

# 合肥工业大学



学院：软件学院

班级：软件工程 **18-4** 班

学号：**2018214937**

姓名：刘铭源

计算机网络实验报告

课程名称	网络及其计算	班级	软件工程 18-4 班	实验日期	
姓名	刘铭源	学号	2018214937		
实验名称	TCP 实验				
实验目的及要求	掌握 TCP 协议的原理，深入理解 TCP 协议中的连接管理、可靠传输机制、流量控制机制和拥塞控制				
实验环境	Wireshark				
实验内容	(1) TCP 协议基础 传输层源地址结构 传输层目的地址结构 (2) 分析 TCP 连接管理的机制 建立连接机制 释放连接机制				

容	<p>连接过程中的异常处理</p> <p>(3) 分析 TCP 可靠传输原理</p> <p>确认机制</p> <p>重传机制</p> <p>分析其传输模型</p> <p>(4) 分析 TCP 的流量控制原理</p> <p>流量控制机制</p> <p>零窗口处理机制</p> <p>小窗口、傻瓜窗口问题</p> <p>(5) 分析 TCP 的拥塞控制原理</p>
实验步骤	<p>1. TCP 协议基础</p> <p>1.1 分析传输层源地址结构</p> <p>1.2 传输层目的地址结构：16 位目的地址端口号</p> <p>2. TCP 连接管理机制</p> <p>2.1 建立连接机制</p> <p>2.2 释放连接机制</p> <p>2.3 连接过程中的异常处理</p> <p>3. TCP 可靠传输原理</p> <p>3.1 确认机制</p> <p>3.2 重传机制</p> <p>3.3 分析传输模型</p> <p>4. 流量重传机制</p> <p>4.1 流量控制机制</p> <p>4.2 零窗口处理机制</p> <p>4.3 小窗口，傻瓜窗口问题</p> <p>5. 分析 TCP 的拥塞控制原理</p>
调试过程及实验结果	<p>1. TCP 协议基础</p> <p>(1) 在 wireshark 中，与百度 <a href="http://www.baidu.com">www.baidu.com</a> 建立连接，来验证整个实验，通过 ping <a href="http://www.baidu.com">www.baidu.com</a> 得知百度 ip 为 182.61.200.7</p> <p>(2) 停止捕捉，使用 wireshark 查看 tcp 包的详细信息，使用 filter 命令 <code>ip.addr == 182.61.200.7 and tcp</code></p> <p>查询结果如下：</p>

sdad.pcapng [Wireshark 1.8.5 (SVN Rev 47350 from /trunk-1.8)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: tcp Expression: Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
3260	2017-05-25 14:59:02.351490000	10.111.134.110	10.111.100.210	TCP	62	15bconference1 > ftp [SYN] S
3261	2017-05-25 14:59:02.352020000	10.111.100.210	10.111.134.110	TCP	62	ftp > 15bconference1 [SYN, A
3262	2017-05-25 14:59:02.352042000	10.111.134.110	10.111.100.210	TCP	54	15bconference1 > ftp [ACK] S
3264	2017-05-25 14:59:02.366504000	10.111.100.210	10.111.134.110	FTP	92	Response: 220 serv-u FTP Ser
3265	2017-05-25 14:59:02.366595000	10.111.134.110	10.111.100.210	FTP	70	Request: USER anonymous
3266	2017-05-25 14:59:02.369964000	10.111.100.210	10.111.134.110	FTP	124	Response: 331 User name okay
3267	2017-05-25 14:59:02.370052000	10.111.134.110	10.111.100.210	FTP	68	Request: PASS IEuser@
3268	2017-05-25 14:59:02.375315000	10.111.100.210	10.111.134.110	FTP	84	Response: 230 User logged in
3269	2017-05-25 14:59:02.375420000	10.111.134.110	10.111.100.210	FTP	68	Request: opts utf8 on
3270	2017-05-25 14:59:02.376840000	10.111.100.210	10.111.134.110	FTP	84	Response: 502 Command not in
3271	2017-05-25 14:59:02.376918000	10.111.134.110	10.111.100.210	FTP	59	Request: PWD
3272	2017-05-25 14:59:02.379561000	10.111.100.210	10.111.134.110	FTP	85	Response: 257 "/" is current
3273	2017-05-25 14:59:02.379644000	10.111.134.110	10.111.100.210	FTP	74	Request: CWD /\275\314\312\2
3274	2017-05-25 14:59:02.382217000	10.111.100.210	10.111.134.110	FTP	94	Response: 250 Directory chan
3275	2017-05-25 14:59:02.382399000	10.111.134.110	10.111.100.210	FTP	60	Request: noop
3276	2017-05-25 14:59:02.383888000	10.111.100.210	10.111.134.110	FTP	73	Response: 200 Command okay.
3277	2017-05-25 14:59:02.383978000	10.111.134.110	10.111.100.210	FTP	74	Request: CWD /\275\314\312\2
3279	2017-05-25 14:59:02.386451000	10.111.100.210	10.111.134.110	FTP	94	Response: 250 Directory chan
3280	2017-05-25 14:59:02.386533000	10.111.134.110	10.111.100.210	FTP	62	Request: TYPE A
3281	2017-05-25 14:59:02.387965000	10.111.100.210	10.111.134.110	FTP	74	Response: 200 Type set to A.
3282	2017-05-25 14:59:02.388152000	10.111.134.110	10.111.100.210	FTP	60	Request: PASV

