计算机网络通讯的【系统性】扫盲——从“基本概念”到“OSI 模型”

文章目录

★本文的目标读者

今天这篇的标题是“扫盲”，也就是说：即使那些完全不懂 IT 领域，也不懂通讯领域的读者，依然能看懂（至少能看懂一部分）。为了做到这点，俺会尽量使用通俗的比喻，并适当加一些示意图。

另外，就算你已经比较了解网络通讯领域，本文中提到的某些部分，也可能是你所不知道的。也就是说：懂行的同学，看看此文，也会有帮助。

本文的标题特地强调了【系统性】——俺希望这篇教程能帮助读者对“计算机网络”这个领域进行系统性学习（何为“系统性学习”？请看这篇教程）

为了做到【系统性】这个目的，这篇教程很长。俺开博12年，这篇的长度估计能排到前5名。建议大伙儿慢慢看，不要着急。

★基本概念

为了足够通俗，俺先要介绍一些基本概念。

◇信道（channel ）

这是通讯领域非常基本的概念，肯定要先聊聊它。

通俗地说，信道就是“传送信息的通道”。

◇信道的类型

首先，信道可以从广义上分为“物理信道 & 逻辑信道”。

顾名思义，“物理信道”就是直接使用某种【物理介质】来传送信息；至于“逻辑信道”——是基于“物理信道”之上抽象出来的玩意儿（待会儿讲到“协议栈”的时候再聊）。

◇信道的带宽

“带宽”指的是：某个信道在单位时间内最大能传输多少比特的信息。

请注意：

电气领域 & 计算机领域都有“带宽”这个概念，但两者的定义不太一样。电气领域所说的“带宽”指的是“模拟带宽”，单位是“赫兹/Hz”；计算机领域所说的“带宽”指“数字带宽”，单位是“比特率”或“字节率”。

后续章节提到“带宽”，都是指计算机领域的术语。

◇带宽的单位——容易把外行绕晕

“比特率”或“字节率”很容易搞混淆。用英文表示的话——大写字母 B 表示【字节】；小写字母 b 表示【比

特】。

由于带宽的数字通常很大，要引入“K、M、G”之类的字母表示数量级，于是又引出一个很扯蛋的差异——“10进制”与“2进制”的差异。

【10进制】的 K 表示 1000；M 表示 1000x1000（1百万）

【2进制】的 K 表示 1024（2的10次方）；M 表示 1024x1024（2的20次方）

为了避免扯皮，后来国际上约定了一个规矩：对【2进制】的数量级要加一个小写字母 i。比如说：Ki 表示 1024；Mi 表示 1024x1024 ..... 以此类推。

举例：

1Kbps 表示“1000比特每秒”

1KiBps 表示“1024字节每秒”

◇信道的工作模式：单工 VS 半双工 VS 全双工

再来说说信道的工作模式。大致可以分为如下三种。为了让大伙儿比较好理解，俺对每一种都举相应的例子。

单工（simplex）

比如“电台广播”就是典型的【单工】。“电台”可以发信号给“收音机”，但“收音机”【不能】发信号给“电台”。

半双工（half-duplex）

比如“单条铁路轨道”，就是典型的【半双工】。火车在单条铁轨上，可以有两种运行方向；但对于同一个瞬间，只能选其中一个方向（否则就撞车了）。

全双工（full-duplex）

比如“光纤”就是典型的【全双工】。在同一根光导纤维中，可以有多个光束【同时相向】传播，互相不会干扰对方。

◇端点

为了叙述方便，俺把参与通讯的对象（主体）称作“通讯端点”，简称“端点”。

这里的“端点”是广义的，可以是硬件（比如某个网卡），也可以是软件（比如某个应用程序）。

◇单播、组播/多播、广播、选播

对于“网络通讯”，至少得有 N 个端点参与，并且【N ≥ 2】才有意义。

当 N 个端点构成一个网络，这时候就会涉及到“单播、组播、广播”这几个概念。

通俗地说：

单播（unicast）——发送给网络中的指定的【单个】端点

组播/多播（multicast）——发送给网络中的指定的【多个】端点

广播（broadcast）——发送给网络中的【所有】端点

选播（anycast）——发送给网络中随机选择的【单个】端点

◇通讯协议（protocol）

所谓的“通讯协议”就是：参与通讯的各方所采用的某种【约定】。只有大家都遵守这个约定，才有可能相互传递信息。

打个比方：如果两个人要用自然语言交流，前提是：双方使用相同（或相互兼容）的自然语言。

“通讯协议”就类似某种自然语言，参与通讯的多个端点，都必须能理解这个语言。

★从“分层”到“参考模型”

◇分层

在聊“分层”之前，先说说“分工”。比如在一个公司中，通常设有不同的工种/岗位，这就【分工】。

对于网络通讯也是如此，不太可能用一种通讯协议完成所有的信息传递任务（注：对于特别简单的网络，或许有可能只用单一协议；但如今的网络通讯已经很复杂，用【单个】通讯协议包办所有事情，已经不太可能）

一旦采用了多种通讯协议，这几种协议之间，该如何配合捏？

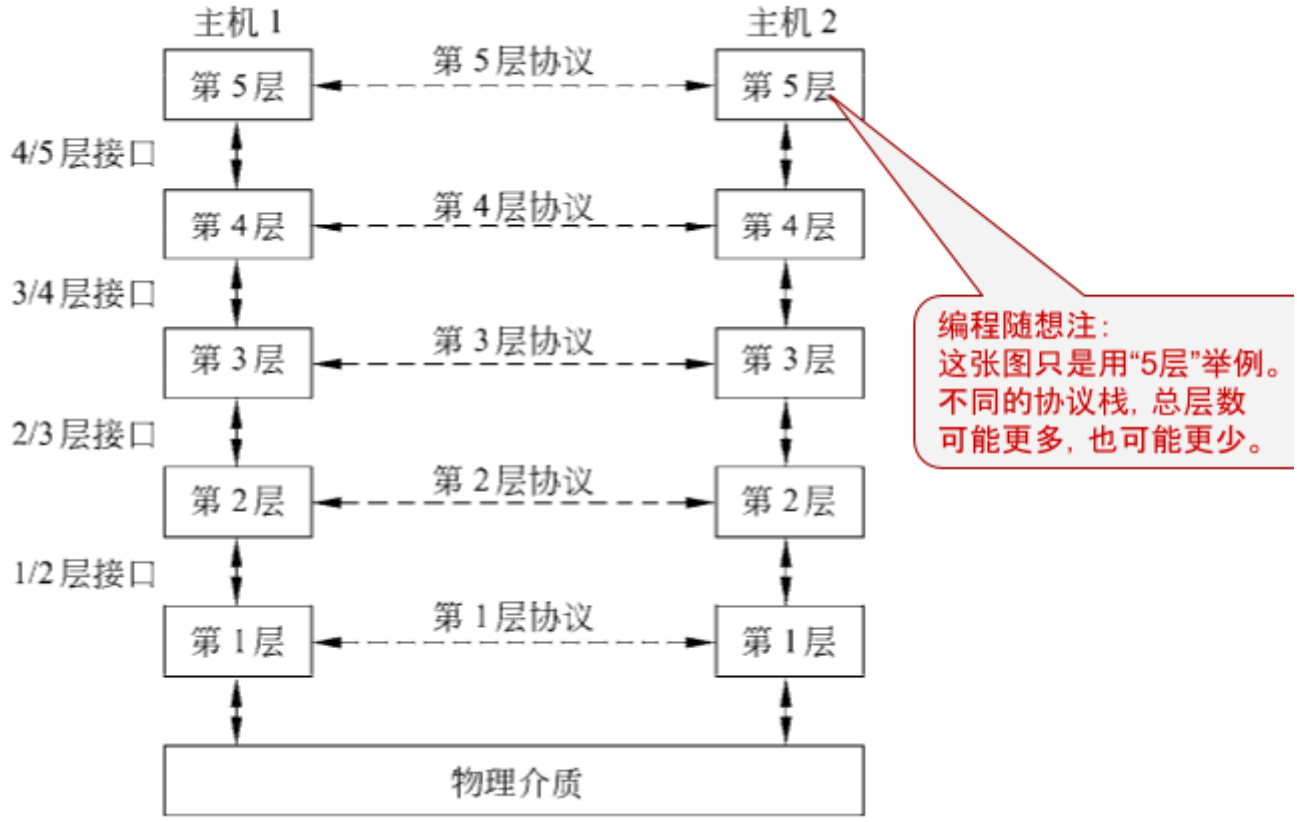
在网络通讯领域，采用的是【分层】的设计思路。多个层次的协议在一起协同工作，技术上称作“协议栈”

（洋文叫做“protocol stack”）。

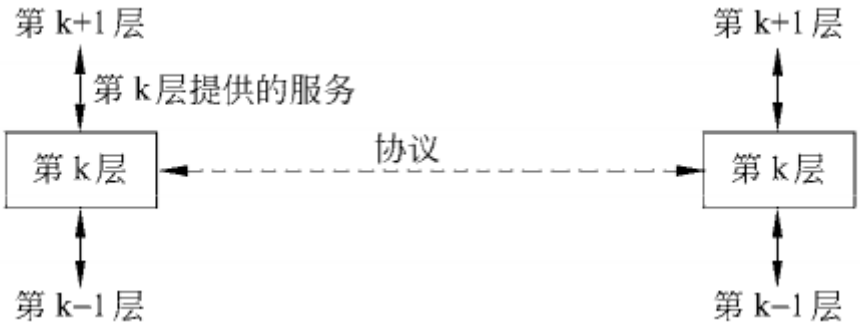
◇协议栈的原理

对于多层次的协议栈。每个层次都有各自的“端点”（进行通讯的主体）。处于【同一层次】的两个端点会使用该层次的协议进行通讯（注：同一个层次的协议，可能只有一个，也可能有多个）。

除了最顶层，每个层次的端点会向其【直接】上层提供“服务”；除了最底层，每个层次的端点会调用【直接】下层提供的“服务”（这里所说的“服务”指某种“编程接口”，技术行话叫 API）。



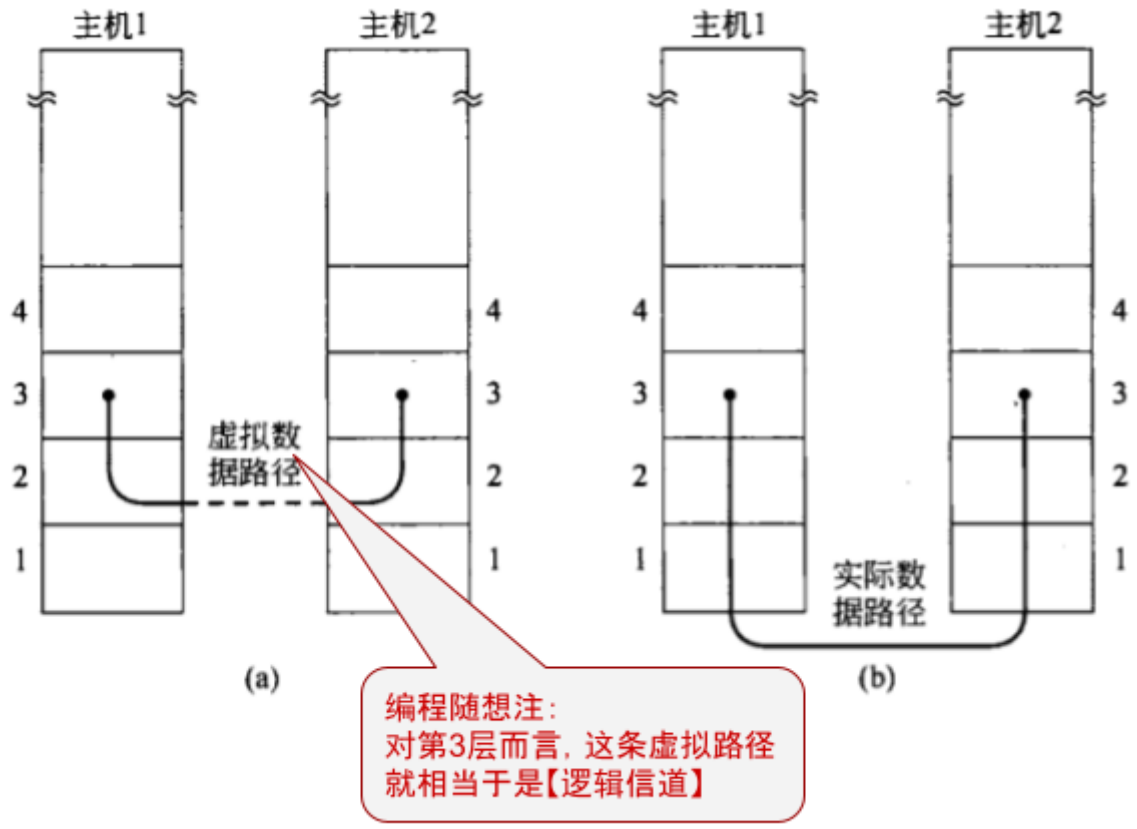
（“协议栈”的示意图）



（“服务”与“协议”之间的关系）

◇逻辑信道

（前一个小节说了）每个层次会向上一个层次提供服务（API 调用）。对上层而言，调用下层提供的 API 发送信息，其效果相当于在使用某种【信道】进行通讯，这也就是俺在 "★基本概念" 那个章节所说的“逻辑信道”。



（“逻辑信道”示意图）

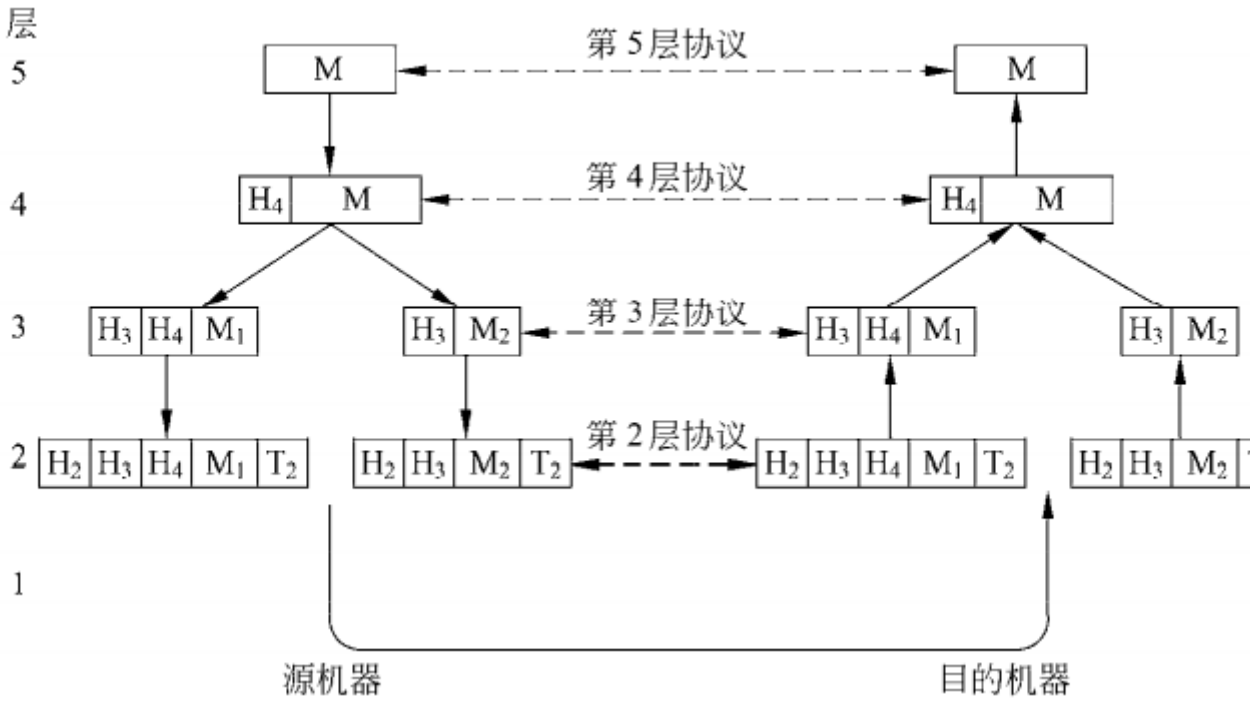
◇数据格式的原理

大部分协议会把要传送的数据切割为 N 份，每一份就是一个数据包。
通常来说，数据包的格式有如下三部分：

头部
身体（也称作“有效载荷”）
尾部（注：很多协议没有尾部）

如果你收过快递，可以把“网络数据包”与“快递包裹”作一个对照——
数据包的“头/尾”，就类似于快递包裹的【包装袋】。数据包的“身体”，就类似于快递包裹里面的东西。

