实验 1.3 TCP 与拥塞控制

一． 实验目的

TCP(Transmission Control Protocol 传输控制协议) 是一种面向连接的、可靠的、基于字节流的传输层通信协议。本实验通过运用 Wireshark 对网络活动进行分析，观察 TCP 协议报文，分析通信时序，理解 TCP 的工作过程，掌握 TCP 工作原理与实现；学会运用 Wireshark 分析 TCP 连接管理、流量控制和拥塞控制的过程，发现 TCP 的性能问题。

二． 实验内容

启动 Wireshark，捕捉网络活动中的 TCP 报文并按要求分析。

1. 连接管理：观察正常 TCP 连接中的三次握手与四次挥手报文，绘制出时序图，并标出双方 TCP 状态变化。
2. 异常情况分析：观察分析 TCP 连接建立过程的异常 (例如，尝试连接未存活的主机或未监听端口，客户端发送了第一个 SYN 连接请求而服务端无响应)；观察 SYN 洪泛影响；观察分析 TCP 通信过程中的各类异常报文 (例如数据超时、乱序)，了解其触发机制与含义。
3. 流量控制 (进阶)：运行一组 TCP 连接客户端/服务器程序，制造收发不平衡场景，观察收发报文中通告窗口的变化，分析与窗口机制相关的类型报文，了解滑动窗口工作原理。
4. 拥塞控制 (进阶)：改变带宽、时延、丢包率等网络参数，观察大文件传输过程，分析识别 TCP 的慢启动、拥塞避免、快速恢复等拥塞控制阶段；在构建的网络试验环境下运行 iperf3 进行网络性能测试，比较不同拥塞控制策略的性能表现。

三． 实验原理、方法和手段

3.1 TCP 协议

作为 TCP/IP 协议簇中的骨干，TCP 协议基于“尽力而为”的网络层为应用层提供可靠的进程间通信服务，具体地说是可靠的全双工的端对端字节流传输服务。在 TCP 的协议传输单元中 (TCP 报文段，TCP Segment)，收发双方以字节为单位使用序号 (Sequence

Number) 明确收发的数据，使用 ACK 反馈 (Acknowledgment) 机制，实现端对端的可靠传输控制。

3.1.1 TCP 报文段 (Segment)

TCP 报文段结构如图1.3–1所示，采用 20 字节的报文段头以及不超过 40 字节的可选项。

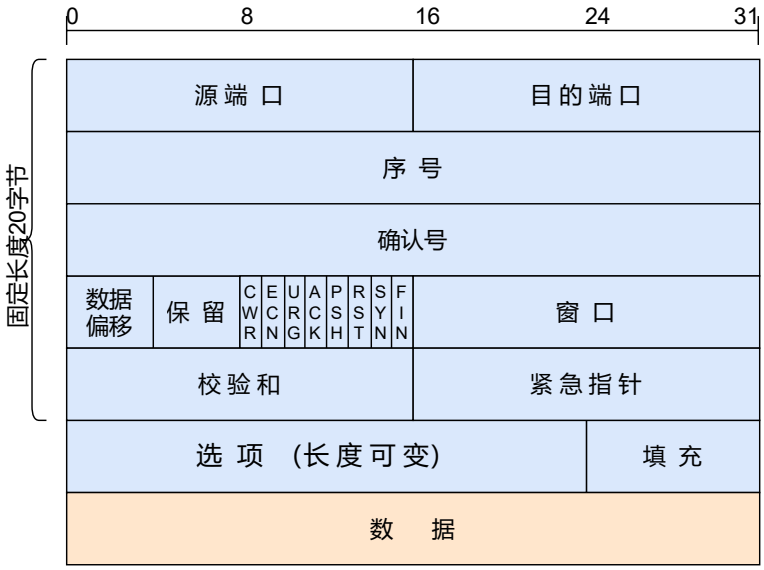

图 1.3–1 TCP 报文段结构示意图

主要字段如下：

1. **源端口号 (Source Port)**：16 位的源端口，与源 IP 地址一起标识发送该 TCP 报文段的通信进程。端口号范围 0-65535。
2. **目的端口号 (Destionation Port)**：16 位目的端口，与目的 IP 地址一起标识接收该 TCP 报文段的通信进程。
3. **序号 (Sequence Number)**：该 TCP 报文段中第一个数据字节的序号，占 4 个字节。在 TCP 连接建立时，通常生成一个随机数作为字节序号的初始值 (ISN)。
4. **确认号 (Acknowledgement Number)**：表示期望收到对方下一个报文段的字节序号，占 4 个字节。
5. **标志位 (TCP Flags)**：
   (a) 确认 ACK(Acknowledgement)：置 1 表示确认号字段有效。

