两万字长文50+张趣图带你领悟网络编程的内功心法

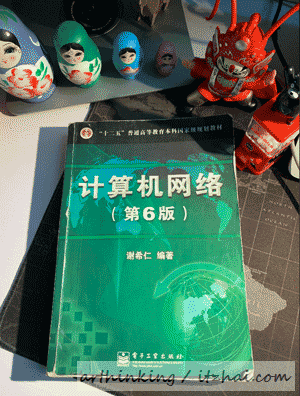
**前言**

我大学是学网络工程专业，也就是那种拉网线，面向网线编程的。依稀记得学习计算机网络这门课程的时候搭建的IT宅 itzhai.com个人网站。

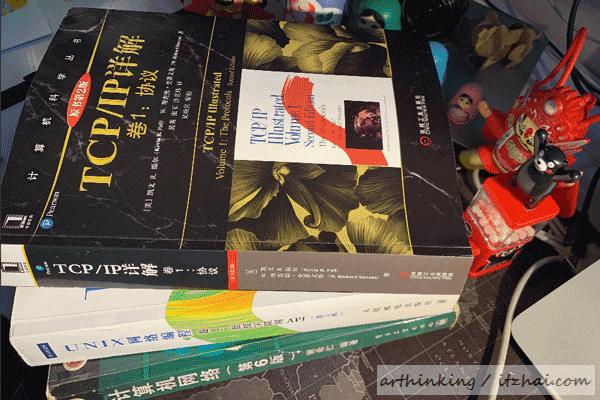
算一下，学这门课程也已经快十年了。

image-20200726210546358

某一天，偶然又看到了这本书：



翻了下，发现里面的内容竟然还是毫不过时，真的是越底层的知识越有价值呀。**我擦了擦书面的灰尘，决定要为它写点什么**，于是又从书架上找了相关的书籍：



来回翻阅和梳理总结，逐渐输出了这篇文章，献给对网络不太熟悉，又想快速从入门到熟练的朋友们。

相信大家拿到Socket API，就可以很快写好代码，收发消息，传送文件什么的，可是底层究竟发生了什么？TCP、UDP、HTTP是什么关系、为啥要有WebSocket编程。我们从TCP/IP协议栈以及一根网线说起，逐步揭开面向网线编程内功心法的面纱。

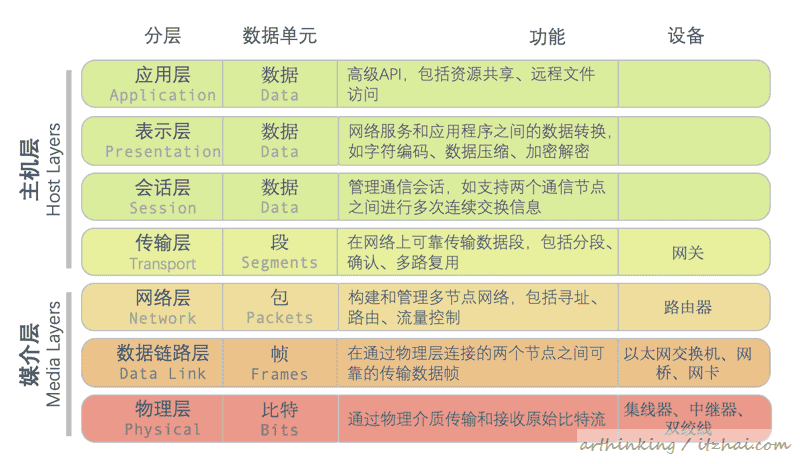
最后，在这里解答一个问题：有人问我为什么要写公众号技术文章呢？工作越久，发现身边比自己年纪小的人越多，我也时常在想，那些同龄人或者比我大的人都去哪里了，也许有些人忙于家庭生活不亦乐乎，有些人因为公司上市拿到可观的收入转行了，也许有人在大公司做起了管理工作，开始走管理路线，带领团队创造新的产品。我写公众号的原因之一，也就是想告诉大家，我一直在做技术，一个坚持写代码的大龄技术人，并且希望能够结实更多志同道合的技术人。没错，在说你们呢，不要求三连，这篇文章对你感兴趣就点个在看呗。Thanks♪(･ω･)ﾉ

(本文的各种电脑、服务器、路由器小图标都是我一笔一划亲手设计创建，旨在希望能够有助于大家理解文章内容)

**1、网络分层**

**1.1、OSI七层模型**

为了制定一个统一的计算机网络体系，国际标准化组织ISO提出了一个试图使各种计算机可以在世界范围内互联成网的标准框架：OSI/RM(Open System Interconnection Reference Model 开放系统互连基本参考模型)，该模型如下：



**媒介层：**第一到第三层称为媒体层，它们主要与硬件相关，例如路由，交换和电缆规格；

**主机层：**第四到第七层称为主机层，它们是实现网络服务相关的软件。

大致介绍一下各层（**注意：看看就好，这不是重点，重点是后面的TCP/IP协议**）：

* 物理层：物理层负责在设备和物理传输介质直接传输和接收非结构化原始数据。这一层中，把数字位转换为电，无线电或光信号。可以发现这一层往往跟各种材质啊信号呀什么的打交道，所以称为物理层；
* 数据链路层：在通过物理层连接的两个节点之间进行传输数据帧，检测并且纠正物理层中可能发生的错误。它定义了在两个物理连接的设备之间建立和终止连接的协议，还定义了他们之间的流控制协议；
* 网络层：构建和管理多节点网络，包括寻址、路由、流量控制。网络层是一种可以连接许多节点的介质，每个节点都在其上有一个地址，通过目标地址就可以在节点之间传输数据到目标地址。网络层消息传输不一定要保证可靠，网络层系可以可以提供可靠的消息传递，但不必这样做；
* 传输层：传输层提供了将可变长度数据序列从源传输到目标主机的功能和过程方法，同时又保持了服务功能的质量。一些协议是面向状态和面向连接的，这意味着传输层可以进行分段传输、支持失败重传；
* 会话层：控制计算机之间的连接，负责建立、管理和终止本地和远程引用程序之间的连接，提供全双工、半双工或者单工操作；
* 表示层：网络服务和应用程序之间的数据转换，如字符编码、数据压缩、加密解密；
* 应用层：最接近最终用户的OSI层，该层直接与实现通信组件的软件应用程序进行交互。

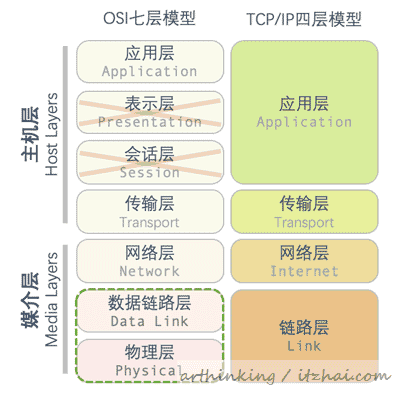
可以发现，这个模型还真有点复杂。但很可惜，这个模型似乎不怎么流行，原因如下：

* OSI专家缺乏实际经验，缺少商业驱动；
* OSI协议实现过于复杂，运行效率低；
* 标准制定周期长，市场已被其他标准占据；
* 层次划分不太合理，部分功能在多个分层中出现。

在1980年代末和1990年代初的一段时间内，工程师、组织和国家对哪种标准（OSI模型或[Internet协议套件](https://en.wikipedia.org/wiki/Internet_protocol_suite" \t "_blank)）将更能塑造最佳和最强大的计算机网络存在争端，导致两极分化。尽管OSI在1980年代后期开发了其网络标准，后来更多的供应商网络上更多采用的却是TCP / IP标准，最终TCP/IP成为了实时的国际标准。

**1.2、TCP/IP四层模型**

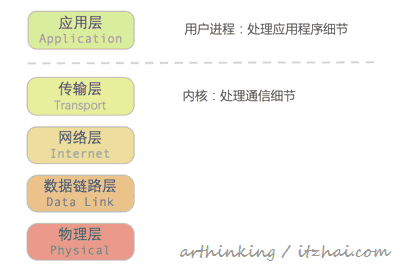
下面我们把TCP/IP模型和OSI模型放一起对比下：



可以发现，TCP/IP体系少了表示层和会话层，数据链路层和物理层用链路层取代。

* 应用层：最高层，应用层的任务是通过应用进程间交互来实现特定网络应用。主要负责把应用程序中的用户数据传达给另一台主机或同一主机上的其他应用程序。这是所有应用程序协议的运行层，如SMTP、FTP、SSH、HTTP等；
* 传输层：负责向两个主机中的进程之间的通信提供通用的数据传输服务。UDP是基本的传输层协议，提供了不可靠的无连接数据报传输服务；
* 网络层：负责为分组交换网上的不同主机提供通信服务。该层定义了寻址和路由功能，主要协议是IP协议(Internet Protocol)，它定义了IP地址，它在路由中的功能是将数据报传输到充当IP路由器的下一个主机，该主机更接近最终数据目的地；
* 链路层：也称为数据链路层或者网络接口层，通常包括操作系统中设备驱动程序和计算机对应的网络接口卡。它们负责处理与传输媒介的物理接口细节。

而后面我们除了讲到各种协议之外，还会顺便提及一些底层硬件，为了更好的进行阐述，我们将把链路层再分为物理层和数据链路层，采用以下这种五层模型：



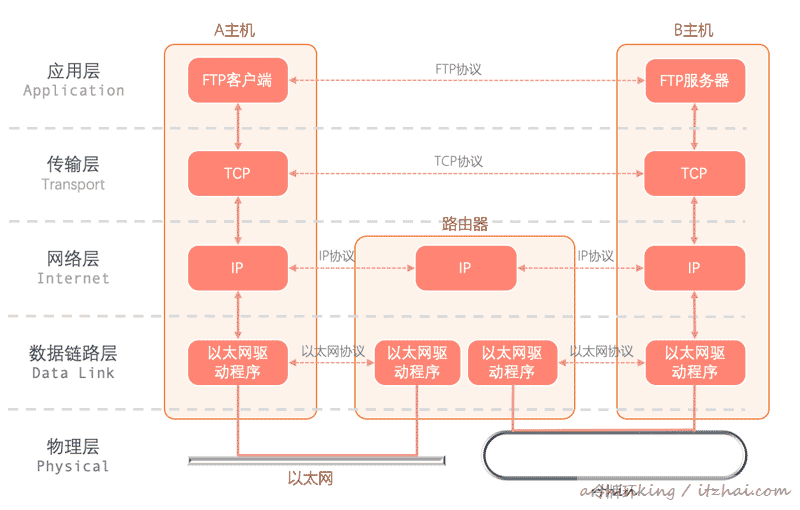
根据运行模式分为以下两种：

* 运行于用户进程：应用层，关注应用程序的细节，不关注底层网络通信细节；
* 运行于内核：传输层、网络层、链路层，在内核中执行，主要处理所有的通信细节。

介绍了这么多概念，是不是比较难懂呢，没关系，我们列一下每一层主要的协议，接下来我们会详细的讲解各种协议的原理。

**1.3、分层模型如何工作**

这里我们通过一个FTP客户端的通信流程，来说明下两台主机是如何工作在TCP/IP分层模型上的：



如上图，A主机的FTP客户端要与B主机的FTP服务器进行交互。

我们对设备做一下划分：

* 端系统（End system）：A主机和B主机
* 中间系统（Intermediate system）：路由器

其中用到的协议也做一下划分：

* 端对端协议（End-to-end）：包括应用层和传输层，端系统直接直接进行交互；
* 逐跳（Hop-by-hop）协议：网络层，需要经过端系统中所有的中间系统。

对于应用层来说，他们好像是直接与端系统进行交互的，应用层根本不知道底层通信用了多少个路由器，是在以太网上还是在令牌环网上的。

**什么是三层设备，二层设备？**

如上图，路由器工作在网络层，属于第三层，所以经常有人称他为三层设备；而后面我们会讲到交换机，他是工作在第二层-数据链路层，所以也成为二层设备。

***1.3.1、为什么要分层？***

正如上面的例子，分层之后是的顶层屏蔽了底层的物理和通信细节。底层的通信原理是在较低的协议层中实现的，因此每个进程都将隐藏大多数通信细节。以此类推，在传输层，通信表现为主机到主机，而无需了解应用程序数据结构和连接的路由器。而在互联网络层，则在每个路由器上遍历各个网络边界。

就像我们搞软件开发划分层次一样，**分层之后，提高了软件的复用度，封装每层细节，使用者只需要关注使用的API就可以了，不用关注实现细节。**你不要告诉我你的一个功能涉及的一万行代码是写在一个函数里面的，那太可怕了。

想要了解底层细节的人，就只能拆开TCP/IP协议潘多拉之盒，逐个协议去了解了，这也是本文后边会继续探讨的内容。高度的封装，使得顶层开发人员能更快速的通过API开发应用程序。**作为一个API工程师，你知道怎么调用API发送HTTP请求就够了，但是作为一个有追求的工程师，你了解了这些细节之后，就能够胜任程序调优以及更加底层的开发工作了。**这也是为什么我坚持写[IT宅 itzhai.com](https://www.itzhai.com/)博客的原因：我想探索技术的本质，而不是生活在API构造的童话世界里面，这样即使童话世界谎言被拆穿的那天，也不至于失掉技术的信仰，因为我仍然有能力构建一个新的童话世界。





接下来我们看看数据包从在传输过程中是如何封装的。

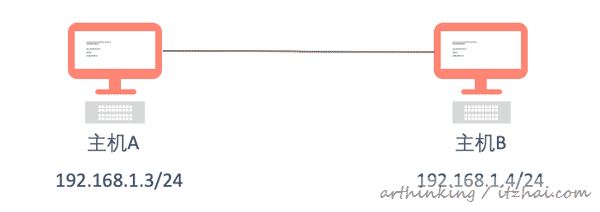
***1.3.2、数据包的封装和分用***

为了演示，我们需要构建一个局域网（LAN）。

假设我们现在直接通过两个网线把两台电脑连起来进行通信，需要做哪些工作呢。多亏我大学学的是网络工程，也是拉过网线的，所以多少还知道一点：

* 准备一根网线，两个水晶头；
* 水晶头要做交叉线，采用1-3，2-6交叉接法，保证两个水晶头之间能够正常收发信号；
* 把两个水晶头分别插在两台主机的电脑上；
* 给两台电脑配置IP、子网掩码和网关，必须要配置到同一个网络中。

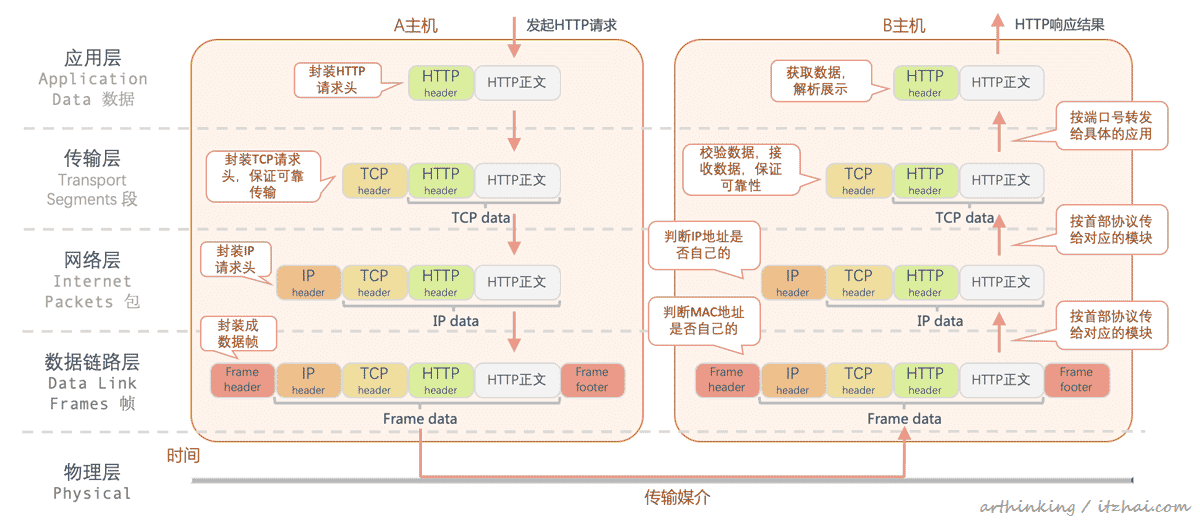
这样我们就构建好了一个最简单的局域网了。



***1.3.2.1、封装***

所谓封装，就是每一层都会根据用到的协议，把数据封装成最终的一个数据单元(不同分层有不同的叫法，参考上图最左边每层的描述)，每一层拿到的上一层的内容，把上一层封装好的内容作为当前层的数据，然后加上自己的协议头或者尾，接着执行该层协议的相关处理逻辑。