对于【相邻】两层的协议，【下】层包含【上】层。也就是说：下层协议的【载荷】就是上层协议的【整体】。
还是以快递举例：
假设你从网上买了一台笔记本电脑。电脑出厂时，电脑厂商肯定会提供一个包装盒。快递公司在寄送这台笔记本的时候，又会在笔记本的盒子外面再加一个包装袋。对应到网络协议——“快递公司的包装袋”相当于【下层】协议；“电脑厂商的包装盒”，相当于【上层】协议。



（上下层协议的格式及包含关系）

◇网络分层的参考模型

上述所说的“分层 & 协议栈”只是一个抽象的（笼统的）思路。具体要分几层？每一层要干啥事儿？这些都是很有讲究滴！网络技术发展了几十年，已经有很多牛人提出了各种不同的划分方案，称之为“网络分层的参考模型”（为了打字省力，以下简称“模型”）。

在各种模型中，名气最大的当然是“OSI 模型”（洋文称作“OSI model ”）。在后续的章节中，俺会以这个模型为主体，进行介绍。

除了“OSI 模型”还有一个很出名的模型是“TCP/IP 模型”（因为互联网很成功，它才跟着出名）。

对“TCP/IP 模型”的分层，不同的文章或书籍，说法不太一样（“3层、4层、5层”皆有），这就引发了一些争议。包括几位热心读者也在博客留言，表达不同意见。为了避免一家之言，贴出维基百科的“这个链接 ”，其中给出了几种比较有名的说法。

另外，俺想提醒一下：

由于本文是基于【OSI 模型】进行展开。对于 TCP/IP 模型到底算几层，这方面的争论【不】影响本文后续的内容。

★OSI 概述

◇OSI 的历史

“OSI”的全称是“Open System Interconnection”。先说说它的历史。

上世纪70年代，“国际电信联盟”（ITU）想对各国的电信系统（电话/电报）建立标准化的规格；与此同时，“国际标准化组织”（ISO）想要建立某种统一的标准，使得不同公司制造的大型主机可以相互联网。

后来，这两个国际组织意识到：“电信系统互联”与“电脑主机互联”的性质差不多。于是 ISO 与 ITU 就决定合作，两家一起干。这2个组织的2套班子，从上世纪70年代开始搞，搞来搞去，搞了很多年，一直到1984年才终于正式发布 OSI 标准。

◇OSI 标准的两个组成部分

严格来讲，OSI 包括两大部分——
其一，抽象的概念模型，也就是前面提到的【OSI model】；
其二，针对这个概念模型的具体实现（具体的通讯协议），洋文叫做【OSI protocols】。

（前面说了）OSI 是由 ISO & ITU 联手搞出来滴。这两个国际组织里面的人，要么是来自各国的电信部门，要么是来自各国的高校学者。总而言之，既有严重的官僚风气，又有明显的学究风气。（正是因为这两种风气叠加，所以搞了很多年，才搞出 OSI）

OSI 的协议实现（OSI protocols），不客气地说，就是一堆垃圾——据说把 OSI protocols 所有的协议文档，全部打印成 A4 纸，摞起来得有一米多高！是不是很吓人？协议搞得如此复杂，严重违背了 IT 设计领域的 KISS 原

则 。

由于 OSI protocols 实在太复杂，后来基本没人用。但 OSI model 反而广为流传，并且成为“网络分层模型”中名气最大，影响力最广的一个。

因此，本文后续章节中，凡是提到 OSI，指的是【OSI model】。

◇OSI 模型的7层

OSI 模型总共分7层，示意图参见如下表格：

层次	中文名	洋文名
第7层	应用层	Application Layer
第6层	表示层	Presentation Layer
第5层	会话层	Session Layer
第4层	传输层	Transport Layer
第3层	网络层	Network Layer
第2层	数据链路层	Data Link Layer
第1层	物理层	Physical Layer

（注：为了打字省力，在后续章节把“数据链路层”直接称为“链路层”）

考虑到本文是针对一般性读者的【扫盲教程】，俺重点聊第1~4层。搞明白这几个层次之后，有助于你更好地理解网络的很多概念，也有助于你更好地理解很多信息安全的概念。

网上已经有很多关于 OSI 的文章，可惜大部分写得粗糙——很多文章只是在照抄定义。

俺曾经写过一篇《学习技术的三部曲：WHAT、HOW、WHY》，其中提到【理解技术】的不同层次。要想更好地理解 OSI 模型，你得搞明白：为啥需要引入某某层？（请注意：这是一个 WHY 型的问题）

接下来在讨论 OSI 的每个层次时，俺都会专门写一个小节，谈该层次的【必要性】。搞明白【必要性】，你就知道为啥要引入这个层次。

★物理层：概述

◇物理层的必要性

通俗地说：直接与物理介质打交道的层次，就是物理层。这一层的必要性比较明显。

因为所有的通讯，归根结底都要依赖于【物理介质】。与物理介质打交道，需要牵涉到很多与【物理学】相关的东东。比如：“无线电通讯”需要关心“频率/波长”；电缆通讯需要跟“电压”打交道；“光纤通讯”需要关心“玻璃的折射率＆光线的入射角” ……

“物理层”的主要职责是：屏蔽这些细节，使得“物理层”之上的层次不用再去操心物理学。

◇物理信道的类型

何为“物理信道”，在本文开篇的“基本概念”已经提到了。

对于“物理信道”，还可以进一步细分为如下三大类：

1. 有线信道（比如：双绞线、同轴电缆、光纤、等等）
2. 无线信道（比如：微波通讯、电台广播、卫星通讯、等等）
3. 存储信道

“存储信道”比较少见，很多人没听说过，稍微解释一下。

假设你要把一大坨信息传送给另一个人，除了用“有线 or 无线”这两种通讯方式，还可以把信息先保存到某种【存储介质】（比如硬盘），然后再把存储介质用某种方式（比如快递）转交给对方。这就是所谓的“存储信道”。

◇信噪比 (Signal-to-noise ratio)

俺在很多篇关于“学习＆心理学”的博文中提到过【信噪比】这个概念。其实这个概念是从通讯领域借用的术语。

对于“物理信道”，总是会存在某些环境干扰，称之为“噪声”（Noise）。“信道传输的有用信息”与“无用的干

扰噪声”，这两者的比值就是“信噪比”。

“信噪比”单位是【分贝】。“分贝”洋文叫做“decibel ”（简写为 dB）。“deci”表示“十进制”；“bel”是为了纪念大名鼎鼎的贝尔（电话它爹）。

◇带宽的限制因素

“物理信道”要依赖于物理传输介质。不管使用何种物理介质，都要受限于某些基本的物理学定律（比如“光速上限”）。另外，不管何种物理介质，总是会有或多或少的环境干扰（噪声）。这两个因素导致了：任何“物理信道”的最大传输率总是有限滴。

由于物理层是最底下的一层，物理层之上的其它层次总是要直接或间接地依赖【物理信道】。因此，其它层次建立的“逻辑信道”，其带宽只会比“物理信道”的最大带宽更小。换句话说：“物理信道”的带宽上限也就是整个协议栈的带宽上限。

◇多路复用 (Multiplexing)

一般来说，凡是能实现【长距离】通讯的“物理信道”，都有相当的经济成本。比如铺设“光纤、同轴电缆”都要花钱。无线电通讯虽然免去了铺设线路的成本，但需要竞标购买频段。因此，物理信道非常强调“多路复用”。

所谓的“多路复用”，通俗地说就是：尽可能地共享物理信道，不要浪费了。

“多路复用”有很多种类型；不同的类型，原理也不同。为了展示各种不同的原理，俺拿【无线通信】来说事儿。

无线通信领域的“多路复用”，【至少】有如下几种：

频分多路复用/FDM（Frequency-Division Multiplexing）

这个最简单，就是根据频率拆分。不同的线路占用不同的频段，互不干扰。（电台广播用的就是这个思路）

但这个思路的缺点很明显——

其一，要依赖足够宽的频段（频段是稀缺资源）；

其二，不同线路的流量可能会动态变化。如果某个线路空闲，其占用的频段就浪费了。

（注：光纤通讯中有个“波分多路复用/WDM”，本质上就是 FDM）

时分多路复用/TDM（Time-Division Multiplexing）

这种思路只用一个很窄的频段。为了在同一个频道发送多个信道，采用【分时机制】，把时间切割成很小的时间片，每个线路占用一个时间片。周而往复。

这个思路有点像十字路口的红绿灯——每隔一段时间，其中一条路可以通行。

这个思路的优点是：可以只使用一个很窄的频段。缺点是：线路越多，每条线路等待越久；即使某个线路空闲，依然会占用时间片（浪费了资源）。

码分多路复用/CDM（Code-Division Multiplexing）

这种思路采用某种【编码】的技巧，使得多个端点可以在同一个时间点使用同一频段发送数据；由于他们采用不同的编码方式，不会相互干扰。

一般来说，CDM 要依赖于“扩频技术”（spread spectrum ），需占用一个比较宽的频道范围。这算是缺点。但其优点很明显——

其一，可以支持 N 个线路（N 动态变化）；

其二，即使任何一个线路的流量动态变化，也不会浪费物理信道的资源。

显然，这种思路明显优于 FDM & TDM。如今在移动通讯领域大名鼎鼎的 CDMA（码分多址），采用的就是这个思路。

★物理层：具体实例

◇物理层的【协议】

物理层的协议主要有如下：

USB 协议

蓝牙协议

IEEE 802.11 的一部分（Wi-Fi）

IEEE 802.16 （WiMAX）

IEEE 1394 （火线接口）

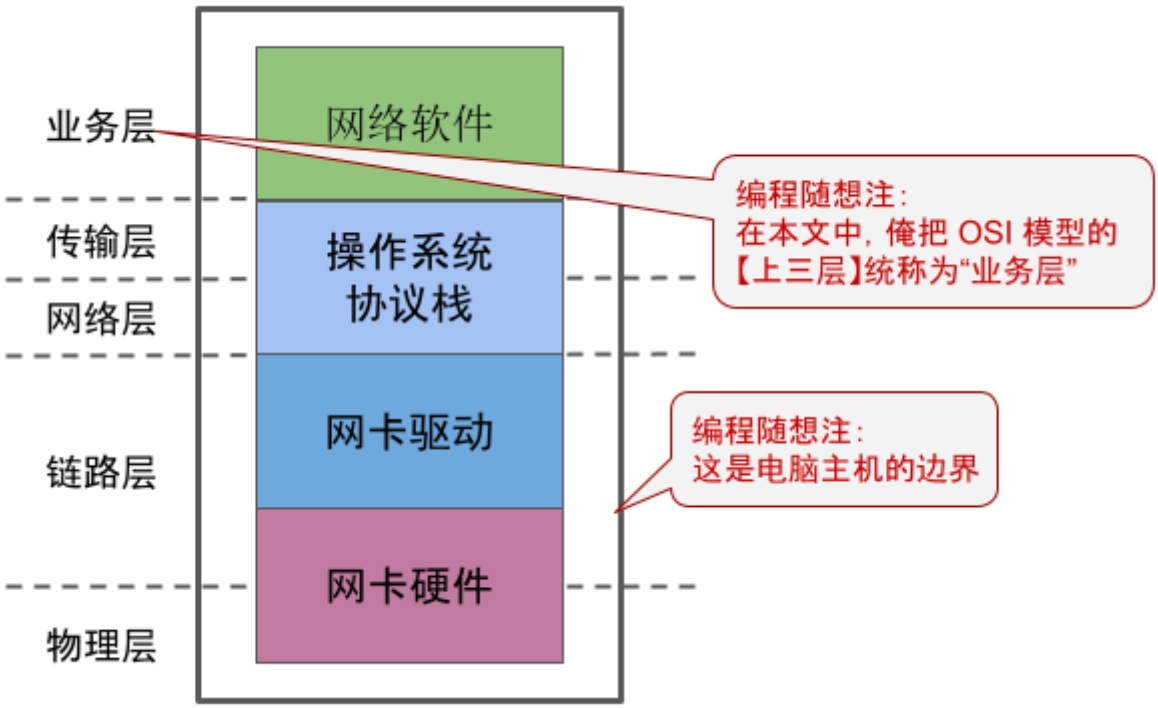
RS-232 协议 （串行接口/串口）

……

（考虑到篇幅）俺不可能具体细聊这些协议，只是贴出每个的维基百科链接，感兴趣的同学自己点进去看。

◇物理层的【协议实现】

对于电脑主机（含移动设备），“网卡硬件”包含了物理层的协议实现（参见如下示意图）
另外，还有一些专门的【1层】网络设备，也提供物理层的功能（参见下一个小节）。



（OSI 模型中，不同层次的协议实现）

◇物理层相关的【网络设备】

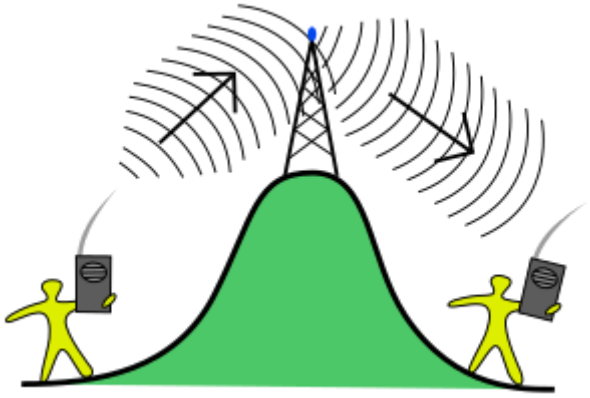
调制解调器　（modem）
通俗地说，“调制解调器”就是用来翻译“数字信号 & 模拟信号”。
在发送信息时，modem 把电脑要发送的“字节流”（数字信号）翻译成“模拟信号”，然后通过物理介质发送出去；当它从物理介质收到“模拟信号”，再翻译成“数字信号”，传回给电脑。
早期的拨号上网，modem 面对的物理介质是“固话线路”；如今家庭宽带普及，光纤入户，modem 面对的物理介质是“光纤线路”。

