TCP 协议分析

前言

在工作中遇到很多tcp 协议相关得问题,有时需要临时看下资料。但是看完过段时间还是会忘记，想不起来具体处理细节，令我倍感恼火与无奈。这次按照完整得逻辑顺序重新梳理TCP协议相关知识，本篇内容更加关注实际的报文分析。

TCP 传输控制协议，是当今互联网世界上最重要和最著名的协议之一。它用于全球各种类型的网络，使数以百万计的数据传输能够到达目的地，并充当数据的载体，将主机彼此连接起来，并允许它们使用各种程序来交换数据。

TCP 由 RFC 793 定义，并于 1981 年底发布。创建此类协议背后的动机是，早在 80 年代初期，计算机通信系统开始在军事、教育和办公环境兴起。因此，需要创建一种机制，在各种介质上进行稳健、可靠和完整的数据传输。TCP 提供了上述所有功能，因此迅速被世界各地迅速采用。

由于 TCP 不太容易在短短一篇文章中讲述清楚，因此本篇文档将其分为6个章节，以涵盖其所有特性并帮助我更好地了解它的工作原理和功能。

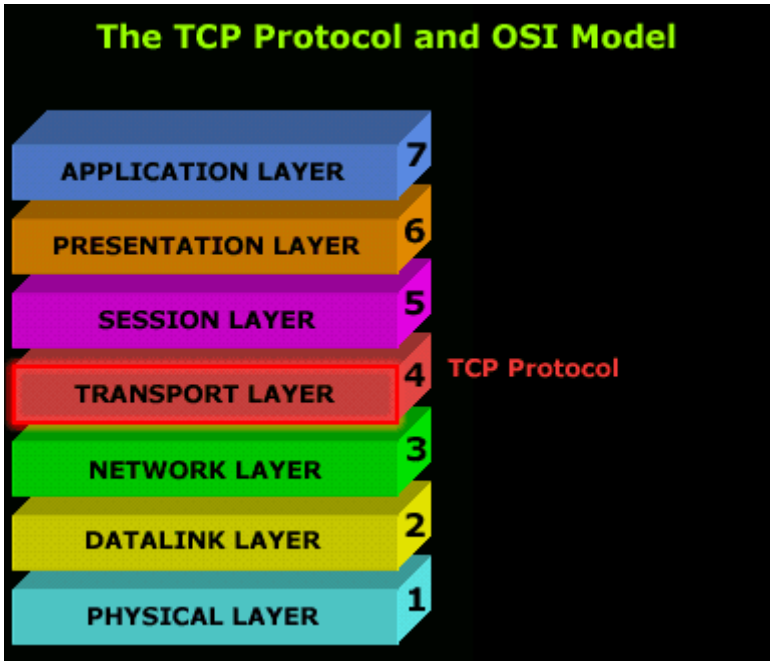
道阻且长，行则将至

第一章、**TCP** 和**OSI** 七层模型

了解每个协议和OSI 模型的关系对于所有网络工程师来说都是必不可少的。本页分析了 TCP 是如何被归类为“传输协议”。

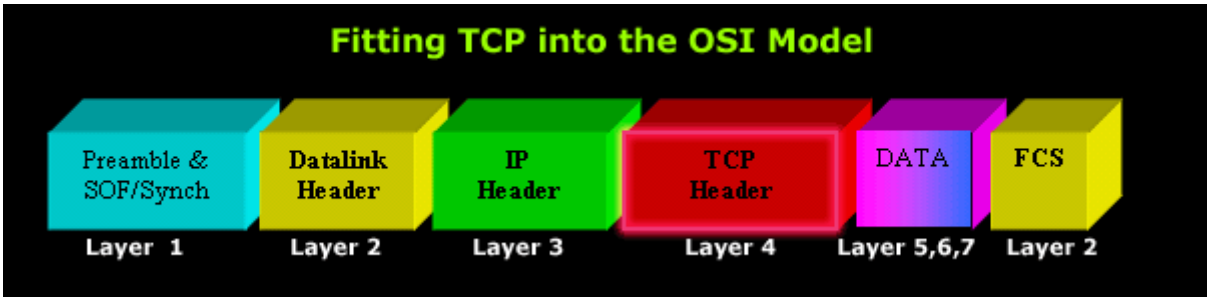
TCP 在**ISO** 模型中的位置

大多数人都知道，每个协议在 OSI 模型中都有自己的位置。 OSI 模型表明了协议的复杂性和智能性。作为一般规则，您在 OSI 模型中向上移动的位置越高，协议就变得越智能。该层的定位也反映了它们的 CPU 密集程度，而 OSI 模型的较低层则恰恰相反，即 CPU 密集程度较低且智能程度较低。



TCP被放置在OSI模型的第四层，也被称为传输层。如果你读过OSI模型的内容，你会记得，传输层负责建立会话、数据传输和拆解虚拟连接。考虑到这一点，你会希望任何放在传输层的协议实现某些特征和特性，使其能够支持该层提供的功能。

下图向你展示了TCP头在一个由计算机生成并发送至网络的帧中的大致顺序。



这个框架是由六个三维块组成的，所以你可以看到每个OSI层都会添加哪一块。你可以看到，包含协议所支持的所有选项的TCP头，就放在IP头（第3层）之后，在包含上层信息的数据部分（第5、6、7层）之前。

注意：对于那些想知道FCS块是否存在的人来说，它包含一个特殊的校验和，由数据链路层放置，以便让接收主机检测当前帧是否在传输过程中被破坏。

我们在什么地方使用**TCP**？

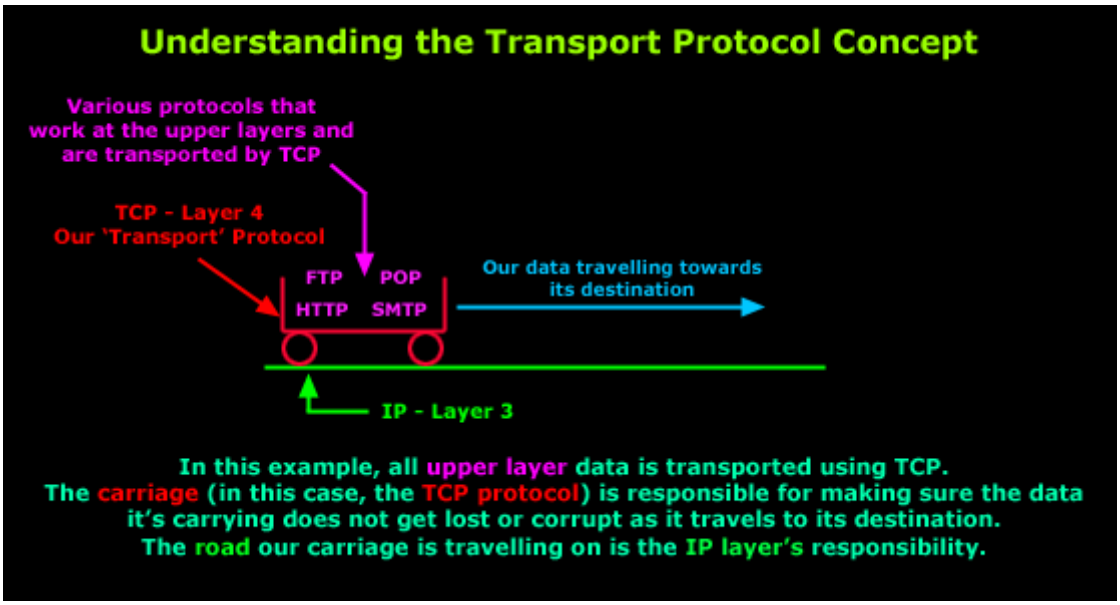
TCP几乎被用于每一种类型的网络中。作为一个协议，它不受限于任何类型的网络拓扑结构，无论是局域网（LAN）还是广域网（WAN）。作为一个传输协议，我们称它为传输协议，因为它位于OSI模型的传输层，它的主要工作是将数据从一个地方送到另一个地方，而不管物理网络和位置如何。

正如大多数人已经知道的，有两种类型的传输协议，TCP是其中之一，UDP（用户数据报协议）是另一种。这两种传输协议的区别在于，TCP提供了一种极其可靠和稳定的数据传输方法，确保所传输的数据不会以任何方式损坏。另一方面，UDP提供了一种不可靠的数据传输方式，不能保证数据已经到达目的地，也不能保证数据到达时的完整性。

传输协议的概念

正如我们提到的，TCP是一个传输协议，这意味着它被用来传输其他协议的数据。起初，这可能听起来很奇怪或令人困惑，但这正是设计它的原因，为它承载的协议增加了大量功能模块。

下图是展示 "传输 "协议概念的最简单方式。



TCP 协议为上层协议数据报文提供传输控制服务。

第二章、**TCP** 协议快速概述

正如之前多次提到的，TCP是传输层的两个协议之一，用于将数据从一台主机传输到另一台主机。TCP之所以如此受欢迎，是因为它在发送和接收数据时的工作方式。与UDP不同，TCP会在它收到的每一个数据包中检查错误，以避免数据损坏，这是一场无休止的工作。

一些常见的使用TCP的协议是: FTP、Telnet、HTTP、HTTPS、DNS、SMTP和POP3。让我们仔细看看这个奇妙协议的主要特点。

当我们提到 "TCP/IP "时，请记住他们谈论的是一套协议簇，而不是像大多数人认为的那样只有一个协议。TCP/IP不是一个协议。

TCP 协议特点

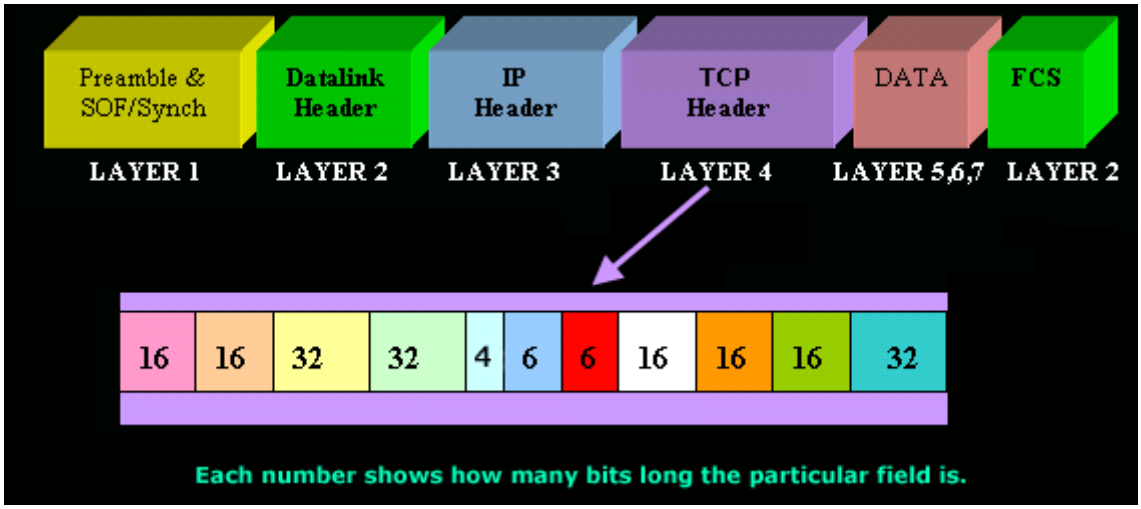
下面是我们要分析的TCP的主要特点。

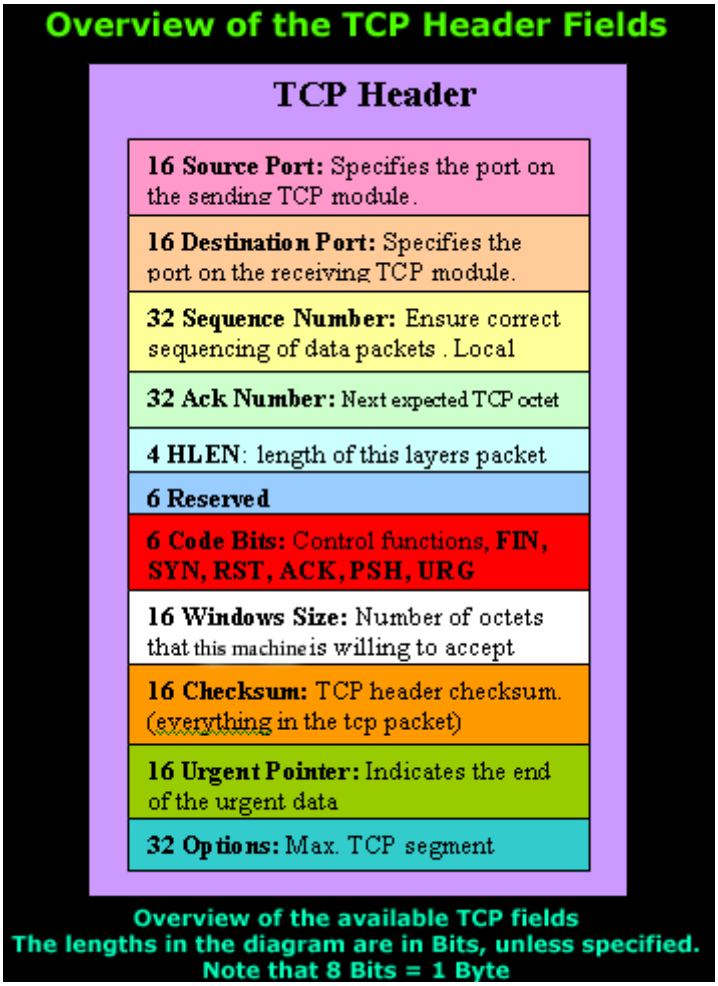
- 可靠的传输
- 以连接为导向
- 流量控制
- 滑动窗口协商
- 确认机制
- 更多的开销

可靠的传输

它是一种可靠的传输，因为它使用不同的技术来确保收到的数据没有错误。

这张图片显示了以太网II帧中的TCP头。在这下面，你会发现第二张图，它放大了TCP头，显示了协议包含的字段。





上边的图显示了TCP报头中每个字段的细分，以及它的长度（比特）。

请记住，8比特等于1字节。

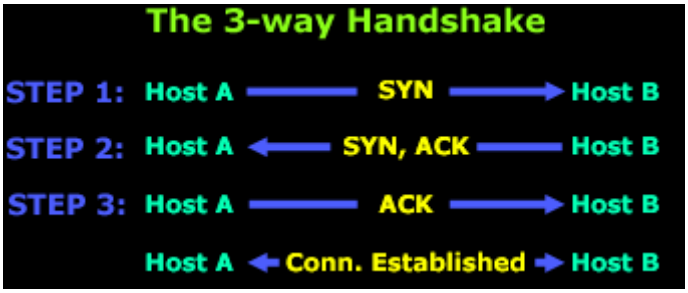
TCP报头中最常用的字段是源端口、目的端口和代码位。这些代码位也被称为 "标志"。

其余的字段有助于确保所有的TCP段都能到达目的地，并以正确的顺序重新组合，同时提供一个无错误的机制。

以连接为导向

这基本上意味着，在传输任何数据之前，两台主机或更确切地说，两台计算机之间建立了一个连接。当使用 "建立连接 "这一术语时，这意味着两台计算机都知道对方，并同意交换数据。这也是著名的三向握手发生的地方。你会在标志位区域发现SYN和ACK位，它们被用来执行3路握手。由于3-way handshake，TCP是面向连接的。

下图解释了三向握手的程序:



- 第1步：主机A向主机B发送初始数据包，该数据包的 "SYN "位已启用。主机B收到该数据包并看到 "SYN "位，其值为 "1"（在二进制中，这意味着打开），因此它知道主机A正试图与它建立连接。
- 第2步：假设主机B有足够的资源，它向主机A发送一个数据包，"SYN和ACK "位被激活（1）。在这一步，主机B发送的SYN意味着 "我想与你同步"，ACK意味着 "我承认你之前的SYN请求"。
- 第3步：所以.....在这之后，主机A向主机B发送另一个数据包，并设置 "ACK "位（1），它实际上告诉主机B'是的，我承认你之前的请求'。

