



# 计算机网络

实验（五）

姓 名	熊恪峥
学 号	22920202204622
日 期	2022年12月1日
学 院	信息学院
课程名称	计算机网络

# 实验（五）

## 目录

1	任务1、TCP 正常连接观察	1
2	任务2、TCP异常传输观察分析	2
2.1	尝试连接未存活的主机或对未监听端口 . . . . .	2
2.2	客户端发送了第一个 SYN 连接请求，服务器无响应 . . . . .	3
3	任务3、拥塞控制	4
3.1	实验准备 . . . . .	4
3.2	实验结果 . . . . .	4
4	任务4、HTTP协议分析	5
4.1	实验准备 . . . . .	5
4.2	实验结果 . . . . .	6

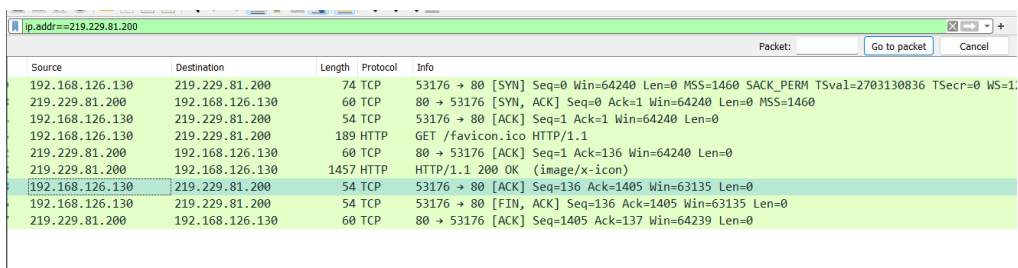
## 1 任务1、TCP 正常连接观察

在终端运行命令

```
wget http://mba.xmu.edu.cn/favicon.ico --no-http-keep-alive
```

传输上述URL中的文件，观察TCP连接的建立和关闭过程。抓取到的数据段如图 1所示。 作出TCP流图，如

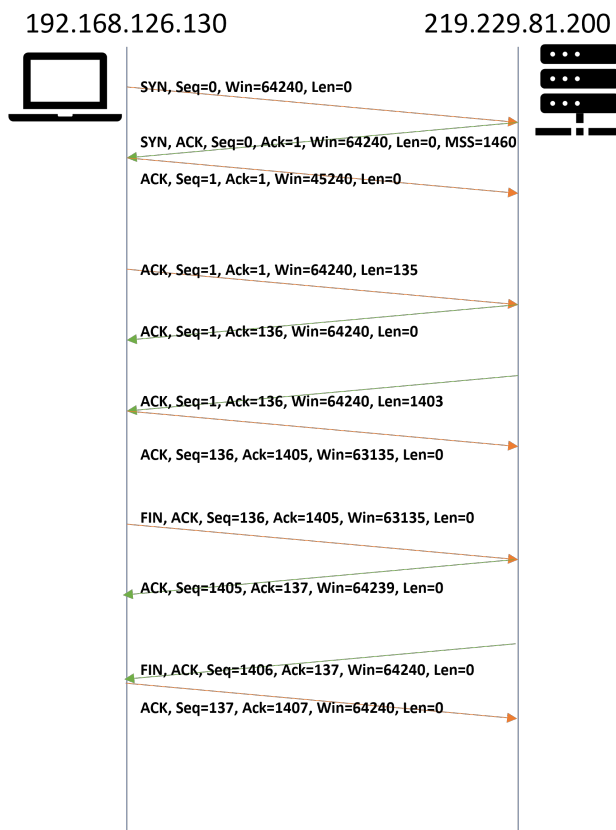
图 1: TCP数据段



Source	Destination	Length	Protocol	Info
192.168.126.130	219.229.81.200	74	TCP	53176 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2703130836 TSecr=0 WS=1
219.229.81.200	192.168.126.130	60	TCP	80 → 53176 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
192.168.126.130	219.229.81.200	54	TCP	53176 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
192.168.126.130	219.229.81.200	189	HTTP	GET /favicon.ico HTTP/1.1
219.229.81.200	192.168.126.130	60	TCP	80 → 53176 [ACK] Seq=1 Ack=136 Win=64240 Len=0
192.168.126.130	219.229.81.200	1457	HTTP	HTTP/1.1 200 OK (image/x-icon)
192.168.126.130	219.229.81.200	54	TCP	53176 → 80 [ACK] Seq=136 Ack=1405 Win=63135 Len=0
192.168.126.130	219.229.81.200	54	TCP	53176 → 80 [FIN, ACK] Seq=136 Ack=1405 Win=63135 Len=0
219.229.81.200	192.168.126.130	60	TCP	80 → 53176 [ACK] Seq=1405 Ack=137 Win=64239 Len=0

