

Wireshark 实验指导书

目 录

实验一 Wireshark 的安装与使用

实验二 使用 Wireshark 分析 IP 协议

实验三 使用 Wireshark 分析 UDP 协议

实验四 使用 Wireshark 分析 TCP 协议

实验五 利用 Wireshark 分析协议 HTTP

实验一 Wireshark 的安装与使用

一、实验目的

- 1、熟悉并掌握 Wireshark 的基本使用；
- 2、了解网络协议实体间进行交互以及报文交换的情况。

二、实验环境

与因特网连接的计算机，操作系统为 Windows，安装有 Wireshark、IE 等软件。

三、预备知识

要深入理解网络协议，需要观察它们的工作过程并使用它们，即观察两个协议实体之间交换的报文序列，探究协议操作的细节，使协议实体执行某些动作，观察这些动作及其影响。这种观察可以在仿真环境下或在因特网这样的真实网络环境中完成。

观察正在运行的协议实体间交换报文的基本工具被称为分组嗅探器（packet sniffer），又称分组捕获器。顾名思义，分组嗅探器捕获（嗅探）你的计算机发送和接收的报文。

图 1 显示了一个分组嗅探器的结构。

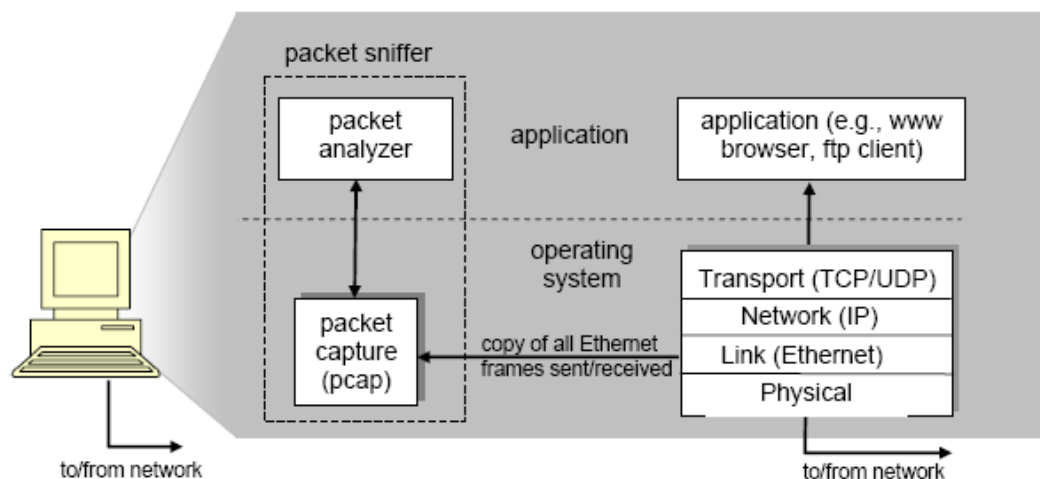


图 1

图 1 右边是计算机上正常运行的协议和应用程序（如：Web 浏览器和 FTP 客户端）。分组嗅探器（虚线框中的部分）主要有两部分组成：第一是分组捕获器，其功能是捕获计算机发送和接收的每一个链路层帧的拷贝；第二个组成部分是分组分析器，其作用是分析并显示协议报文所有字段的内容（它能识别目前使用的

各种网络协议)。

Wireshark 是一种可以运行在 Windows, UNIX, Linux 等操作系统上的分组嗅探器, 是一个开源免费软件, 可以从 <http://www.wireshark.org> 下载。

运行 Wireshark 程序时, 其图形用户界面如图 2 所示。最初, 各窗口中并无数据显示。Wireshark 的界面主要有五个组成部分:

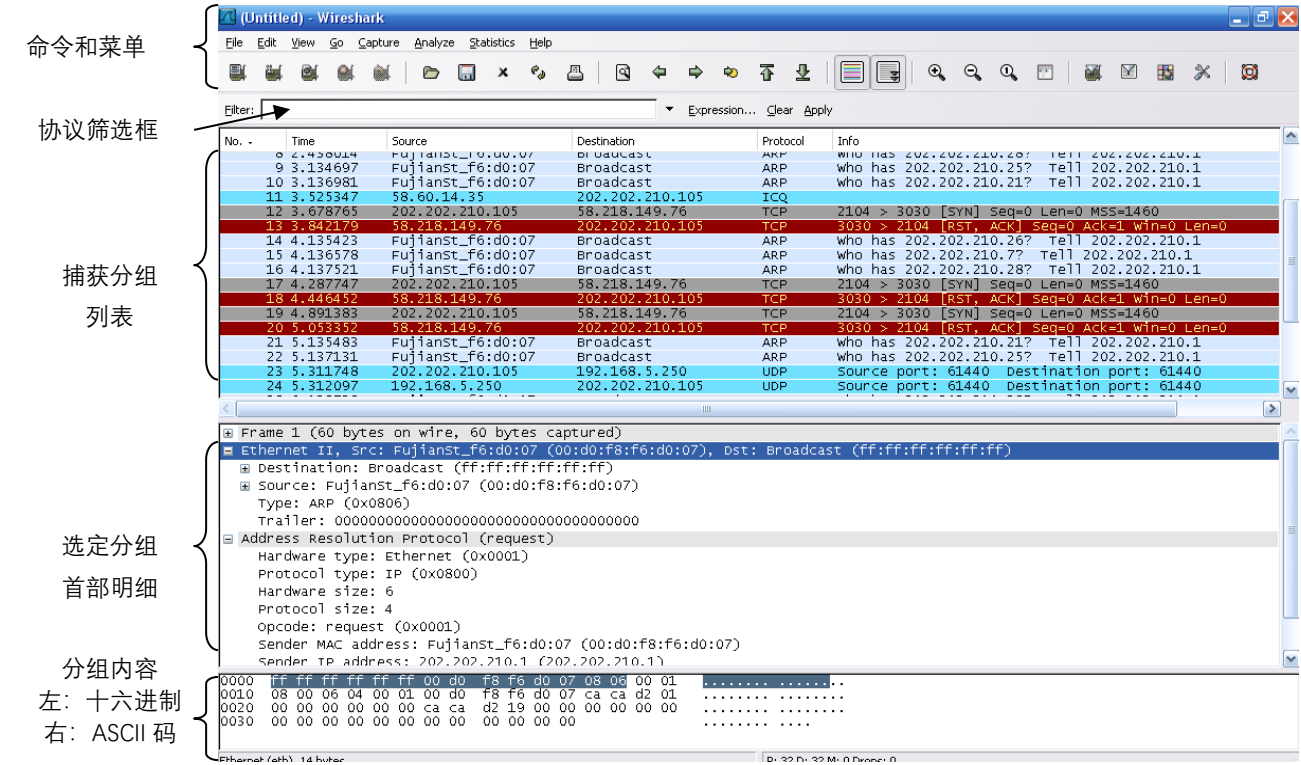


图 2


- **命令菜单 (command menus):** 命令菜单位于窗口的最顶部, 是标准的下拉式菜单。
- **协议筛选框 (display filter specification):** 在该处填写某种协议的名称, Wireshark 据此对分组列表窗口中的分组进行过滤, 只显示你需要的分组。
- **捕获分组列表 (listing of captured packets):** 按行显示已被捕获的分组内容, 其中包括: 分组序号、捕获时间、源地址和目的地址、协议类型、协议信息说明。单击某一列的列名, 可以使分组列表按指定列排序。其中, 协议类型是发送或接收分组的最高层协议的类型。
- **分组首部明细 (details of selected packet header):** 显示捕获分组列表窗口中被选中分组的首部详细信息。包括该分组的各个层次的首部信息, 需要查看哪层信息, 双击对应层次或单击该层最前面的 “+” 即可。

- 分组内容窗口（packet content）：分别以十六进制（左）和 ASCII 码（右）两种格式显示被捕获帧的完整内容。

四、实验步骤

1. 启动 Web 浏览器（如 IE）；

2. 启动 Wireshark；

3. 开始分组捕获：单击工具栏的按钮，出现如图 3 所示对话框，[options]按钮可以进行系统参数设置，在绝大部分实验中，使用系统的默认设置即可。当计算机具有多个网卡时，选择其中发送或接收分组的网络接口（本例中，第一块网卡为虚拟网卡，第二块为以太网卡）。单击“Start”开始进行分组捕获；

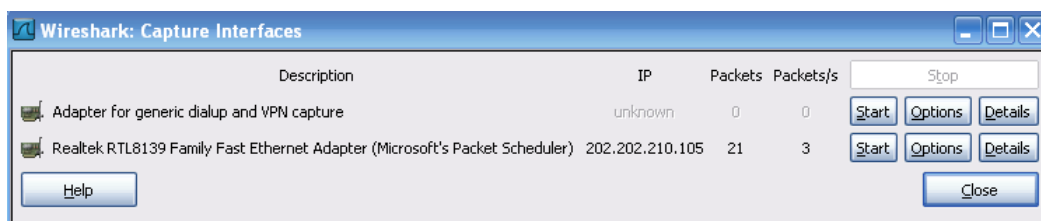


图 3

4. 在运行分组捕获的同时，在浏览器地址栏中输入某个网页的 URL，如：

<http://www.nxist.com>

5. 当完整的页面下载完成后，单击捕获对话框中的“stop”按钮，停止分组捕获。此时，Wireshark 主窗口显示已捕获的你本次通信的所有协议报文；

6. 在协议筛选框中输入“http”，单击“apply”按钮，分组列表窗口将只显示 HTTP 协议报文。

7. 选择分组列表窗口中的第一条 http 报文，它是你的计算机发向服务器（www.nxist.com）的 HTTP GET 报文。当你选择该报文后，以太网帧、IP 数据报、TCP 报文段、以及 HTTP 报文首部信息都将显示在分组首部子窗口中，其结果如图 4。