(b) 推送 PSH(Push)：置 1 表示该报文段优先级高，接收方 TCP 应尽快推送给接收应用程序。

(c) 复位 RST(Reset)：置 1 表示需要释放 TCP 连接并重新建立连接。一般称带 RST 标志的 TCP 报文段为“复位报文段”。

(d) 同步 SYN(Synchronization)：置 1 表示这是 TCP 请求连接报文段。一般称带 SYN 标志的 TCP 报文段为“同步报文段”。

(e) 终止 FIN(Finish)：置 1 表示发送方的数据已经发送完毕，并要求释放 TCP 连接。

6. **窗口大小 (Window)**：表示接收缓存大小。最早 TCP 协议首部只设置了 16 位的窗口大小，允许的最大缓存大小不超过 64KB；而 RFC1323 打破此限定，设置了 TCP 窗口缩放因子 (Window size scaling factor)，使窗口大小等于二者的乘积。

3.1.2 TCP 连接管理

为维护一个可靠的端对端传输，TCP 设计实现了完整的连接管理，重点是三次握手的连接建立和四次挥手的连接释放过程，如图1.3–2。

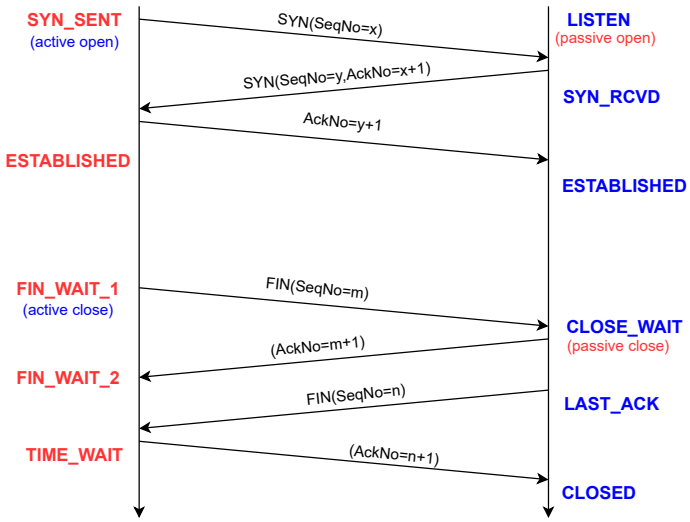

图 1.3–2 TCP 连接示意时序

建立过程：TCP 是面向连接的，数据传输之前必须在双方之间建立一条连接，并通过三次握手过程完成。其主要目的是，同步连接双方的序号和确认号，并交换 TCP 窗口大小等控制信息。一般地，客户端主动向服务器端发起连接请求，具体过程如下：

1. 第一次握手：建立连接。客户端发送连接请求报文段，将 SYN 位置为 1，Sequence Number 为 x；然后，客户端进入 SYN_SEND 状态，等待服务器的确认；
2. 第二次握手：服务器收到 SYN 报文段。服务器收到客户端的 SYN 报文段，需要对这个 SYN

报文段进行确认，设置确认号为 x+1；同时，将 SYN 位置为 1，序号为 y。服务器端将上述信息放入 SYN+ACK 报文段中，一起发送给客户端，此时服务器进入 SYN_RECV 状态；

3. 第三次握手：客户端收到服务器的 SYN+ACK 报文段之后，将确认号设置为 y+1，向服务器发送 ACK 报文段。这个报文段发送完毕以后，客户端和服务器端都进入 ESTABLISHED 状态，完成 TCP 三次握手。

释放连接 (四次挥手)：客户端没有新数据要发送时，就会释放 TCP 连接，发送一个报文段 (FIN=1，LEN=0)，进入 FIN-WAIT1 状态。服务器端收到后，返回客户端一个 ACK 报文段，进入 CLOSE-WAIT 状态。此时客户端进入 FIN-WAIT2 状态，不能再发送信息，但仍可接收信息。服务器端完成数据发送之后，也发出 FIN=1 的报文请求释放连接，并进入 LAST-ACK 状态，直至客户端返回确认（第四次挥手）才关闭 TCP 连接。客户端收到服务器的 FIN 报文后，则进入 TIME_WAIT 状态，并等待 2MSL(最大存活时间- Maximum Segment Lifetime) 时间之后，完全关闭 TCP 连接。(注：释放连接也可由服务器端先发起)