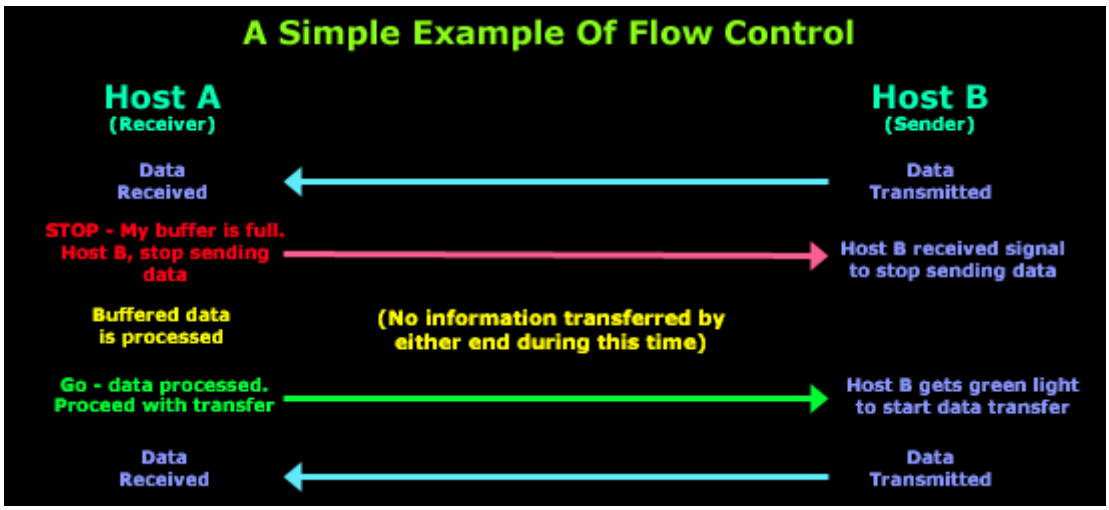
一旦三方握手完成，连接就建立了（虚拟电路），数据传输就开始了。

流量控制

流量控制是用来控制连接之间的数据流。出于任何原因，两台主机中的一台无法跟上数据传输的速度，它可以向另一头发送特殊的信号，要求它停止或放慢速度，以便能够跟上。

例如，如果主机B是一个网络服务器，人们可以从中下载游戏，那么显然主机A不会是唯一从这个网络服务器下载的计算机，所以主机B必须调节数据流到每个从它下载的计算机。这意味着它可能会转向主机A，告诉它等待一段时间，直到有更多的资源可用，因为它有另外20个用户在同时试图下载。

下面是一张图，说明了两个主机之间的简单流控会话。在这一点上，我们只需要理解流控制的概念。



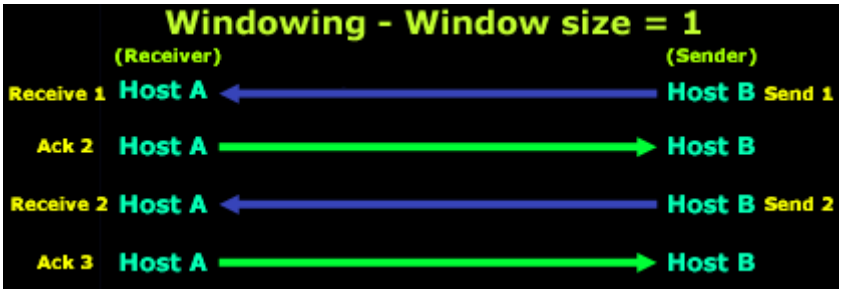
一般来说，当一台机器收到的数据量过快而无法处理时，它会将其存储在一个称为缓冲区的内存部分。只有当数据突发量较小且持续时间不长时，这种缓冲动作才会解决问题。

然而，如果数据突发持续下去，最终会耗尽接收端的内存，这将导致到达的数据被丢弃。因此，在这种情况下，接收端将简单地发出 "未准备好 "或 "停止 "指标给发送方。在接收端处理完其内存中的数据后，会发出一个 "准备好了 "或 "开始 "的传输指示器，而发送机收到 "开始 "指示器后会恢复其传输。

滑动窗口

如果发送机在发送每个数据包（正确的术语是段，我们将在下一页看到）后不得不等待确认，那么数据吞吐量，或传输效率就会很低。因为在发送方发送数据段后，在完成处理来自接收机的确认之前，有可用的时间，发送方利用这段休息时间来发送更多的数据。如果我们想简单地定义窗口（Windowing），我们可以这样想：它是允许发送方机器在没有收到确认的情况下发送的数据段的数量。

窗口控制了多少信息从一端传输到另一端。一些协议通过观察数据包的数量来量化信息，而TCP/IP通过计算字节数来衡量信息。



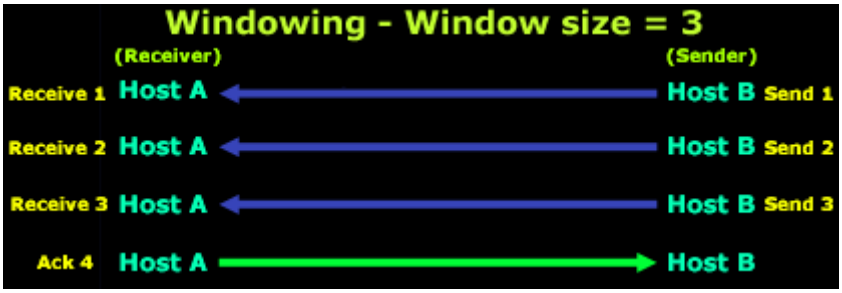
让我解释一下上图中发生了什么。

主机B正在向主机A发送数据，使用的窗口大小等于1。这意味着主机B对它发送给主机A的每一个数据段都期待着一个 "ACK"。

ACK 2 "被主机B翻译为："我确认（ACK）你刚发给我的数据包，我准备接收第二（2）段"。因此，主机B准备好第二个数据段，并将其发送给主机A，期待主机A的 "ACK 3 "响应，这样它就可以发送第三个数据段，如图所示，它收到 "ACK 3"。

然而，如果它再次收到 "ACK 2"，这将意味着之前的传输出了问题，主机B将重新传输丢失的数据段。我们将在稍后的 "确认 "部分看到这一点。现在让我们试试不同的窗口大小，以获得更好的理解.....让我们说3!

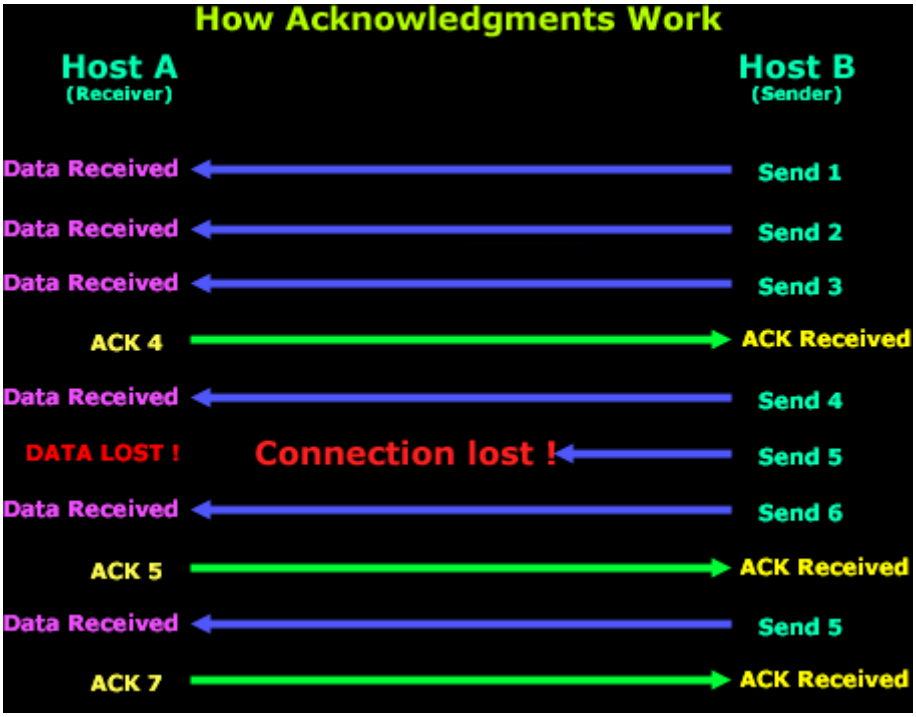
牢记 "ACK "的工作方式，否则你可能会发现下面的例子有点令人困惑。如果你不能理解，请再读一下前面的例子，其中窗口大小等于1。



在上面的例子中，我们有一个等于3的窗口大小，这意味着主机B可以发送3个数据段给主机A，然后期待一个 "ACK "回来。主机B发送了前3个数据段（发送1、发送2和发送3），主机A收到这些数据段的情况良好，然后向主机B发送 "ACK 4"，这意味着主机A确认了主机B发送的3个数据段，并等待下一个数据段，在这种情况下，就是4、5和6。

ACK 确认字段

可靠的数据传输确保了通过一个功能齐全的数据链从一台机器发送到另一台机器的数据流的完整性。这保证了数据不会被重复或丢失。实现这一目标的方法被称为带重传的积极确认。这种技术要求接收机在收到数据时，通过向发送机发送确认信息来与发送源通信。发送方记录它所发送的每一个数据段，并在发送下一个数据段之前等待该确认信息。当它发送一个网段时，发送机启动一个计时器，如果在接收端返回确认信息之前计时器过期，就会重新发送。



该图显示了确认的工作方式。如果你仔细观察这个图，你会发现这个传输的窗口大小等于3。起初，主机B向主机A发送了3个数据段，它们被完美地接收，因此，根据我们学到的知识，主机A发送了一个 "ACK 4"，确认了这3个数据段，并请求下一个3个数据段，这将是4、5、6。结果，主机B发送了数据段4、5、6，但5在途中某个地方丢失了，主机A没有收到

它，所以在等待一段时间后，它意识到5丢失了，于是向主机B发送了 "ACK 5"，表明它希望重新传送数据段5。现在你明白了为什么这种方法被称为 "积极的确认与重传""*positive acknowledgment with retransmission*".。

此时，主机B发送了第5个数据段，并等待主机A发送 "ACK"，这样它就可以继续发送其余的数据。主机A收到第5个数据段并发送 "ACK 7"，意思是 "我收到了上一个数据段，现在请给我发送下一个3段"。下一步在图中没有显示，但它将是主机B发送数据段7、8和9。

更多开销

正如你所看到的，在TCP的架构下有一个相当整洁的机制，使数据的传输没有错误。该协议支持的所有功能都是有代价的，这就是与TCP相关的开销。

当我谈论开销时，我们指的是TCP头中包含的所有不同的字段和错误检查，以确保没有部分数据被破坏。虽然对大多数人来说，这是一个公平的交易，但有些人根本无法接受TCP交互所需的额外处理能力、带宽和增加的时间，因此我们有替代的UDP协议，你使用UDP协议。

第三章、**TCP** 头字段

这篇文章展示了TCP头和段。我解释了TCP头和段在以太网帧中的位置，还简要地查看了TCP头中的可用选项。我们易于理解的详细图表有助于确保所提供的所有信息都易于理解。

所以请做好准备。这一切其实都很简单，你只需要理清思路，尝试以最简单的形式看待事物，你就会发现TCP到底有多简单和友好。只有当你了解了某件事情之后，你才能感到舒服。当你感到困惑时请慢慢的审视自己。

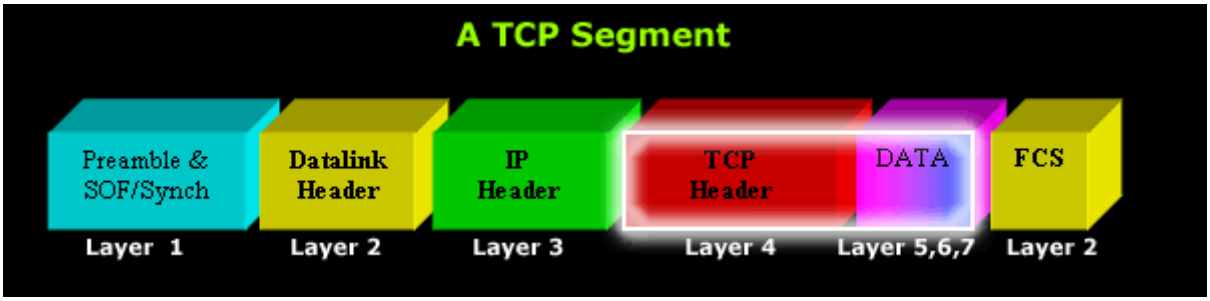
TCP头和**TCP**段

如果我想更准确地使用术语，那么也许我们可以将本页标题定为 "分析一个TCP段"。为什么？因为它在网络世界中就是这么叫的，所以我们需要知道它的正确名称。

这当然会让我们想到另一个新的定义：TCP段***TCP segment***:。

"机器上的TCP软件之间的传输单位被称为TCP段"。

理解这个术语比你阅读定义时想象的要容易，只要好好看看下面的图就可以了。



现在你看到，一个TCP段基本上是TCP头加上紧随其后的数据，当然，数据属于上层（5、6、7）。

数据内容可以是文件传输的一部分，也可以是http请求的响应，事实上，我们对数据的内容不感兴趣，只对它是TCP段的一部分内容感兴趣。

下面的屏幕截图来自我的数据包嗅探器，它显示了属于TCP头的数据部分。

```
应用显示过滤器 … <Ctrl-/>
Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
Ethernet II, Src: ASUSTekC_b3:01:84 (00:1d:60:b3:01:84), Dst: Actionte_2f:47:87 (00:26:62:2f:47:87)
Internet Protocol Version 4, Src: 192.168.1.140, Dst: 174.143.213.184
Transmission Control Protocol, Src Port: 57678, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 57678
    Destination Port: 80
    [Stream index: 0]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 2387613953
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    1010 .... = Header Length: 40 bytes (10)
  > Flags: 0x002 (SYN)
    Window: 5840
    [Calculated window size: 5840]
    Checksum: 0x8f47 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
  > [Timestamps]
```

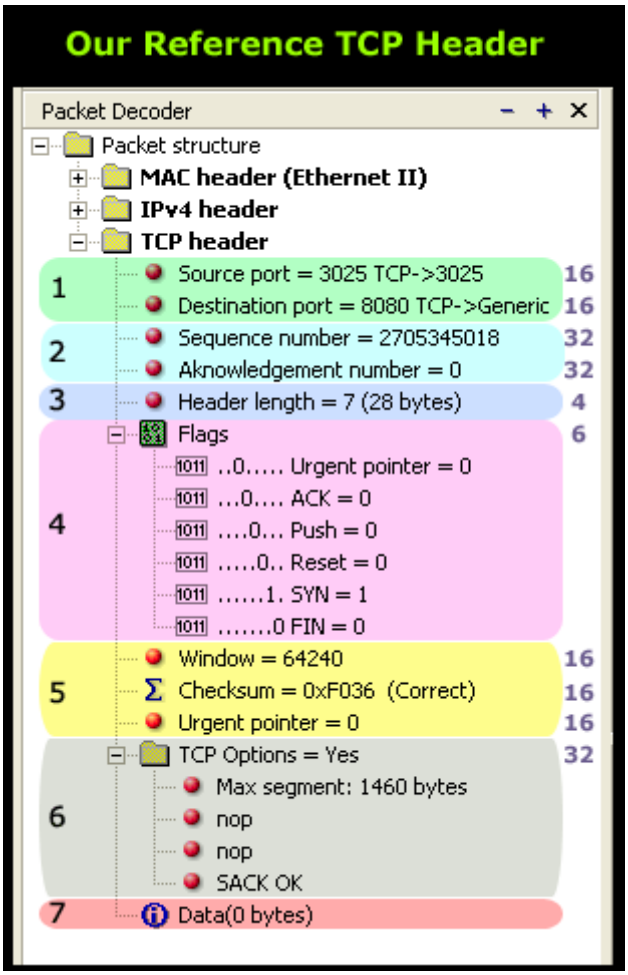
如果你试图从任何一个数据包嗅探器中捕捉一个类似的数据包，它更有可能显示TCP头中的数据部分，就像上边的屏幕截图一样。

因此，问题是TCP头和TCP段是否基本上是一回事。尽管看起来它们是一样的，但在大多数情况下，当提到TCP头时，我们谈论的是没有数据的头，而TCP段则包括数据。

我现在准备开始研究TCP头的结构了。但是，一定要记住，"TCP头 "与 "TCP段 "是一回事，也就是说，它是TCP头信息加上数据，就像上面的图所示。

第四章、深入分析TCP 头字段

下图显示了从我在网络上运行的一个数据包中捕获的TCP头。我们将用它来帮助我们完成对TCP的逐步分析。



正如你所看到的，TCP头已经被完全展开，向我们展示了该协议所包含的所有字段。右边的数字是每个字段的长度，单位是比特。这也显示在快速TCP概述页面中。

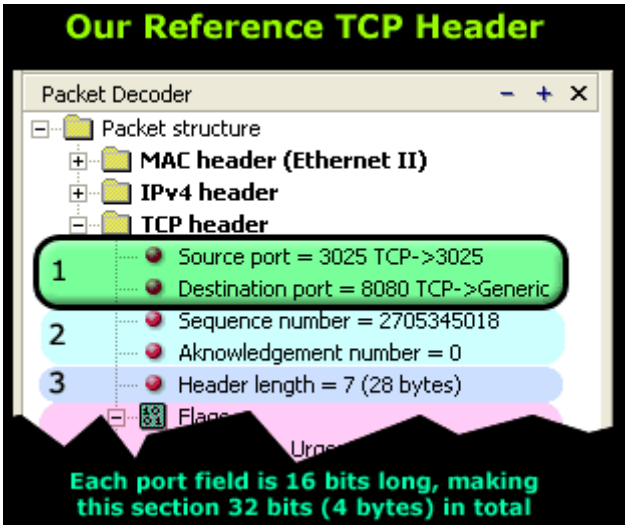
我们还应该指出，当我们例子中的数据包到达目的地时，只有第7节（最后一节）被发送到上层OSI层，因为它包含了它所等待的数据。其余的信息（包括MAC头、IP头和TCP头）都是开销，其目的是将数据包送到目的地，并允许接收端弄清如何处理数据包，例如将数据发送到正确的本地应用程序。

