实验一报告

姓名：钟赟
学号：2016K8009915009

一、互联网协议实验

实验内容

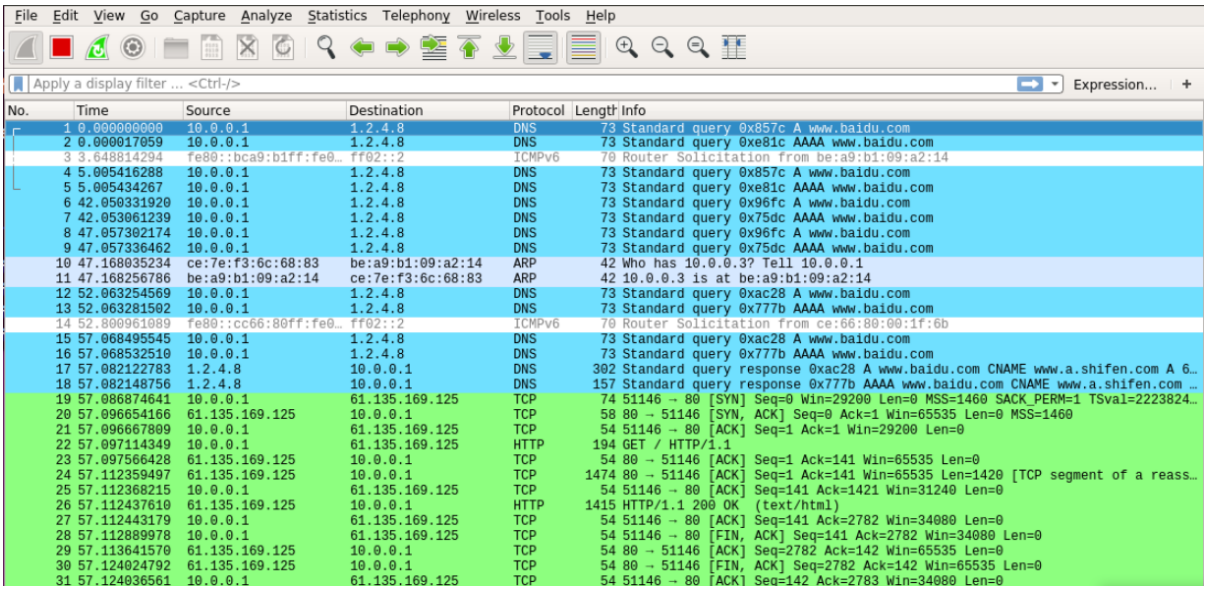
- 在节点h1上开启wireshark抓包，用wget下载www.baidu.com页面
- 调研说明wireshark抓到的几种协议：ARP, DNS, TCP, HTTP
- 调研解释h1下载baidu页面的整个过程：几种协议的运行机制

实验流程及结果分析

- **搭建实验环境**
  - 运行如下命令：

```
$ sudo apt install wireshark
$ sudo mn --nat
mininet> xterm h1
h1 # echo "nameserver 1.2.4.8" > /etc/resolv.conf
h1 # wireshark &
h1 # wget www.baidu.com
```

