Mininet 实验环境报告

李昊宸

2017K8009929044

（一）互联网协议实验

一、实验内容

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

二、实验流程

1. 搭建实验环境

$ sudo mn --nat
mininet> xterm h1
h1 # echo "nameserver 1.2.4.8" > /etc/resolv.conf

2. 启动 wireshark

h1 # wireshark &

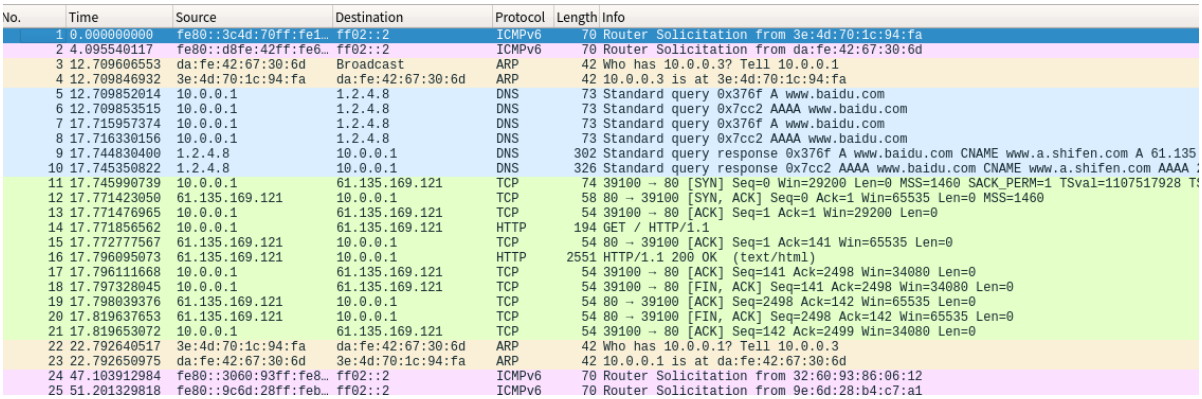
3. 抓取网站

h1 # wget www.baidu.com

三、实验结果及分析

1. 实验结果

图一 xterm 下 h1 的保存结果



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::3c4d:70ff:fe1…	ff02::2	ICMPv6	70	Router Solicitation from 3e:4d:70:1c:94:fa
2	4.095540117	fe80::d8fe:42ff:fe6…	ff02::2	ICMPv6	70	Router Solicitation from da:fe:42:67:30:6d
3	12.709606553	da:fe:42:67:30:6d	Broadcast	ARP	42	Who has 10.0.0.3? Tell 10.0.0.1
4	12.709846932	3e:4d:70:1c:94:fa	da:fe:42:67:30:6d	ARP	42	10.0.0.3 is at 3e:4d:70:1c:94:fa
5	12.709852014	10.0.0.1	1.2.4.8	DNS	73	Standard query 0x376f A www.baidu.com
6	12.709853515	10.0.0.1	1.2.4.8	DNS	73	Standard query 0x7cc2 AAAA www.baidu.com
7	17.715957374	10.0.0.1	1.2.4.8	DNS	73	Standard query 0x376f A www.baidu.com
8	17.716330156	10.0.0.1	1.2.4.8	DNS	73	Standard query 0x7cc2 AAAA www.baidu.com
9	17.744830400	1.2.4.8	10.0.0.1	DNS	302	Standard query response 0x376f A www.baidu.com CNAME www.a.shifen.com A 61.135
10	17.745350822	1.2.4.8	10.0.0.1	DNS	326	Standard query response 0x7cc2 AAAA www.baidu.com CNAME www.a.shifen.com AAAA
11	17.745990739	10.0.0.1	61.135.169.121	TCP	74	39100 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1107517928 T
12	17.771423050	61.135.169.121	10.0.0.1	TCP	58	80 → 39100 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
13	17.771476965	10.0.0.1	61.135.169.121	TCP	54	39100 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
14	17.771856562	10.0.0.1	61.135.169.121	HTTP	194	GET / HTTP/1.1
15	17.772777567	61.135.169.121	10.0.0.1	TCP	54	80 → 39100 [ACK] Seq=1 Ack=141 Win=65535 Len=0
16	17.796095073	61.135.169.121	10.0.0.1	HTTP	2551	HTTP/1.1 200 OK (text/html)
17	17.796111668	10.0.0.1	61.135.169.121	TCP	54	39100 → 80 [ACK] Seq=141 Ack=2498 Win=34080 Len=0
18	17.797328045	10.0.0.1	61.135.169.121	TCP	54	39100 → 80 [FIN, ACK] Seq=141 Ack=2498 Win=34080 Len=0
19	17.798039376	61.135.169.121	10.0.0.1	TCP	54	80 → 39100 [ACK] Seq=2498 Ack=142 Win=65535 Len=0
20	17.819637653	61.135.169.121	10.0.0.1	TCP	54	80 → 39100 [FIN, ACK] Seq=2498 Ack=142 Win=65535 Len=0
21	17.819653072	10.0.0.1	61.135.169.121	TCP	54	39100 → 80 [ACK] Seq=142 Ack=2499 Win=34080 Len=0
22	22.792640517	3e:4d:70:1c:94:fa	da:fe:42:67:30:6d	ARP	42	Who has 10.0.0.1? Tell 10.0.0.3
23	22.792650975	da:fe:42:67:30:6d	3e:4d:70:1c:94:fa	ARP	42	10.0.0.1 is at da:fe:42:67:30:6d
24	47.103912984	fe80::3060:93ff:fe8…	ff02::2	ICMPv6	70	Router Solicitation from 32:60:93:86:06:12
25	51.201329818	fe80::9c6d:28ff:feb…	ff02::2	ICMPv6	70	Router Solicitation from 9e:6d:28:b4:c7:a1

