

# TCP 网络传输机制实验一报告

张磊 2017K8009922027

## 一、 实验题目

TCP 网络传输机制实验一

## 二、 实验内容

1. 运行给定网络拓扑(tcp\_topo.py);
2. 在节点 h1 上执行 TCP 程序:  
在节点 h1 上运行 TCP 协议栈的服务器模式(`./tcp_stack server 10001`);
3. 在节点 h2 上执行 TCP 程序:  
在节点 h2 上运行 TCP 协议栈的客户端模式, 连接至 h1, 显示建立连接成功后自动关闭连接(`./tcp_stack client 10.0.0.1 10001`);
4. 可以在一端用 `tcp_stack.py` 替换 `tcp_stack` 执行, 测试另一端;
5. 通过 wireshark 抓包来验证建立和关闭连接的正确性;

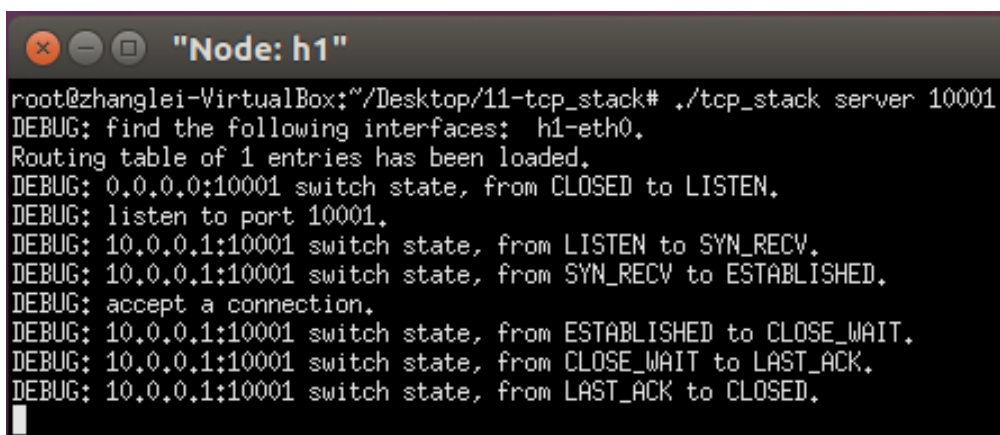
## 三、 实验流程

1. 实现附件中 `tcp_sock.c` 中的相关函数;
2. 实现附件中 `tcp_in.c` 中的相关函数;
3. 实现附件中 `tcp_timer.c` 中的相关函数;
4. 在节点 h1 上运行 wireshark 抓包;
5. 在节点 h1 上运行 `./tcp_stack server 10001`;
6. 在节点 h2 上运行 wireshark 抓包;

7. 在节点 h2 上运行 `./tcp_stack client 10.0.0.1 10001`;
8. 观察 h1 和 h2 连接建立到关闭过程的 DEBUG 信息;
9. 观察 h1 和 h2 的 wireshark 抓包结果

## 四、 实验结果

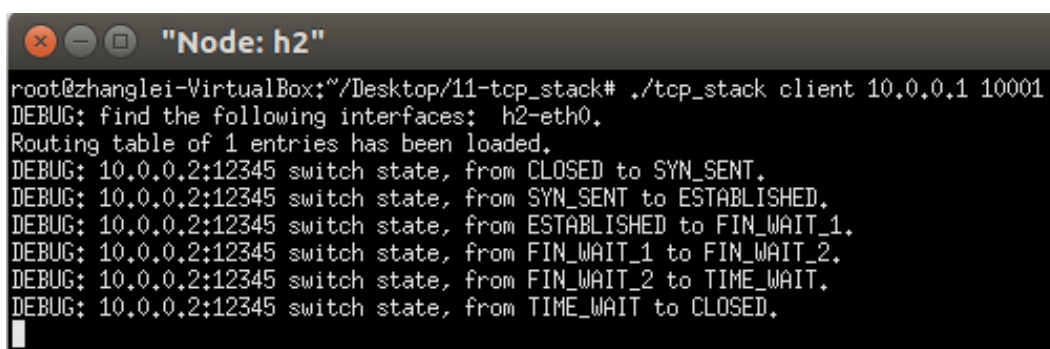
1. H1 输出信息:



```
root@zhanglei-VirtualBox:~/Desktop/11-tcp_stack# ./tcp_stack server 10001
DEBUG: find the following interfaces: h1-eth0.
Routing table of 1 entries has been loaded.
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 LISTEN to SYN_RECV.
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: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK.
DEBUG: 10.0.0.1:10001 switch state, from LAST_ACK to CLOSED.
```