图 2所示。 由于Wireshark默认显示Sequence Number的相对值，因此这两个值都从0开始。实际上Sequence

图 2: TCP流图



Number 由TCP连接的建立时随机生成，客户端从354954929开始，服务器端从794129432开始。如图 3所示。

图 3: TCP Sequence Number

```
Sequence Number: 354954930
[Next Sequence Number: 1      (relative sequence number)]
Acknowledgment Number: 1      (relative ack number)
Acknowledgment number (raw): 354954930
```

2 任务2、TCP异常传输观察分析

2.1 尝试连接未存活的主机或对未监听端口

首先，随便指定一不存在的IP，使用wget访问该IP，例如192.168.3.2。如图 4所示。这样不存在的IP无

图 4: 不存在的IP

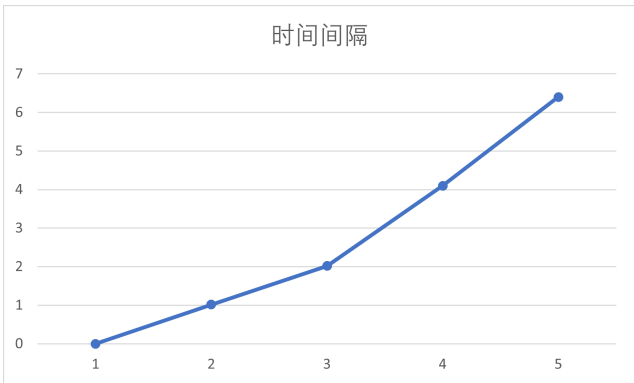
```
Info
60662 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2729655920 TSecr=0 WS=128
[TCP Retransmission] [TCP Port numbers reused] 60662 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=
[TCP Retransmission] [TCP Port numbers reused] 60662 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=
[TCP Retransmission] [TCP Port numbers reused] 60662 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=
80 → 58262 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
```

法回应SYN请求，会引起不断地重传。Ubuntu中的重传次数默认为3。以图中的重传为例，SYN发送后进行了三次重传，然后由于无响应发送了RST。可以计算时间间隔如表 1和图 5所示。

表格 1: 时间间隔

1 → 2	2 → 3	3 → 4	4 → 5
1.013	2.015	4.095	6.400

图 5: 时间间隔



可以发现忽略测量误差，时间间隔是按 $2^n$ 指数增长的。

借助RST，可以实现端口扫描。为针对TCP扫描，目前存在防御方式：若发现网络中的某台设备进行了端口扫描，会将其加入黑名单。实现这种防御的原理是：每次TCP连接后会将信息记录到日志中，当发现某IP多次连接设备的不同端口，就可以判断是TCP扫描，此时就可以将此IP加入黑名单。为避免被TCP扫描抓到，可以采用SYN扫描，原理同样是利用了TCP三次握手，过程如下：

- 1. 扫描端向目标端发送SYN请求建立连接
- 2. 目标端收到请求后，回复ACK同意连接并同意发送SYN请求建立连接
- 3. 扫描端收到后，发送RST拒绝建立连接。

图 6: nmap扫描

```
Starting Nmap 7.80 ( https://nmap.org ) at 2022-12-01 17:44 CST
Failed to resolve "sS".
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000073s latency).
Not shown: 995 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
80/tcp    open  http
81/tcp    open  hosts2-ns
631/tcp   open  ipp
Nmap done: 1 IP address (1 host up) scanned in 1.37 seconds
```

使用 `nmap -sS localhost` 命令扫描本机，如图 6 所示。它指出本机的 SSH 端口、FTP 端口等是开放的。用 Wireshark 观察抓到的包，例如 FTP 端口 21，如图 ?? 所示。

可见，由于 21 端口是开放的，服务器得到 SYN 请求后会回复 ACK 准备建立连接。此时扫描程序得知了 21 端口是开放的，但为了防止 IP 黑名单，就发送 RST 拒绝建立连接，此时服务器不能建立连接。因此无法记录客户 IP 进行屏蔽。这样就使得扫描能够顺利完成。如图 7 所示。又例如端口 587 未开放。因此服务器就会发