有没有发现，**这有点像装饰着模式，每一层拿到上一层的内容之后都添加额外的处理逻辑，但是不改变上层传过来的内容。**

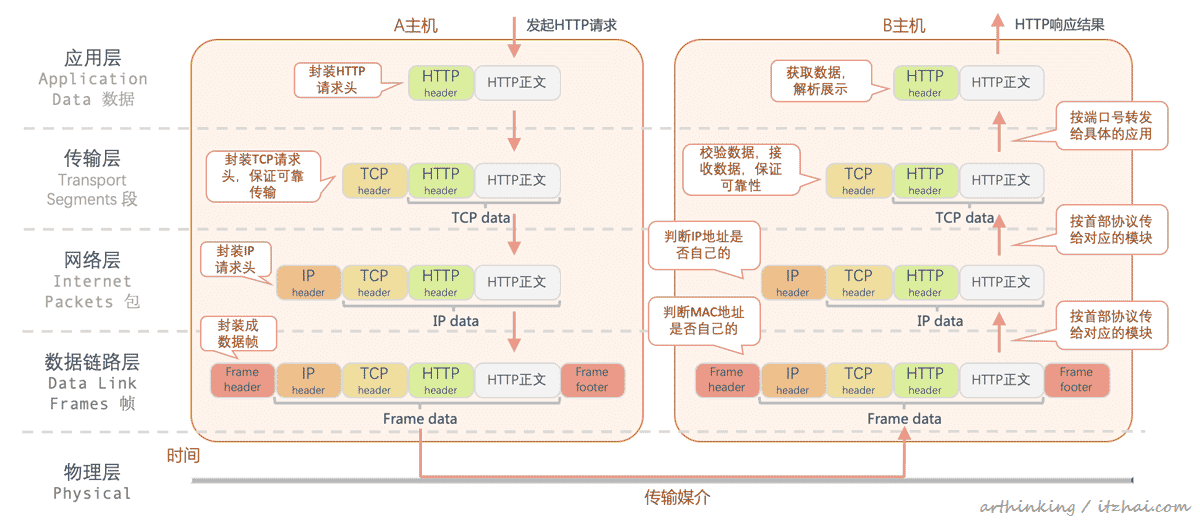


如上图，左边部分为封装的过程：

* A主机请求B主机的HTTP服务，应用层使用HTTP协议对请求内容进行封装，加上HTTP请求头，然后传给下一层；
* 传输层拿到HTTP数据，使用TCP协议进行处理，加上自己的TCP头，封装成数据段，通过TCP协议传输给下一层。这一层通过TCP协议保证了可靠的传输；
* 网络层拿到TCP传输段之后，使用IP协议进行处理，加上IP头，封装成包进一步传给下一层。这个IP决定了什么路由或者主机需要接收处理这个包；
* 数据链路层拿到网络层的包之后，进一步封装成数据帧，最终通过数据链路层进行发送处理，最终数据帧通过物理层传输给接收端。

***1.3.2.2、分用***

所谓分用，指的是主机或者中间设备接收到一个物理层传输过来的数据帧时，数据开始从协议栈中由底向上升，逐层处理，每层去掉对应的协议的报文首部。每层协议盒都要检查报文首部的协议标识进行对应的协议处理。



在上图中，右边部分为分用的过程：

* 主机B接收到物理层传过来的数据帧之后，首先从首部找到MAC地址，判断是否发送给自己的，如果不是则进行丢弃；
* 如果发送包是自己的，则从数据帧确定数据协议类型，再传给对应的协议模块，如IP、ARP等；
* IP模块接收到数据后获取IP首部，判断首部接收的IPIP地址匹配，如果匹配则根据首部协议类型转发给对应的模块，如TCP、UDP等；
* 传到TCP模块之后，首先TCP模块会计算校验和，判断数据的完整性，然后处理数据包的顺序接收相关逻辑；最后检查端口号，确定具体应该要转发给应用层的哪个应用程序。
* 应用层接收到数据之后，解析数据进行展示，这里是HTTP数据包，所以按照HTTP协议的约定进行解析展示。

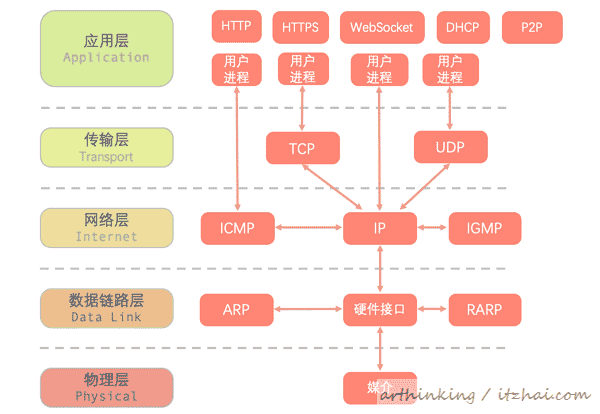
可以发现，以上流程中，大家感知最深刻的就是传输层的TCP或者UDP协议，以及应用层的HTTP协议了，因为做网站开发，或者网络通信编程经常会用到它们的API。

上面介绍的并不是很详细，不过没关系，后面我们会把主要的协议都拿出来详细的讲解。

当然，**正常的网站请求中，中间肯定会涉及到很多路由器，交换机，光纤等底层的物理设备，中间会产生很多的逐跳（Hop-by-hop），每个中间系统都会对数据帧进行分用和封装的过程。**

**1.4、TCP/IP协议簇**

TCP/IP协议簇内容非常多，这里列出的是本文可能会介绍到的相关协议，以及他们之间的交互关系：



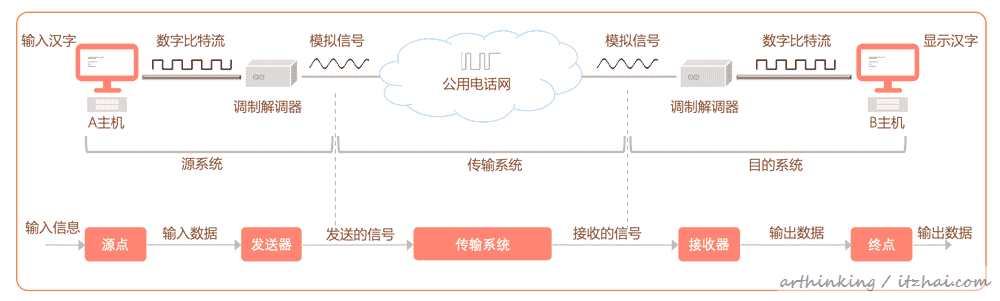
**2、物理层**

我们的数据帧究竟是怎么传给不同的主机呢。前面我们了解到每一个上层都依赖于下层的API，而物理层是最底层的了，它是真的要把数据传出去了。而数据最终都会变为0和1，物理层依赖于各种不同硬件技术，通过网络的电子传输技术，把0和1在传输介质中进行传输。

**2.1、通信系统的模型**

下我我们举一个最简单的例子来说明通信系统的模型[1](https://www.itzhai.com/network/comprehend-the-underlying-principles-of-network-programming.html" \l "fn:1)。

很久以前，有些同学家里都是用的电话线进行上网的，这种网络传输模型类似如下这样：



如上图，主要包括源系统，传输系统，目的系统，可以抽象为下半部分的模型：

* 源点：源点产生要传输的数据；
* 发送器：源点产生的数据经过发送器编码之后进行传输；
* 传输系统：传输系统可能是简单的传输线，也可能是复杂的网络系统；
* 接收器：接收传输系统的信号，转换为能够被目的设备处理的信息；
* 终点：从接收器获取传送过来的数字比特流，最终输出信息。

**2.2、物理层解决什么**

传输媒介的种类非常多：双绞线、对称电缆、同轴电缆、光缆、无线信道等，导致物理层的协议种类较多。

物理层的主要作用是屏蔽掉这些传输媒介和通信手段的差异，使物理层上面的数据链路层感觉不到这些差异。为此，**物理层需要处理以下事情**：

* 规定接口所用接线器的形状和尺寸，引脚数目和排列，固定和锁定装置等；
* 规定接口电缆各条线上的电压范围；
* 规定某一电平电压的意义；
* 规定不同功能的各种可能事件出现顺序。

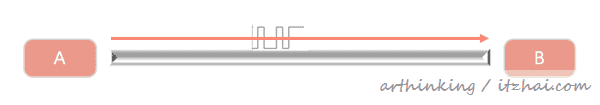
**2.3、物理层也出面试题？**

最后我列几个物理层常见的面试题，一般的开发人员都是工作在传输层以上，所以考一些TCP，UDP，HTTP，HTTPS等协议我觉得更贴近开发人员真实的工作场景。当然，如果是通信领域的工程师，物理层都是家常便饭，这些可是通信的基础知识。即使知识应用开发工程师，了解这些也不会吃亏，说不定哪天亲戚还需要叫你帮忙拉网线呢。

下面是几个常见的物理层面试题：

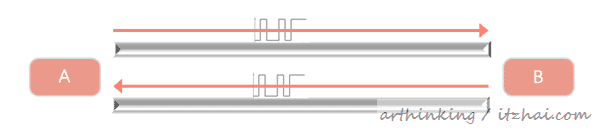
***有哪些通信交互方式？单工、半双工通信、全双工通信？***

**单工通信**，又称为单向通信，只有一个方向的通信，如无线电广播，电视广播；



**半双工通信**，又称为双向交替通信，双方都可以收发信息，只能交替进行；

**全双工通信**，又称为双向同时通信，双方可以同时发送和接收数据。



***为了提高信道利用率，有哪些信道复用技术？***

所谓信道复用技术，指的是大家共享一个信道进行通信，在接收端在使用分用器，把合起来传输的信息分别送到相应的终点；

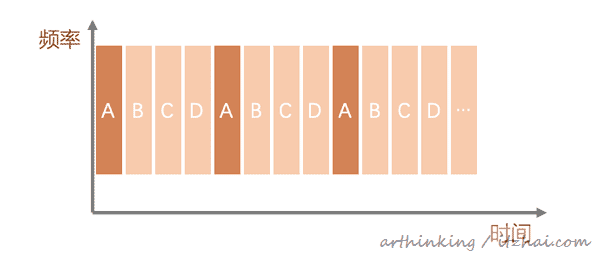
**频分复用**

用户在分配到一定的频带后，通信过程中使用都占用这个频带；



**时分复用**

将时间划分为一段段等长时分复用帧，每一个时分复用的用户周期性的占用帧位；

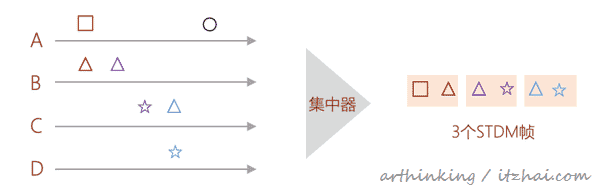


**统计时分复用**

时分复用，如果用户没有任何数据要传输，也会周期性的给他分配时隙，这就导致了信道利用率不高。

为此出现了统计时分复用。

统计时分复用使用STDM帧来传送复用的数据，把所有用户数据按时间顺序组成STDM帧，放入一个队列中，依次发送出去，这样就能够更合理的共享信道。STDM帧中的数据需要添加用户地址首部信息，以便能够正确的分发给目标用户：



这里的集中器也叫智能复用器。

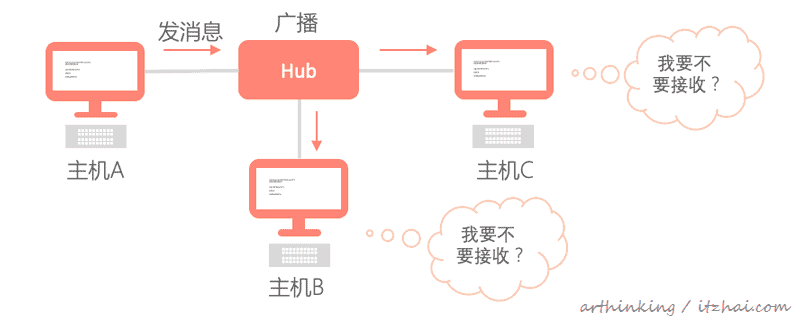
除了以上三种，还有波分复用和码分复用，感兴趣的朋友可以自行搜寻资料了解，这里就不继续展开来讲了。

***物理层要解决什么问题？***

这个问题上一小节已经回答了。

**2.4、物理层设备之集线器**

如果我们只是想用几台电脑搭建一个局域网，那么可以通过集线器(Hub)进行搭建，这个硬件工作在物理层，会把自己收到的字节都复制到其他端口，如下图：



如上图，其中一台电脑发送信息之后，Hub以广播的方式发给其他三台机器，但是究竟哪台电脑才会把消息接收下来呢？这里我们就要讲到数据链路层了，在这一层判断数据包是不是自己的。

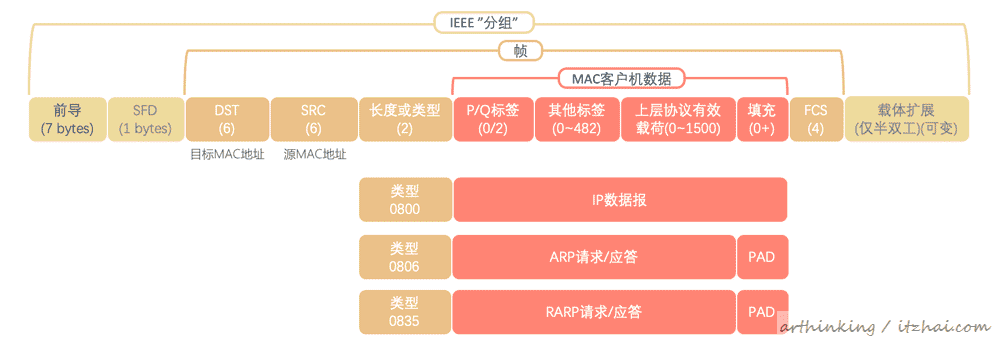
**3、数据链路层**

**3.1、数据帧格式**

我们首先来看看数据链路层的传输数据帧的格式。

所有的以太网(802.3)帧都基于一个共同的格式。在原有规范的基础上，帧格式已被改进以支持额外功能。

当前以太网的帧格式[2](https://www.itzhai.com/network/comprehend-the-underlying-principles-of-network-programming.html" \l "fn:2)如下：



