|  |  |  |  |
| --- | --- | --- | --- |
| **《计算机网络》实验报告** | | | |
| **实验编号** | 实验1 | **实验名称** | Protocol Layers |
| **姓名** | 李彦浩 | **学号** | 202100300063 |
| **班级** | 工业软件班 | **成绩** | （空着） |
| 1. **实验目的**   学习各协议层在包中的表现形式。协议层是计算机网络架构中重要的基本概念。 | | | |
| 1. **实验要求**   wireshark：使用wireshark软件工具用于捕获和检查数据包跟踪。  wget: 使用wget(windows下)来抓取web资源。 | | | |
| 1. **实验内容** 2. Capture a Trace. 3. Inspect the Trace 4. Packet Structure 5. Protocol Overhead 6. Demultiplexing Keys 7. Explore on your own | | | |
| 1. **实验过程**  Step 1:Capture a Trace（链路追踪） wget方面: Fetch一个网页  wget -P 'path/to/directory' <http://www.xxx.com>  # wget -P path target\_website  -P选项：将fetch到的文件放到后续指定路径中(path/to/directory)  1  wireshark方面: 链路追踪与分析  使用过滤器: tcp port 80  仅记录标准网络流量(standard web traffic)，无其他任何包  2  开启IP解析: Resolve network names  IP解析域名  开启捕获，然后用上述wget的命令fetch一个网页。3  开启捕获，然后用上述wget的命令fetch一个网页。  4  5  我抓的百度的包。 Step 2: Inspect the Trace(分析链路) 把每一层的信息叫做包(packet)，这只是一个泛指。  实际上：   1. 在物理层：比特流(bit stream) 2. 在链路层：帧(frame) 3. 在网络层：包(packet) 4. 在传输层：段(segment) 5. 在应用层：消息(message)   方便起见，统一用包(packet)来进行指代。  找到一个协议为HTTP且请求方式为GET的包。它携带了从我的电脑到目标服务器的web(HTTP)请求  6  No.18 HTTP GET请求  NO.22 响应 200表示成功  7  应用层协议:HTTP  网络层协议:TCP/IP  链路层协议:以太网  抓包结果解析：  (No.18 HTTP GET)   1. 第一个块：帧(Frame)本身。不是协议部分，而是记录了整个包的信息，包括抓到的内容以及长度。   8  第二个块：以太网(Ethernet)。当我们要求wireshark以以太网协议抓取网络流量，他就会将802.11（无线网络）的帧头替换为伪以太网帧头（pseudo-Ethernet)  9  第三个块：IP/TCP/HTTP。（顺序从协议栈底部向上走，因为实际的数据包从上往下走，而更低层协议的帧头信息会被添加到高层协议的帧头信息的前面，所以顺序就是IP/TCP/HTTP）  10  （No.22 200)  多出来两个块。   1. ${X} Reassembled TCP Segments: 一般来说服务器的响应是将数据拆分成多个包，然后在抵达请求方后重新拼接起来的。这里面的X对于每次请求都不一样，他代表着每次为了得到完整的响应需要拼接在一起的包的数量。   11  (注意看右边16进制转换过来的文本）   1. Line-based text data: 每次请求不同。代表了抓取到的网页的内容。这里是text/html。但也有可能会是别的东西，比如text/xml , image/jpeg等。  