3.1.3 TCP 流量控制

流量控制的目的是让发送方的数据发送速率不要过快，以便接收方能及时接收。TCP 利用滑动窗口 (Sliding Window) 机制实施流量控制，其基本原理是用 TCP 报文段中的窗口大小字段来控制数据发送速率，即发送方的发送窗口应小于接收方回应报文中的通告窗口值。为了提高信道利用率，TCP 采用了连续 ARQ(Automatic Repeat reQuest)，TCP 两端都可以连续发出若干个分组然后等待确认，也都设有发送/接收窗口和发送/接收缓存。其中发送窗口可以用 3 个指针表示，而且发送窗口随着 TCP 报文段中窗口 (Window) 字段的数值动态变化。该窗口字段是告知对方自己还能接收多少字节的数据。发送方只发送序号在发送窗口之内的数据，避免发送过快，从而实现流量控制。具体的缓存、窗口和指针变化过程请参阅教材相关内容。

若接收方没有空余缓存，就会发送零窗口大小 (Window=0) 的报文段，即要求发送端停止数据发送。然而，当接收方释放出足够缓存后，发送方往往无法及时恢复数据发送，这就产生了死锁问题。为解决零窗口 (Zero-Window) 问题，TCP 为每一个连接设置一个持续计时器 (persistence timer)。只要 TCP 的一方收到对方的零窗口报文段，就启动该计时器，并周期性的发送一个长度为 1 字节的探测报文段 (周期亦会增加)。收到探测报文段的一方，在返回 ACK 报文段时，将使用 Window 字段发送其最新缓存大小；若最新缓存足够大，数据发送恢复，解决死锁问题。

3.1.4 TCP 拥塞控制

计算机网络中的带宽、节点 (路由器和主机) 中的缓存和处理机等，都属于网络的资源，在某段

时间，若对网络中某一资源的需求超出该资源所能提供的可用部分，网络的性能就会变坏，这种情况就叫做拥塞。TCP 拥塞控制的目的是避免过多的数据注入网络引发路由器或链路资源过载。拥塞控制是一个全局性的系统化的工程，将报文段丢失视作网络拥塞发生的信号，通过调整拥塞控制窗口 (cwnd - Congestion Window) 和利用 ACK 反馈机制来控制数据发送速率。

TCP Tahoe 是 TCP 的最早版本，主要用三种算法去控制数据流和拥塞窗口：慢启动 (Slow Start)、拥塞避免 (Congestion Avoidance)、快重传 (Fast Retransmit)。TCP 事先并不知道网络资源的状况，首先采用慢启动算法开始发送数据：cwnd 初始值为 1 个 MSS(Maximum Segment Size)；每收到一个有效的 ACK 确认，cwnd+1，直至出现网络丢包超时或 cwnd 达到阀值 ssthresh（slow start thresh）。当 cwnd 等于或大于 ssthresh，采用拥塞避免算法，由 AIMD(加性增、积性减) 策略控制发送速率。若 TCP 报文段发送后 RTO (Retransmission Time Out) 时间仍未得到确认或收到三个确认号重复的 ACK 报文段，则启用快重传算法，并将 cwnd 重置为 1，ssthresh 减半。

TCP 拥塞控制算法一直处在不断的改进之中，围绕对网络环境因素感知和拥塞避免的控制，涌现出新的策略算法。TCP Reno 继承 Tahoe 的三个算法并增加了快速恢复 (Fast Recovery) 算法。收到三个重复的 ACK 后，Reno 会把当前的 ssthresh 的值设置为当前 cwnd 的一半，并将 cwnd 更新为 ssthresh+3MSS；然后每收到一个重复 ACK 则 cwnd+1，直至收到新确认号的 ACK 则将 cwnd 更新为 ssthresh。TCP NewReno 则进一步改进了快速恢复算法。

随着网络速度增长，传统拥塞控制算法的 cwnd 增长速度影响了 TCP 的性能，CUBIC[4] 应运而生。CUBIC 的关键特征是：cwnd 窗口的增长依赖两次丢包的时间。2016 年，谷歌提出了 BBR[5] 拥塞控制算法，它不再基于丢包感知来调整 cwnd，而是利用估算的带宽和延迟直接推测拥塞程度进而确定发送窗口。