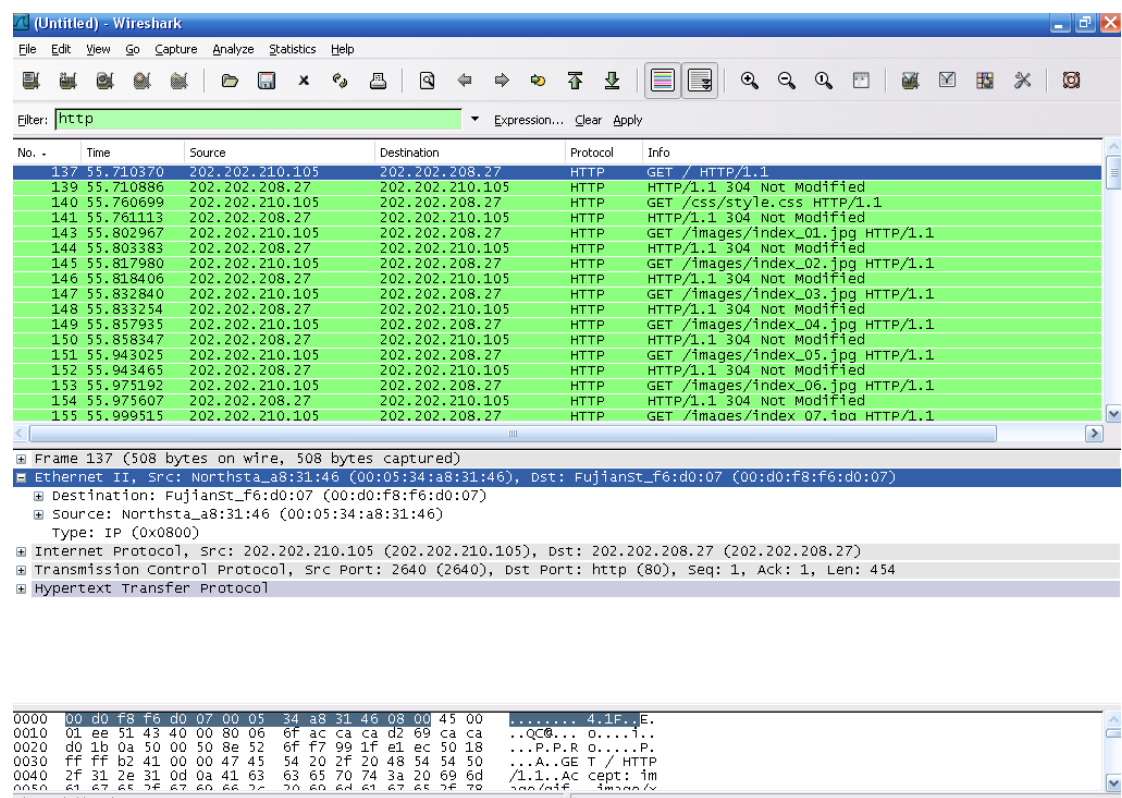


图 4

五、实验报告内容

在实验基础上，回答以下问题：

- (1) 列出在第 5 步中分组列表子窗口所显示的所有协议类型；
- (2) 从发出 HTTP GET 报文到接收到对应的 HTTP OK 响应报文共需要多长时间？（分组列表窗口中 Time 列的值是从 Wireshark 开始追踪到分组被捕获的总的时间数，以秒为单位）
- (3) 你主机的 IP 地址是什么？你访问的服务器的 IP 地址是什么？

实验二 使用 Wireshark 分析 IP 协议

一、实验目的

- 1、分析 IP 协议
- 2、分析 IP 数据报分片

二、实验环境

与因特网连接的计算机，操作系统为 Windows，安装有 Wireshark、IE 等软件。

三、实验步骤

IP 协议是因特网上的中枢。它定义了独立的网络之间以什么样的方式协同工作从而形成一个全球户联网。因特网内的每台主机都有 IP 地址。数据被称作数据报的分组形式从一台主机发送到另一台。每个数据报标有源 IP 地址和目的 IP 地址，然后被发送到网络中。如果源主机和目的主机不在同一个网络中，那么一个被称为路由器的中间机器将接收被传送的数据报，并且将其发送到距离目的端最近的下一个路由器。这个过程就是分组交换。

IP 允许数据报从源端途经不同的网络到达目的端。每个网络有它自己的规则和协定。IP 能够使数据报适应于其途径的每个网络。例如，每个网络规定的最大传输单元各有不同。IP 允许将数据报分片并在目的端重组来满足不同网络的规定。

在 4_1_JoiningTheInternet 下打开已俘获的分组，分组名为：dhcp_isolated.cap。感兴趣的同学可以在有动态分配 IP 的实验环境下俘获此分组，此分组具体俘获步骤如下：

1、使用 DHCP 获取 IP 地址

- (1) 打开命令窗口,启动Wireshark。
- (2) 输入“*ipconfig /release*”。这条命令会释放主机目前的IP地址，此时，主机IP地址会变为0.0.0.0
- (3) 然后输入“*ipconfig /renew*”命令。这条命令让主机获得一个网络配置，包括新的IP地址。
- (4) 等待，直到“*ipconfig /renew*”终止。然后再次输入“*ipconfig /renew*”命令。

(5) 当第二个命令“*ipconfig /renew*”终止时，输入命令“*ipconfig /release*”释放原来的已经分配的IP地址

(6) 停止分组俘获。如图 1 所示：

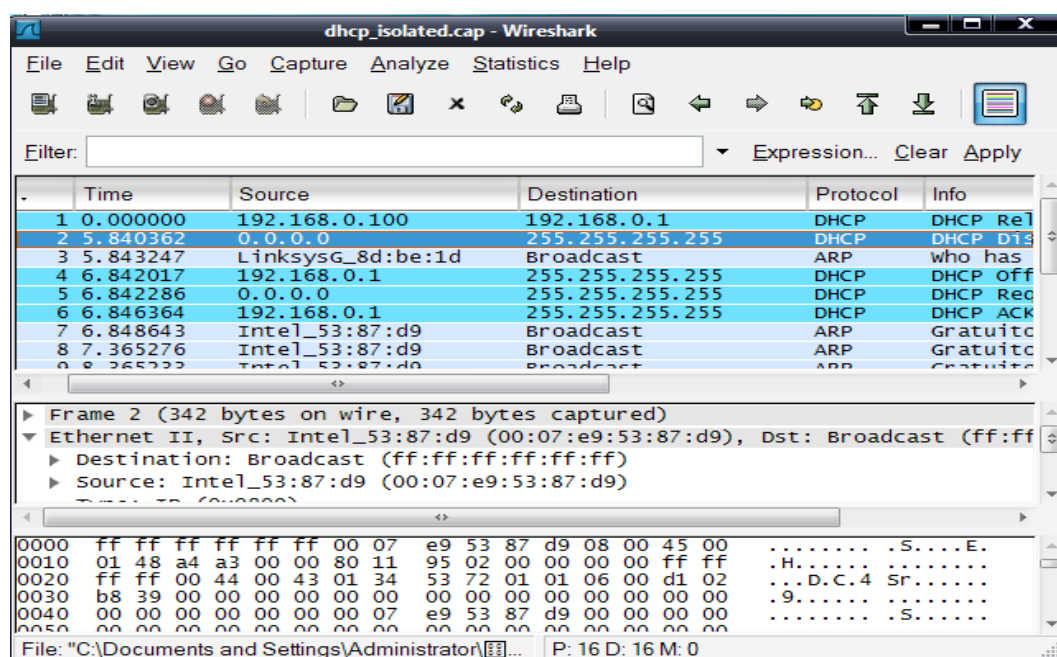


图 1 Wireshark 俘获的分组

下面，我们对此分组进行分析：

IPconfig 命令被用于显示机器的 IP 地址及修改 IP 地址的配置。当输入命令 *ipconfig /release* 命令时，用来释放机器的当前 IP 地址。释放之后，该机没有有效的 IP 地址并在分组 2 中使用地址 0.0.0.0 作为源地址。

分组 2 是一个 DHCP Discover(发现)报文，如图 2 所示。当一台没有 IP 地址的计算机申请 IP 地址时将发送该报文。DHCP Discovery 报文被发送给特殊的广播地址：255.255.255.255，该地址将到达某个限定广播范围内所有在线的主机。理论上，255.255.255.255 能够广播到整个因特网上，但实际上并不能实现，因为路由器为了阻止大量的请求淹没因特网，不会将这样的广播发送到本地网之外。

在 DHCP Discover 报文中，客户端包括自身的信息。特别是，它提供了自己的主机名(MATTHEWS)和其以太网接口的物理地址(00:07:e9:53:87:d9)。这些信息都被 DHCP 用来标识一个已知的客户端。DHCP 服务器可以使用这些信息实现一系列的策略，比如，分配与上次相同的 IP 地址，分配一个上次不同的 IP 地址，或要求客户端注册其物理层地址来获取 IP 地址。

在 DHCP Discover 报文中，客户端还详细列出了它希望从 DHCP 服务器接

收到的信息。在 Parameter Request List 中包含了除客户端希望得到的本地网络的 IP 地址之外的其他数据项。这些数据项中许多都是一台即将连入因特网的计算机所需要的数据。例如，客户端必须知道的本地路由器的标识。任何目的地址不在本地网的数据报都将发送到这台路由器上。也就是说，这是发向外网的数据报在通向目的端的路径上遇到的第一台中间路由器。