Frame 3267: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface 0  
 Ethernet II, Src: Micro-St\_23:d6:3b (d4:3d:7e:23:d6:3b), Dst: FujianSt\_1b:96:af (14:14:4b:1b:96:af)  
 Internet Protocol Version 4, Src: 10.111.134.110 (10.111.134.110), Dst: 10.111.100.210 (10.111.100.210)  
 Transmission Control Protocol, Src Port: 15bconference1 (1244), Dst Port: ftp (21), Seq: 17, Ack: 109, Len: 14  
 File Transfer Protocol (FTP)

```

0000  14 14 4b 1b 96 af d4 3d 7e 23 d6 3b 08 00 45 00  ..K....~#...E.
0010  00 36 11 34 40 00 80 06 e9 6f 0a 6f 86 6e 0a 6f  .6.40...o.o.n.o
0020  64 d2 04 dc 00 15 a8 c9 8a 45 d4 0f 20 37 50 18  d.....E..7P.
0030  ff 93 00 47 00 00 50 41 53 53 20 49 45 55 73 65  ...G..PA SS IEuse
0040  72 40 0d 0a                                         r8..
  
```

File: "F:\计算机网络\shivan\sdad.pcapng"... Packets: 294584 Disposed: 19379 Marked: 0 Load time: 0:03.939

在每一个 tcp 包中，wireshark 给出了具体内容

Frame 217: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
Ethernet II, Src: dc1b:a1:73:56:68 (dc1b:a1:73:56:68), Dst: 20:76:93:3d:03:e9 (20:76:93:3d:03:e9)
Internet Protocol Version 4, Src: 180.168.123.32 (090:168:123:32), Dst: 180.66.200.7 (0102:66:200:7)
Transmission Control Protocol, Src Port: 49848 (49848), Dst Port: Netps (443), Seq: 1, Ack: 1, Len: 0

Frame: 物理层的数据帧概况

Ethernet II: 数据链路层以太网帧头部信息

Internet Protocol Version 4: 互联网层 IP 包头部信息

Transmission Control Protocol: 传输层 T 的数据段头部信息，此处是 TCP

至此准备工作已经做好，开始实验

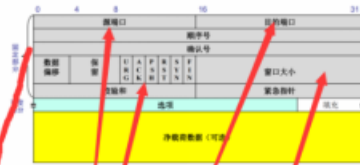
### 1.1 传输层目的地址源地址结构:

IP 地址和端口号用来唯一的确定网络上数据的地址。所以，每个 TCP 传输连接有两个端点，即源地址加端口号，目的地址加端口号

TCP 包的具体内容如下图所示，我截取了上课的 ppt 以及 wireshark 截取到的 TCP 包:

#### 6.4.2 TCP报文段格式

TCP报文段是TCP协议的数据交换单元，其格式如下：



```
[Source GeoIP: Unknown]
[Destination GeoIP: Unknown]
Transmission Control Protocol, Src Port: 49848 (49848), Dst Port: https (443), Seq: 1, Ac
Source port: 49848 (49848)
Destination port: https (443)
[Stream index: 7]
Sequence number: 1 (relative sequence number)
Acknowledgment number: 1 (relative ack number)
Header length: 20 bytes
Flags: 0x010 (ACK)
window size value: 516
[Calculated window size: 132096]
[window size scaling factor: 256]
Checksum: 0xba41 [validation disabled]
[SEQ/ACK analysis]
```

传输地址唯一地标识主机进程，传输地址=网络号+主机号+端口号，端口号即传输服务访问点（TSAP），用来标识应用进程。

在 IP 网络，传输地址= IP 地址+端口号

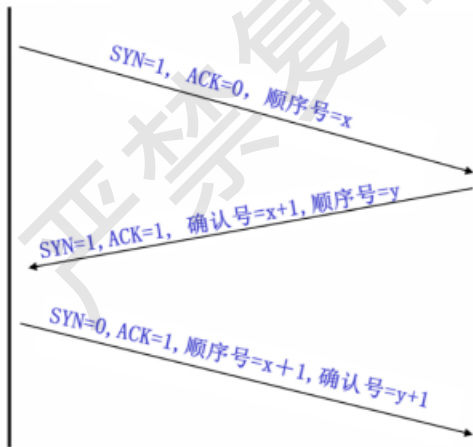
TCP 连接用四元组<源 IP 地址，源端口号，目的 IP 地址，目的端口号>表示

