网络实验报告一

雷正宇 2016K8009909005 3月12日

互联网协议实验

实验内容

打开 mininet, 在节点 h1 上开启 wireshark 抓包，用 wget 下载 www.baidu.com，观察 wireshark 的输出并分析。

实验流程

在 Linux 环境下键入命令：

```
$ sudo mn --nat
```

然后在 mininet 窗口键入：

```
mininet> xterm h1
```

在 h1 节点中将地址 8.8.8.8 配置到解析配置表：

```
h1 # echo "nameserver 1.2.4.8"
```

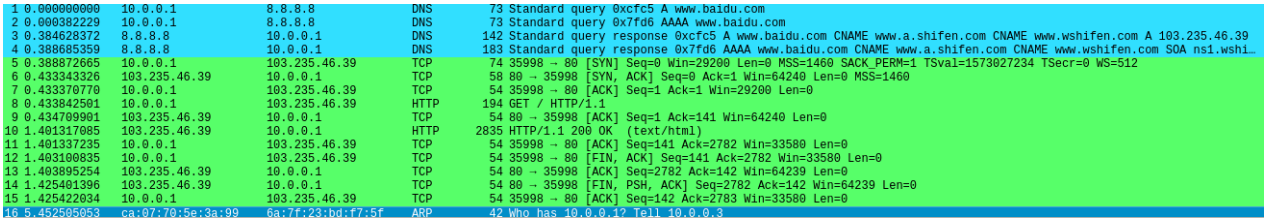
在 h1 节点打开 wireshark 窗口：

```
h1 # wireshark &
```

在 h1 用 wget 下载 www.baidu.com，同时在 wireshark 观察输出。

实验结果及分析

配合截图来分析输出：



1	0.000000000	10.0.0.1	8.8.8.8	DNS	73 Standard query 0xcfc5 A www.baidu.com
2	0.000382229	10.0.0.1	8.8.8.8	DNS	73 Standard query 0x7fd6 AAAA www.baidu.com
3	0.384628372	8.8.8.8	10.0.0.1	DNS	142 Standard query response 0xcfc5 A www.baidu.com CNAME www.a.shifen.com CNAME www.wshifen.com A 103.235.46.39
4	0.388685359	8.8.8.8	10.0.0.1	DNS	183 Standard query response 0x7fd6 AAAA www.baidu.com CNAME www.a.shifen.com CNAME www.wshifen.com SOA ns1.wshi…
5	0.388872665	10.0.0.1	103.235.46.39	TCP	74 35998 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1573027234 TSecr=0 WS=512
6	0.433343326	103.235.46.39	10.0.0.1	TCP	58 80 → 35998 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
7	0.433370770	10.0.0.1	103.235.46.39	TCP	54 35998 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
8	0.433842501	10.0.0.1	103.235.46.39	HTTP	194 GET / HTTP/1.1
9	0.434709901	103.235.46.39	10.0.0.1	TCP	54 80 → 35998 [ACK] Seq=1 Ack=141 Win=64240 Len=0
10	1.401317085	103.235.46.39	10.0.0.1	HTTP	2835 HTTP/1.1 200 OK (text/html)
11	1.401337235	10.0.0.1	103.235.46.39	TCP	54 35998 → 80 [ACK] Seq=141 Ack=2782 Win=33580 Len=0
12	1.403100835	10.0.0.1	103.235.46.39	TCP	54 35998 → 80 [FIN, ACK] Seq=141 Ack=2782 Win=33580 Len=0
13	1.403895254	103.235.46.39	10.0.0.1	TCP	54 80 → 35998 [ACK] Seq=2782 Ack=142 Win=64239 Len=0
14	1.425401396	103.235.46.39	10.0.0.1	TCP	54 80 → 35998 [FIN, PSH, ACK] Seq=2782 Ack=142 Win=64239 Len=0
15	1.425422034	10.0.0.1	103.235.46.39	TCP	54 35998 → 80 [ACK] Seq=142 Ack=2783 Win=33580 Len=0
16	5.452505053	ca:07:70:5e:3a:99	6a:7f:23:bd:f7:5f	ARP	42 Who has 10.0.0.1? Tell 10.0.0.3