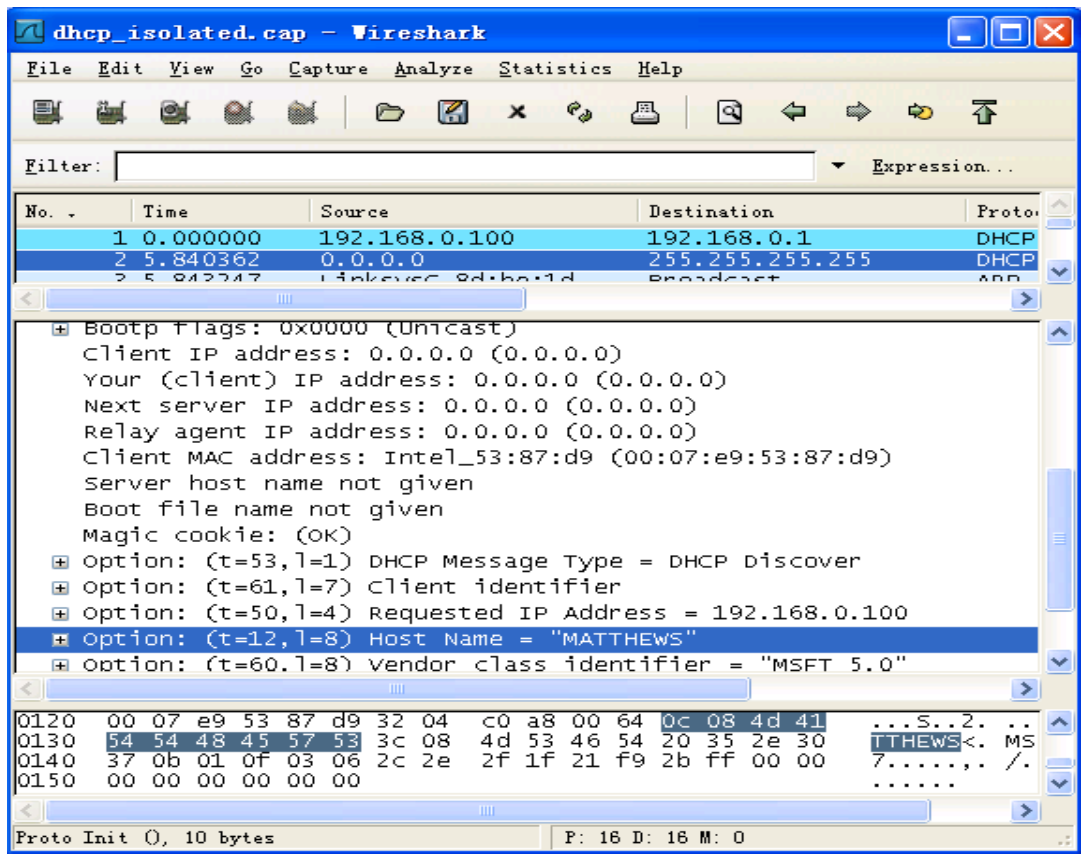


图 2 DHCP Discovery

客户端必须知道自己的子网掩码。子网掩码是一个 32 位的数，用来与 IP 地址进行“按

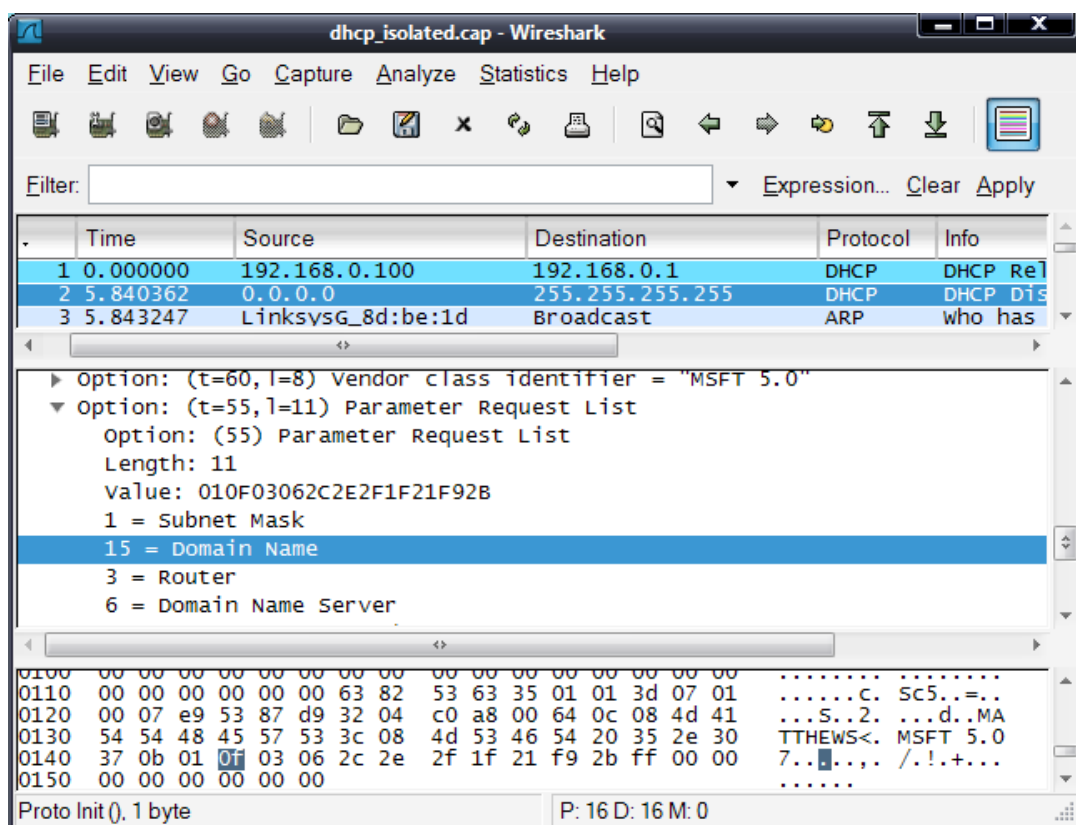


图 2 Parameter Request List

位逻辑与运算”从而得出网络地址。所有可以直接通信而不需要路由器参与的机器都有相同的网络地址。因此，子网掩码用来决定数据报是发送到本地路由器还是直接发送到本地目的主机。

客户端还必须知道它们的域名和它们在本地域名服务器上的标识。域名是一个可读的网络名。

IP 地址为 192.168.0.1 的 DHCP 服务器回复了一个 DHCP OFFER 报文。该报文也广播到 255.255.255.255，因为尽管客户端还不知道自己的 IP 地址，但它将接收到发送到广播地址的报文。这个报文中包含了客户端请求的信息，包括 IP 地址、本地路由器、子网掩码、域名和本地域名服务器。

在分组 5 中，客户端通过发送 DHCP Request（请求）报文表明自己接收到的 IP 地址。最后，在分组 6 中 DHCP 服务器确认请求的地址并结束对话。此后，在分组 7 中客户端开始使用它的新的 IP 地址作为源地址。

在分组 3 和分组 7 到 12 的地址 ARP 协议引起了我们的注意。在分组 3 中，DHCP 服务器询问是否有其它主机使用 IP 地址 192.168.0.100（该请求被发送到广播地址）。这就允许 DHCP 服务器在分配 IP 之前再次确认没有其它主机使用

该 IP 地址。在获取其 IP 地址之后，客户端会发送 3 个报文询问其他主机是否有与自己相同的 IP 地址。前 4 个 ARP 请求都没有回应。在分组 10—13 中，DHCP 服务器再次询问哪个主机拥有 IP 地址 192.168.0.100，客户端两次回答它占有该 IP 同时提供了自己的以太网地址。

通过 DHCP 分配的 IP 地址有特定的租用时间。为了保持对某个 IP 的租用，客户端必须更新租用期。当输入第二个命令 `ipconfig /renew` 后，在分组 14 和 15 中就会看到更新租用期的过程。DHCP Request 请求更新租用期。DHCP ACK 包括租用期的长度。如果在租用期到期之前没有 DHCP Request 发送，DHCP 服务器有权将该 IP 地址重新分配给其他主机。

最后，在分组 16 时输入命令 `ipconfig /release` 后的结果。在 DHCP 服务器接收到这个报文后，客户停止对该 IP 的使用。如有需要 DHCP 服务器有权重新对 IP 地址进行分配。

2、分析 IPv4 中的分片

在第二个实验中，我们将考察 IP 数据报首部。俘获此分组的步骤如下：

（1） 启动 Wireshark，开始分组俘获（“Capture”-----“interface...”-----“start”）。

（2） 启动 pingplotter（这是一个路由跟踪软件，可以发出各种类型的 ICMP 包用于网络测试和诊断。pingplotter 的下载地址为 www.pingplotter.com），在“Address to trace:”下面的输入框里输入目的地址，选择菜单栏“Edit”---“Options”---“Packet”，在“Packet size(in bytes defaults=56):”

