计算机网络实验 1 报告

学号　2017K8009929059
姓名　於修远

互联网协议实验

一、实验内容

- 使用 wireshark 软件，在 mininet 环境中观察结点 h1 执行 wget www.baidu.com 命令下载页面时，在网站上产生的各种包。
- 调研说明 wireshark 抓取的 ARP，DNS，TCP，HTTP 等协议。
- 解释 h1 下载 baidu 页面的整个过程中，不同协议的运行机制。

二、实验流程

（一）必要程序安装

1、 重新编译 linux-4.9 内核，并设置为 boot 的默认选项。

2、 利用 aptitude 工具下载安装 xterm 和 mininet 和 wireshark。

（二）网络环境创建

1、 使用`sudo mn --nat`指令创建 mininet 实验环境。

2、 在 mninet 环境中执行`xterm h1`，调出 h1 的 shell。

3、 执行`echo “nameserver 1.2.4.8”>/etc/resolv.comf`维护环境配置。

（三）命令执行

1、 执行`wireshark &`命令调出 wireshark 的 gui，并选中 h1 对应网络，开始监控。

2、 在 mninet 环境中执行命令`wget www.baidu.com`，观察 wireshark 抓取网络包的输出情况。

三、结果分析

（一）各种网络协议的功能与联系

1、ARP 协议

地址解析协议，实现从 IP 地址到 MAC 地址的映射，属于链路层协议。主机和交换机间通过广播查询，单播回复，获取映射关系。

2、DNS 协议

域名解析协议，实现从域名到 IP 地址的映射，属于应用层协议。主机首先查询自身的 DNS 缓存表，若无结果则向上级域名服务器查询，依次包括本地域名服务器、权限域名服务器、顶级域名服务器和根域名服务器。一旦查询成功，再向下回复查询结果。

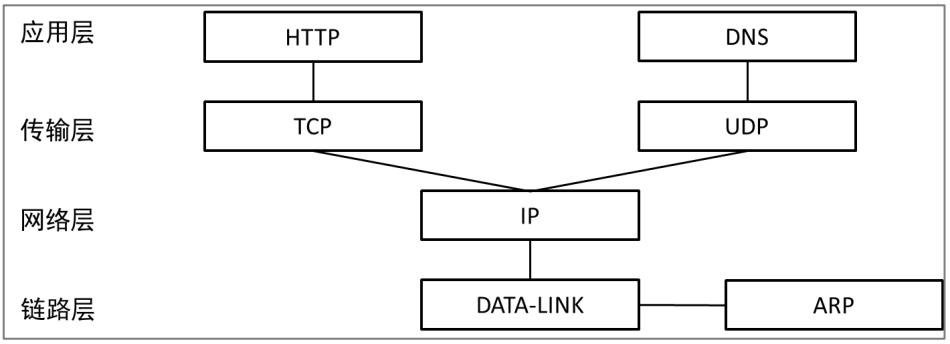
3、TCP 协议

传输控制协议，实现从 IP 地址到 MAC 地址的映射，属于传输层协议。TCP 与处于网络层的 IP 协议一起工作。TCP 传输的最大特征是建立连接需要 3 次握手、解除连接需要 4 次挥手。TCP 还包含了慢启动、拥塞窗口等机制

