实验报告

姓名： 袁峥 学号： 2015K8009929008

一、实验题目：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

执行脚本(disable\_arp.sh, disable\_icmp.sh, disable\_ip\_forward.sh, disable\_tcp\_rst.sh, disable\_offloading.sh)，禁止协议栈的相应功能

在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

执行脚本(disable\_arp.sh, disable\_icmp.sh, disable\_ip\_forward.sh, disable\_tcp\_rst.sh, disable\_offloading.sh)，禁止协议栈的相应功能

在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、通过wireshark抓包来来验证建立和关闭连接的正确性

三、实验流程

1、TCP连接管理和状态迁移

①状态迁移图



②Socket与元组信息的绑定

在刚分配socket时，socket不与任何地址、端口绑定。在bind和listen阶段，socket与本地地址和监听端口绑定。在accept和connect时，socket与四元组绑定。

协议栈维护listen\_table和established\_table两个hash表，来分别组织只绑定源地址、端口的socket和绑定四元组的socket

③通过数据包信息查找对应的Socket

对于源目的地址、源目的端口都已经确定下来的socket，按照上述4元组，将hash\_list节点hash到established\_table

对于只知道源地址、源端口的socket，按照上述2元组，将hash\_list节点hash到listen\_table

任何占用一个本地端口的socket，按照该端口号将bind\_hash\_list 节点hash到bind\_table

对于一个新到达的数据包，先在established\_table中查找相应socket，如果没有找到，再到listen\_table中查找相应socket

④Socket队列

socket.list

用于将该socket放入到parent socket的队列中

socket.listen\_queue

当被动建立连接的parent socket收到SYN数据包后，会产生一个child socket来服务该连接，放到parent socket的listen\_queue队列中

socket.accept\_queue

当接收到三次握手中的最后一个包（ACK）时，在listen\_queue中的child socket会放到accept\_queue中，等待应用程序读取(tcp\_sock\_accept)

Socket加入到accept\_queue中时，parent socket的accept\_backlog值加一，离开队列时该值减一，注意accept\_backlog < backlog