前导：用在发送方和接收方之间同步时钟和bit流；

SFD：帧开始界定符，只有一个byte，内容固定为：10101011 (0xAB)；

DST：目标MAC地址；

SRC：源MAC地址；

长度或类型：0800时，表示IP数据报，0806表示ARP请求/应答，0835表示RARP请求/应答；

FCS：帧检验序列，用于数据帧的差错检测；

***这个包应该发给谁？***

**判断是否应该接受这个包，就是通过帧的MAC地址进行判断的。**

这是一个物理地址，叫做链路层地址，因为链路层主要解决媒体接入控制问题，所以称为**MAC地址**(Media Access Control Address)。实际上，MAC地址就是适配器地址或适配器标识符，当适配器插入到某台计算机之后，适配器上的标识符就成为这台计算机的MAC地址了。

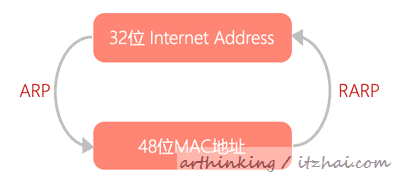
***怎么校验包是否出现错误***

FCS是帧校验序列，也就是循环冗余检测，收到数据报之后，会通过一个检验计算规则，把计算结果与FCS字段匹配，如果匹配补上，则帧可能在传输过程中受损，通常会丢弃该帧。

**3.2、ARP: 如何获取目标机器的MAC地址？**

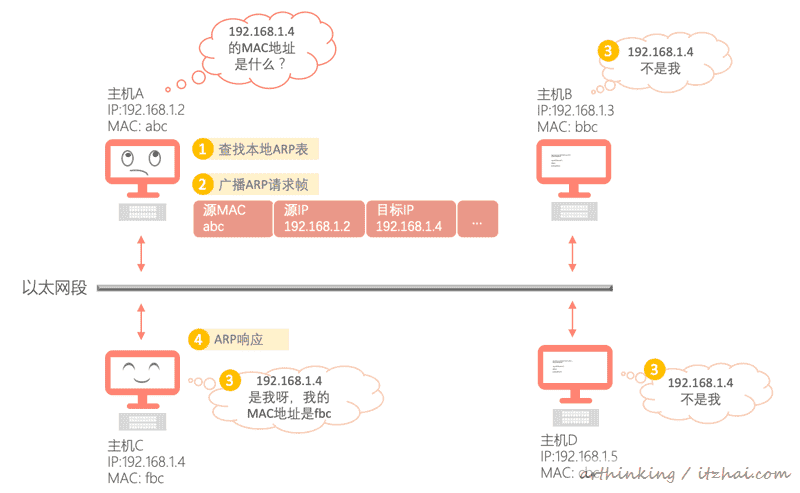
我们知道，在数据链路层，是通过MAC地址判断某一个接收到的包是不是要进一步处理的。但是如果我们不知道对方的MAC地址的时候，如何发送数据链路层的帧呢？这就需要用到**数据链路层的ARP协议**了。

ARP协议：ARP为IP地址到硬件地址之间提供了动态映射，我们通过ARP可以把32位的Internet地址转换为48位的MAC地址。另外，我们可以使用RARP，把48位的MAC地址转换为32位的Internet地址。



另外，为了保证ARP的高效运行，ARP会维护每个主机和路由器上的ARP缓存，把Internet地址和MAC地址的映射关系保存起来，缓存正常到期时间是20分钟。

下面是这个过程的演示，其中ARP数据帧只把关键信息描述出来了，想要了解完整的帧格式可以用参考 TCP/IP协议详解卷1[3](https://www.itzhai.com/network/comprehend-the-underlying-principles-of-network-programming.html#fn:3)



如上图：

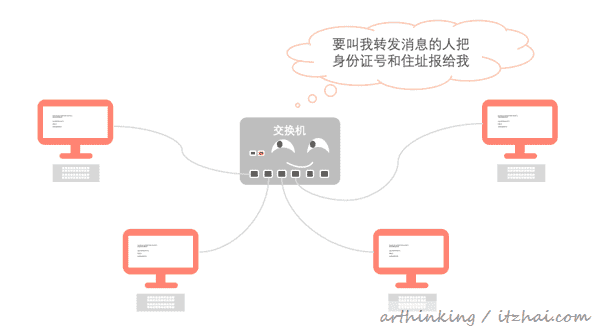
主机A想知道192.168.1.4这个IP地址的MAC地址是什么，发现本地缓存中找不到，于是广播了一个ARP请求，主机B和主机D收到之后，发现自己不是192.168.1.4于是忽略这个消息，主机C发现自己就是192.168.1.4，于是响应了一个ARP数据帧。最终主机A收到主机C响应的数据帧，拿到了MAC地址，并把IP地址和MAC地址映射关系保存下来。

**3.3、链路层设备之交换机**

***3.3.1、为什么需要交换机？***

前面我们用了集线器组件网络，这个时候所有消息都会广播到其他端口，可以发现集线器转发了很多不必要的消息，能不能只发给需要的端口呢？这个时候就需要用到交换机了。

当一台电脑A向交换机发送数据时，交换机会把电脑A的IP和MAC地址记住，保存到一个转发表中，如果转发表中暂时找不到目标IP地址的MAC地址，那么首先还是会广播消息，最终转发表会记录所有请求过交换机的电脑IP和MAC。当然，转发表也是有过期时间的。



如上图，看到交换机的奸笑没有，与集线器不同，交换机是有灵魂的的，你告诉他你的身份证号和住址了，他就会偷偷记下来。

***3.3.2、为什么有了IP地址，还需要有MAC地址？***

IP地址是工作在网络层的，后面会讲到；

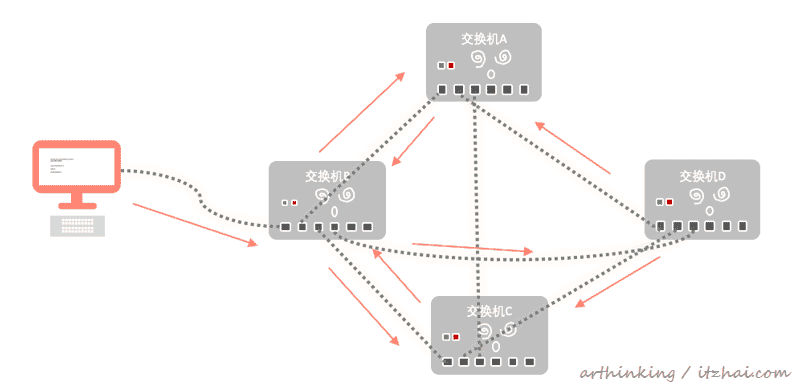
MAC地址是工作在数据链路层的，也就是交换机这一层，交换机之间的主机进行通信，都是用的MAC地址，但是一旦走出了局域网，我们就得用大家都公认的IP地址了。

MAC地址就好像是我们的身份证，IP就像是我们的住址，可以根据住址寄送快递，但是不能根据身份证号码寄快递，别人不知道怎么走呢。

一个局域网，用身份证没有问题呀，因为要找某个人，ARP会大喊一声名字，那个人就会告诉你他的身份证号码了，这个时候直接以身份证作为标识传消息，别人听到不是自己的身份证就不管了。最终交换机这个小管家记住了所有人的名字跟身份证号码，就会使用悄悄话的方式传达消息了。这也就是用到了交换机的转发表。

***3.3.3、交换机拓扑环路问题***

假设我现在拉网线，搞了一个这样的拓扑结构：



如上图，主机准备发送一个消息出去，结果交换机B收到后，复制数据帧，发送给了交换机A、C、D，此时交换机B认为主机是在左边。但是不妙的事情发生了，交换机D收到消息后，由于转发表还是空的，又是也复制数据帧，转发到了交换机A、B、C，这个时候交换机B发现怎么主机的数据又从右边传过来了，这些彻底晕了，不知道主机究竟在哪里。就这样数据一致在这个网络里面打转，这就拓扑环路导致的问题。

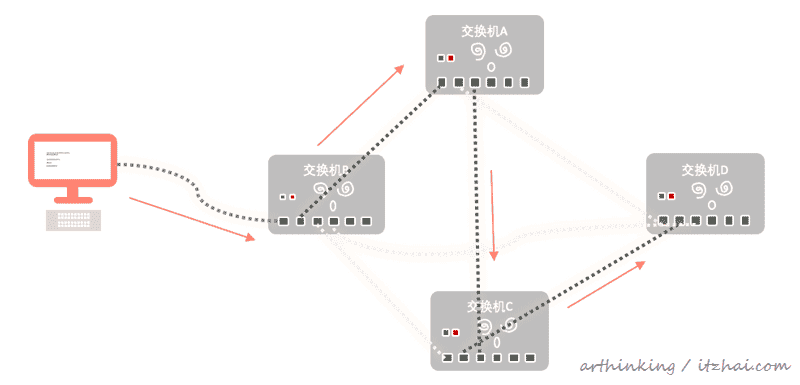
***生成树协议***

为了解决以上问题，于是 有了生成树协议（Spanning Tree Protocol，STP）。

STP通过在每个交换机禁用某些端口工作，来避免拓扑环路，保证不会出现重复路径。

STP会找到拓扑结构的一个生成树，通过生成树避免环路。生成树的形成和维护有多个网桥完成，在每个网桥上运行一个分布式算法。

以上拓扑，结构，最终应用了生成树，禁用一些端口之后，可能会是这样：



这样，消息就不可能再传回左边的主机了，从而避免了拓扑环路导致的问题。

**建立生成树：**

网桥会发送一种称为网桥协议数据单元（BPDU）的帧来辅助形成和维护生成树。

STP首先会尝试选举根网桥，根网桥是在网络中标识符最小的网桥(也就是说优先级与MAC地址结合)，网桥初始化的时候，假设自己是最小的网桥，然后用自己的网桥ID作为根ID字段的值发送配置BPDU消息，如果发现ID更小的网桥，那么会停止发送自己的帧，并基于接收到的ID更小的帧构造下一步发送的BPDU消息。发出根ID更小的BPDU端口被标记为根端口，剩余端口被设置为阻塞或者转发状态。

**3.4、VLAN**

**3、网络层**

网络层，Internet layer，最熟知的就是IP协议了（Internet Protocol）。

前面我们将的数据链路层，其实只能在局域网内进行通信，因为都是通过MAC地址进行传达信息的，要想跨局域网，那么就得用到IP地址了，这就是网络层要做的事情了。

首先我们来介绍下网络的一个协议：ICMP协议。

**3.1、ICMP协议**

IP协议本身不支持发现发往目的地地址失败的IP数据包，也没有提供直接的方式获取诊断信息，比如在发送途中，经过了哪些路由器，以及往返时间。

为此，就有了ICMP协议（Internet Control Message Protocol，ICMP）专门来负责这些事情。

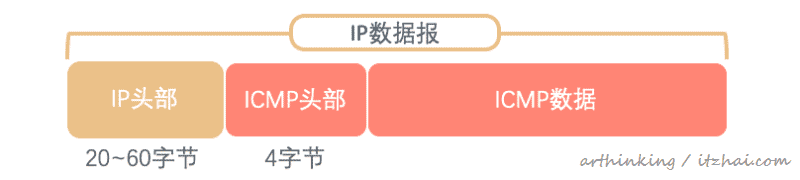
**ICMP并不为IP网络提供可靠性，它只是用于反馈各种故障和配置信息。丢包不会触发ICMP。**

ICMP是[RFC 792中](https://tools.ietf.org/html/rfc792" \t "_blank)定义的Internet协议套件的一部分。ICMP消息通常用于诊断网络或探测网络目的，或者是为了响应[IP](https://en.wikipedia.org/wiki/Internet_Protocol" \t "_blank)操作中的错误而生成（如[RFC 1122中](https://tools.ietf.org/html/rfc1122" \t "_blank)所指定），ICMP错误响应给原始数据包的源IP地址。

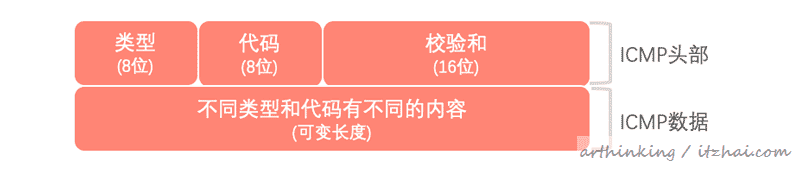
但是黑客经常用ICMP来做坏事，于是网络管理员可能会用防火墙阻止掉ICMP报文，这样的话，很多ping、traceroute之类的诊断程序就无法正常工作了。

***3.1.1、格式***

ICMP报文是在IP数据报内部传输的，格式如下：



而ICMP报文的格式如下：



其中：

* 类型有15个不同的值，描述特定类型的ICMP报文；
* 某些ICMP报文还是用代码字段的值来进一步描述不同的条件；
* 校验和字段用于ICMP报文的差错检查。

以下是常见的差错报文类型：

| **类型** | **代码** | **描述** | **查询** | **差错** |
| --- | --- | --- | --- | --- |
| 0 | 0 | 回显应答(ping应答) | √ |  |
| 3 |  | 目标不可达 |  |  |
|  | 0 | 网络不可达 |  | √ |
|  | 1 | 主机不可达 |  | √ |
|  | 2 | 协议不可达 |  | √ |
|  | 3 | 端口不可达 |  | √ |
|  | 4 | 需要进行分片但设置了不分片比特 |  | √ |
|  | 5 | 源站选路失败 |  | √ |
|  | 6 | 目的网络不认识 |  | √ |
|  | 7 | 目的主机不认识 |  | √ |
|  | 8 | 源主机被隔离(作废不用) |  | √ |
|  | 9 | 目的网络被强制禁止 |  | √ |
|  | 10 | 目的主机被强制禁止 |  | √ |
|  | 11 | 由于服务类型 T O S ， 网 络 不 可 达 |  | √ |
|  | 12 | 由于服务类型 T O S ， 主 机 不 可 达 |  | √ |
|  | 13 | 由于过滤，通信被强制禁止 |  | √ |
|  | 14 | 主机越权 |  | √ |
|  | 15 | 优先权中止生效 |  | √ |
| 4 | 0 | 源端抑制 |  | √ |
| 5 |  | 重定向 |  | √ |
|  | 0 | 对网络重定向 |  | √ |
|  | 1 | 对网络重定向 |  | √ |
|  | 2 | 对服务类型和网络重定向 |  | √ |
|  | 3 | 对服务类型和主机重定向 |  | √ |
| 8 | 0 | 回显请求(ping) | √ |  |
| 9 | 0 | 路由器通告 | √ |  |
| 10 | 0 | 路由器请求 | √ |  |
| 11 |  | 超时 |  |  |
|  | 0 | 传输期间生存时间为0(Traceroute) |  | √ |
|  | 1 | 在数据报组装期间生存时间为0 |  | √ |
| 12 |  | 参数问题 |  |  |
|  | 0 | 坏的 I P 首部(包括各种差错) |  | √ |
|  | 1 | 缺少必需的选项 |  | √ |
| 13 | 0 | 时间戳请求 | √ |  |
| 14 | 0 | 时间戳应答 | √ |  |
| 15 | 0 | 信息请求 | √ |  |
| 16 | 0 | 信息应答 | √ |  |
| 17 | 0 | 地址掩码请求 | √ |  |
| 18 | 0 | 地址掩码应答 | √ |  |