下面我们将对TCP 头字段逐个分析:

4.1 源和目的端口号

本节包含TCP头中最著名的字段之一，即源和目的端口号。这些字段用于指定本地或远程主机上提供的应用程序或服务。我们解释了TCP源端口和目的端口的重要性和功能，同时还有大量的例子。

你将了解到端口有多重要，以及它们如何被用来获取被作为攻击目标的远程系统的信息。我们将使用详细的例子和彩色图表来介绍基本的和高级的端口通信，但现在，我们将从一些基础知识开始，以帮助分解这个主题，使我们能够顺利地理解到更高级和复杂的信息。



当一个主机需要产生一个请求或发送数据时，它需要一些信息。

- 所需主机的IP地址，它想向其发送数据或请求。
- 数据或请求应该发送到远程主机上的端口号。在请求的情况下，它允许发送者指定它打算使用的服务。我们将很快对此进行分析。

1.IP地址是用来唯一识别我们需要联系的主机的。这一信息没有显示在上述数据包中，因为它存在于位于我们正在分析的TCP头上方的IP头部分。

2.第二个重要的方面，端口号，允许我们识别我们的数据或请求必须发送到的服务或应用程序，正如我们之前所说。当一个主机，无论是简单的计算机还是专用服务器，提供各种服务，如http、ftp、telnet，所有连接到它的客户必须使用一个端口号来选择他们想使用的特定服务。

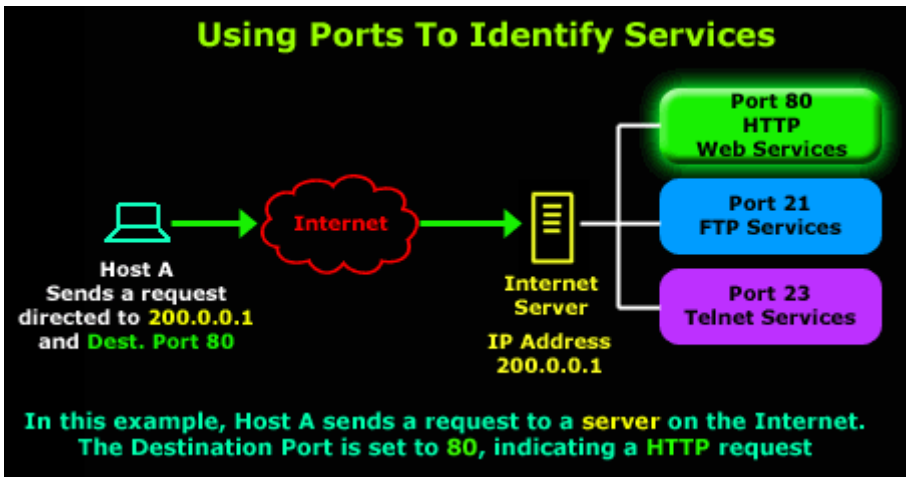
理解这个概念的最好方法是通过例子，下面有很多例子，所以让我们看一看，从一个简单的例子开始，然后向稍微复杂的东西发展。

目的端口

当您发送一个下载网页的 http 请求时，它必须被发送到正确的 Web 服务器，以便它接收、处理它并允许您查看您想要的页面。这是通过 DNS 解析获取正确的 IP 地址并将请求发送到远程计算机（Web 服务器）的正确端口号来实现的。对于 http 请求，端口值通常为 80。

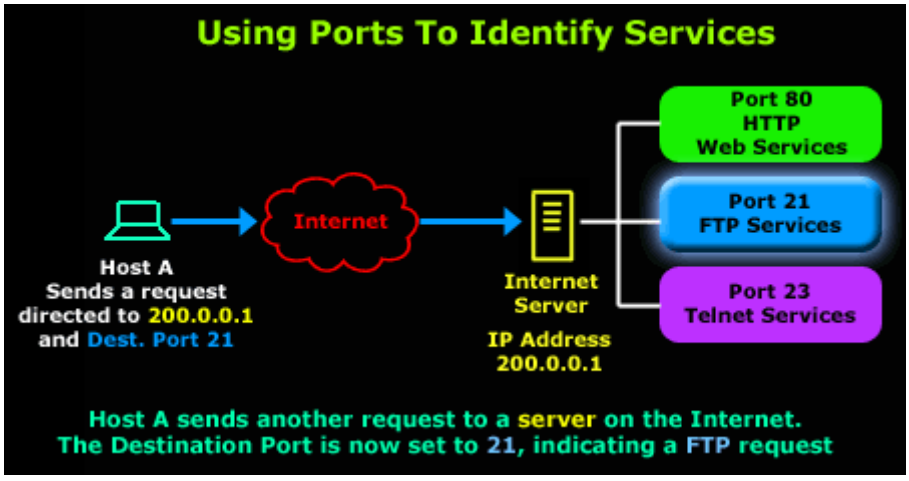
一旦您的请求到达 Web 服务器，它就会检查数据包是否确实是发给它自己的。这是通过观察新收到的数据包的目标 IP 地址来完成的。请记住，此特定步骤是网络层的功能。

一旦它验证数据包实际上是发往本地机器的，它将处理数据包并看到目标端口号等于 80。然后它意识到它应该将数据（或请求）发送到正在等待的 http 守护进程服务客户的背景：



使用这种简洁的方法，我们能够使用服务器提供的其余服务。因此，为了使用 FTP 服务，我们的工作站生成一个数据包，该数据包指向服务器的 IP 地址，即 200.0.0.1，但这次目标端口为 21。

下图说明了这个过程：



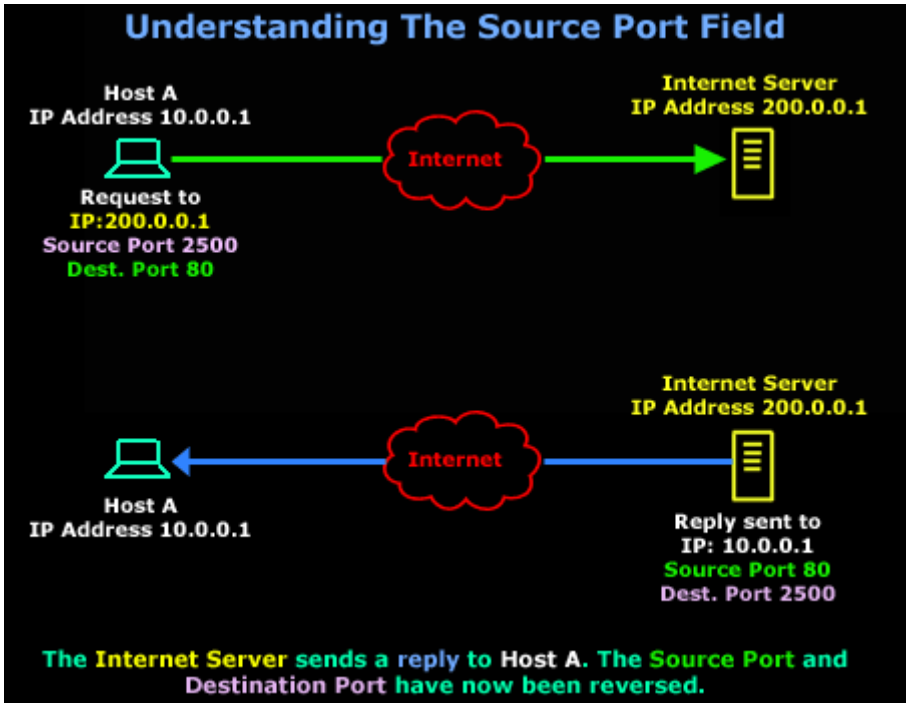
到目前为止，您应该了解目标端口的用途以及它如何允许我们从提供它们的主机中选择我们需要的服务。

源端口

源端口类似于目标端口，但被发送主机用来帮助跟踪新的传入连接和现有数据流。

众所周知，在 TCP/UDP 数据通信中，主机将始终提供目标端口号和源端口号。我们已经分析了目标端口，以及它如何让主机选择它需要的服务。源端口提供给远程机器，在我们的示例中，这是 Internet 服务器，以便它回复另一方发起的正确会话。

这是通过颠倒目标端口和源端口来实现的。当主机（在我们的示例中为主机 A）收到此数据包时，它将将该数据包识别为对其发送的先前数据包的回复：



当主机 A 收到 Internet 服务器的回复时，传输层将注意到反向端口并将其识别为对其发送的前一个数据包（带有绿色箭头的数据包）的响应。

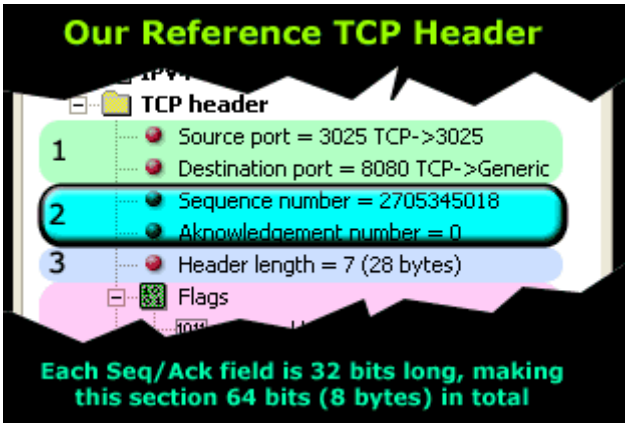
传输层和会话层跟踪所有新连接、已建立连接和正在断开的连接，这解释了主机 A 如何记住它正在等待来自 Internet 服务器的回复。

当然，页面开头显示的捕获数据包显示的端口号与这些图中的端口号不同。在这种特殊情况下，工作站将请求发送到其在端口 8080 上运行的本地 http 代理服务器，使用端口 3025 作为其源端口。

4.2 序列号和确认号

此章将仔细了解序列号和确认号。它们存在的真正目的与以下事实直接相关：Internet 以及通常大多数网络都是数据包交换的（我们将在稍后解释），并且因为我们几乎总是发送和接收大于最大传输单元（又名 MTU）的数据- 在第 5 节和第 6 节中分析）在大多数网络上为 1500。

先来看看我们要分析的字段：



如您所见，序列号在确认号之后。

我们将解释这些数字如何增加以及它们的含义，各种操作系统如何以不同的方式处理它们，最后，这些数字如何成为那些需要可靠安全网络的人的安全隐患。

TCP - 面向连接的协议

Sequence 和 Acknowledgement 字段是帮助我们将 TCP 分类为面向连接的协议的众多特征中的两个。因此，当通过 TCP 连接发送数据时，它们帮助远程主机跟踪连接并确保在到达目的地的途中没有丢失数据包。

TCP 利用积极确认、超时和重传来确保无差错、按顺序交付用户数据。如果重传计时器在收到确认之前超时，则从流中最后一个确认字节之后的字节开始重传数据。

值得一提的另一点是序列号在每个操作系统上的生成方式不同。使用特殊算法（有时是弱算法），操作系统将生成这些数字，用于跟踪发送或接收的数据包，并且由于序列和确认字段都是 32 位，因此有 2^32= 4,294,967,296 种可能性生成一个不同的数字！

初始序列号 **(ISN)**

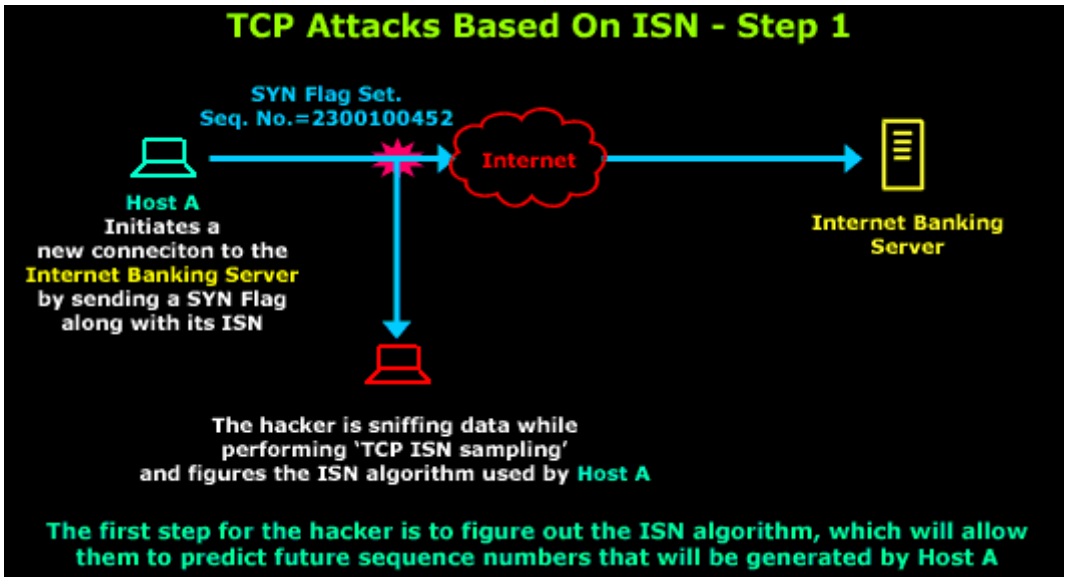
当两台主机需要使用 TCP 传输协议传输数据时，就会创建一个新的连接。这涉及到希望启动连接的第一台主机，以生成所谓的初始序列号 (ISN)，这基本上是我们正在查看的序列字段中包含的第一个序列号。 ISN 一直是安全问题的主题，因为它似乎是黑客最喜欢“劫持”TCP 连接的方式。

信不信由你，劫持一个新的 TCP 连接是经验丰富的黑客只需很少的尝试就能惊人地实现的事情。这个安全问题的根源始于 ISN 的生成方式。

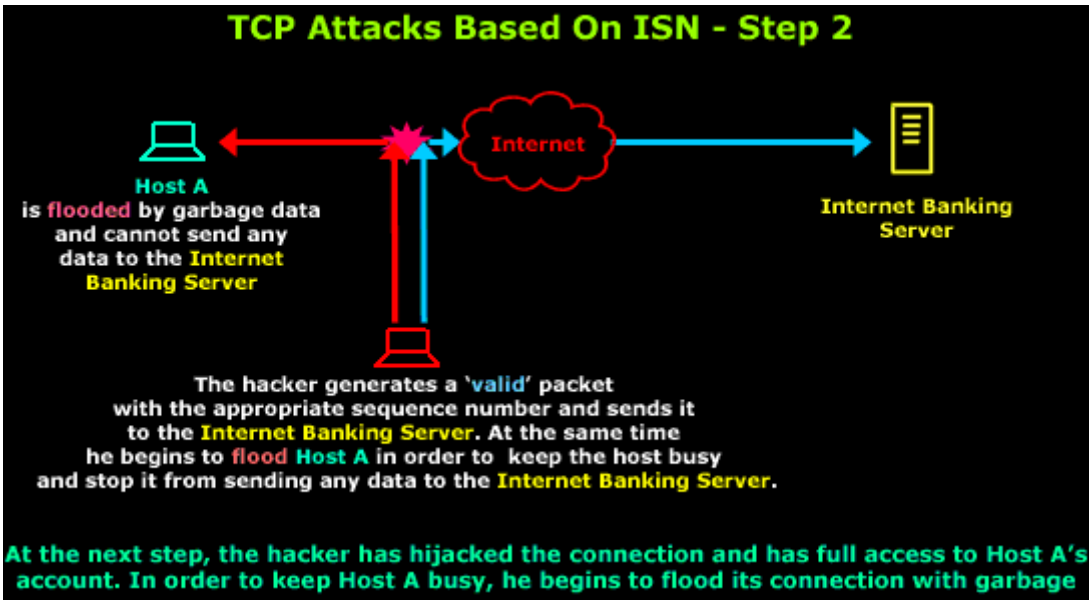
每个操作系统都使用自己的算法为每个新连接生成一个 ISN，因此黑客需要做的就是弄清楚，或者更确切地说，预测特定操作系统使用的算法，生成下一个预测序列号并将其放置在发送到另一端的数据包中。如果攻击者成功，接收端就会被愚弄，并认为数据包是来自发起连接的主机的有效数据包。

同时，攻击者会对发起TCP连接的主机发起泛洪攻击，使其保持忙碌状态，从而不会向试图发起连接的远程主机发送任何数据包。

以下是上述攻击的简要说明：



如前所述，黑客必须通过采样主机 A 在所有新连接中使用的初始序列号来找到 ISN 算法。一旦完成并且黑客知道了算法，他们就可以开始攻击了：



黑客攻击的时机至关重要，因此他将第一个伪造数据包发送到网上银行服务器，同时开始向主机 A 发送大量垃圾数据，以消耗主机的带宽和资源。这样一来，主机 A 就无法处理它接收到的数据，也不会向网上银行服务器发送任何数据包。