## 2. 分析 TCP 连接管理的机制

2.1. TCP 建立连接时，会有三次握手过程，如下图所示：

客户机C（发起方）

服务器S（接收方）



在 wireshark 中，找到与服务器端建立链接的三次握手

第一次握手：SYN=1, Seq=0, ACK=0

第二次握手: SYN=1, ACK=1, Seq=0

第三次握手: SYN=0, ACK=1, Seq=1

2.2 TCP 释放连接时, 会有“三次握手”过程, 如下图所示:

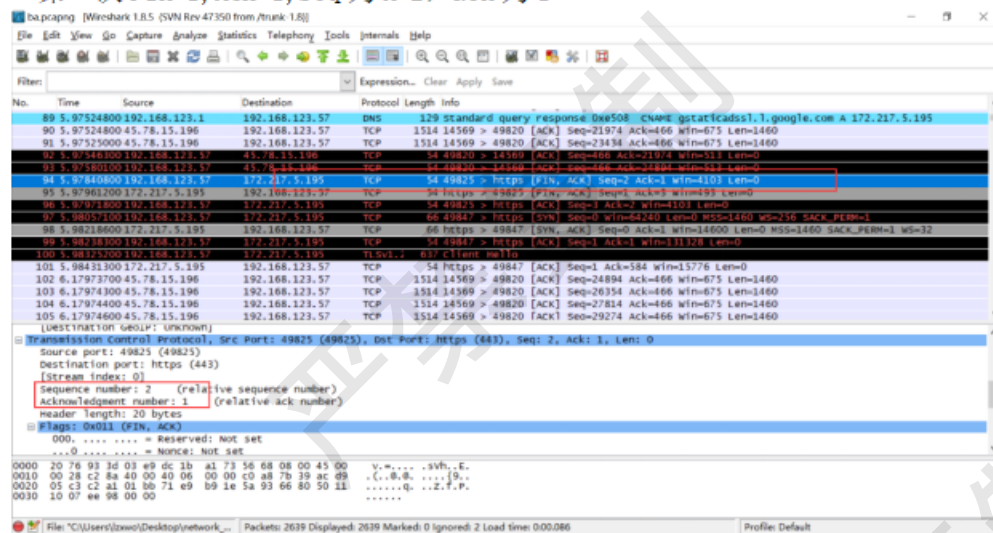


● TCP连接释放：仍采用“三次握手”