- **观察wireshark输出(一)**
  - 执行上述命令后，wireshark输出如下：

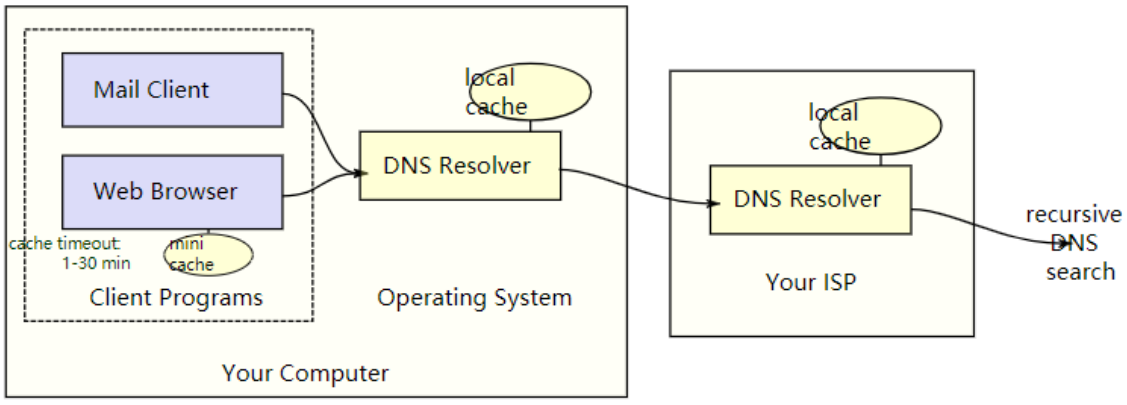


- 观察Protocol一列，可看到不同类型的传输协议，查阅相关资料了解如下：

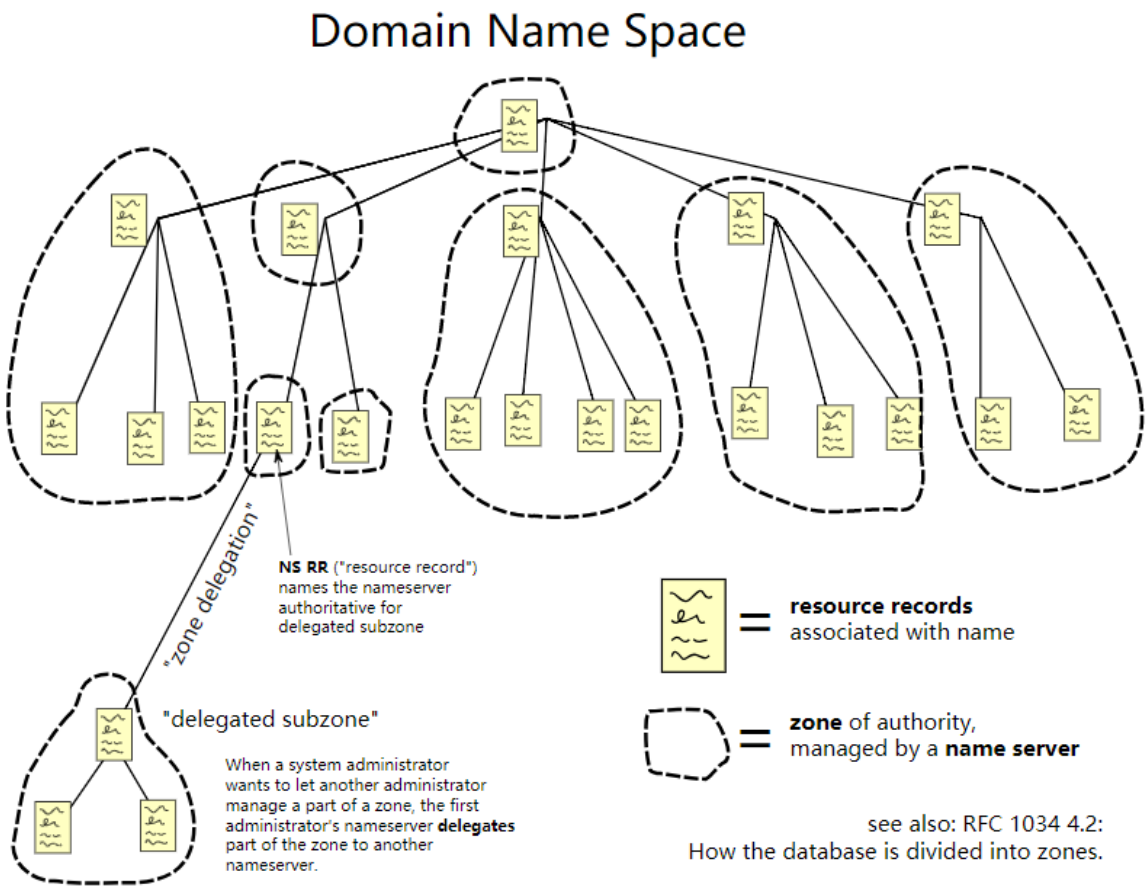
- **ARP** ARP全称为Address Resolution Protocol，即**地址解析协议**，用于实现从网络层使用的IP地址(如典型的IPv4地址)，解析出在数据链路层的MAC地址。
  ARP是一种"请求-响应"协议，其消息由数据链路层封装。它在单个网络内进行通信，不跨过结点或路由。ARP协议在主机ARP高速缓存中存放一个从IP地址到硬件地址的映射表，且此表经常动态更新（新增或超时删除）。^1 **请求应答原理**：在APR协议下，同一个局域网中，当PC1需要跟PC2进行通信时（此处以`PC1 ping PC2`为例），根据OSI数据封装顺序，PC1会自顶向下（从应用层到物理层）封装数据，发送给PC2。由于`ping`命令中只有PC2的IP地址，却没有MAC地址，因此在真正进行通信之前，PC1和PC2会进行一次ARP请求和回复过程：

  通过这个交互过程，PC1便具备了PC2的MAC地址信息，并将PC2的MAC信息放入本地的ARP缓存表。接下来，PC1再次进行数据封装，正式进入`ping`通信，发送数据包。^2

- **DNS** DNS(Domain Name System)指**域名系统**，用于把便于人们使用的域名转换为用于定位和识别计算机的IP地址。域名系统负责分配域名，并通过为每个域名制定权威的名称服务器将这些名称映射到互联网资源。网络管理员可以将其分配的名称空间的子域的权限委派给其他名称服务器。此机制提供分布式和容错服务。



  域名系统：域名系统用于查询域名对应的IP地址，其结构近似为树状结构，树的顶端到底端分别为：跟域->顶级域->次级域名->主机名。



请求应答原理：DNS分为Client和Server，Client利用一致的域名向Server询问，Server需要回答该域名的真正IP地址，在这个过程中需要进行域名解析过程。
域名解析总体可分为两个步骤，第一个步骤是本机向本地域名服务器发出一个DNS请求报文，报文里携带需要查询的域名；第二个步骤是本地域名服务器向本机回应一个DNS响应报文，里面包含域名对应的IP地址。第二个步骤中采用了递归查询和迭代查询。下图为DNS协议报文形式。^3