其中，最常用的类型是8：回显请求(ping)，以及0：回显应答(ping应答)。

***3.1.2、查询报文***

查询报文是有关信息采集和配置的ICMP报文。

**我们经常用到的ping程序就用到了ICMP查询报文。**

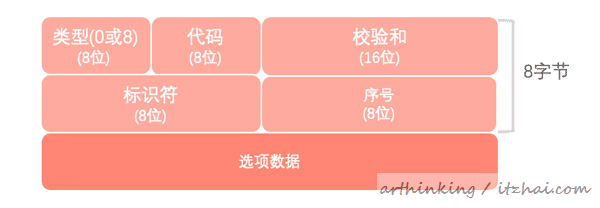
***PING程序***

ping程序会发送一份ICMP回显请求给主机，并等待返回ICMP回显应答。

**ping程序ping不通了，就不能访问对应的主机了吗？**

我们知道，网络管理员可能会用防火墙阻止掉ICMP报文的，这样我们可能就ping不通了，但是主机的可达性不能只取决于IP层是否可达，还与端口号和协议有关，而ping是运行在网络层的，用于测试网络连接状态和信息包发送接收状况，即使ping不通，我们也可能用telnet远程登录到主机的其他端口，如25号端口。

ping程序用到了回显请求和回显应答报文，报文格式如下：

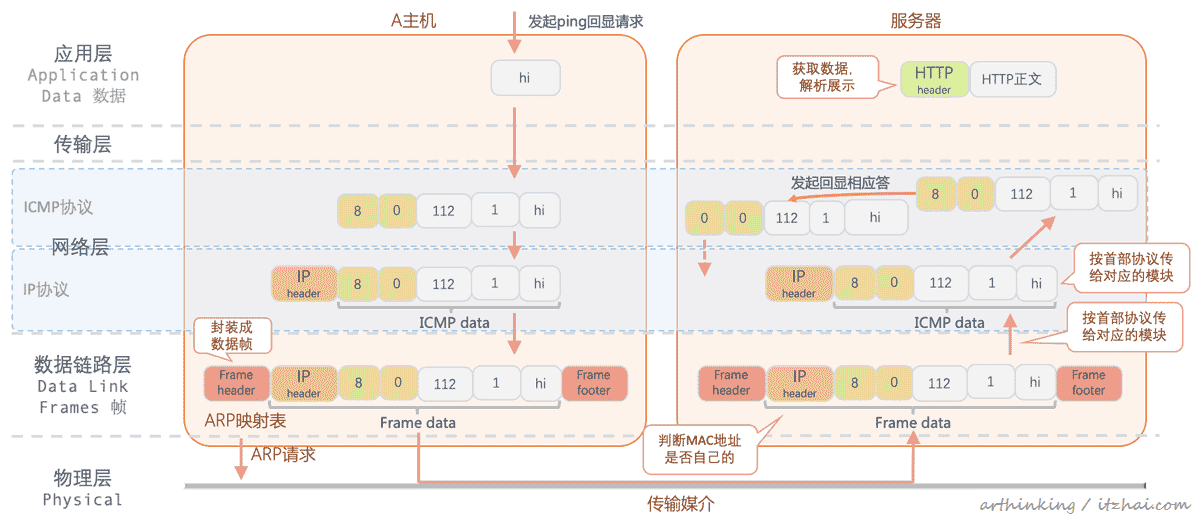


Unix系统实现ping程序时，把ICMP报文的标识符设置为进程ID，在进程内，序号从0开始，每发送一次新的 回显请求就加1，这样就可以同时运行多个ping进程了。

**ping程序的端口号是什么？**

端口号是传输层的东西，ping程序是使用ICMP协议，直接跳过了传输层，所以呢，ping程序是没有所谓的端口号的。

我们发送一个ping请求，数据在协议栈中的处理流程如下：



1. A主机的ping应用程序向服务器发起回显请求，说了一句：**hi**
2. 直接传输到网络层的ICMP协议，进行ICMP数据封装：
   1. image-20200716085942614
   2. 8表示回显请求，112是发起请求的进程号，1表示请求序号
3. IP协议拿到数据后进一步加上IP头，加上自己的IP和目标IP，传输给数据链路层；
4. 数据链路层拿到IP数据包，准备封装成帧，这个时候会去寻找目标IP的MAC地址，如果在A主机的ARP映射表找到了IP的MAC地址，那就直接拿来用了，否则会发起一个ARP广播请求，获取到MAC地址。至于跨网关这种ping，会多了转发的功能更，后面会进行介绍。最终数据链路层封装成数据帧，从网络接口发出去；
5. 服务器拿到数据帧之后，拿到MAC头，判断MAC地址是自己的，就基于拿到Frame data，按首部协议传给对应的模块，即IP模块；
6. IP模块拿到数据，判断到IP跟自己对上了，与是继续拿到IP data，传输给ICMP协议，ICMP协议收到消息，准备应答：
   1. image-20200716085953616
   2. 0表示回显应答
7. 然后按照同样的流程，把数据包发送回A主机。

可以发现，**ping程序是直接用到了网络层的ICMP协议，不经过传输层**。

**是什么原因导致ping失败了？**

ping失败的原因有很多：

* 可能是输错了IP；
* 可能是网络配置不正确，如错误的子网掩码；
* 可能有防火墙软件组织了ping；
* 可能是硬件故障，如损坏了的以太网适配器，电缆，路由器，集线器等。

***如果叫你自己实现一个PING程序，你会怎么做呢？***

提示：为了能处理ICMP网络报文，我们需要用到原始套接字（SOCK\_RAW），而不是SOCK\_STREAM或者SOCK\_DGRAM套接字。

更多提示：[Homework 6: A raw socket ping tool](https://cs450.class.uic.edu/assignments/homework-6-a-raw-socket-ping-tool/" \t "_blank)，思路都在这里了，大家动手做一做，然后就可以有直接操作网络层的工作经验了。😏

***3.1.3、差错报文***

差错报文是有关**IP数据报传递**的ICMP报文。要是发送IP数据报中途产生了异常，那么就会响应ICMP差错报文。

但是不是所有情况都会响应ICMP差错报文，如以下场景：

* ICMP差错报文不会产生另一个ICMP差错报文；
* 目的地址是广播地址或者多播地址的IP数据报不会产生差错报文；
* 作为链路层广播的数据报不会产生差错报文；
* 源地址不是单个主机(源地址为零地址、环回地址、广播地址或者多波地址)的数据报不会产生差错报文；

为什么要这些规则呢？**假如允许ICMP差错报文对广播分组响应，那么就会导致广播风暴了。**

下面我们举一个ICMP差错报文的例子来说明下。

***目标不可达***

上面的表格我们了解到，如果类型为三则表示目标不可达，而根据具体的代码可以进一步划分：

| **类型** | **代码** | **描述** | **查询** | **差错** |
| --- | --- | --- | --- | --- |
| 3 |  | 目标不可达 |  |  |
|  | 0 | 网络不可达 |  | √ |
|  | 1 | 主机不可达 |  | √ |
|  | 2 | 协议不可达 |  | √ |
|  | 3 | 端口不可达 |  | √ |
|  | 4 | 需要进行分片但设置了不分片比特 |  | √ |

下面我们看一个端口不可达的例子来演示下ICMP差错报文附加的信息。

**ICMP端口不可达案例**

这里我们演示通过tftp访问一个不存在的端口号，查看其返回的ICMP响应差错报文。tftp应用在传输层是通过UDP来进行传输数据的

下面我们tftp请求之前先开启tcpdump抓包：

|  |  |
| --- | --- |
| 1 | sudo tcpdump -i en0 -nn host 我的IP and 目标IP |

然后执行tftp命令：

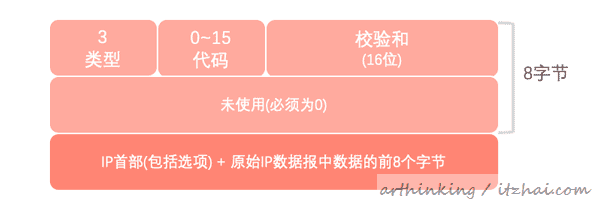
|  |  |
| --- | --- |
| 1 2 3 4 | tftp tftp> connect 目标IP 8090 tftp> get test.foo Transfer timed out. |

可以发现，在等待了大约25秒之后，最终输出：Transfer timed out.

观察tcpdump抓包日志：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 | 16:51:16.739632 IP 我的IP.64517 > 目标IP.8090: UDP, length 20 16:51:16.768590 IP 目标IP > 我的IP: ICMP host 193.112.43.165 udp port 8090 unreachable, length 56 16:51:21.743056 IP 我的IP.64517 > 目标IP.8090: UDP, length 20 16:51:21.751958 IP 目标IP > 我的IP: ICMP host 193.112.43.165 udp port 8090 unreachable, length 56 16:51:26.748387 IP 我的IP.64517 > 目标IP.8090: UDP, length 20 16:51:26.757631 IP 目标IP > 我的IP: ICMP host 193.112.43.165 udp port 8090 unreachable, length 56 16:51:31.752851 IP 我的IP.64517 > 目标IP.8090: UDP, length 20 16:51:31.794217 IP 目标IP > 我的IP: ICMP host 193.112.43.165 udp port 8090 unreachable, length 56 16:51:36.755172 IP 我的IP.64517 > 目标IP.8090: UDP, length 20 16:51:36.774400 IP 目标IP > 我的IP: ICMP host 193.112.43.165 udp port 8090 unreachable, length 56 |

可以发现这里执行了五次UDP请求，每次请求都响应了一个ICMP包，为udp port 8090 unreachable端口不可达，产生了ICMP不可达报文，该报文一般格式如下：



为什么需要返回IP首部：因为IP首部包含了协议字段，使得ICMP知道如何解释后面的8个字节；

为什么需要原始IP数据报中数据的前8个字节：因为这里面包含了源端口和目的端口。

不过看起来我的电脑好像忽略了ICMP报文，还是继续重试了4次。

注意：ICMP报文是在主机之间交换的，网络层的协议，不需要端口号，而以上20个字节的UDP数据报是包含了源端口号和目标端口号信息的。

**为什么TFTP客户程序会继续重发呢？**

因为网络编程中，BSD系统不把从socket接收到的ICMP报文中的UDP数据通知用户进程，除非该进程以及发送了一个connect命令给该接口。标准的BSDTFTP客户程序并不发送connect命令，所以它永远也不会受到ICMP差错报文的通知。

***TRACEROUTE程序***

traceroute工具用于确定从发送者到目的地路径上的路由器。

traceroute主要是通过故意设置特殊的TTL，来达到追踪目的地路径上的路由器的功能。

**TTL运行原理**

TTL：是 Time To Live的缩写，该字段指定IP包被路由器丢弃之前允许通过的最大网段数量。每经过一个路由器，TTL就会减一，然后再把IP包转发出去，如果TTL减到0了，路由器就会丢弃收到的TTL=0的IP包，并向IP包的发送者发送一个ICMP差错报文，类型为11，代码为0：传输期间生存时间为0。

第一轮，traceroute设置TTL值为1，那么遇到第一个路由就返回ICMP容错报文了，下一轮，TTL设置为2…这样依次增加。最终就把整个链路的路由器都试出来了。

当然，有点路由器不会回整个ICMP，这也是为什么你去traceroute一个公网地址，看不到中间路由的原因。

除此之外，**traceroute也可以通过不设置分片，来确定传输链路的MTU(Maximum Transmission Unit, 最大传输单元)**：首先发送一个分组的长度正好与出口MTU相等，如果中间遇到窄点的关口，就被卡主了，这个时候会接收到一个ICMP差错报文，然后调小分组长度重试…

**3.2、网关与路由器**

在讲数据链路层的时候，我们用一个交换机，就构建了一个局域网。但是现在我们局域网里面的一台机器，想要访问另一个局域网的机器，怎么办呢。这就是本节讨论的内容。

我们必须先连接下IP协议。

***3.2.1、IP协议***

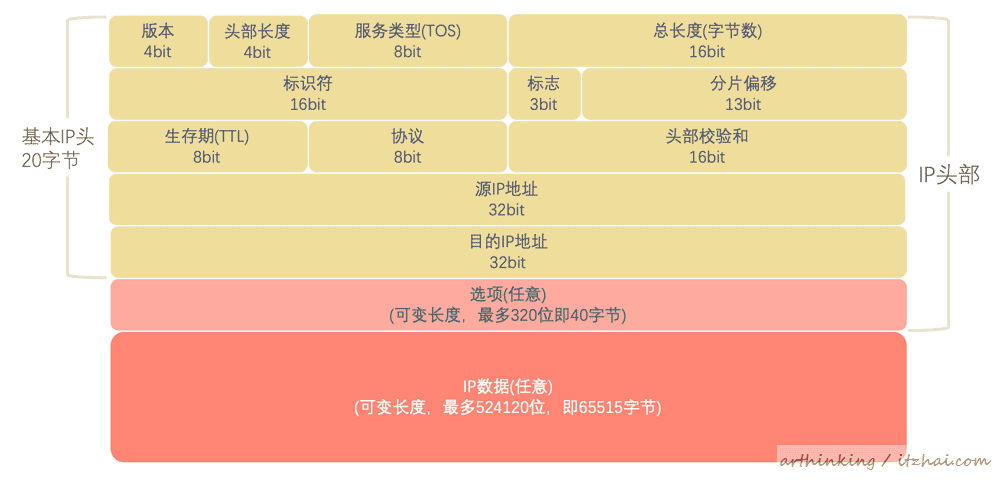
IP是TCP/IP协议簇中最核心的协议，所有TCP、UDP、ICMP等数据都已IP数据报格式进行传输。

***3.2.1.1、IP协议特点***

* **IP协议是不可靠的传输协议**，上一级我们讲到到了ICMP协议，每当传输出现异常，IP层都会丢弃数据包，并且可能会响应一个ICMP差错消息给发送端，而任**何要求的可靠性必须由上层如TCP协议来提供**；
* **IP协议是无连接的**，也就是说IP不维护任何关于后续数据报的状态信息，每个数据报相互独立。具体表现在：可以不按发送顺序接收，不用维护连接状态，免去了维护复制的链接状态信息(后面讲传输层的TCP协议的时候会介绍到)。

***3.2.1.2、IP数据报格式***

下面是IP数据报的格式：



* 版本：协议版本号，指明IPv4还是IPv6；
* 头部长度：最长60个字节；
* 服务类型：包含3bit优先权子字段(已被忽略)，4bit TOS子字段(分别代表最小时延、最大吞吐量、最高可靠性和最小费用)和1bit未用位但必须置0；
* 总长度：指的是整个IP数据报的长度，单位字节；
* 标识符：唯一地标识主机发送的每一份数据报，通常每发送一个数据报就+1；
* 标志：主要用于IP分片；
* 分片偏移：主要用于IP分片；
* 生存期：设置数据报可以经过最多的路由器数；
* 协议：主要表明IP数据是什么协议，用于对数据报进行分用；
* 头部校验和：校验数据报是否正确；
* 源IP地址：发送IP数据报的IP地址；
* 目的IP地址：IP数据报目的IP地址；
* 选项：可选数据；
* IP数据：具体的IP数据；

***3.2.2、路由器***

路由器一般充当一个网关，属于三层设备。会把MAC和IP头取下来根据内容进行处理。路由器有五个网口，分别可以连接5个局域网，每个网口和局域网的IP地址相同的网段，每个网口都是对应的局域网的网关。

5个网口中一般包含一个外网网口，外网网口用于连接到WAN上。

**路由器除了有交换机的功能外，更拥有路由表作为发送数据包时的依据，在有多种选择的路径中选择最佳的路径。**

**一层设备、二层设备、三层设备分别有什么区别？**

路由器是属于OSI第三层的产品，交換机是OSI第二层的产品。

第二层的产品功能在于，将网络上各个电脑的MAC地址记在MAC地址表中，当局域网中的电脑要经过交换机去交换传递数据时，就查询交换机上的MAC地址表中的信息，将数据包发送给指定的电脑，而不会像第一层的产品（如集线器）每台在网络中的电脑都发送。