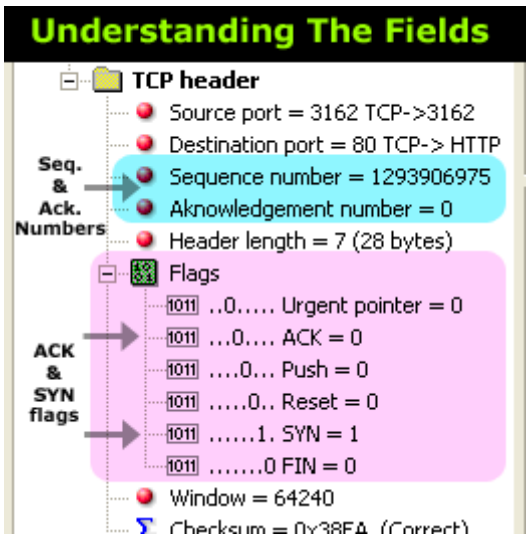
发送到网上银行服务器的虚假数据包将包含有效的标头，这意味着它看起来像是来自主机 A 的 IP 地址，并将发送到网上银行服务器正在侦听的正确端口。

网上发布了大量报告，讨论每个操作系统用于生成其 ISN 的方法以及预测它的难易程度。不要惊慌地发现 Windows 操作系统的 ISN 算法是迄今为止最容易预测的！

诸如“nmap”之类的程序将进行实际测试，以了解发现在任何操作系统中使用的 ISN 算法有多难。在大多数情况下，黑客将首先从主机受害者那里采样 TCP ISN，在响应连接请求时寻找 TCP 实现选择的初始序列号中的模式。一旦发现模式，主机发起的连接就会被劫持，这只是几分钟的事情。

序列号和确认号示例

在我们继续之前，我们应该注意到您会遇到术语“ACK 标志”或“SYN 标志”；这些术语不应与序列号和确认号混淆，因为它们是 TCP 标头中的不同字段。下面的屏幕截图可帮助您理解：

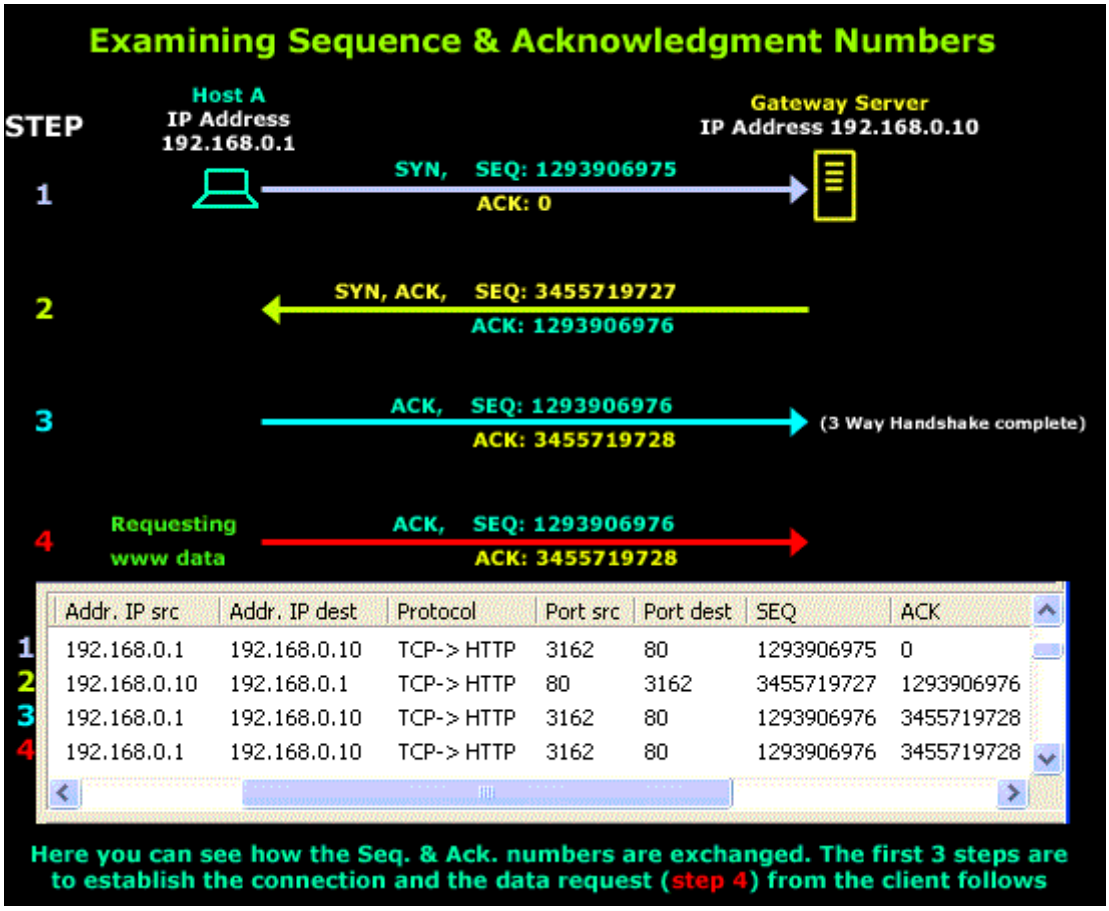


您可以看到序列号和确认号字段，然后是我们所指的 TCP 标志。

TCP 标志（浅紫色部分）将在页面上进行更深入的介绍，但是因为我们现在需要使用它们来帮助我们检查序列号和确认号是如何工作的，所以我们不得不分析一小部分他们中的。

为简单起见，请记住，在谈论序列号和确认号时，我们指的是蓝色部分，而 SYN 和 ACK Flags指的是浅紫色部分。

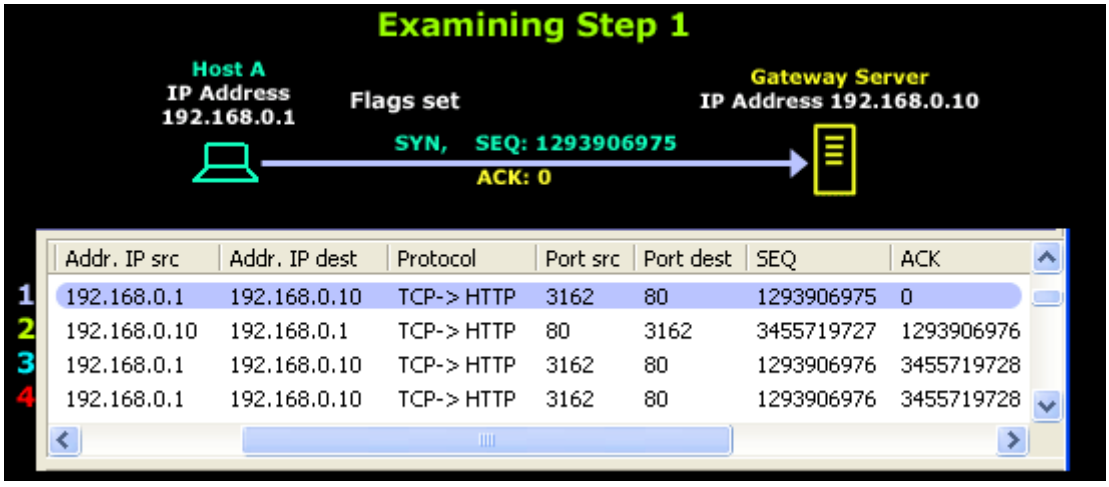
下图显示了与 Web 服务器（网关服务器）的新连接的建立。前三个数据包是 TCP 在两台主机之间传输任何数据之前执行的 3 次握手的一部分，而图表下方的小屏幕截图是由我们的数据包嗅探器捕获的：



为了确保我们理解这里发生了什么，我们将逐步分析这个例子。

步骤**1**

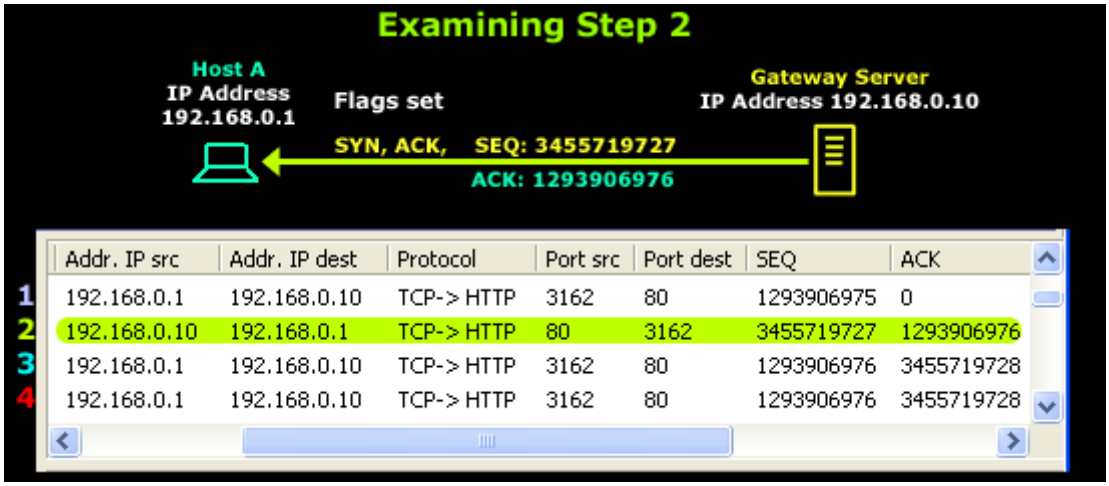
主机 A 希望从网关服务器下载网页。这需要在两者之间建立新的连接，以便主机 A 向网关服务器发送数据包。此数据包设置了 SYN 标志，还包含主机 A 的操作系统生成的 ISN，即 1293906975。由于主机 A 正在启动连接并且尚未收到网关服务器的回复，因此确认号设置为零（ 0).



简而言之，主机 A 告诉网关服务器以下内容：“我想与您建立新连接。我的序列号是 1293906975”。

步骤2

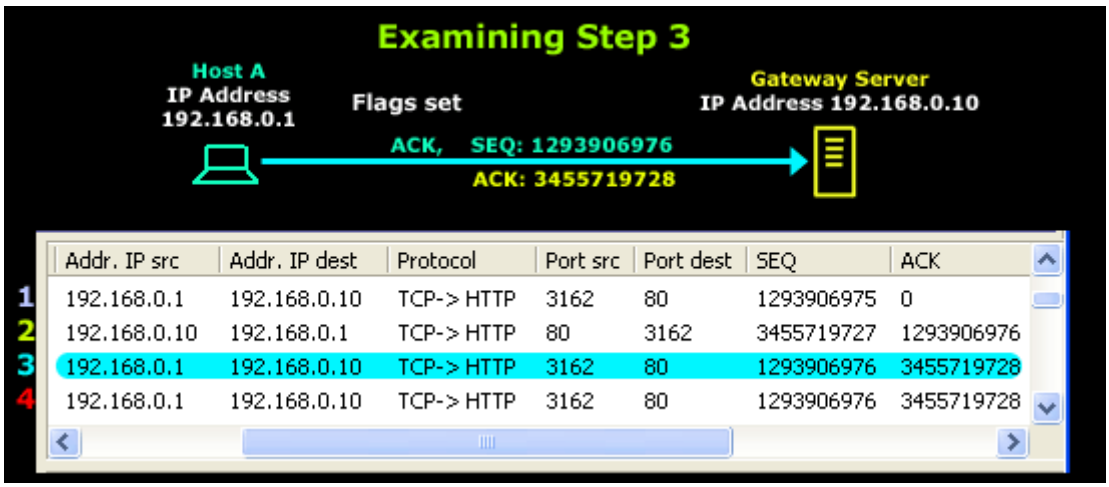
网关服务器收到主机 A 的请求并生成一个回复，其中包含它自己生成的 ISN，即 3455719727，以及它期望来自主机 A 的下一个序列号，即 1293906976。服务器还设置了 SYN 和 ACK 标志，确认先前的它收到的数据包并通知主机 A 它自己的序列号。



简而言之，网关服务器告诉主机 A 以下内容：“我确认您的序列号并期待您的下一个序列号为 1293906976 的数据包。我的序列号是 3455719727”。

步骤3

主机 A 收到回复，现在知道网关的序列号。它生成另一个数据包以完成连接。此数据包设置了 ACK 标志，还包含它希望网关服务器接下来使用的序列号，即 3455719728。



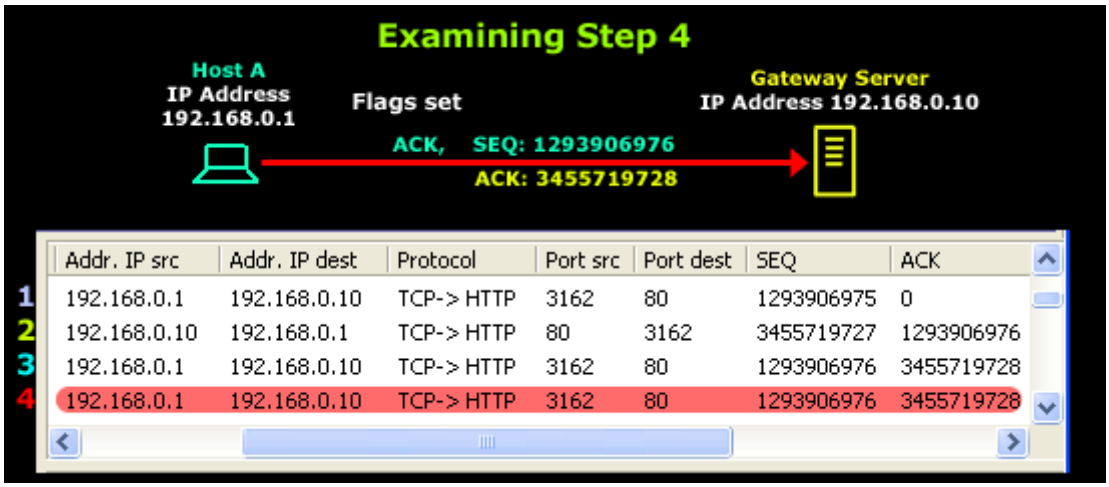
简而言之，主机 A 告诉网关服务器如下：“我确认你的最后一个数据包。这个数据包的序列号是 1293906976，这是你所期望的。我也期待你发送给我的下一个数据包有一个序列号 3455719728”。你发现了三次握手确认号是对端的序列号。

现在，有人可能期望网关服务器发送下一个数据包，但事实并非如此。您可能还记得主机 A 发起连接是因为它想从网关服务器下载网页。由于 3 次 TCP 握手已经完成，两者之间的虚拟连接现已存在，网关服务器已准备好监听主机 A 的请求。

考虑到这一点，现在是主机 A 请求它想要的网页的时候了，这将我们带到了第 4 步。

步骤**4**

在此步骤中，主机 A 生成一个包含一些数据的数据包并将其发送到网关服务器。数据告诉网关服务器它想发送哪个网页。



请注意，第 4 行中的段的序号与第 3 行中的相同，因为 ACK 不占用序号空间。

所以请记住，任何生成的数据包，只是对先前接收到的数据包的确认（换句话说，只设置了 ACK 标志，不包含任何数据），永远不会增加序列号。而SYN 一般会加1。

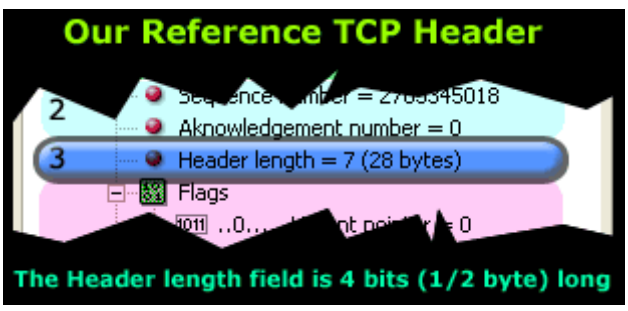
最后

在两台主机的通信过程中，序列号和确认号还有其他重要作用。由于段（或数据包）在 IP 数据报中传输，它们可能会丢失或乱序传送，因此接收方使用序列号对段重新排序。接收方从到达的段中收集数据并重建正在发送的流的精确副本。