DNS协议报文格式

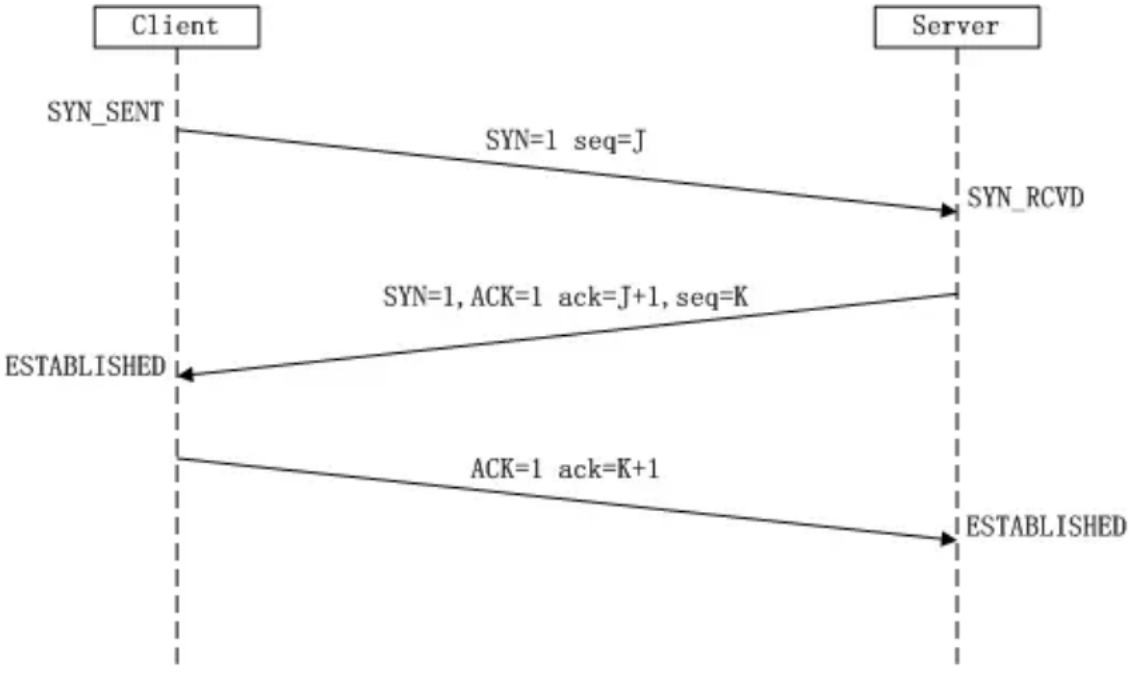
- **TCP** TCP(Transmission Control Protocol)为**传输控制协议**，在通过IP网络通信的主机上运行的应用程序之间提供可靠、有序和纠错的数据传递。主要的互联网应用程序, 如万维网、电子邮件、远程管理和文件传输, 依赖于 tcp。不需要可靠数据流服务的应用程序可以使用用户数据报协议 (udp), 该协议提供无连接数据报服务, 强调减少延迟、提高可靠性。 TCP常与IP一起工作，也被称为TCP/IP。TCP负责应用软件（比如你的浏览器）和网络软件之间的通信；IP负责计算机之间的通信。TCP 负责将数据分割并装入IP包，在它们到达的时候重新组合它们。下图为TCP报文形式：

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																	Urgent pointer (if URG set)														
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