而路由器除了有交换机的功能外，更拥有路由表作为发送数据包时的依据，在有多种选择的路径中选择最佳的路径。此外，并可以连接两个以上不同网段的网络，而交换机只能连接两个。路由表存储了（向前往）某一网络的最佳路径、该路径的“路由度量值”以及下一个（跳路由器）

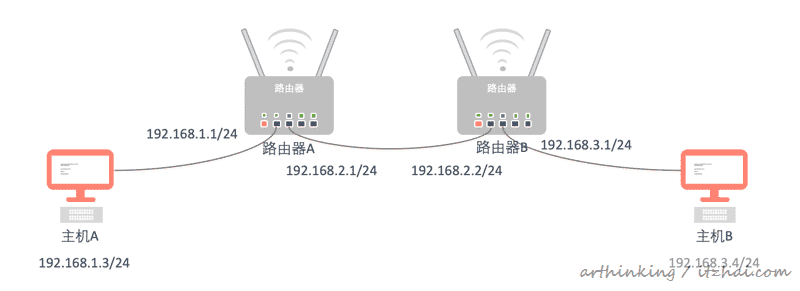
网关地址一般是网段的第一个或者第二个，如192.168.23.0/24这个网段，网关地址可能是192.168.23.1/24或者192.168.23.2/24。

在不同的局域网中，私有IP地址是会重复的，而我们要访问公网的时候，一定要分配一个共有IP地址，所以，我们在访问公网的时候，需要路由器帮忙把私有IP变为共有IP，这种叫做NAT网关，普通内网之间的通信用到的称为转发网关。

我们先来看看转发网关

***3.2.2.1、转发网关***

假设主机A和主机B属于同一个内网，他们通过两个路由器连接起来，如下图：



主机A要访问主机B，流程如下：

* 主机A发现要访问的主机B不是在同一个网段，准备先找到网关，把消息发给网关，网关地址是192.168.1.1，主机A通过ARP获取到了网关的MAC地址，然后发送如下数据包：

|  |  |
| --- | --- |
| 1 2 3 4 | SRC MAC: 主机A的MAC DST MAC: 路由器A的192.168.1.1网口的MAC SRC IP : 192.168.1.3 DST IP : 192.168.3.4 |

* 路由器A的192.168.1.1网口接收包之后，准备把包转发出去。而路由器A中的路由表中匹配到了，要想发送给192.168.3.4/24，需要从192.168.2.1这个网口出去，下一跳地址为192.168.2.2/24。路由器通过ARP拿到了下一跳192.168.2.2/24d的MAC地址，然后发送如下数据包：

|  |  |
| --- | --- |
| 1 2 3 4 | SRC MAC: 路由器A的192.168.2.1网口的MAC DST MAC: 路由器B的192.168.2.2网口的MAC SRC IP : 192.168.1.3 DST IP : 192.168.3.4 |

* 路由器B的192.168.2.2网口接收包之后，准备把包转发出去。路由器B中判断到目标IP在192.168.3.1这个网口所在的局域网，于是通过ARP拿到了192.168.3.4的MAC地址，然后发送如下数据包：

|  |  |
| --- | --- |
| 1 2 3 4 | SRC MAC: 路由器B的192.168.3.1网口的MAC DST MAC: 主机192.168.3.4网口的MAC SRC IP : 192.168.1.3 DST IP : 192.168.3.4 |

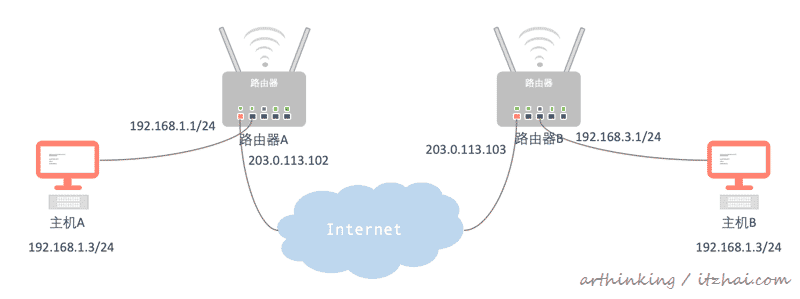
最终，主机B收到数据包。

可以发现在转发网关中，**源IP和目的IP地址都是不会变的，因为整个内网不可能有冲突的IP**。

但是，假如我们要访问外网，情况就不一样了，最终可能会请到到另一个局域网，另一个局域网的私有IP是可能跟我们所在的局域网一样的，为了避免冲突，于是就有了NAT网关。专门在把数据包发送出去之前，把IP改为公网IP。

***3.2.2.2、NAT网关***

现在假设主机A要访问另一个城市的主机B，这里为了演示NAT，我们把模型简化一下，假设路由器出去之后就是公网IP了，如下：



假设路由器A和路由器B都直接接入了互联网。

现在主机A想访问主机B：

* 由于是不同的局域网，主机A不会知道主机B的IP的，而主机B接入互联网的之后，领取到了一个互联网的IP，就是上图路由器WAN口的IP：203.0.113.103，所以主机B会把这个IP作为主机B的IP，最终发出如下IP数据包：

|  |  |
| --- | --- |
| 1 2 3 4 | SRC MAC: 主机A的MAC DST MAC: 路由器A的192.168.1.1网口的MAC SRC IP : 192.168.1.3 DST IP : 203.0.113.103 |

* 192.168.1.1网口接收到包之后，发现要想访问203.0.113.103，就要从203.0.113.102这个网口出去，发给路由器B，路由器B中判断到目标IP就是203.0.113.103这个网口，于是通过ARP拿到了203.0.113.103的MAC地址，然后发送如下数据包：

|  |  |
| --- | --- |
| 1 2 3 4 | SRC MAC: 路由器A的203.0.113.102网口的MAC DST MAC: 路由器B的203.0.113.103网口的MAC SRC IP : 203.0.113.102 DST IP : 203.0.113.103 |

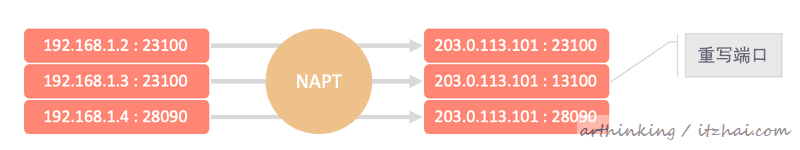
* + 因为消息是要发到公网的，最终SRC IP会被NAT为公网的IP 203.0.113.102；
* 最终路由器B接收到消息，通过NAPT得到最终接收数据报的IP为当前局域网的192.168.1.3/24，最终把消息转发给了这个IP所在的主机B。

**NAPT是如何把一个公网IP翻译为局域网IP的？**

传统的NAT(traditional NAT)包括基本NAT(basic NAT)和网络地址端口转换(Network Address Port Translation, NAPT)。基本NAT只执行IP地址的重写，本质上是将私有地址改写为一个公共地址，这往往取自于一个由ISP提供的地址池或共有地址范围，这种NAT不是最流行的，因为无助于减少需要使用的IP地址数量。

比较流行的做法是使用NAPT，NAPT使用传输层标识符如TCP或者UDP端口，或者ICMP查询标识符来确定一个特定的数据报到底和NAT内部哪台私有主机相关联。

如果局域网两个端口号一样，那么NAPT会重写端口号，保证不一致。如下图，三个局域网的IP需要转换为公网IP，由于有两个的端口重复了，于是NAPT进行了端口重写：



**3.3、路由策略**

***3.3.1、静态路由***

我们通过route命令和iproute命令都可以进行路由策略的配置和查询。

* 可以指明去哪个网络，走哪个网口，网口的IP是什么；
* 也可以创建不同的路由表，针对不同的请求来源，走不同的路由表配置；
* 当然，也可以按照权重给下一跳地址走配置；
* 同一个路由，也可以配多个运营商的网络，针对不同的IP，采用不同的运营商网络
* …

配置时非常灵活的，但是在复杂的网络环境下手动配置路由成本太大了，并且网络结构也是经常发生改版的。

所以，我们可以使用动态路由路由器，这种路由器会根据路由协议算法生成动态路由表，动态的随着网络运行状况调整路由表。

***3.3.2、动态路由协议***

网络是复杂的，为了生成动态的路由表，需要配合特定的算法，主流的动态路由主流有两种算法。

***内网路由协议***

基于链路状态算法实现的OSPF协议(Open Shortest Path First, 开放式最短路径优先)：主要用于数据中心内部，因此也成为内网路由协议(Interior Gateway Protocol，IGP)，关键是找到最短的路径。

OSPF是一种链路状态路由协议。可以将其视为网络的分布式地图。

***外网路由协议***

基于距离矢量算法实现的BGP协议(Border Gateway Protocol，外网路由协议)：距离矢量，就是每个路由器都保存一个路由表，路由表每行保存了下一跳的路由器，以及距离下一跳路由器的距离。也成为边界网关协议。

在BGP的世界中，每个路由域都称为自治系统或AS。BGP所做的工作通常是通过选择遍历最少自治系统的路由：最短的AS路径来帮助选择通过Internet的路径。

我们会把重点放在传输层以上，所以动态路由协议这部分我们暂时不做不深入研究。

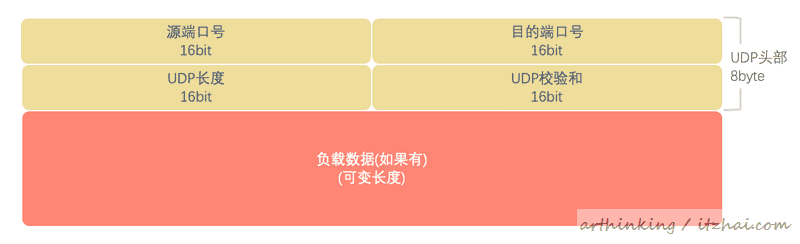
**4、传输层**

传输层涉及到两个重要的协议：UDP和TCP，本节我们重点介绍这两个协议。

**4.1、UDP协议**

***4.1.1、UDP数据报格式***

UDP基本没干啥事，继承了IP包的特性：数据可能丢失，顺序传输无法保证。UDP与后边介绍的TCP不一样，是无状态的。我们来看看UDP数据报的格式：



* 源端口号：发送数据报方使用的端口号，用于标识发送进程；
* 目的端口号：接收数据包方使用的端口号，用于标识接收进程；
* UDP长度：UDP头部和UDP负载数据的字节长度；
* UDP校验和：UDP校验和覆盖UDP头部和UDP数据和一个伪头部(区别：IP头部校验和只覆盖IP头部)，伪头部衍生子IPv4头部字段的12个字节，或者衍生子IPv6头部字段的一个40字节的伪头部；
* 负载数据：具体的UDP数据。

可以发现，UDP与下层不同，是需要端口号的。

***为什么UDP需要端口号，TCP和UDP端口号可以相同吗？***

类似ICMP协议回显请求的标识符，UDP的端口用于区分是哪个进程的数据包，如果没有端口号，那么就不知道应该把数据包最终交给哪个进程来处理了。

TCP端口号由TCP来查看，UDP端口号由UDP来查看，TCP端口号和UDP端口号是相互独立的，所以是可以相同的。每个请求都有源IP、目标IP、源端口号、目标端口、协议五个元素来标识的，每个协议的端口池是完全独立的。

***为什么UDP的端口号最多是65535个？***

在UDP/TCP协议中源端口和目的端口都只有16位，也就是说端口的取值范围为0~65535。

***4.1.2、UDP特点***

UDP在IP层之上，没有做其他的封装，主要表现如下特点：

* 数据可能丢失，顺序传输无法保证；
* 无状态，不需要像TCP那样要建立连接；
* 没有拥塞控制，来一个包就发一个。

***4.1.3、UDP使用场景***

基于UDP的特点，UDP主要用于以下场景：

* 需要资源少，在网络情况比较好的内网，或者对对包不敏感的场合。如DHCP和TFTP就是基于UDP的；
* 广播场景，不需要一对一建立连接，如DHCP；
* 需要时延低，允许丢包，不关注网络拥塞的场景，如视频直播这种流媒体，实时游戏，通信，物联网等领域。

**4.2、TCP协议**

TCP是我们平时用到最多的协议，特别是做web开发的时候，或者互联网后端开发，真的是时时刻刻都会用到，这里我会展开来讲。《TCP/IP详解-卷1：协议》一书中花了6章来讲解TCP的各种功能，单单是从TCP/IP协议栈的名称就可以看出，TCP协议的分量有多重了。为此，面试官张口就聊TCP咋的咋的。



与UDP不同，TCP做了很多功能的封装与实现。

先来**简单介绍下TCP协议**：

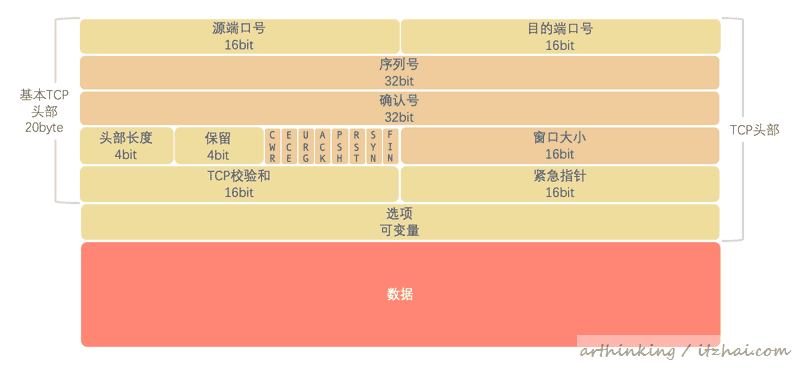
TCP给应用程序提供给了一种与UDP完全不同的服务。

**TCP是面向连接的可靠的服务**：面向连接指TCP的两个应用程序必须在它们可交换数据之前，通过相互联系来建立一个TCP连接；

**TCP提供了一种字节流抽象概念给应用程序**：TCP不会自动插入记录标志或者消息边界，这意味着TCP没有限制应用程序的写范围。发送端分两次发10字节和30字节，接收端可能会以两个20字节的方式读入。

我们还是先来看看TCP数据报的格式吧，这个可比UDP复杂多了，但是也是设计的恰到好处的。

***4.2.1、TCP数据报格式***



如上图，头部深黄色部分为TCP特有的重点字段，后面TCP相关功能基本都是靠这些特有的字段来实现的。

* 源端口号和目的端口号：同UDP一样，主要用于区分数据应该转发给哪个应用；
* 序号：这个序号是为了解决乱序问题，32位无符号数，到达2^32-1后再重新从0开始；
* 确认号：确认已经接收到了哪里，该确认序号表示该确认号的发送方期望接收的下一个序列号。该字段只有在ACK位字段被启用的情况下才有效，所以也成为ACK号或者ACK段；
* 状态位：该状态位会让TCP连接双方的状态发生流转，常见的状态为，后面讲建立连接和断开连接的时候会用到：
  + ACK：回复状态，启用该状态的情况下，确认号有效，连接建立之后一般都是启用状态；
  + SYN：发起一个连接；
  + RST：重置连接，连接去表，经常是因为错误导致；
  + FIN：结束连接，表示该报文的发送方已经结束想对方发送数据；
  + CWR：拥塞窗口减小，发送方降低发送速率；
  + ECE：ECN回显，发送方接收到了一个更早的拥塞通告；
  + URG：紧急，表示紧急指针字段有效，很少用到；
  + PSH：推送，表示接收方应该尽快给应用程序传送这个数据——没有被可靠的实现或用到；
* 窗口大小：流量的窗口大小，用于流量控制，通信双方各声明一个窗口，这个大小表明了自己当前的处理能力；
* 校验和：覆盖了TCP的头部和数据，以及伪头部数据(与UDP使用的相似的伪头部进行计算)；
* 紧急指针：只有在URG位启用的时候才有效；
* 选项：如最大段大小等其他的可选项；
* 数据：TCP数据报的数据内容。