（老式 modem，用于固定电话线路）

中继器　（repeater）
信号在物理介质中传输，会出现【衰减】（不论是“有线 or 无线”都有可能衰减）。“中继器”的作用是【信号

增益】，使得信号能传得更远。

另外，比如“微波通讯”是直线传播，而地球表面有弧度，还有地形的起伏。所以每隔一定距离要建“微波塔”。这玩意儿也相当于“中继器”。



（微波塔示意图）

集线器　(hub)

可以把“集线器”视作更牛逼的“中继器”——“中继器”只有两个口（只能连接两个通讯端点），而“集线器”有多个口（同时连接多个通讯端点）。

通常所说的“集线器”是指“以太网集线器”。这种设备如今已经逐步淘汰，很少见到了。

（老式的10兆以太网集线器）

另外，很多同学应该都用过“USB hub”，就是针对 USB 线的“集线器”（“USB 线”也可以视作某种通讯介质）。

★链路层：概述

◇链路层的必要性

对信息的打包

物理层传输的信息，通俗地说就是【比特流】（也就是一长串比特）。但是对于计算机来说，“比特流”太低级啦，处理起来极不方便。“链路层”要干的第一个事情，就是把“比特流”打包成更大的一坨，以方便更上层的协议进行处理。在 OSI 模型中，链路层的一坨，称之为“帧”（frame）。

差错控制

物理介质的传输，可能受到环境的影响。这种影响不仅仅体现为“噪声”，有时候会出现严重的干扰，导致物理层传输的“比特流”出错（某个比特“从0变1”或“从1变0”）。因此，链路层还需要负责检查物理层的传输是否出错。在 IT 行话中，检测是否出错，称之为“差错控制机制”（后面有一个小节会简单说一下这个话题）。

流量控制

假设两个端点通过同一个物理信道进行通讯，这两个端点处理信息的速度可能不同。如果发送方输出信息的速度超过接收方处理信息的速度，通讯就会出问题。于是就需要有某种机制来协调，确保发送方的发送速度不会超出接收方的处理速度。在技术行话中，这称之为“流量控制”，简称“流控”。

信道复用

在上一个章节已经讲到：用于远距离通讯的“物理介质”，总是有成本。因此需要对物理信道进行“多路复用”，就会导致多个端点共用同一个物理信道。如果同时存在多个发送者和多个接收者。接收者如何知道某个信息是发给自己而不是别人？

另外，某些物理介质可能不支持并发（无法同时发送信息）。某些物理介质可能是【半双工】，所有这些物理层的限制，都使得“多路复用”变得复杂。为了解决这些问题，链路层需要提供了某种相应的机制（协议），术语叫做“介质访问控制”（洋文是“Media Access Control”，简称 MAC）。后续小节会聊它。

◇差错控制

为了发现传输的信息是否出错，设计了很多相应的数学算法。这些算法大体分为两类：“检错算法 & 纠错算法”。

简而言之，“检错算法”只能检测出错误，而“纠错算法”不但能检测出错误，还能纠正错误。很显然，“纠错算法”更牛逼，但是它也更复杂。

常见的“检错算法”对传输的数据计算出一个【校验值】，接收方收到数据会重新计算校验和，如果算出来不对，就把收到的数据丢弃，让对方重发。“校验算法”的原理类似于《扫盲文件完整性校验——关于散列值和数字签名》一文中提到的“散列算法/哈希算法”。

“纠错算法”更高级，由于涉及到更多数学，俺就不展开啦。

对于【无线】物理信道，由于出错的概率更高，并且重新传输数据的成本也更高。所以【无线】通讯的链路层协议，更倾向于用【纠错】机制；作为对比，【有线】通讯的链路层协议，更倾向于用【检错】机制。

◇MAC 协议

“MAC 协议”用来确保对下层物理介质的使用，不会出现冲突。为了形象，俺拿“铁路系统”来比喻，说明“MAC 协议”的用途。

假设有一条【单轨】铁路连接 A/B 两地。有很多火车想从 A 开到 B，同时还有很多火车想从 B 开到 A。

首先，要确保不发生撞车（如果已经有车在 A 开往 B 的途中，那么 B 就不能再发车）；其次，即使是同一个方向的车，出发时间也要错开一个时间间隔。

所有这些协调工作，都是靠“MAC 协议”来搞定。

◇MAC 地址

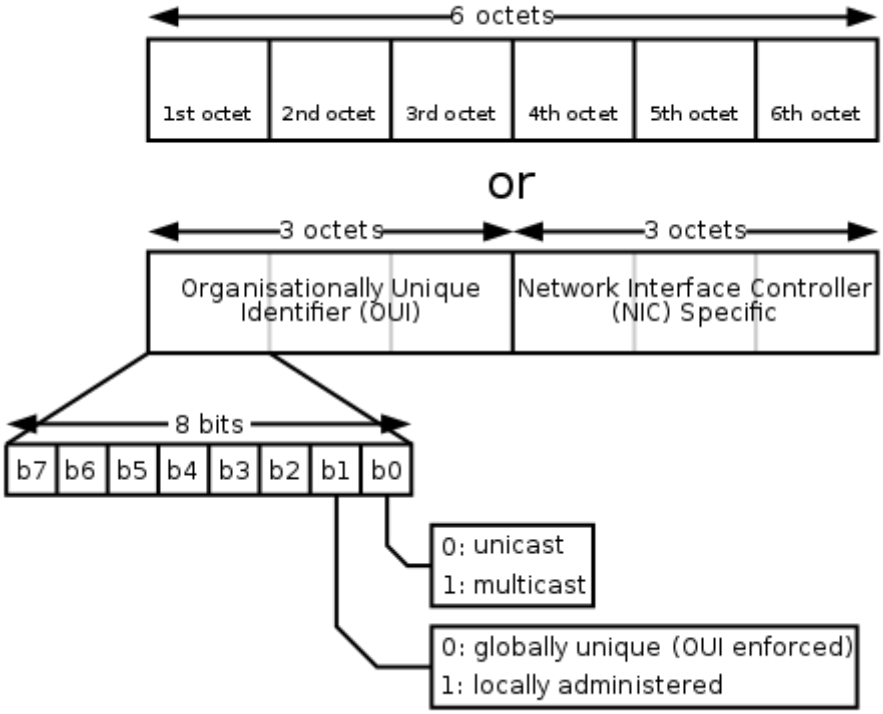
为了完成上述任务，光有“MAC 协议”还不够，还需要为每一个端点引入【惟一的】标识。这个标识就称作“MAC 地址”。

通俗地说，每个网卡都内置了一个“MAC 地址”。这个地址是网卡在出厂的时候就已经设置好的，并且用某种机制确保该地址【全球唯一】。

如何保证 MAC 地址全球唯一捏？简单说一下：

MAC 地址包含6个字节（48个比特），分为两半。第一部分称作【OUI】，OUI 的24个比特中，其中2个比特有特殊含义，其它22个比特，用来作为网卡厂商的唯一编号。这个编号由国际组织 IEEE 统一分配。

MAC 地址第二部分的24比特，由网卡厂商自己决定如何分配。每个厂商只要确保自己生产的网卡，后面这24比特是唯一的，就行啦。



（MAC 地址的构成）

由于俺在很多安全教程中鼓吹大伙儿使用“操作系统虚拟机”，再顺便说说【虚拟网卡】的 MAC 地址。

“虚拟网卡”是由【虚拟化软件】创建滴。IEEE 也给每个虚拟化软件的厂商（含开源社区）分配了唯一的 OUI。因此，虚拟化软件在创建“虚拟网卡”时，会使用自己的 OUI 生成前面24个比特；后面的24比特，会采用某种算法使之尽可能【随机化】。由于“2的24次方”很大（2^{24} = 16777216），碰巧一样的概率很低。

（注：如果手工修改 MAC 地址，故意把两块网卡的 MAC 地址搞成一样，那确实就做不到唯一性了。并且会导致链路层的通讯出问题）

★链路层：具体实例

◇链路层的【协议】

链路层的协议主要有如下：

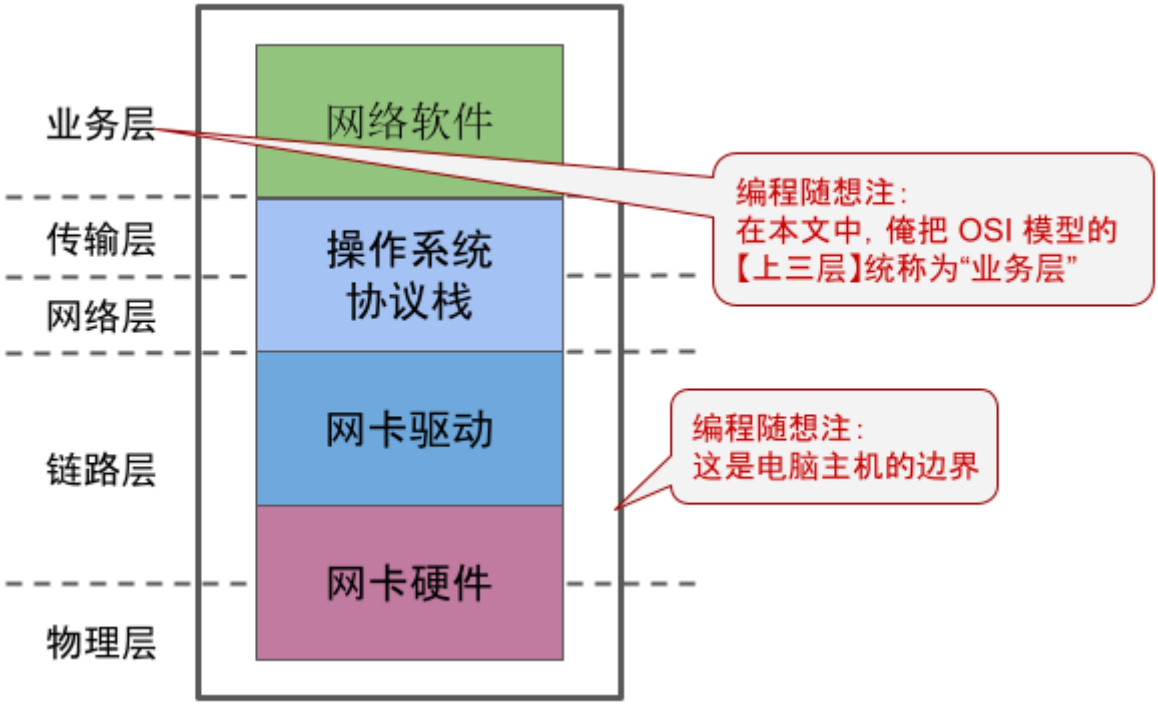
MAC 协议　（介质访问控制）
LLC 协议　（逻辑链路控制）
ARP 协议　（解析 MAC 地址）
IEEE 802.3　（以太网）
IEEE 802.11　的一部分（Wi-Fi）
L2TP 协议　（2层VPN）
PPP 协议　（拨号上网）
SLIP 协议　（拨号上网）
……

（考虑到篇幅）俺不可能具体细聊这些协议，只是贴出每个的维基百科链接，感兴趣的同学自己点进去看。

◇链路层的【协议实现】

对于电脑主机（含移动设备），“网卡硬件 & 网卡驱动”会包含链路层协议的实现（参见如下示意图）。

另外，还有一些专门的【2层】网络设备，也提供链路层的功能（参见下一个小节）。



（OSI 模型中，不同层次的协议实现）

◇链路层相关的【网络设备】

网络交换机　(network switch)

（注：一般提到“网络交换机”，如果不加定语，指的就是“2层交换机”；此外还有更高层的交换机，在后续章节介绍）

为啥要有交换机捏？俺拿“以太网的发展史”来说事儿。

以太网刚诞生的时候，称之为“经典以太网”，电脑是通过【集线器】相连。“集线器”前面提到过，工作在【1层】（物理层），并不理解链路层的协议。因此，集线器的原理是【广播】模式——它从某个网线接口收到的数据，会复制 N 份，发送到其它【每个】网线接口。假设有4台电脑（A、B、C、D）都连在集线器上，A 发数据给 B，其实 C & D 也都收到 A 发出的数据。显然，这种工作模式很傻逼（低效）。由于“经典以太网”的工作模式才“10兆”，所以集线器虽然低效，还能忍受。

后来要发展“百兆以太网”，再用这种傻逼的广播模式，就不能忍啦。于是“经典以太网”就发展为“交换式以太网”。用【交换机】代替“集线器”。

交换机是工作在2层（链路层）的设备，能够理解链路层协议。当交换机从某个网线接口收到一份数据（链路层的“帧”），它可以识别出“链路帧”里面包含的目标地址（接收方的 MAC 地址），然后只把这份数据转发给“目标 MAC 地址相关的网线接口”。

由于交换机能识别2层协议，它不光比集线器的性能高，而且功能也强得多。比如（稍微高级点的）交换机可以实现“MAC 地址过滤、VLAN、QoS”等多种额外功能。

网桥/桥接器　(network bridge)

“交换机”通常用来连接【同一种】网络的设备。有时候，需要让两台不同网络类型的电脑相连，就会用到【网桥】。

下面以“操作系统虚拟机”来举例（完全没用过虚拟机的同学，请跳过这个举例）。

在这篇博文，俺介绍了虚拟机的几种“网卡模式”，其中有一种模式叫做【bridge 模式】。一旦设置了这种模式，Guest OS 的虚拟网卡，对于 Host OS 所在的外部网络，是【双向】可见滴。也就是说，物理主机所在的外部网络，也可以看见这块虚拟网卡。

现在，假设你的物理电脑（Host OS）只安装了【无线网卡】（WiFi），而虚拟化软件给 Guest OS 配置的通常是【以太网卡】。显然，这是两种【不同】的网络。为啥 Guest OS 的以太网卡设置为“bridge 模式”之后，外部 WiFi 网络可以看到它捏？

奥妙在于——虚拟化软件在内部悄悄地帮你实现了一个“网桥”。这个网桥把“Host OS 的 WiFi 网卡”与“Guest OS 的以太网卡”关联起来。WiFi 网卡收到了链路层数据之后，如果接收方的 MAC 地址对应的是 Guest OS，网桥会把这份数据丢给 Guest OS 的网卡。

