華中科技大學

“网络安全综合实验（I）”实验指导

题目：网络流量分析

1　实验九：网络流量分析

1.1　实验环境及要求

1.1.1　实验平台及说明

实验软件：Wireshark 3.0.3；

操作系统：windows；

参考资料：

1、Chirs Sanders 著，诸葛建伟等译，Wireshark 数据包分析实战，人民邮电出版社，2018

2、Wireshark 在线帮助

3、课程群文件共享资料

4、其他在线文档资源。

提交时间及文件说明：本实验环节，每位同学提交独立完成的实验报告电子版一份；按指导老师要求的时间和方式提交；文件名：U2019XXXX（学号）-姓名-网 络安全综合实验（I）网络流量分析实验。

报告格式要求：正文为宋体小 4 号，段首缩进 2 字符汉字，行间距 1 倍行距，字 符间距为标准；图保证清晰大小合适；每页尽量不留大段空白。图片需要编号及 命名；正文、图片、参考文献的格式，请参考华中科技大学毕业论文规范中关于排版的要求。

文档中包含的内容：

1 封面首页信息及作者、完成时间； 2 小结：总体感受、实验中遇到的最突出问题及收获、对实验环节的意见和建议；4 实验中为解决问题，查阅资料，请记录资料出处，包括资料名次、页码、网址，作为 参考文献部分列表给出； 5 参考网络上资料的，请通过浏览器的打印功能，以 pdf 文件方式保存；资料可归档为：参考资料.zip，与报告一并提交。

学习通要求：实验过程中，请各位同学按照实验指导手册中红色文字部分（例如：【验证实验 1】）的要求截图和回答问题，并将问题的答案提交至“学习通软件”。

1.2　实验任务

本次实验主要对截获的网络流量进行分析，如何截获网络报文，可以参考其他老师的介

绍。

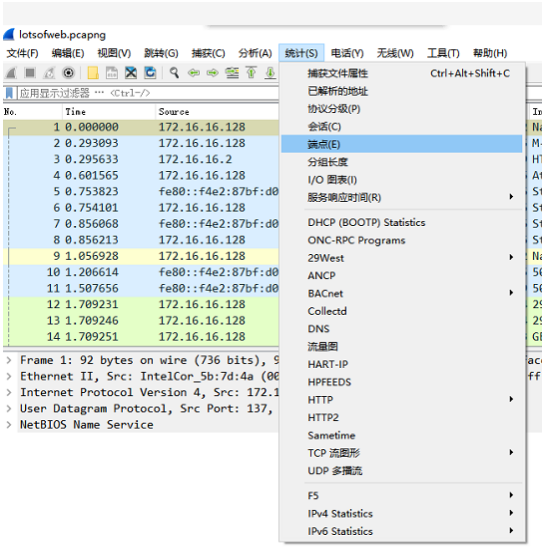
1.2.1 任务 1 端点和会话统计分析

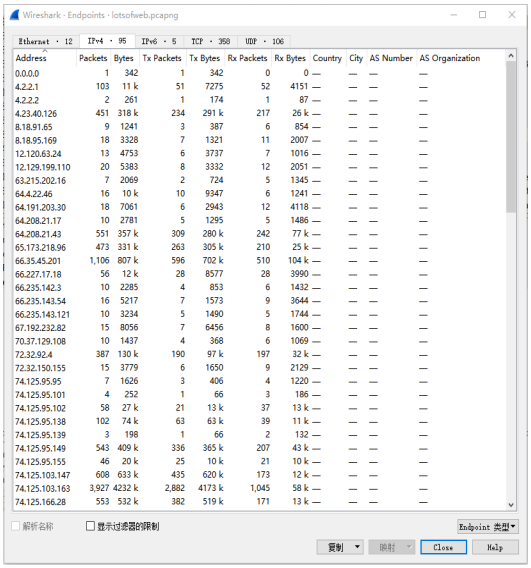
要想让网络通信正常进行，必须至少拥有两台设备进行数据流的交互。端点（endpoint）就是指网络上能够发送或者接受数据的一台设备。两个端点之间的通信被称为会话（conversation）。Wireshark 根据交互的特性来标识端点会话，特别是在多种协议之间所使用的的地址。