4、HTTP 协议

超文本传输协议，从 www 服务器传输超文本到本地浏览器，属于应用层协议。HTTP 协议基于 TCP 协议，由请求和响应构成，具有多次请求和无状态等特点。

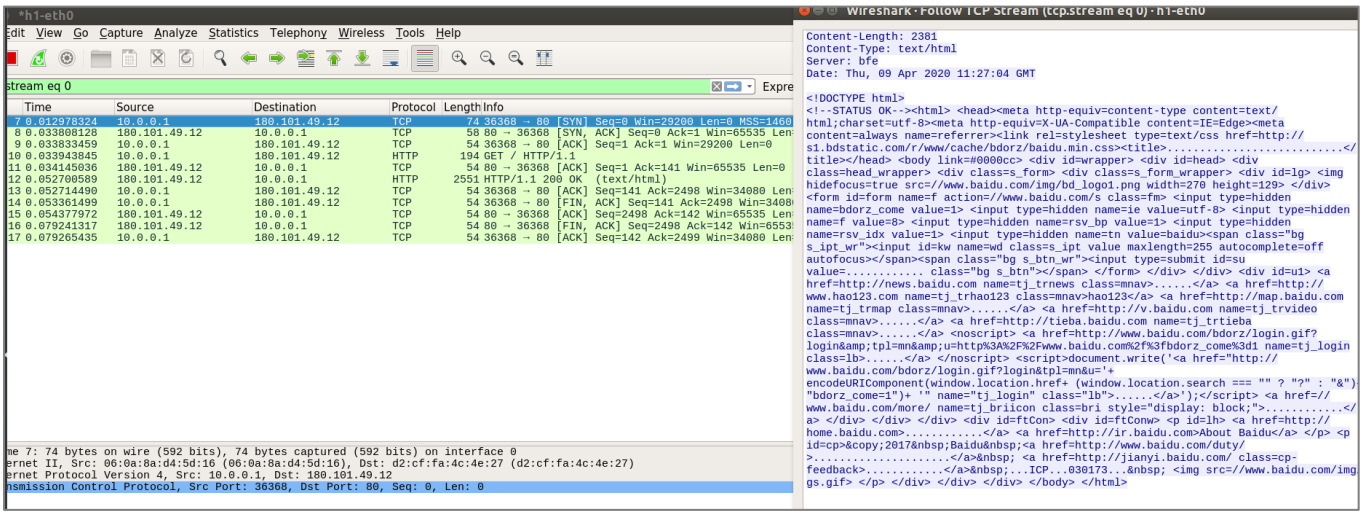
5、协议间关系



（二）实验过程中各协议运行机制

1、抓包结果

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	06:0a:8a:d4:5d:16	Broadcast	ARP	42	Who has 10.0.0.3? Tell 10.0.0.1
2	0.000162742	d2:cf:fa:4c:4e:27	06:0a:8a:d4:5d:16	ARP	42	10.0.0.3 is at d2:cf:fa:4c:4e:27
3	0.000165676	10.0.0.1	1.2.4.8	DNS	73	Standard query 0x4a20 A www.baidu.com
4	0.000270063	10.0.0.1	1.2.4.8	DNS	73	Standard query 0x3ce8 AAAA www.baidu.com
5	0.008402184	1.2.4.8	10.0.0.1	DNS	302	Standard query response 0x4a20 A www.baidu.com CNAME www.a
6	0.012022233	1.2.4.8	10.0.0.1	DNS	157	Standard query response 0x3ce8 AAAA www.baidu.com CNAME ww
7	0.012978324	10.0.0.1	180.101.49.12	TCP	74	36368 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
8	0.033808128	180.101.49.12	10.0.0.1	TCP	58	80 → 36368 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
9	0.033833459	10.0.0.1	180.101.49.12	TCP	54	36368 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
10	0.033943845	10.0.0.1	180.101.49.12	HTTP	194	GET / HTTP/1.1
11	0.034145036	180.101.49.12	10.0.0.1	TCP	54	80 → 36368 [ACK] Seq=1 Ack=141 Win=65535 Len=0
12	0.052700589	180.101.49.12	10.0.0.1	HTTP	2551	HTTP/1.1 200 OK (text/html)
13	0.052714490	10.0.0.1	180.101.49.12	TCP	54	36368 → 80 [ACK] Seq=141 Ack=2498 Win=34080 Len=0
14	0.053361499	10.0.0.1	180.101.49.12	TCP	54	36368 → 80 [FIN, ACK] Seq=141 Ack=2498 Win=34080 Len=0
15	0.054377972	180.101.49.12	10.0.0.1	TCP	54	80 → 36368 [ACK] Seq=2498 Ack=142 Win=65535 Len=0
16	0.079241317	180.101.49.12	10.0.0.1	TCP	54	80 → 36368 [FIN, ACK] Seq=2498 Ack=142 Win=65535 Len=0
17	0.079265435	10.0.0.1	180.101.49.12	TCP	54	36368 → 80 [ACK] Seq=142 Ack=2499 Win=34080 Len=0
18	5.023461021	d2:cf:fa:4c:4e:27	06:0a:8a:d4:5d:16	ARP	42	Who has 10.0.0.1? Tell 10.0.0.3
19	5.023470214	06:0a:8a:d4:5d:16	d2:cf:fa:4c:4e:27	ARP	42	10.0.0.1 is at 06:0a:8a:d4:5d:16
20	16.027749138	fe80::d0cf:faff:fe4…	ff02::2	ICMPv6	70	Router Solicitation from d2:cf:fa:4c:4e:27
21	20.124107604	fe80::40a:8aff:fed4…	ff02::2	ICMPv6	70	Router Solicitation from 06:0a:8a:d4:5d:16
22	20.124255953	fe80::e0c8:b8ff:fe3…	ff02::2	ICMPv6	70	Router Solicitation from e2:c8:b8:38:c0:b8
23	23.468721703	10.0.0.3	224.0.0.251	MDNS	183	Standard query 0x0000 PTR _ipps._tcp.local, "QM" question
24	24.786051637	fe80::ec6a:faff:fe8…	ff02::fb	MDNS	203	Standard query 0x0000 PTR _ipps._tcp.local, "QM" question
25	24.938433398	fe80::d0cf:faff:fe4…	ff02::fb	MDNS	203	Standard query 0x0000 PTR _ipps._tcp.local, "QM" question
26	40.603367359	fe80::ec6a:faff:fe8…	ff02::2	ICMPv6	70	Router Solicitation from ee:6a:fa:87:82:31