Step 3: Packet Structure (包结构) 1.Ehternet块  12   1. IPv4块   13  根据截图中的信息，包结构如下：   |  |  |  | | --- | --- | --- | | Ethernet Destination 0-5 | Ehternet Source 6-11 | Ethernet Type 12-13 | | IP Version 14高四位 | IP Header Length 14低四位 | IP Differentiated Services Field 15 | | IP Total Length 16 17 | IP Identification 18 19 | IP Don’t fragment 20高三位 | | IP Fragment Offset 20低五位 21 | IP Time to Live 22 | IP Protocol 23 | | IP Header Checksum 24 25 | IP Source Address 26 -29 | IP Destination Address 30-33 |  Step 4: Protocol Overhead (协议开销) 计算方式：   1. 将HTTP数据作为网络携带的数据 2. 将低层协议的帧头作为开销   计算区间：  仅计算单次web抓包中下载方向的包（按照Destination排序）   1. 以一个短TCP包开始，这个TCP包的FLAG是[SYN ACK] 2. 中间是下载的主要部分，以一个HTTP包结束 3. 最终以一个TCP包结尾，来终止连接   对应到WireShark中即：  14  逐帧分析：  第一帧  14_1  仅有protocol overhead。共66字节。  第二帧  14_2  仅有protocol overhead。共56字节。  第三帧  14_3  TCP payload字段有628字节。protocol overhead 54字节。  第四帧  14_3  TCP payload 1452字节。protocol overhead54字节。  第五帧。  14_3  TCP payload 84字节。protocol overhead 54字节。  第六帧。  14_3  TCP payload 333字节。protocol overhead 54字节。  第七帧。  14_3  protocol overhead 56字节。  所以protocol overload = 66+56+54+54+54+54+56 = 394字节，总共394+628+1452+84+333字节等于2891字节。开销率 = 394/2891 = 13.63%  [我wget的baidu.com](http://xn--wgetbaidu-k57rv31p.com)，这个开销率还是比较大的。 Step 5: Demultiplexing Keys (多路分解键) 什么是demultiplexing?  当一个帧通过以太网到达目标计算机后，以太网层将递交该帧到更高层进行加工。而找寻应该递交到的最高层的过程叫做demultiplexing。  虽然我们知道更高一层应该是IP协议层，但以太网层怎么知道的？  同样的问题，IP怎么知道它的更高层是HTTP的？  答案是，协议利用帧头中的demultiplexing key来确定更高层是哪一层。  例如，在Ethernet层中，有一个字段，叫做Type，里面的内容为0x0800，对应位IPv4。  19  而在IP层中，有一个字段，叫做Protocol，内容为0x06，对应TCP。  20 Explore on your ownQ1: 查看一个不带高层数据的短的TCP包，它的目标点是哪个实体？如果它不携带数据，它是不是没啥用？ 找了两帧。  21  看一下各自TCP头部段的数据。  22  前者源端口61423，目标端口80。而源主机是10.27.255.99（这是我们校园网分配的静态IP），目标主机是39.156.66.10。这是百度的服务器节点。  23  而后者就反过来，源端口80，目标端口61423，源主机39.155.66.10，目标主机10.27.255.99。  注意到HTTP GET之前这样的TCP包有三个，于是上网查阅资料，得知这就是TCP三次握手的过程。第一次是从校园网到百度服务器的，申请连接；第二次百度服务器向校园网IP，声明可以连接。