1. 查看端点统计

当分析流量时，你也许会察觉到可以将问题定位到网络中的一个特定端点上去。此时就需要用到 Wireshark 的***查看端点统计***功能了。

使用 Wireshark 打开捕获的数据包文件 *lotsofweb.pcapng*，然后在打开 Wireshark 中打开***端点***窗口。





窗口顶部的选项卡根据协议将当前捕获数据中所有支持和被识别的端点进行分类，单击某一个选项卡就可以只显示针对一个具体协议的端点。

窗口右下角的 ***Endpoints 类型***多选框可以添加额外的协议过滤标签；

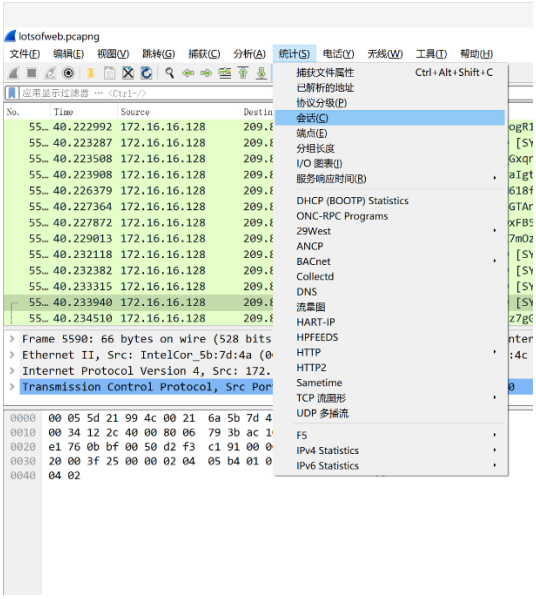
勾选***解析名称***复选框，可以开启名称解析功能来查看端点名称；

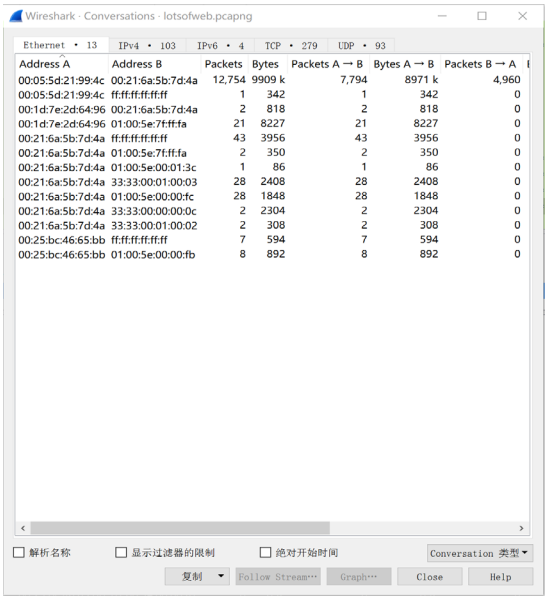
勾选***显示过滤器的限制***复选框，可以让端点窗口只显示与显示过滤器相匹配的端点。

另外一个非常便利的功能是可以使用端点窗口将特定的数据包过滤出来，使其显示在数据包列表（Packet List）面板中，使用这种方法可以快速定位与某个特定端点相关联的数据包。具体的方法是右键单击一个特定端点，然后再菜单中选择相应的选项。

2. 查看网络会话

打开捕获文件 ***lotsofweb.pcapng*** 后，访问 Wireshark 的***网络会话***窗口。





网络会话窗口看起来和查看端点窗口有些类似，该窗口显示的是一行由两个地址组成的会话，以及每个设备发送或者接收到的数据包和字节数等等信息。地址 A 代表源端点，地址 B 代表目的端点。

网络会话窗口中列出的会话以不同的协议分开。要查看针对特定协议的会话，可以单击窗口顶部的选项卡进行切换，或者在右下角的 Conversation 类型多选框中增加一个其它的协议类型。

与在端点窗口中的操作一样，在网络会话窗口中也可以使用解析名称、显示过滤器限制功能，或者右键单击一个会话来在数据包列表面板中创建基于该会话的过滤器。基于网络会话方式过滤流量可以帮助分析者深入研究一些有趣的信息交互序列中的细节问题。

3. 使用端点和网络会话窗口定位最高用量者

端点和会话窗口是排查网络问题的得力助手，特别是当你试图寻找网络中巨大流量的源头的时候。类似的问题是网络管理员非常关心的问题，它需要弄清楚他所管理的网络中每天产生的巨大流量到底来自于哪里，这些流量在干什么，特别是当某一天的流量出现异常增加的时候，就需要我们有这样的能力进行分析。

进行异常流量溯源的基本步骤：

1）通过端点窗口找到发送/接受数据包最多，数据量最大的端点们；例如，找到通信量最大的两个端点，我们可以假设他们正在各自与很多其它的设备进行大量通信，或者，这两个 IP 之间正在进行彼此的通信。实际上，最大通信量之间的端点通信是比

较常见的。

2）打开会话窗口就可以验证多个最大通信量端点之间是否存在通信，通信量达到什么程度。同时，将该会话作为过滤器应用，就可以在数据包面板中过滤出该会话的所有通信数据包，以进行进一步的分析。

【验证实验 1】

当发现网络中出现异常增加的流量之后，管理员使用 Wireshark 进行了数据包嗅探捕获，***lotsofweb.pcapng*** 就是数据包捕获文件，请对该数据包捕获文件进行分析，并回答下列问题：