2、下载过程

(1) h1 通过 ARP 协议获得交换机的 mac 地址。如上图所示，第一个 ARP 包为 h1 广播的信号，目的是让拥有 10.0.0.3 条目的交换机回复其 mac 地址。第二个 ARP 包即为该交换机对 h1 单播的回复信号。

(2) h1 通过 DNS 协议获得域名 www.baidu.com 的 IP 地址。如上图所示，前两个 DNS 包（编号 3 和 4）向自身对应的域名服务器（启动时设置为 1.2.4.8）发出域名查找，后两个 DNS 包（编号 5 和 6）是域名服务器回复的 IP 地址查询结果。

(3) h1 通过 TCP 协议与 baidu 服务器建立连接。首先发送含 seq=0 的 SYN 包（编号 7，第一个 TCP 包）到服务器，并进入 SYN-SEND 状态，收到服务器回复的含 ack=1 且 seq=0 的 SYN+ACK 包（编号 8，第二个 TCP 包）后，对应返回一个含 ack=1 的 ACK 包（编号 9，第三个 TCP 包）。至此三次握手全部完成。

(4) h1 通过 HTTP 协议进行数据传输。h1 向服务器发送 HTTP 包（编号 10，第一个 HTTP 包）进行下载请求，服务器收到后向 h1 发出 ACK 包（编号 11，第四个 TCP 包）进行通道连接，随后发出包含下载内容的 HTTP 包（编号 12，第二个 HTTP 包）。h1 在接收完成后发出 ACK 包（编号 13，第五个 TCP 包）断开本次传输通道。

(5) h1 通过 TCP 协议与 baidu 服务器解除连接。首先向服务器发送含 seq=141 的 FIN+ACK 包（编号 14，

第六个 TCP 包），并进入 FIN_WAIT_1 状态。收到该包后，服务器首先发送一个含 ack=142 的 ACK 包确认（编号 15，第七个 TCP 包），再发送一个含 seq=2498 的 FIN+ACK 包（编号 16，第八个 TCP 包），来关闭数据传输。收到这两个挥手包后 h1 发送含 ack=2499 的 ACK 包（编号 17，最后一个 TCP 包）确认。至此四次挥手全部完成，网页下载从建立到传输到中断的全过程也就此结束。

（三）ARP 与 DNS 机制

1、ARP 缓存机制

为了提高性能，主机和交换机上一般都会维护一个 ARP 缓存表，来保存 IP 地址到 MAC 地址的映射。这一缓存表会定期更新，保留近期常用的通讯地址；每条条目也都会有一定的生命周期。在自身的缓存表中缺少需要通信的条目时，通过广播得到持有此条目映射的主机或交换机的回复。

2、DNS 缓存机制

与 ARP 缓存机制类似，主机或交换机维护一个 DNS 缓存表，来保存域名到 IP 地址的映射。当所需条目缺失时，机器会向一个设定的上级域名服务器发出请求，以获取映射关系。但是，虽然 Windows 系统中包含有 DNS 缓存，许多 Linux 发行版并不具有，所以域名解析只能靠收发 DNS 包来完成。