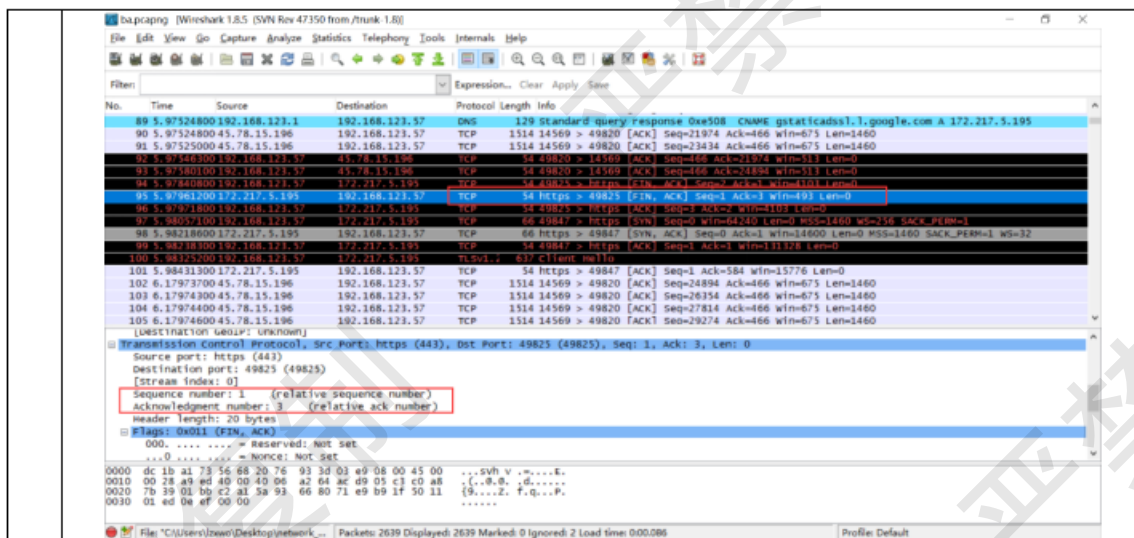


由于我在本例实验数据中没有找到四次挥手，只有三次，所以实验结果如下

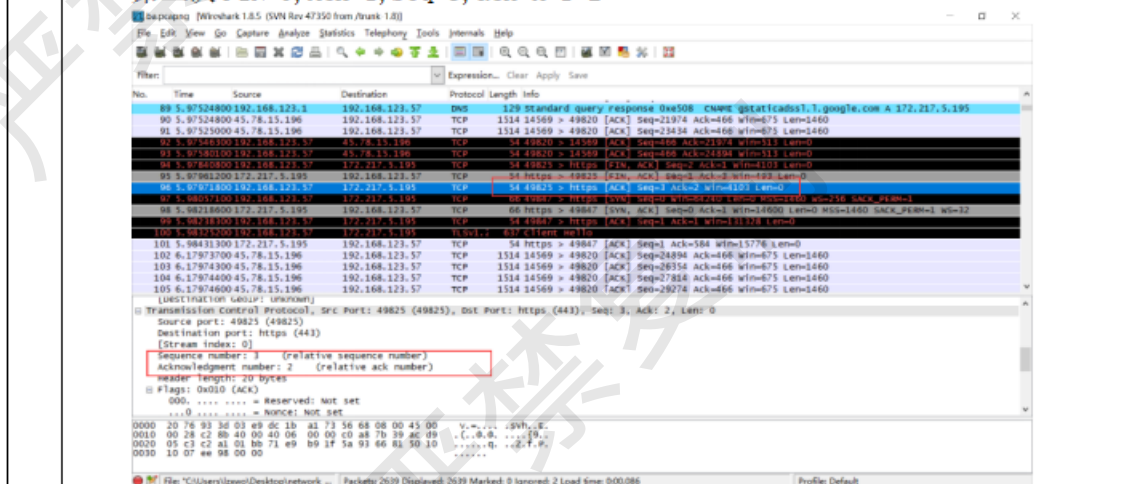
第一次 Fin=1, ACK=1, Seq 为 X=2, ack 为 1



第二次 FIN=1, ACK=1, Seq=X=1, ack=X+1=3



第三次 FIN=0, ACK=1, Seq=3, ack=X+1=2

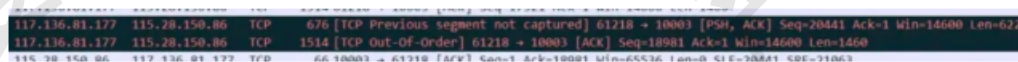


三次握手后，连接被释放。

## 2.3. 连接过程中的异常处理

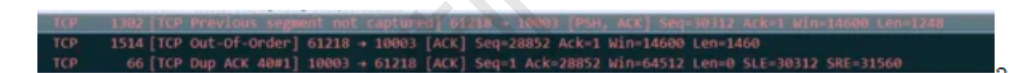
### 1. [TCP Previous segment not captured] 丢包

在 TCP 传输过程中，同一台主机发出的数据段应该是连续的，即后一个包的 Seq 号等于前一个包的 Seq + Len（三次握手和四次挥手是例外）当后 Seq > 前 Seq + Len，就知道中间缺失了一段数据



### 2. [TCP Dup ACK x#y]

当乱序或者丢包发生时，接收方会收到一些 Seq 号比期望值大的包。此时 Ack 表示，想获取 seq 数据包而给了其他的数据包





### 3. [TCP Spurious Retransmission] 丢包重传

451.13.8596610	192.168.123.57	59.78.173.33	TCP	54	49854 > https [RST, ACK] Seq=519 Ack=4097 Win=0 Len=0
453.13.8597410	192.168.123.57	59.78.173.33	TCP	54	49853 > https [RST, ACK] Seq=519 Ack=1413 Win=0 Len=0
463.13.8621750	192.168.123.57	59.78.173.33	TCP	54	49852 > https [RST, ACK] Seq=519 Ack=1413 Win=0 Len=0
467.13.8621900	192.168.123.57	59.78.173.33	TCP	54	49851 > https [RST, ACK] Seq=519 Ack=1413 Win=0 Len=0
486.13.8697840	192.168.123.57	59.78.173.33	TCP	54	49856 > https [RST, ACK] Seq=599 Ack=4605 Win=0 Len=0
843.14.0156460	192.168.123.57	59.78.173.33	TCP	54	49860 > https [RST, ACK] Seq=599 Ack=4605 Win=0 Len=0
848.14.0189800	192.168.123.57	59.78.173.33	TCP	54	49861 > https [RST, ACK] Seq=599 Ack=4605 Win=0 Len=0
1374.17.1420170	192.168.123.57	182.61.200.7	TCP	54	49851 > https [RST, ACK] Seq=5112 Ack=3058 Win=0 Len=0
1395.17.1631080	192.168.123.57	182.61.200.7	TCP	54	49848 > https [RST, ACK] Seq=5526 Ack=11159 Win=0 Len=0