H1 输出信息

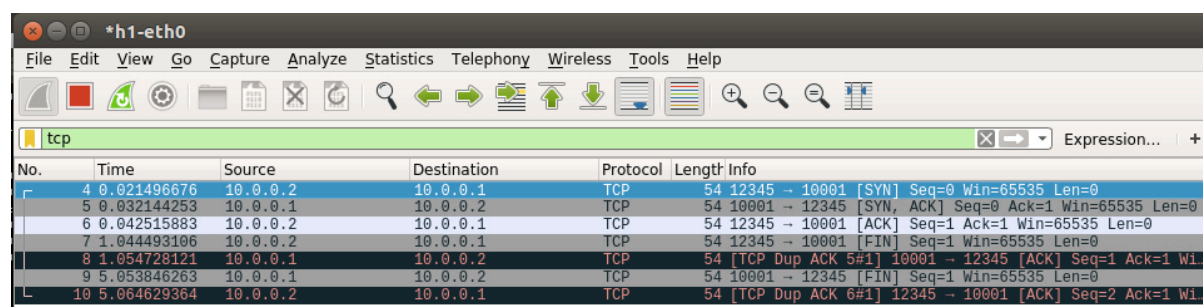
2. H2 输出信息:



```
root@zhanglei-VirtualBox:~/Desktop/11-tcp_stack# ./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: 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: 10.0.0.2:12345 switch state, from TIME_WAIT to CLOSED.
```

H2 输出信息

### 3. H1-Wireshark 抓包结果:

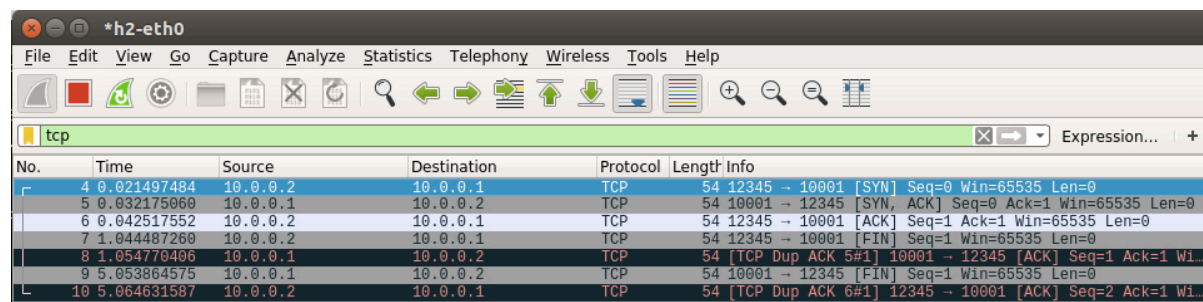


The screenshot shows a Wireshark capture on interface h1-eth0. The filter is set to 'tcp'. The packet list shows the following packets:

No.	Time	Source	Destination	Protocol	Length	Info
4	0.021496676	10.0.0.2	10.0.0.1	TCP	54	12345 → 10001 [SYN] Seq=0 Win=65535 Len=0
5	0.032144253	10.0.0.1	10.0.0.2	TCP	54	10001 → 12345 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
6	0.042515883	10.0.0.2	10.0.0.1	TCP	54	12345 → 10001 [ACK] Seq=1 Ack=1 Win=65535 Len=0
7	1.044493196	10.0.0.2	10.0.0.1	TCP	54	12345 → 10001 [FIN] Seq=1 Win=65535 Len=0
8	1.054728121	10.0.0.1	10.0.0.2	TCP	54	[TCP Dup ACK 5#1] 10001 → 12345 [ACK] Seq=1 Ack=1 Win=65535 Len=0
9	5.053846263	10.0.0.1	10.0.0.2	TCP	54	10001 → 12345 [FIN] Seq=1 Win=65535 Len=0
10	5.064629364	10.0.0.2	10.0.0.1	TCP	54	[TCP Dup ACK 6#1] 12345 → 10001 [ACK] Seq=2 Ack=1 Win=65535 Len=0