⑤建立连接

被动建立连接

申请占用一个端口号 （bind操作）

监听该端口号 （listen操作）

收到SYN数据包 -> TCP\_SYN\_RECV （accept操作）

回复ACK并发送SYN数据包

收到ACK数据包 -> TCP\_ESTABLISHED

主动建立连接

发送目的端口的SYN数据包 -> TCP\_SYN\_SENT （connect操作）

收到SYN 数据包（设置TCP\_ACK标志位）

回复ACK数据包 -> TCP\_ESTABLISHED

⑥断开连接

主动关闭

发送FIN包，进入TCP\_FIN\_WAIT\_1状态

收到FIN对应的ACK包，进入TCP\_FIN\_WAIT\_2状态

收到对方发送的FIN包，回复ACK，进入TCP\_TIME\_WAIT状态

等待2\*MSL时间，进入TCP\_CLOSED状态，连接结束

被动关闭

收到FIN包，回复相应ACK，进入TCP\_CLOSE\_WAIT状态

当自己没有待发送数据时，发送FIN包，进入TCP\_LAST\_ACK状态

收到FIN包对应的ACK，进入TCP\_CLOSED状态，连接结束

2、TCP数据包处理流程

①检查TCP校验和是否正确

②检查是否为RST包，如果是，直接结束连接

③检查是否为SYN包，如果是，进行建立连接管理

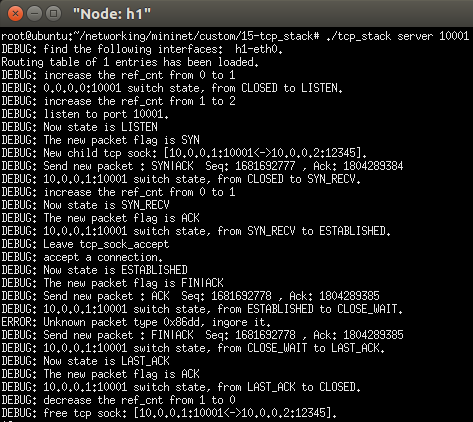
④检查ack字段，对方是否确认了新的数据

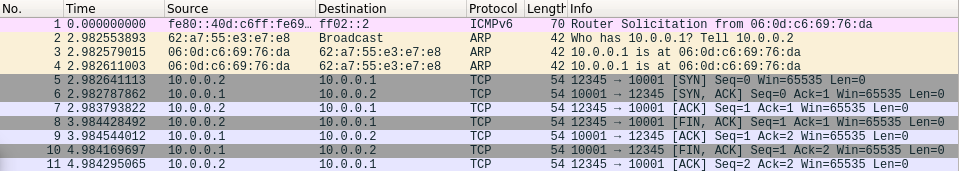
⑤本次实验中只有SYN和FIN包会确认新数据

⑥检查是否为FIN包，如果是，进行断开连接管理

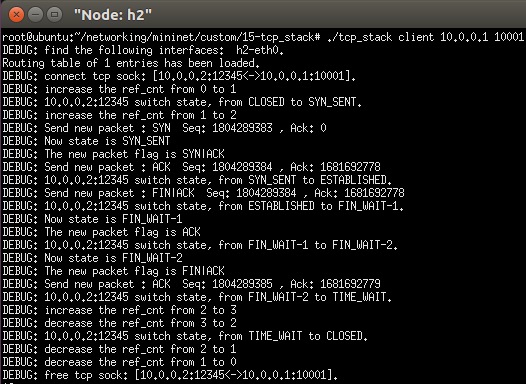
四、实验结果

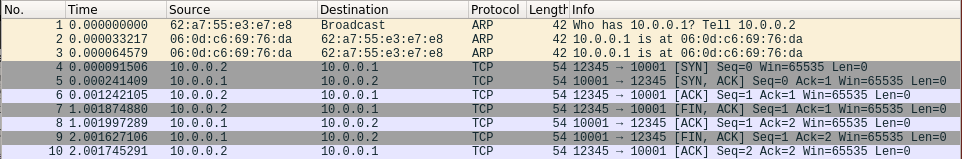
1、h1节点的连接过程





2、h2节点的连接过程





五、结果分析

1、连接过程中TCP状态

在server端，我们可以看到用于监听的socket首先从CLOSED状态到LISTEN状态，等收到来自client端的连接请求后，生成了child socket，并变成SYN\_RECV状态，三次连接握手后来到ESTABLISHED状态。在断开连接阶段，当收到client发送的FIN信号后变成CLOSE\_WAIT状态，等到主动关闭连接时变成LAST\_ACK状态，收到来自client端最后的ACK信号后，最终到达CLOSED状态，从而完成整个socket的建立和释放流程。

在client端，主动发送连接请求SYN后来到SYN\_SENT状态，收到来自server端的响应SYN信号后变成ESTABLISHED状态，并发送连接三次握手中的最后一个ACK信号。断开连接时，首先从client端发送FIN信号，也相应变成FIN\_WAIT-1状态，收到server端的ACK信号后来到FIN\_WAIT-2状态，等待到来自server端的释放请求FIN信号后，回复ACK信号并变成TIME\_WAIT状态，等待2\*MSL时间后最终变成CLOSED状态。

2、发送的包中Seq和Ack序号正确性

从wireshark结果观察发送的TCP包中的Seq和Ack序号的正确性，首先client端发送SYN信号请求连接，此时初始Seq为0（此处为相对值），接受server端收到该信号后回复SYN|ACK信号，由于上个SYN消耗一个序号值，因此此时Ack的值为1，而server端的Seq初始值为0（此处也为相对值），client端收到SYN|ACK信号后，发送三次握手连接中的最后一个ACK信号，由于SYN|ACK信号也消耗一个序号值，因此发送的ACK信号中Ack为1，Seq也为1，至此，TCP连接完成。

在断开连接的过程中，首先由client端断开连接，发送FIN|ACK包，由于上一次回复的ACK包不消耗序号值，因此该包中的Ack和Seq仍然均为1。接着server端回复ACK信号，由于收到的FIN|ACK需要消耗一个序号，因此ACK包中的Ack值为2，而Seq值仍然为1。此后server端再发送其断开连接的FIN|ACK包，由于上次该端发送的ACK包并不消耗序号值，因此此时的Seq和Ack值仍然分别为1和2。最后client端收到server端的FIN|ACK信号后回复ACK包，由于FIN|ACK包需要消耗一个信号值，因此回复的ACK包中的Ack值为2，而上次client端发送的FIN|ACK包需要消耗一个信号值，因此Seq值也为2。

六、实验总结

此次实验可以说是到目前为止最复杂的一次实验，其复杂主要体现在逻辑顺序的复杂性上，从代码量上来看其实并不是特别大。

本次实验总体来说可以分成两个部分，一部分是完成TCP连接的管理，包括TCP连接的建立和释放，这部分的处理在tcp\_sock.c中，相关函数包括tcp\_sock\_bind、tcp\_sock\_listen、tcp\_sock\_connect、tcp\_sock\_accept和tcp\_sock\_close，其中有部分函数已经实现。另一部分是完成对TCP数据包的处理流程，包括socket处于不同阶段时，对于收到的不同TCP数据包的处理，该部分位于tcp\_in.c，处理过程中最关键的函数为tcp\_process。

在完成代码前，我先反复阅读了现有代码及课件，争取弄清楚所有需要完成的函数间的关系，尤其是其中一些涉及到先sleep\_on后wake\_up的操作。在确定充分理解了实验内容并明确了每个函数的任务后，才开始编写代码。

在阅读现有代码的过程中，我重点关注到了socket结构中的ref\_cnt变量，这个变量表示的含义是socket结果被关联的次数，与其有关的结构主要有listen\_queue、listen\_table、establish\_table、hash\_table、timer\_list，当该变量为0时，表示socket无效，需要删除，因此在管理socket中需要释放socket时需要特别留意该变量。还有另一处需要关注的是rcv\_nxt、snd\_una和snd\_nxt这三个变量的更新，其中snd\_nxt是在每次发送TCP包时会自动更新，但rcv\_nxt和snd\_una这两个变量就需要在每次收到TCP包时相应的进行更新。

另外在此次实验的初始代码中还有一处不太理解的地方是在函数tcp\_sock\_close中，如果当前的TCP状态为TCP\_LISTEN，会先执行tcp\_sock\_clear\_listen\_queue将parent socket的listen\_queue中管理的socket进行释放，但是为什么accept\_queue中的socket不需要进行处理呢？

【课后更新】课上又和老师交流了一下，也解决了上面提出的问题，首先是ref\_cnt变量在listen\_queue的处理时不需要更改，其次在TCP\_LISTEN的状态时执行tcp\_sock\_close时，accept\_queue中的表项也需要被释放。此外又发现了在tcp\_scan\_timer\_list函数中，当等待的2\*MSL时间到后进行socket释放时，ref\_cnt的计数上也出现了问题，这次没有把引用释放完全，导致最后其实没有真正实现socket的释放，此处将原有代码中的free\_tcp\_sock(tsk);修改为tcp\_unhash(tsk);和tcp\_bind\_unhash(tsk);即可。