丢包异常是我这个数据见到的最多的异常，经常出现

### 3. 分析 TCP 可靠传输原理

#### 3.1 确认机制

确认号 ACK 会告诉发送端哪些数据段已经成功接收，并且确认号会向发送端指出接收端希望收到的下一个序列号。即确实号 ACK 为上个数据序列号的下个数据序列号。

此时序列号 Seq=4403

1.200.7	192.168.123.57	TLSv1.2	99	Hello Request, Hello Request
88.123.57	182.61.200.7	TCP	54	49849 > https [ACK] Seq=4177 Ack=644 Win=79488 Len=0
1.200.7	192.168.123.57	TCP	54	https > 49849 [ACK] Seq=4177 Ack=644 Win=79488 Len=0
1.200.7	192.168.123.57	TLSv1.2	229	New Session Ticket
1.200.7	192.168.123.57	TLSv1.2	60	Change Cipher Spec
1.200.7	192.168.123.57	TLSv1.2	99	Hello Request, Hello Request
1.200.7	192.168.123.57	TCP	54	https > 49848 [ACK] Seq=4403 Ack=1310 Win=80896 Len=0
88.123.57	182.61.200.7	TCP	54	49849 > https [ACK] Seq=4403 Ack=1310 Win=131328 Len=0
1.200.7	192.168.123.57	TCP	1506	[TCP segment of a reassembled PDU]
88.123.57	182.61.200.7	TCP	54	49849 > https [ACK] Seq=4403 Ack=1310 Win=132096 Len=0
1.200.7	192.168.123.57	TCP	1506	[TCP segment of a reassembled PDU]
1.200.7	192.168.123.57	TLSv1.2	1179	Application Data
1.200.7	192.168.123.57	TLSv1.2	179	Application Data
1.200.7	192.168.123.57	TCP	1506	[TCP segment of a reassembled PDU]
1.200.7	192.168.123.57	TCP	1506	[TCP segment of a reassembled PDU]
88.123.57	182.61.200.7	TCP	54	49849 > https [ACK] Seq=4403 Ack=1310 Win=132096 Len=0
1.200.7	192.168.123.57	TLSv1.2	1179	Application Data
1.200.7	192.168.123.57	TLSv1.2	1506	Application Data

此时确认号 ACK=4403，说明上个数据报已成功接收，并发送期望接收的下个数据报。

1.200.7	192.168.123.57	TLSv1.2	99	Hello Request, Hello Request
88.123.57	182.61.200.7	TCP	54	49848 > https [ACK] Seq=1310 Ack=4403 Win=131328 Len=0
1.200.7	192.168.123.57	TCP	54	https > 49849 [ACK] Seq=4177 Ack=644 Win=79488 Len=0
1.200.7	192.168.123.57	TLSv1.2	229	New Session Ticket
1.200.7	192.168.123.57	TLSv1.2	60	Change Cipher Spec
1.200.7	192.168.123.57	TLSv1.2	99	Hello Request, Hello Request
1.200.7	192.168.123.57	TCP	54	https > 49848 [ACK] Seq=4403 Ack=1310 Win=80896 Len=0
88.123.57	182.61.200.7	TCP	54	49849 > https [ACK] Seq=4403 Ack=1310 Win=131328 Len=0
1.200.7	192.168.123.57	TCP	1506	[TCP segment of a reassembled PDU]
88.123.57	182.61.200.7	TCP	54	49849 > https [ACK] Seq=4403 Ack=1310 Win=132096 Len=0
1.200.7	192.168.123.57	TCP	1506	[TCP segment of a reassembled PDU]
1.200.7	192.168.123.57	TLSv1.2	1179	Application Data
1.200.7	192.168.123.57	TLSv1.2	179	Application Data
1.200.7	192.168.123.57	TCP	1506	[TCP segment of a reassembled PDU]
1.200.7	192.168.123.57	TCP	1506	[TCP segment of a reassembled PDU]
88.123.57	182.61.200.7	TCP	54	49848 > https [ACK] Seq=1310 Ack=11461 Win=132096 Len=0
1.200.7	192.168.123.57	TLSv1.2	1179	Application Data
1.200.7	192.168.123.57	TLSv1.2	1506	Application Data

#### 3.2 重传机制

当发送端在给定时间间隔内收不到那个数据段的应答时，发送端就会重传那个数据段。三个情况下，会发生数据重传

情况 1：网络延时/环路，数据段丢失

情况 2：网络延时，数据段推迟到达

情况 3：数据段成功到达，应答不能达到

TCP 重传过程在实验数据中发生如下图:

在通常情况下, 第一个报文发送之后很快会收到 TCP ACK 报文, 意外情况发生时, 紧接着的是重传报文:

859 13.8596230	192.168.123.57	59.78.173.33	UDP	54	standard query response v0x100	LNAM www.3miren.com A 192.168.123.57
859 13.8596230	192.168.123.57	59.78.173.33	TLsv1.2	393	Application data	
859 13.8596810	192.168.123.57	59.78.173.33	TCP	54	49854 > https [RST, ACK] Seq=519 Ack=4097 win=0 Len=0	
859 13.8597170	192.168.123.57	59.78.173.33	TLsv1.2	154	Application data	
859 13.8597410	192.168.123.57	59.78.173.33	TCP	54	49854 > https [RST, ACK] Seq=519 Ack=1413 win=0 Len=0	
859 13.8597600	192.168.123.57	59.78.173.33	TLsv1.2	154	Application data	
859 13.8598190	192.168.123.57	59.78.173.33	TLsv1.2	154	Application data	
859 13.8599270	192.168.123.57	59.78.173.33	TLsv1.2	158	Application data	
859 13.8599920	192.168.123.57	59.78.173.33	TLsv1.2	153	Application data	