3.2 实验方法和手段

两台实验机本地相互连接 (如图1.3–3)，在实验机中仿真不同的网络条件，以便观察 TCP 的各种控制现象。方案一使用 VMware Player 运行两台虚拟机，并通过“虚拟机设置-> 硬件-> 网络适配器-> 高级”（如图1.3–4）设置虚拟机的网卡传入/传出带宽、数据包丢失率、延迟等。方案二直接在两台 PC 机上操作，使用 tc 进行流量控制场景仿真，使用 wondershaper 对网卡进行限速。本实验需要使用的命令和工具，如表1.3–1所列。常用的 Linux 命令还包括：echo、cat、sysctl、ping、ftp 等。

四． 实验条件

1. 硬件：处于同一局域网的两台 PC 机 (可使用虚拟机也可使用物理机)。

[4]CUBIC: A New TCP-Friendly High-Speed TCP Variant

[5]BBR: Congestion-Based Congestion Control

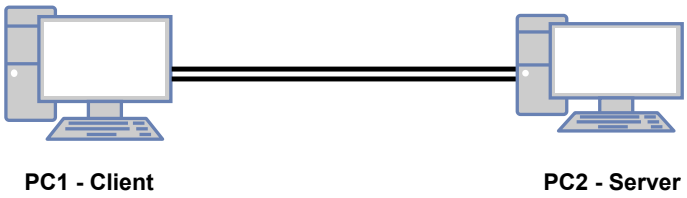

图 1.3–3 实验拓扑图

图 1.3–4 虚拟机网络适配器高级设置

2. 软件：Ubuntu 系统 (18.04 版)，预装 wireshark、curl、vsftp、netwox、telnet、nmap 和 iperf3。
3. 环境准备：分别以 PC1、PC2 作为 TCP 的客户端与服务端；启动两台实验机后，可使用 ping 进行连接性测试，也可使用 nmap 扫一下对方打开的端口，确保实验环境正常 (指导书中 PC2 的 IP 为 192.168.100.144，实验中应替换为实际的 IP) 。

参考资料：

- TCP 协议 RFC
- 计算机网络--自顶向下方法：Wireshark 实验指导书–TCP
- Congestion Control in Linux TCP

五． 实验步骤

表 1.3–1 主要工具及命令列表

命令	作用	参考
ifconfig	配置网络	https://man.linuxde.net/ifconfig
nmap	网络扫描	https://nmap.org/man/zh/index.html
curl	文本浏览器	https://man.linuxde.net/curl
wget	下载 Web 文件	wget <IP>/<PathAndFileName>
tc	流量控制	tc 命令手册
iptables	防火墙配置	https://man.linuxde.net/iptables
netwox	网络工具	https://sourceforge.net/projects/ntwox/
ss	Socket 状态	ss -atn
netstat	显示网络状态	netstat -atn
wondershaper	网卡限速工具	wondershaper [网 口] [下 载 速 率] [上 行 速 率] wondershaper clear [网口]
iperf3	网络性能分析	https://iperf.fr/

5.1 TCP 正常连接观察

1. 利用 python 自带的 SimpleHTTPServer 模块，在 PC2 上启动一个简易的 web 服务器。终端上运行 `echo "TCP lab test" > index.html` 创建 index.html 文件为测试站首页，运行 `sudo python -m SimpleHTTPServer 80` 启动一个简易 web 服务器；打开新终端，键入 `ss -tln` 查看当前主机打开的 TCP 连接，确认 80 端口处理监听状态。
2. 在 PC1 上打开一个终端，键入 `sudo wireshark` 启动抓包软件；再打开一个新终端，键入 curl <PC2 的 IP> ；停止抓包，在 wireshark 过滤出 TCP 类型报文。观察首个 TCP 报文头，并分析各段值代表的意义。如果想要关闭相对序号/确认号，可以选择 Wireshark 菜单栏中的 Edit→Preference→protocols→TCP，去掉 Relative sequence number 勾选项。使用 Wireshark 内置的绘制流功能，选择菜单栏中的 Statistics→Flow Graph，Flow Type 选择 TCP flows 可以直观地显示 TCP 序号和确认号是如何工作的。
3. 观察 TCP 三次握手与四次挥手报文，注意报文收发过程中，双方 TCP 状态的变化。以本次捕获的报文为依据，分别画出本次 TCP 连接三次握手与四次挥手的时序图，结合 TCP 状态机，在双方各阶段标出对应的 TCP 状态。选择其中一个 TCP 报文，配合 Wireshark 截图，分析该报文 TCP 首部各字段的定义、值及其含义。