3、缓存机制验证

如下图所示，在很短的时间间隔内连续执行多次`wget www.baidu.com`命令，并观察收发包情况即可。如下图所示，第一次执行页面下载时，h1 既收发了 ARP 包也收发了 DNS 包，但第二次则只收发了 DNS 包（编号为 18 和 19 的 ARP 包不是 wget 后进行的收发包，从内容上也可以看出这是由 10.0.0.3 发送的确认包）。重复实验多次，可以发现从第二次起每次都要重新进行 DNS 包的收发，但无需进行 ARP 包收发。这就说明了 ARP 存在缓存机制，但本机版本的 ubuntu 不具有 DNS 缓存。

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	0e:e8:0e:ce:f3:62	Broadcast	ARP	42	Who has 10.0.0.3? Tell 10.0.0.1
2	0.000377147	d6:76:8e:19:4c:ee	0e:e8:0e:ce:f3:62	ARP	42	10.0.0.3 is at d6:76:8e:19:4c:ee
3	0.000381455	10.0.0.1	1.2.4.8	DNS	73	Standard query 0x0824 A www.baidu.com
4	0.000383224	10.0.0.1	1.2.4.8	DNS	73	Standard query 0x4127 AAAA www.baidu.com
5	0.018843927	1.2.4.8	10.0.0.1	DNS	302	Standard query response 0x0824 A www.baidu.com CNAME w…
6	0.018862249	1.2.4.8	10.0.0.1	DNS	157	Standard query response 0x4127 AAAA www.baidu.com CNAM…
7	0.019067831	10.0.0.1	180.101.49.12	TCP	74	36390 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_P…
8	0.088589502	180.101.49.12	10.0.0.1	TCP	58	80 → 36390 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=…
9	0.088619183	10.0.0.1	180.101.49.12	TCP	54	36390 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
10	0.089686577	10.0.0.1	180.101.49.12	HTTP	194	GET / HTTP/1.1
11	0.089928109	180.101.49.12	10.0.0.1	TCP	54	80 → 36390 [ACK] Seq=1 Ack=141 Win=65535 Len=0
12	0.122576063	180.101.49.12	10.0.0.1	HTTP	2551	HTTP/1.1 200 OK (text/html)
13	0.122591189	10.0.0.1	180.101.49.12	TCP	54	36390 → 80 [ACK] Seq=141 Ack=2498 Win=34080 Len=0
14	0.123898604	10.0.0.1	180.101.49.12	TCP	54	36390 → 80 [FIN, ACK] Seq=141 Ack=2498 Win=34080 Len=0
15	0.124278373	180.101.49.12	10.0.0.1	TCP	54	80 → 36390 [ACK] Seq=2498 Ack=142 Win=65535 Len=0
16	0.155117035	180.101.49.12	10.0.0.1	TCP	54	80 → 36390 [FIN, ACK] Seq=2498 Ack=142 Win=65535 Len=0
17	0.155141065	10.0.0.1	180.101.49.12	TCP	54	36390 → 80 [ACK] Seq=142 Ack=2499 Win=34080 Len=0
18	5.257224168	d6:76:8e:19:4c:ee	0e:e8:0e:ce:f3:62	ARP	42	Who has 10.0.0.1? Tell 10.0.0.3
19	5.257236949	0e:e8:0e:ce:f3:62	d6:76:8e:19:4c:ee	ARP	42	10.0.0.1 is at 0e:e8:0e:ce:f3:62
20	8.213340552	10.0.0.1	1.2.4.8	DNS	73	Standard query 0x0bb8 A www.baidu.com
21	8.213419843	10.0.0.1	1.2.4.8	DNS	73	Standard query 0x2252 AAAA www.baidu.com
22	8.223900577	1.2.4.8	10.0.0.1	DNS	302	Standard query response 0x0bb8 A www.baidu.com CNAME w…
23	8.224282491	1.2.4.8	10.0.0.1	DNS	157	Standard query response 0x2252 AAAA www.baidu.com CNAM…
24	8.224560771	10.0.0.1	180.101.49.12	TCP	74	36392 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_P…
25	8.245292002	180.101.49.12	10.0.0.1	TCP	58	80 → 36392 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=…
26	8.245316550	10.0.0.1	180.101.49.12	TCP	54	36392 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
27	8.245558144	10.0.0.1	180.101.49.12	HTTP	194	GET / HTTP/1.1
28	8.245871827	180.101.49.12	10.0.0.1	TCP	54	80 → 36392 [ACK] Seq=1 Ack=141 Win=65535 Len=0
29	8.268103797	180.101.49.12	10.0.0.1	HTTP	2551	HTTP/1.1 200 OK (text/html)
30	8.268118330	10.0.0.1	180.101.49.12	TCP	54	36392 → 80 [ACK] Seq=141 Ack=2498 Win=34080 Len=0
31	8.269682495	10.0.0.1	180.101.49.12	TCP	54	36392 → 80 [FIN, ACK] Seq=141 Ack=2498 Win=34080 Len=0
32	8.270058589	180.101.49.12	10.0.0.1	TCP	54	80 → 36392 [ACK] Seq=2498 Ack=142 Win=65535 Len=0
33	8.288372868	180.101.49.12	10.0.0.1	TCP	54	80 → 36392 [FIN, ACK] Seq=2498 Ack=142 Win=65535 Len=0
34	8.288388410	10.0.0.1	180.101.49.12	TCP	54	36392 → 80 [ACK] Seq=142 Ack=2499 Win=34080 Len=0