***4.2.2、TCP特点***

TCP基于以上数据报的各种字段，实现了以下功能：

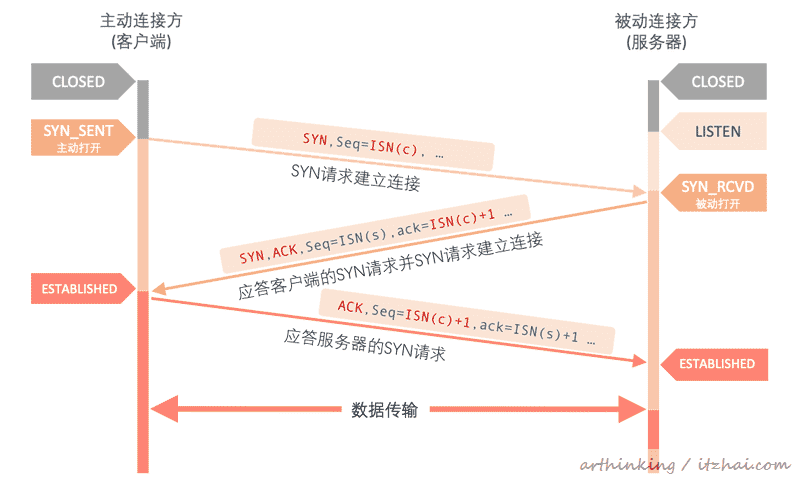
* 数据的顺序传输；
* 丢包重传，保证可靠；
* 连接维护；
* 流量控制，保证稳定；
* 拥塞控制，及时调整，最大程度保证传输正常进行。

***4.2.3、连接管理***

我们首先来看看连接是如何建立的，这里就涉及到TCP的三次握手了。

***4.2.3.1、TCP三次握手***

三次握手流程如下：



可以发现，为了实现可靠连接，双方都需要发起建立连接。具体流程如下：

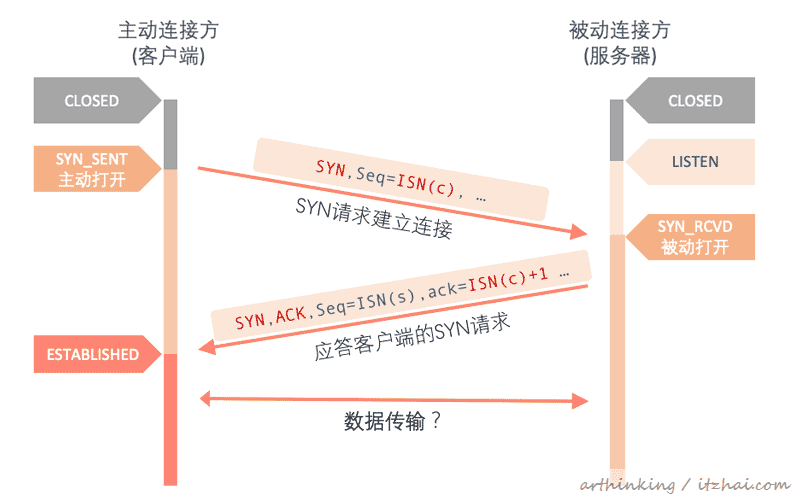
* 第一次握手：主动连接方发送一个SYN报文段指明自己想要连接的端口号，以及客户端消息的初始化序列化ISN(c)；
* 第二次握手：服务器接收到消息后，也发送自己的SYN报文，包含了服务端的初始化序列号ISN(s)，并设置确认号ack=客户端序列号+1；
* 第三次握手：客户端应答服务器的SYN，将服务端的序列号+1作为ack返回给服务端。

总结一下：**客户端与服务端利用SYN报文交换彼此的初始化序列号。在我们熟悉的Socket编程中，三次握手在执行connect的时候触发。**

其中的ACK应答和递增的序列化是可靠性的保证。

**为什么是三次握手，而不是两次或者四次？**

如果是两次：



客户端请求建立连接，服务端收到了请求，并且做出了响应，很明显，服务器没法知道这个响应究竟有没有被接收，也许可能客户端迟迟收不到SYN响应，于是结束了请求。这个时候再传消息网络层就会收到一个ICMP目的不可达的差错报文。

同理：客户端的SYN请求如果迟迟没有服务器的响应，那么也会重发SYN，最终如果服务端可能收到两个SYN，客户端想要建立一个连接，但是服务器收到两个SYN之后，建立了两个连接（当然，实际上的三次握手服务端是会判断客户端的请求序列号的，发现是同一个序列号，并不会建立多个连接，这也说明**序列号的重要性**）。

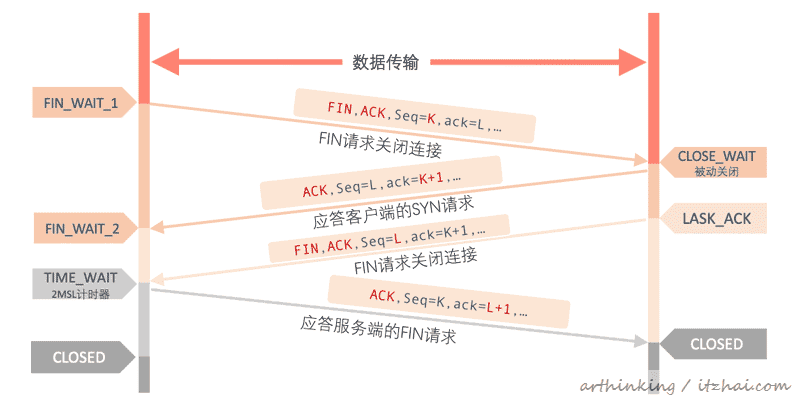
为什么不需要四次呢？因为如果服务端和客户端双方都发起SYN，并且收到ACK之后，就都知道对方接受了自己的请求了，已经没有必要再继续确认下去了。

为什么UDP端口号为65535个？

***4.2.3.2、TCP四次挥手***

接下来我们看看连接关闭的流程，**连接的任何一方都可以发起关闭操作，此外，也支持双方同时关闭连接**。在传统的情况下，负责发起关闭连接请求的通常是客户端。

这个流程又被称为四次挥手：



* 连接的主动关闭者发送一个FIN段请求关闭连接，携带了Seq=K，指明接收方希望看到的自己的当前序列号；携带了ack=L，指明自己想要接受到的下一个消息的序号。这个时候，连接主动关闭者表明了自己已经没有数据要发送了，但是仍然可以接受被动关闭者发送的数据；
* 连接的被动关闭者进行了ACK回应，**ack为K+1，表明自己已经成功接收到了主动关闭者发送的FIN**。但是自己还未准备好关闭，所以主动关闭者会进入FIN\_WAIT\_2等待状态；
* 紧接着被动关闭者也发送了一个FIN端请求关闭连接，携带了Seq=L。告诉主动关闭者自己也准备好了关闭；
* 最后连接的主动关闭者接收到了对方的FIN关闭请求，也回应了一个ACK，同样的ack=L+1，表明自己已经成功接收到了被动关闭者发送的FIN；

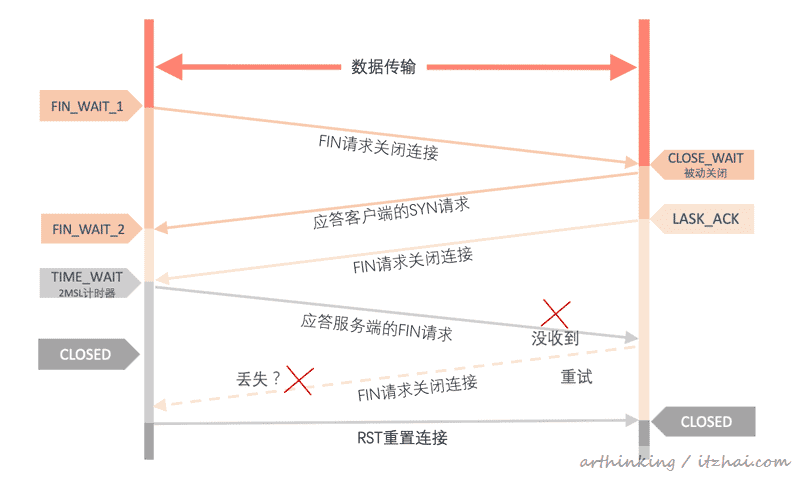
可以发现，**因为TCP是全双工的，双方都要单独发起关闭请求，只有当连接双方都发起FIN关闭请求操作，并且得到确认之后，才完成一个完整的关闭操作**，这也是被称为四次握手的原因。

信息发送期间的状态流转如上图所示。其中主动关闭者在CLOSED状态之前，有一个TIME\_WAIT状态，那么问题来了：

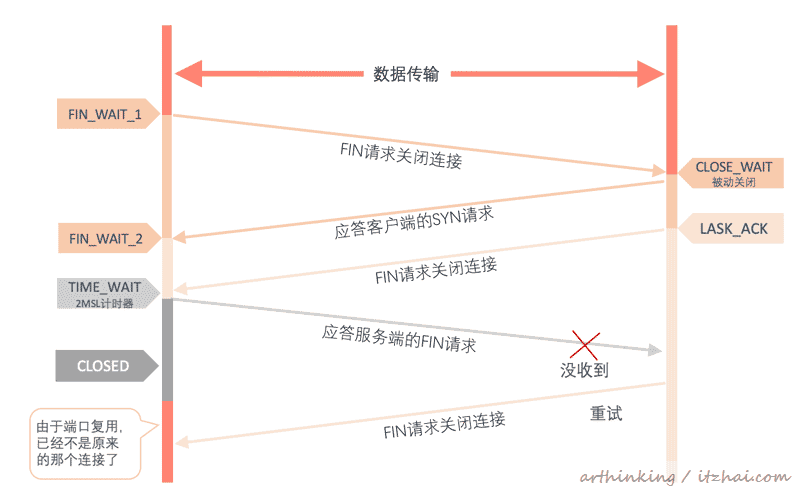
**为什么要有TIME\_WAIT状态呢？**

我们知道主动关闭者在应道对方的FIN请求，有可能对方是收不到的，如果收不到的情况下，那么对方就可能认为自己的FIN请求丢失了，需要重新发起FIN请求，所以**主动关闭者需要有一个足够长的等待时间，让对方有重试的机会**。

等待时间是2MSL(Maximum Segment Lifetime，报文最大生存时间)，这也是报文在网络上最大的生存时间，超过了这个时间就会被丢弃。RFC 793中规定MSL为2分钟，实际应用中常用的是30秒，1分钟和2分钟等。如果超过了这个时间，那么主动关闭者就会发送一个RST状态位的包，表示重置连接，这个时候被动关闭者就知道对方已经关闭了连接：



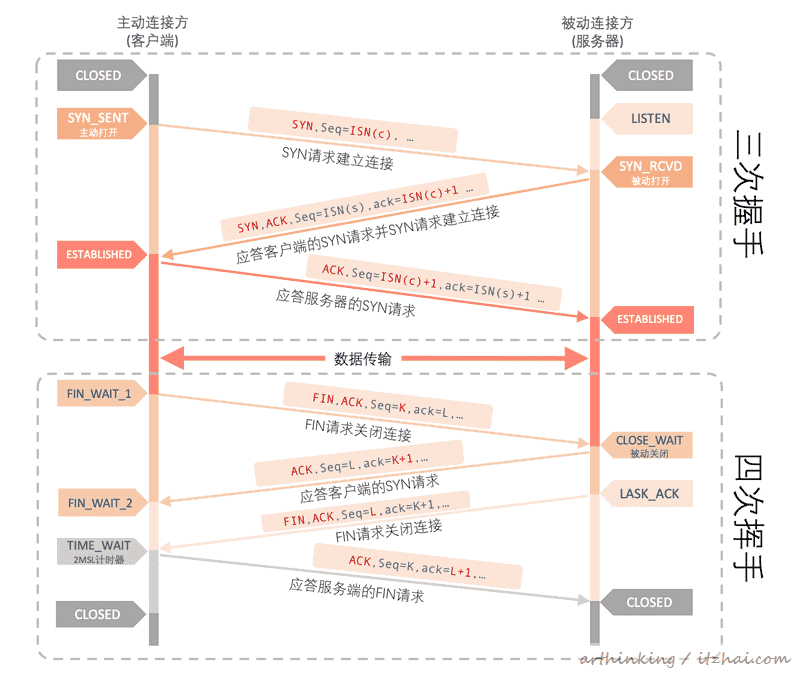
如果主动关闭者不进行等待，会出现什么问题呢？如下：



可以发现，由于端口复用，主动关闭者已经开启了另一个连接，这个时候被动关闭者还在重试发起FIN请求，导致新主动关闭者新的连接收到了很多没用的包。因为包是有序列号的，所以可以判断到不是本次连接该接收的包。为此，我们需要让主动关闭者进行等待，确保被动关闭者不会再发FIN请求了，再进行端口复用。

***4.2.3.3、完成连接流程***

完整连接流程如下：



可以发现，**每个TCP连接在正常的建立和关闭的基本开销是7个报文段**，如果只是需要交换很少量的数据时，有写程序更愿意选择使用UDP协议。但是UDP会面临数据丢失，拥塞管理，流量控制等问题。

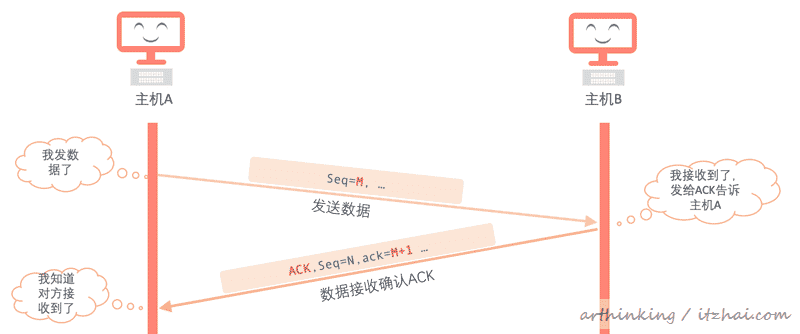
***4.2.4、TCP状态机***

介绍了三次握手和四次挥手，我们再看看看以下这个TCP状态机就清晰多了。（如果没有看过三次握手和四次挥手流程，不建议直接看这个状态机，真的是太复杂了…）

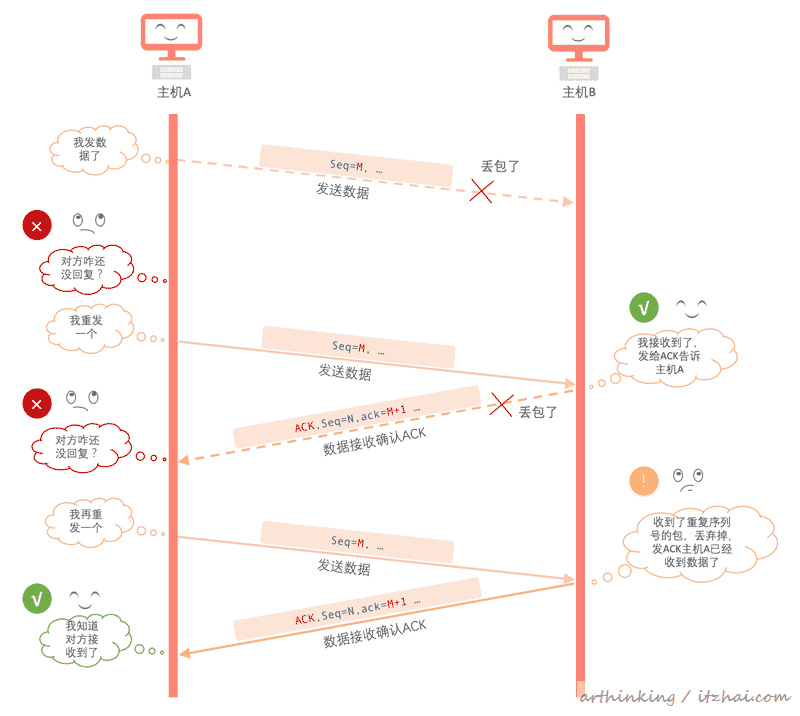
***4.2.5、数据传输***

***4.2.5.1、如何保证可靠传输：ACK+序列号***

假设主机A通过TCP向主机B发送数据，当主机A的数据到达主机B时，主机B会发送一个确认应答消息ACK。主机A收到ACK之后，就知道自己的数据已经被对方接收了：

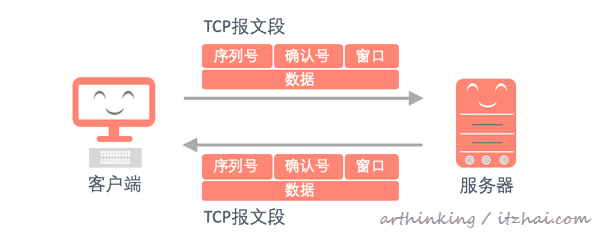


如果主机一直没有收到ACK，一定时间之后，就会重发，因此，即使主机A的数据报没有发到主机B，或者主机B的ACK数据包丢失了，也有重传机制，确保双方最终可以通过重传确保能够正确收到消息：