右边输入 IP 数据报大小：5000，按下“OK”。最后按下按钮“Trace”，你将会看到 pingplotter 窗口显示如下内容，如图 3 所示：

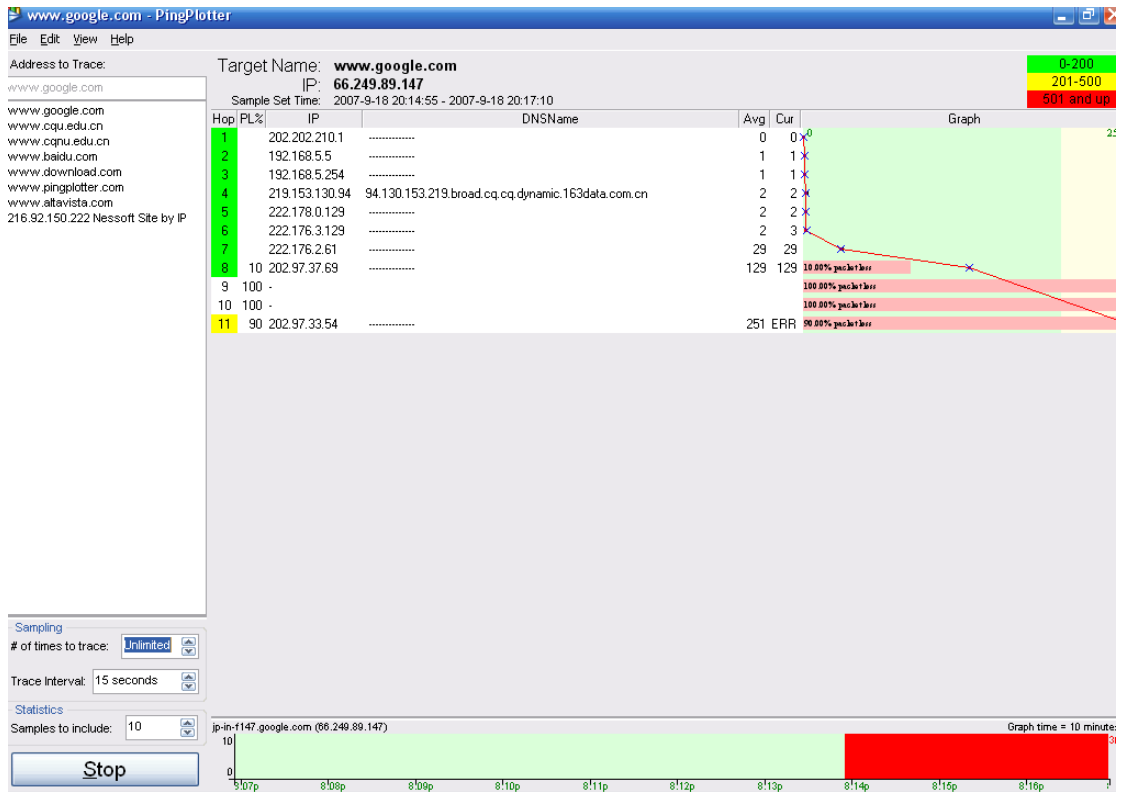


图 3 ping plotter

(3) 停止 Wireshark。设置过滤方式为：IP，在 Wireshark 窗口中将会看到如下情形,如图 4 所示。

在分组俘获中，你应该可以看到一系列你自己电脑发送的“ICMP Echo Request”和由中间路由器返回到你电脑的“ICMP TTL-exceeded messages”。

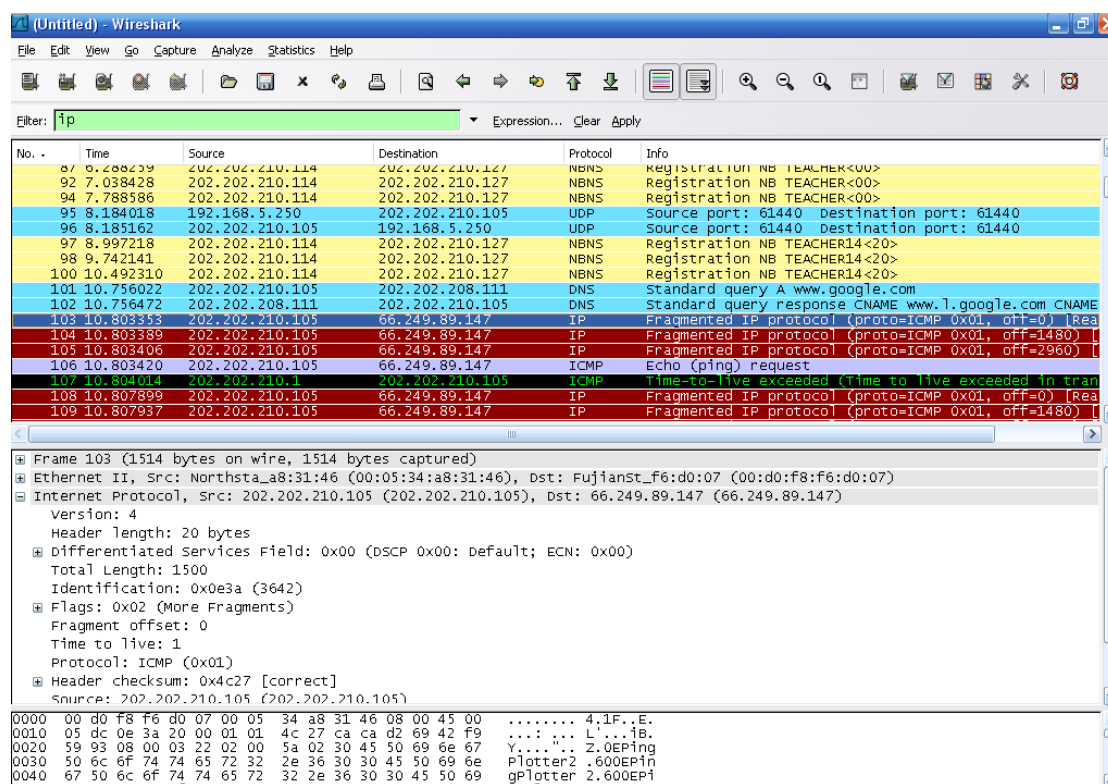


图 4 用 Wireshark 所俘获的分组

(4) 如果你无法得到上图的分组信息，也可使用已经下载的 IP 分片分组文件：fragment_5000_isolated.cap。然后在 Wireshark 中，选择菜单栏“File”--“Open”导入上述文件进行学习。

下面，我们来分析 fragment_5000_isolated.cap 中的具体分组：

IP 层位于传输层和链路层之间。在 fragment_5000_isolated.cap 中传输层协议是 UDP，链路层协议是以太网。发送两个 UDP 数据报，每个包含 5000 个字节的数据部分和 8 字节的 UDP 首部。在分组 1 到 4 和分组 5 到 8 分别代表了先后发送的两个 UDP 数据报。

当 IP 层接收到 5008 字节的 UDP 数据报时，它的工作是将其作为 IP 数据报在以太网传输。以太网要求一次传输的长度不大于 1514 个字节，其中有 14 字节是以太网帧首部。

IP 被迫将 UDP 数据报作为多个分片发送。每个分片必须包含以太网帧首部、IP 数据报首部。每个分片还会包含 UDP 数据报的有效负载（首部和数据）的一部分。

IP 将原始数据报的前 1480 个字节（含 1472 个字节的数据和含 8 字节的 UDP 首部）放在第一个分片中。后面两个分片每个均含 1480 个字节的数据，最

后一个分片中包含的数据为 568 个字节)。

为了让接收段重组原始数据, IP 使用首部的特殊字段对分片进行了编号。标识字段用于将所有的分片连接在一起。分组 1 到 4 含有相同的标识号 0xfd2b, 分组 5 到 8 的标识号是 0xfd2c。片偏移量指明了分组中数据的第一个字节在 UDP 数据报中的偏移量。例如分组 1 和分组 5 的偏移量都是 0, 因为它们都是第一个分片。

最后在标识字段中有一位用来指明这个分片后是否还有分片。分组 1 到分组 3 和分组 5 到分组 7 均对该位置进行了置位。分组 4 和分组 8 由于是最后一个分片而没有对该位置位。

四、实验报告内容

打开文件 dhcp_isolated.cap、fragment_5000_isolated.cap, 回答以下问题:

- 1、DHCP服务器广播的本地路由器或默认网关的IP地址是多少?
- 2、在dhcp_isolated.cap中, 由DHCP服务器分配的域名是多少?
- 3、在fragment_5000_isolated.cap中, 我们看到通过UDP数据报发送的5000字节被分成了多少分片? 在网络中, 一次能传输且不需要分片的最大数据单元有多大?

实验三 使用 Wireshark 分析 UDP 协议

一、实验目的

比较 TCP 和 UDP 协议的不同

二、实验环境

与因特网连接的计算机，操作系统为 Windows，安装有 Wireshark、IE 等软件。

三、实验步骤

1、打开两次 TCP 流的有关跟踪记录，保存在 tcp_2transmit.cap 中，并打开两次 UDP 流中的有关跟踪文件 udp_2transmit.cap 。如图所示：