ARP协议包

首先查看截图中ARP协议的两个包：

```
16 5.452505053   ca:07:70:5e:3a:99    6a:7f:23:bd:f7:5f    ARP         42 Who has 10.0.0.1? Tell 10.0.0.3
17 5.452525049   6a:7f:23:bd:f7:5f    ca:07:70:5e:3a:99    ARP         42 10.0.0.1 is at 6a:7f:23:bd:f7:5f
```

第一个包的具体内容如下：

```
▸ Frame 16: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▾ Ethernet II, Src: ca:07:70:5e:3a:99 (ca:07:70:5e:3a:99), Dst: 6a:7f:23:bd:f7:5f (6a:7f:23:bd:f7:5f)
  ▸ Destination: 6a:7f:23:bd:f7:5f (6a:7f:23:bd:f7:5f)
  ▸ Source: ca:07:70:5e:3a:99 (ca:07:70:5e:3a:99)
    Type: ARP (0x0806)
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: ca:07:70:5e:3a:99 (ca:07:70:5e:3a:99)
    Sender IP address: 10.0.0.3
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 10.0.0.1
```

结合 ARP 协议工作的过程：每台主机或路由器在其内存中具有一个 ARP 表，这张表包含 IP 地址到 MAC 地址的映射关系。如果发送方（本例为 ca:07:70:5e:3a:99）的 ARP 表具有该目的节点的表项，这个任务是很容易完成的。而此时这个 ARP 表其实并没有目的主机的表项，通过包中的这个部分内容就能看出：

```
Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
Target IP address: 10.0.0.1
```

此时发送方用 ARP 协议来解析这个地址。首先，发送方构造一个 ARP 分组。一个 ARP 分组有几个字段，包括发送喝接收 IP 地址及 MAC 地址。ARP 查询分组喝响应分组都有相同的格式。ARP 查询分组的目的是询问子网上所有其他主机和路由器，以确定对应于要解析的 IP 地址的 MAC 地址。

接下来看第二个 ARP 包：

```
▸ Frame 17: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▾ Ethernet II, Src: 6a:7f:23:bd:f7:5f (6a:7f:23:bd:f7:5f), Dst: ca:07:70:5e:3a:99 (ca:07:70:5e:3a:99)
  ▸ Destination: ca:07:70:5e:3a:99 (ca:07:70:5e:3a:99)
  ▸ Source: 6a:7f:23:bd:f7:5f (6a:7f:23:bd:f7:5f)
    Type: ARP (0x0806)
▾ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: 6a:7f:23:bd:f7:5f (6a:7f:23:bd:f7:5f)
    Sender IP address: 10.0.0.1
    Target MAC address: ca:07:70:5e:3a:99 (ca:07:70:5e:3a:99)
    Target IP address: 10.0.0.3
```

之前第一个 ARP 包被其他适配器接收到，并且每个适配器都把 ARP 分组向上传递给 ARP 模块。这里，MAC 地址 6a:7f:23:bd:f7:5f 有 10.0.0.1 这个 IP，IP 地址匹配成功后按照上一个 ARP 包的请求，把这个有效的 ARP 分组传递回去。

DNS协议包

DNS 是域名系统，可以简单的将 DNS 请求包和响应包看成是客户端在 DNS 服务器（一个庞大的数据库）通过主机名查询其 IP 地址。关于 DNS 包，本例中有两对：

```
1 0.000000000   10.0.0.1          8.8.8.8           DNS        73 Standard query 0xcfc5 A www.baidu.com
2 0.000382229   10.0.0.1          8.8.8.8           DNS        73 Standard query 0x7fd6 AAAA www.baidu.com
3 0.384628372   8.8.8.8           10.0.0.1          DNS       142 Standard query response 0xcfc5 A www.baidu.com CNAME www.a.shifen.com CNAME www.wshifen.com A 103.235.46.39
4 0.388685359   8.8.8.8           10.0.0.1          DNS       183 Standard query response 0x7fd6 AAAA www.baidu.com CNAME www.a.shifen.com CNAME www.wshifen.com SOA ns1.wshi…
```

