计算机网络实验13-17总结

chuan-325

2021年7月15日

实验13：网络传输机制实验一

实验13：网络传输机制实验一

- 实验内容
  - 在给定框架中实现最简单的 TCP 连接
    - 流程：建立连接 → 关闭连接
  - 使用 wireshark 抓包验证

实验13：网络传输机制实验一

- 实验设计
  - TCP 状态机实现
  - TCP sock 以及相关 table 的维护
  - TCP 计时器

实验13：网络传输机制实验一

- 实验设计
  - TCP 状态机实现
    - Server: Closed - Listen - SYN_Recv - Established - Last_Ack - Closed
    - Client: Closed - SYN_sent - Established - FIN_wait-1 – FIN_wait-2 - Time_wait - Closed
  - TCP sock 以及相关 table 的维护
  - TCP 计时器

实验13：网络传输机制实验一

- 实验设计
  - TCP sock 以及相关 table 的维护
    - sock 绑定信息元组（ip和端口）
    - listen table & established table，bind table
  - TCP 计时器

实验13：网络传输机制实验一

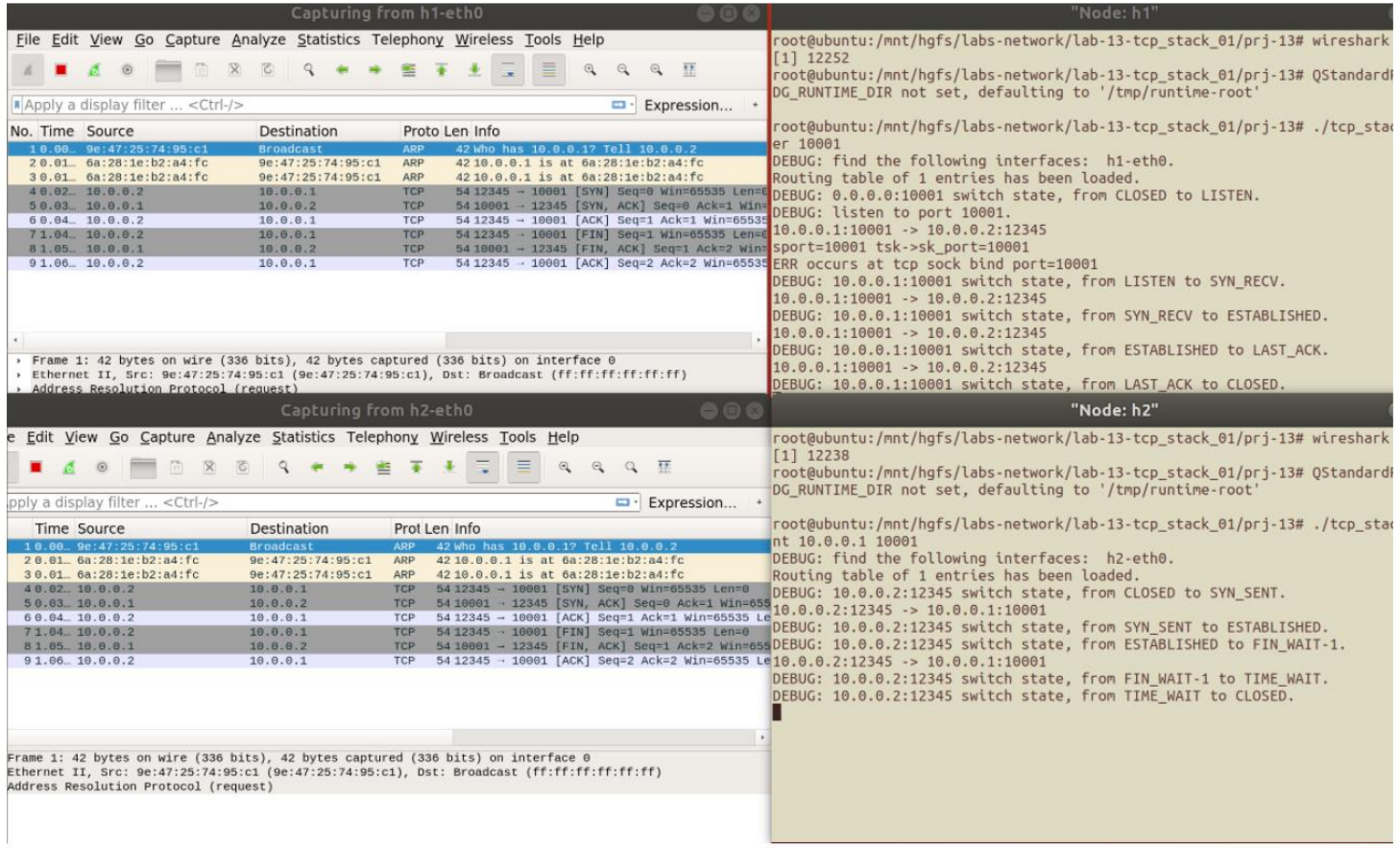
- 实验设计
  - TCP 计时器
    - 提供设置timewait-timer的方法 tcp_set_timewait_timer
    - 定时扫描定时器列表，对 timeout 的定时器进行老化
      - 删除计时器
      - 关闭相应 sock 的 TCP 连接，如果 sock无 parent 则从 bind table 中释放
      - 将 sock 从其所属的 table (established table / listen table) 中释放