这种网卡模式之所以称作“bridge 模式”，原因就在于此。

◇链路层相关的【软件工具】

嗅探抓包工具（Sniffer）

要了解链路层的数据包结构，需要用到“嗅探工具”。这类工具能捕获流经你网卡的所有【链路层】数据包。

前面聊“协议栈”的时候说过：下层数据包的载荷就是上层数据包的整体。因此，拿到【链路层】数据包也就意味着：你已经拿到2层之上的所有数据包的信息了。

有些抓包工具自带图形界面，可以直接显示数据包的内容给你看。还有些只提供命令行（只是把获取的数据包保存为文件），然后要搭配其它图形化的工具来展示数据包的内容。

抓包的工具有很多，名气最大的是 Wireshark （原先叫做 Ethereal）。

ARP 命令

首先，ARP 是“MAC 地址解析协议”的洋文名称。该协议根据“IP 地址”解析“MAC 地址”。

Windows 自带一个同名的 arp 命令，可以用来诊断与“MAC 地址”相关的信息。比如：列出当前子网中其它主机的 IP 地址以及对应的 MAC 地址。这个命令在 Linux & Mac OS 上也有。

★网络层：概述

◇网络层的必要性

路由机制（routing）

在 OSI 模型中，链路层本身【不】提供路由功能。你可以通俗地理解为：链路层只处理【直接相连】的两个端点（注：这么说不完全严密，只是帮助外行理解）

对于某个复杂网络，可能有很多端点，有很复杂的拓扑结构。当拓扑足够复杂，总有一些端点之间【没有直连】。那么，如何在这些【没有直连】的端点之间建立通讯捏？此时就需要提供某种机制，让其它端点帮忙转发数据。这就需要引入“路由机制”。

为了避免把“链路层”搞得太复杂，路由机制放到“链路层”之上来实现，也就是“网络层”。

基于【路由】的地址编码方式

链路层已经提供了某种全球唯一的地址编码方式（MAC 地址）。但“MAC 地址”有如下几个问题：
其一，它是固定的（虽然可以用技术手段去修改 MAC 地址，但很少这么干）
其二，MAC 地址的编码是基于【厂商】，无法体现网络拓扑结构。或者说，“MAC 地址”对于“路由机制”是不够友好滴。

因此，需要引入一种更抽象（更高层）的地址，也就是“网络层地址”。咱们常说的“IP 地址”，是“网络层地址”的实现方式之一。

为了帮你理解，举个例子：

每个人都有身份证号（这就类似于“MAC 地址”）。当某人加入了某个公司，公司会为此人再分配一个“员工号”（这就类似于“网络地址”）。既然有身份证号，为啥公司还要另搞一套“员工编号”捏？因为“员工编号”有额外的好处。比如说：可以把员工号划分为不同的区间，对应不同的部门。这样一来，只要看到员工号，就知道此人来自哪个部门。

类似道理，每个网卡都有自己固定的 MAC 地址，当这个网卡接入到不同的网络，每次都可以再分配不同的“网络地址”。通过“网络地址”可以看出这个网卡属于哪个网络（对路由比较方便）。

网际互联（internetwork ）

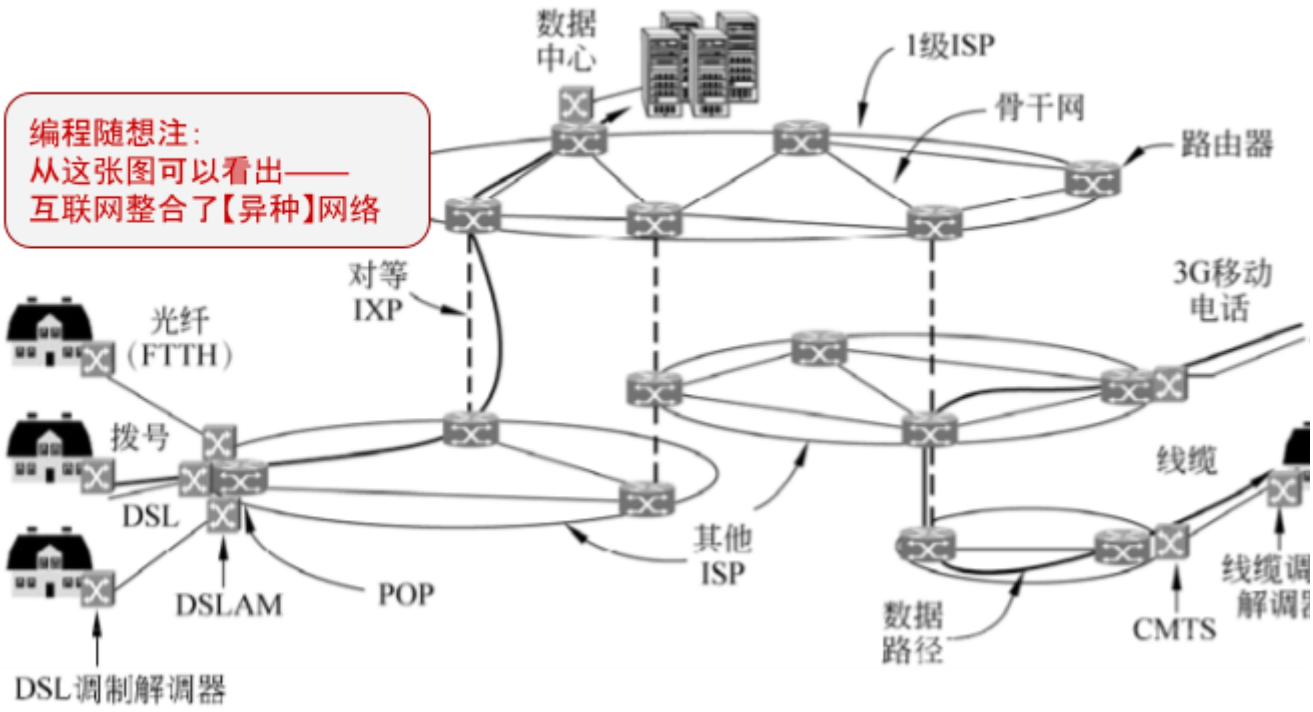
引入“网络层”的另一个目的是：屏蔽不同类型的网络之间的差异，从而有利于【网际互联】（也就是建立“网络的网络”）。

一般来说，要想联通【异种】网络，就要求每个网络中都有一台主机充当【网关】（gateway）。【网关】起到“中介/翻译”的作用——帮不同的网络翻译协议，使得不同的网络可以互相联通。

假设【没有】统一的网络层，网关的工作就很难做。就好比说：如果全球没有某种通用的自然语言，就需要培养非常多不同类型的翻译人才（假设有30种主要语言，任意两种互译，就需要几百种不同的翻译人才）。

反之，如果有了某种统一的网络层标准，问题就好办多了（还是假设有30种主要语言，只要选定某种作为通用语，然后培养29种翻译人才，就可以实现任意两种语言互译）。

如今的互联网时代，【IP 协议】就是那个充当统一标准的网络层协议。

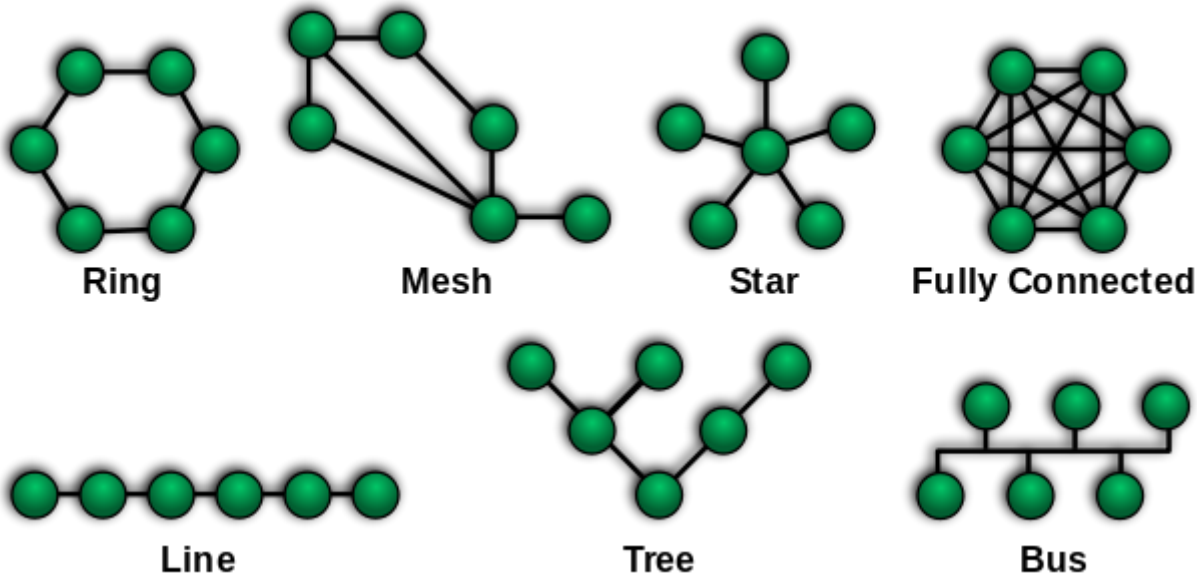


（互联网整合了各种类型的网络）

◇网络拓扑（network topology）

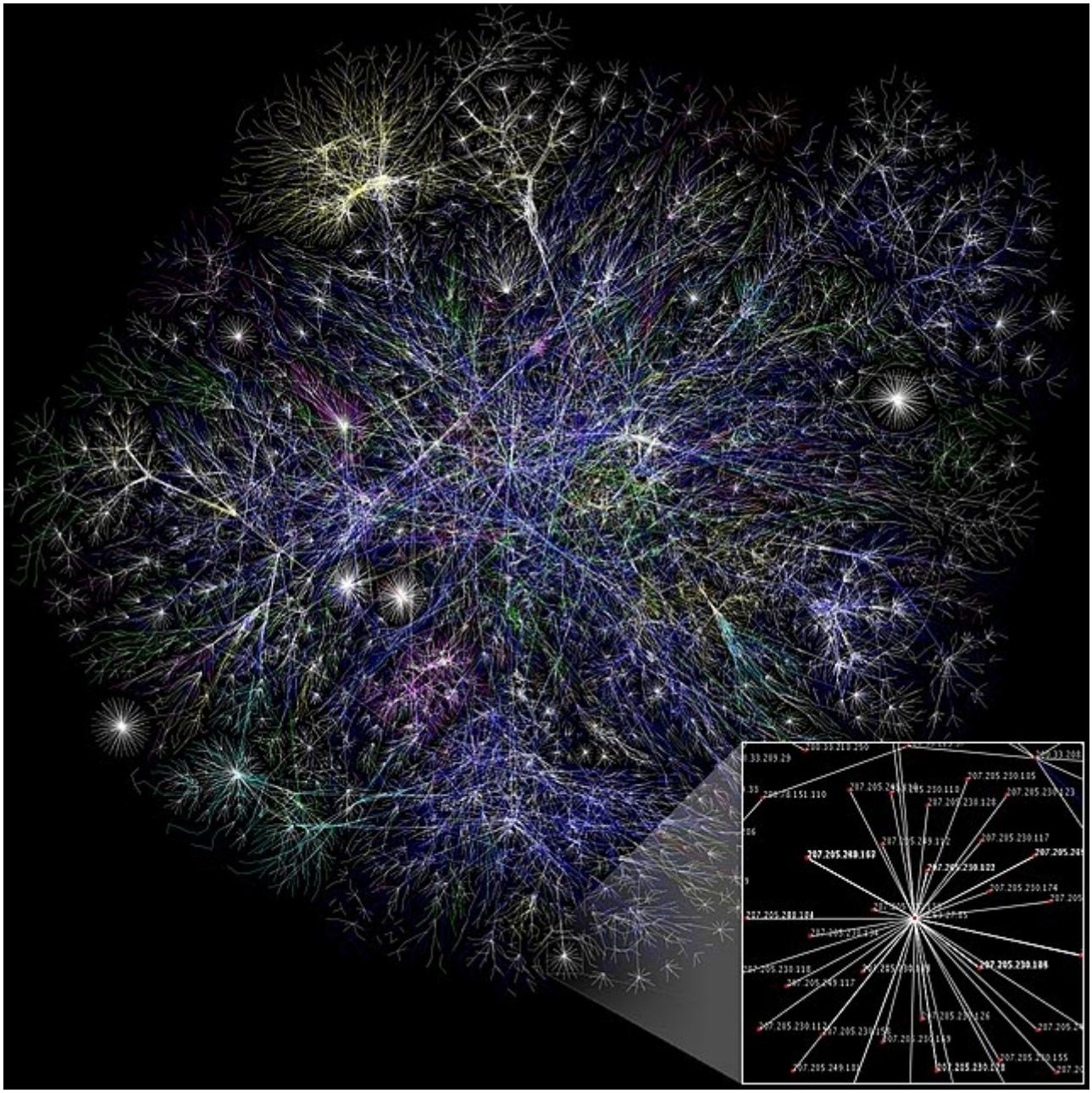
网络的拓扑结构有很多种，有简单的，有复杂的。一般来说，再复杂的拓扑，也可以逐步分解为若干简单拓扑的组合。

对拓扑的研究，有专门一个数学分支（拓扑学）。考虑到本文只是扫盲，俺不可能再去聊“拓扑学”。因此，只挑几种简单的拓扑结构，让大伙儿有个直观的印象。



（常见的网状拓扑结构：星形拓扑、环形拓扑、总线拓扑、网状拓扑、等等）

如今的互联网，整体的拓扑结构超级复杂。但还是可以逐步分解为上述几种基本的拓扑结构。



（互联网的复杂拓扑，右下角是图中某个小点的放大。
为节省大伙儿的翻墙流量，俺贴的是缩小图。点“这里　”看原始图）

◇互联网的拓扑——从“历史”的角度看其健壮性

从上面那张图可以看出：互联网拓扑的【局部】有很多是“星形拓扑”（当然也有其它的）。但从【宏观】上看，更像是“网状拓扑”。

在现实生活中，对于复杂结构，通常都会采用“树状层次结构”，以便于管理。比如：域名系统、公司组织结构、官僚系统 …… 那为啥互联网的【宏观】拓扑结构是“网状”捏？这就要说到互联网的历史。