其中 1, 3 是一个请求-响应对，2, 4 是一个请求-响应对。简单地看第3个包对第1个包的请求作出的回应：

```
▼ Domain Name System (response)
    Transaction ID: 0xcfc5
  ▶ Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 3
    Authority RRs: 0
    Additional RRs: 0
  ▼ Queries
    ▶ www.baidu.com: type A, class IN
  ▼ Answers
    ▶ www.baidu.com: type CNAME, class IN, cname www.a.shifen.com
    ▶ www.a.shifen.com: type CNAME, class IN, cname www.wshifen.com
    ▶ www.wshifen.com: type A, class IN, addr 103.235.46.39
    [Request In: 1]
    [Time: 0.384628372 seconds]
```

在 Answers 部分可以看到，DNS 服务器连续查询了 www.baidu.com 的几个同名主机名，最终查询到 www.wshifen.com （据说这是百度的竞分系统遗留下来的域名，但并不能通过该域名打开网页）的 IP 为 103.235.46.39.

HTTP和TCP

HTTP 是 Web 的应用层协议，HTTP 使用 TCP 作为它的支撑运输协议。HTTP 用户（本例中的 h1 主机 10.0.0.1）首先发起一个与服务器的 TCP 连接，一旦连接建立，客户端和服务器进程就可以通过套接字接口访问 TCP。在本例中，这个连接的发起和响应于 5, 6 两个包：

```
5 0.388872665   10.0.0.1          103.235.46.39     TCP      74 35998 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1573027234 TSecr=0 WS=512
6 0.433343326   103.235.46.39     10.0.0.1          TCP      58 80 → 35998 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
```

接下来就是主机发出的 HTTP 请求。一下是这个 HTTP 请求报文的内容：

```
▼ Hypertext Transfer Protocol
  ▶ GET / HTTP/1.1\r\n
    User-Agent: Wget/1.19.4 (linux-gnu)\r\n
    Accept: */*\r\n
    Accept-Encoding: identity\r\n
    Host: www.baidu.com\r\n
    Connection: Keep-Alive\r\n
    \r\n
    [Full request URI: http://www.baidu.com/]
    [HTTP request 1/1]
    [Response in frame: 10]
```

这个报文是用 ASCII 文本书写的。主要关注该报文的请求行。该请求行的方法字段是一个 GET 方法，URL 字段表示请求的对象，也比较简单，就是根目录。后接的是浏览器实现的 HTTP 版本，本例中是 1.1.

再看服务器对于该 HTTP 请求作出的回应：

HTTP 响应报文包含三个部分：初始状态行，首部行和实体体。实体体是报文的主要部分，包含了所请求的对象本身（在这个部分看不到，在正文部分可以看见一个完整的 HTML 文本）。状态行有3个字段：协议版本字段、状态码和相应状态信息。在本例中，版本是1.1，并且一切正常。

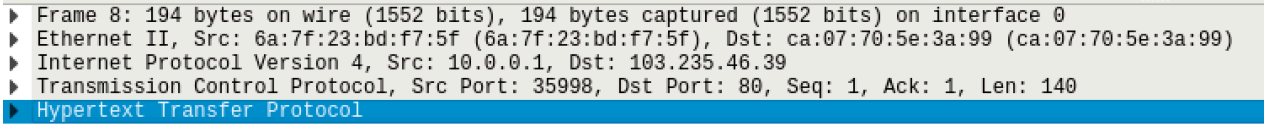
首部行内容有些多，Server 表示该报文是由一台 Apache Web 服务器产生的，它类似于 HTTP 请求报文中的 User-agent。Content-Length 和 Content-Type 分别指示被发送对象的字节数和类型，本例中，发送对象是一个 html 文本。

wireshark 自带 follow stream 功能，可以跟踪整个 TCP 连接的传输流：

```
GET / HTTP/1.1
User-Agent: Wget/1.19.4 (linux-gnu)
Accept: */*
Accept-Encoding: identity
Host: www.baidu.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Server: bfe/1.0.8.18
Date: Wed, 13 Mar 2019 07:50:44 GMT
Content-Type: text/html
Content-Length: 2381
Last-Modified: Mon, 23 Jan 2017 13:28:11 GMT
Connection: Keep-Alive
ETag: "588604eb-94d"
Cache-Control: private, no-cache, no-store, proxy-revalidate, no-transform
Pragma: no-cache
Set-Cookie: BDORZ=27315; max-age=86400; domain=.baidu.com; path=/
Accept-Ranges: bytes

<!DOCTYPE html>
<!--STATUS OK--><html> <head><meta http-equiv=content-type content=text/
```