如果仔细查看上图，我们会注意到 TCP 确认号指定接收方期望的下一个段的序列号。只需滚动回第 2 步，您就会明白我的意思。

4.3 TCP 头字段长度

让我们快速浏览一下 TCP 报头长度字段，注意它在 TCP 结构中的位置：



您可能还会在其他数据包嗅探器或应用程序中看到标头长度表示为“Data offset数据偏移量”，这实际上与标头长度相同，只是名称更“漂亮”。

分析包头长度

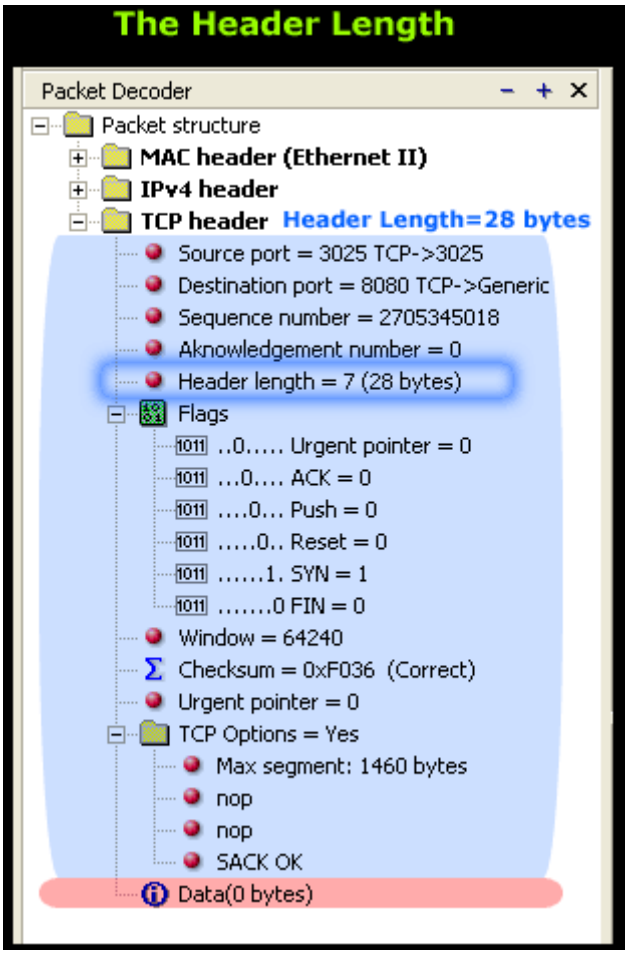
如果您打开任何涵盖 TCP 标头的网络书籍，您几乎肯定会找到该特定字段的以下描述：

“指定以 32 位倍数测量的段标头长度的整数”（Internetworking with TCP/IP，Douglas E. Comer，第 204 页，1995 年）。这个描述听起来令人印象深刻，但是当您查看数据包时，您很可能会挠头思考：这到底是什么意思？

好吧，您可以不再感到困惑，因为我们将逐步介绍它，为您可能遇到的所有可能问题提供答案。如果我们没有完全涵盖您的问题，那么...总有我们的论坛可以求助！

第 **1** 步 - “包头长度”是什么部分？

在我们深入分析这个字段中使用的值的含义之前，顺便说一句，它随着每个数据包而变化，我们需要了解数据包的哪一部分是“Header length""”。



查看上侧的屏幕截图，浅蓝色突出显示的部分向我们显示计入标题长度值的部分。考虑到这一点，您可以看到浅蓝色部分的总长度（标头长度）为 28 个字节。

Header length 字段是必需的，因为 TCP Options 字段包含可能使用或可能不使用的各种选项。从逻辑上讲，如果不使用任何选项，则标头长度会小得多。

如果您查看我们的示例，您会注意到“TCP Options”等于“yes”，这意味着此字段中有用于此特定连接的选项。我们扩展了该部分以显示使用的 TCP 选项，这些选项是“Max Segment”和“SACK OK”。这些将在接下来的页面中进行分析，目前我们对是否使用 TCP 选项感兴趣。

当我们屏幕截图中的数据包到达接收端时，接收方将读取标头长度字段并确切知道数据部分从哪里开始。

该数据将被传送到上面的层，而 TCP 标头将被剥离和忽略。在这个例子中，我们没有数据，这是正常的，因为数据包正在启动 3 次握手（标志，SYN=1），但我们将在下一页更深入地介绍它。

需要我们注意的主要问题涉及用于标头长度字段的值以及学习如何正确解释它们。

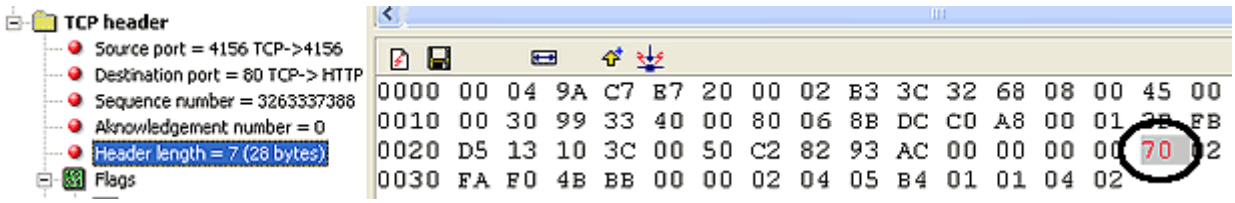
第 2 步 - **header** 头值分析

从上面的屏幕截图中，我们可以看到我们的数据包嗅探器指示该字段的值为 7（十六进制），这被解释为 28 字节。要计算这个，您可以将值 7 乘以 32，然后将结果除以 8：7x32=224/8=28 字节。

您还记得本篇开头给出的定义吗？“指定以 32 位倍数测量的段标头长度的整数”。这是描述这些计算的正式方式 :)

给出的计算是由我们的数据包嗅探器自动执行的，这很周到，你不同意吗？如果您愿意，可以将其视为大多数严肃的数据包嗅探器上的附加“功能”。

下面你会发现我们的数据包嗅探器的另一个屏幕截图，它显示了包含头部长度字段的 TCP 头部（左框架）的一部分。在右边的框架中，数据包嗅探器以十六进制显示数据包的内容：



通过选择左侧的标题长度字段，程序会自动突出显示右侧框架中的相应部分和十六进制值。根据数据包嗅探器，十六进制值“70”是标头长度字段的值。

注意：在十六进制中，每个字符例如“7”代表 4 bit。这意味着在右侧框架中，只有“7”应该突出显示，而不是“70”。此外，如果我们将“7”十六进制转换为二进制，结果将是“0111”（注意总位数等于 4）。

概括

'Header length' 字段非常简单，因为它仅包含一个数字，允许接收端计算 TCP 标头中的字节数。同时，它是强制性的，因为没有它，接收者就无法知道数据部分从哪里开始！

从逻辑上讲，无论 TCP 标头在哪里结束，数据都会开始 - 这在本页提供的屏幕截图中很清楚。因此，如果您发现自己在分析数据包并试图找出数据的起始位置，您需要做的就是找到 TCP 标头，读取“标头长度”值，您就可以准确地找到数据部分的起始位置！

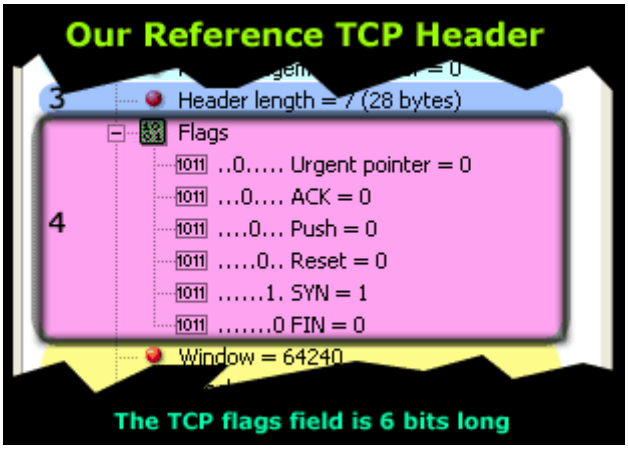
接下来是我们大多数人在谈论著名的 3 次握手和 TCP 在交换数据之前创建的虚拟连接时遇到的 TCP 标志。

4.4 TCP Flags 标志选项

正如我们在前几页中看到的，一些 TCP 段携带数据，而其他段是对先前接收到的数据的简单确认。流行的 3 次握手利用 TCP 中可用的 SYN 和 ACK 来帮助在传输数据之前完成连接。

我们的结论是每个 TCP 段都有一个目的，这是在 TCP 标志选项的帮助下确定的，允许发送方或接收方指定应使用哪些标志，以便另一端正确处理该段。

让我们看一下 TCP 标志字段以开始我们的分析：



您可以看到在 3 次握手（SYN、ACK）和数据传输期间使用的 2 个标志。

与所有标志一样，值“1”表示特定标志已“设置”，或者，如果您愿意，也可以“打开”。在此示例中，仅设置了“SYN”标志，表示这是新 TCP 连接的第一段。

除此之外，每个标志位长一位，由于有 6 个标志位，因此标志部分总共为 6 位。

您必须同意，最流行的标志是“SYN”、“ACK”和“FIN”，用于建立连接、确认成功的段传输以及最后终止连接。虽然其余标志并不为人所知，但它们的作用和目的使它们在某些情况下同样重要。

我们将从检查所有六个标志开始我们的分析，从顶部开始，即紧急指针：

第一个标志 **- Urgent pointer**

第一个标志是Urgent pointer 标志，如前一个屏幕截图所示。此标志用于将传入数据标识为“紧急”。这样的传入段不必等到接收端消耗掉之前的段，而是直接发送并立即处理。

在主机将数据发送到远程计算机上运行的应用程序的数据传输流期间，可以使用紧急指针。如果出现问题，主机需要中止数据传输并停止另一端的数据处理。在正常情况下，中止信号会被发送并在远程机器上排队，直到所有先前发送的数据都被处理，但是，在这种情况下，我们需要立即处理中止信号。

通过将中止信号的段紧急指针标志设置为“1”，远程机器将不会等到所有排队的数据都被处理后再执行中止。相反，它将给予特定段优先级，立即处理它并停止所有进一步的数据处理。

如果您觉得难以理解，请考虑这个现实生活中的例子：

- 在您当地的邮局，数百辆卡车正在卸下来自世界各地的成袋信件。因为进邮局大楼的货车很多，他们一个接一个地排着队，等着卸行李。
- 结果，队列最终变得很长。然而，一辆挂着大红旗的卡车突然加入了队列，负责确保没有卡车逃过队列的安保人员看到了红旗，就知道它载有非常重要的信件，需要紧急到达目的地.按照正常程序，安保人员示意卡车跳过队列，一直走到最前面，让它优先于其他卡车。
- 在此示例中，卡车代表到达目的地并在缓冲区中排队等待处理的路段，而带有红色标志的卡车是设置了紧急指针标志的路段。

需要注意的另一点是紧急指针字段的存在。该字段在第 5 节中有所介绍，但我们可以简单地提一下，当 Urgent Pointer 标志设置为“1”（即我们在此分析的那个）时，Urgent Pointer 字段指定紧急数据在段中结束的位置.

第二个标志 **- ACK**

确认标志用于确认数据包的成功接收。

如果您在使用 TCP 传输数据时运行数据包嗅探器，您会注意到，在大多数情况下，对于您发送或接收的每个数据包，都会收到一个确认。因此，如果您从远程主机收到一个数据包，那么您的工作站很可能会发回一个 ACK 字段设置为“1”的数据包。

在某些情况下，发送方每发送 3 个数据包就需要一个确认，接收端将发送一次预期的 ACK（收到第 3 个连续数据包）。这也称为窗口化，并在后续页面中进行了广泛介绍。

第三个标志 **- PUSH**

与 Urgent 标志一样，Push 标志的存在是为了确保数据被赋予优先级（它应得的）并在发送端或接收端进行处理。这个特殊的标志在数据传输的开始和结束时使用得非常频繁，影响了两端处理数据的方式。

当开发人员创建新的应用程序时，他们必须确保遵循 RFC 给出的特定指南，以确保他们的应用程序正常工作并完美地管理进出 OSI 模型应用程序层的数据流。使用时，Push 位确保数据段得到正确处理，并在虚拟连接的两端给予适当的优先级。

当主机发送它的数据时，它会暂时在 TCP 缓冲区（内存中的一个特殊区域）中排队，直到该段达到一定大小，然后发送给接收方。这种设计保证了数据传输尽可能高效，不会通过创建多个段来浪费时间和带宽，而是将它们组合成一个或多个更大的段。

当报文段到达接收端时，在被传递到应用层之前，它被放置在 TCP 传入缓冲区中。在传入缓冲区中排队的数据将保留在那里，直到其他段到达，一旦完成，数据将传递到等待它的应用程序层。

虽然此过程在大多数情况下运行良好，但在很多情况下这种数据“排队”是不可取的，因为排队期间的任何延迟都会给等待的应用程序带来问题。一个简单的例子是 TCP 流，例如真实播放器，其中必须立即发送和处理数据（由接收方）以确保流畅的流，没有任何中断。

这里要提到的最后一点是 Push 标志通常设置在文件的最后一段以防止缓冲区死锁。当用于通过代理发送 HTTP 或其他类型的请求时，也可以看到它 - 确保请求得到适当和有效的处理。

第 **4** 个标志 **- 复位 (RST)** 标志

当到达的段不是用于当前连接时，将使用重置标志。换句话说，如果您要向主机发送数据包以建立连接，而远程主机上没有此类服务等待应答，则主机会自动拒绝您的请求，然后向您发送回复RST 标志设置。这表明远程主机已重置连接。

虽然这可能证明非常简单和合乎逻辑，但事实是，在大多数情况下，大多数黑客都使用此“功能”来扫描主机的“开放”端口。由于“重置”功能，所有现代端口扫描器都能够检测“打开”或“侦听”端口。

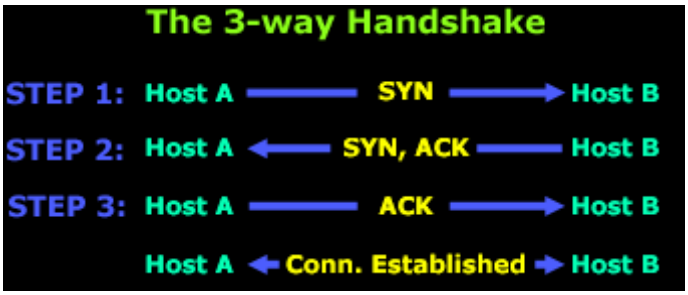
用于检测这些端口的方法非常简单：当尝试扫描远程主机时，将构造一个有效的 TCP 段并设置 SYN 标志 (1) 并将其发送到目标主机。如果没有服务侦听特定端口上的传入连接，则远程主机将使用 ACK 和 RST 标志设置 (1) 进行回复。另一方面，如果有一个服务正在侦听该端口，则远程主机将构造一个 ACK 标志设置为 (1) 的 TCP 段。当然，这是我们介绍的标

准 3 次握手的一部分。

一旦扫描开放端口的主机收到这个报文段，它将完成 3 次握手，然后使用 FIN（见下文）标志终止它，并将特定端口标记为“active”。

第 5 个标志 - **SYN**同步标志

TCP 标志选项中包含的第五个标志可能是 TCP 通信中使用的最广为人知的标志。您可能知道，SYN 标志是在两台主机之间建立经典的 3 次握手时发送的：



在上图中，主机 A 需要使用 TCP 作为其传输协议从主机 B 下载数据。该协议要求进行 3 次握手，以便两端可以建立虚拟连接以交换数据。

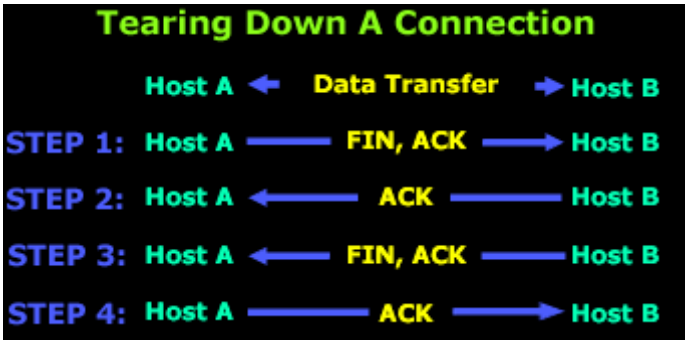
在 3 次握手期间，我们能够计算出总共传输了 2 个 SYN 标志，每个主机一个。随着文件的交换和新连接的创建，我们将看到更多的 SYN 标志被发送和接收。

第 6个标志 - **FIN**标志

最后一个可用的标志是 FIN 标志，代表单词 FINished。该标志用于拆除使用先前标志（SYN）创建的虚拟连接，因此由于这个原因，FIN 标志总是在连接之间交换最后一个数据包时出现。

重要的是要注意，当主机发送 FIN 标志以关闭连接时，它可能会继续接收数据，直到远程主机也关闭连接为止，尽管这仅在某些情况下发生。一旦双方都断开了连接，就会释放两端为连接预留的缓冲区。

正常的拆解过程如下图所示：



上图表示主机 A 和 B 之间的现有连接，其中两台主机正在交换数据。数据传输完成后，主机 A 发送一个设置了 FIN、ACK 标志的数据包（步骤 1）。

有了这个数据包，主机 A 确认先前的流，同时启动 TCP 关闭过程以终止此连接。此时，Host A 的应用程序将停止接收任何数据，并从这一端关闭连接。

为响应主机 A 关闭连接的请求，主机 B 将发回确认（第 2 步），并通知其应用程序连接不再可用。完成后，主机 (B) 将发送自己的 FIN、ACK 标志（第 3 步）以关闭其部分连接。

如果您想知道为什么需要此过程，那么您可能需要回想一下 TCP 是全双工连接，这意味着数据流有两个方向。在我们的示例中，这是从主机 A 到主机 B 的连接流，反之亦然。此外，它要求两台主机都从他们这边关闭连接，因此这就是两台主机都必须发送 FIN 标志并且另一台主机必须确认它这一事实背后的原因。

最后，在第 4 步，主机 A 将确认主机 B 在第 3 步发送的请求，双方的关闭程序现已完成！

概括

此页面处理可用的 TCP 标志选项，使生活变得更困难或更容易，具体取决于您如何看待图片:)

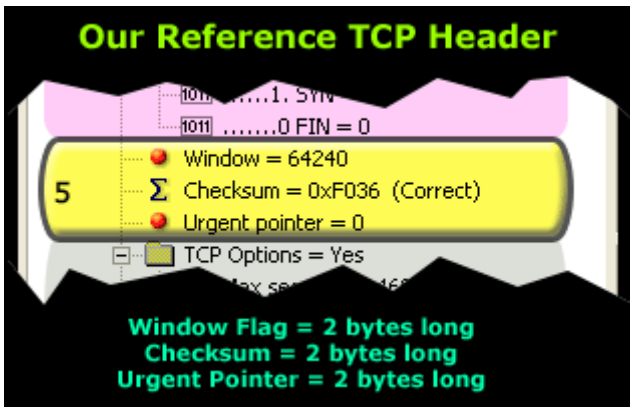
也许此页面上提供的最重要的有助于记住的信息是 TCP 握手过程以及 TCP 是全双工连接的事实。

以下部分将检查 TCP 窗口大小、校验和和紧急指针字段，所有这些都是相关且非常重要的。出于这个原因，我们强烈建议您通读这些主题，而不是跳过它们。

4.5 TCP 窗口大小、校验和和紧急指针

我们的第五部分包含 TCP 传输协议使用的一些非常有趣的字段。我们看到 TCP 如何帮助控制每个段传输的数据量，确保段中没有错误，最后将我们的数据标记为紧急数据，以确保它在离开发送方和到达接收方时获得所需的优先级.