图二 wireshark 抓取的报文

2. 实验分析

从 wireshark 抓取的报文中可以看出，整个过程中共有四种类型的报文：ARP、DNS、TCP 和 HTTP。下面逐一阐述这四种协议。

ARP 协议：Address Resolution Protocol，地址解析协议。在以太网中，一个主机要和另一个主机进行直接通信，必须要知道目标主机的 MAC 地址。网络层中，实际传输的是“帧”，帧内含有传送目标主机的 MAC 地址，该地址通过地址解析协议获得的。也就是说，ARP 协议的基本功能就是通过目标设备的 IP 地址（32 位），查询目标设备的 MAC 地址（48 位），以保证通信的顺利进行。

假设我们的计算机 IP 地址是 10.0.0.1，要执行这个命令：ping 10.0.0.2。该命令会通过 ICMP 协议发送 ICMP 数据包。该过程需要经过下面的步骤：

1. 应用程序构造数据包，该示例是产生 ICMP 包，被提交给内核（网络驱动程序）
2. 内核检查是否能够转化该 IP 地址为 MAC 地址，也就是在本地的 ARP 缓存中查看 IP-MAC 对应表
3. 如果存在该 IP-MAC 对应关系，那么跳到步骤 7；如果不存在该 IP-MAC 对应关系，那么接续下面的步骤
4. 内核进行 ARP 广播，目的地的 MAC 地址是 FF-FF-FF-FF-FF-FF，ARP 命令类型为 REQUEST（1），其中包含有自己的 MAC 地址
5. 当 10.0.0.2 主机接收到该 ARP 请求后，就发送一个 ARP 的 REPLY（2）命令，其中包含自己的 MAC 地址
6. 本地获得 10.0.0.2 主机的 IP-MAC 地址对应关系，并保存到 ARP 缓存中
7. 内核将把 IP 转化为 MAC 地址，然后封装在以太网头结构中，再把数据发送出去