实验13：网络传输机制实验一

结果分析与反思

将 h1 作为 TCP server, h2 作为 TCP client, 分别执行:

```
h1 # ./tcp_stack server 10001
h2 # ./tcp_stack client 10.0.0.1 10001
```

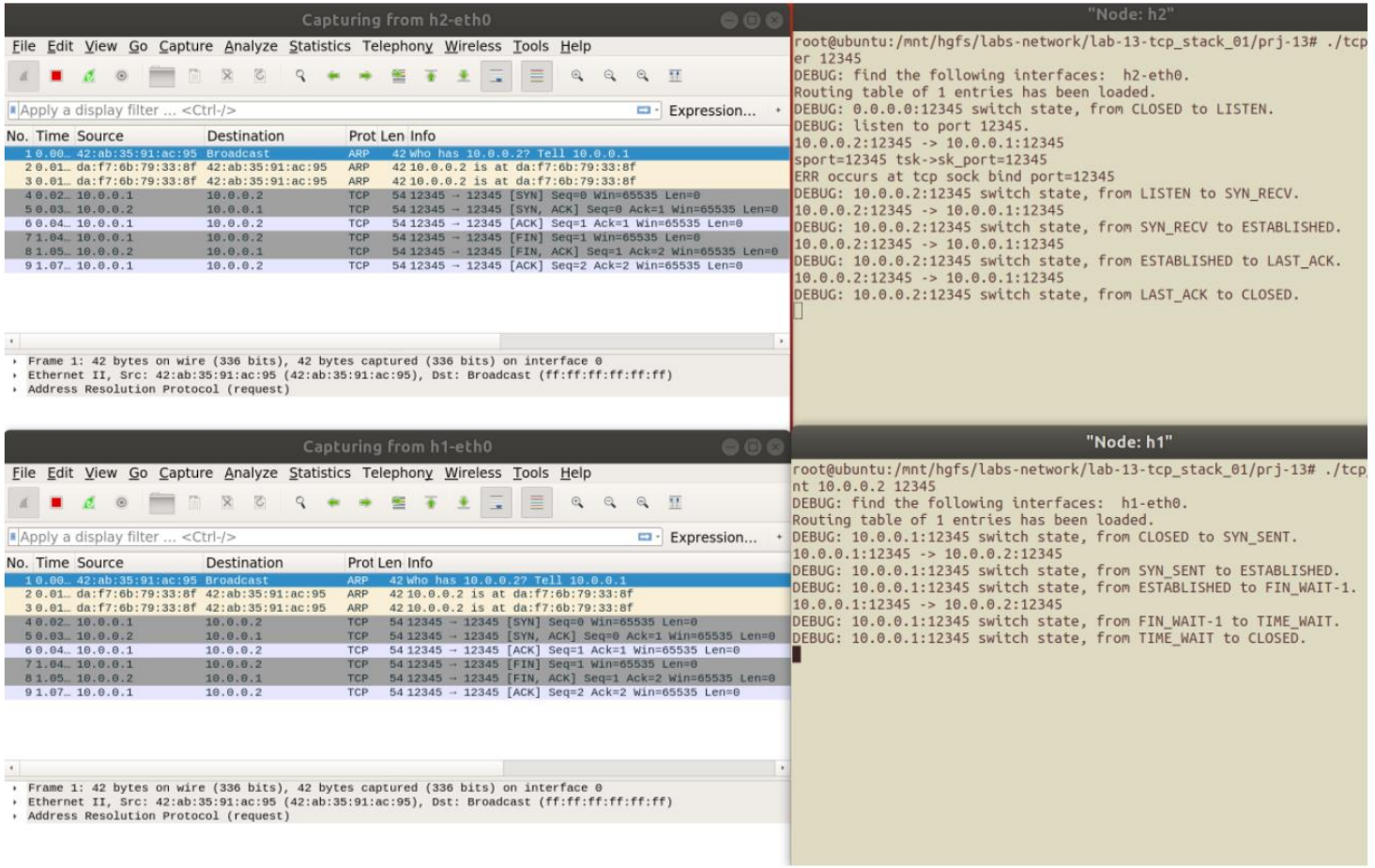


实验13：网络传输机制实验一

结果分析与反思

将 h2 作为 TCP server, h1 作为 TCP client, 分别执行:

```
h2 # ./tcp_stack server 12345
h1 # ./tcp_stack client 10.0.0.2 12345
```



实验13：网络传输机制实验一

调试中出现的问题

- 问题：
  - 误报 port in use ，实验无法正常进行
- 原因：
  - tcp_process 的 switch-case 中，收到 TCP_SYN 消息时为新 sock 赋值的过程有误
- 解决：
  - 追溯到使用 bind_table 的所有场景后定位 bug 成功，修改赋值过程后程序通过测试

实验15：网络传输机制实验二

实验15：网络传输机制实验二

- 实验内容
  - 补充 tcp_sock_read 和 tcp_sock_write ，以实现最简单的数据传输；
  - 调整框架的其他部分，使其适配相关新需求和通用的 API ；
  - 运行指定的网络拓扑，验证程序实现的正确性。

实验15：网络传输机制实验二

实验设计

- tcp_sock 读写函数
- TCP 状态机的更新
- 文件收发

实验15：网络传输机制实验二

实验设计

- tcp_sock 读写函数
  - tcp_sock_write(struct tcp_sock *tsk, char *buf, int len)
  - tcp_sock_read(struct tcp_sock *tsk, char *buf, int len)
- TCP 状态机的更新
- 文件收发

实验15：网络传输机制实验二

实验设计

- TCP 状态机的更新（增加的处理）
  - 在 ESTABLISHED 状态下收到 ACK
    - 唤醒 wait_send
  - 在 ESTABLISHED 状态下收到 PSH & ACK
    - 根据 rcv_buf 和 payload 写 ring buffer，唤醒 wait_recv
    - 发送 ACK，唤醒 wait_send
- 文件收发