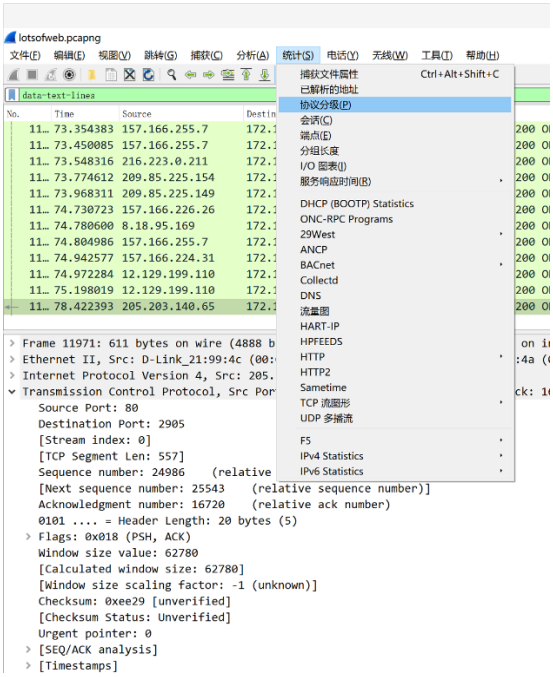
1）捕获文件反映出来的通信量最大、最活跃的端点有哪些？请提供关键分析证据截图。

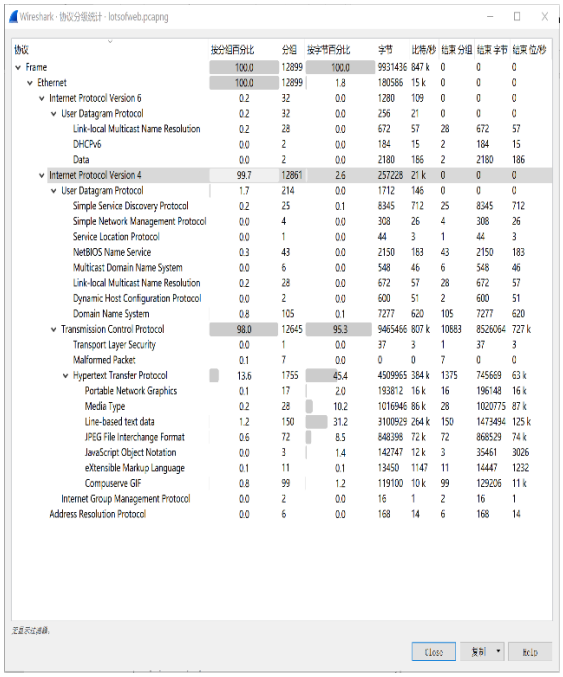
2）最大的数据通信量来自于哪些节点的通信？请提供关键分析证据截图。

3）对通信量最大的端点之间的会话进行分析，通信数据又什么特点呢？请提供关键分析证据截图。

4. 扩展阅读——基于协议分层结构的统计

当在与未知的捕获文件打交道时，有时候需要知道文件中协议的分布情况，也就是捕获文件中 TCP、IP、DHCP 等所占百分比的多少，除了计算并汇总数据包之外，使用 Wireshark 的协议分级统计窗口也是一个对网络进行基准分析的好方法。





这些信息给了我们了解我们的网络的一个很好的方法，它帮助我们建立了一个正常网络

的流量大致应该是个什么样子的印象，当某一天的流量发生不同于正常情况的变化时，就应该引起我们足够的重视了。例如：通常情况下你的网络中有 10%的流量是 ARP 流量，但是某一段时间，你发现网络中 ARP 力量的比例突然增大，这意味着什么呢？

【验证实验 2】

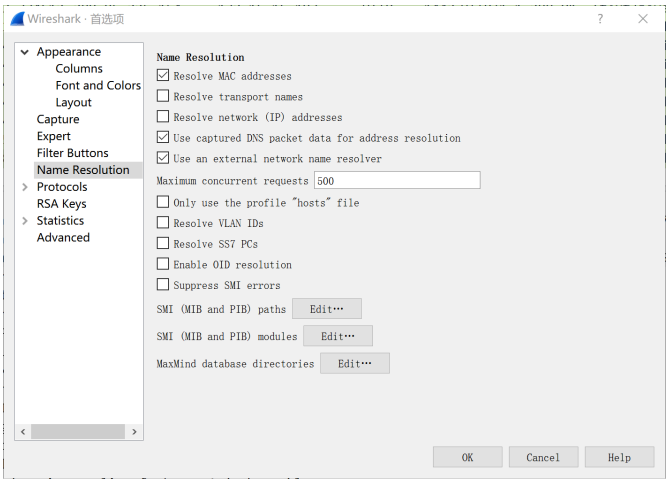
我们通过 Wireshark 的协议分层结构统计功能发现某一段时间内 ARP 数据包的数量大增，利用过滤器将这一段时间内的 arp 数据包过滤出来，保存在捕获文件 ***lotsofarp.pcapng*** 中，请你对该捕获文件进行分析，你有什么猜测或者结论呢？请给出你的理由并提供证据截图。

1.2.2 任务 2 关于名称解析

1. 名称解析

网络数据通过使用各种字母、数字组成的寻址系统进行传输，但这些地址系统通常都应为太长或者太复杂而不容易被记住。名称解析就是一个协议，用来将一个独特的地址转换为另外地址，其目的是为了让地址方便记忆。

Wireshark 在分析捕获的数据包时，往往使用名称解析来简化分析过程。



1）**解析 MAC 地址（resolve MAC address）：**这种类型的名称解析使用 ARP 协议，试图将第 2 层的物理地址转换为网络层的 IP 地址。如果这种尝试失败，那么 Wireshark 会使用程序目录中的 ethers 文件尝试进行转换。

2）**解析传输名称（resolve transport name）：**这种类型的名称解析尝试将一个端口号转换成一个与其相关的名字。例如：将端口号 80 显示为 http。当你碰到一个不常见的端口号而你又不知道这代表着什么协议的时候，这个功能就非常有用了。

3）**解析网络名称（resolve network/IP name）：**这种类型的名称解析试图将第三层的 IP 地址转换成为更容易记忆的域名，如果这个域名具有高描述性的话，这种转换对我们理解系统的目的或者所有者，将是非常有帮助的哟。