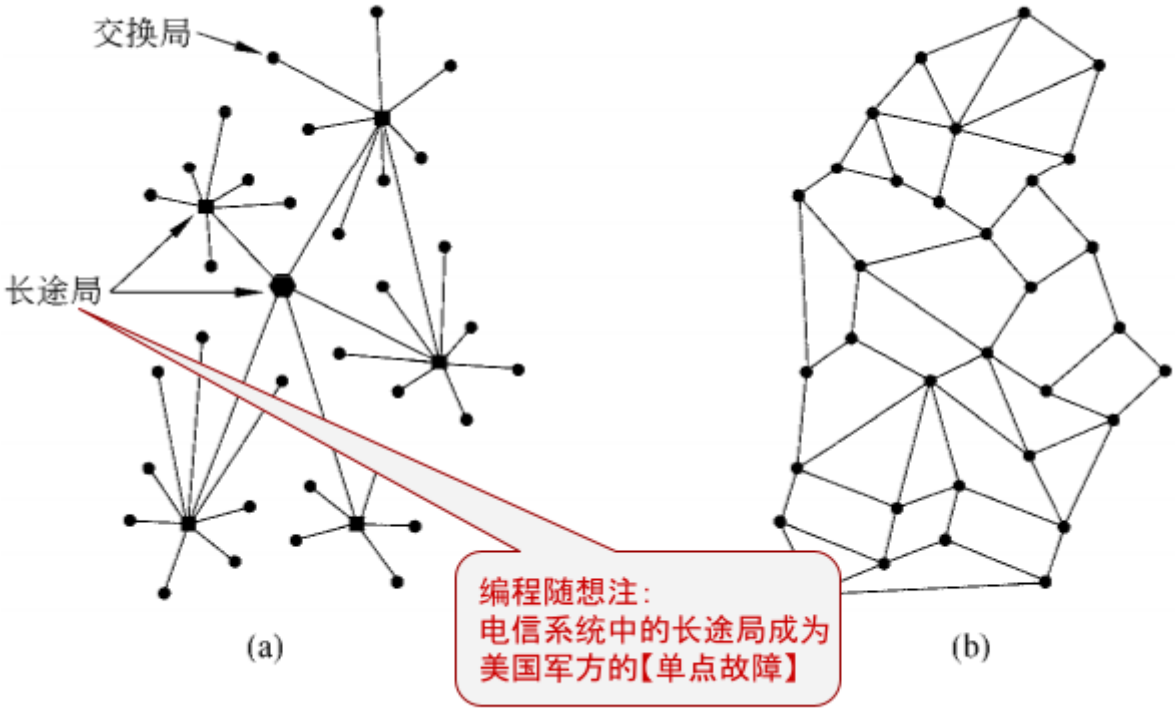
在上世纪50年代（冷战高峰期），美国军方的指挥系统高度依赖于电信公司提供的电话网络。当时的电话网络大致如下——
在基层，每个地区有电话交换局，每一部电话都连入当地的交换局。
在全国，设有若干个长途局，每个交换局都接入某个特定的长途局（不同地区的交换局通过长途局中转）。

简而言之，当时美国的电话网络是典型的【多级星形拓扑】。这种拓扑的优点是：简单、高效、便于管理；但缺点是：健壮性很差。从这个案例中，大伙儿可以再次体会到“效率”与“健壮性”之间的矛盾。俺写过一篇很重要的博文（这里）深入讨论了这个话题。

话说1957年的时候，苏联成功试射第一颗洲际弹道导弹（ICBM），美国军方开始担心：一旦苏联先用洲际导弹攻击美国，只要把少数几个长途局轰掉，军方的指挥系统就会瘫痪。也就是说，“长途局”已经成为美国军方的【单点故障】（何为“单点故障”？参见这篇博文）。

1960年，美国国防部找来大名鼎鼎的兰德公司进行咨询，要求提供一个应对核打击的方案。该公司的研究员Paul Baran　设计了一个方案，把“星形拓扑”改为【网状拓扑】。采用【网状拓扑】的好处在于：即使发生全面

核大战，大量骨干节点被摧毁，整个网络也不会被分隔成几个孤岛，军方的指挥系统依然能正常运作。



（左边：互联网诞生前——美国的电话网络　　右边：兰德公司的“Baran 方案”）

有了兰德公司的方案，美国军方找到当时最大的电信公司 AT＆T，想要实现这个系统，结果被否决了。AT＆T 高层认为：搞这样一种系统根本不切实际。于是 Baran 的方案中途夭折。

为啥 AT＆T 反对这个方案捏？一方面，成功的大公司总是有很强的思维定势（关于这点，参见这篇文章）；另一方面，Baran 的设计方案确实很超前——其前瞻性不仅包括“拓扑结构”，而且把当时电信行业的几大核心观念完全颠覆掉了（具体如何颠覆，后续章节还会再聊）。

时间一晃又过了好多年，到了60年代末，由于一系列机缘巧合，英国佬发现了“Baran 方案”的价值，并据此搞了一个小型的 NPL 网络（NPL 是“国家物理实验室”的缩写）。然后在某次 ACM 会议上，美国佬看到英国佬的论文，才意识到：Baran 方案完全可行。经历了“出口转内销”的命运之后，该方案重新被美国国防部重视。之后，（国防部下属的）“高级计划研究局　”（ARPA）开始筹建“阿帕网”（ARPANET），才有了如今的互联网。

◇路由的大致原理

聊完“拓扑”，再来聊“路由”。

当主机 A 向主机 B 发送网络层的数据时，大致会经历如下步骤：

1.
A 主机的协议栈先判断“A B 两个地址”是否在同一个子网（“子网掩码”就是用来干这事儿滴）。
如果是同一个子网，直接发给对方；如果不是同一个子网，发给本子网的【默认网关】。
（此处所说的“网关”指“3层网关/网络层网关”）

2.
对于“默认网关”，有可能自己就是路由器；也可能自己不是路由器，但与其它路由器相连。
也就是说，“默认网关”要么自己对数据包进行路由，要么丢给能进行路由的另一台设备。
（万一找不到能路由的设备，这个数据就被丢弃，于是网络通讯出错）

3.
当数据到达某个路由器之后，有如下几种可能——

3.1
该路由器正好是 B 所在子网的网关（与 B 直连），那就把数据包丢给 B，路由过程就结束啦；

3.2
亦或者，路由器会把数据包丢给另一个路由器（另一个路由器再丢给另一个路由器） …… 如此循环往复，最终到达目的地 B。

3.3
还存在一种可能性：始终找不到“主机 B”（有可能该主机“断线 or 关机 or 根本不存在”）。为了避免数据包长时间在网络上闲逛，还需要引入某种【数据包存活机制】（洋文叫做“Time To Live”，简称 TTL）。
通常会采用某个整数（TTL 计数）表示数据包能活多久。当主机 A 发出这个数据包的时候，这个“TTL 计数”就已经设置好了。每当这个数据包被路由器转发一次，“TTL 记数”就减一。当 TTL 变为零，这个数据包就死了（被丢

弃）。

对于某些大型的复杂网络（比如互联网），每个路由器可能与其它 N 个路由器相连（N 可能很大）。对于上述的 3.2 情形，它如何判断：该转发给谁捏？

这时候，“路由算法”就体现出价值啦——

一般来说，路由器内部会维护一张【路由表】。每当收到一个网络层的数据包，先取出数据包中的【目标地址】，然后去查这张路由表，看谁距离目标最近，就把数据包转发给谁。

上面这段话看起来好像很简单，其实路由算法挺复杂滴。考虑到本文是“扫盲性质”，而且篇幅已经很长，不可能再去聊“路由算法”的细节。对此感兴趣的同学，可以去看《计算机网络　》的第5章。

◇路由算法的演变史（以互联网为例）

（技术菜鸟可以跳过这个小节）

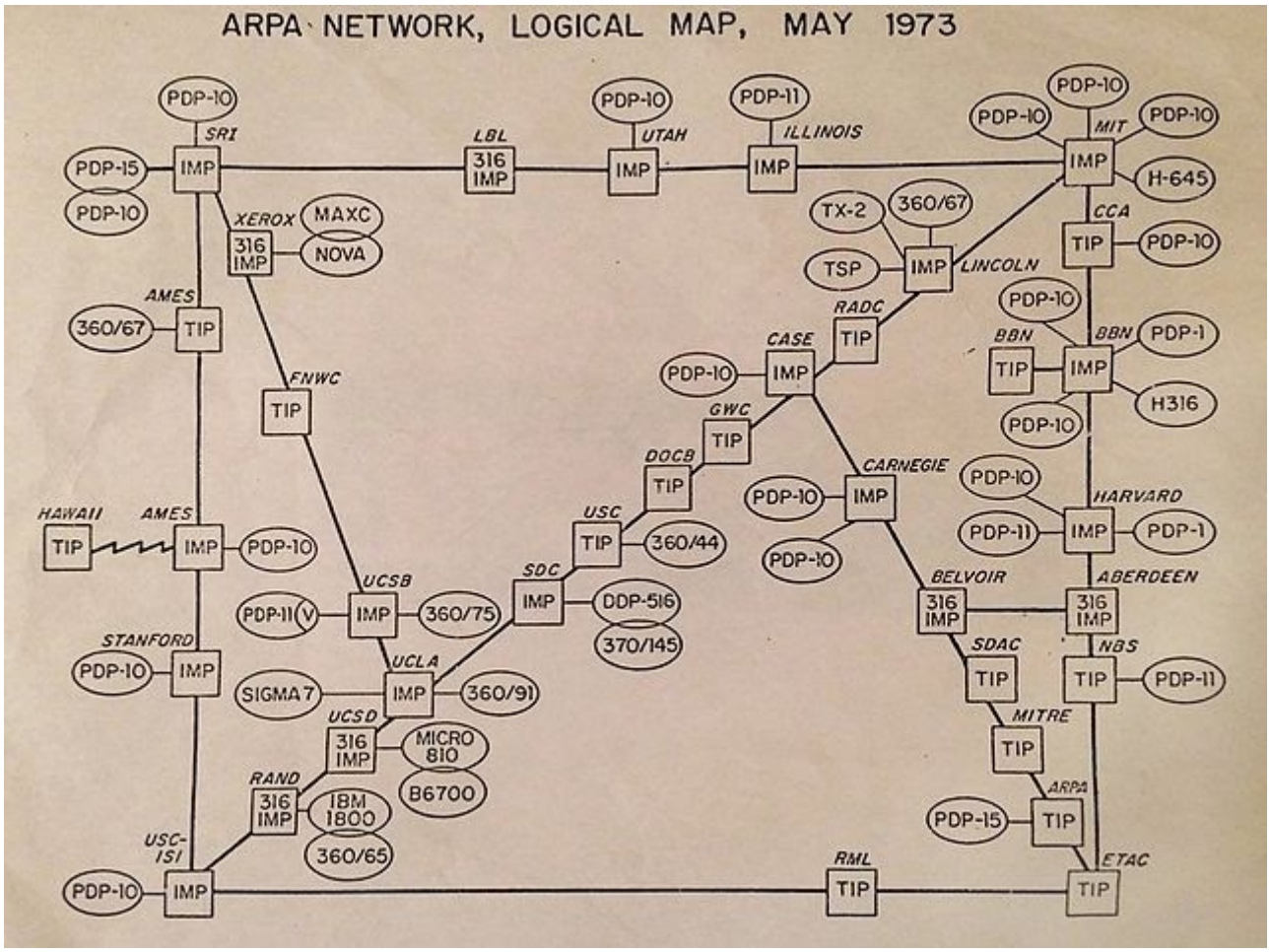
由于互联网的 IP 协议已经成为“网络层协议”的事实标准，俺简单聊一下互联网的路由机制是如何进化滴。

第1阶段：静态全局路由表

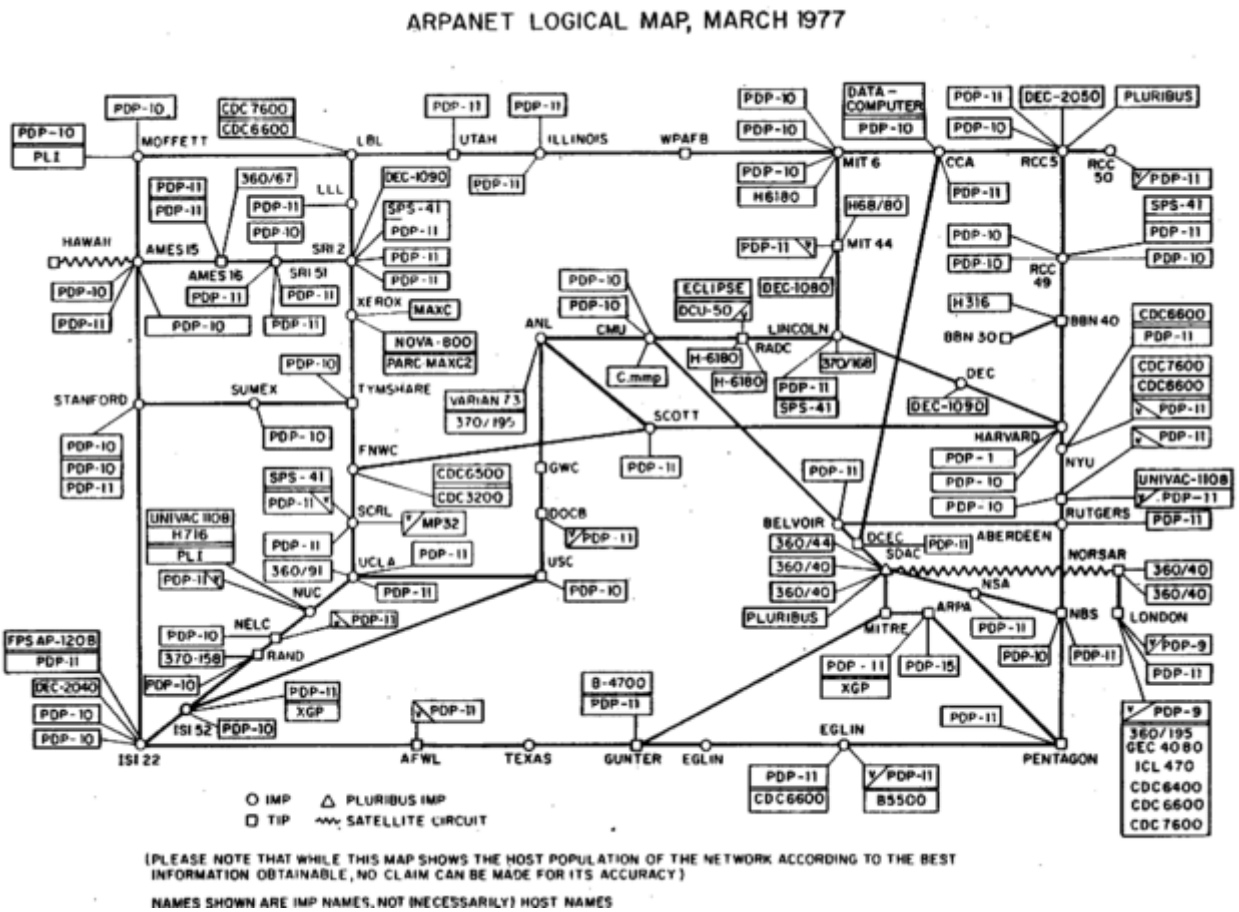
（前面说了）互联网的前身是“阿帕网/ARPANET”。在阿帕网诞生初期（上世纪70年代），全球的主机很少。因此，早期的路由表很简单，既是“全局”滴，又是“静态”滴。简而言之，每个路由器内部都维护一张“全局路由表”，这个“路由表”包含了全球所有其它路由器的关联信息。每当来了一个数据包，查一下这张全局路由表，自然就清楚要转发给谁，才能最快到达目的地。

早期的阿帕网，主机的变化比较少，也很少增加路由器。每当出现一个新的路由器，其它路由器的管理员就手工编辑各自的“全局路由表”。

为了加深大伙儿印象，特意找来两张70年代初的阿帕网拓扑图（注：图中的 IMP 是“Interface Message Processor”的缩写，也就是如今所说的“路由器”）。



（1973年的阿帕网）



(1977年的阿帕网)