随后第三次，建立连接。  也就是说这些不带有数据包的TCP包，本质上就是为了建立连接存在的，是源主机和目标主机之间传输数据的前置条件。看似它们没有带有数据的HTTP包有用。但其实这一环是不可缺失的。  关于TCP协议格式与三次握手过程，下文再说。 Q2: 在一个经典分层模型中，从更高层的消息前会有一个更低层附带的头部段。比如从高到底的过程是HTTP→TCP→IP→ETHERNET,则一个消息的头部段格式应该为：  |  |  |  |  | | --- | --- | --- | --- | | ETHERNET HEADER | IP HEADER | TCP HEADER | HTTP HEADER |  但有时候会有不同。在之前做链路追踪时，一个web响应（一个HTTP响应消息包含了一个HTTP头部段和HTTP的有效负载(payload))被转换成多个更低层的消息（分成了多个TCP包）。这些TCP包的结构应该是怎样的？ 前置知识是TCP的协议格式。  24  从网上查到了一张TCP协议的格式图。   * **16位源端口号**：数据发送方的端口号，表示数据从哪里来 * **16位目标端口号**：数据接收方的端口号，表示数据要到哪里去 * **32位序号**：每一次通信TCP报文的编号 * **32确认序号**：用于对发送方发送的报文的确认，为接收到的报文序号+1 * **4位首部长度**：表示TCP报文头部有多少个4字节，因为4位最大表示15，所以TCP报文头最大长度为15\*4=60 * **6位标志位**：   + **URG**: 紧急指针是否有效   + **ACK**: 确认应答   + **PSH**: 提示接收端应用程序立刻从TCP缓冲区把数据读走   + **RST**: 表示要求对方重新建立连接   + **SYN**: 请求建立连接，我们把携带SYN标识的称为同步报文段   + **FIN**: 通知对方本端要关闭了，我们把携带FIN标识的为结束报文段 * **16位窗口大小**：TCP流量控制的一个手段，这里指接收窗口，用于告知发送端本端的TCP缓冲区还能容纳多少字节的数据，这样发送方就可以控制发送的数据量 * **16位校验和**：由发送方填充，接收端对TCP报文段执行CRC校验以检查数据是否损坏，这个校验不仅包含TCP头部，也包含数据部分 * **16位紧急指针**：一个正的偏移量，它与序号字段相加表示最后一个紧急数据的下一个序号，所以这个字段是紧急指针相对当前序号的偏移，用于发送方向接收方发送紧急数据 * **0-40字节选项数据**：存储一些可能需要的额外信息 * **数据**：有效负载（payload)  Q2.1 三次握手 看一下三次握手部分的TCP包。  25  首先可以知道的是这些包是没有payload的，所以这个就不用管了。  而源端口、目标端口这些是有实际含义的，跟控制字段没关系。  所以我们这里想要关注的部分就是seq、ack和标志位（校验和是为了检错纠错的，也不关注）  三次握手过程：   1. SYN=1，表示客户端请求建立连接。Seq=0。发送帧的相对序号是0（绝对序号不是这个，相对序号只是为了标识是否和上一帧相同，滚动数组的思想）   26   1. SYN=1，表示服务器请求建立连接。ACK=1，表示服务器确认了在第一步中客户端请求建立连接的帧。可以看到的是，在第一步中帧的绝对序号是2106528215，在第二帧的ack的绝对序号中就是这个序号+1。   27   1. ACK=1，表示客户端确认了服务端发来的请求建立连接的帧并应答。至此双方均可以建立连接，于是客户端与服务器进入ESTABLISHED状态，开始传输数据。可以看到第三步确认的序号就是第二步中帧的序号+1。   28  附一个三次握手的示意图。  29  附加的小问题：为什么三次握手不是两次或更少？不是四次或更多？   1. 不是两次或更少  * 如果客户端发往服务端的**SYN消息滞留在网络**中，客户端发起**重传SYN**并建立连接，如果等到**连接关闭**时，此时之前滞留在网络中的SYN消息到达服务端，服务端回复，就默认**建立连接**，但是此时客户端已经断开连接了，造成**连接资源的浪费** * 如果只有两次，发送端发往服务端的SYN消息，服务端**收到SYN消息**，如果服务端**回复的消息丢失**的话，服务端认为建立连接了，但其实客户端并没有建立连接  1. 