在实验中，我们需要获得一个网站的内容，所以该网站的 MAC 地址起初并不在主机的 ARP 缓存中，所以抓取到两个 ARP 报文，一个为发送报文，一个为回应报文。

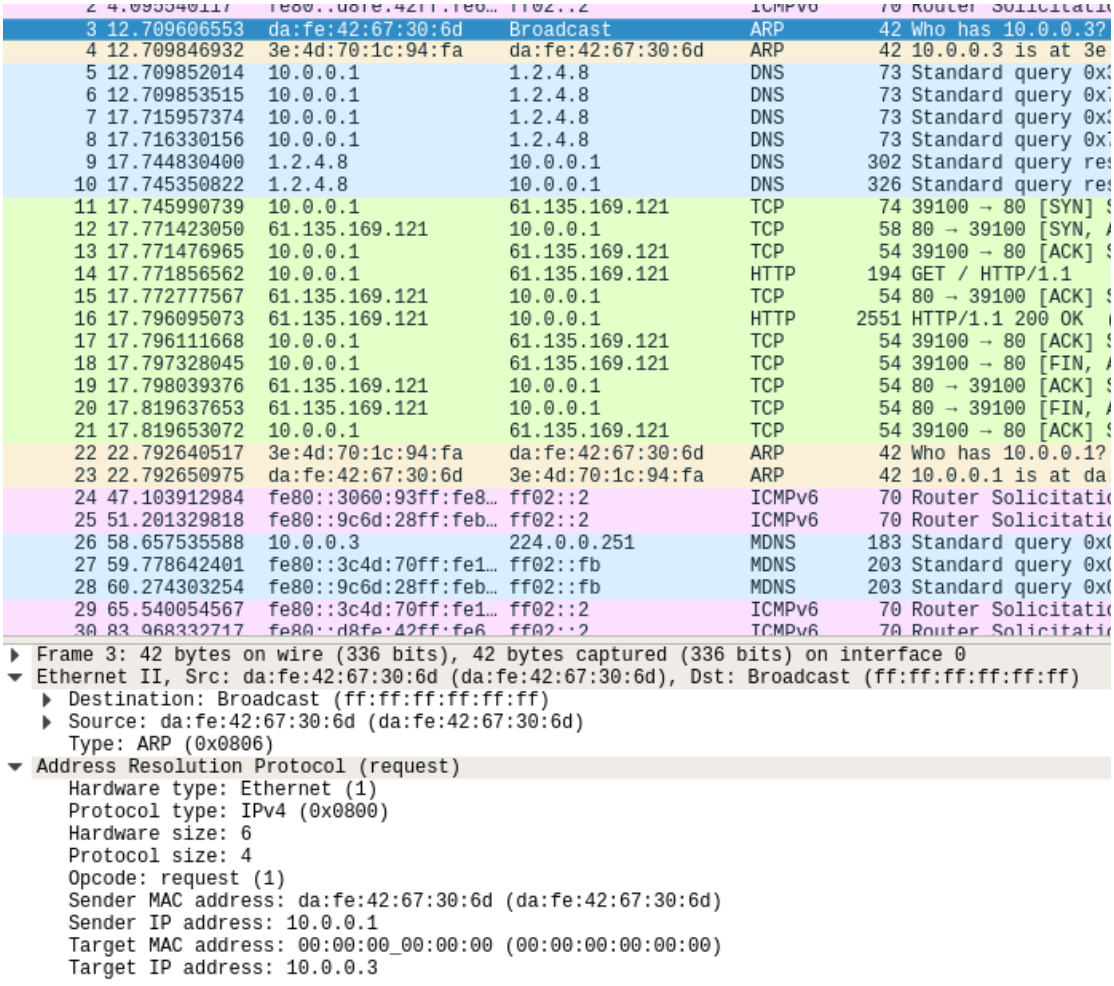
发送报文：ARP 请求包：

opcode 为 request（1），发送方为 h1（IP 地址 10.0.0.1），传输方式为广播 Broadcast。

ARP 回应包：

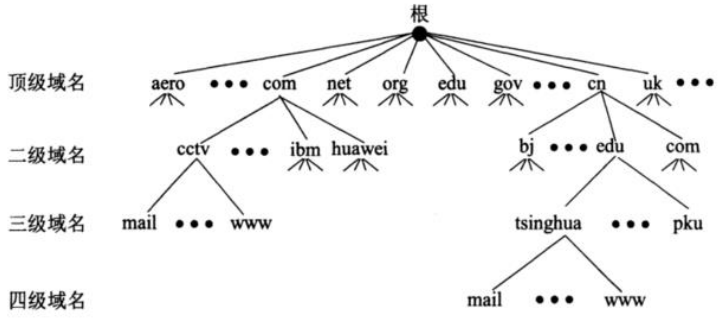
opcode 为 request（2），发送方的 mac 地址为 3e:4d:70:1c:94:fa,表明 h1 想询问的 mac 地址。目标方 IP 地址为 10.0.0.1，即发送请求的主机 h1，传输

方式为单播，



图三 ARP 报文 Ethernet < ARP

DNS 协议：Domain Name System，域名系统。该系统用于命名组织到域层次结构中的计算机和网络服务。域名是由圆点分开一串单词或缩写组成的，每一个域名都对应一个惟一的 IP 地址，在 Internet 上域名与 IP 地址之间是一一对应的，DNS 就是进行域名解析的服务器。



图四 域名分级

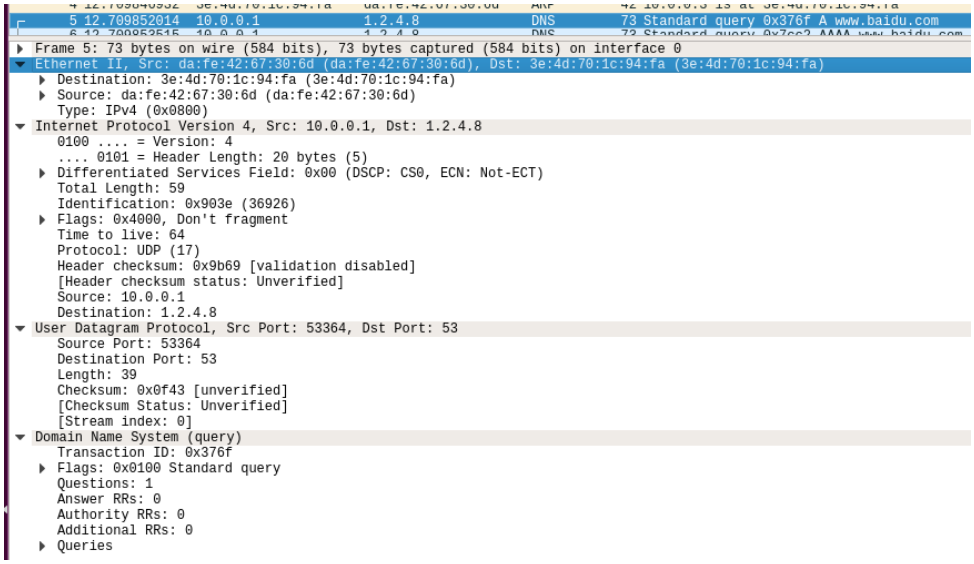
根域名服务器：最高层次的域名服务器，本地域名服务器解析不了的域名就会向其求助
顶级域名服务器：负责管理在该顶级域名服务器下注册的二级域名
权限域名服务器：负责一个区域的域名解析工作