4）**Use captured DNS packet data for address resolve：**从已经捕获的 DNS 数据包中解析出 IP 地址和域名的映射关系。

5）**Use an external network name resolve：**允许 Wireshark 使用你当前设备配置的 DNS 服务器来解析捕获数据包中的 IP 地址对应的名称。这个功能在捕获的数据包中没有 DNS 数据，但是仍然需要名称帮助分析的时候，非常有用。

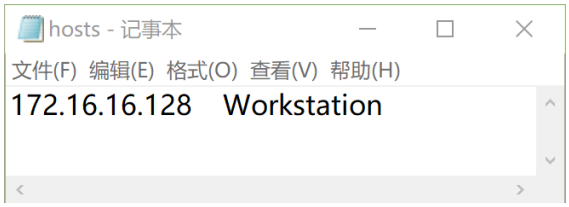
2．使用自定义 hosts 文件

在一个大型捕获文件中，不断从多个主机之间跟踪流量是一件非常乏味的事情，特别是当外部主机解析服务不能访问的时候，这个时候我们可以根据他们的 IP 地址并且通过一个叫做 Wireshark hosts 的文件来手工标识系统。

Wireshark hosts 文件的实质是由 IP 地址列表和与之对应的名字组成的文本文件，为了快速查询，可以在 Wireshark 中使用 hosts 文件来标记地址，这些名字会显示在数据包列表面板里面，其具体的使用步骤如下：

1）在 Wireshark 的***首选项***窗口中，选择 Only use the profile “hosts” file。

2）使用 windows 记事本或者类似的文件编辑器创建一个新文件。该文件应该至少包含一条 IP 和对应名称的记录，Wireshark 会根据这个映射来把相应的 IP 地址替换为 hosts 文件里对应的名称并最终显示到数据包列表面板里。



3）把文件以文本格式命名为 hosts，并保存到<USERPROFILE>\Application Data\Wireshark\hosts 目录下。

【验证实验 3】

用 Wireshark 打开捕获文件 ***lotsofweb.pcapng***，通过设置名字解析的相关选项和参数，使

得数据包列表面板中的 source 或者 destination 列中出现与 IP 地址对应的名字，并提供截图。

3. 扩展阅读——名称解析的潜在弊端

名称解析有很多的优点，使用名称解析看上去很容易，但是也存在一些潜在的弊端。

1）名称解析可能失败，尤其是当没有可用的 DNS 服务器时；

2）名称解析会带来额外的开销；

3）名称解析的数据包可能占据你所捕获的数据包中的一部分，影响到你对捕获数据的分析，甚至如果你分析的捕获数据中包含恶意地址的话，对恶意地址的解析会暴露你的行为甚至是你自己成为攻击着的靶子。

1.2.3 任务 3 协议解析器

Wireshark 最大的优势在于对上千种协议解析的支持，它其中的协议解析器允许你将数据包拆分成多个协议区段以方便分析。

1. 更换协议解析器

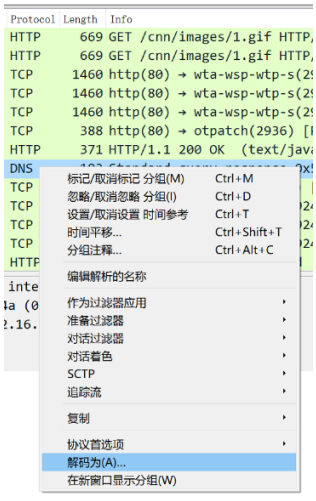
Wireshark 使用协议解析器自动的识别每个协议并且决定该如何显示网络数据包信息，但是 Wireshark 在给一个数据包选择解析器的时候也不是每次都能够选择正确，尤其是当网络上的一个协议使用了不同于标准的配置的时候，比如非缺省端口。

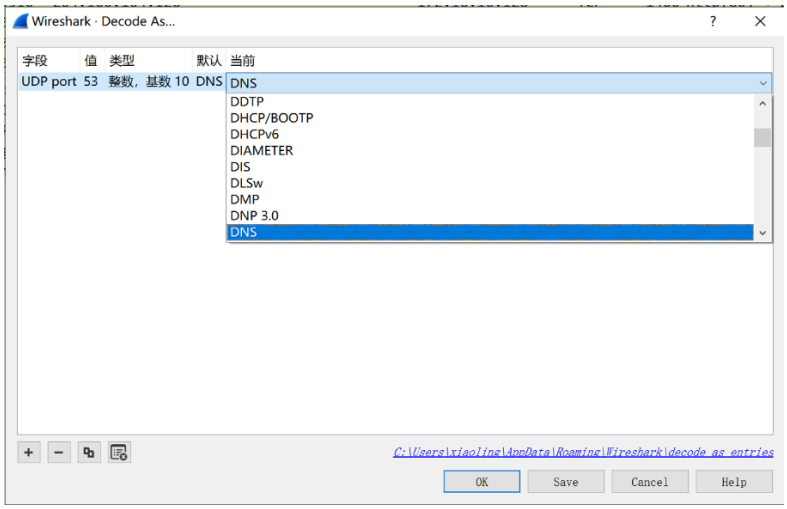
当 Wireshark 错误的选择了协议解析器的时候，我们就必须进行认为的干预了。