这里展示的内容分别是我刚才分析过的 HTTP 请求报文、HTTP 响应报文和请求对象（一个 html 文本）。

最后，通过 wireshark 所展示的结构可以发现，这些协议是一层一层封装的：



流完成时间实验

实验内容

利用 python 脚本构建网络拓扑，并调研解释流完成时间图。

实验流程

编写（可以直接 copy 讲义代码）python 脚本，运行以搭建 mininet 虚拟网络拓扑。启动两个节点，一个用于制作本地文件，另一个通过 mininet 来请求文件。变换参数反复试验，观察结果。

实验结果及分析

一下为多组实验结果截图（在实验中，我没有给文件重命名，所以1MB.dat的大小不一定为1MB，具体大小以后面的数字为准）：

带宽为10Mbps：

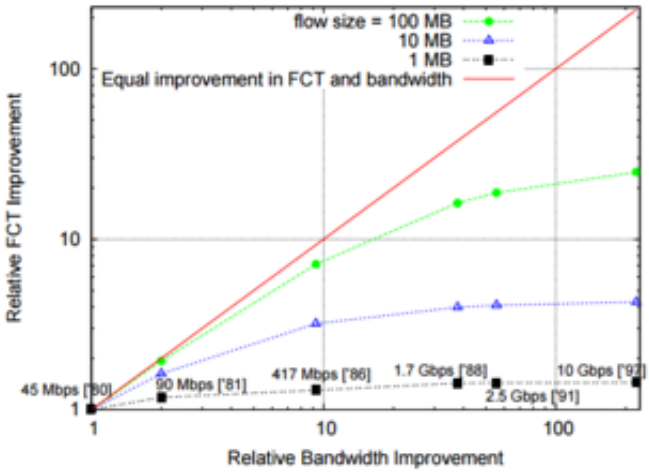
```
1MB.dat.4           100%[===================>]   1.00M  1.13MB/s    in 0.9s

1MB.dat.5           100%[===================>]  10.00M  1.14MB/s    in 8.8s

1MB.dat.6           100%[===================>] 100.00M  1.14MB/s    in 88s
```

带宽为100Mbps：

```
1MB.dat.10          100%[===================>]   1.00M  6.50MB/s    in 0.2s

1MB.dat.11          100%[===================>]  10.00M  10.2MB/s    in 1.0s

1MB.dat.12          100%[===================>] 100.00M  10.8MB/s    in 9.4s
```

带宽为1Gbps：

```
1MB.dat.13          100%[===================>]   1.00M  --.-KB/s    in 0.1s

1MB.dat.14          100%[===================>]  10.00M  41.4MB/s    in 0.2s

1MB.dat.15          100%[===================>] 100.00M  81.8MB/s    in 1.2s
```



实验结果和上图基本吻合。

传输速度和带宽正相关，但不是成正比，原因在于 TCP 传输机制。在第一个实验中已经分析过 TCP 的传输，造成与带宽关系不大的不确定时延的一个原因在于：在发送的数据包中，第一对必定是请求和确认应答（ACK），这两个包很小，占用的时间和带宽关联不大。

在连接建立的初期，如果窗口比较大，发送方可能会突然发送大量数据，导致网络瘫痪。因此，在通信一开始时，TCP 会通过慢启动算法得出窗口的大小，对发送数据量进行控制。流量控制是由发送方和接收方共同控制的。接收方会把自己能够承受的最大窗口长度写在 TCP 首部中，实际上在发送方这里，也存在流量控制，它叫拥塞窗口。

对于本例中的情况，在带宽上升到一定值后，FCT 增长的相对值变得缓慢。以下以 cwnd 记拥塞窗口，ssthresh 记门限值。当 cwnd < ssthresh 时，cwnd 随 ACK 线性提升，随 RTT 指数上升。当 cwnd >= ssthresh 后，进入拥塞避免算法，cwnd 随 RTT 线性增加。

感想

第一次实验动手内容比较简单，也没有需要自己动手编写的代码。但实验涉及的知识很多，我花了很多时间学习关于 ARP, TCP, HTTP 和 DNS 的知识。对于后半部分实验，TCP 的慢启动机制我甚至并没有完全弄清楚，交付报告后会再抽时间解决遗留问题。