其中，6位标志域的各个功能为：

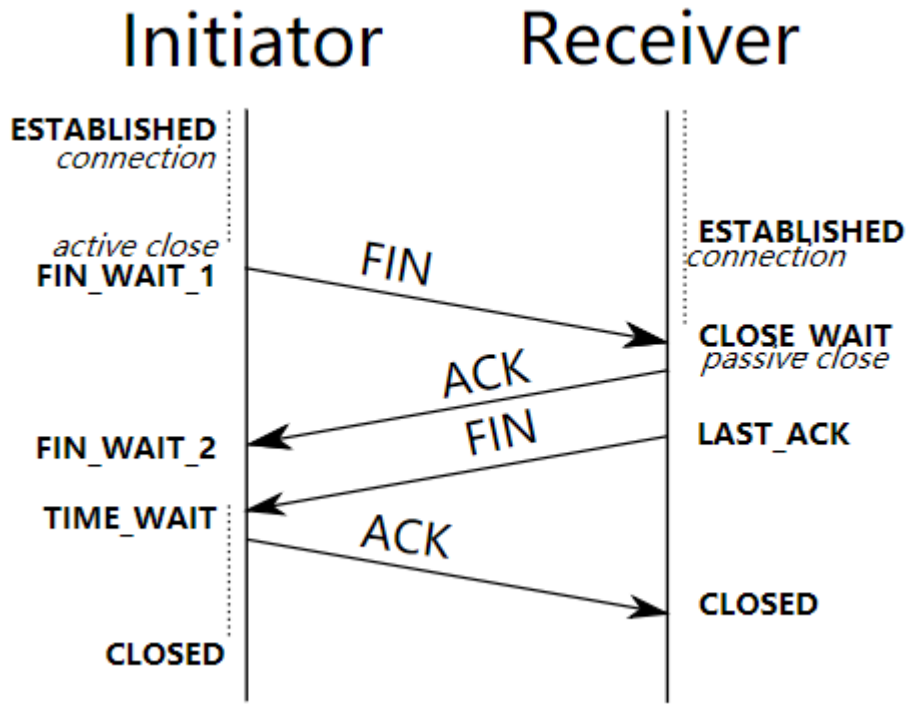
- URG：紧急标志。紧急标志为"1"表明该位有效。
- ACK：确认标志。表明确认编号栏有效。大多数情况下该标志位是置位的。TCP报头内的确认编号栏内包含的确认编号（w+1）为下一个预期的序列编号，同时提示远端系统已经成功接收所有数据。
- PSH：推标志。该标志置位时，接收端不将该数据进行队列处理，而是尽可能快地将数据转由应用处理。在处理Telnet或rlogin等交互模式的连接时，该标志总是置位的。
- RST：复位标志。用于复位相应的TCP连接。
- SYN：同步标志。表明同步序列编号栏有效。该标志仅在三次握手建立TCP连接时有效。
- FIN：结束标志。

TCP三次握手(Three-Way Handshake)：即建立TCP连接，就是指建立一个TCP连接时，需要客户端和服务端总共发送3个包以确认连接的建立。



- 1.建立连接时，客户端发送SYN包（SYN=i）到服务器，并进入到SYN-SEND状态，等待服务器确认。
- 2、服务器收到SYN包，必须确认客户的SYN（ack=i+1），同时自己也发送一个SYN包（SYN=k），即SYN+ACK包，此时服务器进入SYN-RECV状态。
- 3、客户端收到服务器的SYN+ACK包，向服务器发送确认包ACK（ack=k+1），此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手，客户端与服务器开始传送数据。

TCP四次挥手(Four-Way Wavehand)： 四次挥手即终止TCP连接，指断开一个TCP连接时，需要客户端和服务端总共发送4个包以确认连接的断开。



- 1.：Client发送一个FIN，用来关闭Client到Server的数据传送，Client进入FIN_WAIT_1状态。
- 2.Server收到FIN后，发送一个ACK给Client，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号），Server进入CLOSE_WAIT状态。
- 3.Server发送一个FIN，用来关闭Server到Client的数据传送，Server进入LAST_ACK状态。
- 4.Client收到FIN后，Client进入TIME_WAIT状态，接着发送一个ACK给Server，确认序号为收到序号+1，Server进入CLOSED状态，完成四次挥手。^4

- **HTTP** HTTP(Hypertext Transfer Protocol)即**超文本传输协议**，用于从万维网(www: World Wide Web)服务器传输超文本到本地浏览器的传送协议，它基于TCP/IP通信协议来传输数据(HTML文件、图片问价、查询结果等)，属于应用层的面向对象的协议。
  HTTP请求/响应过程

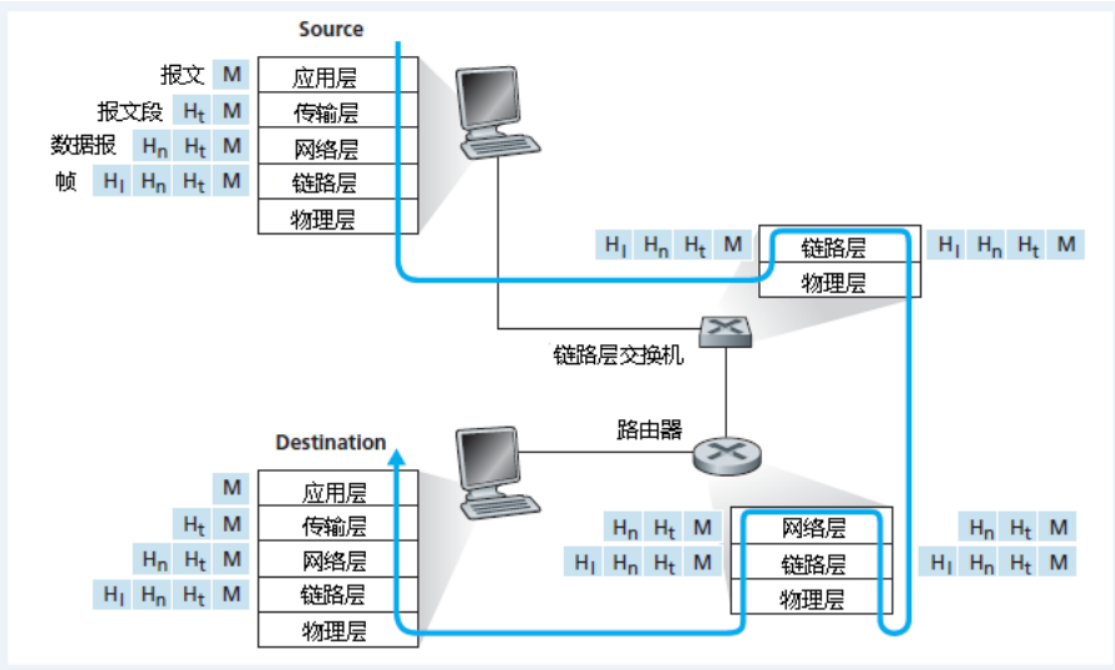
  - 客户端连接到Web服务器
  - 客户端向Web服务器发送HTTP请求
  - 服务器接收请求并返回HTTP响应
  - 服务器释放TCP连接
  - 客户端浏览器解析HTML内容并显示