不是四次或更多   三次握手足够确认双方的**发送和接收能力**，再多的消息就多余了  如果转化成大白话的话，我觉得意思就是，客户端申请连接，服务器端是同意连接的，但这件事情客户端不知道，所以你不能在我不知道你同意建立连接的前提下直接就跟我建立连接了，你得先告诉我你是同意的，然后咱俩再建立连接。 Q2.2 四次挥手 相对于三次握手建立连接。四次挥手是TCP关闭连接的过程。  抓包结果如下：  30  这里虽然看似是3次挥手，但本质上是4次。后面会说为什么。  先来看四次挥手的流程：  31  状态释义：   * FIN-WAIT-1：表示想主动关闭连接。向对方发送FIN报文后会进入到FIN-WAIT-1状态。 * CLOSE-WAIT：表示在等待关闭。当对方发送FIN给自己，自己会回应一个ACK报文，此时进入CLOSE—WAIT状态。在此状态下，是需要考虑自己还有没有数据要发给对方，如果没有就发送FIN报文给对方。 * FIN-WAIT-2：接收到了对方的ACK确认后就会进入该状态，并等待对方发送FIN报文。如果接收到了对方同时带FIN，和ACK的报文，就可以直接进入到了TIME-WAIT状态，而无需经过FIN-WAIT-2状态 * LAST-ACK：被动关闭方发送FIN报文后，等待对方的ACK报文，当收到对方的ack报文后进入到close状态。 * TIME-WAIT：表示主动方收到了对方的FIN报文，并发送了ACK报文，在等待2MSL后即可进入到CLOSED状态了。 MSL：（Maximum Segment Lifetime，最大分段生存期），是TCP报文在internet上的最长存活时间，每个TCP实现都需要一个具体的MSL，RFC 1122建议是2分钟。所以2MSL就是4分钟。 * CLOSED：关闭状态   一般来说四次挥手应该是这样的：  32   1. FIN,ACK=1（ACK确认的对方上一帧，主要是FIN代表了客户端想要和服务器断开连接），seq=1,ack=1(期望对方下一帧序号为1） 2. ACK=1。服务器收到了客户端断开连接的请求，但是可能这个时候数据还没有传完，所以不会立刻关闭连接的。而是选择确认这个请求，等到传输结束后再来处理。因为上一次ack=1，所以这次seq=1。因为上一次seq=1，所以这一次ack=2。 3. FIN,ACK=1。服务器传输结束，申请和客户端断开连接。 4. ACK=1。客户端回复收到（断开连接），并断开连接。服务器接到ACK后断开连接。   那么为什么我抓到的包只有三次挥手呢？  33  可以看到，第一次是客户端到服务器的。但服务器回发的并不是ACK，而是FIN,ACK。这其实是客户端和服务器同时想要断开连接，也就是说，客户端想要断开连接时，服务器恰好传完了数据，也想断开连接。因此这两条FIN,ACK看似是先后的，其实是同时的。也就是说他把原来第二次和第三次挥手合并成了第二次挥手，因此后续客户端只要回复得知断开连接的ACK即可。 Q3：在一般分层模型中，低层将把协议信息附在高层信息的头部，如果底层选择加密，那么模型将变成怎样的？ 加密的类型、加密密钥不能与加密数据同时发送。应当在传输数据前进行收发双方协商确定好。否则，如果有人截取了这段流量，它可以直接拿到密钥和密文，直接本地解密。 Q4：继Q3，低层协议加入了压缩后的协议信息，模型将变成怎样的？ 压缩目的在于节省传输流量，最大化利用。压缩的协议可以在下层协议的header中表现出来，而不是像加密一样提前商定。这样在header中指出便于接收方解压缩，同时也能节约传输流量。所以在传输包时应该将压缩数据、压缩编码方式一并放入header。 | | | |