可以观察到, 在 RST 前后都是 154 字节, 字节数一样, 可以看出的是重传了一次

### 3.3. 分析传输模型

这个问题我查阅了有关资料, 《Linux 内核网络栈源代码情景分析》在我的理解中, TCP 协议可靠性数据传输实现是依靠数据重传和数据确认应答机制来完成 TCP 协议的可靠性数据传输。

而数据超时重传和数据应答机制的基本前提是对每个传输的字节进行编号, 即序列号。

数据超时重传是发送端在某个数据包发送出去, 在一段固定时间后如果没有收到对该数据包的确认应答, 则 (假定该数据包在传输过程中丢失) 重新发送该数据包。

数据确认应答是指接收端在成功接收到一个有效数据包后, 发送一个确认应答数据包给发送端主机, 该确认应答数据包中所包含的应答序列号即指已接收到的数据中最后一个字节的序列号加 1, 加 1 的目的在于指出此时接收端期望接收的下一个数据包中第一个字节的序列号。

数据超时重传和数据确认应答以及对每个传输的字节分配序列号是 TCP 协议提供可靠性数据传输的核心本质。重传应答机制与序列号结合能解决四个问题:

1> 能够处理数据在传输过程中被破坏的问题。

通过对所接收数据包的校验, 确认该数据包中数据是否存在错误。如果有, 则简单丢弃或者发送一个应答数据包重新对这些数据进行请求。发送端在等待一段时间后, 则会重新发送这些数据。

2> 能够处理接收重复数据问题。

首先利用序列号可以发现数据重复问题。因为每个传输的数据均被赋予一个唯一的序列号, 如果到达的两份数据具有重叠的序列号 (如由发送端数据包重传造成),

则表示出现数据重复问题。

3> 能够发现数据丢失以及进行有效解决。

数据丢失的判断是猜测性的，无法确定一个数据包一定丢失在传输过程中，大多是被延迟在网络中，即实质的问题只是数据包乱序到达。可能的数据丢失显然的结果是在接收端接收的数据出现序列号不连续现象。

4> 能够处理接收端数据乱序到达问题。

如果通信双方存在多条传输路径，则有可能出现数据乱序问题，即序列号较大的数据先于序列号较小的数据到达，而发送端确实是按序列号由小到大的顺序发送的。这个问题的解决只需对这些数据进行重新排序即可。

#### 4. 分析 TCP 流量控制原理

##### 4.1 流量控制机制

窗口数决定了当前传输的最大流量。当我们在传输过程中，通信双方可以根据网络条件动态协商窗口大小，调整窗口大小时，即可实现流量控制。（在 TCP 的每个确认中，除了 ACK 外，还包括一个窗口通知）TCP 通过滑动窗口机制检测丢包，并在丢包发生时调整数据传输速率。滑动窗口机制利用数据接收端的接收窗口来控制数据流。并且调整窗口大小在两个方向都是可行的。当服务器能够更加快速的处理报文时，它会发送一个较大窗口的 ACK 报文。

在 Wireshark 中，可以很清楚的看到，在传输过程中经常会协商窗口的大小

57	203.208.43.98	TLSv1.2	1255 Application Data
57	45.78.15.196	TCP	66 49846 > 14569 [SYN] Seq=0 win=64240 Len=0 MSS=1460 WS=256 SACK_PER
38	192.168.123.57	TCP	54 https > 49770 [ACK] Seq=251 Ack=279 win=452 Len=0
98	192.168.123.57	TCP	54 https > 49807 [ACK] Seq=1 Ack=69 win=331 Len=0
98	192.168.123.57	TCP	54 https > 49807 [ACK] Seq=1 Ack=108 win=331 Len=0
98	192.168.123.57	TCP	54 https > 49807 [ACK] Seq=1 Ack=1309 win=343 Len=0
98	192.168.123.57	TLSv1.2	93 Application Data
98	192.168.123.57	TLSv1.2	119 Application Data
57	203.208.43.98	TCP	54 49807 > https [ACK] Seq=1309 Ack=105 win=508 Len=0
98	192.168.123.57	TLSv1.2	85 Application Data
98	192.168.123.57	TLSv1.2	93 Application Data