因此，让我们不要浪费任何时间，直接进入我们的分析！



我们这里分析的第五段在TCP头中一共占用了6个字节。

这些值与协议标头中的大多数字段一样，无论应用程序数据量如何，其大小都保持不变。

这意味着虽然它们包含的值会改变，但字段占用的空间总量不会改变。

窗口标志

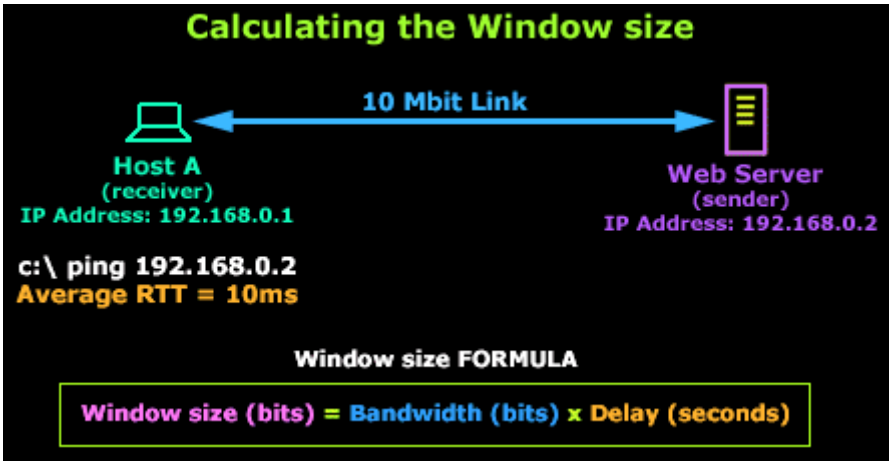
窗口大小被认为是 TCP 标头中最重要的标志之一。接收方使用此字段向发送方指示它能够接受的数据量。无论发送者或接收者是谁，该字段将始终存在并被使用。

您会注意到此页面的最大部分专用于“窗口大小”字段。这背后的原因是因为这个领域非常重要。窗口大小字段是高效数据传输和流量控制的关键。一旦您开始意识到这个标志的重要性以及它包含的功能有多少，它真的是惊人的。

窗口大小字段使用“字节”作为度量标准。因此，在我们上面的示例中，数字 64,240 等于 64,240 字节或 62.7 kb (64,240/1024)。

62.7 kbytes 反映了接收方在向发送方（服务器）传输新的 Window 值之前能够接受的数据量。当传输的数据量等于当前 Window 值时，发送方将期望从接收方收到一个新的 Window 值，以及对刚刚收到的 Window 的确认。

为了保持数据传输的完美无缺和高效率，需要以上过程。然而，我们应该注意，选择的窗口大小字段在任何情况下都不是随机值，而是使用特殊公式计算的值，如下例所示：

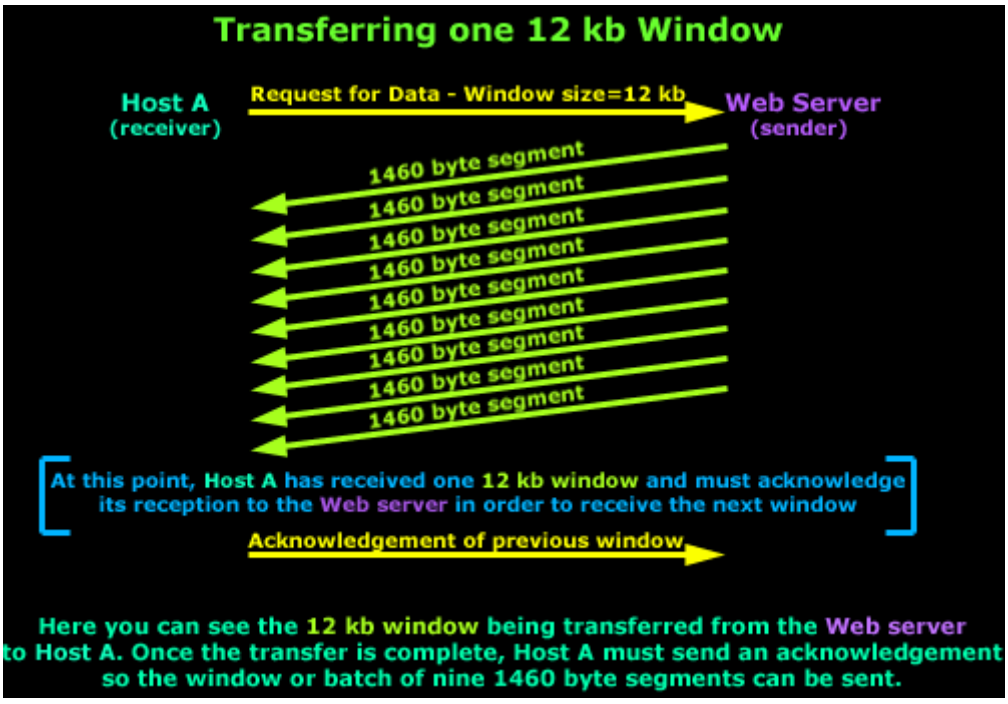


在此示例中，主机 A 通过 10 Mbit 链路连接到 Web 服务器。根据我们的公式，要计算最佳窗口值，我们需要以下信息：带宽和延迟。我们知道链路的带宽：10,000,000 位（10 Mbits），我们可以通过从主机 A 向 Web 服务器发出“ping”来轻松找出延迟，这给我们提供了 10 的平均往返时间响应 (RTT)毫秒或 0.01 秒。

然后我们可以使用此信息来计算最有效的窗口大小 (WS)：

```
WS = 10,000,000 x 0.01 => WS = 100,000 位或 (100,000/8)/1024 = 12.5 KB
```

对于 10 Mbps 带宽和 0.01 秒的往返延迟，这给出了大约 12 kb 或九个 1460 字节段的窗口大小：



这应该在 10 Mbps LAN 上产生最大吞吐量，即使延迟高达 10 ms，因为大多数 LAN 的往返延迟不到几毫秒。当带宽较低时，对于相同的固定窗口大小可以容忍更多的延迟，因此 12 kb 的窗口大小在较低速度下也能很好地工作。

窗口 - 一种流量控制形式

除了窗口概念是有效数据传输的关键因素之外，它还是一种流量控制形式，主机（接收方）能够向另一方（发送方）指示它可以接受多少数据，然后等待如需进一步说明。

事实上，在几乎所有情况下，62 KB 的默认值都用作窗口大小。此外，即使接收方可能选择了 62 kbytes 的窗口大小，在两个主机的数据传输过程中，链路仍会持续监控数据包丢失和延迟，从而导致原始窗口大小的小幅增加或减少优化带宽利用率和数据吞吐量。

这种自动的自我纠正机制确保两台主机将尝试以最佳方式使用连接它们的管道，但请记住，这并不能保证它们总能成功。这通常是用户能够手动修改窗口大小直到找到最佳值的原因，正如我们所解释的，这在很大程度上取决于主机之间的链接及其延迟。

在窗口大小降为零的情况下，远程 TCP 不能再发送数据。它必须等到缓冲区空间可用并且它收到一个数据包，该数据包宣布非零窗口大小。

最后，对于与 Cisco 路由器打交道的人，您可能有兴趣知道您能够在运行 Cisco IOS v9 及更高版本的 Cisco 路由器上配置窗口大小。 12.2(8)T 及以上版本的路由器支持 Window Scaling，此功能会自动启用 Window 大小超过 65,535，最大值为 1,073,741,823 字节！

下一页将更深入地讨论窗口缩放。

在服务器端：更大的窗口大小 = 更多内存

大多数使用过非常繁忙的 Web 服务器的网络管理员都会回忆起他们需要的大量内存。由于我们现在了解“窗口大小”的概念，我们能够快速分析它如何影响繁忙的 Web 服务器，这些服务器有成千上万的客户端连接到它们并请求数据。

当客户端连接到 Web 服务器时，服务器需要为客户端会话预留少量内存 (RAM)。所需内存量与窗口大小相同，正如我们所见，该值取决于客户端和服务器之间的带宽和延迟。

为了让您了解窗口大小如何影响服务器对内存的要求，让我们举个例子：

如果您有一个 Web 服务器在局域网 (LAN) 上为 10,000 个客户端提供服务，速度为 100 Mbits，往返延迟为 0.1 秒，并且希望文件传输的性能/效率最高，根据我们的公式，您需要分配每个客户端一个 1.25 MB 的窗口，或者所有客户端 12 G 的内存！当然假设所有 10,000 个客户端同时连接到您的 Web 服务器。

要支持双向大文件传输（服务器到客户端，反之亦然），您的服务器需要：[(100,000,000 x 0.1) 10,000 x 2] = 超过 24 G 的内存仅用于套接字缓冲区！

所以你可以看到客户不要使用过大的窗口值是多么重要！事实上，当前的 TCP 标准要求接收方必须始终能够接收整个窗口的数据量。如果接收方过度超过其缓冲区空间，它可能不得不丢弃传入的数据包。即使网络不拥塞，发送方也会发现这种丢包并调用 TCP 拥塞控制机制。

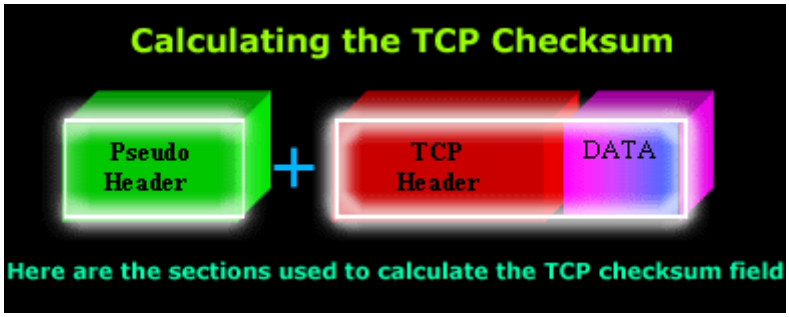
很明显，如果接收器希望保持高性能并避免数据包丢失，则它们不应过度分配缓冲区空间（窗口大小）。

很明显，如果接收器希望保持高性能并避免数据包丢失，则它们不应过度分配缓冲区空间（窗口大小）。

校验和标志

创建 TCP 校验和字段是为了确保 TCP 段中包含的数据到达正确的目的地并且没有错误。对于那些想知道 TCP 如何确保数据段到达正确目的地（IP 地址）的网络专家，您会很高兴地知道，除了 TCP 标头之外，还使用了更多的信息来计算校验和，自然而然, 它将包括 IP 标头的一部分。

这条“额外”信息称为伪标头，我们将很快分析其内容，但现在，让我们查看用于计算 TCP 校验和的部分的可视化表示：



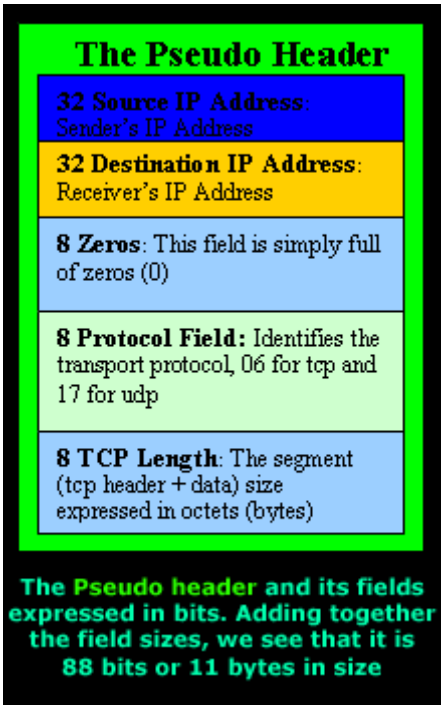
上图显示了伪标头，后面是 TCP 标头和该段包含的数据。但是，请再次记住，伪报头包含在校验和计算中，以确保该段已到达正确的接收方。

现在让我们看看接收方如何能够通过分析伪报头来验证它是刚刚收到的段的正确接收方。

伪标题

伪报头是 5 个不同字段的组合，在计算 TCP 校验和时使用。重要的是要注意（并记住！）伪报头不会传输到接收方，而只是参与校验和计算。

以下是 TCP RFC 定义的 5 个字段：



当数据段到达目的地并通过 OSI 层进行处理时，一旦到达传输层（第 4 层），接收方将重新创建伪标头以重新计算 TCP 标头校验和并将结果与存储在中的值进行比较它收到的段。

如果我们假设该段以某种方式设法找到了通往错误机器的路径，则当重新创建伪标头时，错误的 IP 地址将被插入到目标 IP 地址字段中，结果将是计算出的校验和不正确。因此，不应该接收该段的接收器将丢弃它，因为它显然不应该在那里。

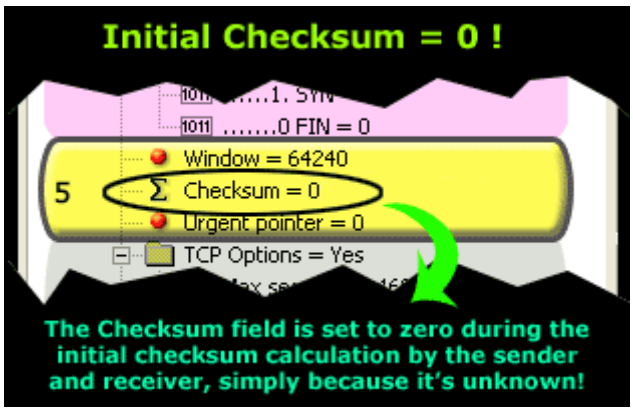
现在您知道了校验和字段如何保证正确的主机将接收数据包，或者它会毫无错误地到达那里！

然而，一定要记住，即使这些机制在理论上存在并且工作得很好，但在实际操作中，有错误的数据包有可能进入应用程序！

一旦您坐下来想一想，在使用 TCP 和 UDP（UDP 计算其校验和的方式相同）作为其传输协议的主机之间发送和接收的每个数据包都会发生此过程，您会感到非常惊奇！

最后，在 TCP 标头校验和计算期间，该字段设置为零 (0)，如下所示。此操作仅在任一端的校验和计算期间执行，因为当时它是未知的。一旦计算出该值，它就会被插入到字段中，替换初始的零 (0) 值。

下面的屏幕截图也说明了这一点：



总结计算校验和时遵循的过程，从发送方一直到接收方发生以下过程：

发送方准备要发送给接收端的段。校验和设置为零，实际上是 4 个零（十六进制）或 8 个零（0000 0000），如果您以二进制形式查看它，因为校验和是一个 8 位字段。

