测试——网络传输实验（echo），以及大文件传输的测试——网络传输实验（文件传输）。