如上图，正在沟通自己的 win 窗口还剩多少个

##### 4.2 零窗口机制

可能是由于内存不足，处理能力不够，或其他原因导致服务器无法再处理从客户端发送的数据。这可能会造成数据被丢弃以及传输暂停，但接收窗口能够帮助减小负面影响。

当上述情况发生时，服务器会发送窗口为 0 的报文。

Time	Source	Destination	Protocol	Length	Info
446.13.8195510	192.168.123.33	192.168.123.57	DNS	302	Standard query response 0x6f07 CNAME www.x.shifen.com A 182.61.200.7 A 182.61.200.6
447.13.8195510	59.78.173.33	192.168.123.57	TLV1.2	1326	Application Data
448.13.8195540	59.78.173.33	192.168.123.57	TLV1.2	1466	Server hello, change cipher spec, Application data
449.13.8195540	192.168.123.57	192.168.123.33	DNS	131	Standard query response 0x6f0e CNAME www.x.shifen.com A 182.61.200.6 A 182.61.200.7
450.13.8196030	192.168.123.57	59.78.173.33	TLV1.2	391	Application Data
451.13.8196010	192.168.123.57	59.78.173.33	TCP	54	48854 → https [RST, ACK] Seq=519 Ack=4037 Win=0 Len=0
452.13.8197170	192.168.123.57	59.78.173.33	TLV1.2	134	Application Data
453.13.8197410	192.168.123.57	59.78.173.33	TCP	54	48853 → https [RST, ACK] Seq=519 Ack=4413 Win=0 Len=0
454.13.8197500	192.168.123.57	59.78.173.33	TLV1.2	134	Application Data
455.13.8198190	192.168.123.57	59.78.173.33	TLV1.2	134	Application Data
456.13.8199270	192.168.123.57	59.78.173.33	TLV1.2	134	Application Data
457.13.8199270	192.168.123.57	59.78.173.33	TLV1.2	134	Application Data
458.13.8606020	59.78.173.33	192.168.123.57	TCP	1466	[TCP segment of a reassembled PDU]
459.13.8606040	59.78.173.33	192.168.123.57	TLV1.2	1326	Application Data
460.13.8606040	59.78.173.33	192.168.123.57	TLV1.2	275	Application Data
461.13.8606090	59.78.173.33	192.168.123.57	TLV1.2	275	Application Data
462.13.8621360	59.78.173.33	192.168.123.57	TLV1.2	1466	Server hello, change cipher spec, Application data

当客户端接收到此报文时，它会暂停所有数据传输，但会保持与服务器的连接以传输探测报文。一旦服务器能够再次处理数据，将会返回非零值窗口大小，传输会恢复。如下图所示，win 窗口变为 71168

461.13.8621360	192.168.123.57	59.78.173.33	TCP	54	48852 → https [RST, ACK] Seq=519 Ack=1411 Win=0 Len=0
464.13.8631170	192.168.123.57	192.168.123.33	DNS	73	Standard query response 0x6f0c A sp2.baidu.com
465.13.8630890	59.78.173.33	192.168.123.57	TCP	1466	[TCP segment of a reassembled PDU]
466.13.8682240	59.78.173.33	192.168.123.57	TLV1.2	1466	Server hello, change cipher spec, Application data
467.13.8687190	192.168.123.57	59.78.173.33	TCP	54	48857 → https [RST, ACK] Seq=519 Ack=1411 Win=0 Len=0
468.13.8683200	59.78.173.33	192.168.123.57	TCP	1466	[TCP segment of a reassembled PDU]
469.13.8684210	59.78.173.33	192.168.123.57	TLV1.2	1326	Application Data
470.13.8686070	59.78.173.33	192.168.123.57	TLV1.2	275	Application Data
471.13.8686030	59.78.173.33	192.168.123.57	TLV1.2	1326	Application Data
472.13.8670140	192.168.123.57	192.168.123.33	DNS	302	Standard query response 0x6f0c CNAME www.x.shifen.com A 182.61.200.6 A 182.61.200.7
473.13.8670400	59.78.173.33	192.168.123.57	TLV1.2	275	Application Data
474.13.8670710	192.168.123.57	182.61.200.7	TCP	60	[TCP Dup ACK 4152] 48849 → https [ACK] Seq=424 Ack=598 Win=71168 Len=0
476.13.8680110	59.78.173.33	192.168.123.57	TCP	54	https → 48855 [ACK] Seq=418 Ack=598 Win=71168 Len=0
477.13.8682920	59.78.173.33	192.168.123.57	TLV1.2	341	Application Data
478.13.8682920	59.78.173.33	192.168.123.57	TLV1.2	341	Application Data

#### 4.3 小窗口，傻瓜窗口问题

如下图，当发送零窗口后，将小包转化为大包传送

484.13.8697380	59.78.173.33	192.168.123.57	TLV1.2	341	Application Data
485.13.8697390	59.78.173.33	192.168.123.57	TCP	54	https → 48856 [ACK] Seq=4892 Ack=599 Win=71168 Len=0
486.13.8697440	192.168.123.57	59.78.173.33	TCP	54	48856 → https [RST, ACK] Seq=599 Ack=4893 Win=0 Len=0
487.13.8703820	59.78.173.33	192.168.123.57	TCP	54	https → 48856 [FIN, ACK] Seq=4892 Ack=599 Win=71168 Len=0
488.13.8706580	59.78.173.33	192.168.123.57	TCP	54	https → 48855 [ACK] Seq=4954 Ack=690 Win=71168 Len=0
489.13.8710310	59.78.173.33	192.168.123.57	TCP	54	https → 48853 [ACK] Seq=4318 Ack=519 Win=71168 Len=0
490.13.8710310	59.78.173.33	192.168.123.57	TCP	54	https → 48852 [ACK] Seq=4318 Ack=519 Win=71168 Len=0
491.13.8710310	59.78.173.33	192.168.123.57	TLV1.2	85	Application Data