- **观察wireshark输出(二)：不同层次的协议封装**

  - 根据TCP/IP五层协议，因特网的协议栈由从下到上5个层次组成：物理层、链路层、网络层、传输层和应用层。
    - **应用层** 应用层是网络应用程序以及它们的应用层协议存留的地方。应用层协议分布在多个端系统上，一个端系统中的应用程序使用协议与另外一个端系统中的应用程序交换信息的分组称为报文。
      因特网应用层包括许多协议，例如HTTP（Web应用的主要协议）、SMTP（邮件传输）、FTP（文件传送），和DNS（域名系统）等。

- **传输层** 因特网的传输层在应用程序端点之间传送应用层报文。在因特网中，有TCP和UDP两个传输层协议。
  TCP向它的应用程序提供了面向连接的服务，这种服务包括了应用层报文向目的地的确保传递和流量控制。TCP也将长报文划分成短报文，并提供拥塞控制机制，因此，当网络拥塞时，发送方可以抑制其传输速率。
  UDP协议向它的应用程序提供无连接服务。这是一种不提供不必要服务的服务，没有可靠性，没有流量控制，也没有拥塞控制。
  我们把传输层分组称为报文段。
- **网络层** 因特网的网络层负责将称为数据报的网络层分组从一台主机移动到另一台主机。在一台源主机中的因特网传输层协议(TCP或者UDP)向网络层递交传输层报文段和目的地址。网络层包括著名的IP协议，该协议定义了在数据报中的各个字段以及端系统和路由器如果作用于这些字段。网络层也包括决定路由的路由选择协议，它使得数据报根据该路由从源传输到目的地。
- **链路层** 链路层的任务是将整个帧从一个网络元素移动到邻近的网络元素。在每个结点，网络层将数据报下传给链路层，链路层沿着路径将数据报传递给下一个结点，在下一个结点链路层将数据报上传给网络层。
  由链路层提供的服务取决于应用于该链路的特定的链路层协议，比如我们常见的以太网，WIFI等。
  我们把链路层分组称为帧。
- **物理层** 物理层的任务是将该帧中的比特从一个结点移动到下一个结点。物理层的协议是和链路相关的，并且进一步与该链路的实际传输媒体相关。比如，以太网具有许多物理层协议：有关于双绞铜线、同轴电缆的，还有关于光纤的等等。

- 可以得出不同传输协议的的层级顺序：
  Ethernet < IP < UDP < DNS， Ethernet < IP < TCP < HTTP。
- **封装**
  - 封装是发送端对应用层数据一层一层加头的过程，传送到接收端后，接收端再一层一层去掉头部信息，然后将数据交给对应的应用程序。



  - 在发送主机端，一个**应用层报文**被传送给传输层。传输层收到报文之后，在报文上附上传输层首部信息，该首部信息将被接收端的传输层使用。应用层报文和传输层首部信息一起构成了**传输层报文段**。

- 传输层则向网络层传递该报文段，网络层增加了网络层首部信息，比如源和目的端系统的地址等，由此产生了**网络层数据报**。
- 数据报接下来被传递给链路层，链路层增加它自己的链路层首部信息，创建了**链路层帧**。
- 所以在每一层，一个分组都具有两种类型的字段：首部字段和有效载荷字段，其中有效载荷即来自于上一层的分组。^5