| **四．实验心得**  本次是计算机网络8个Lab中的首个Lab。首先先安装了wireshark和wget的环境。wireshark提供了一个图形化用户界面，但wget只有命令行界面。这是因为他们的作用是不同的，wireshark用于捕获并跟踪数据包，因此需要有一个界面来显示捕获到的数据的各个字段与实际含义。而wget主要用来抓取网络资源，这个过程是黑盒的，不需要贴任何中间变量出来，因此做成了命令行窗口。  了解并熟悉了这两个软件的操作后，我就开始按照实验指导书中提供的步骤，利用wget获取网络资源（比如www.baidu.com下的html网页文件），然后利用wireshark在fetch的过程中捕获80端口（tcp 80端口为HTTP超文本传输协议开放的端口，主要用于万维网www.传输信息的协议）相关的数据包。这个过程本身还是比较简单的。但是重难点在于分析这些数据包中的协议层字段与控制交互的逻辑。  实验指导书首先向我们提供了Protocol Layers的前置知识，即各协议的层次结构。比如运行在应用层的HTTP协议，运行在传输层的TCP协议，运行在网络层的IP协议以及运行在链路层的Ethernet协议等。并且每一层的数据包传到下一层之后，更低层会在该数据包的头部附上本层的协议字段。这样一来每个帧中都是Ethernet->IP->TCP->HTTP这样逆协议栈的分层结构。  在大致了解了整个协议层总体结构后，我根据自己的理解，尝试解答了实验指导书中Step的相关问题。这个问题是关于数据包协议字段的结构的。这就需要深入地去分析我们抓取到的数据包中的各个字段。其中有很多字段的实际含义我并不是很清楚，不过实验指导书告诉我这些内容将在以后的Lab中涉及，我现在需要做的就是搞清协议字段的总体结构。根据真个指导思想，我成功利用wireshark中对帧的具体分析的图形化界面，理清了各协议层字段在帧中的所在位置与包含的字段。这个问题的解答对于后续TCP三次握手与四次挥手的概念有着重要作用。  之后来到第四步，这一步要求我们做一些协议开销的计算。实验指导书给出了开销的定义：即各帧中用于协议信息的部分，即为协议的开销。与之相对的就是有效负载，也即帧中数据的部分。我通过分析、归类和计算，得到了我抓取www.baidu.com下的网络文件资源时的开销大致为13.63%左右，这是一个不小的开销。但后续我又做了几组对比，如果我抓取的资源比较大，比如上千字节，那么这个协议开销的占比也会相应降低。也就是说，协议开销率根据抓取资源的不同会有所改变。  随后第五步中提到了一个很重要的概念，即多路分解键(Demultiplexing Keys)。这一步中实验指导书提出了一个很有趣的问题：我们知道协议栈的层级结构，但计算机是怎么知道的呢？比如，Ethernet层怎么知道它应该把接收到的数据包解封装之后发给IP层呢？这就涉及到了多路分解键的概念。每一层中都有一个字段标识了更高层协议，比如，Ethernet层有一个字段叫做IP Version。内容为0x04。也就是这个数据包在链路层协议之上的更高层协议为网络层下的IPv4协议。而IP协议字段中也包含了Protocol字段，内容为0x06。根据驱动表我们知道这个就是TCP协议。这样就解答了解封装路径的问题。  至此整个Lab1结束。但指导书还留给读者一些问题用于自学与发掘(Explore on your own)。第一个问题是基于上述实验内容来的。在抓取过程中，我们得到了一些完全不包含有效负载，只包含协议信息的帧。这些TCP帧的用处是什么？如果不携带数据，是否没有HTTP那么“有用”？通过观察，我发现这些TCP帧都处于HTTP帧之前或之后，之前的有3帧，之后的有4帧。我不禁想到之前听说过的一个概念：TCP三次握手与四次挥手。因此我猜测这些帧就是TCP用于创建或关闭一个连接的部分。  果然在第二题中，就问到了TCP协议的格式与结构。我在网上查阅资料，了解了TCP协议的格式，比如16字节的源端口，16字节的目标端口，32字节的序号，32字节的确认号，6个标志位等等。这6个标志位至关重要，它告知了接受方在这个数据包中有什么重要的信息。比如ACK（对方对当前方上一次发送的数据包的确认），SYN（同步数据包字段，代表申请创建连接），FIN（请求结束连接），RST（重置连接）等。我将这些字段跟TCP三次握手与四次挥手的过程做了比对，了解了各字段在创建或销毁连接时的作用。对TCP协议有更深的了解。  之后的四五问是关于协议层加密或压缩对于模型结构的影响的问题。我在网上查阅了资料，并结合了自己的理解，给出了现阶段我对这两个问题的解答。  本次Lab，让我对计算机网络协议层有更深的理解，从中学到的知识也将为之后Lab与计算机网络学习做铺垫。 | | | |