实验15：网络传输机制实验二

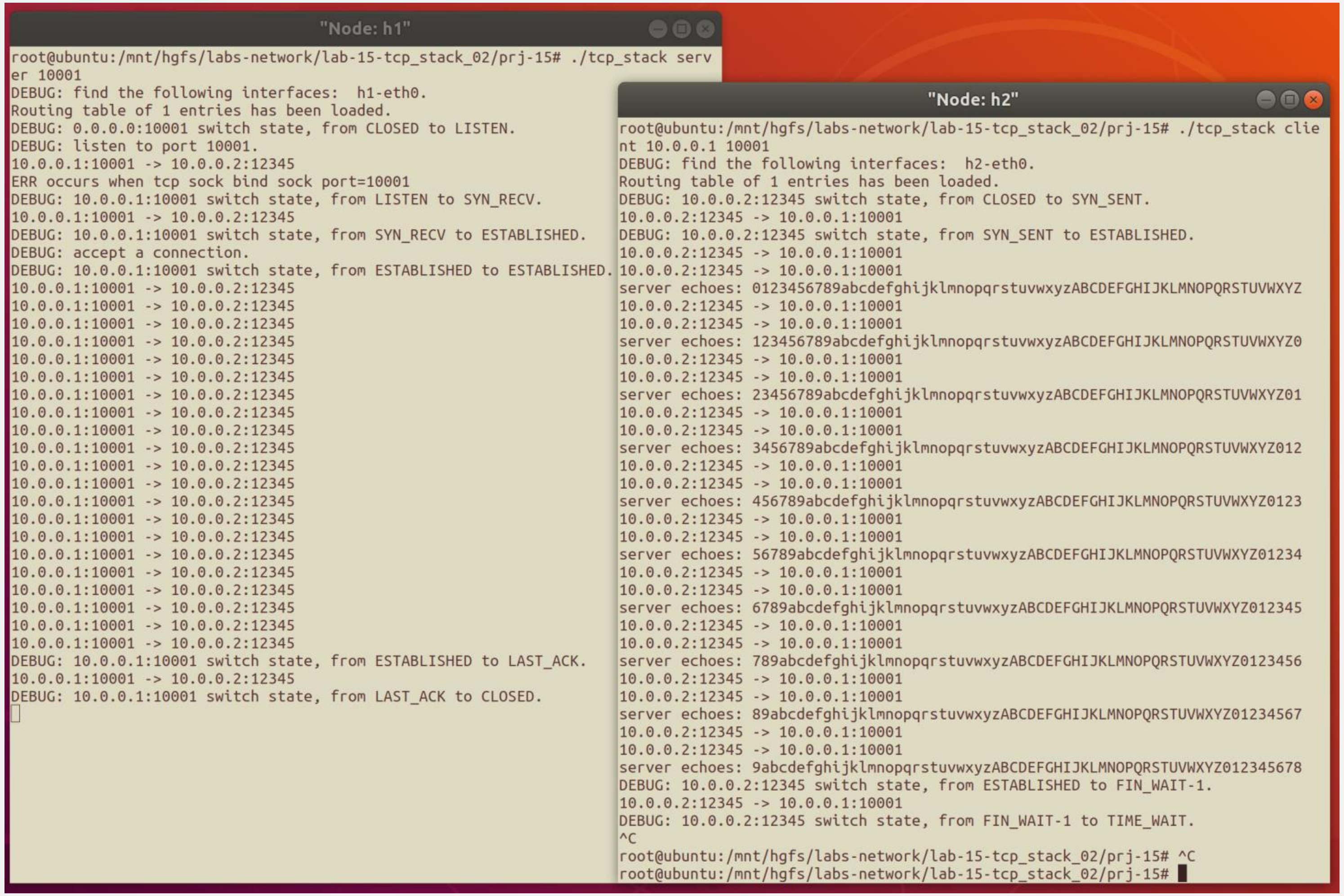
实验设计

- 文件收发
  - 服务器：接收文件
    - 核心逻辑：在文件读取完毕之前/出错之前不断读入 ring buffer 中的数据，并写入本地文件
  - 客户端：发送文件
    - 核心逻辑：获取文件流指针，不断读取文件并分片进行发送，直到文件指针指向 EOF 为止

