|  |  |  |  |
| --- | --- | --- | --- |
| **《计算机网络》实验报告** | | | |
| **实验编号** | 实验7 | **实验名称** | TCP |
| **姓名** | 李彦浩 | **学号** | 202100300063 |
| **班级** | 工业软件班 | **成绩** | （空着） |
| 1. **实验目的**   探究TCP(传输控制协议)的详细信息。TCP是互联网上使用的主要传输层协议。 | | | |
| 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。  Browser：寻找并抓取网页资源。 | | | |
| 1. **实验内容**   例子：   1. Capture a Trace. 2. Inspect the Trace 3. TCP Segment Structure 4. TCP Connection Setup/Teardown 5. TCP Data Transfer 6. Explore on your own | | | |
| 1. **实验过程**  Step 1:Capture a Trace(链路追踪) 许多应用使用TCP作为传输层协议，比如web浏览器。因此我们可以用一次web资源的下载来建立TCP连接。  但是，TCP其实是一个全双工协议，它可以同时在两个方向上传输数据。如果我们从web上下载资源的话，那么只能收到从远程服务器到本地计算机的数据包（在初始请求之后）。  实验指导书推荐ACM SIGCOMM会议，在里面下个PDF。  <https://conferences.sigcomm.org/>  28  用wget抓取网路资源。  wget -P C:\Users\Administrator\Desktop\junior1\computer\_network\target http://conferences.sigcomm.org/sigcomm/2022/files/posters/26.pdf  1​启动wireshark。过滤器：  tcp and host xx.xx.xx  其中xx.xx.xx是远程主机的域名。比如我抓的是SIGCOMM，那么过滤器就是  tcp and host conferences.sigcomm.org  2  3 Step 2:Inspect the Trace(追踪分析) 找一帧从远程服务器到本地计算机的下载数据包。  4  TCP帧头大致有以下字段：  Source/Destination Port:在IP地址后跟的TCP地址。一般来说标准的web server端口是80，不过我这里是443。因为wget时有这么一句：  Plain Text  Copy  Caption  URL transformed to HTTPS due to an HSTS policy  ​  即便我用的是HTTP协议，由于HSTS政策URL被转为HTTPS。而HTTPS的标准端口就是443。  Sequence Number:给出字节流中首个负载的字节地址  Acknowledgement:告知在反向字节流中的最终接收位置，说白了就是ACK字段在TCP帧头中的位置。  Header Length:给出TCP帧头的长度  Flags:2B。拆成了很多细碎的bit，每一种标识了该帧属于TCP协议中的哪种类型，比如ACK,SYN,FIN等等。  Checksum:校验和  可能还有Options字段，其中有许多选项。  TCP payload:TCP负载，传输的实际数据包。 **Step 3:TCP Segment Structure(TCP段结构)** 画出TCP段的结构，包括其中的字段长度与位置。Flags或Options字段不需要拆开。   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | Source Port 2B | Destination Port 2B | | Sequence Number 4B | | | Ackowledgement Number 4B | | | Header Length 1B | | Flags 2B | | Window 2B | Checksum 2B | | TCP Payload | |   TCP payload长度是不定的。  另外我的帧里面有类似于Urgent Pointer这种不是每个TCP帧里都有的字段，因此就不算在内。 Step 4:TCP Connection Setup/Teardown(TCP连接建立/断开)Three-Way Handshake(三次握手) 想看三次握手的过程。先找到带有SYN标识的TCP段（一般来说就在捕获的数据包的最前面），以及后面的包。或者使用过滤器：  tcp.flags.syn == 1  SYN包是三次握手的开始。它会从你的计算机发送到远程服务器上。远程服务器会响应一个带有SYN和ACK标识的包。随后你的计算机会再次响应一个ACK包。建立起连接之后就会开始传输数据。如下图：  5  画出一个TCP三次握手的示意图，在其中包含HTTP GET请求帧。把你的计算机放在图的左侧，远程服务器放在图的右侧。时间轴竖直向下。在图中包含以下内容：   1. Sequence 和 ACK序号，后者可能没有。 2. 时间戳。 3. round-trip time:在SYN和SYN-ACK包的时间戳之差，估算TCP段的往返时间。   6  根据抓包结果整理如上图。 Connection Options(连接选项) TCP的SYN数据包通过使用选项（Options）在两个终端之间协商参数。每个终端描述它的容量，然后通过在TCP帧中包含一个选项发送到另一端。 Q1:TCP的SYN包一般携带什么TCP选项？ 7   1. Maximum segment size:可被接受的最大段长 2. No-Operation(NOP):无操作 3. Window scale:窗口大小 4. SACK permitted:选择确认许可，仅用于乱序接收  FIN/RST Teardown(完成/重传断开) TCP协议在下载完成后会断开连接。一般采用FIN段(Finalize)。两个终端都会发送一个FIN帧并确认他们收到的FIN帧。如果有一方发送了RST(Reset)帧，那么连接将立即断开。