图 7: 21 端口开放

```
58 TCP    192.168.126.130 → 127.0.0.1 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
58 TCP    34094 → 21 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
58 TCP    21 → 34094 [SYN, ACK] Seq=0 Ack=1 Win=65495 Len=0 MSS=65495
54 TCP    34094 → 21 [RST] Seq=1 Win=0 Len=0
```

送 RST, ACK 表示收到请求但拒绝建立连接。扫描程序收到后，就会认为端口 587 未开放。如图 8 所示。

图 8: 587 端口未开放

```
58 TCP    34094 → 587 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
54 TCP    587 → 34094 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
```

## 2.2 客户端发送了第一个 SYN 连接请求，服务器无响应

为了简单地测试，可以通过下载远程的文件来测试。这里使用的文件如下：

```
https://mirrors.tuna.tsinghua.edu.cn/ubuntu-cdimage/releases/kinetic/
release/ubuntu-22.10-live-server-s390x.iso
```

首先用 `nslookup` 命令获取下载该文件访问的 IP 地址，得知 IP 地址为 101.6.15.130。然后使用以下命令屏蔽 IP 地址发来的 TCP SYN 和 ACK 报文。

```
sudo iptables -I INPUT -s 101.6.15.130 -p tcp -m tcp --tcp-flags ALL SYN,ACK -j DROP
```

这样其它的 TCP 内容可以正常通过，而建立连接的 TCP SYN-ACK 不能正常被收到。

然后使用 `wget` 命令下载文件，进行实验。首先，最初的 SYN 报文到达了服务器，然而服务器返回的 SYN-ACK 无法被 `wget` 程序收到。因此导致了多次重传，如图 9。接下来，由于服务器相应被 `iptables` 丢弃，因此在指定时间之内收不到 ACK 报文的客户端认为无从得知 SYN 是否被服务器收到，就会产生 SYN 请求的重传，如图 10。

由于 SYN-ACK 无法到达，因此这个过程会反复持续多次。最终，`wget` 程序无法建立连接会发送 RST，结束这一过程。如图 11。

图 11: RST

```
192.168.126.130 101.6.15.130 54 TCP 48870 → 443 [RST] Seq=18256 Win=0 Len=0
```

图 9: SYN-ACK重传

192.168.126.130	101.6.15.130	74 TCP	39990 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=172412998 TSecr=0 Len=28
101.6.15.130	192.168.126.130	60 TCP	[TCP Port numbers reused] 443 → 39990 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
101.6.15.130	192.168.126.130	60 TCP	[TCP Retransmission] 443 → 39990 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
101.6.15.130	192.168.126.130	60 TCP	[TCP Retransmission] 443 → 39990 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
101.6.15.130	192.168.126.130	60 TCP	[TCP Retransmission] 443 → 39990 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460

图 10: SYN重传

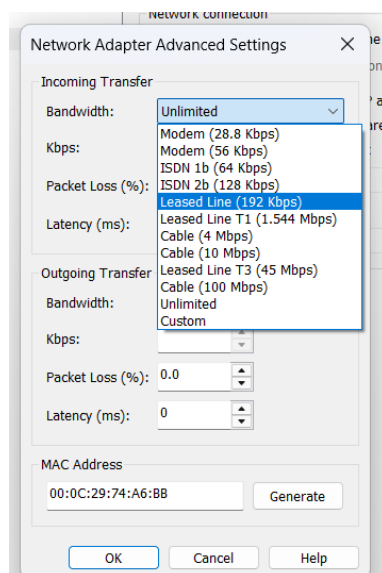
192.168.126.130	101.6.15.130	74 TCP	[TCP Retransmission] [TCP Port numbers reused] 39990 → 443 [SYN] Seq=0
101.6.15.130	192.168.126.130	60 TCP	[TCP Retransmission] 443 → 39990 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
192.168.126.130	101.6.15.130	74 TCP	[TCP Retransmission] [TCP Port numbers reused] 39990 → 443 [SYN] Seq=0
101.6.15.130	192.168.126.130	60 TCP	[TCP Retransmission] 443 → 39990 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
192.168.126.130	101.6.15.130	74 TCP	[TCP Retransmission] [TCP Port numbers reused] 39990 → 443 [SYN] Seq=0
101.6.15.130	192.168.126.130	60 TCP	[TCP Retransmission] 443 → 39990 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460