5.2 异常传输观察分析

1. 尝试连接未存活的主机或对未监听端口。
   (a) 用 curl 访问一个不存在的主机 IP，抓包观察共发送了几次 SYN 报文。根据每次时间间隔变化，估算 RTO（重传超时）。
   (b) 查看 Linux 主机的系统的 TCP 参数 SYN 重传设定：
   `cat /proc/sys/net/ipv4/tcp_syn_retries`
   (c) 更改 SYN 重传次数为 3：
   `echo "3" > /proc/sys/net/ipv4/tcp_syn_retries`
   (d) 再次 curl 访问，观察抓包内容。
   (e) 关闭服务器端的 SimpleHTTPServer(ctrl+C 中断，或关闭所在终端)，客户端 curl 访问服务器 80 端口，观察应答报文。
   (f) 运行 `nmap -sS <PC2 的 IP>` 扫描服务器，并抓包。
   (g) 在报告中总结以上观察结果，解释 SYN 扫描原理。
2. 客户端发送了第一个 SYN 连接请求，服务器无响应的情景。
   (a) 服务器开启 telnet 或 ssh 服务，客户端先尝试连接服务器，连接成功后，在双方键入 ss -tan 查看所有 TCP 连接状态。我们看到的 TCP 连接建立过程同 1 中的 HTTP 访问类似。在客户端，利用 iptables 拦截服务器回应的 SYN ACK 包，命令如下：
   `sudo iptables -I INPUT -s 192.168.100.144 -p tcp -m tcp -tcp-flags ALL SYN, ACK -j DROP`
   (b) 再次尝试连接并启动 wireshark 抓包，并在双方多次用 ss -tan 观察 TCP 状态。
   (c) 观察 TCP 的状态变化，分析 wireshark 捕获的 TCP 异常报文。
   (d) 服务端的 SYN-RECV 状态何时释放？
   (e) SYN ACK 重传了几次，时间间隔有何变化？
   (f) 参考 1 中的操作，在服务端修改 SYN ACK 重传次数 (tcp_synack_retries)，再次观察，此任务结束后清空防火墙规则 (iptables -F)。
3. SYN 洪泛。在服务器端 `sudo echo "0">/proc/sys/net/ipv4/tcp_syncookies` 禁用 syncookies，通过 `sudo sysctl -w net.ipv4.tcp_max_syn_backlog = 6` 指定所能接受 SYN 同步包的最大客户端数量为 6；在客户端运用 netwox 工具对服务器监听的端口产生大量 SYN 连接请求 (如 `sudo netwox 76 -i 192.168.100.144 -p 23`)，再使用正常的连接工具 (如 telnet) 尝试连接，观察交互情况 (特别是服务器端的 TCP 连接状态)，抓包分析，解释 SYN 洪泛攻击原理与对策。
4. 异常报文分析。在服务器端产生一个 100M 的大文件，利用 1 中的 web 服务器，在客户端上

用 wget 下载它。参考命令：

(a) 产生一个 100M 文件：

```
dd if=/dev/zero of=100M.file bs=1M count=100
```

(b) 模拟网络延迟、包重复、包乱序、包损坏：

```
tc qdisc add dev ens33 root netem delay 70ms 10ms 30% duplicate 1% reorder
5% 10% corrupt 0.1%
```

(调整上述命令中的数值，以达到期望效果；将此行命令的 add 改为 change/del 即修改/删除此规则。)

(c) 下载服务器上的大文件：`wget 192.168.100.144/100M.file`

抓包记录以上过程，分析黑色标签错误报文，结合 TCP 实现机制，分析这些报文产生的原因。此类报文也可以从现实网络行为捕获，请结合捕获的报文进行分析，报文附件随报告提交。包括但不限于以下几种类型报文：[Duplicate ACK]、[TCP Retransmission]、[Fast Retransmission]、[TCP Spurious Retransmission]、[TCP Out-Of-Order]、[TCP Previous segment not captured]。