本地域名服务器：当一个主机发出 DNS 查询请求时，这个查询请求首先发给本地域名服务器

下面以一个例子说明在无缓存的情况下，通过 DNS 来查询域名的步骤。假设域名为 m.xyz.com 的主机想要查询 y.abc.com 这个域名对应的 IP 地址，那么它就会按照以下方式方式进行查询：

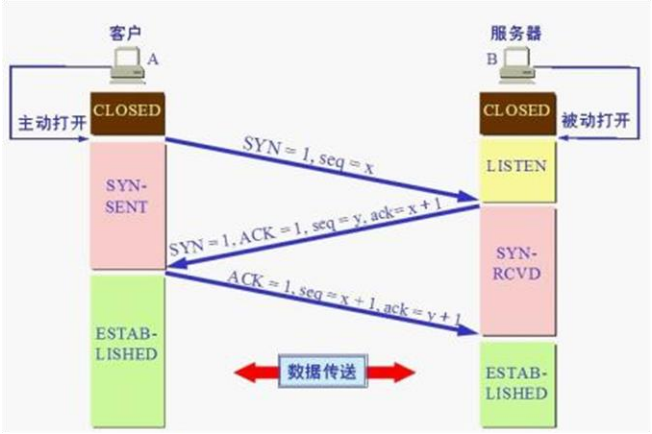
1. 主机 m.xyz.com 先向本地域名服务器 dns.xyz.com 进行递归查询。
2. 本地域名服务器无法给出 IP 地址，所以本地域名服务器向（离自己最近的）根域名服务器查询，这是的查询已经变为了迭代查询。
3. 根域名服务器根据本地域名服务器发送的报文，知道了下一步应该查询的是哪个顶级域名服务器，这时根域名服务器告诉本地域名服务器，下一步应该查询的顶级域名服务器 dns.com 的 IP 地址。
4. 本地域名服务器向顶级域名服务器 dns.com 发送请求查询。
5. 顶级域名服务器 dns.com 告诉本地域名服务器，下一步应该查询的权限域名服务器 dns.abc.com 的 IP 地址。
6. 本地域名服务器向权限域名服务器 dns.abc.com 发送请求查询。
7. 权限域名服务器 dns.abc.com 告诉本地域名服务器想要查询的域名 y.abc.com 的 IP 地址。
8. 本地域名服务器在拿到 IP 地址后，将 IP 地址返回给主机 m.xyz.com。

但是，通常情况下，本地域名服务器中会有常用域名的缓存，比如本例中的 www.baidu.com，所以可以直接将 IP 地址返回。



图五 DNS 报文 Ethernet < IP < UDP < DNS

TCP 协议：Transmission Control Protocol，传输控制协议。TCP 层是位于 IP 层之上，应用层之下的传输层。应用层向 TCP 层发送用于网间传输的、用 8 位字节表示的数据流，然后 TCP 把数据流分割成适当长度的报文段。之后 TCP 把结果包传给 IP 层，由它来通过网络将包传送给接收端实体的 TCP 层。TCP 为了保证不发生丢包，就给每个字节一个序号，同时序号也保证了传送到接收端实体的包的按序接收。然后接收端实体对已成功收到的字节发回一个相应的确认（ACK）；如果发送端实体在合理的往返时延（RTT）内未收到确认，那么对应的数据（假设丢失了）将会被重传。TCP 用一个校验和函数来检验数据是否有错误；在发送和接收时都要计算和校验。