1）在 Wireshark 数据包面板中的协议列右键单击选中一个数据包，选择“解码为…”；

2）在“解码为…”窗口的“当前”下拉框中选择想要使用的解码器；

3）当选择好了之后，就可以立即将修改应用到捕获文件中去了。





【验证实验 4】

使用 Wireshark 打开捕获文件 ***wrongdissector.pcapng***，对数据包进行分析，并回答下列问题：

1）Wireshark 是否选择了错误的协议解析器，为什么？请提供截图证据；

2）你认为正确的协议解析器应该是什么？为什么？

3）请为 Wireshark 的更换正确的协议解析器，并提供截图证据。

4）你认为本案例中 Wireshark 错误选择协议解析器的原因时什么？

2. **扩展阅读——是否保存更换的协议解析器**

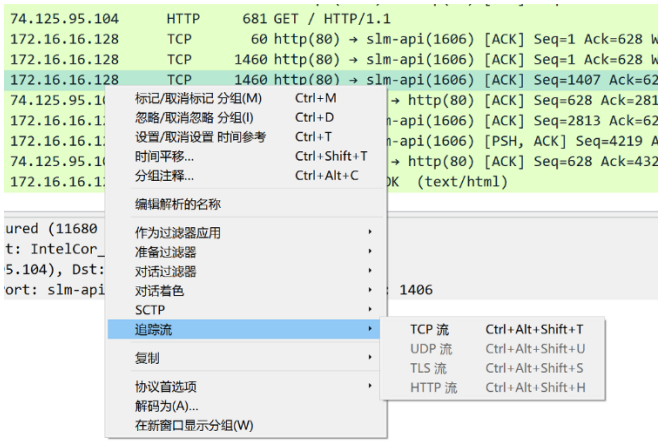
在默认情况下，当你关闭捕获文件时，强制解码的设置不会保存，补救的方法就是在强制解码对话框中点击“保存”按钮，这会将协议解码的规则保存到你的 Wireshark 用户配置文件中，当你使用 Wireshark 打开任何的捕获文件，该解码规则都会生效。

个人建议不这样做。因为将保存过的强制解码规则忘在脑后是一件非常容易的事情，将会造成很多的混乱，因此我们要特别留意保存过的强制解码规则，避免我们自己掉进自己挖的大坑中哟。

1.2.4 任务 4 流追踪

Wireshark 分析功能中令人满意的一点就是他能够将来自不同包的数据重组成统一易读的格式。流跟踪功能看可以把从客户端发往服务器的数据都排列好顺序使其变得更容易查看，这样就不需要从一堆小块数据里面一个包一个包的跟踪了。

目前 Wireshark 可以跟踪 4 种类型的流，他们是 TCP 流、UDP 流、SSL 流和 HTTP 流。在 Wireshark 的数据包面板中右键单击任意一个相关的数据包，选择“追踪流”即可。



【验证实验 5】

使用 Wireshark 打开捕获文件 ***http_google.pcapng***，对数据包进行分析，完成下面的操作：

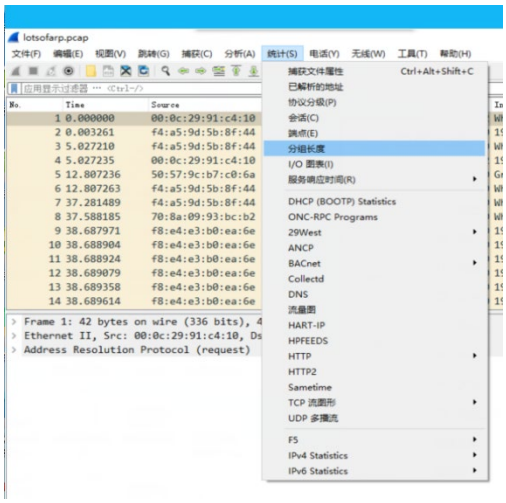
1）在 Wireshark 的数据包列表面板中找到第一个 http 请求数据包；

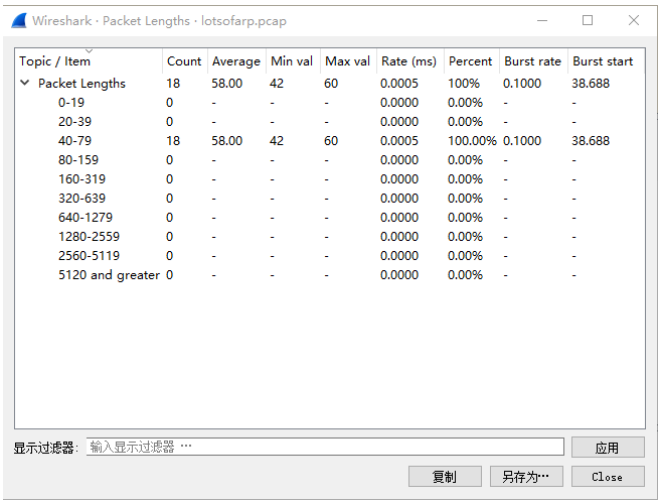
2）跟踪该 http 流，找到第一个 http 请求数据包的响应数据包；

3）该响应数据包响应了一个什么样的内容呢，从 http 的响应数据包提取服务器响应的对象，并使用浏览器查看之？并请提交截图。

1.2.5 任务 5 数据包长度统计

一般情况下，一个以太网帧的最大长度是 1518 字节，除去封装在数据首部的控制信息，还剩下 1460 字节供应用层协议使用。如果你知道报文传输的最小需求，那么我们就可以通过捕获文件中数据包长度的分布情况，做一些对流量的合理猜测。





Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Packet Lengths	18	58.00	42	60	0.0005	100%	0.1000	38.688
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	18	58.00	42	60	0.0005	100.00%	0.1000	38.688
80-159	0	-	-	-	0.0000	0.00%	-	-
160-319	0	-	-	-	0.0000	0.00%	-	-
320-639	0	-	-	-	0.0000	0.00%	-	-
640-1279	0	-	-	-	0.0000	0.00%	-	-
1280-2559	0	-	-	-	0.0000	0.00%	-	-
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

特别注意哪些大小为 1280-2559 字节的数据包的统计，这些较大的数据包往往以为这数据传输，而较小的数据包则意味着协议控制序列。

查看数据包长度的统计情况是对网络流量状况进行了解的一个好方法。如果存在很多较大的数据包，那么很可能网络中在进行的大量的数据传输；如果绝大多数数据包都很小，我们便可以假设网络中此时存在协议控制命令，但是没有大规模的数据传输。虽然这不是一个必要的操作，但是在深入分析网络流量之前能够做一些类似的假设，还是非常有帮助的。

【验证实验 6】

使用 Wireshark 打开捕获文件 download_slow.pcapng，对其数据包长度进行统计分析。这些网络流量中是否存在大规模的数据传输呢？请说明你的理由并提供证据截图。

1.2.6 任务 6 丢失的页面

【综合分析实验 7】

在这个场景中，我们的用户是 Peter，一位从不熬夜导致经常错过球赛的大学生球迷。每天上午，他都要打开计算机访问网站，查看前一天晚上的最终比赛成绩。

这个上午，当 Peter 和 往常一样打开网站 ESPN 查看比赛成绩的时候，他发现加载网页耗费了很长的时间，而当加载过程终于完成时，页面丢失了大部分图片和内容。这是怎么回事儿呢？我们一块儿来帮 Peter 分析下。

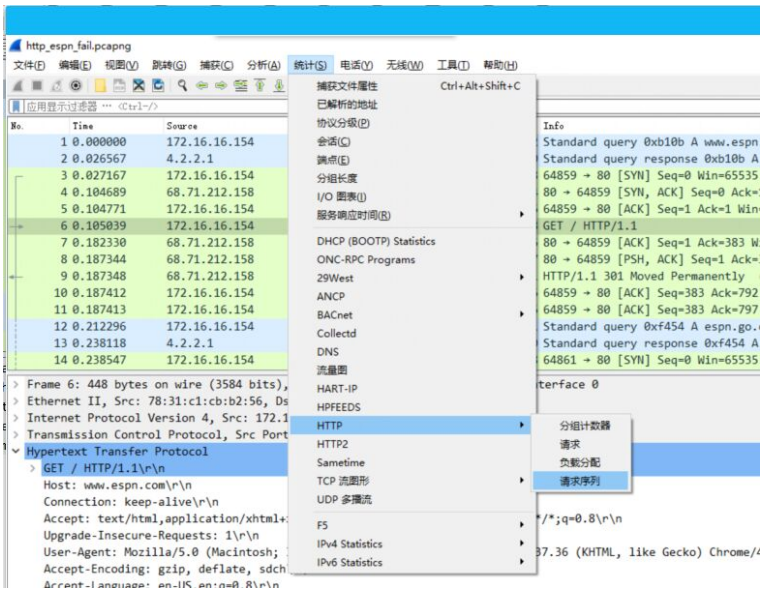
首先需要明确的是，这个问题只在 Peter 一个人的计算机上发生，并没有影响到其他人，因此我们在他的计算机上进行数据包嗅探工作，获取了 ***http_espn_fail.pcapng*** 捕获文件。

从何入手呢？

HTTP。

【分析过程】：

1）首先，我们需要从大量的网络数据包中找到 http 请求数据包。我们有两种方法从捕获文件中过滤出这些数据包。一是通过设置过滤器筛选 http GET 请求数据包；二是使用通过使用 Wireshark 统计功能中的 HTTP 请求序列功能。（实验 7-1：请提供两种方法过滤 http 请求数据包的截图）



2）从过滤出的 HTTP 请求可以看出，捕获文件中的网络流量一共进行了 7 次不同的 HTTP 请求，对这些 HTTP 请求的目标主机的名称的分析，每个 HTTP 请求目标的名字部分都包含了 espn 字符串，这样就导致我们没有明确的流追踪的目标，怎么办？

No.	Time	Source	Destination	Protocol	Length	Info
6	0.105039	172.16.16.154	68.71.212.158	HTTP	448	GET / HTTP/1.1
17	0.316340	172.16.16.154	199.181.133.61	HTTP	447	GET / HTTP/1.1
71	0.554110	172.16.16.154	72.21.91.8	HTTP	398	GET /js/310987714.js HTTP/1.1
74	0.565877	172.16.16.154	72.246.56.35	HTTP	511	GET /combiner/i?img=%2Fphoto%2F2
202	0.616633	172.16.16.154	69.31.75.194	HTTP	491	GET /i/columnists/kapadia_sheil_
387	0.700630	172.16.16.154	72.246.56.35	HTTP	490	GET /combiner/i?img=%2Fphoto%2F2
390	0.702291	172.16.16.154	72.246.56.83	HTTP	594	GET /combiner/i?img=%2Fmedia%2Fm