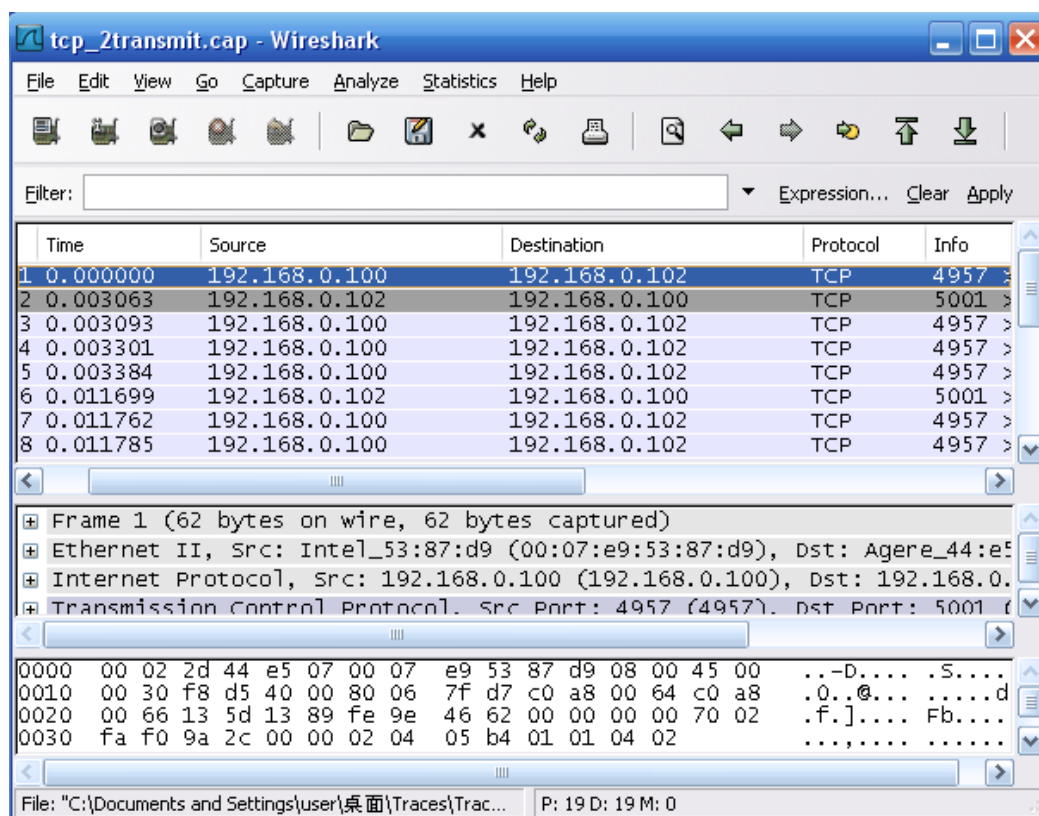


图 1 TCP 流跟踪记录

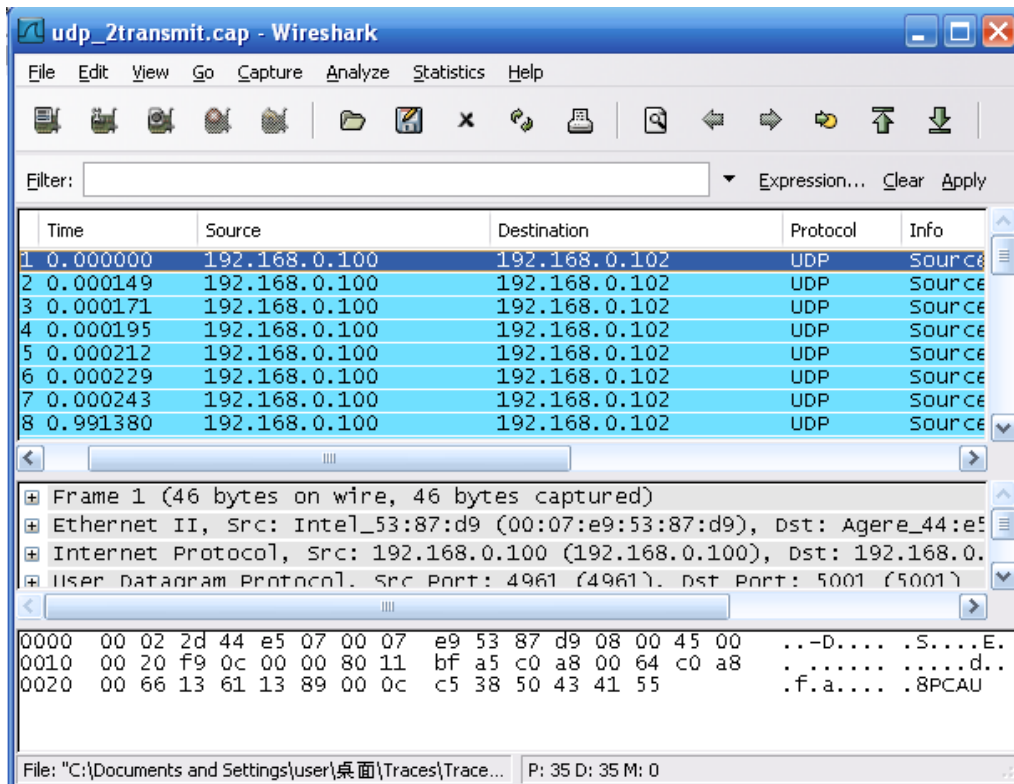


图 2 UDP 流跟踪记录

2、分析此数据包：

(1) TCP 传输的正常数据：

tcp_2transmit.cap 文件的分组 1 到 13 中显示了 TCP 连接。这个流中的大部分信息与前面的实验相同。我们在分组 1 到分组 3 中看到了打开连接的三次握手。分组 10 到分组 13 显示的则是连接的终止。我们看到分组 10 既是一个带有 FIN 标志的请求终止连接的分组，又是一个最后 1080 个字节的（序号是 3921—5000）的重传。

TCP 将应用程序写入合并到一个字节流中。它并不会尝试维持原有应用程序写入的边界值。我们注意到 TCP 并不会在单个分组中传送 1000 字节的应用程序写入。前 1000 个字节会在分组 4 中被发送，而分组 5 则包含了 1460 个字节的数据——一些来自第二个缓冲区，而另一些来自第三个缓冲区。分组 7 中含有 1460 个字节而分组 8 中则包含剩余的 1080 个字节。（ $5000-1000-1460-1460=1080$ ）

我们注意到实际报告上的 2.48 秒是从初始化连接的分组 1 开始到关闭连接的分组 10 结束。分组 11—13 未必要计入接收端应用程序的时间内，因为一旦接收到第一个 FIN，TCP 层便马上发送一个关闭连接的信号。分组 11—13 只可能由每台计算机操作系统得 TCP 层后台传输。

如果我们注意到第一个包含数据的分组 4 和最后一个分组 8 之间的时间,我们就大约计算出和由 UDP 接收端所报告的 0.01 秒相同的时间。这样的话,增加 TCP 传输时间的主要原因就是分组 10 中的重传。公平的说,UDP 是幸运的,因为它所有的分组都在第一时间被接受了。

在这个跟踪文件中,另一个值得注意的是没有包含数据的分组的数量。所有来自接收端的分组和几个来自发送端的分组只包含了 TCP 报文段的首部。总的来说(包括重传)一共发送了 6822 个字节来支持 5000 个字节的数据传输。这个开销正好 36%。

(2)UDP 正常数据传输

现在我们来观察 UDP 流,在 `udp_transmit.cap` 文件的分组 1 到分组 11 中显示。虽然像 TCP 流那样传输了相同的数据,但是在这个跟踪文件中还是很多的不同。

和 TCP 不同,UDP 是一个无连接的传输协议。TCP 用 SYN 分组和 SYN ACK 分组来显示地打开一个连接,而 UDP 却直接开始发送包含数据的分组。同样,TCP 用 FIN 分组和 FIN ACK 分组来显示地关闭一个连接,而 UDP 却只简单地停止包含数据的分组的传输。

为了解决这个问题,在文件 `udp_2transmit.cap` 俘获的分组中,采取的办法是 `ttcp` 发送端发送一个只包含 4 个字节的特殊 UDP 数据报来模拟连接建立和连接终止。在发送任何数据之前,发送端总是发送一个只包含 4 个字节的特殊数据报(分组 1),而在发送完所有的数据之后,发送端又发送额外的 5 个分组(分组 7-11)。

接收端也使用第一个特殊的数据报来启动数据传输的计时器。如果这个特殊的数据报丢失了,它可能用真实数据的第一个分组计时器。不过,如果接收端没有看到这个特殊的数据报,它就不能精确地确定数据传输的开始和传输的所有时间。与 TCP 不同,UDP 在传输的数据中,不会加上序号,因此对于接收端来说不可能确定丢失和重排序重传的情况。

类似的,接收端根据最后的特殊数据报来停止数据传输计时器。当接收端接收到这 5 个包中的任一个便停止计时,但是发送 5 个分组是因为在传输的过程中可能丢失其中的一些。如果 5 个分组全部丢失了,那么接收端便会无限制的等待

更多数据的到来达。

实际数据的传输是在分组 2-6 里。每一个分组都包含 1000 个字节。1000 个字节的应用程序写入直接转换成 UDP 数据报。另一方面，TCP 并不打算保存应用程序写入边界而只是将它们并入一个字节流中。

与 TCP 不同，UDP 没有提供接收端到发送端的反馈。在 TCP 的例子中，接收端返回只包含有 TCP 报文段首部而没有数据的报文段。首部本身则携带着关于哪些数据已经被成功接收以及接收端能够接收多少数据的信息。我们已经知道 UDP 不提供可靠的数据传输，因此并不要求什么数据已经被成功接收的信息。它也不提供任何信息高速发送端降低速率，因为接收端或者网络本身已经淹没。

虽然 UDP 本身并不提供接收端到发送端的反馈，但是我们确实看到几个从接收端到发送端的 ICMP 分组（分组 12—14）。ICMP 是网络层协议（IP）的一个伴随协议，并且有提供一定控制和错误报告的功能。在这种情况下，ICMP 分组暗示一些 UDP 数据报没有被传送到，因为端口不可达。这就意味着当数据报到达那个端口的时候，没有接收端在那个端口监听。我们注意到 ICMP 分组携带着一些未传递 UDP 数据报的信息。

当 tcp 接收端看到一个只具有 4 个字节数据的特殊数据报时，它便会知道数据传输是完整的，并且会因此关闭正在监听的端口。事实上 tcp 发送端发送 5 个这样的分组，并且后面的分组到达的时候发现接收端已经没有在监听了。当发送端发送所有的数据而没有相应的接收标志的时候，将会看到相似的行为。

TCP 和 UDP 的另外一个不同之处在于 TCP 连接时点对点的，换句话说，TCP 的使用是在一个连接端和一个发送端之间的。而对于 UDP 来说，一个发送端可能发向多个接收端（例如广播和组播通信）或者多个发送端能够发送给一个接收端。如果多个发送端都发给这个接收端的话，这个接收端会为每个发送端报告统计信息。