图六 TCP 建立连接的三次握手

TCP 建立连接：三次握手

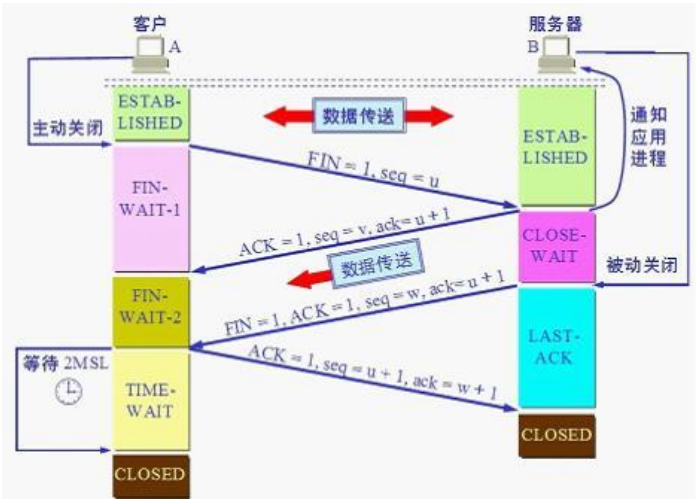
1. A 的 TCP 向 B 发出连接请求报文段,其首部中的同步位 SYN = 1,并选择序号 seq = x,表明传送数据时的第一个数据字节的序号是 x

2. B 的 TCP 收到连接请求报文段后,如同意,则发回确认(B 在确认报文段中应使 SYN = 1,使 ACK = 1,其确认号 ack = x + 1,自己选择的序号 seq = y)

3. A 收到此报文段后向 B 给出确认,其 ACK = 1,确认号 ack = y + 1(A 的 TCP 通知上层应用进程,连接已经建立,B 的 TCP 收到主机 A 的确认后,也通知其上层应用进程：TCP 连接已经建立)

该过程对应图三的 11-13 号报文。

TCP 协议的数据传送使用滑动窗口方式进行传送。在此不再赘述。



图七 TCP 释放连接的四次握手

TCP 释放连接：四次握手

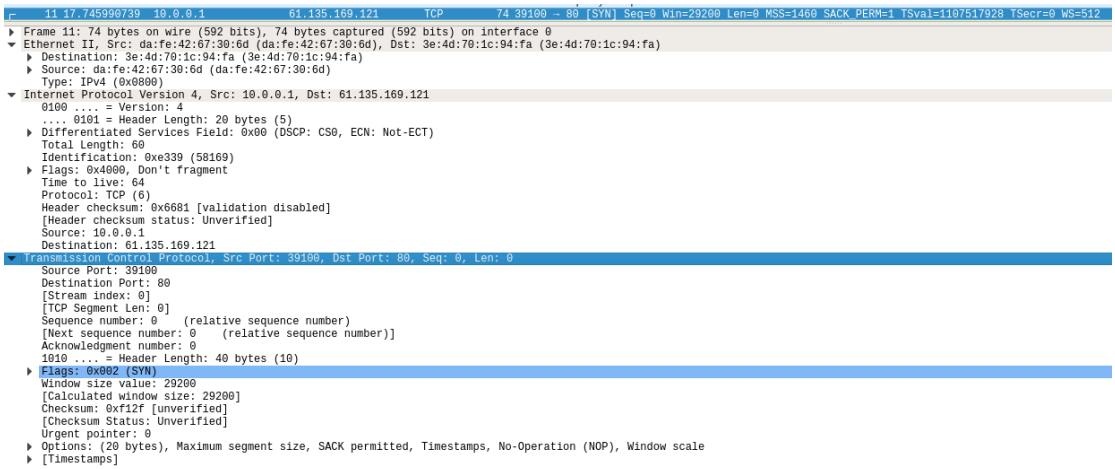
1. 数据传输结束后,通信的双方都可释放连接.现在 A 的应用进程先向其 TCP 发出连接释放报文段,并停止再发送数据,主动关闭 TCP 连接(A 把连接释放报文段首部的 FIN = 1,其序号 seq = u,等待 B 的确认)