RST帧不需要任何ACK。  画出TCP挥手的过程。从传输完成后的第一个FIN或RST帧开始。  同样：   1. Seq和ACK number 2. round-trip time   由于我的TCP连接是采用RST断开的，因此下图中Seq和ACK序号无法填写。  我采用过滤器搜索了一下：  tcp.flags.reset == 1  8  tcp.flags.fin == 1  9  可以看到断开连接的方式是RST。  而一般TCP四次挥手是采用FIN和ACK的。如下图：  10  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  Step 5:TCP Data Transfer(TCP数据传输) 先查看下载速率。  Statistics → IO Graph  11  调整X,Y轴：   1. X轴，间隔100ms 2. Y轴，每100ms发送的比特数   添加过滤器：  绘制两条线：一条指定源端口，一条指定目标端口。  tcp.srcport == 443  tcp.dstport == 50583  绘制出来如下图：  12  看起来不太像两条线，但却是是两条，只是重合的部分有点多。 Q1:下载方向上每秒大致的比特率是多少？每秒大致有多少包？ 13  wget告诉我们数据传输速率为293KB/s，换算一下就是29310008/1e6 = 2.344Mbps。  总共2.4s，wireshark告诉我们发了332个包。  14  因此每秒有332/2.4 ≈ 138个包。 Q2:实际内容的下载速率是多少？给出计算过程。想知道这一点，找一个比较常见的数据帧，然后计算它的负载占比是多少。 15  一个非常典型的数据帧。  16  负载占比1380/1443 ≈ 95.63%。  可以由此估算实际下载速率为0.952931000\*8/1e6 = 2.27Mbps。 Q3:由于ACK帧在上载方向上的大致数据率（包含比特率和每秒帧数）是多少？ 首先上载方向决定了源主机一定是本机。其次由于ACK帧指定了Flags中ACK那一位要是1。  过滤器：  ip.src\_host == 10.27.255.99 && tcp.flags.ack == 1  ip.src\_host填自己主机ip，我在实验室，给我分配了10.27.255.99。  由于连接建立后传输率基本是稳定的，因此我们可以看两针之间的间隔。  17  基本上稳定在了0.26s。  所以比特率大致在54\*8/0.26 = 1.66Kbps  而总共2.4s，发了32个包。  18  所以每秒32/2.4 ≈ 13.3个包  查看下载的数据包：   1. TCP段携带数据，并将ack发送回服务器。通常情况下，每两个数据包会有一个ACK。这些ack被称为延迟ack(Delayed ACK)。通过延迟一小段时间，ack的数量减半。而我抓到的延迟比2还要大，通常每3-7个包来一次ACK。   19   1. 因为是下载，所以收到的TCP段的seq将会增加。而响应（传输）的ACK也会增加。其实上图就是一个例子。长度为1434的TCP段的seq在增加，而长度为54的TCP段的ACK也在增加。 2. 因为是下载，所以响应（传输）的TCP段的seq不会增加，而收到的ACK也不会增加。亦如上图。长度为54的TCP段的seq不增加(711)，而长度为1434的TCP段的ACK也不增加(711)。 3. 每个段都携带了窗口大小来告知对方自己的缓存区里还有多少空间。这个窗口大小>0，否则连接将由于控制流失速。  Q4:如果最近从服务器收到的TCP段的seq为x，那么下一个响应传输的TCP段的ACK是多少？ 这涉及到TCP负载的长度了。假设负载长度为p，那么ACK = x+p。  20  就拿这俩举例子吧，后者ACK = 24025，前者seq = 22645。差值1380正好是前者的payload长度。  21 Explore on your ownQ1:探究拥塞控制以及TCP经典的AIMD行为。可以使用wireshark中的TCP Stream Graph工具来分析并观察拥塞窗口的行为。 查看拥塞控制需要我们自己发包。  22  利用TCP流形图，查看拥塞窗口  23  可以看到，在流量高峰期（中间那段），TCP会调整流量窗口大小。  TCP AIMD（Additive Increase Multiplicative Decrease）是一种TCP拥塞控制算法，全称是TCP自适应的拥塞控制算法。它根据网络状况动态调整发送速率，以避免网络拥塞的加剧。  在AIMD算法中，当网络状态良好时，发送方会以较快的速度发送数据，这被称为“加法增大”。而当网络出现拥塞时，发送方会减少发送速率，这被称为“乘法减小”。这种自适应的调整方式可以有效地避免网络拥塞的加剧，同时保证数据的可靠传输。 Q2:探究TCP协议的可靠性机制。捕获一个含有segment loss(段缺失)的帧，查看什么触发了重传以及重传时间。并且查看round-trip-estimator。 过滤器：  tcp.analysis.lost\_segment  24  wireshark分析：  25  这本来是对1714号帧的ACK。但是对方确没有收到。触发了重传。RTT = 0.032123000S。 Q3:查看包括SACK选项的TCP帧。 过滤器：  tcp.options.sack  可以看到接收到的字节范围。  26 Q4:TCP是web的传输层协议。查看浏览器是怎么使用TCP并发连接的。 访问多个网站，可以看到本地计算机和不同远程服务器之间的流量。  27 | | | |
| **四．实验心得**  本次TCP实验中，我学到了很多关于网络通信的知识，并且深刻体会到了TCP协议的工作原理。  实验的目的是为了更好地理解TCP协议的工作机制。通过搭建TCP/IP网络环境，配置网络连接，以及编写TCP客户端和服务端程序，我深入了解了TCP协议的握手、数据传输、确认和重传等过程。  对于实验中得到的数据和经验，我进行了深入的分析和思考。例如，在实验中，我发现TCP协议的重传机制是非常重要的，它可以确保数据的完整性和可靠性。此外，我也思考了如何在实际情况中应用TCP协议，比如如何处理数据传输错误等问题。  通过这次实验，我深入了解了TCP协议的工作原理和机制，也掌握了一些网络编程的基本技能。这些经验和启示为我之后的计算机网络学习打下了坚实的基础。 | | | |