另一方面，在 h1 的 shell 中执行`ip neigh`，可以看到 d6:768e:19:4c:ee 的状态为 STABLE，即需要发送验证报文。上述编号为 18 的 ARP 包也印证了这一点。

流完成时间实验

一、实验内容

- 使用 fct_exp.py 脚本复现课件中流完成时间变化曲线图。
- 调研解释图中现象。

二、实验流程

（一）网络环境创建

1、 执行`sudo python fct_exp.py`以创建设定带宽和延迟等参数的 mininet 实验环境。

2、 在 mninet 环境中执行`xterm h1 h2`，调出 h1 和 h2 两个结点的 shell。

（二）命令执行

1、 在 h2 的 shell 中执行命令`dd if=/dev/zero of=1MB.dat bs=1M count=1`，生成等待下载的 1MB 大小文件。改变文件名和文件大小，同样生成 10MB 和 100MB 的文件。

2、 在 h1 的 shell 中执行命令`wget http://10.0.0.2/1MB.dat`，并更改文件名，尝试下载 3 种不同大小的文件，每种至少 5 次。同时记录下载耗时。

3、 更改脚本中带宽的设置（10MBps~1GBps），重复上述步骤。

（三）数据处理和计算

1、 汇总三种不同大小的文件在五种不同带宽下的下载耗时（流完成时间），并计算各条件下的均值。

2、 根据不同的文件大小计算在平均流完成时间下文件传输的平均速率。

3、 对每种文件大小，以带宽为 10MBps 情况下的平均速率为基准，对其余带宽下的平均速率进行规格化处理。

4、 以对数尺度横纵坐标作出流完成时间图。

三、结果分析

（一）数据处理

1、原始数据记录表

如下三表所示，分别为延时为 10ms 条件下不同大小文件每次的流完成时间。

1MB	10Mbps	50Mbps	100Mbps	500Mbps	1000Mbps
1	4.0s	0.8s	0.8s	0.3s	0.1s
2	2.9s	0.3s	0.3s	0.4s	0.3s
3	3.9s	1.2s	0.3s	0.4s	0.5s
4	0.9s	0.9s	1.3s	0.5s	0.8s
5	3.3s	1.2s	0.3s	0.8s	0.5s
avg	3.0s	0.9s	0.6s	0.5s	0.4s

10MB	10Mbps	50Mbps	100Mbps	500Mbps	1000Mbps
1	39.0s	2.4s	1.2s	1.8s	2.1s
2	35.0s	9.1s	3.8s	1.9s	1.2s
3	34.0s	2.1s	3.2s	0.6s	0.6s
4	38.0s	2.3s	1.6s	1.3s	0.8s
5	32.0s	6.1s	1.2s	0.6s	0.4s
avg	35.6s	4.4s	2.2s	1.2s	1.0s

100MB	10Mbps	50Mbps	100Mbps	500Mbps	1000Mbps
1	88.0s	29.0s	14.0s	4.0s	4.0s
2	244.0s	24.0s	12.0s	4.5s	3.2s
3	285.0s	26.0s	12.0s	3.8s	3.3s
4	369.0s	24.0s	12.0s	4.4s	2.8s
5	236.0s	25.0s	13.0s	7.7s	4.6s
avg	244.4s	25.6s	12.6s	4.9s	3.6s