2. B 发出确认,确认号 ack = u+1,而这个报文段自己的序号 seq = v(TCP 服务器进程通知高层应用进程.从 A 到 B 这个方向的连接就释放了,TCP 连接处于半关闭状态.B 若发送数据,A 仍要接收)

3. 若 B 已经没有要向 A 发送的数据,其应用进程就通知 TCP 释放连接

4. A 收到连接释放报文段后,必须发出确认,在确认报文段中 ACK = 1,确认号 ack=w + 1,自己的序号 seq = u + 1

该过程对应图三的 18-21 号报文。

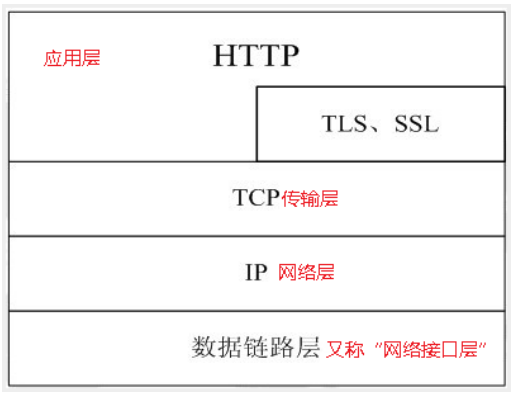


图八 TCP 报文 Ethernet < IP < TCP

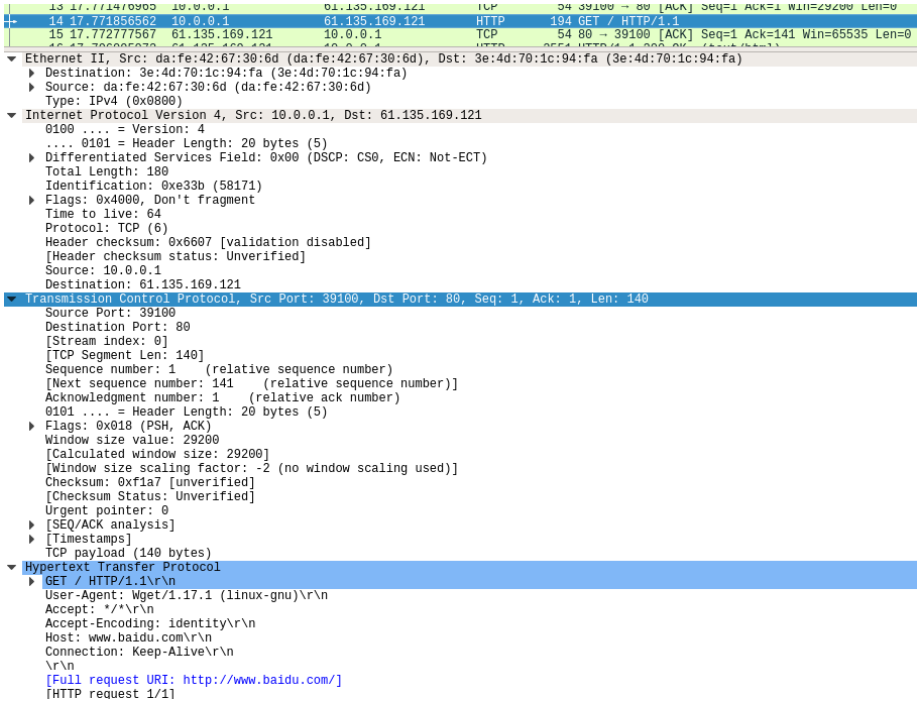
HTTP 协议：Hyper Text Transfer Protocol，超文本传输协议。是用于从 WWW 服务器传输超文本到本地浏览器的传送协议。

一次 HTTP 操作称为一个事务，其工作过程可分为四步：

1. 首先客户机与服务器需要建立连接。只要单击某个超级链接，HTTP 的工作开始。
2. 建立连接后，客户机发送一个请求给服务器，请求方式的格式为：统一资源标识符（URL）、协议版本号，后边是 MIME 信息包括请求修饰符、客户机信息和可能的内容。
3. 服务器接到请求后，给予相应的响应信息，其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码，后边是 MIME 信息包括服务器信息、实体信息和可能的内容。
4. 客户端接收服务器所返回的信息通过浏览器显示在用户的显示屏上，然后客户机与服务器断开连接。



图九 HTTP 模型



图十 HTTP 报文 Ethernet < IP < TCP < HTTP

图十一 追踪到的 TCP 流

红色区域为源到目的地，本次实验中为主机 h1，使用 http1.1 协议，目的主机为百度。

蓝色区域为目的地到源，本次实验中目的主机为百度的服务器，连接方式为 text/html，接下来的蓝色区域就是将百度页面的 html 编码发送给 h1 主机。

整个过程：

1. h1 发现 www.baidu.com 不在本地局域网内，于是通过 ARP 协议获取路由器的 MAC 地址。h1 发送广播，询问路由器的 MAC 地址，路由器接收到包后将其 MAC 地址返回 h1。
2. h1 通过 DNS 协议将 www.baidu.com 该域名转化为 IP 地址。
3. h1 将 MAC 地址和 IP 地址封装到帧中，通过 TCP 协议与百度服务器建立连接。
4. h1 通过 http 协议获取数据