- **观察wireshark输出(三)：TCP承载HTTP协议**

  通过wireshark观察TCP的三次握手：

  - 在h1中用wget获取百度页面后，在wireshark中输入http过滤，可以看到wireshark接获到了三次握手的三个数据包，即TCP的三次握手过程：

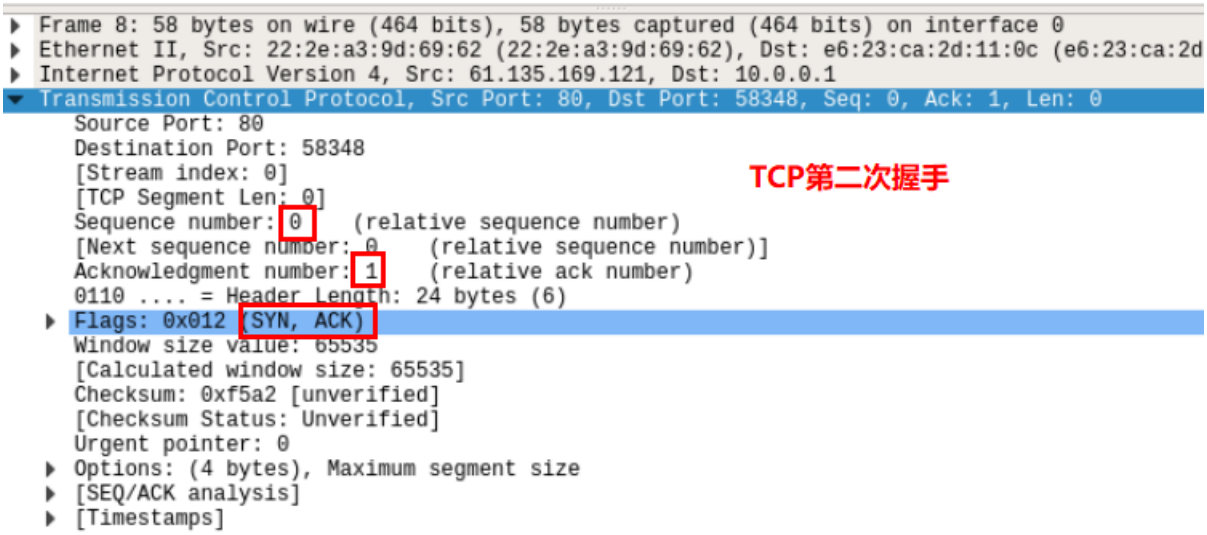
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	e6:23:ca:2d:11:0c	Broadcast	ARP	42	Who has 10.0.0.…
2	0.001401471	22:2e:a3:9d:69:62	e6:23:ca:2d:11:0c	ARP	42	10.0.0.3 is at …
3	0.001405679	10.0.0.1	1.2.4.8	DNS	73	Standard query …
4	0.001407098	10.0.0.1	1.2.4.8	DNS	73	Standard query …
5	0.013453255	1.2.4.8	10.0.0.1	DNS	302	Standard query …
6	0.013755439	1.2.4.8	10.0.0.1	DNS	157	Standard query …
7	0.014326651	10.0.0.1	61.135.169.121	TCP	74	58348 → 80 [SYN…
8	0.035541855	61.135.169.121	10.0.0.1	TCP	58	80 → 58348 [SYN…
9	0.035555097	10.0.0.1	61.135.169.121	TCP	54	58348 → 80 [ACK…
10	0.035589556	10.0.0.1	61.135.169.121	HTTP	194	GET / HTTP/1.1
11	0.036146058	61.135.169.121	10.0.0.1	TCP	54	80 → 58348 [ACK…
12	0.044110066	61.135.169.121	10.0.0.1	TCP	1474	80 → 58348 [ACK…
13	0.044121051	10.0.0.1	61.135.169.121	TCP	54	58348 → 80 [ACK…
14	0.044411428	61.135.169.121	10.0.0.1	HTTP	1415	HTTP/1.1 200 OK…
15	0.044417123	10.0.0.1	61.135.169.121	TCP	54	58348 → 80 [ACK…
16	0.052175158	10.0.0.1	61.135.169.121	TCP	54	58348 → 80 [FIN…
17	0.052944525	61.135.169.121	10.0.0.1	TCP	54	80 → 58348 [ACK…
18	0.056665567	61.135.169.121	10.0.0.1	TCP	54	80 → 58348 [FIN…
19	0.056678094	10.0.0.1	61.135.169.121	TCP	54	58348 → 80 [ACK…
20	5.056809072	22:2e:a3:9d:69:62	e6:23:ca:2d:11:0c	ARP	42	Who has 10.0.0.…
21	5.056818970	e6:23:ca:2d:11:0c	22:2e:a3:9d:69:62	ARP	42	10.0.0.1 is at …