2、数据处理与规格化

根据 FCT 均值计算出下载速率的均值，如下表所示。

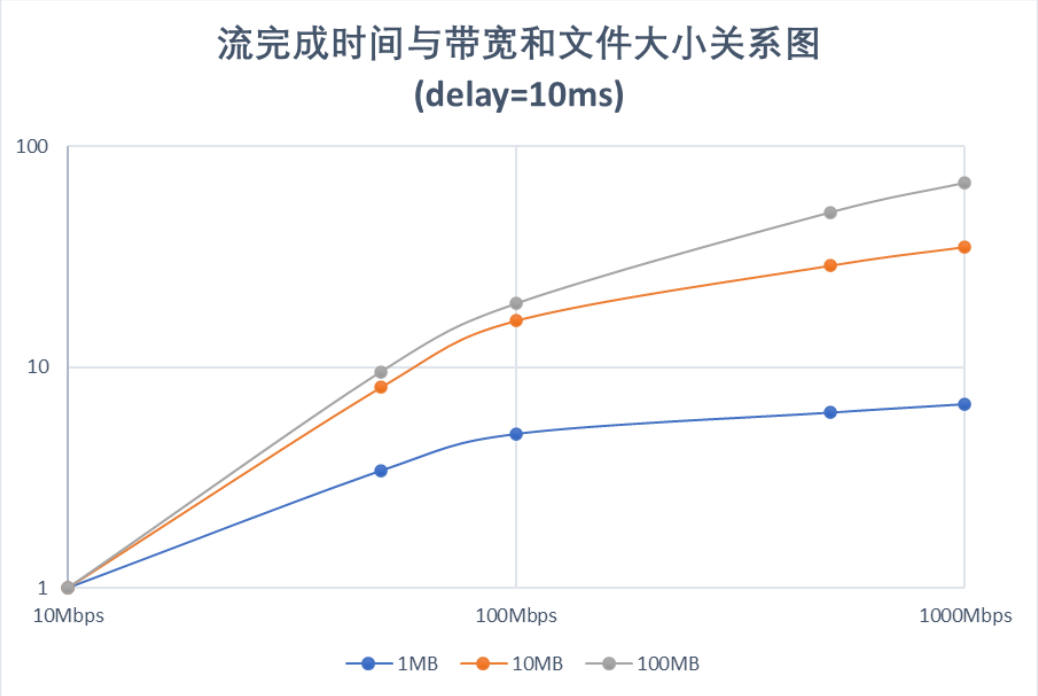
	10Mbps	50Mbps	100Mbps	500Mbps	1000Mbps
1MB	2.7Mbps	9.1Mbps	13.3Mbps	16.7Mbps	18.2Mbps
10MB	2.2Mbps	18.2Mbps	36.4Mbps	64.5Mbps	78.4Mbps
100MB	3.3Mbps	31.3Mbps	63.5Mbps	163.9Mbps	223.5Mbps

取每列首项进行规格化，得到如下表的 FCT 提升率。

	10Mbps	50Mbps	100Mbps	500Mbps	1000Mbps
1MB	1	3.41	5.00	6.25	6.82
10MB	1	8.09	16.18	28.71	34.90
100MB	1	9.55	19.40	50.08	68.27

3、图像绘制

取对数坐标轴，得到如图的曲线图。其中横坐标为带宽，单位为 Mbps；纵坐标为下载加速比。



（二）现象分析

从图像中可以看到，对于一定大小的文件，增加带宽可以提高传输的平均速率，缩短平均流完成时间。但是这一增幅并不是随带宽线性变化的。随着带宽增大，平均速率的增加也会逐步放缓。

究其原因，是 TCP 传输存在慢启动机制：通过拥塞窗口来限制发送方的数据传输量，避免在大规模数据的突发传输引起网络瘫痪。这也与下载过程中观察到的现象相符：尤以 100MB 数据下载时最为明显，瞬时下载速率有一个缓慢增长的过程，从起初的几百 KB，经过一定的时间后才增长到 MB 的数量级。这也很好地解释了为什么图像中 100MB 数据的加速比较 10MB 数据而言增长不明显，慢启动导致 100MB 数据下载的开始阶段被迫被拖到和 10MB 数据相近的传输率。

但是，拥塞窗口的大小限制了发送流量的上限，同时慢启动机制本身也拖慢了达到最大传输速率的时间，所以传输速率无法获得对应于带宽的线性增长，最终趋于平缓。