### 3 任务3、拥塞控制

#### 3.1 实验准备

首先为了观察普通的拥塞控制算法，需要对使用的拥塞控制算法进行修改。所以使用命令 `sysctl -w net.ipv4.tcp_congestion_control=reno` 改拥塞控制算法为Reno。然后在虚拟机设置中对带宽进行限制。如图 12所示。

图 12: 对带宽进行限制



这里为了使得效果明显，将带宽限制为192Kbps，然后下载第2.2节中提到的大文件。

#### 3.2 实验结果

使用Wireshark中的IO Graph功能做出了如图 13所示的图表。从I/O图中可以看出首先，每秒传播的字节数按照指数级增长，然后按照线性增长。后来有一些下降的过程，然后快速地增长，大致对应了慢启动、拥塞避免和快速重传的过程。此外，借助Wireshark绘制吞吐量图如图 14。可以发现也遵循了类似的变化过程。

例如2秒左右时发生的快速重传，首先由于链路带宽的问题，客户端收到了不正常的序列号，因此启动了快速重传机制。如图 15所示。由于Duplicate Ack的存在，服务器端重传了相应的报文，如图 16所示。此时重传的报文还加入了PSH标志，让客户端尽快交付。从图 13中可以观察到，虽然由于重传 *cwnd* 减少了，但是由于快速重传机制，*cwnd* 开始快速回复了，进行了指数级的增长。

图 13: IO Graph

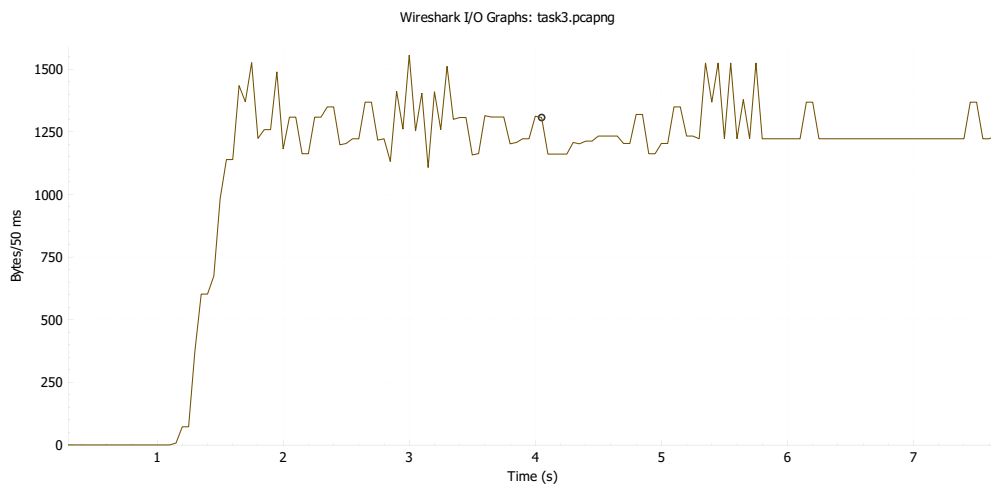
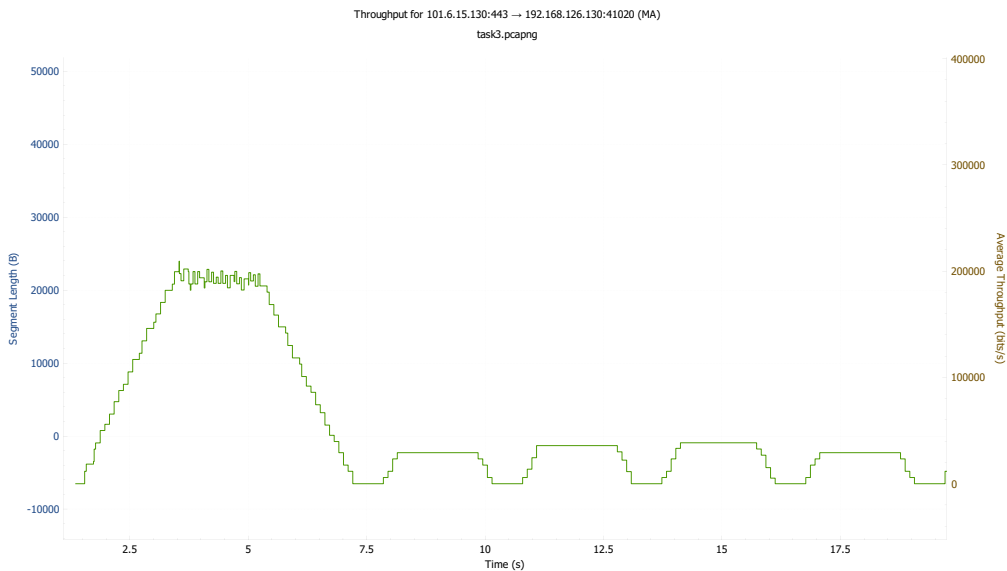