    可以看到在TCP三次握手后，第四个包才是HTTP的，说明HTTP是利用TCP建立连接的。

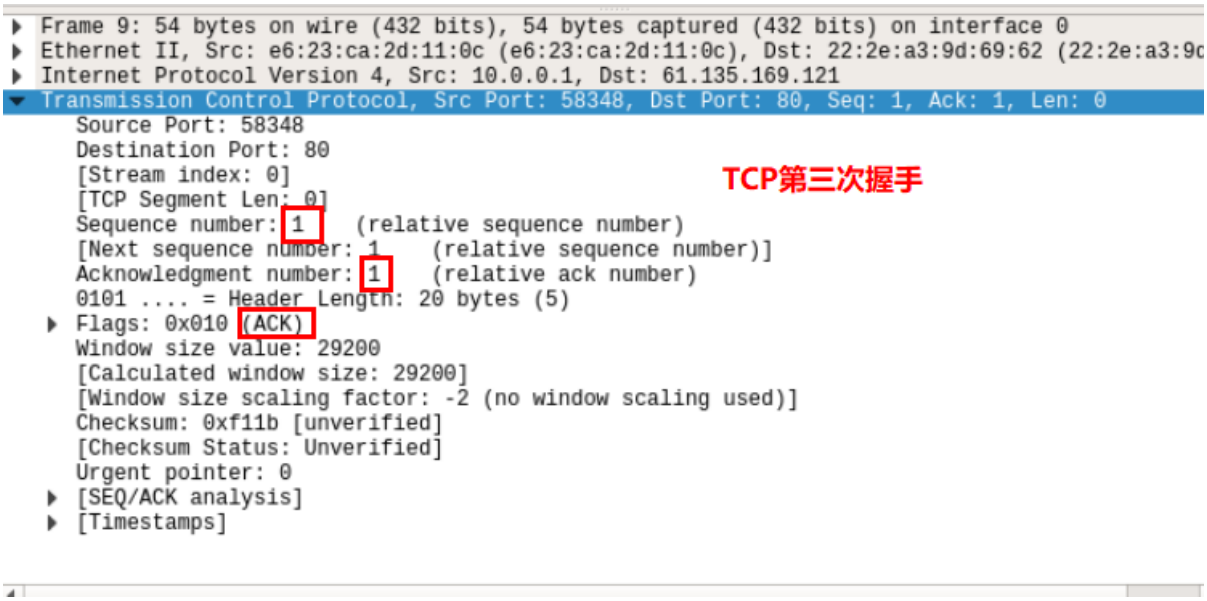
  - **第一次握手数据包** 客户端发送一个TCP，标志位为SYN，序列号为0， 代表客户端请求建立连接。如下图：

```
Frame 7: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
Ethernet II, Src: e6:23:ca:2d:11:0c (e6:23:ca:2d:11:0c), Dst: 22:2e:a3:9d:69:62 (22:2e:a3:9d:
Internet Protocol Version 4, Src: 10.0.0.1, Dst: 61.135.169.121
Transmission Control Protocol, Src Port: 58348, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 58348
    Destination Port: 80
    [Stream index: 0]
    [TCP Segment Len: 0]                                  TCP第一次握手
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 0
    1010 .... = Header Length: 40 bytes (10)
    Flags: 0x002 (SYN)
    Window size value: 29200
    [Calculated window size: 29200]
    Checksum: 0xf12f [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
    Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP),
    [Timestamps]
```

  - **第二次握手数据包** 服务器发回确认包，标志位为 SYN，ACK。将确认序号(Acknowledgement Number)设置为客户的ISN加1，即0+1=1，如下图：



- **第三次握手数据包** 客户端再次发送确认包(ACK)，SYN标志位为0，ACK标志位为1，并且把服务器发来ACK的序号字段+1，放在确定字段中发送给对方，如下图:



这样通过TCP的三次握手，建立了连接，开始传输数据。

二、流完成时间实验

实验内容

- 利用fct_exp.py脚本复现P25中的图
- 调研解释图中的现象(TCP传输、慢启动机制)

实验步骤

- 搭建实验环境

```
$ sudo python fct_exp.py
mininet> xterm h1 h2
h2 # dd if=/dev/zero of=1MB.dat bs=1M count=1
h1 # wget http://10.0.0.2/1MB.dat
```