（二）流完成时间实验

一、实验内容

1. 在给定带宽、延迟和文件大小前提下，查看流完成时间

2. 变化文件大小(10MB, 100MB)、带宽(10Mbps, 100Mbps, 1Gbps)、延迟(10ms, 100ms), 查看不同条件下的流完成时间

二、实验流程

1. 搭建实验环境

$ sudo python fct_exp.py

mininet> xterm h1 h2

2. 生成传输文件

h2 # dd if=/dev/zero of=1MB.dat bs=1M count=1

3. 发起传输

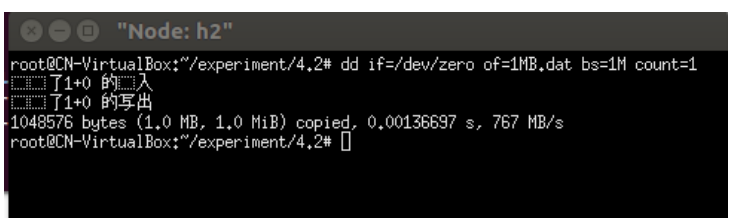
h1 # wget http://10.0.0.2/1MB.dat

三、实验结果及分析

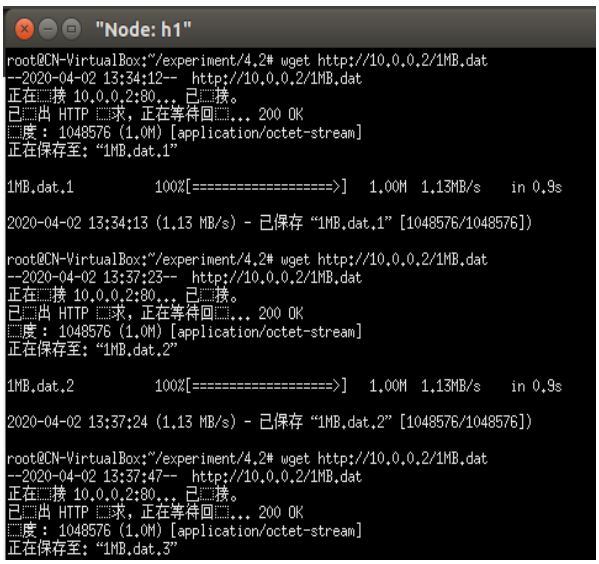
1. 实验结果

	A	B	C	D	E	F	G	H
		文件大小						
		1MB		10MB		100MB		
带宽		10ms	100ms	10ms	100ms	10ms	100ms	延迟
10Mbps		0.9s, 1.6s, 2.2, 2.3s, 2.4s	2.2s, 3.1s, 3.3s, 4.2s, 4.3s	10s, 11s, 12s, 16s, 18s	12s, 13s, 17s, 20s, 22s	121s, 127s, 135s, 140s, 150s	140s, 153s, 159s, 162s, 180s	
平均		1.88s | 545KB/S	3.42s | 299KB/S	13.4s | 764KB/S	16.8s | 610KB/S	134s | 764KB/S	158s | 648KB/S	
100Mbps		0.3s, 0.3s, 0.4s, 0.6s, 0.9s	1.2s, 1.2s, 1.5s, 2.1s, 3.2s	1.1s, 1.6s, 2.1s, 2.8s, 2.9s	7.5s, 9.5s, 11s, 12s, 13s	11s, 15s, 15s, 16s, 24s	12s, 26s, 26s, 38s, 47s	
平均		0.5s |2048KB/S	1.84s | 557KB/S	2.1s | 4876KB/S	10.6s | 966KB/S	16.2s | 6320KB/S	29.8s | 3436KB/S	
1Gbps		0.3s, 0.3s, 0.4s, 0.5s, 0.5s	1.2s, 1.3s, 1.5s, 1.7s, 2.8s	0.5s, 0.6s, 0.8s, 1.7s, 2.0s	5.4s, 5.4s, 6.9s, 9.2s, 12s	2.6s, 4.9s, 5.5s, 5.5s, 6.8s	7.1s, 16s, 21s, 23s, 28s	
平均		0.4s | 2048KB/S	1.7s | 602KB/S	1.1s | 9309KB/S	7.8s | 1312KB/S	5.1s | 20078KB/S	19s | 5390KB/S	