第2阶段：动态全局路由表

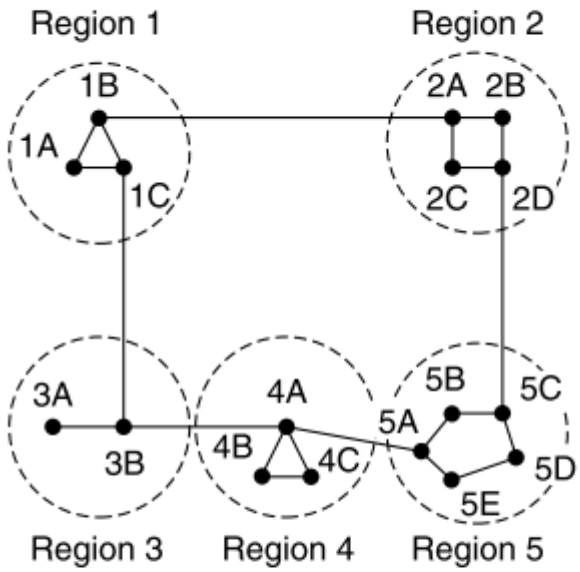
后来，“阿帕网/互联网”的规模猛增，路由器数量也跟着猛增，隔三差五都有新的路由器冒出来。再用“静态路由表”这种机制，（编辑路由表的）管理员会被活活累死。于是改用“动态路由表”，并引入某种“路由发现机制”。但“路由表”依然是【全局】滴。

第3阶段：动态分级路由表

再到后来，全球的路由器越来越多，成千上万，再搞“全局路由表”已经不太现实了——
一方面，“全局路由表”越来越大（查询的速度就越来越慢）
另一方面，由于互联网的流量越来越大，每来一个数据包都要查表，查询越来越频繁。

于是，路由器开始吃不消了。为了解决困境，想出一个新招数：引入“分级路由”（hierarchical routing）。所谓的“分级路由”就是：把整个互联网分为多个大区域，每个大区域内部再分小区域，小区域内部再分小小区域 ..... 看到这里，熟悉“数据结构与算法”的同学就会意识到——这相当于构造了一个【树状】层次结构。

有了这个层次结构，每个路由器重点关注：自己所在的那个最小化区域里面的网络拓扑。如此一来，每个路由器的“路由表”都会大幅度减小。



Full table for 1A

Dest.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

Hierarchic

Dest.	Lin
1A	–
1B	1B
1C	1C
2	1B
3	1C
4	1C
5	1C

(a) (b)

（全局路由表 VS 分级路由表）

◇互联网的路由——从“CAS”的角度看其健壮性

去年（2020）俺写了一篇博文《“政治体制”与“系统健壮性”——基于“复杂性科学”的思考》，其中介绍了“CAS”（复杂自适应系统）的概念。互联网的路由机制，就是一个典型的 CAS。

如果把互联网视作一个系统，每个公网上的路由器都是一个自适应的主体。假如某个地区的网络流量突然暴涨，骨干网路由器会自动分流；假如因为地震或战争，导致某个地区的骨干网路由器全部下线，周边地区的路由器也会自动避开这个区域 .....

所有这些工作，【不需要】依靠任何最高指挥中枢，去进行协调。

相反，如果互联网的路由系统中，设立了某种“中央委员会”进行实时调度，那互联网早就完蛋了，根本无法成长为今天这种规模。

◇网络层的两种交换技术——电路交换（circuit switching ） VS 分组交换（packet switching ）

（技术菜鸟可以跳过这个小节）

前面聊“互联网诞生”，说到兰德公司的“Baran 方案”。该方案对当时的电信系统提出几大革命性的变化，其中之一就是“分组交换”技术（也称“数据包交换”or“封包交换”）。

一般来说，网络层的设计有两种截然不同的风格：【电路交换 VS 分组交换】。有时候也分别称之为“有连接的网络层 VS 无连接的网络层”。此处所说的“连接”指的是某种“虚电路”（洋文叫做“virtual circuit ”，简称 VC）。

要理解“虚电路”，首先要从老式的电话系统说起。

最早期的电话，既没有拨号盘也没有按键，全靠一张嘴。当你拿起电话，先告诉接线员你要打给谁，接线员会用一根跳接线，插入电话交换设备的某个插孔，从而把你的电话机与对方的电话机相连。于是建立了一条两人之间的电话通路，也就是“电路”。你可以把“接线员”想象成某种“人肉路由器” :)

（1900年法国巴黎的电话交换局，可以看到接线员在操作电话交换设备）

后来发明了“自动电话交换机”，导致“接线员”全体下岗。虽然自动化了，但原理还是一样——当你在电话上拨了某人的号码，电话局的交换机会自动选择一条线路。只有当这条线路建立起来，对方的电话才会响。一旦双方开始通话，双方之间的语音都是通过这条线路传输。并且这条线路是独占的——只要通话不挂断，这条线路就不会再分配给其他人使用。

前面提到“互联网诞生的历史”，当时军方推动的“Baran 方案”被 AT＆T 断然拒绝。因为这个方案完全颠覆了传统的电话系统——
颠覆之1：把“模拟信号”颠覆为“数字信号”（这点比较好理解，俺就不解释了）
颠覆之2：把“星形拓扑”颠覆为“网状拓扑”（关于这点，前面的小节已经讨论了）
颠覆之3：把“电路交换”颠覆为“分组交换”（这就是本小节的重点）

为了帮大伙儿理解上述第3点，举个例子：
假设主机 A 要向主机 B 发送一大坨数据。因为数据太多，肯定要分成好几坨小一点的（分成多个数据包）。如何把这些数据包发送给对方捏？

“电路交换”的实现方式
在发送数据之前，要先建立连接通道（通过路由算法，找出 A ＆ B 之间的某条通路）。这条通路就是所谓的“虚电路/VC”。一旦 VC 建立，每一个数据包都是从这条拓扑路径进行路由。

“分组交换”的实现方式
在发送数据之前，【不需要】建立通道，让每个数据包独立进行路由。这种情况下，这几个数据包可能会走【不同的】拓扑路径。因此，数据包到达的顺序与发送的顺序【不一定】相同。接收方收到所有数据包之后，还要自己进行排序。
维基百科上有一个 GIF 动画（这个链接 ），比较直观地演示“分组交换/封包交换”的效果。由于这个动画稍微有点大（超过 1MB），俺就不贴到博文中了。

当时的电话系统主要承载语音传输，“电路交换”显然性能更高。那为啥 Baran 的设计要采用“分组交换”捏？俺又要再次提到【效率 VS 健壮性】之间的矛盾与均衡。
对于“电路交换”，一旦建立连接，同一个连接的所有数据都走相同的路径（会经过完全相同的路由器）。也就是说，传输的过程中，如果某个路由器挂掉了（网络掉线 or 硬件当机 or 软件崩溃）。那么，该路由器正在处理的 N 个连接全都要报废。而“分组交换”则更加灵活——即使某个路由器挂掉了，后续的数据包会自动转向另外的路由器，损失很小。
“Baran 方案”之所以采用“分组交换”的设计，因为人家这个方案是提交给军方用来应对【全面核战争】滴，当然要考虑健壮性啦。

话说这两种交换机制，各有很多支持者，并分裂为两大阵营，分别是：“电信阵营 VS 互联网阵营”。两大阵营的口水战持续了 N 年，都无法说服对方。到了后来设计 OSI 模型的时候，为了保持中立性与通用性，OSI 模型本身并没有强制要求网络层采用哪一种风格。
经过几十年之后，咱们已经可以看出来：“互联网阵营”占据主导地位。如今，连电信系统都是架构在互联网之上。

★网络层：具体实例

◇网络层的【协议】

网络层的协议有很多。由于“互联网”已经成为全球的事实标准，因此俺只列出属于“互联网协议族”的那些“网络层协议”：

IP 协议（含 IPv4 & IPv6 ）
ICMP
IGMP
IPSec
……

（考虑到篇幅）俺不可能具体细聊这些协议，只是贴出每个的维基百科链接，感兴趣的同学自己点进去看。

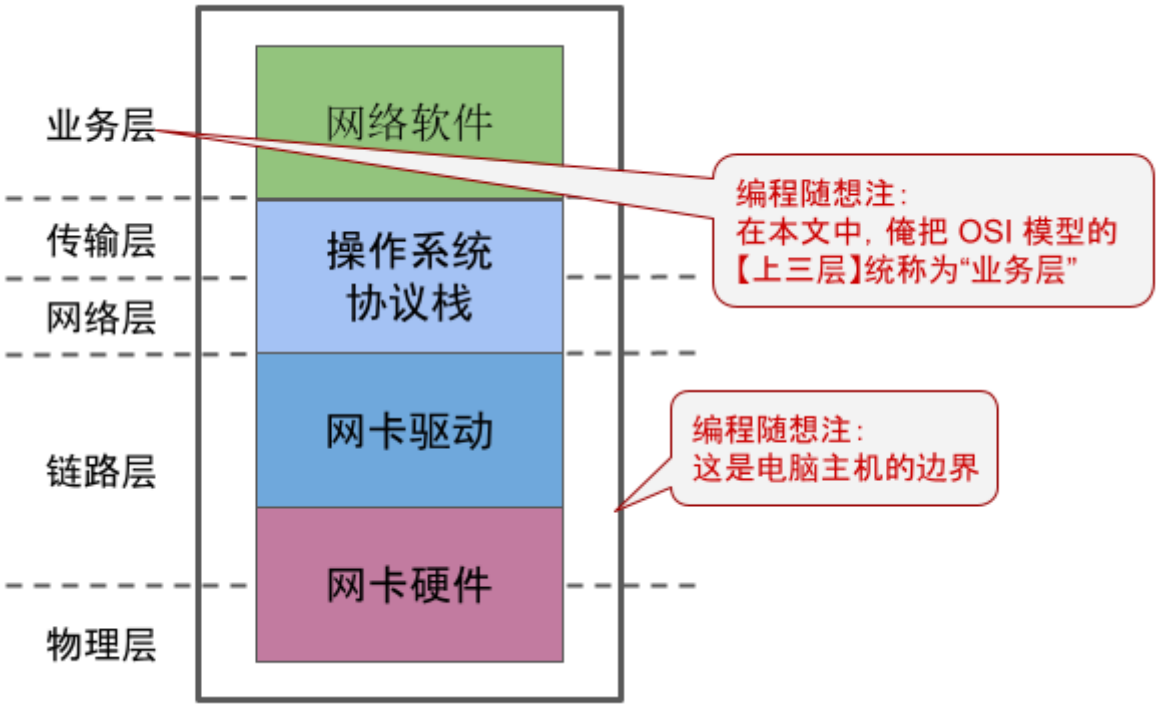
对上述这些协议，最重要的当然是 IP 协议。如果你想要深入了解 IP 协议，可以参考如下这本书。关于 IP 协议的书，此书的影响力最大。这本书共3卷，通常只需看第1卷。

《TCP-IP 详解 》

◇网络层的【协议实现】

对于电脑主机（含移动设备），网络层的协议实现通常包含在操作系统自带的网络模块中（也就是“操作系统协议栈”）。具体参见如下示意图。

另外，还有一些专门的【3层】网络设备，也提供网络层的功能（参见本章节的后续小节）。



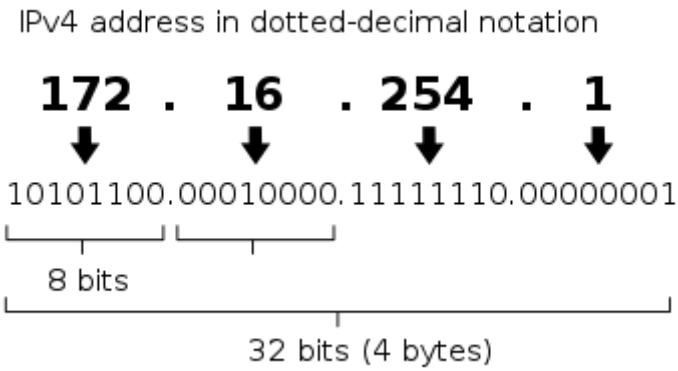
（OSI 模型中，不同层次的协议实现）

◇IP 地址的格式及含义

当年设计阿帕网的时候，采用了【4字节】（32比特）来表示“网络层地址”（也就是 IP 地址）。

“IP 地址”的含义很重要，俺有必要解释一下：

咱们平时所说的 IP 地址，采用【点分十进制】来表示。就是把地址的4个字节，先翻译为十进制，然后每个字节用一个小数点分隔开（参见如下示意图）：



（4字节 IP 地址：“二进制”与“点分十进制”的对照示意图）

“IP 地址”的32比特，分为两部分：第1部分用来标识【子网】，第2部分用来标识该子网中的【主机】。

这两部分各占用多少比特，是不确定的。在这种情况下，“操作系统协议栈”如何知道哪些比特标识“子网”，哪些比特标识“主机”捏？奥妙在于【子网掩码】。所以，大伙儿在给系统配置 IP 地址的时候，通常都需要再设置一个【子网掩码】，就这个用途。

◇IP 地址枯竭，及其解决方法

前一个小节提到：IP地址包含【4字节】（32比特）。因此，最多只能表示【2的32次方】（42亿左右）的不同地址。考虑到还有很多地址保留给特殊用途，实际可用地址远远不到42亿。

到了如今，全球网民都已经几十亿了，IP 地址开始枯竭。咋办捏？为了解决这个问题，发展出若干技术手段。简单说一下最常见的几种手段：

IPv6

名气最大（最多人知道）的技术手段，大概是 IPv6 了。这招想要一劳永逸地解决地址枯竭的问题，采用了16字节（128比特）来表示 IP 地址。

设计 IPv6 的人自豪地宣称：即使给地球上的每一粒沙子分配一个 IPv6 地址，依然绰绰有余（确实没有吹牛，“2的128次方”是天文数字）。

但 IPv6 的缺点在于，【无法】向下兼容原有的 IP 协议（原有的协议叫“IPv4”）。IPv6 的普及一直比较慢，这是主要原因。

代理服务器（proxy）

一看到代理，很多人就想到翻墙。其实它也可以用来解决“地址枯竭”的问题。

比如说，某个公司有100人，100台电脑。如果每台电脑都分配公网 IP 地址，就要消耗100个公网地址（太浪费啦）。

可以只申请一个公网 IP，然后在内网搞一个代理服务器，公网 IP 分配给它（代理服务器有两个网卡，一个接内网，一个接公网）。然后在其它电脑上设置代理，指向这台代理服务器，就都可以上外网啦。

（注：在本文的末尾有一个 "★杂项" 的章节，会专门聊一下“代理”这个话题）

网络地址转换　（NAT）

前面 proxy 那招有个缺点：内网的每台电脑里面的每个上网软件，都要单独设置代理。实在太麻烦啦！

后来就发明了某种更牛逼的招数——网络地址转换（洋文是“Network Address Translation”，简称 NAT）。

用了这招，还是只要申请一个公网 IP，分配给内网的网关（网关有两个网卡，一个接内网，一个接公网）。然后在内网的网关配置 NAT 功能，自动就可以让内网的每台电脑访问外网。

在这篇博文，俺介绍了虚拟机的几种“网卡模式”，其中有一种模式叫做【NAT 模式】，就是指这个玩意儿。

采用了 NAT 技术之后，可能会对某些应用软件（尤其是 P2P 类型的）造成兼容性问题，于是又发明了一些“NAT 穿透技术”（NAT traversal　）。这类技术有好几种，如果有空的话，俺会单独写教程介绍。