从上图也可以看出，主机A实际发了两次同样的数据给主机B，主机B可以通过序列号，判断是重复数据，然后就丢弃了，但是还是会发送一个ACK告诉主机A已经收到消息。

***4.2.5.2、流量控制与窗口管理***

在TCP头部中，为了实现流量控制，包括顺序问题与丢包问题，我们重点关注TCP头部的这三个字段：序列号，序列号与确认号：  


（注意：后面部分数据传输图中的发送方统一称为客户端或者发送端，接收方统一称为服务端或者接收端，实际的数据传输，可以是两台电脑之间，或者是两台服务器之间）

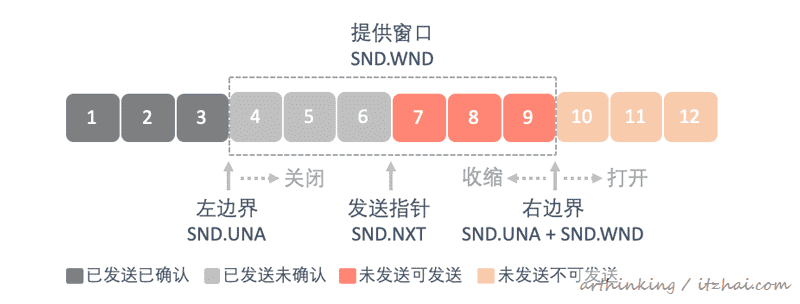
其中TCP头部的窗口字段表明自己的处理能力，代表着可用缓存空间的大小，以字节为单位。

接下来再看看滑动窗口。

TCP连接的每个端都可以收发数据，每个端的收发数据量是通过一组窗口结构来维护的。**每个端都会包含一个发送窗口结构和接收窗口结构。**

**发送窗口结构**

发送窗口结构如下图所示：



其中：

* SND.WND：提供窗口大小是由接收返回的ACK中的窗口大小字段控制的；
* SND.UNA：记录窗口左边界的值；
* SND.UNA + SND.WND：记录窗口右边界的值；
* SND.NXT：记录下次发送的数据

所谓窗口，就是左右边界会根据情况进行调整的窗口，由主要三个动作：

* **关闭**：窗口左边界右移，当已发送的数据得到ACK的时候，就会进行关闭，提供窗口大小减小；
* **打开**：窗口左边界右移，当已确认的数据得到处理后，那么接收端可用缓存就会变大，这个时候通过打开操作让提供窗口大小变大；
* **收缩**：窗口右边界左移，使得提供窗口大小减小；

**接收窗口结构**

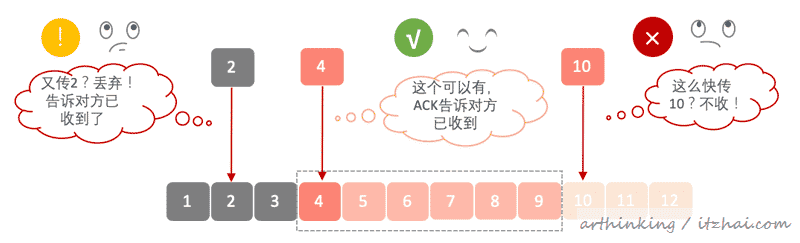
接收窗口与发送窗口结构类似，如下图：



**从滑动窗口看如何保证可靠传输：顺序与丢包问题**

为了避免接收重复数据：接收到的数据包小于左边界，说明是已经确认过的，将把数据报丢弃；如果接收到的数据报序列号大于右边界，说明暂时超出了处理能力范围，也将会被丢弃。

为了保证已确认数据包的连续性，接收到的数据包的序列号与已确认 已接受部分连续的时候，才表示真正的已确认，左边界才可以右移。

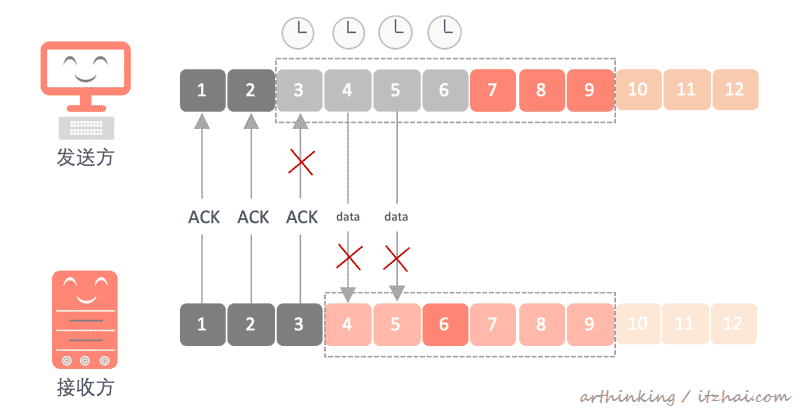


***4.2.5.3、超时重传机制***

**基于计时器的重传超时机制(Retransmission Ttimeout, RTO)**

TCP在发送数据的时候会设置一个重传计时器，如果计时器超时仍然没有收到ACK确认信息，那么会进行重传操作。

如下图是超时重传的演示说明例子：



对于接收方来说，1，2，3都已经接收并且发送ACK了，3的ACK丢失了。

**ACK丢失的场景：**过了一段时间，3的计时器发现超时了，于是会触发超时重传。但是这个时候接收方发现3是在已接受已确认区域，于是会丢弃3，并反馈一个ACK；

**数据丢失的场景：**4和5的数据传输丢失了，计数器发现超时，也会进行超时重传，保证4和5可以传给接收方，并拿到ACK反馈。

**关于重传时间间隔**

在ICMP端口不可达案例中，采用UDP的TFTP客户端使用简单且低效的超时重传策略：设置足够大的超时间隔，每5秒进行一次重传；

而TCP的基于计时器的重传策略是如果发生重试，可以有两种处理方式：

* 一种是基于拥塞控制机制，减小发送窗口大小；
* 另一种是超时时间间隔会一直加倍。

**关于重传时间**

重传时间需要讲到自适应重传算法，一种计算重传时间的算法，大致流程：

TCP通过采样RTT的时间，进行加权平均，算出一个值，最终得到一个估计的重传时间。

|  |  |
| --- | --- |
| 1 2 3 4 | > 初始值：原始值 > 测量之后：RTO = RTTs + 4\*RTTd > (RTTs：加权平均值，RTTd：偏差值) > |

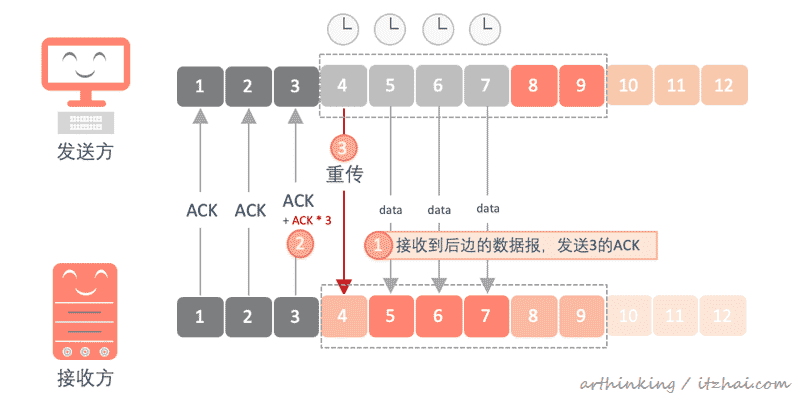
因为网络是不断变化的，所以重传时间也会处于变动状态。

**基于反馈信息的快速重传机制**

快速重传机制是这样的：当接收方接收到一个序列号大于下一个所期望的报文段的时候，就会检测到数据流中间丢失的间隔，然后发送冗余的ACK，向发送者索要确实的间隔。当发送者收到一定数量的冗余的ACK(称为重复ACK的阈值或dupthresh)之后，就不等定时器过期了，直接重传丢失的 报文。

重复ACK的阈值通常为3，一些非标准化的实现可基于当前的失序程度动态调整。

如下例所示：发送方的4、5、6、7都已经发送出去了，但是接收方接收到了5、6、7，少了4，会在分别收到5、6、7的时候都发一个3的ACK，向发送方索要下一个数据4。这样发送方就收到到3个3的ACK了，于是就主动发起了4的重传，不等待重传计时器超时了：



**带选择确认的重传SACK**

虽然重传保证了数据的到达，但是重传应该尽可能保证不重传以正确接收到的数据，而SACK信息能更快速的实现空缺填补并且减少不必要的重传。

随着选择确认选项的标准化[RFC2018]，TCP接收端可以提供SACK的功能了，通过TCP头部的累计ACK号字段来描述其接收到的数据。

每当缓存存在失序数据时，接收端就可以生成SACK，代表着缓存接收状态地图，这样通过将缓存的接收状态地图发给发送方，发送方就很快可以知道是什么数据丢失并发起重传了。

这种重传机制下，**窗口内的其他报文段也可以被接收确认**，但只有在接收到等于窗口的左边界的序列号时，窗口才会前移。这样就减少了窗口内的不必要的重传。

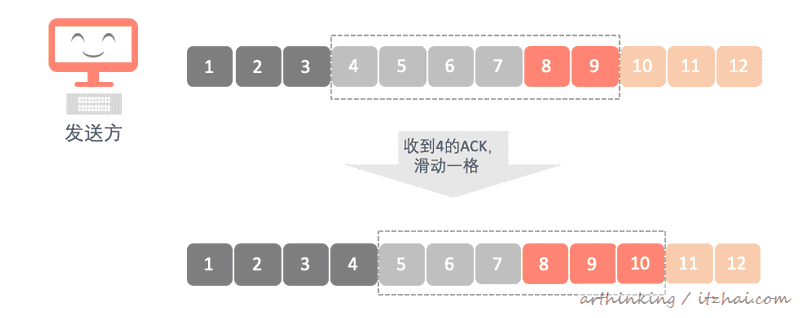
***4.2.5.4、流量控制***

流量控制指的是通过控制发送方和接收方的窗口大小，以使得接收方缓存中已接受的数据处理不过来时，通过减小发送方的窗口大小，让接收方能有足够的时间来接收数据包；或者是接收方比较空闲时，尝试让发送方调大窗口大小，以加快传输，合理利用空闲的网络资源。

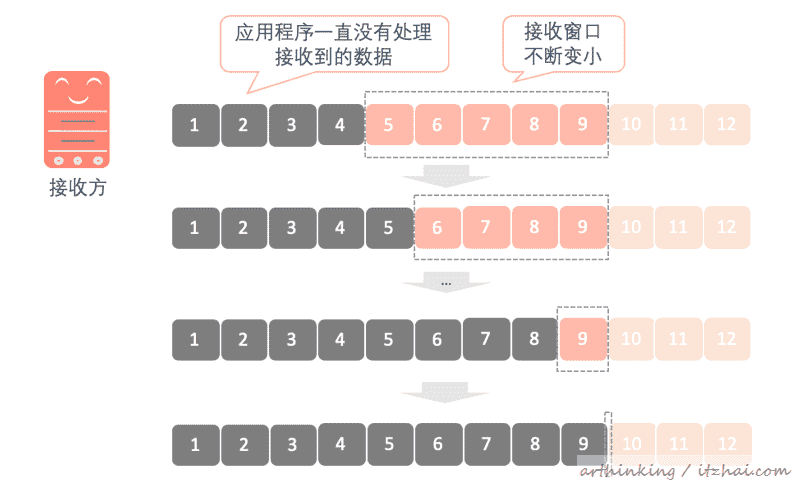
**流量控制主要是通过TCP头的窗口大小来调节的。**发送端收到接收端的通告窗口之后，得知接收端可接收的数据量。

下面举例来说明。

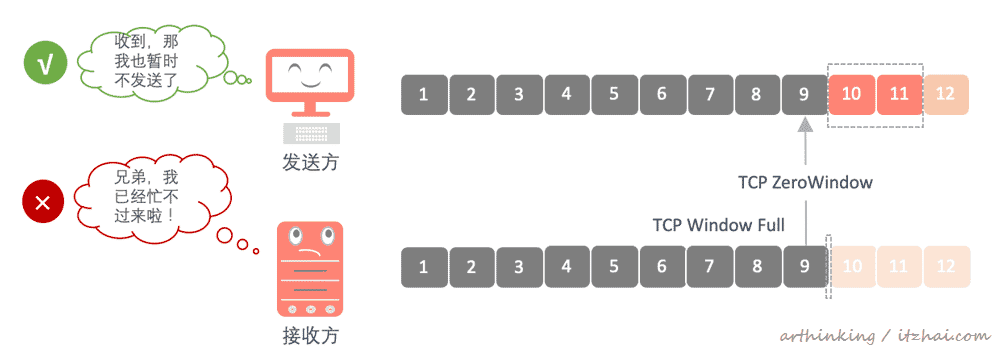
正常情况下，发送方左边界每关闭一格，右边界就打开一个，多一个可发送的单元：



我们知道接收端接收并确认数据之后，会放到缓存中，等待应用程序处理，如果应用程序一直没有处理，最终会导致接收端没有更多空间来存储到达的数据了，如果应用程序一直没有处理数据，那么窗口右边界可能就不会打开了，最终接收的窗口大小变为0：

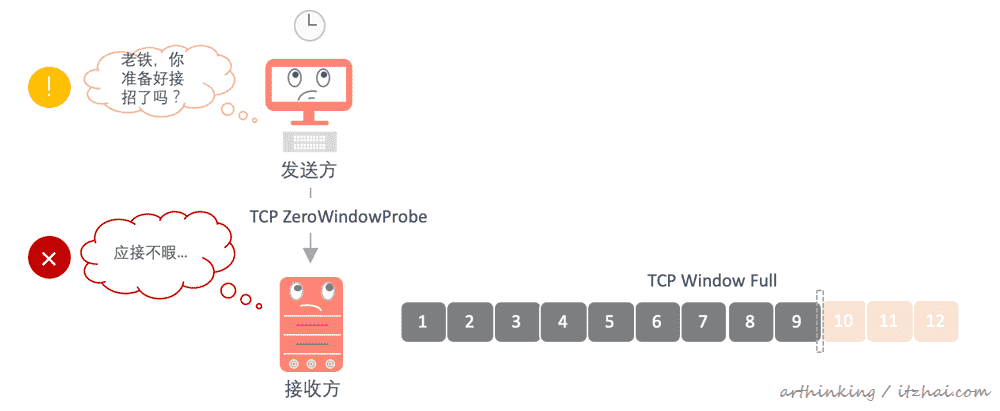


这个时候接收端就会发送一个零窗口通告（TCP ZeroWindow），告知发送端不要再发送数据了，我已经处理不过来了，于是发送方就暂停发送数据了，等待接收端的窗口更新（TCP Window Update）通知：