TCP 和 UDP 的最后一个不同之处是它们首部的大小。UDP 首部总是 8 个字节，而 TCP 首部大小是变化的，但是它绝对不会少于 20 个字节。这也就是说传输 5000 个字节的实际数据 TCP 的开销是 36%。

四、实验报告

回答下面的问题：

1、在 `udp_2transmit.cap` 中观察 DUP 首部。长度字段是包括首部和数据还是只包括数据？

2、我们观察到使用 ICMP 报文来报告 UDP 数据报不可达。为什么 TCP 不用这个来指示丢失的报文段呢？

3、我们计算 TCP 成功传输 5000 个字节的实际数据的开销是 36%。在这个开销中都包括什么？如果没有重传，这个开销是多少？

实验四 使用 Wireshark 分析 TCP 协议

一、实验目的

分析 TCP 协议

二、实验环境

与因特网连接的计算机，操作系统为 Windows，安装有 Wireshark、IE 等软件。

三、实验步骤

1、TCP 介绍

（1）连接建立：

TCP 连接通过称为三次握手的三条报文来建立的。在 Wireshark 中选择 open->file,选择文件 tcp_pcattcp_n1.cap，其中分组 3 到 5 显示的就是三次握手。

第一条报文没有数据的 TCP 报文段，并将首部 SYN 位设置为 1。因此，第一条报文常被称为 SYN 分组。这个报文段里的序号可以设置成任何值，表示后续报文设定的起始编号。连接不能自动从 1 开始计数，选择一个随机数开始计数可避免将以前连接的分组错误地解释为当前连接的分组。观察分组 3，Wireshark 显示的序号是 0。选择分组首部的序号字段，原始框中显示“94 f2 2e be”。Wireshark 显示的是逻辑序号，真正的初始序号不是 0。如图 1 所示：

The image shows a Wireshark capture of a TCP connection. The filter is set to 'not arp'. The packet list shows 16 packets, all TCP, between 192.168.0.100 and 192.168.0.102. The packet details pane shows the raw data for packet 4, which is a SYN packet. The raw data is displayed in hexadecimal and ASCII. The sequence number (tcp.seq) is 4 bytes, and the initial sequence number (P: 16 D: 14 M: 0) is shown.

| Time | Source | Destination | Protocol | Info |
|-------------|---------------|---------------|----------|--------|
| 3 0.002936 | 192.168.0.100 | 192.168.0.102 | TCP | 2440 > |
| 4 0.005476 | 192.168.0.102 | 192.168.0.100 | TCP | 5001 > |
| 5 0.005500 | 192.168.0.100 | 192.168.0.102 | TCP | 2440 > |
| 6 0.005897 | 192.168.0.100 | 192.168.0.102 | TCP | 2440 > |
| 7 0.005919 | 192.168.0.100 | 192.168.0.102 | TCP | 2440 > |
| 8 0.014136 | 192.168.0.102 | 192.168.0.100 | TCP | 5001 > |
| 9 0.014189 | 192.168.0.100 | 192.168.0.102 | TCP | 2440 > |
| 10 0.014212 | 192.168.0.100 | 192.168.0.102 | TCP | 2440 > |
| 11 0.014226 | 192.168.0.100 | 192.168.0.102 | TCP | 2440 > |
| 12 0.023425 | 192.168.0.102 | 192.168.0.100 | TCP | 5001 > |
| 13 0.023465 | 192.168.0.100 | 192.168.0.102 | TCP | 2440 > |
| 14 0.028116 | 192.168.0.102 | 192.168.0.100 | TCP | 5001 > |
| 15 0.029205 | 192.168.0.102 | 192.168.0.100 | TCP | 5001 > |
| 16 0.029216 | 192.168.0.100 | 192.168.0.102 | TCP | 2440 > |

| Offset | Hex | ASCII |
|--------|---|------------------|
| 0000 | 00 02 2d 44 e5 07 00 07 e9 53 87 d9 08 00 45 00 | ...-D....S....E. |
| 0010 | 00 30 d0 28 40 00 80 06 a8 84 c0 a8 00 64 c0 a8 | .0.(@.....d.. |
| 0020 | 00 66 09 88 13 89 94 f2 2e be 00 00 00 00 70 02 | .f.....p. |
| 0030 | fa f0 25 52 00 00 02 04 05 b4 01 01 04 02 | ..%R..... |

Sequence number (tcp.seq), 4 bytes P: 16 D: 14 M: 0

图 1：逻辑序号与实际初始序号

SYN 分组通常是从客户端发送到服务器。这个报文段请求建立连接。一旦成功建立了连接，服务器进程必须已经在监听 SYN 分组所指示的 IP 地址和端口号。如果没有建立连接，SYN 分组将不会应答。如果第一个分组丢失，客户端通常会发送若干 SYN 分组，否则客户端将会停止并报告一个错误给应用程序。

如果服务器进程正在监听并接收到来的连接请求，它将以一个报文段进行相应，这个报文段的 SYN 位和 ACK 位都置为 1。通常称这个报文段为 SYNACK 分组。SYNACK 分组在确认收到 SYN 分组的同时发出一个初始的数据流序号给客户端。

分组 4 的确认号字段在 Wireshark 的协议框中显示 1，并且在原始框中的值是“94 f2 2e bf”（比“94 f2 2e be”多 1）。这解释了 TCP 的确认模式。TCP 接收端确认第 X 个字节已经收到，并通过设置确认号为 X+1 来表明期望收到下一个字节号。分组 4 的序号字段在 Wireshark 的协议显示为 0，但在原始框中的实际值却是“84 ca be b3”。这表明 TCP 连接的双方会选择数据流中字节的起始编号。所有初始序号逻辑上都视同为序号 0。

最后，客户端发送带有标志 ACK 的 TCP 报文段，而不是带 SYN 的报文段

来完成三次握手的过程。这个报文段将确认服务器发送的 SYNACK 分组，并检查 TCP 连接的两端是否正确打开合运行。

（2）关闭连接

当两端交换带有 FIN 标志的 TCP 报文段并且每一端都确认另一端发送的 FIN 包时，TCP 连接将会关闭。FIN 位字面上的意思是连接一方再也没有更多新的数据发送。然而，那些重传的数据会被传送，直到接收端确认所有的信息。在 tcp_pcattcp_n1.cap 中，通过分组 13 至 16 我们可以看到 TCP 连接被关闭。

2、TCP 重传

当一个 TCP 发送端传输一个报文段的同时也设置了一个重传计时器。当确认到达时，这个计时器就自动取消。如果在数据的确认信息到达之前这个计时器超时，那么数据就会重传。

重传计时器能够自动灵活设置。最初 TCP 是基于初始的 SYN 和 SYN ACK 之间的时间来设置重传计时器的。它基于这个值多次设置重传计时器来避免不必要的重传。在整个 TCP 连接中，TCP 都会注意每个报文段的发送和接到相应的确认所经历的时间。TCP 在重传数据之前不会总是等待一个重传计算器超时。TCP 也会把一系列重复确认的分组当作是数据丢失的征兆。

在 Wireshark 中选择 file- > open, 打开文件 pcattcp_retrans_t.cap 和 pcattcp_retrans_r.cap，对所俘获的分组进行分析如下：

（1） SACK 选项协商

在上面的每次跟踪中，我们能观察建立连接的三次握手。在 SYN 分组中，发送端在 TCP 的首部选项中通过包括 SACK permitted 选项来希望使用 TCP SACK。在 SYN ACK 包中接收端表示愿意使用 SACK。这样双方都同意接收选择性确认信息。SACK 选项如图 2 所示：

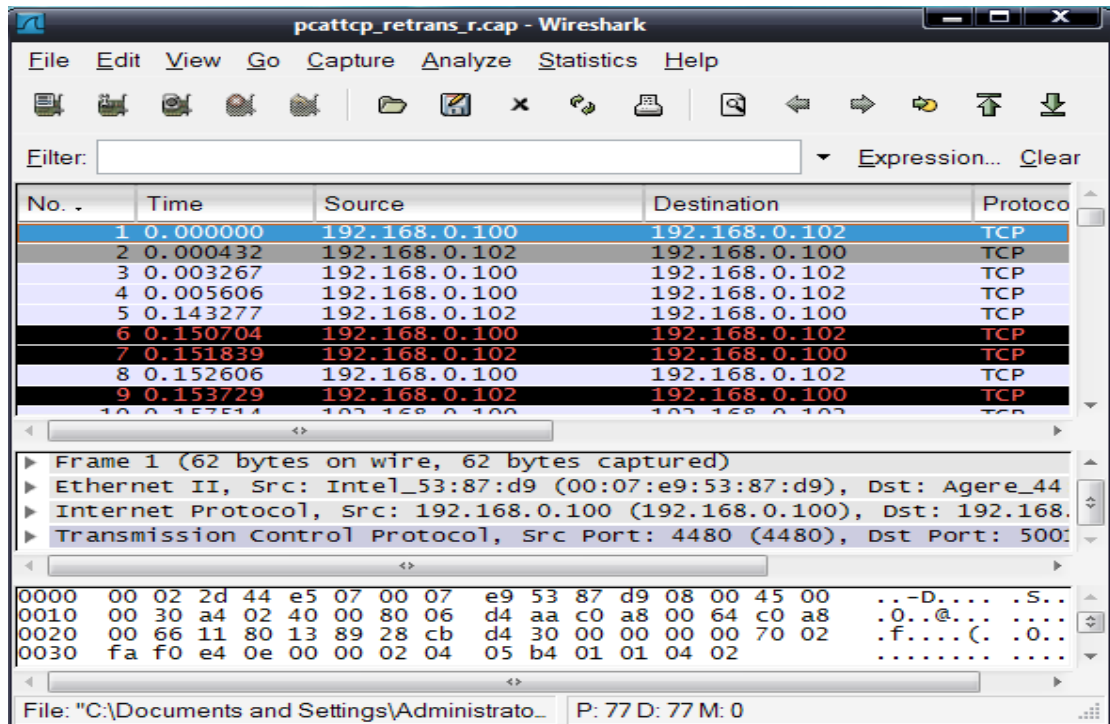


图 2 SACK 选项

在 TCP SACK 选项中，如果连接的一端接收了失序数据，它将使用选项区字段来发送关于失序数据起始和结束的信息。这样允许发送端仅仅重传丢失的数据。TCP 接收端不能传递它们接收到的失序数据给处于等待状态的应用程序，因为它总是传递有序数据。因此，接收到的失序数据要么被丢掉，要么被存储起来。