图 14: 吞吐量图



## 4 任务4、HTTP协议分析

### 4.1 实验准备

`http.server`库提供直接使用命令行的方式启动HTTP服务器，但是早期版本不方便切换协议，因此先把python版本升级到3.11，如图 17所示。

这样就可以用`--protocol`参数直接指定协议了，方便了后续的实验。

图 15: DupACK

192.168.126.130	101.6.15.130	54	TCP	[TCP Dup ACK 109#1] 41020 -> 443 [ACK] Seq=726 Ack=92002 Win=65535 Len=0
192.168.126.130	101.6.15.130	54	TCP	[TCP Dup ACK 109#2] 41020 -> 443 [ACK] Seq=726 Ack=92002 Win=65535 Len=0
192.168.126.130	101.6.15.130	54	TCP	[TCP Dup ACK 109#3] 41020 -> 443 [ACK] Seq=726 Ack=92002 Win=65535 Len=0
192.168.126.130	101.6.15.130	54	TCP	[TCP Dup ACK 109#4] 41020 -> 443 [ACK] Seq=726 Ack=92002 Win=65535 Len=0
192.168.126.130	101.6.15.130	54	TCP	[TCP Dup ACK 109#5] 41020 -> 443 [ACK] Seq=726 Ack=92002 Win=65535 Len=0
192.168.126.130	101.6.15.130	54	TCP	[TCP Dup ACK 109#6] 41020 -> 443 [ACK] Seq=726 Ack=92002 Win=65535 Len=0
192.168.126.130	101.6.15.130	54	TCP	[TCP Dup ACK 109#7] 41020 -> 443 [ACK] Seq=726 Ack=92002 Win=65535 Len=0
192.168.126.130	101.6.15.130	54	TCP	[TCP Dup ACK 109#8] 41020 -> 443 [ACK] Seq=726 Ack=92002 Win=65535 Len=0
192.168.126.130	101.6.15.130	54	TCP	[TCP Dup ACK 109#9] 41020 -> 443 [ACK] Seq=726 Ack=92002 Win=65535 Len=0

图 16: 重传

101.6.15.130	192.168.126.130	1514 TCP	443 → 41020 [PSH, ACK] Seq=92002 Ack=726 Win=64240 Len=1460 [TCP segment of a reassembled PDU]
101.6.15.130	192.168.126.130	1514 TCP	443 → 41020 [ACK] Seq=93462 Ack=726 Win=64240 Len=1460 [TCP segment of a reassembled PDU]

图 17: 升级python版本

By default, the server is conformant to HTTP/1.0. The option `-p/--protocol` specifies the HTTP version to which the server is conformant. For example, the following command runs an HTTP/1.1 conformant server:

```
python -m http.server --protocol HTTP/1.1
```

New in version 3.11: `--protocol` argument was introduced.

## 4.2 实验结果

实验结果如图 18所示。图 18a显示了HTTP1.0多次请求的结果，由于HTTP1.0不支持持久连接，因此每次请求都需要重新建立连接。可以看到，每次请求都先包含了 TCP的三次握手。在请求结束后都包含了TCP的四次挥手。这样多次建立连接，既浪费了几个RTT的时间，又浪费了服务器端的资源。