Topic / Item
- HTTP Request Sequences
  - http://www.espn.com/
    - http://espn.go.com/
      - http://cdn.optimizely.com/js/310987714.js
      - http://assets.espn.go.com/i/columnists/kapadia_sheil_m.jpg
      - http://a4.espncdn.com/combiner/i?img=%2Fphoto%2F2014%2F0806%2Fnfl_g_colts5_cr_1296x729.jpg&w=556&cquality
      - http://a2.espncdn.com/combiner/i?img=%2Fmedia%2Fmotion%2F2016%2F0108%2Fdm_160108_Trainer_should_get_the
      - http://a1.espncdn.com/combiner/i?img=%2Fphoto%2F2016%2F0108%2Fsubzero_5x2.png&w=1296&h=518&scale=cro

3）在没有明确的目标会话的情况下，进行协议层次统计分析。这将帮助我们定位非预期的协议和通信过程中各种协议分布的异常。请注意，协议分层统计是依据当前筛选器的条件设置进行的，如果希望对整个捕获文件进行分析，则需要清除之前的筛选过滤器的设置。（实验 7-2：请提供对整个报文的协议分层统计分析的截图）

4）协议分层视图并不复杂，我们很快就能识别出网络会话只涉及两个应用层协议：HTTP 和 DNS。我们在前面的学习中已经知道 DNS 是用于将域名转换为 IP 地址的一个应用。当访问一个网站，例如：ESPN 时，如果你的系统中没有缓存这个名称对应的 DNS 记

录时，系统将会向远程的 DNS 服务器请求来查询名字对应的 IP 地址，当 DNS 服务器返回一个可用的 IP 地址时，则该域名解析信息将被缓存到本地。此时，HTTP 通信就可以开始了。

5）所以，从步骤 4 来看，在进行 HTTP 会话之前，存在 DNS 会话也是一种正常的现象。但是仔细分析我们发现，一个名字的 DNS 查询请求通常只包含一个数据包，相应的查询的回应数据包也只有一个，但是捕获文件中总共有 14 个 DNS 数据包，也就是说存在 7 次 DNS 查询。Peter 总共在浏览器中输入了一个 URL，为什么产生 7 次 DNS 查询呢？

⌄ User Datagram Protocol	2.5	14	0.0	112	9	0	0	0
Domain Name System	2.5	14	0.3	1039	87	14	1039	87

在理想状况下，浏览一个网页只需要查询一个服务器地址，然后在一次 HTTP 会话中就可以获取所有内容。但在实际情况中，一个网页可能提供多个服务器上的内容。可能的场景是，全部基于文字的内容在一处，图像内容在一处，而内嵌的视频在另外一处。当 HTTP 客户端解析了 HTML 代码后，发现了对其他服务器上资源的引用，为了获取引用的资源，客户端会尝试查询相关的服务器，这就导致了额外的 DNS 查询和 HTTP 请求。从这个角度来理解，Peter 在访问 ESPN 网站时，网站的代码引用其他源的内容，所以他的浏览器自动的其他多个域名请求了内容。

6）一个一个的“惊天内幕”被我们揪出来，有一个一个被我们自己否决了，怎么办？没有办法，我们只好老老实实的一个一个的分析与前面的 7 个 HTTP 请求相关的会话了。（实验 7-3：请使用 Wireshark 统计会话功能列出本案例捕获数据包中的所有会话，并提交截图）

7）等等，似乎又有新的发现呢。7 个 HTTP 请求应该对应着 7 个网络会话，但是对捕获数据文件的会话统计有 8 个会话存在，有什么猫腻？

Ethernet · 1 | IPv4 · 8 | IPv6 | TCP · 16 | UDP · 7

Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
4.2.2.1	172.16.16.154	14	1627	7	1106	7	521	0.000000	0.6639	13 k	
68.71.212.158	172.16.16.154	13	2032	6	1200	7	832	0.027167	90.8752	105	
69.31.75.194	172.16.16.154	19	9949	10	8942	9	1007	0.579477	90.6593	789	
72.21.91.8	172.16.16.154	92	70 k	49	67 k	43	3170	0.526867	60.5532	8863	
72.246.56.35	172.16.16.154	247	196 k	134	188 k	113	8315	0.527902	90.8063	16 k	
72.246.56.83	172.16.16.154	30	20 k	15	19 k	15	1518	0.659868	45.3449	3384	
172.16.16.154	199.181.133.61	61	49 k	24	1953	37	47 k	0.238547	91.0836	171	
172.16.16.154	203.0.113.94	93	6774	93	6774	0	0	0.430071	94.5936	572	