接收端的存储空间是有限的，TCP 发送端必须保存一份已发送的数据的副本，以防止数据需要重发。发送端必须保存数据直到它们收到数据的确认信息为止。

接收端通常会分配一个固定大小的缓冲区来存储这些失序数据和需要等待一个应用程序读取的数据。如果缓冲区空间不能容纳下更多数据，那么接收端只有将数据丢弃，即使它是成功到达的。接收端的通知窗口字段用来通知发送端还有多少空间可以用于输入数据。如果数据发送的速度快于应用程序处理数据的速度，接收端就会发送一些信息来告知发送端其接收窗口正在减小。在这个跟踪文件中，接收端通知窗口的大小是变化的，从 16520 个字节到 17520 个字节。

TCP 发送端在发送之前有一个容纳数据的有限空间。然而，和接收端不同的是，发送端是限制自己的发送速率。如果缓冲区的空间满了，尝试写入更多数据的应用程序将被阻塞直到有更多的空间可以利用为止。

(2) 分组的丢失与重传

用显示过滤器 `tcp.analysis.retransmission` 搜索重传。在 `pcattcp_retrans_t.cap` 中应用该过滤器,在这个跟踪文件中,我们看见分组 12 是这 9 次重传的第一次。如图所示 3:

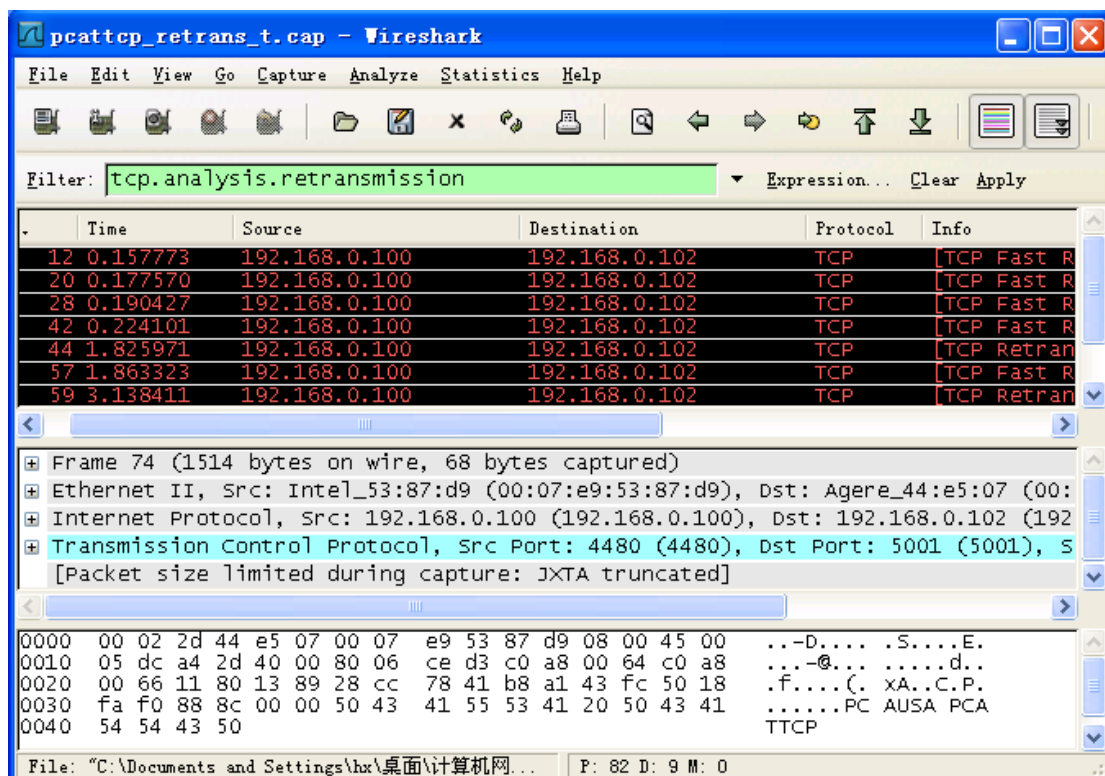


图 3 pcattcp_retrans_t.cap 中 9 次重传

通过观察分组 12 的细节,我们发现序号是 1001,我们发现分组 5 也有同样的序号。有趣的是,分组 5 是对 1001 到 2460 号字节的传输,而分组 12 却是对 1001 到 2000 号字节的重传。分组 20 是对 2001 到 2460 号字节的重传。

分组 4 是对 1 到 1000 号字节的传输,分组 5 是对 1001 到 2460 号字节的传输,分组 7 是对 2461 到 3920 号字节的传输。

我们已检查了发送端上获取的所有跟踪记录。我们从接收端的角度来看同一个连接,我们会发现有些不同。在 `pcattcp_retrans_r.cap` 中,我们发现 1 到 1000 号字节是在分组 4 里被传送的,而 2461 到 3920 号字节是在分组 6 (而不是分组 7) 中被传送的。在这个跟踪文件中,分组 5 是 1 到 1000 号字节的确认。我们没有看到 1001 到 2460 号字节的传输。但是他们确实被传输了,只是在发送端和接收端的某个环节丢失了。

现在我们来查看接收端是如何处理这些丢失字节的。在分组 4 达到以后,接收

端会以确认号 1001（分组 5）作为响应。在分组 6 的 2461 到 3920 号字节达到之后，接收端仍然以确认号 1001（分组 7）作为响应。即使它接到的是附加数据，确认号仍然是它期望收到的下一个有序字节的序号。同样在含有 3921 到 5381 号字节的分组 8 到达之后，接收端仍然以 1001 响应。

最后，1001 到 2000 号字节被重传。在这两次跟踪中，我们都在分组 12 里看到这种情况。于是接收端增加它的确认号到 2001。

最终 2001 到 2460 号字节被重传。在这次重传之后，接收端可以立即从确认字节 2001 跳到确认字节 11221。

四、实验报告内容

在实验的基础上，回答以下问题：

1、客户服务器之间用于初始化 TCP 连接的 TCP SYN 报文段的序号（sequence number）是多少？在该报文段中，是用什么来标识该报文段是 SYN 报文段的？

2、服务器向客户端发送的 SYNACK 报文段序号是多少？该报文段中，Acknowledgement 字段的值是多少？

3、找出 `pcattcp_retrans_t.cap` 中所有在到达接收端之前丢失的分组。对于每个丢失的报文段，找出重传分组（提示：找出第一个丢失的字节和重传它们的分组）。

4、当接收端发送一个 TCP 报文段来确认收到的数据时，这个报文段也可能丢失。在 `pcattcp_retrans_t.cap` 和 `pcattcp_retrans_r.cap` 中存在这样的丢失吗？你是怎么得到这样的答案的？

实验五 利用 Wireshark 分析协议 HTTP

一、实验目的

分析 HTTP 协议

二、实验环境

与因特网连接的计算机，操作系统为 Windows，安装有 Wireshark、IE 等软件。

三、实验步骤

1、利用 Wireshark 俘获 HTTP 分组

(1) 在进行跟踪之前，我们首先清空 Web 浏览器的高速缓存来确保 Web 网页是从网络中获取的，而不是从高速缓冲中取得的。之后，还要在客户端清空 DNS 高速缓存，来确保 Web 服务器域名到 IP 地址的映射是从网络中请求。在 WindowsXP 机器上，可在命令提示行输入 ipconfig/flushdns 完成操作。