5.3 流量控制

1. 编写一对简单的 TCP 连接程序，也可以直接运行指导书提供的 Python 程序 (源代码见本节最后的附件)。在客户端快速发送数据给服务端，而服务端则有意缓慢地接收数据，观察 TCP 如何用窗口大小值进行流量控制。虚拟机两端分别运行 `python3 server.py` 和 `python3 client.py`。
2. 捕捉两端通信报文数据，分析报文中的 Win 值变化，联系上下报文，解释为什么出现 [TCP Windows Full]、[TCP ZeroWindows]、[TCP Keep-Alive] 等和窗口大小相关的流量控制报文。捕获的原始报文存成附件随实验报告提交。

5.4 拥塞控制

1. 任一端限制网卡传入/传出带宽为 10Mbps 以下：使用虚拟机作为实验机，可在 VMWare Player 中的虚拟机设置 → 网络适配器 → 高级中设置；物理机可使用 wondershaper 命令进行限速。再启动应用 (可以是 http wget，也可以 ftp 下载/上传) 传输大文件观察。
2. Wireshark 捕捉全部传输过程数据，找出该网络活动的拥塞点，并结合 Analyze→Expert Information、Statistic→IO Graphs、Statistic→TCP Stream Graphs(如图1.3–5)，分析此传输过程中的慢启动、拥塞避免、快速恢复等阶段。
3. TCP 竞争观察：类似以上试验，我们在一个大文件传输过程中，迅速启动另一应用，双向进行大文件传输，观察两路（或两路以上）TCP 连接的速率变化。

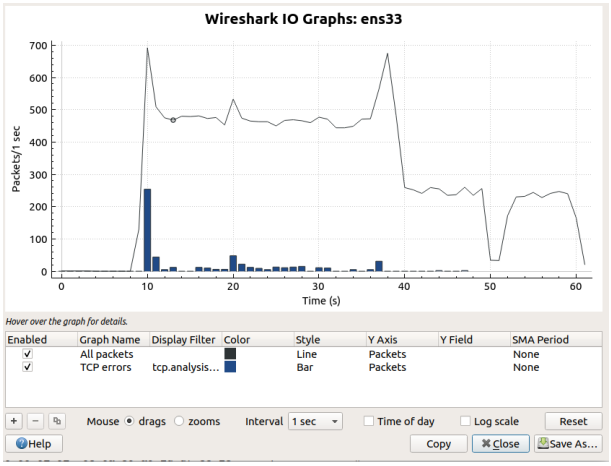

图 1.3–5 IO Graphs

4. TCP 拥塞控制算法比较：运用 iperf3 性能分析工具，设置 ubuntu 系统的 tcp_congestion_control 策略，分析比较不同网络环境下、不同拥塞控制算法的表现。参考如下命令（灵活组合使用）：
   (a) 可用策略：
   ```
   cat /proc/sys/net/ipv4/tcp_available_congestion_control
   ```
   (b) 修改策略：
   ```
   sysctl -w net.ipv4.tcp_congestion_control=bbr
   ```
   (c) iperf3 服务端启动：`iperf3 -s -p 5555`
   (d) iperf3 客户端启动：`iperf3 -c 192.168.100.144 -p 5555`
   (e) 设置链路带宽、时延、丢包率可以使用虚拟网卡高级属性配置，也可以参考前面使用 tc 命令配置。

六． 思考题

1. TCP 在不可靠的 IP 层上建立了可靠的端对端连接，如果要在不可靠的 UDP 上建立可靠的端对端传输系统，需要考虑哪些方面？
2. TCP 连接建立过程中，存在哪些等待队列？这些队列是否可能出现溢出状况？该如何避免？
3. 本次实验观察了 Linux 环境下的 TCP 实现，在 Windows、macOS 环境下，操作系统又是如何实现 TCP 的呢？类似 Linux TCP 参数，在不同系统环境下如何查看或设置，请尝试通过捕捉其通信过程、比较其实现异同。
4. 在 TCP 状态机 (图1.3–6) 中，有些状态停留时间较长，易观察到，有些状态很短暂不易观察到。试列出不易观察到的状态，并考虑观察到它们的可能方法。
5. TCP 是封装单元为 MSS，可是我们在捕捉过程中常发现远大于此值的 TCP 报文，为什么 TCP