H1-Wireshark 抓包结果

### 4. H2-Wireshark 抓包结果:



The screenshot shows a Wireshark capture on interface h2-eth0. The filter is set to 'tcp'. The packet list shows the following packets:

No.	Time	Source	Destination	Protocol	Length	Info
4	0.021497484	10.0.0.2	10.0.0.1	TCP	54	12345 → 10001 [SYN] Seq=0 Win=65535 Len=0
5	0.032175000	10.0.0.1	10.0.0.2	TCP	54	10001 → 12345 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
6	0.042517552	10.0.0.2	10.0.0.1	TCP	54	12345 → 10001 [ACK] Seq=1 Ack=1 Win=65535 Len=0
7	1.044487260	10.0.0.2	10.0.0.1	TCP	54	12345 → 10001 [FIN] Seq=1 Win=65535 Len=0
8	1.054770406	10.0.0.1	10.0.0.2	TCP	54	[TCP Dup ACK 5#1] 10001 → 12345 [ACK] Seq=1 Ack=1 Win=65535 Len=0
9	5.053864575	10.0.0.1	10.0.0.2	TCP	54	10001 → 12345 [FIN] Seq=1 Win=65535 Len=0
10	5.064631587	10.0.0.2	10.0.0.1	TCP	54	[TCP Dup ACK 6#1] 12345 → 10001 [ACK] Seq=2 Ack=1 Win=65535 Len=0

H2-Wireshark 抓包结果

## 五、 实验分析

1. 首先，通过观察 h1 终端输出的 DEBUG 信息，我们可以看到，h1 作为 TCP 栈的 server 端，依次经历了 CLOSED, LISTEN, SYN\_RECV, ESTABLISHED, CLOSE\_WAIT, LAST\_ACK, CLOSED 这几个状态，成功的建立并关闭 TCP 连接；
2. 观察 h2 终端输出的 DEBUG 信息，我们可以看到，h2 作为 TCP 栈的 client 端，依次经历了 CLOSED, SYN\_SENT, ESTABLISHED, FIN\_WAIT\_1, FIN\_WAIT\_2, TIME\_WAIT, CLOSED 这几个状态，成功的建立并关闭了 TCP 连接；
3. 通过观察 h1 和 h2 的抓包结果，我们看到，h2 首先向 h1 发送了一个 syn 包，随后 h1 向 h2 回复 syn 和 ack 包，h2 收到后向 h1 发送 ack 包，至此，h1 和 h2 之间完成了 TCP 连接的建立；

4. 随后, h2 向 h1 发送 fin 包, h1 收到 fin 包后向 h2 回复 ack, 再然后 h1 向 h2 发送 fin 包, h2 收到 fin 包后向 h1 回复 ack 包, h1 收到来自 h2 的 ack 包, 至此, h1 和 h2 完成 TCP 连接的关闭;

## 六、 反思总结

1. 本次实验较为繁琐, 一开始没有找到头绪, 先从 main 函数开始看了一遍代码, 和需要实现的函数, 然后又看了一次老师的讲解视频, 然后又看了一遍理论课上讲解的 TCP 的内容, 最后晚上睡觉的时候过了一遍内容, 才把条理弄清楚;
2. 最后完成之后, 感觉这次实验和之前写的 CPU 有点像, 都是根据状态机状态进行相应的处理, 本次实验的几个状态非常完美的讲 server 端和 client 端的状态分开了(除了 CLOSE 状态), 这样实现起来就方便很多了;
3. 本来一开始, 我是想先根据收到的包是 syn 包, 还是 fin 包, 还是 ack 包, 再用不同状态来确定具体操作, 但是看到 tcp\_process 上老师给的注释是根据状态来区分, 于是就改了, 后来发现, 还是根据状态来确定做些什么逻辑更加清晰;
4. 通过这次实验, 让我对 TCP 的连接的建立和关闭有了更加清晰和深刻的理解, 之前一直不太清楚的三次握手协议现在也搞清楚了;

## 七、 参考文献

i

ii

---

<sup>i</sup> 中国科学院大学 2020 春计算机网络研讨课 11-网络传输机制实验一实验课件

<sup>ii</sup> 中国科学院大学 2020 春计算机网络研讨课 11-网络传输机制实验一实验附件