这样，接收方就可以有时间来处理接收的数据了，等到有了足够多的缓存之后，于是会给发送端传输一个窗口更新通知。

为了避免由于窗口更新通知ACK丢失，到时双方陷入等待的僵局，在发送方停止发送数据之后，会采用一个持续计时器间歇性的查询接收端，给接收端发送窗口探测（TCP ZeroWindowProbe）请求，要求接收端返回TCP ZeroWindowProbeAck，看看是否窗口是否已经增加了：



***4.2.5.5、拥塞控制***

前面我们讲到，可以通过滑动窗口大小来控制流量，从而为接收方缓解压力，避免不必要的丢包。

而拥塞控制，就需要用到拥塞窗口了。拥塞控制主要用于避免丢包和超时重传。

**反映网络传输能力的变量称为拥塞窗口(congestion window)，记为cwnd。**

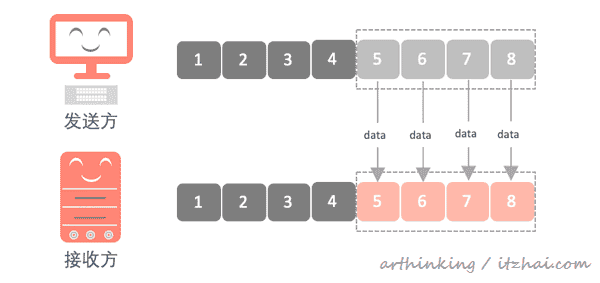
可以理解为**滑动窗口是为接收方服务的，而拥塞窗口是为整个网络通道服务的，拥塞窗口大小又会受制于接收方滑动窗口大小，并且会因为网络原因进行调整。**因为网络通道中的任何一个环节都有可能影响整体的传输效率。

**4.2.5.5.1、发送实际可用窗口**

那么我们发送端实际可用窗口应该是多少了，这里我们记实际可用窗口大小为W，那么W为接收端通知窗口awnd和拥塞窗口cwnd的较小者：

|  |  |
| --- | --- |
| 1 | W = min(cwnd, awnd) |

假设网络没有任何问题，并且带宽足够宽，数据包不会在传输过程遇到需要排队等待的情况下，这种理想状况下，也就是没有网络延迟，接收方收到一个数据包，立刻就ACK一个，立刻空出一个可传输单元，发送的实际可用窗口就是接受方的滑动窗口大小了，如下：



理想是很美好的，但是实际网络情况是非常复杂的，TCP根本不知道里面会发生什么情况，也许W还没到达接收端滑动窗口大小，网络中就因为中间的瓶颈导致丢包了，那么更加会增加重传的频率。所以为了能减少丢包和超时重传，需要有一些动态发送端窗口大小的策略。

**4.2.5.5.2、发送端窗口调整策略**

虽然可以通过接收方的ACK得到对方的接收窗口大小，但是因为刚开始并不知道拥塞窗口是多少，所以只能以越来越快的速率不断发送数据，直到出现数据包丢失为止。

**通常TCP在建立新连接的时候会执行慢启动，直到有包丢失，然执行拥塞避免算法进入稳定状态。**

**慢启动**

初始窗口设为IW(Initial Window, IW)，IW=SMSS(发送方的最大段大小)。

先发送初始窗口大小的数据，没有出现丢包，并且每收到一个ACK，慢启动算法就会以min(N,SMSS)来增加cwnd的值。可见这是指数性的增长。

直到出现了网络拥塞，出现丢包、超时重传，说明已经到达了慢启动的阈值ssthresh(slow start threshold)，这个时候cwnd减少一半，并作为新的ssthresh。

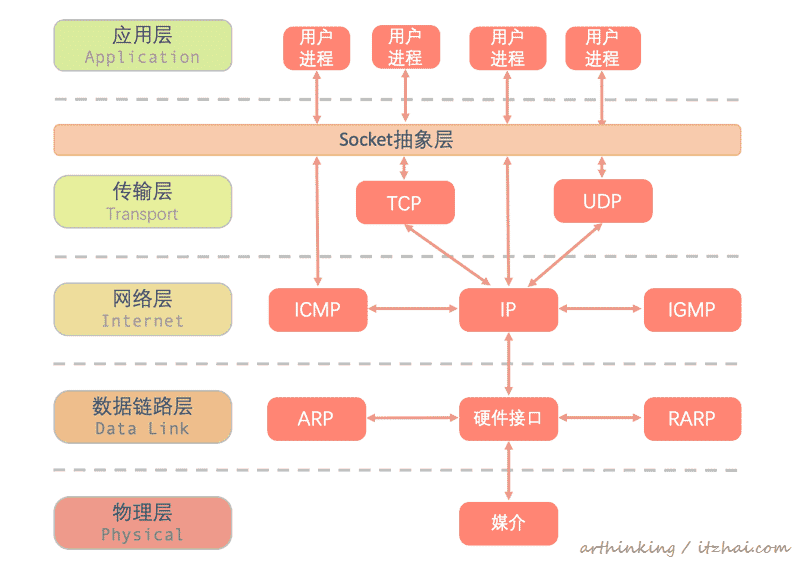
**避免拥塞**

一旦达到慢启动的阈值之后，为了得到更多的传输资源而不影响其他连接的传输，TCP实现了拥塞避免算法。一旦确定慢启动阈值，TCP会进入拥塞避免阶段，这个时候cwnd每次的增长值近似于成功传输的数据段大小。也就是说由原来慢启动的指数增长，变为了线性增长。

**4.3、Socket编程**

***4.3.1、SOCKET是什么***

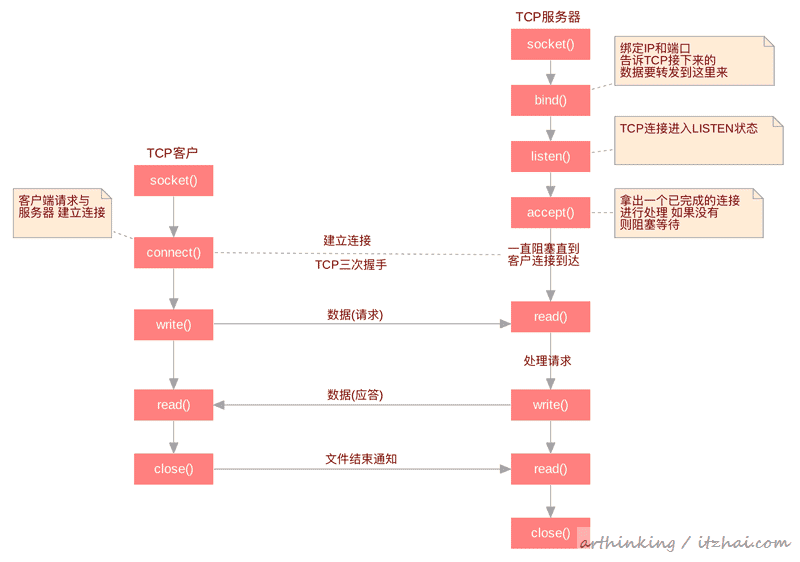
Socket是一个抽象层，主要是把TCP/IP层复杂的操作抽象为几个简单的接口提供给应用层调用，进而实现应用进程在网络中通信。Socket主要是端到端之间的传输协议(网络层之上的协议)。因**为Socket是一种高层的抽象网络API，是一种端到端的通信，只能访问到端到端协议之上的网络层和传输层**。



Socket起源于Unix，**在Unix中，一切皆文件，Socket也不例外**，是一种打开-读/写-关闭的模式实现的。在服务器和客户端各自维护了一个文件。

***4.3.2、基于TCP的SOCKET通信交互流程***

我们先来看一下基本TCP客户/服务器程序的套接字函数调用过程：



**TCP Socket的文件结构**

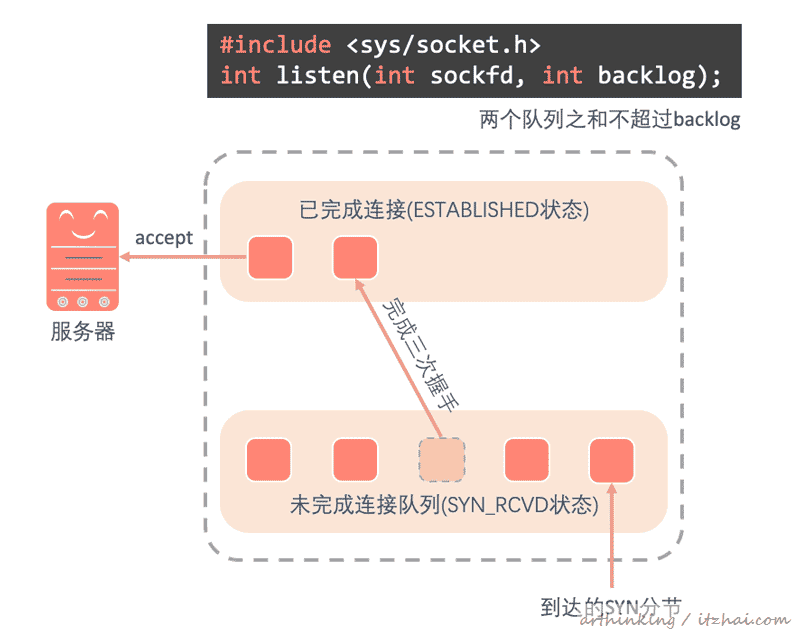
在内核中，Socket是一个文件，不过Socket对应的inode不是保存在硬盘上，而是在内存中，该inode指向了Socket在内核的Socket结构。内核的Socket接口主要由两个队列：发送队列，接收队列。

**内核为监听套接字维护的两个队列**

对于每个监听Socket，内核都为其维护了两个队列：

* 未完成队列(incomplete connection queue)：这个队列的套接字服务端正在等待完成TCP三次握手，处于SYN\_RCVD状态；
* 已完成连接队列(completed connection queue)：完成了三次握手的Socket连接会进入这个队列，处于ESTABLISHED状态。

如下图：

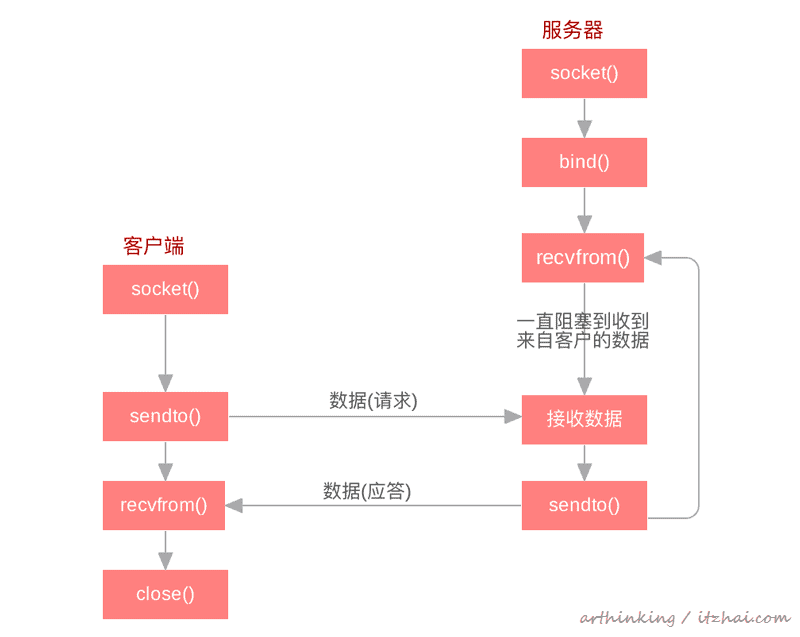


***4.3.3、基于UDP的SOCKET通信交互流程***

UDP不需要三次握手，所以不需要listen和connect，但是交互仍然需要IP和端口号，需要bind。

UDP不用维护连接状态，所以不需要针对每个连接建立一组Socket，只需要一个就可以了。

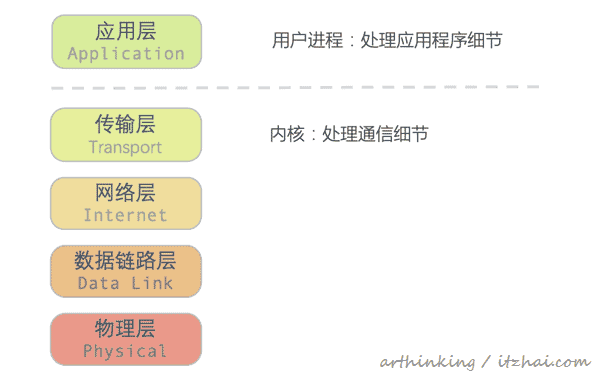
以下是UDP的Socket通信交互流程图：



到目前为止，我们把物理层、数据链路层、网络层、传输层主要的协议和功能都介绍了一遍。基于这些底层的协议栈支撑，我们可以很快的构建出应用层的程序，接下来我们简单讲一下应用层。

**5、应用层**

应用层位于操作系统用户态运行，而我们前面讲到的那层是运行在操作系统内核态的：



一般我们都是通过Socket网络API来访问内核态的各层的协议模块。

常见的应用层协议如下：

* HTTP：Hypertext Transfer Protocol，超文本传输协议，是一个基于请求与响应，无状态的，应用层的协议，常基于TCP/IP协议传输数据，互联网上应用最为广泛的一种网络协议，所有的WWW文件都必须遵守这个标准。设计HTTP的初衷是为了提供一种发布和接收HTML页面的方法；
* HTTPS：Hypertext Transfer Protocol Secure，安全超文本传输协，是HTTP的扩展，用于在计算机网络上进行安全的通信，并在Internet上广泛使用；
* 流媒体：流媒体（streaming media）是指将一连串的媒体数据压缩后，经过网上分段发送数据，在网上即时传输影音以供观赏的一种技术与过程，此技术使得数据包得以像流水一样发送；如果不使用此技术，就必须在使用前下载整个媒体文件。
* P2P协议：Peer-to-peer，计算或联网是一种分布式应用程序体系结构，可在对等体之间划分任务或工作负载；对等方可以将其部分资源（例如处理能力，磁盘存储或网络带宽）直接提供给其他网络参与者，而无需服务器或稳定主机的集中协调。大家经常用的迅雷下载，百度网盘下载就用到了这个协议；
* WebSocket：WebSocket是一种计算机通信协议，与HTTP不同，WebSocket可通过单个TCP连接提供全双工通信通道。WebSocket在HTML5规范中最初被称为TCPConnection，是基于TCP的套接字API的占位符。很多在线聊天室，办公协同软件都用到了WebSocket。

在应用层，大部分的开发工程师可以大展拳脚。

* HTTP协议的实现，最经典的莫过于Tomcat服务器了，关于具体的实现，可以参考这本书：《深入剖析Tomcat》；
* Github上面有一个WebSocket协议的Java实现，感兴趣的朋友可以研究下：[Java-WebSocket](https://github.com/TooTallNate/Java-WebSocket)[5](https://www.itzhai.com/network/comprehend-the-underlying-principles-of-network-programming.html#fn:5)