其它解决方法

关于“IPv4 地址空间耗尽”，解决方法肯定不止上面这几招。限于篇幅，就此打住。更多的讨论参见维基百科的“这个链接　”。

◇网络层相关的【网络设备】

路由器　（router）

（前面章节聊“路由原理”的时候，已经介绍过它；这里就不再浪费口水啦）

3层交换机（Layer 3 switching）

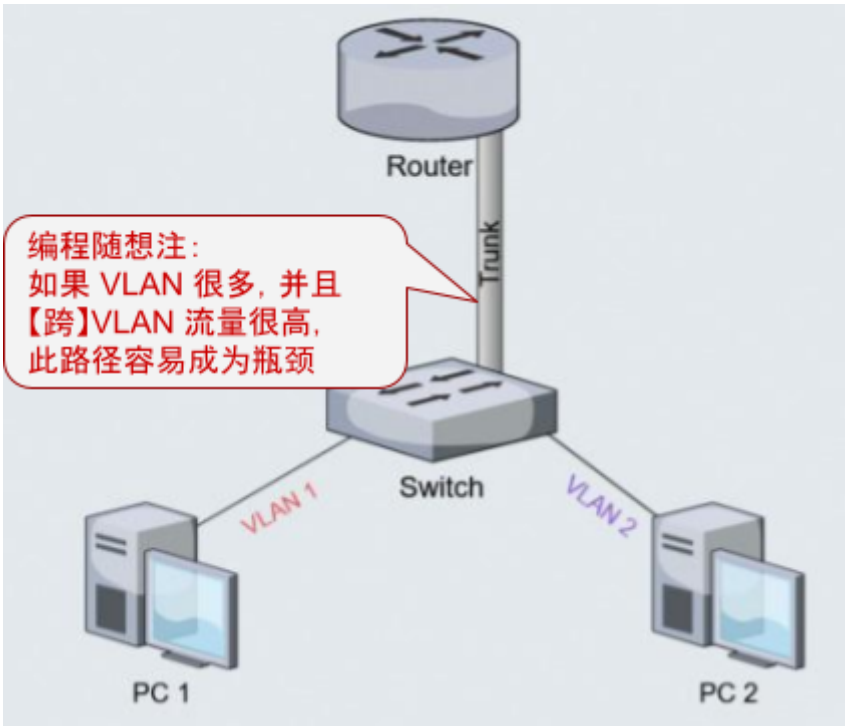
“3层交换机”是在“2层交换机”的基础上，增加了对网络层的处理。因此，它可以做到类似路由器的效果——在几个子网之间转发数据。

与路由器的差别在于——“3层交换机”链接的几个子网是【同种】网络；而路由器可以连接【异种】网络。

从上面这句话看，“3层交换机”的能力显然不如“路由器”。既然已经有“路由器”，为啥还要发明“3层交换机”捏？这就要说到【单臂路由器】的弊端。

对于企业内网的“2层交换机”，通常都支持 VLAN 功能。通俗地说：可以在交换机中划分多个【虚拟子网】。其实这些子网的中所有的电脑，都还是接入这台交换机，只不过这些子网配置了不同的网络地址。对于同一个 VLAN 内部的通讯，“2层交换机”自己就可以搞定（只需要用到2层协议）；但对于【跨】VLAN 主机之间的通讯，“2层交换机”就没戏啦（它没有路由功能）。因此，就必须在它旁边外加一个路由器，形成如下拓扑结构。在这个拓扑中，路由器只与单个设备（2层交换机）相连，所以称之为“单臂”。

请注意：如下示意图只画了两台电脑，位于两个 VLAN。实际上可能有很多个 VLAN，每个里面有几十台电脑。于是，交换机与路由器之间的传输通道就会成为瓶颈——【跨】VLAN 的任意两台电脑通讯，数据包都要到路由器那里兜一圈。为了消除这种瓶颈，才发明了“3层交换机”——把路由功能直接集成到交换机内部。



（“单臂路由器”的拓扑结构）

无线热点 **(Wireless Access Point)**

“无线热点”通常用来提供无线接入，使得某个【无线】设备能接入到某个【有线】网络中。一般来说，热点都内置了路由功能，那么它就是“无线路由器”，对应到“3层”（网络层）。反之，如果没有路由功能，它就是“网桥”，属于“2层”（链路层）。

◇网络层相关的【软件工具】

ping

这个命令，很多人应该都知道。早在 Win9x 就有这个命令了。它使用（网络层的）ICMP 协议来测试某个远程主机是否可达。

提醒一下：

如果 ping 命令显示某个 IP 地址不可达，有很多种情况。比如说：

这个 IP 地址对应的主机已经关机
这个 IP 地址对应的主机已经断线
这个 IP 地址对应的主机拒绝响应 ICMP 协议
从你本机到这个 IP 地址之间，有某个防火墙拦截了 ICMP 协议
……

traceroute

这是一个通用的工具，用来测试路由。很早以前的 Windows 就已经内置了它，命令是 `tracert`。在 POSIX（Linux＆UNIX）上通常叫 `traceroute`

你可以用这个命令，测试你本机与互联网另一台主机之间的路由（也就是：从你本机到对方主机，要经过哪些路由器）

★传输层：概述

◇传输层的必要性

屏蔽“有连接 or 无连接”的差异

（上一个章节提到）网络层本身已经屏蔽了【异种网络】的差异（比如“以太网、ATM、帧中继”之间的差异），而且网络层也屏蔽了路由的细节。但网络层本身还有一个差异，也就是网络层的两种交换技术：电路交换（有连接） VS 分组交换（无连接）。

前面章节也提到了：上述两种交换技术各有很多支持者，并分裂为两大阵营。当年设计 OSI 模型的时候，为了保持中立性与通用性，并没有强制规定“网络层”必须采用何种交换机制。

对于开发网络软件的程序员来说，当然不想操心“网络层用的是哪一种交换机制”。因此，需要对网络层的上述差异再加一个抽象层（也就是“传输层”）。

从“主机”到“进程”

前面介绍的“网络层”，其设计是面向主机（电脑）。“网络层地址”也就是某个主机的地址。

而“传输层”是面向【进程】滴！因为传输层要提供给【网络软件】使用，而网络软件打交道的对象是【另一个网络软件】。因此，传输层必须在“网络层地址”的基础上，再引入某种新的标识，用来区分同一台主机上的不同【进程】。

◇传输层的特殊性

在 OSI 7层模型中，传输层正好居中。这是一个很特殊的位置。

OSI 模型最下面3层，与【网络设备】比较密切。这里面所说的“网络设备”，既包括那些独立的主机（比如“路由器、交换机、等”），也包括电脑上的硬件（比如“网卡”）。

OSI 模型最上面3层，与【网络软件】比较密切（或者说，与“用户的业务逻辑”比较密切）。

而中间的传输层，正好是承上启下。对于开发应用软件的程序猿/程序媛，“传输层”是他们能感知的最低一层。

◇传输层的【端口】

刚才谈“传输层的必要性”，提到说——“网络层地址”只能标识【主机】，而传输层必须要能标识【进程】。为了达到这个目的，于是就引入了“传输层端口”这个概念（为了打字省力，后续讨论简称为“端口”）。

在 OSI 模型中，“端口”的官方称呼是“传输服务访问点”（洋文缩写 TSAP）。但是作为程序员，俺已经习惯于“端口”这个称呼。后续介绍依然用“端口”一词。

当程序员使用传输层提供的 API 开发网络软件时，通常把“端口”与“网络地址”一起使用（构成“二元组”），就可以定位到某个主机上的某个进程。

★传输层：具体实例

◇传输层的【协议】

为了让程序员可以更爽地使用传输层来开发网络软件，传输层既要提供“有连接”的风格，也要提供“无连接”的风格。关于这两种风格的对比，前面已经聊过，这里不再浪费口水。

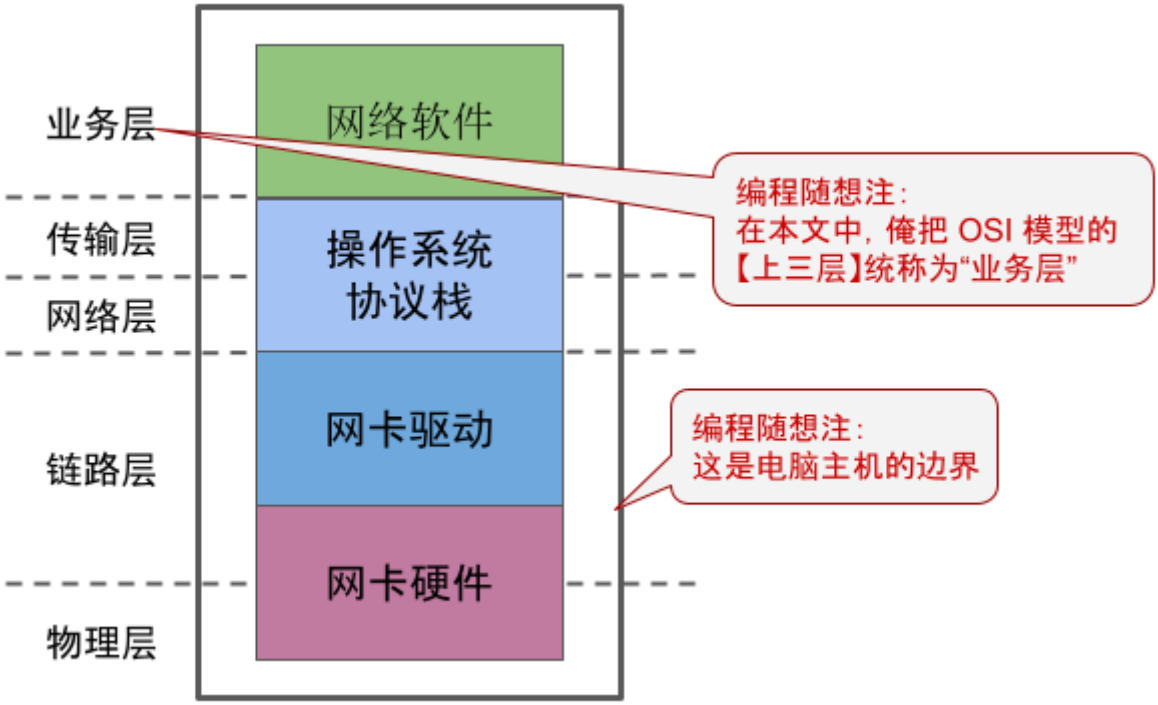
具体到“互联网协议族”，有两个主要的传输层实现，分别是 TCP & UDP（前者是“有连接”，后者是“无连接”）。

除了 TCP & UDP，“互联网协议族”还提供了其它一些传输层协议。因为比较冷门，俺就不介绍啦。

◇传输层的【协议实现】

对于电脑主机（含移动设备），传输层的协议实现通常包含在操作系统自带的网络模块中（也就是“操作系统协议栈”）。具体参见如下示意图。

另外，还有一些专门的【4层】网络设备，也提供传输层的功能（参见后续的小节）。



（OSI 模型中，不同层次的协议实现）

◇套接字（socket API）

前面说了：传输层是面向程序员（让他们可以更方便地开发网络软件）。因此，就需要提供一些封装传输层的【库】（API）。程序员只需要调用这些【库】，就可以使用传输层的协议进行通讯啦。

影响力最大的传输层封装库，当然是 socket API。它来自加州大学伯克利分校。

在互联网诞生初期，伯克利分校开发了一个 UNIX 操作系统的的变种，叫做“伯克利 UNIX 发行版”（BSD Unix），也就是如今 BSD 操作系统的前身。伯克利发行版内置了一套用来进行网络编程的 API，当时叫做“伯克利套接字”（Berkeley sockets ）。由于这套 API 用起来很方便，很多其它的 UNIX 变种也移植了这套 API，于是就逐渐成了业界的事实标准。到了上世纪90年代，Windows & Linux 也都提供了这套 API。

由于大部分读者不是程序员，“套接字”这个话题就到此为止。如果你是个程序员，并且对网络编程感兴趣，可以参考俺的电子书清单 ，其中有一个分类目录是【IT 类 / 软件开发 / 网络相关】。

◇传输层相关的【网络设备】

4层交换机（Layer 4 switching）

前面已经介绍了“3层交换机”，“4层交换机”是其进一步的改良，可以识别传输层的协议，获取 TCP or UDP 的端口号。

有了这个能力，网管就可以在这种交换机上配置一些管理策略。比如说：（根据传输层端口号）过滤掉某种流量，或者对某种流量设置转发的优先级。

状态防火墙（stateful firewall ）

网络防火墙分好几种，大部分属于这种。它能完全处理 TCP 协议的状态，显然它属于“4层”（传输层）。

◇传输层相关的【软件工具】

netcat 家族——传输层的“瑞士军刀”

关于 netcat，俺已经写过一篇比较详细的教程：《扫盲 netcat（网猫）的 N 种用法——从“网络诊断”到“系统入侵”》。看完这篇教程，你肯定能体会它功能的强大——很多与 TCP/UDP 相关的事情，都可以用 netcat 搞定。

另外，netcat 还有很多衍生品（衍生的开源项目），构成一个丰富的 netcat 家族。在上述教程也有介绍。

netstat & ss

Windows 和 POSIX（Linux＆UNIX）都有一个 netstat 命令，可以查看当前系统的 TCP/UDP 状态（包括当前系统开启了哪些监听端口）。

另外，Linux 上还有一个 ss 命令，功能更强（但这个命令在 Windows 上默认没有）

nmap

这是最著名的开源的扫描器，可以扫描远程主机监听了哪些传输层端口（注：前面提到的“netcat 家族”也可以干这事儿）

nmap 的功能很强，“端口扫描”只是其功能之一。

★业务层（OSI 上三层）：概述

一不小心，这篇教程已经写了这么长。为了照顾那些有“阅读障碍”的读者，俺要稍微控制一下篇幅，就把 OSI 的【上三层】合在一起讨论。

前面的章节说过：【上三层】更接近于“网络软件”，对应的是应用软件的业务逻辑，因此俺统称为“业务层”。

注：有些书（比如《计算机网络　》）会把 OSI 的上三层统称为“应用层”。由于 OSI 模型中本来就有一个“应用层”，俺认为这样容易搞混（尤其不利于技术菜鸟），所以另外起了一个“业务层”的名称。

◇业务层的必要性

业务层显然是必要滴。因为传输层位于操作系统，它不可能去了解网络软件的业务逻辑。为了让网络软件能够相互通讯，肯定要在传输层之上再定义更高层的协议。

问题在于：网络软件千奇百怪，其业务逻辑各不相同，因此，“业务层如何设计”，【无】一定之规。有些软件只用一个协议来搞定所有的业务逻辑（只有一层）；有些软件会参考 OSI，把业务逻辑的协议分为三层；还有些软件可能会分出更多的层次。