图十二 流传输时间记录



图十三 生成传输文件示例



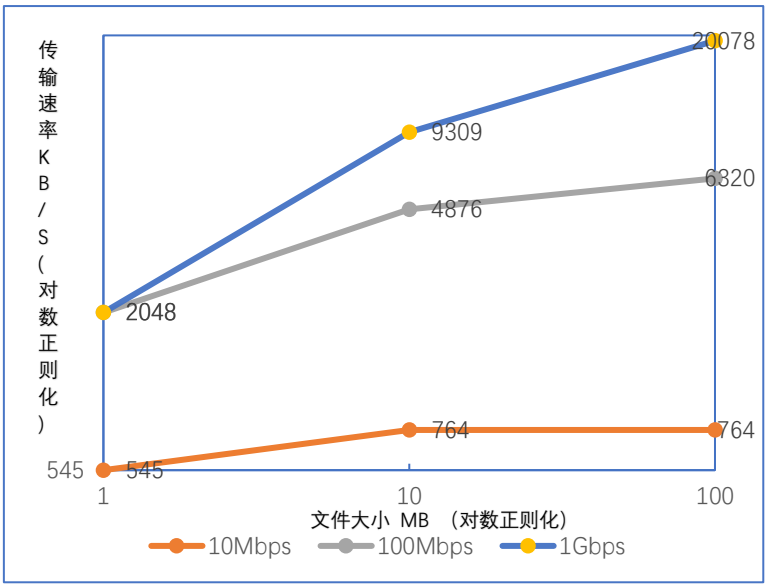
图十四 发起传输示例

2. 实验分析

1. 10ms 延迟

10ms 延迟	文件大小		
带宽	1MB	10MB	100MB
10Mbps	545KB/s	764 KB/s	764 KB/s
100Mbps	2048 KB/s	4876 KB/s	6320 KB/s
1Gbps	2048 KB/s	9309 KB/s	20078 KB/s

表一 10ms 延迟下的平均传输速率表



图十五 10ms 延迟下的传输速率-文件大小双对数图

分析：

作图使用的数据为表格中的数据。首先，使用了传输速率作为流完成时间的替代，这样一来数据更加直观。其次，为了放大数据之间的直观程度，对横轴和纵轴均取了对数坐标，纵坐标的底数为 545，即针对第一个点的传输速率进行正则化；横坐标以 10 为底数，即针对第一个点的文件大小进行正则化；随后移动坐标轴，使原点位于第一个点处，做出传输速率-文件大小的双对数正则化折线图。需要说明的是，为了数据的直观性，坐标轴上标注的值并非取对数之后的值，而为取对数之前的值。

现象复现了课件中要求的目标，即在延迟固定、带宽固定的情况下，增大文件大小，传输速率呈现出增长速率逐渐减小的增长，最终会逐渐逼近带宽的理论值。

解释：

一部分原因是 TCP 传输需要经历建立传输连接的三次握手阶段和释放连接的四次握手阶段，该部分需要占用一定固定的时长，文件越小，该部分时间占整个传输时间的比重越大，导致直观上的传输速率的下降。

另一个原因原因是 TCP 的慢启动机制。TCP 协议为了做到效率与可靠性的统一，设计了慢启动（slow start）机制。开始的时候，发送得较慢，然后根据丢包的情况，调整速率：如果不丢包，就加快发送速度；如果丢包，就降低发送速度。

Linux 内核里面设定了（常量 TCP_INIT_CWND），刚开始通信的时候，发送方一次性发送 10 个数据包，即"发送窗口"的大小为 10。然后停下来，等待接收方的确认，再继续发送。

默认情况下，接收方每收到两个 TCP 数据包，就要发送一个确认消息。发送方有了这两个信息，再加上自己已经发出的数据包的最新编号，就会推测出接收方大概的接收速度，从而降低或增加发送速率。在实际操作中，可以明显的观察到刚启动时的速率往往比较低下，随着发送时间的延长，速率才会被提升到逐渐逼近带宽的水平。对于较小的文件，带宽往往还没来得及拉满传输就结束了，这直接导致了直观上的小文件传输速率远低于大文件的传输速率。另一方面，固定带宽的文件传输速率并不伴随传输文件变大而线性增长，是因为文件在变得足够大后，TCP 有充足的时间测试满带宽，理想情况下之后的瞬时传输速率与带宽相同而并不会继续增长，所以传输一个无穷大的文件的传输速率就是带宽完全拉满的速率。

2. 100ms 延迟

100ms 延迟	文件大小		
带宽	1	10	100
10Mbps	299	610	648
100Mbps	557	966	3436
1Gbps	602	1312	5390

表二 100ms 延迟下的平均传输速率表

图十六 100ms 延迟下的传输速率-文件大小双对数图

分析：

发现对于较大带宽（100Mbps 和 1Gbps）而言，在 100ms 延迟下，增大文件大小，传输速率提升的速率反而增加，看似与 10ms 延迟时出现了相矛盾的现象。但是低带宽（10Mbps）表现出的行为却与 10ms 延迟时更为相似。

解释：

延迟增大到 100ms，对于低带宽而言，延迟的量级与传输速率的量级较为接近，也就是延迟并未超过传输时间，一定程度上可以视作略微降低了带宽，于是低带宽表现出和低延迟相近的曲线。

但是对于高带宽而言，在较小文件的传输上，100ms 的延迟甚至要超过传输的时间，占到整个传输过程的大部分，在这个意义上，慢启动带来的影响远远不及降低延迟所占时间比重带来的速率提升，于是在起初的这段曲线呈现出加速上升的趋势。但在延迟的影响降到较低的比重时，慢启动带来的影响开始呈现，以及速率逐渐可以拉满，随后依旧会出现和 10ms 延迟时类似的现象。