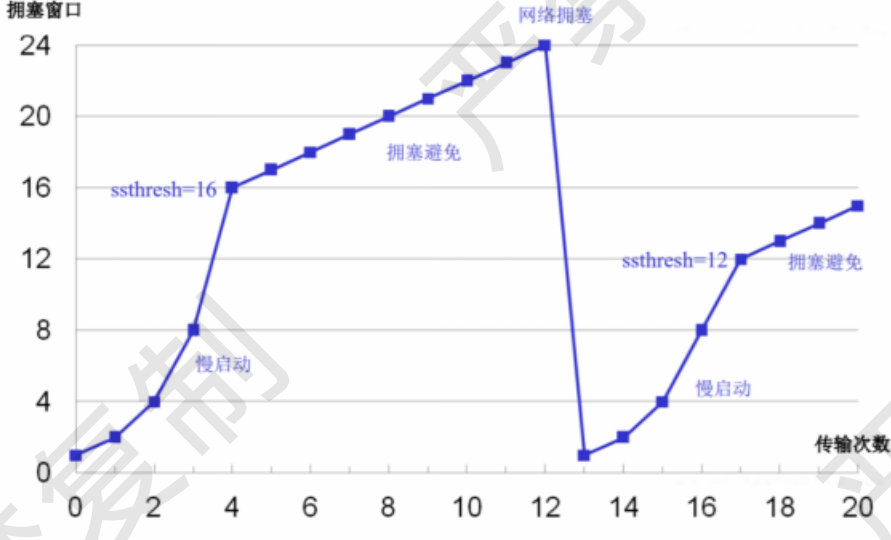
### 5. 分析 TCP 的拥塞控制原理

发送端设置拥塞窗口来反映网络容量，通过拥塞窗口来限制发送方向网络注入数据的速度，即发送端允许发送的数据既不能超过接收窗口的大小，也不能超过拥塞窗口的大小；前者是为了流量控制，后者是为了拥塞控制。通过慢启动和拥塞避免策略来控制拥塞窗口的大小

#### 慢启动:

设:  $ssthresh$  为慢启动阈值,  $MSS$  为最大 TCP 段长度,  $cwnd$  为拥塞窗口大小。发送端初始  $cwnd$  一般设置为 1 个  $MSS$ ,  $ssthresh$  设置为接收窗口大小。每收到一个确认 ACK, 则  $cwnd = cwnd + 1$  个  $MSS$ 。初始  $cwnd=1$ , 发送端发送 1 个报文段  $M_0$ , 接收端收到后返回  $ACK_1$ 。发送端收到  $ACK_1$  后, 将  $cwnd$  从 1 增大到 2, 于是接着发送  $M_1$  和  $M_2$  两个报文段。接收端收到后返回  $ACK_2$  和  $ACK_3$ 。发送端收到后, 将  $cwnd$  从 2 增大到 4, 于是接着发送  $M_3 \sim M_6$  共 4 个报文段。当  $cwnd > ssthresh$  时, 进入拥塞避免阶段



	 <p><b>拥塞避免策略:</b></p> <p>每经过一个往返传输时间 RTT (不管在 RTT 时间内收到几个 ACK ), 则 <math>cwnd = cwnd + 1</math> 个 MSS; RTT 是动态变化的。如果超过一段时间 (TCP 重传超时) 没有收到 TCP 报文段, 则认为网络拥塞。不管是慢启动阶段, 还是拥塞避免阶段, 如果 TCP 检测到拥塞, 则将 <math>ssthresh</math> 缩减成 <math>cwnd</math> 的一半, 且 <math>cwnd</math> 恢复为初始大小, 即 1 个 MSS</p>
<p><b>总 结</b></p>	<p>实验之前, 我重温了 TCP 协议的定义及特点。</p> <p>TCP 是一种面向连接 (连接导向) 的、可靠的基于字节流的传输层通信协议。TCP 将用户数据打包成报文段, 它发送后启动一个定时器, 另一端收到的数据进行确认、对失序的数据重新排序、丢弃重复数据。</p> <p>TCP 的特点有:</p> <p>TCP 是面向连接的传输层协议, 每一条 TCP 连接只能有两个端点, 每一条 TCP 连接只能是点对点的</p> <p>TCP 提供可靠交付的服务</p> <p>TCP 提供全双工通信。数据在两个方向上独立的进行传输。因此, 连接的每一端必须保持每个方向上的传输数据序号。</p> <p>TCP 面向字节流。虽然应用程序和 TCP 交互是一次一个数据块, 但 TCP 把应用程序交下来的数据仅仅是一连串的无结构的字节流。</p> <p>在本次实验中, 我掌握了上课老师所教授的 TCP 内容, 通过实验了解了 TCP 的原理, 什么情况下会重传报文, 什么时候会因网络堵塞进行流量控制。这次实验整体坐下来, 让我对课本的理解更加深入了。</p>
<p><b>附 录</b></p>	<p>无</p>