再强调一下：业务层的协议如何分层，完全看具体的业务逻辑，不要生搬硬套任何现有的参考模型。

◇会话层 & 表示层 & 应用层

对于大部分读者来说，【没必要】花时间去了解 OSI 最上面三层之间的区别。你只需把最上面三层视作【一坨】——他们都是与网络软件的业务逻辑密切相关滴。

那么，哪些人需要详细了解“这三层的差异”捏？

如果你是个程序员，并且你正好是开发【网络】软件，俺建议你了解一下 OSI 模型的最上面三层，有助于你更深刻地思考某些网络协议的设计。所谓的“更深刻”指的是：你不能光停留在 WHAT 层面，要提升到 HOW 甚至 WHY 层面（参见《学习技术的三部曲：WHAT、HOW、WHY》）

★业务层（OSI 上三层）：具体实例

◇业务层的【协议】

业务层的协议非常多。即使光把各种协议的名称列出来，也很费劲。所以俺就偷懒一下，只点评几个特别重要的协议。

HTTP 协议

如果让俺评选最重要的业务层协议，俺首推 HTTP 协议。互联网的普及推动了 Web 的普及，而 Web 的普及使得 HTTP 成为信息时代的重要支柱。当你上网的时候，你看到的网页（HTML 页面）就是通过 HTTP 协议传输到你的浏览器上。

如今 HTTP 已经不仅仅用来展示网页，还有很多业务层的协议是建立在 HTTP 协议之上。比如说：如果你用 RSS 订阅俺的博客，RSS 阅读器需要调用 blogspot 博客平台提供的 RSS 接口，这些 RSS 接口就是基于 HTTP 协议传输滴。

考虑到本文的篇幅，俺不可能在这里细聊 HTTP 协议的规格，有兴趣的同学可以去看《HTTP 权威指南　》这本书。

SSL/TLS 协议

最早的 HTTP 协议是【明文】滴；为了强化安全性，后来又设计了 SSL 协议，用来【加密】HTTP 流量；再后来，SSL 升级为 TLS（这俩是同义词）。如今经常看到的 HTTPS 相当于“HTTP over TLS”。

SSL/TLS 设计得比较优雅（很灵活），使得其它业务层的协议可以很方便地架构在 SSL/TLS 之上。这样的好处是：其它协议就不用自己再设计一套加密机制&认证机制。

SSL/TLS 对于安全性很重要，因此俺专门写了一个系列教程（如下），详细介绍该协议的技术细节。
《扫盲 HTTPS 和 SSL / TLS 协议》（系列）

域名相关的协议（DNS 及其它）

域名相关的协议，也很重要。因为域名系统是整个互联网的基础设施。最早的域名查询协议是“DNS 协议”，由于这个协议【没有】加密，导致了一些安全隐患。比如 GFW 就利用 DNS 的这个弱点，搞“域名污染/域名投毒”。因此，后来又设计了一系列新的域名协议，引入了加密的机制。

关于这些协议的扫盲教程，可以参考如下几篇博文：

《扫盲 DNS 原理，兼谈“域名劫持”和“域名欺骗 / 域名污染”》
《对比4种强化域名安全的协议——DNSSEC，DNSCrypt，DNS over TLS，DNS over HTTPS》

◇业务层相关的【网络设备】

应用层防火墙（application firewall ）

前面提到了：大多数网络防火墙处于4层（状态防火墙），另外还有少数处于7层，也就是“应用层防火墙”（有时候也称之为“7层防火墙”）。

一般来说，这类防火墙具备了【深度包检测】（deep packet inspection，简称 DPI）的能力，可以分析应用层协议的【内容】。

简单说一下“深度包检测”：

如果某个网络设备，仅仅分析“应用层协议”本身，它还【不够格】称之为 DPI。为了做到 DPI，还要能理解应用层协议所承载的【内容】。

比如说：某人通过【明文】的 HTTP 协议从网上下载了一个 zip 压缩包。对于这个下载行为，那些做得好的 DPI 设备不光能识别出“HTTP 协议的内容是 ZIP 压缩包”，而且还能从 ZIP 压缩包中提取出里面的文件。

入侵检测（intrusion detection system ）

一般来说，“入侵检测”如果不加定语，通常指“【网络】入侵检测”（洋文叫 NIDS）；另外还有一种“【主机】入侵检测”（洋文叫 HIDS）。HIDS 与本文无关。

“入侵检测”是一种网络安全设备，它通过嗅探（sniffer）的方式抓取网上的数据包，然后进行分析，尝试发现网络中是否存在黑客/骇客的入侵的行为。故名“入侵检测”。

由于 IDS 需要理解【应用层】（7层）的内容，因此它与“应用层防火墙”有个共同点，需要具备某种程度的 DPI（深度包检测）能力。它俩的一大差异是【部署方式】。

考虑到很多读者是 IT 外行，简单说一下“旁路部署”——
如果你学过中学物理，应该知道电路有“串联 & 并联”。所谓的“旁路部署”类似于电路中的【并联】。通俗地说：IDS 是【并联】部署，防火墙是【串联】部署。

GFW（Great Firewall）

本博客已经写了很多翻墙教程，大伙儿肯定都知道 GFW 了。

由于“Great Firewall”中有“Firewall”字样，很多天朝网民【误以为】GFW 是防火墙，其实不然！GFW 本质上就是 IDS——其部署方式类似于 IDS（旁路部署），其工作方式有很大一部分也类似于 IDS（当然啦，GFW 的功能比 IDS 更多）。

大约七八年前，就有热心读者建议俺写一篇技术博文，介绍 GFW 的工作原理。由于俺比较懒，拖到今年（2021）都没动手，很惭愧 :(

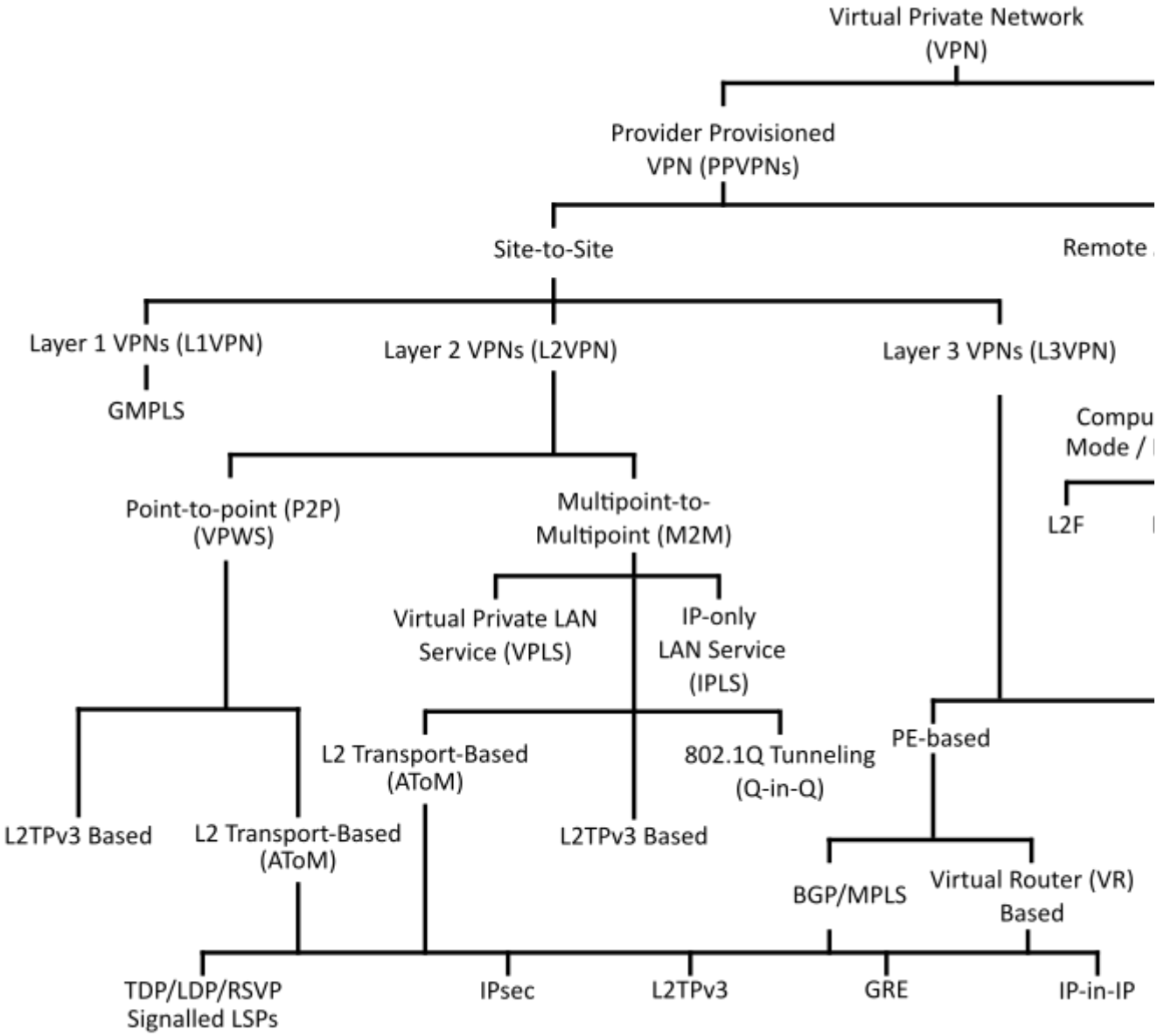
★杂项

有些概念，并不属于某个特定的层次，单独放到这个章节。

◇VPN（virtual private network ）

咱们天朝的网民使用 VPN，一多半是为了翻墙。其实 VPN 的本意（如其名称所示）是为了提供某种虚拟化的私有的网络，让身处异地的多个人，可以用 VPN 构建出一个虚拟的内网，从而能在这个内网中协同工作。

VPN 的类型很多，使用的技术也各不相同，因此 VPN 对应的 OSI 层次很宽（“1层”到“6层”）。俺到维基百科剽窃了如下这张图，让你见识一下 VPN 的多样性。



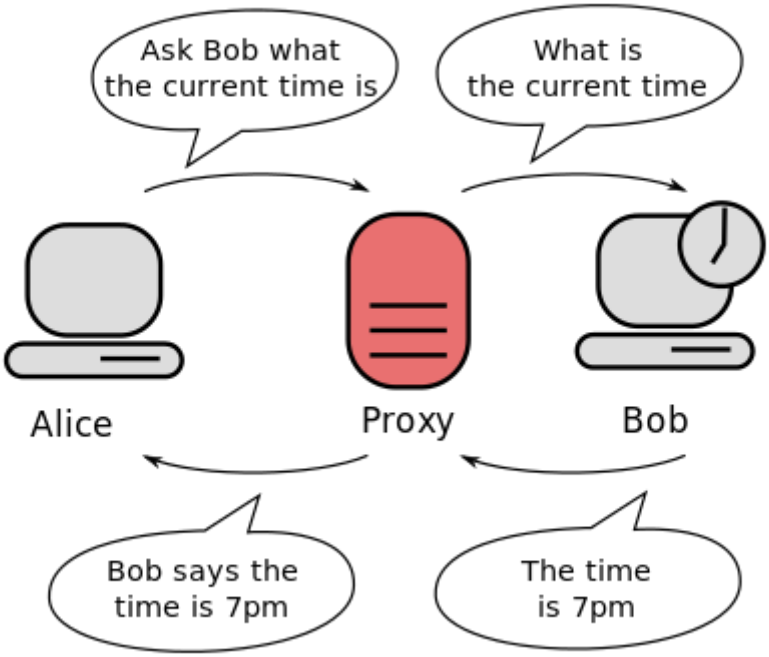
（名目繁多的 VPN，分类示意图）

◇代理（proxy）

那些经常翻墙的同学，对“代理”应该都很熟悉了。“代理”与 VPN 类似，一开始并不是用来翻墙滴，“翻墙”只是这俩的副业。

代理服务器（proxy server）

“代理服务器”部署在“客户端 & 服务端”之间，起到某种“中介”的作用。“代理服务器”的类型有很多，干的事情各不相同。



（“代理服务器”的简单示意图）

代理客户端（proxy client）

早期的代理服务器，【不】需要“代理客户端”。因为早期的“代理服务器”支持的是【标准协议】。比如“HTTP proxy server”支持的是标准 HTTP协议，而用户的电脑上，已经有浏览器（原生支持 HTTP 协议）。这种情况下，自然不需要再有“代理客户端”。

后来，为了满足某些特殊需求（比如翻墙），“代理服务器”必须使用某种特殊的（非标准的）协议。因此，就必须在用户的环境中安装“代理客户端”。对于翻墙来说，你装的翻墙软件，相当于“代理客户端”。

代理的层次

“代理”也分不同的层次。比较常见的有如下几种：

TCP 代理（TCP 端口转发）——4层（传输层）
SOCKS 代理——5层（会话层）
HTTP 代理——7层（应用层）
……

◇网关（gateway ）

前面的某些章节，已经稍微提及了“网关”这个概念，但还没有具体介绍它。

严格来讲，“网关”是一个逻辑概念，【不要】把它当成具体的网络设备。充当“网关”的东东，可能是：路由器 or XX层交换机 or XX层防火墙 or 代理服务器 ……

“网关”也分不同的层次。如果不加定语，通常指的是“3层网关”（网络层网关）。列几种比较常见的，供参考：

路由器充当网关——3层（网络层）
3层交换机充当网关——3层（网络层）
4层交换机充当网关——4层（传输层）
应用层防火墙充当网关——7层（应用层）
代理服务器充当网关——（取决于代理的层次，参见前一个小节）
……

◇隧道协议（tunneling protocol ）

所谓的“隧道协议”，通俗地说就是：用某种协议包裹另一种协议，以满足某些特殊的需求。

看到这里，估计某些同学会感到纳闷——因为俺在本文开头介绍“协议栈”的时候提到说：相邻的两层协议，下层会包裹上层。“隧道协议的包裹”与“上下层协议的包裹”，差别在哪捏？

俺来解释一下：

“隧道协议”可以做到更灵活的包裹——既可以对层次相隔很远的协议进行包裹，也可以对同一层的协议进行包

裹，甚至可以“倒挂”——所谓的“倒挂”就是让【上】层反过来包裹【下】层。

举例：

俺曾经写过一篇《如何让【不支持】代理的网络软件，通过代理进行联网（不同平台的 N 种方法）》，其中介绍了“HTTP 代理”的两种模式：“转发模式 & 隧道模式”。对于“HTTP 代理”的隧道模式，可以实现【TCP over HTTP】（把 TCP 协议打包到 HTTP 协议内部），这就是刚才所说的“倒挂”。

另外，VPN 小节的那张图中，有些类型的 VPN 就是用“隧道协议”的机制实现。

◇ (其它杂项)

可能还有一些杂七杂八的东东，没来得及聊。如果你觉得有些【网络相关】的概念，不太明白，欢迎到博客留言，进行反馈。

俺会根据大伙儿的反馈，再对这篇教程进行补充。

★参考书目

中文书名	英文书名	作者
《计算机网络　》	《Computer Networks》	Andrew Tanenbaum David Wetherall
《TCP-IP 详解　》	《TCP-IP Illustrated》	Richard Stevens
《UNIX 网络编程　》	《UNIX Network Programming》	Richard Stevens
《HTTP 权威指南　》	《HTTP——The Definitive Guide》	David Gourley Brian Totty Marjorie Sayer Sailu Reddy Anshu Aggarwal