结合我们对 HTTP 请求中目标地址的分析，有一个会话进入了我们的视野，这个会话存在几个让人觉得异常的地方：

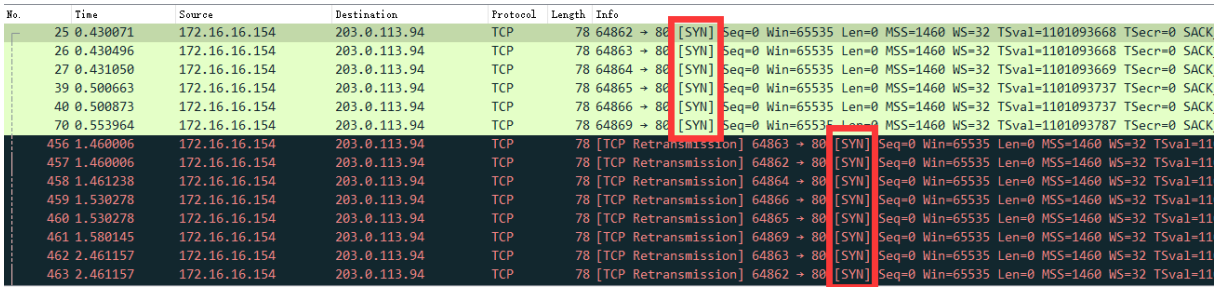
第一：A->B 方向有 93 个，总共 6774 字节数据包，但是 B->A 方向一个数据包都没有；

第二：其他会话的地址都存在相关的名字到 IP 地址转换的 DNS 查询数据包与之对应，而这个会话中地址 B 没有 DNS 查询与之对应。

第三：其他的 7 个会话都有对应 HTTP 请求，但是这个会话没有相关的 HTTP 请求。

综合以上情况，我们似乎可以有这样一个印象：Peter 的计算机不知道什么原因向一个未知的服务器（203.0.113.94）发送了 6774 字节的数据，但是对方一个字节的数据都没有回应。

8）OK，有了目标就好办。我们以这个筛选并查看这个会话。在会话窗口右键单击异常会话，将其设置为过滤器。（实验 7-4：请在 Wireshark 数据包列表面板中展示异常会话，并提交截图）



No.	Time	Source	Destination	Protocol	Length	Info
25	0.430071	172.16.16.154	203.0.113.94	TCP	78	64862 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1101093668 TSecr=0 SACK
26	0.430496	172.16.16.154	203.0.113.94	TCP	78	64863 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1101093668 TSecr=0 SACK
27	0.431050	172.16.16.154	203.0.113.94	TCP	78	64864 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1101093669 TSecr=0 SACK
39	0.500663	172.16.16.154	203.0.113.94	TCP	78	64865 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1101093737 TSecr=0 SACK
40	0.500873	172.16.16.154	203.0.113.94	TCP	78	64866 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1101093737 TSecr=0 SACK
70	0.553964	172.16.16.154	203.0.113.94	TCP	78	64869 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=1101093787 TSecr=0 SACK
456	1.460006	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64863 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=11
457	1.460006	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64862 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=11
458	1.461238	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64864 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=11
459	1.530278	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64866 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=11
460	1.530278	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64865 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=11
461	1.580145	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64869 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=11
462	2.461157	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64863 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=11
463	2.461157	172.16.16.154	203.0.113.94	TCP	78	[TCP Retransmission] 64862 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=11

通过分析异常会话，我们发现全部是 TCP SYN 数据包。HTTP 需要建立在 TCP 链接之上，TCP 建立链接需要有 SYN、SYN ACK、ACK 三个类型的数据包一次交互才能成功。通常情况下，当一台主机向对方发送 TCP SYN 数据包之后，对方应该回应 TCP SYN ACK 数据包，信息交互才能正常进行下去。而从会话分析，这里只有 Peter 的计算机向未知主机发送的 SYN 数据包，而对方始终未回应一个字节的数据。从 Time 列可以看出，这个过程持续了 95s 之长，从这一点看，我们似乎找到了 Peter 的页面打开缓慢最终丢失信息的直接原因了。

9）根本原因没有找到，我们仍然无法解决 Peter 网页丢失的问题。综合我们前面分析的结果，有下面几点：

第一：Peter 的计算机向一个远程计算机请求建立 TCP 链接，对方一直未予回应，是导致 Peter 页面丢失的直接原因；

第二：远程计算机的 IP 地址未在捕获数据包中出现过，也未发现与之关联的 DNS 查询数据包，这个 IP 地址从何而来呢？

突然有一个词从我的脑海中冒出来——**DNS 缓存**。

在需要查询名称对应的 IP 地址时，系统会首先查询 DNS 缓存中是否有相应的缓存记录，如果有，怎直接从 DNS 缓存中读取名称对应的 IP 地址，而不需要向网络发送 DNS 查询数据包。问题分析到这里，Peter 碰到的情况可能是之前使用过某个名称，并对该名称的 IP 进行过 DNS 查询并且该映射关系被系统缓存了，因此这次再次使用该名称时系统直接从 DNS 缓存中使用了这个映射关系。非常不幸的是，这个地址所标志的那个主机却由于某种不知道的原因更换了地址于名称的映射关系或者停止了服务，此时 Peter 的设备就去尝试连接一个无效的地址了，作为结果就是请求超时，内容加载失败。

【解决方案】

1、手动删除 DNS 缓存；

2、等待一段时间，让其 DNS 缓存过期失效。

2　小结：学习心得与体会

学生自己总结本次实验的内容，心得体会，意见和建议。

参考文献：

这部分要求学生把查阅的资料整理出来，并附上 pdf 归档包，作为积累的内容。