实验15：网络传输机制
实验二

结果分析与反思

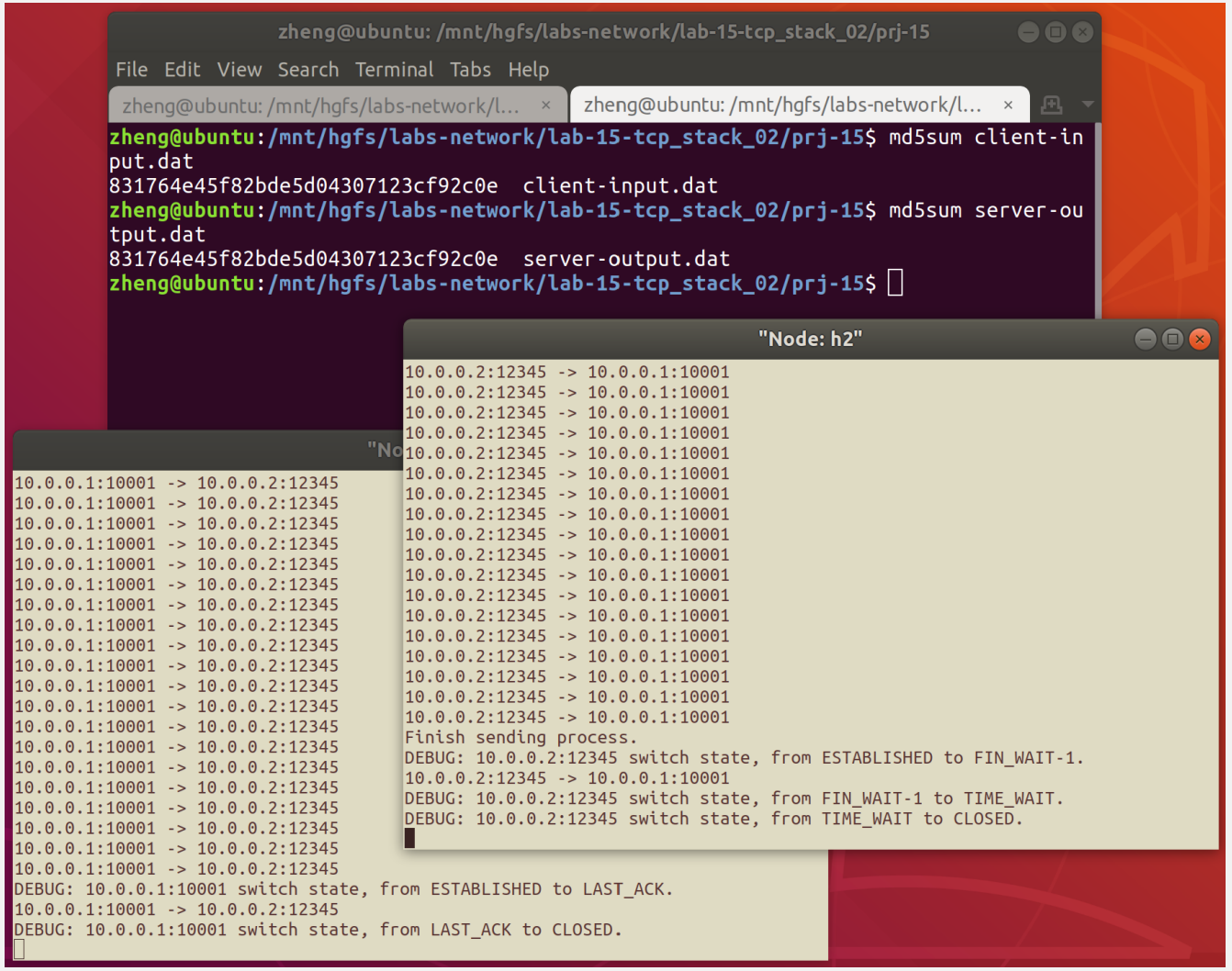
- Basic：字符串传输



实验15：网络传输机制实验二

结果分析与反思

- Normal：文件传输



实验15：网络传输机制实验二

调试中出现的问题

- 问题：
  - 与 Python 脚本客户端协同测试时，发现程序服务端无法回到 CLOSED 状态
- 原因：
  - 客户端在发送完数据之后会同时发送 ACK 和 FIN
  - 当时的 TCP 状态机无法让服务器在 ESTABLISHED 状态下同时完成最后一个 ACK 和 FIN 的处理
- 解决：
  - 调整 TCP 状态机的处理逻辑