然后使用伪标头、TCP 标头和最后要附加到特定段的数据计算校验和。然后将结果存储在校验和字段中并发送该段！
该段到达接收器并被处理。当它到达 TCP 所在的第 4 个 OSI 层时，校验和字段再次设置为零。然后，接收方将通过在目标 IP 地址字段中输入自己的 IP 地址（如前图所示）为接收到的数据段创建自己的伪标头，并使用 TCP 标头和数据计算新的校验和。

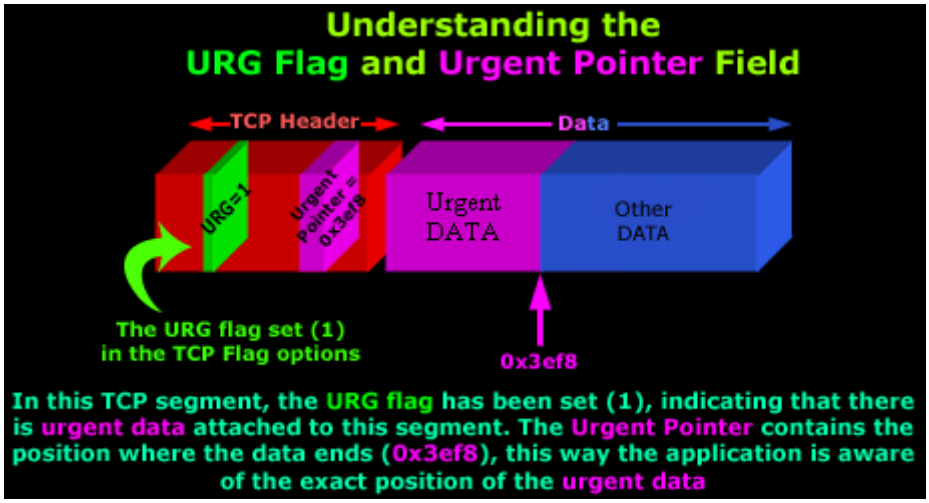
如果一切都成功完成，结果应该与校验和字段段到达时的结果相同。发生这种情况时，将进一步处理数据包，并将数据交给等待它的应用程序。

但是，如果校验和不同，则应丢弃（丢弃）数据包，并根据 TCP 堆栈在接收方操作系统上的实现方式向接收方发送通知。

紧急指针

在第 4 节中，我们分析了 TCP 标志选项，并在其中找到了紧急指针标志。 TCP Flag中的urgent pointer flag允许我们将一段数据标记为“urgent”，而这个urgent pointer字段指定了紧急数据的结束位置。

为了帮助您理解这一点，请看下图：



您可能还想知道在攻击远程主机时也可以使用紧急指针。从我们分析的案例研究中，我们看到某些应用程序，据称可以保护您的系统免受攻击尝试，但在设置 URG 标志时没有正确记录攻击。一个特定的应用程序恰好是著名的 BlackIce Server v2.9，所以要当心！

作为本节的最后结论，如果您发现自己捕获了数千个数据包以查看设置了 URG 位的数据包，如果您无法捕获任何此类数据包，请不要失望！我们发现几乎不可能让我们的工作站使用 telnet、http、ftp 和其他协议生成此类数据包。最好的选择也是迄今为止最简单的方法是寻找数据包制作程序，这些程序允许您创建具有不同标志和选项集的数据包。

概括

虽然这部分内容相当广泛，但我们已经涵盖了 TCP 协议的一些非常重要的部分。您现在知道什么是 TCP 窗口以及如何根据您的带宽和延迟计算它。

我们还检查了 Checksum 字段，接收方使用它来验证它收到的段没有损坏，同时检查以确保它没有意外收到该段！

最后，我们详细检查了 URG 标志和紧急指针字段的用法，它们用于定义包含紧急数据的传入段。

在享受了如此详尽的分析之后，我们相信您已经准备好进行更多分析了！下一节将介绍位于 TCP 标头末尾的 TCP 选项。

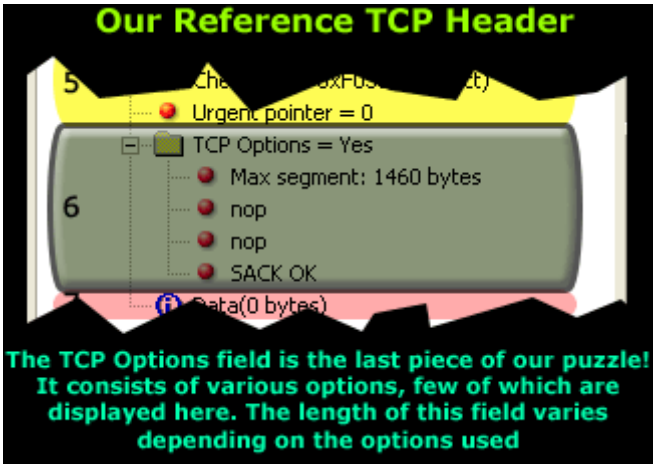
4.6 分析 TCP OPTIONS 标头选项

TCP 选项（MSS、窗口缩放、选择性确认、时间戳、Nop）位于 TCP 标头的末尾，这也是它们最后介绍的原因。多亏了 TCP 选项字段，我们才能够通过引入新功能或“插件”来增强 TCP 协议，有些人喜欢这样称呼它们，它们由各自的 RFC 定义。

随着数据通信变得越来越复杂并且对错误和延迟的容忍度越来越低，很明显，这些新功能必须合并到 TCP 传输中，以帮助克服由可用的新链接和速度带来的问题。

举个例子，在前面几页中提到并在此处详细说明的窗口缩放可以使用 TCP 选项字段，因为原始窗口字段只有 16 位长，允许的最大十进制数为 65,535。显然，当我们想要使用数千到一百万（例如 400,000 或 950,000）范围内的数字来表示“窗口大小”值时，这太小了。

在我们深入研究任何细节之前，让我们看一下 TCP 选项字段：



可以看到，TCP Options字段是TCP Header分析的第六部分。

它位于标头的末尾和数据部分之前，它使我们能够利用工程师推荐的新增强功能，这些工程师帮助设计了我们今天在数据通信中使用的协议。

TCP选项

我们将要分析的大多数 TCP 选项都需要仅在 3 次握手的初始 SYN 和 SYN/ACK 阶段出现，TCP 执行以在传输任何数据之前建立虚拟链接。然而，在 TCP 会话期间可以随意使用其他选项。

还需要注意的是，TCP Options 可能会占用 TCP 标头末尾的空间，并且长度是 8 位的倍数。这意味着如果我们使用一个长度为 4 位的 TCP 选项，则必须有另外 4 位的填充以符合 TCP RFC。因此 TCP 选项长度必须是 8 位的倍数，即 8、16、24、32 等

以下是我们将要分析的 TCP 选项的简要视图：

- 最大段大小 (MSS)
- 窗口缩放
- 选择性确认 (SACK)
- 时间戳Timestamps
- Nop

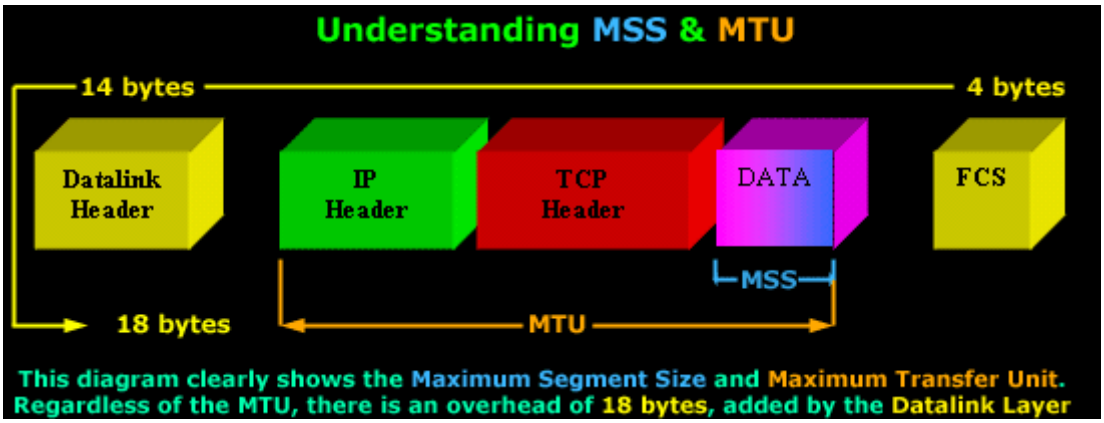
现在让我们看一下可用的令人兴奋的选项并解释每个选项的用途。

最大段大小 (MSS)

最大段大小用于定义在两个主机之间的连接期间将使用的最大段。因此，您应该只在 3 次握手的 SYN 和 SYN/ACK 阶段看到使用此选项。 MSS TCP Option 占用 4 个字节（32 位）的长度。

如果您以前遇到过代表最大传输单元的术语“MTU”，您会很高兴知道 MSS 有助于定义网络上使用的 MTU。

如果您因为 MSS 和 MTU 字段对您没有任何意义而挠头，或者不太清楚，请不要担心，下图将帮助您了解全局：



可以看到Maximum Segment Size由Data段组成，而Maximum Transfer Unit包括TCP Header、MSS和IP Header。

识别与 OSI 模型的每个级别相对应的正确术语也会对我们有所帮助：TCP 报头和数据称为段（第 4 层），而 IP 报头和段称为 IP 数据报（第 3 层）.

此外，无论 MTU 的大小如何，数据链路层都会额外增加 18 字节的开销。此开销包括源和目标 MAC 地址、协议类型，以及位于帧末尾的帧校验序列。

这也是为什么我们最多只能有 1500 字节的 MTU 的原因。由于以太网 II 帧的最大大小为 1518 字节，减去 18 字节（数据链路开销）后，我们还有 1500 字节可供使用。

TCP 通常会计算最大段大小 (MSS)，从而生成与网络 MTU 匹配的 IP 数据报。实际上，这意味着 MSS 将具有这样的值，如果我们也添加 IP 报头，则 IP 数据报（IP 报头+TCP 报头+DATA）将等于网络 MTU。

如果连接的一端或两端省略了 MSS 选项，则将使用 536 字节的值。 536 字节的 MSS 值由 RFC 1122 定义，计算方法是采用 IP 数据报的默认值 576 字节减去 IP 和 TCP 标头的标准长度（40 字节），得到 536 字节。

通常，为您的网络使用尽可能最佳的 MSS 值非常重要，因为如果该值太大或太小，您的网络性能可能会极差。为了帮助您理解原因，让我们看一个简单的示例：

如果您想通过网络传输 1 个字节的数据，则需要创建一个具有 40 个字节开销的数据报，其中 20 个用于 IP 报头，20 个用于 TCP 报头。这意味着您将可用网络带宽的 1/41 用于数据。剩下的只是开销！

另一方面，如果 MSS 非常大，您的 IP 数据报也将非常大，这意味着如果 MTU 太小，它们很可能无法放入一个数据包中。因此，它们将需要被分割，将开销增加 2 倍。

WINDOW SCALING窗口缩放

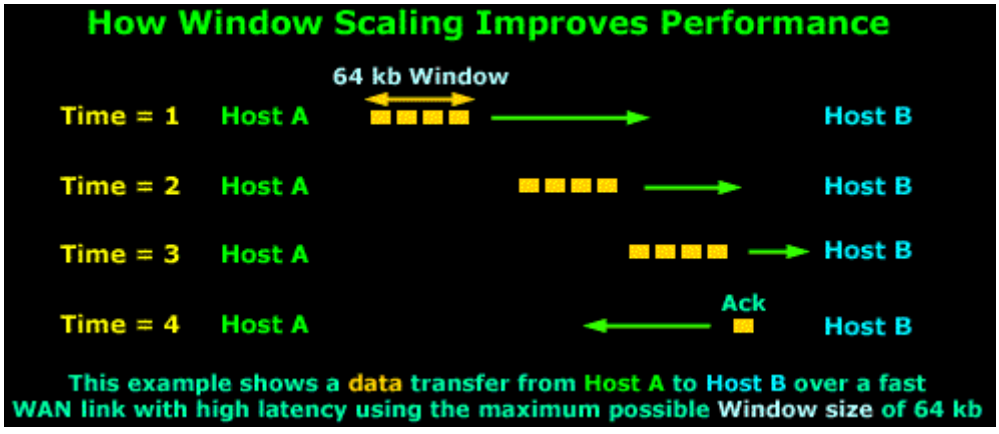
我们在前一节的 TCP 分析中简要提到了 Window Scaling，尽管您很快就会发现这个主题非常广泛并且需要大量关注。

在充分了解窗口大小标志的用途后，窗口缩放本质上是对窗口大小标志的扩展。因为窗口大小标志中的最大可能值仅为 65,535 字节（64 kb），很明显需要更大的字段才能将值增加到惊人的 1 Gig！因此，窗口缩放诞生了。

窗口缩放选项的大小最大为 30 位，其中包括上一节中介绍的原始 16 位窗口大小字段。所以这是 16（原始窗口字段）+ 14（TCP 选项“窗口缩放”）= 总共 30 位。

如果您想知道到底哪里有人会使用这么大的窗口尺寸，请再想一想。窗口缩放是为高延迟、高带宽 WAN 链接创建的，在这些链接中，有限的窗口大小可能会导致严重的性能问题。

为了整合所有这些技术术语和数字，一个例子将被证明是有益的：

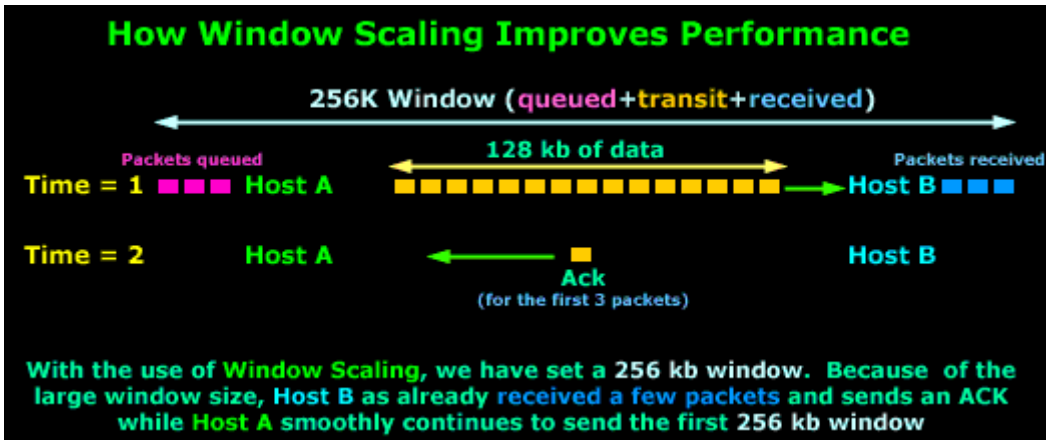


上面的示例假设我们使用 64 kbs 的最大窗口大小，并且由于 WAN 链接具有非常高的延迟，数据包需要一些时间才能到达其目的地，即主机 B。由于高延迟，主机 A 有停止传输数据，因为已发送 64 kbs 的数据，但尚未确认。

当 Time = 4 时，主机 B 已收到数据并向主机 A 发送等待已久的确认，以便它可以继续发送数据，但确认要到 Time = 6 左右才会到达。

因此，从时间 = 1 到时间 = 6，主机 A 一直在等待。你可以想象在这种情况下这个传输的性能会有多差。如果我们要传输一个 10 Mb 的文件，将需要几个小时！

现在让我们考虑使用窗口缩放的同一个例子：



如您所见，使用窗口缩放，窗口大小增加到 256 kb！由于该值相当大，在传输过程中转化为更多数据，主机 B 已经收到前几个数据包，而主机 A 仍在发送第一个 256 kb 窗口。

在时间 = 2 时，主机 B 向仍在忙于发送数据的主机 A 发送确认。主机 A 将在完成 256 kb 窗口之前收到确认，因此将不间断地继续发送数据，因为它很快就会收到来自主机 B 的另一个确认。

显然，大窗口大小所产生的差异是显而易见的，它提高了网络性能并最大限度地减少了发送主机的理想时间。

Window Scale 选项在 RFC 1072 中定义，它允许系统通告 30 位（原始窗口中的 16 + TCP 选项中的 14）窗口大小值，最大缓冲区大小为 1 GB。此选项已在 RFC 1323 中得到澄清和重新定义，这是当今所有实现所采用的规范。

最后，对于与 Cisco 路由器打交道的人来说，了解您还可以在运行 Cisco IOS v9 及更高版本的 Cisco 路由器上配置窗口大小可能会让您受益。此外，12.2(8)T 及以上版本的路由器支持 Window Scaling，窗口大小超过 65,535 字节（64 kb）自动启用，最大值为 1,073,741,823 字节（1 GByte）！