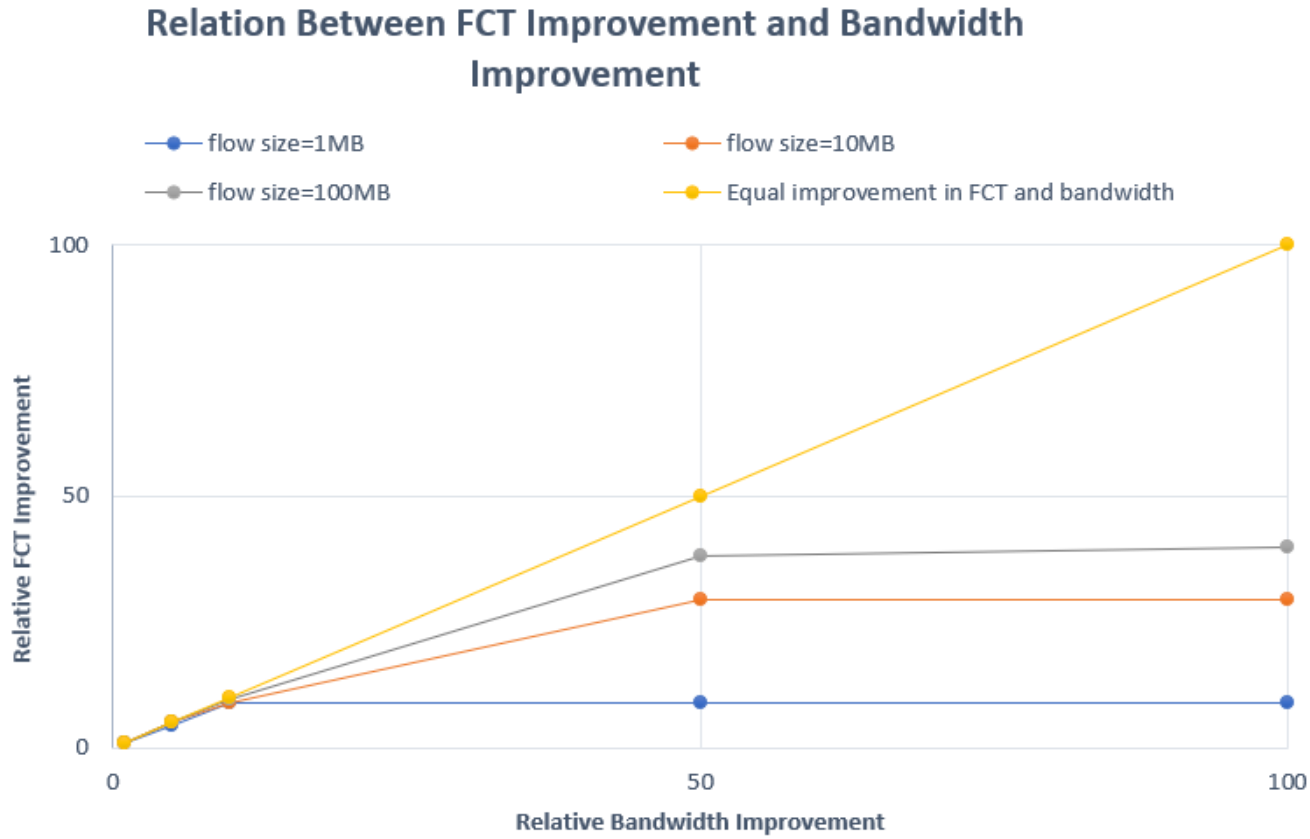
- 利用fct_exp.py脚本,在给定带宽、延迟和文件大小前提下，查看流完成时间。变化文件大小(10MB, 100MB)、带宽(10Mbps, 50Mbps, 100Mbps, 500Mbps, 1Gbps)，查看不同条件下的流完成时间。

结果分析

不同条件下的流完成时间数据如下表：

文件大小(MB)/完成时间(s)/带宽(Mbps)	10	50	100	500	1000
1	0.9	0.2	0.1	0.1	0.1
10	8.8	1.8	1.0	0.3	0.3
100	88	18	9.1	2.3	2.2

利用上表数据得到带宽提高速率与流完成速度提高速率的关系图：



可知两者的关系并不成正比。这与TCP的拥塞控制机制有关。

- 为了防止网络拥塞现象，TCP提出了“慢启动(Slow Start)”和“拥塞避免(Congestion Avoidance)”机制，其主要原理依赖于拥塞窗口(cwnd)来控制。窗口大小代表能够发送出去但还未收到ACK的最大数据报文段，故窗口越大数据传输速率也越大，但也越容易产生网络拥塞。

- TCP还拥有一个对端通告的接收窗口(rwnd)用于流量控制，TCP真正的发送窗口=min(rwnd, cwnd)。由于rwnd是由对端确定的，不受网络环境的影响，所以考虑拥塞时只考虑cwnd的大小。在TCP中cwnd以字节为单位，我们假设TCP每次传输都是按照MSS(最大报文段)大小来发送数据，因此可以认为cwnd以数据包个数为单位，cwnd增加1相当于字节数增加一个MSS大小。

- **TCP的慢启动机制**：为避免拥塞，TCP在刚开始连接建立成功后不能向网络中发送大量的数据包，只能根据网络情况逐步增加每次发送的数据量。当连接新建立时，cwnd初始化为1个MSS大小，发送端开始按照拥塞窗口大小发送数据，每当有一个报文段被确认，cwnd就增加1个MSS大小，因此cwnd的值随着**网络往返时间**(Round Trip Time, RTT)呈指数级增长。简单计算下：

  > 开始时刻 ----> cwnd = 1
  > 经过1个RTT ----> cwnd = 2 * 1 = 2
  > 经过2个RTT ----> cwnd = 2 * 2 = 4
  > 经过3个RTT ----> cwnd = 2 * 4 = 8

- 设**带宽**为W，假设经过T时间可以占满带宽，有：
  2^(T/RTT) = W
  故 **T = RTT * log2 W**
  上式解释了图中传输速率不随带宽增大而线性增长的现象。当RTT一定时，传输速率随着带宽的增长**对数级**提升。

- 同时，cwnd不能无限制增长下去，TCP使用了一个**慢启动门限(ssthresh)**变量，**当cwnd超过该变量的值后，慢启动过程结束，进入拥塞避免阶段。因此，对于图中不同的flow size，传输的文件越大，其慢启动过程所占数据传输总时间的比例越小**，使用最大带宽传输的时间**越长**，故传输速率增长越快。^6

参考资料

参考链接已附于文中。