实验16：网络传输机制实验三

实验16：网络传输机制实验三

- 实验内容
  - 增加**超时重传**的支持，使节点在有丢包网络中也能建立连接并正确传输数据：
    - 维护 sock 中的重传计时器；
    - 在计时器的基础上实现超时重传机制
  - 使用有丢包网络拓扑脚本进行测试，验证该实现的正确性。

实验16：网络传输机制实验三

实验设计

- 超时重传机制
- 发送队列维护
- 接收队列维护

实验16：网络传输机制实验三

实验设计

- 超时重传机制
  - retrains timer: 添加设置、释放的方法
  - scan/trigger：超时则需作相应处理
    - 重传超过三次：RST
    - 重传三次以内：重设计时器时间（翻倍）
- 发送队列维护
- 接收队列维护

实验16：网络传输机制实验三

实验设计

- 发送队列维护
  - **入队**：未确认（未收到相应 ACK）的数据/SYN/FIN 包
  - **出队**：收到 ACK，移出 send buffer 中相应被确认的项
  - **应用（重传）**：在 retrains timer 被触发时，重传 send buffer 中的第一个数据包
- 接收队列维护

实验16：网络传输机制实验三

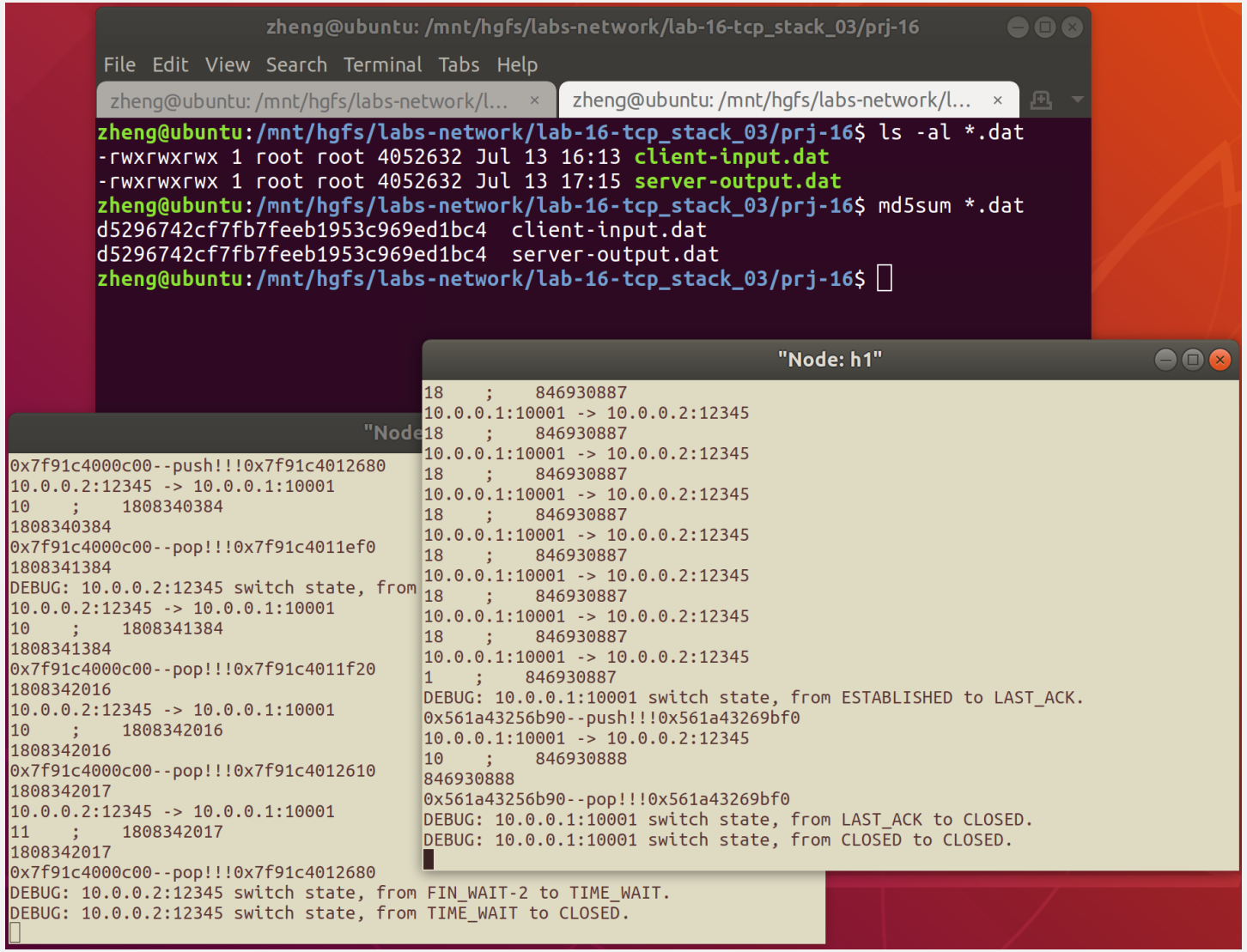
实验设计

- 接收队列维护
  - 维护两个队列：
    - 连续收到的数据：rcv_ring_buffer，app 直接读取
    - 不连续的数据：rcv_ofo_buffer，等待移动
  - 发送方驱动传输，接收方只负责对收到的数据包回复 ACK
  - 收到不连续的数据包：
    - 存入 rcv_ofo_buffer
    - 如果队列中含连续数据，则将其移入 rcv_ring_buffer

实验16：网络传输机制实验三

结果分析与反思

- 有丢包网络 (loss rate=2%) 下的测试结果：



实验16：网络传输机制实验三

实验中的反思

- 问题：