可以提交如此大的报文呢？此类型的包远超出链路层的 MTU，它是如何被处理的呢？请从两端同时抓包观察比对。

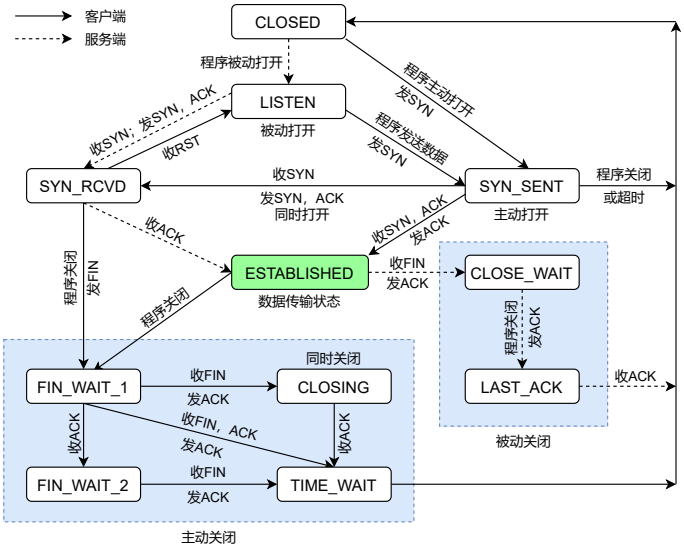

图 1.3–6 TCP 连接管理状态机

七． 注意事项及有关说明

1. Linux 上运行 wireshark 抓包需要 root 权限，请先在终端中通过 su 命令切换为 root 后，再键入 `wireshark &` 启动；或是打开个终端键入 `sudo wireshark`。`netwox/netwag` 同样需要 root 权限。
2. Ubuntu 中装有 python2 和 python3 两种类型版本，实验中用了 python2 启动简易 Web 服务器，附件中的程序用 Python3 运行。
3. 环境还原：前面操作的 iptables、tc 遗留规则可能会影响后面的操作效果，`iptables -list` 查看核对一下当前的规则，`iptables -F` 清空当前规则；同样，使用 `tc qdisc del dev eth0 root RULE` 清除网卡 eth0 队列规则；wondershaper 限速使用 clear 参数清除。使用虚拟机的快照功能是更原始、更彻底的还原方式。
4. 批量网络扫描是危害网络行为，仅在实验室环境下进行试验学习，不得用于运营网络。

八． 考核方法

完成本次实验，并提交一份实验报告和一组 Wireshark 数据存储文件。报告内容应当包括以下部分，相关的分析解释都对应有截图证明，并与数据存储文件吻合。

1. (20 分) 正确绘制出了三次握手报文与四次挥手报文 (须结合捕获的报文)，并正确标识出了各阶段 TCP 状态；
2. (20 分) 观察传输异常现象，并进行分析；
3. (20 分) 完成流量控制操作要求，结合上下分析报文窗口变化，解释说明相关的类型报文成因。
4. (20 分) 完成拥塞控制操作要求，成功从捕获的数据分析出一次完整 TCP 流的各个阶段（慢启动、拥塞控制、快速恢复）。
5. (10 分) 完成任 2 道思考题。
6. (10 分) 记录自己在本次实验中所遇到的问题，以及心得感悟。如果遇到异常情况，或者无法完成任务时，也请分析错误产生的原因。

九． 附件

为了方便进行实验，提供一份 Python3 套接字通信作为附件，具体代码如下：

- 服务端 server.py

```python
import socket
import time
def recv_data(sock,length):
    data=b''
    while len(data) < length:
        more = sock.recv(length - len(data))
        if not more:
            pass
        data += more
    return data
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# 设置接收缓冲区大小为 1024
s.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, 1024)
s.bind(('0.0.0.0', 9999))
s.listen(1)
print("Sever listening on 9999...")
try:
    sc, addr = s.accept()
    while True:
        rcvdata = recv_data(sc,16)
```

```
        time.sleep(0.01)
        pass
finally:
    sc.close()
    s.close()
```

- 客户端 client.py

```
import socket
import time
ip_port = ("192.168.100.144", 9999)
data = "0123456789\n"*100
c = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    c.connect(ip_port)
    print("Connected.")
    time.sleep(1)
    i=0
    while i<10:
        print("Send:"+data)
        c.send(data.encode())
        i=i+1
finally:
    print("Sent. Waiting....")
while True:
    time.sleep(0.001)
```