选择性确认（**SACK**

尽管 TCP 被设计成一个相当健壮的协议，尽管如此，它仍然有几个缺点，其中之一涉及确认，这也恰好是 RFC 1072 引入选择性确认的原因。

好的旧的普通确认的问题是没有机制让接收者声明“我仍在等待字节 20 到 25，但已经收到字节 30 到 35”。如果您想知道这是否可能，那么答案是肯定的！

如果段乱序到达并且接收方队列中有一个空洞，则使用 TCP 支持的“经典”确认只能说“我已经收到了直到字节 20 的所有内容”。然后发送方需要认识到出现了问题并从该点开始继续发送（字节 20）。

您可能已经得出结论，上述情况是完全不能接受的，因此必须创建更强大的服务，因此需要选择性确认！

首先，当使用经典的 3 次握手建立虚拟连接时，主机必须在 TCP 选项中发送“允许的选择性确认”以指示它们能够使用 SACK。从此时开始，只要需要选择性确认，就会发送 SACK 选项。

例如，如果我们有一个正在等待字节 4,268 的 Windows98 客户端，但是 SACK 选项显示 Windows98 客户端也收到了字节 7,080 到 8,486，很明显它缺少字节 4,268 到 7,079，因此服务器应该只重新发送丢失的 2,810 字节，而不是在字节数 4,268 处重新开始整个传输。

最后要注意的是，TCP Options中的SACK字段使用了两个16位的字段，一共是32位。有两个字段的原因是因为接收方必须能够指定它接收到的字节范围，就像我们使用的示例一样。在还使用 Window Scaling 的情况下，这些 2 x 16 位字段可以扩展为两个 24 或 32 位字段。

TIMESTAMPS时间戳

TCP 流量控制和可靠性服务的另一个方面是虚拟电路所经历的往返传输时间。往返传送时间将准确地确定 TCP 在尝试重新传输尚未确认的数据段之前将等待多长时间。

因为每个网络都有独特的延迟特性，TCP 必须了解这些特性才能设置准确的确认计时器阈值。 LAN 通常具有非常低的延迟时间，因此 TCP 可以为确认计时器使用较低的值。如果一个段没有被快速确认，发送方可以快速重传有问题的数据，从而最大限度地减少带宽损失和延迟。

另一方面，在 WAN 上使用低阈值肯定会导致问题，因为确认计时器将在数据到达目的地之前到期。

因此，为了让 TCP 准确地设置虚电路的定时器阈值，它必须测量各个段的往返传递时间。最后，它必须在连接的整个生命周期内监视其他网段，以跟上网络的变化。这就是 Timestamp 选项发挥作用的地方。

与此处介绍的大多数其他 TCP 选项类似，必须在 3 次握手期间发送时间戳选项，以便在任何后续段中启用它。

Timestamp 字段由 Timestamp Echo 和 Timestamp Reply 字段组成，这两个字段的回复字段始终由发送方设置为零，并由接收方完成，然后将其发送回原始发送方。两个时间戳字段都是 4 个字节长！

NOP

nop TCP Option 表示“无选项”，用于分隔 TCP Option 字段中使用的不同选项。 nop 字段的实现取决于所使用的操作系统。例如，如果使用选项 MSS 和 SACK，Windows XP 通常会在它们之间放置两个 nop，如本页第一张图片所示。

最后，我们应该注意到 nop 选项占用 1 个字节。在我们页面开头的例子中，它会占用 2 个字节，因为它被使用了两次。您还应该知道，黑客在尝试确定远程主机的操作系统时通常会检查此字段。

概括

此页面提供了所有可用的 TCP 选项，这些选项已被引入 TCP 协议以扩展其可靠性和性能。虽然这些选项在某些情况下很重要，但大多数用户完全不知道它们的存在，尤其是网络管理员。此处提供的信息对于帮助管理员处理无法通过重启服务器或路由器解决的奇怪的本地和 wan 网络问题至关重要:)

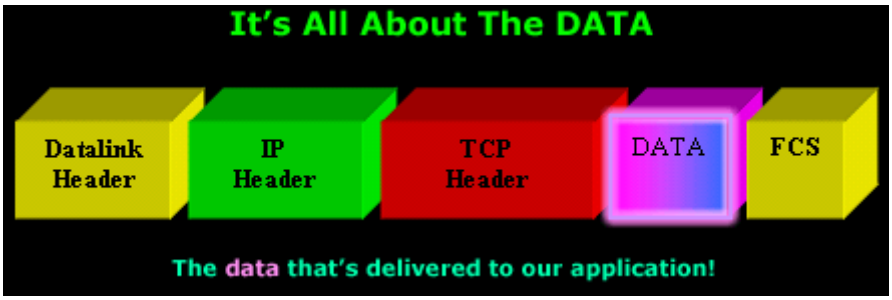
本主题的最后一页是对 TCP 前六页的总结，因为在 TCP 段的数据部分几乎没有要分析的内容。强烈建议您将其作为回顾来阅读，以帮助您记住所涵盖的材料。

4.7 TCP DATA

最后，我们令人难以置信的 TCP 分析的最后一页。正如大多数人所期望的那样，此部分专门用于 TCP 标头之后的数据部分。

数据部分

下图可能很烦人，但是，最后一次显示它以注意数据包的数据部分：



假定您了解上述数据包到达目的地时所遵循的程序。但是，下面给出一个摘要以刷新我们的理解以避免混淆。

当上述数据包到达接收方时，需要进行解封装过程以去除每个 OSI 层的开销并将数据部分传递给等待它的应用程序。因此，当网卡完全接收到数据包时，它会被提供给第二个 OSI 层（数据链路），在对数据包进行错误快速检查后，它将去除与该层相关的开销，这意味着黄色块将被删除。

其余部分，即 IP 标头、TCP 标头和数据，现在称为 IP 数据报，将被传递到第 3 个 OSI 层（网络），在那里将执行另一次检查，如果发现没有错误，IP 标头将被剥离，其余部分（现在称为段）被传递到第 4 个 OSI 层。

TCP 协议（第 4 个 OSI 层）将接受该段并对该段执行自己的错误检查。假设发现它没有错误，则剥离 TCP 标头并将剩余数据提供给上层，最终到达等待它的应用程序。

概括

我们对TCP协议的深入分析已经得出结论。阅读所有这些页面后，我们相信您对 TCP 协议的目的和过程有了更好的了解，并且您能够真正体会到它的功能。

第五章、问题分析

描述：

当您使用 Wireshark捕获分析数据包时，它会对 TCP 连接进行分析，并能够通过 flag 标签了解 TCP 性能的某些行为。其中一些对应于特定的 TCP 消息，而另一些则是 Wireshark 突出显示连接状态。这些标志包括：

```
- TCP Window FullTCP 窗口已满
- TCP ZeroWindowTCP 零窗口
- TCP Retransmission (including variants)TCP 重传
- TCP Dup AckTCP 重复确认
```

问题原因：

TCP 术语:

1. **Receive Window** - 接收窗口 - 也简称为 TCP 窗口，这是由 TCP 接收器向其对等方通告的。它是捕获中可见的窗口，并且在连接中的任一方向可能不同。接收窗口通常是可用接收缓冲区的大小。 TCP 接收者可以通过改变它们发送的 ACK 中的窗口值来调整接收窗口的大小。在负载上，最大大小由接收窗口 TCP 配置文件选项控制。
2. **Send Window** - 发送窗口 - 从技术上讲是拥塞窗口，这是 TCP 发送方的内部值，表示它可以在没有收到确认的情况下发送的最大数据量。这由 TCP 拥塞控制算法不断调整，在 BIG-IP 上，最大大小由发送缓冲区 TCP 配置文件选项控制。在捕获

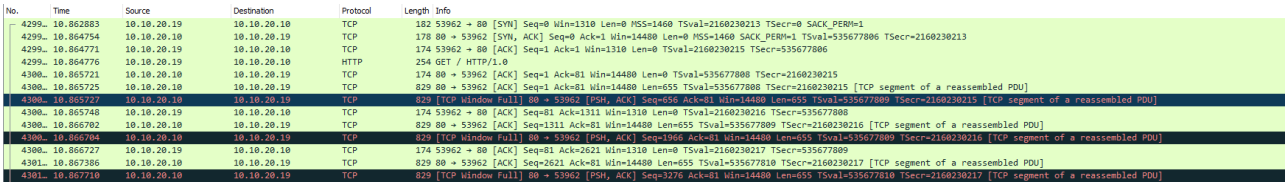
中无法观察到该值；可以推断，如果发送者停止发送数据，即使接收者的接收窗口未满，但实际值从未在网络上发送。

3. **Bytes In Flight** -Bytes In Flight - 这是 Wireshark 用来表示 TCP 发送方已传输的未确认数据量的术语。它总是小于或等于收件人的接收窗口。

TCP Window Full:

如果 TCP 发送方发送的数据包导致未收到确认字节数的报文（或传输中的字节数）与接收方的接收窗口相同，则称为填充窗口，Wireshark 报告 TCP 窗口已满。

Example:客户端配置在 WAN 上，服务器配置在 LAN 上



当您看到 TCP Window Full 标志时,表示您发送的数据等于接收方的窗口大小，相当于正在使用接收方的全部窗口大小。这完全受接收方窗口大小的影响。

TCP Zero Window:

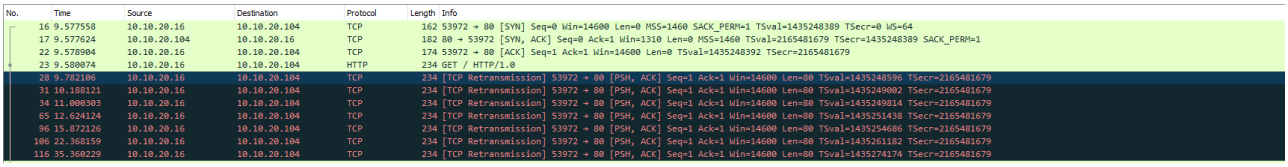
当 TCP 接收器的缓冲区开始填满时，它可以减小其接收窗口。如果它填满，它可以将窗口减小到零，这会告诉 TCP 发送方停止发送。这被称为“关闭窗口”。通常，这表明网络传输流量的速度快于接收器处理它的速度。

当 负载关闭其接收窗口时，通常意味着 负载接收数据的速度快于它在对等流上发送数据的速度。这在某些情况下是正常的，例如:服务器端网络比客户端网络快，并且从服务器到客户端的传输量很大。负载缓冲一定数量的数据，然后在代理缓冲区达到配置的高水位标记时关闭其接收窗口。但是上述场景比较难见到，关闭的也是和服务端的窗口。

每个 TCP 标头中的窗口字段通告接收方可以接受的数据量。如果接收方不能再接受任何数据，它会将窗口值设置为零，这会告诉发送方暂停其传输。在某些特定情况下，这是正常的 — 例如，打印机可能会使用零窗口来暂停打印作业的传输，同时加载或反转一张纸。但是，在大多数情况下，这表明接收端存在性能或容量问题。恢复暂停的连接可能需要很长时间（有时是几分钟），即使导致零窗口的基本条件很快消失。

TCP Retransmission:

当 TCP 发送数据包时，它会等待确认以确认发送方发送的数据包是否已被接收方接收。如果在超时间隔（称为重传超时或 RTO）内未收到确认，则发送方重传数据包。



No.	Time	Source	Destination	Protocol	Length	Info
16	9.577558	10.10.20.16	10.10.20.104	TCP	162	53972 → 80 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=1435248389 TSecr=0 WS=64
17	9.577624	10.10.20.104	10.10.20.16	TCP	182	80 → 53972 [SYN, ACK] Seq=0 Ack=1 Win=1310 Len=0 MSS=1460 TSval=2165481679 TSecr=1435248389 SACK_PERM=1
22	9.578904	10.10.20.16	10.10.20.104	TCP	174	53972 → 80 [ACK] Seq=1 Ack=1 Win=14600 Len=0 TSval=1435248392 TSecr=2165481679
23	9.580074	10.10.20.16	10.10.20.104	HTTP	234	GET / HTTP/1.0
28	9.782106	10.10.20.16	10.10.20.104	TCP	234	[TCP Retransmission] 53972 → 80 [PSH, ACK] Seq=1 Ack=1 Win=14600 Len=80 TSval=1435248596 TSecr=2165481679
31	10.188121	10.10.20.16	10.10.20.104	TCP	234	[TCP Retransmission] 53972 → 80 [PSH, ACK] Seq=1 Ack=1 Win=14600 Len=80 TSval=1435249002 TSecr=2165481679
34	11.000303	10.10.20.16	10.10.20.104	TCP	234	[TCP Retransmission] 53972 → 80 [PSH, ACK] Seq=1 Ack=1 Win=14600 Len=80 TSval=1435249814 TSecr=2165481679
65	12.624124	10.10.20.16	10.10.20.104	TCP	234	[TCP Retransmission] 53972 → 80 [PSH, ACK] Seq=1 Ack=1 Win=14600 Len=80 TSval=1435251438 TSecr=2165481679
96	15.872126	10.10.20.16	10.10.20.104	TCP	234	[TCP Retransmission] 53972 → 80 [PSH, ACK] Seq=1 Ack=1 Win=14600 Len=80 TSval=1435254686 TSecr=2165481679
106	22.368159	10.10.20.16	10.10.20.104	TCP	234	[TCP Retransmission] 53972 → 80 [PSH, ACK] Seq=1 Ack=1 Win=14600 Len=80 TSval=1435261182 TSecr=2165481679
116	35.360229	10.10.20.16	10.10.20.104	TCP	234	[TCP Retransmission] 53972 → 80 [PSH, ACK] Seq=1 Ack=1 Win=14600 Len=80 TSval=1435274174 TSecr=2165481679

TCP sender receives duplicate ACKs.

当 TCP 发送方收到重复的 ACK 时，也可以强制重新传输。DACK 快速重传场景。

Wireshark 区分了几类 TCP 重传；有关详细信息，请参阅 Wireshark TCP 分析文档。

TCP Duplicate ACK

当 TCP 接收器接收到乱序的数据包时，它会将其解释为数据丢失，它会发送一个 ACK 指示预期的序列号。在两个重复的 ACK 之后，TCP 发送方开始重新传输。

优化建议

当您在排除 TCP 性能故障时观察到上述行为时，请使用以下准则。

TCP Window Full

检查负载配置。大多数负载的默认 TCP 配置文件的发送缓冲区和接收窗口设置为 65535，这也将远程端点（即客户端和服务器）的接收窗口限制为 65535，因为它禁用了窗口缩放。如果您认为 负载窗口设置限制了吞吐量，通常可以安全地通过将它们加倍进行试验，直到您不再看到性能改进为止。如果传输到 负载的带宽受到限制，则应增加 Receive Window 设置，如果从 负载传输到另一台设备的带宽受到限制，则应增加 Send Window 设置。对应增加接收和发送的窗口窗口大小。

TCP Zero Window

每个 TCP 标头中的窗口字段通告接收方可以接受的数据量。如果接收方不能再接受任何数据，它会将窗口值设置为零，这会告诉发送方暂停传输。在某些特定情况下，这是正常的——例如，打印机可能会使用零窗口来暂停打印作业的传输，同时加载或反转一张纸。但是，在大多数情况下，这表明接收端存在性能或容量问题。恢复暂停的连接可能需要很长时间（有时需要几分钟），即使导致零窗口的潜在条件很快就会消失。

TCP Dup Ack and TCP Retransmissions

这些行为通常是相关的。这些通常表示网络存在丢包行为。损失可能是问题，也可能不是问题； TCP 旨在增加吞吐量，直到观察到丢失数据报的情况以估计重新估算网络容量，因此如果您偶尔看到 duplicate ACKs 和retransmissions但是网络具有稳定吞吐量，您可能是正常的 TCP 行为。

Additional Information附加信息

如果您了解端到端网络的特征，则可以使用带宽延迟乘积 (BDP) 估算所需的发送窗口和接收缓冲区设置（即所需的 TCP 窗口）。这是带宽（通常以每秒位数衡量）和客户端往返时间（RTT 或延迟）的乘积，以字节表示。请注意，RTT 通常以毫秒表示，带宽通常以每秒千位、兆位或千兆位表示，因此有必要相应地调整单位。

例如，对于可以承载每秒 40 兆比特 (Mbit/s) 和 50 毫秒 RTT 的网络，BDP 为：

```
    40,000,000 bit/s * 0.05 s = 2,000,000 bit ÷ 8 bit/byte =
250,000 bytes
```

为了占满这条链路，也就是在理想条件下使带宽完全饱和，TCP 窗口必须至少为 250000 字节。

Related Content相关内容

K1893: Packet trace analysis

K56401167: Standard virtual servers do not use TCP Window Scaling with certain profile settings

K56401167：标准虚拟服务器不使用具有某些配置文件设置的 TCP 窗口缩放

注意；在设置发送和接收窗口大于65535 时，会自动启动 TCP window scaling

Wireshark manual chapter 7.5: TCP AnalysisWireshark 手册第 7.5 章：TCP 分析

作者:郑岳 日期:2022年