(2) 启动 Wireshark 分组俘获器。

(3) 在 Web 浏览器中输入：<http://www.google.com>

(4) 停止分组俘获。

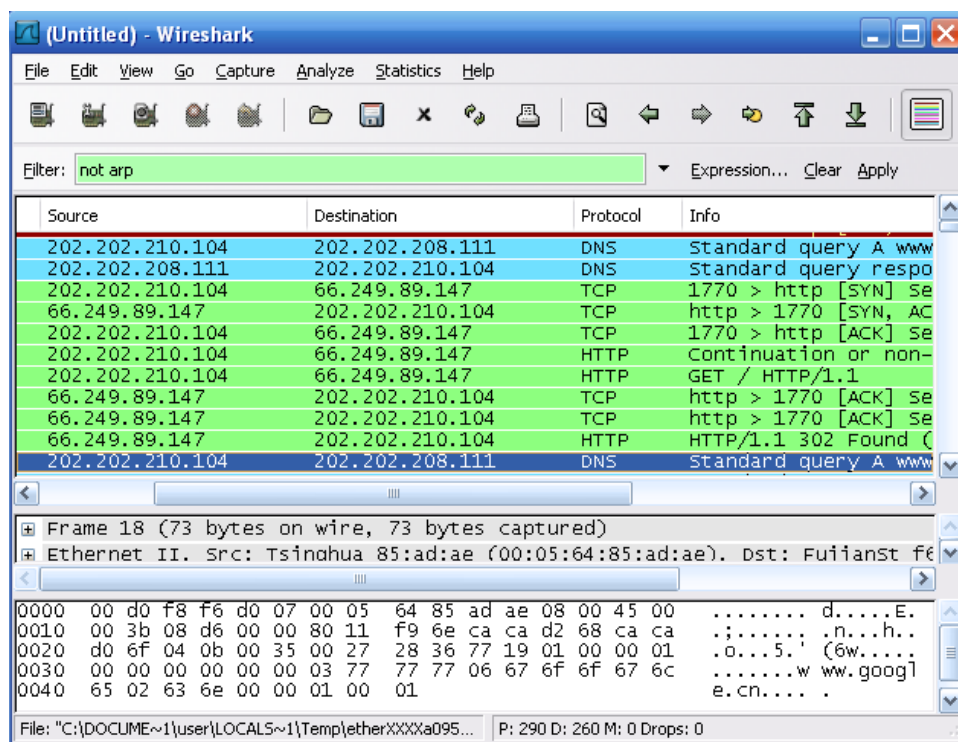


图 1 利用 Wireshark 俘获的 HTTP 分组

在 URL <http://www.google.com> 中, www.google.com 是一个具体的 web 服务器的域名。最前面有两个 DNS 分组。第一个分组是将域名 www.google.com 转换成对应的 IP 地址的请求, 第二个分组包含了转换的结果。这个转换是必要的, 因为网络层协议——IP 协议, 是通过点分十进制来表示因特网主机的, 而不是通过 www.google.com 这样的域名。当输入 URL `http: //www.google.com` 时, 将要求 Web 服务器从主机 www.google.com 上请求数据, 但首先 Web 浏览器必须确定这个主机的 IP 地址。

随着转换的完成, Web 浏览器与 Web 服务器建立一个 TCP 连接。最后, Web 浏览器使用已建立好的 TCP 连接来发送请求“GET/HTTP/1.1”。这个分组描述了要求的行为(“GET”)及文件(只写“/”是因为我们没有指定额外的文件名), 还有所用到的协议的版本(“HTTP/1.1”)。

2、HTTP GET/response 交互

(1) 在协议框中, 选择“GET/HTTP/1.1”所在的分组会看到这个基本请求行后跟随着一系列额外的请求首部。在首部后的“\r\n”表示一个回车和换行, 以此将该首部与下一个首部隔开。

“Host”首部在 HTTP1.1 版本中是必须的, 它描述了 URL 中机器的域名, 本例中是 www.google.com。这就允许了一个 Web 服务器在同一时间支持许多不同的域名。有了这个数不, Web 服务器就可以区别客户试图连接哪一个 Web 服务器, 并对每个客户响应不同的内容, 这就是 HTTP1.0 到 1.1 版本的主要变化。

User-Agent 首部描述了提出请求的 Web 浏览器及客户机器。

接下来是一系列的 Accpet 首部, 包括 Accept (接受)、Accept-Language (接受语言)、Accept-Encoding (接受编码)、Accept-Charset (接受字符集)。它们告诉 Web 服务器客户 Web 浏览器准备处理的数据类型。Web 服务器可以将数据转变为不同的语言和格式。这些首部表明了客户的能力和偏好。

Keep-Alive 及 Connection 首部描述了有关 TCP 连接的信息, 通过此连接发送 HTTP 请求和响应。它表明在发送请求之后连接是否保持活动状态及保持多久。大多数 HTTP1.1 连接是持久的 (persistent), 意思是在每次请求后不关闭 TCP 连接, 而是保持该连接以接受从同一台服务器发来的多个请求。

(2) 我们已经察看了由 Web 浏览器发送的请求, 现在我们来观察 Web 服务器的回答。响应首先发送“HTTP/1.1 200 ok”, 指明它开始使用 HTTP1.1 版本

来发送网页。同样，在响应分组中，它后面也跟随着一些首部。最后，被请求的实际数据被发送。

第一个 Cache-control 首部，用于描述是否将数据的副本存储或高速缓存起来，以便将来引用。一般个人的 Web 浏览器会高速缓存一些本机最近访问过的网页，随后对同一页面再次进行访问时，如果该网页仍存储于高速缓存中，则不再向服务器请求数据。类似地，在同一个网络中的计算机可以共享一些存在高速缓存中的页面，防止多个用户通过到其他网路的低速网路连接从网上获取相同的数据。这样的高速缓存被称为代理高速缓存（proxy cache）。在我们所俘获的分组中我们看到“Cache-control”首部值是“private”的。这表明服务器已经对这个用户产生了一个个性化的响应，而且可以被存储在本地的高速缓存中，但不是共享的高速缓存代理。

在 HTTP 请求中，Web 服务器列出内容类型及可接受的内容编码。此例中 Web 服务器选择发送内容的类型是 text/html 且内容编码是 gzip。这表明数据部分是压缩了的 HTML。

服务器描述了一些关于自身的信息。此例中，Web 服务器软件是 Google 自己的 Web 服务器软件。响应分组还用 Content-Length 首部描述了数据的长度。最后，服务器还在 Date 首部中列出了数据发送的日期和时间。

根据俘获窗口内容，回答“四、实验报告内容”中的 1-6 题。

3、HTTP 条件 GET/response 交互

- (1) 启动浏览器，清空浏览器的缓存。
- (2) 启动Wireshark分组俘获器，开始Wireshark分组俘获。
- (3) 在浏览器地址栏中如下网址：

<http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html>

你的浏览器中将显示一个具有五行的非常简单的 HTML 文件。

- (4) 在你的浏览器中重新输入相同的 URL 或单击浏览器中的“刷新”按钮。
- (5) 停止Wireshark分组俘获，在显示过滤筛选说明处输入“http”，分组列表子窗口中将只显示所俘获到的HTTP报文。

根据操作回答“四、实验报告内容”中的 7-10 题。

4、获取长文件

- (1) 启动浏览器，将浏览器的缓存清空。

(2) 启动 Wireshark 分组俘获器, 开始 Wireshark 分组俘获。

(3) 在浏览器地址栏中输入如下网址:

<http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html>

浏览器将显示一个相当大的美国权力法案

(4) 停止 Wireshark 分组俘获, 在显示过滤筛选说明处输入 “http”, 分组列表子窗口中将只显示所俘获到的 HTTP 报文。

根据操作回答 “四、实验报告内容” 中的 11-14 题。

5、嵌有对象的 HTML 文档

(1) 启动浏览器, 将浏览器的缓存清空。

(2) 启动 Wireshark 分组俘获器。开始 Wireshark 分组俘获。

(3) 在浏览器地址栏中输入如下网址:

<http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file4.html>

浏览器将显示一个具有两个图片的短HTTP文件。

(4) 停止 Wireshark 分组俘获, 在显示过滤筛选说明处输入 “http”, 分组列表子窗口中将只显示所俘获到的 HTTP 报文。

根据操作回答 “四、实验报告内容” 中的 15-16 题。

6、HTTP 认证

(1) 启动浏览器, 将浏览器的缓存清空。

(2) 启动 Wireshark 分组俘获器。开始 Wireshark 分组俘获。

(3) 在浏览器地址栏中输入如下网址:

http://gaia.cs.umass.edu/wireshark-labs/protected_pages/HTTP-wireshark-file5.html

浏览器将显示一个HTTP文件, 输入所需要的用户名和密码(用户名: wireshark-students, 密码: network)。

(4) 停止 Wireshark 分组俘获, 在显示过滤筛选说明处输入 “http”, 分组列表子窗口中将只显示所俘获到的 HTTP 报文。

根据操作回答 “四、实验报告内容” 中的 17-18 题。

四、实验报告内容

在实验的基础上, 回答以下问题:

(1) 你的浏览器运行的是 HTTP1.0, 还是 HTTP1.1? 你所访问的服务器所运行的 HTTP 版本号是多少?

(2) 你的浏览器向服务器指出它能接收何种语言版本的对象?

(3) 你的计算机的 IP 地址是多少? 服务器 `gaia.cs.umass.edu` 的 IP 地址是多少?

(4) 从服务器向你的浏览器返回的状态代码是多少?

(5) 你从服务器上所获取的 HTML 文件的最后修改时间是多少?

(6) 返回到你的浏览器的内容以供多少字节?

(7) 分析你的浏览器向服务器发出的第一个 HTTP GET 请求的内容, 在该请求报文中, 是否有一行是: `IF-MODIFIED-SINCE`?

(8) 分析服务器响应报文的内容, 服务器是否明确返回了文件的内容? 如何获知?

(9) 分析你的浏览器向服务器发出的第二个 “HTTP GET” 请求, 在该请求报文中是否有一行是: `IF-MODIFIED-SINCE`? 如果有, 在该首部行后面跟着的信息是什么?

(10) 服务器对第二个 HTTP GET 请求的响应中的 HTTP 状态代码是多少? 服务器是否明确返回了文件的内容? 请解释。

(11) 你的浏览器一共发出了多少个 HTTP GET 请求?

(12) 承载这一个 HTTP 响应报文一共需要多少个 data-containing TCP 报文段?

(13) 与这个 HTTP GET 请求相对应的响应报文的状态代码和状态短语是什么?

(14) 在被传送的数据中一共有多少个 HTTP 状态行与 “TCP-induced continuation” 有关?

(15) 你的浏览器一共发出了多少个 HTTP GET 请求? 这些请求被发送到的目的地的 IP 地址是多少?

(16) 浏览器在下载这两个图片时, 是串行下载还是并行下载? 请解释。

(17) 对于浏览器发出的最初的 HTTP GET 请求, 服务器的响应是什么 (状态代码和状态短语)?

(18) 当浏览器发出第二个 HTTP GET 请求时，在 HTTP GET 报文中包含了哪些新的字段？