图 18: 不同版本的HTTP

Length	Protocol	Info
74	TCP	3920 → 8000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=62779128 TSecr=0 WS=128
74	TCP	8000 → 3920 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=62779128 TSecr=62779128 WS=128
66	TCP	3920 → 8000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=62779128 TSecr=62779128
558	HTTP	GET /test.txt HTTP/1.1
173	HTTP	HTTP/1.0 304 Not Modified
66	TCP	3920 → 8000 [ACK] Seq=503 Ack=108 Win=65536 Len=0 TSval=62779132 TSecr=62779132
66	TCP	8000 → 3920 [FIN, ACK] Seq=108 Ack=503 Win=65536 Len=0 TSval=62779132 TSecr=62779132
66	TCP	3920 → 8000 [FIN, ACK] Seq=503 Ack=108 Win=65536 Len=0 TSval=62779132 TSecr=62779132
66	TCP	8000 → 3920 [ACK] Seq=109 Ack=504 Win=65536 Len=0 TSval=62779132 TSecr=62779132
74	TCP	54022 → 8000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=62777179 TSecr=0 WS=128
74	TCP	8000 → 54022 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=62777179 TSecr=62777179 WS=128
66	TCP	54022 → 8000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=62777179 TSecr=62777179
558	HTTP	GET /test.txt HTTP/1.1
66	TCP	8000 → 54022 [ACK] Seq=503 Ack=504 Len=0 TSval=62777179 TSecr=62777179
173	HTTP	HTTP/1.0 304 Not Modified
66	TCP	54022 → 8000 [ACK] Seq=503 Ack=108 Win=65536 Len=0 TSval=62777180 TSecr=62777180
66	TCP	8000 → 54022 [FIN, ACK] Seq=108 Ack=503 Win=65536 Len=0 TSval=62777180 TSecr=62777180
66	TCP	54022 → 8000 [FIN, ACK] Seq=503 Ack=108 Win=65536 Len=0 TSval=62777180 TSecr=62777180

(a) HTTP1.0

(b) HTTP1.1

74	TCP	42014 → 8000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=627841533 TSecr=0 WS=128
74	TCP	8000 → 42014 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=627841533 TSecr=627841533
66	TCP	42014 → 8000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=627841533 TSecr=627841533
558	HTTP	GET / HTTP/1.1
66	TCP	8000 → 42014 [ACK] Seq=1 Ack=445 Win=65152 Len=0 TSval=627841670 TSecr=627841670
224	TCP	8000 → 42014 [PSH, ACK] Seq=1 Ack=445 Win=65536 Len=158 TSval=627841670 TSecr=627841670 [TCP segment of a reassembled PDU]
66	TCP	42014 → 8000 [ACK] Seq=445 Ack=159 Win=65488 Len=0 TSval=627841671 TSecr=627841671
378	HTTP	HTTP/2.0 200 OK (text/html)
66	TCP	42014 → 8000 [ACK] Seq=445 Ack=471 Win=65152 Len=0 TSval=627841671 TSecr=627841671
508	HTTP	GET /test.txt HTTP/1.1
173	HTTP	HTTP/2.0 304 Not Modified
66	TCP	42014 → 8000 [ACK] Seq=887 Ack=578 Win=65536 Len=0 TSval=627846568 TSecr=627846568
508	HTTP	GET /test.txt HTTP/1.1
173	HTTP	HTTP/2.0 304 Not Modified
66	TCP	42014 → 8000 [ACK] Seq=1529 Ack=685 Win=65536 Len=0 TSval=627847502 TSecr=627847502
508	HTTP	GET /test.txt HTTP/1.1
173	HTTP	HTTP/2.0 304 Not Modified
66	TCP	42014 → 8000 [ACK] Seq=2071 Ack=792 Win=65536 Len=0 TSval=627848374 TSecr=627848374
508	HTTP	GET /test.txt HTTP/1.1
173	HTTP	HTTP/2.0 304 Not Modified

(c) HTTP2.0

图 18b显示了HTTP1.1多次请求的结果。由于HTTP1.1支持持久连接，y因此只有一开始有三次握手，后面的请求都是直接使用相同的TCP连接发送数据。这样就大大减少了开销。为了让服务器判断是否需要继续保持连接，HTTP1.1也会发送Keep-Alive的请求头，让服务器判断是否需要保持连接。可以看到发送的信息中也有一些TCP Keep-Alive的信息，这些信息是用来维持TCP连接的。

图 18c显示了HTTP2多次请求的结果。由于HTTP2是基于TCP的，因此也是需要三次握手。HTTP2.0的多路复用机制，可以让多个请求共享一个TCP连接，因此可以看到所有请求都使用了客户机42014 端口打开的TCP连接。这大大节省了服务器的资源，也减少了开销。