  - 有一部分 TCP 状态机的逻辑重复较多
- 原因：
  - 设计之初，将 TCP 状态机的结构定为：

```
switch (cb->flags) {
        case xxx:
            switch (tsk->state) {
                case yy:
                …
             }
        …
}
```

- 解决：
  - 将如上的两层 switch case 内外对调，使之更符合相关的设计/控制逻辑

实验17：网络传输机制实验四

实验17：网络传输机制实验四

- 实验内容
  - TCP拥塞控制状态迁移
  - TCP拥塞控制机制
    - 数据包发送
    - 拥塞窗口调整
    - 重传数据包
  - TCP拥塞控制机制实现

实验17：网络传输机制实验四

实验设计

- 拥塞控制状态机
- 拥塞控制

实验17：网络传输机制实验四

实验设计

- 拥塞控制状态机 [init as OPEN]
  - 收到新 ACK
    - state=DISORDER:
      - 跳回 OPEN
    - state=LOSS:
      - 该 ACK > LOSS 下的 snd_nxt, 则跳回 OPEN
    - state=RECOVERY:
      - 该 ACK > RECOVERY下的 snd_nxt, 则跳回 OPEN
  - 收到 DUP ACK
  - 拥塞控制

实验17：网络传输机制实验四

实验设计

- 拥塞控制状态机 [init as OPEN]
  - 收到新 ACK
  - 收到 DUP ACK
    - state=OPEN:
      - dupacks++
      - state => DISORDER
    - state=LOSS:
      - dupacks++
      - 当 dupacks == 3 则 state => RECOCERY
- 拥塞控制

THANKS