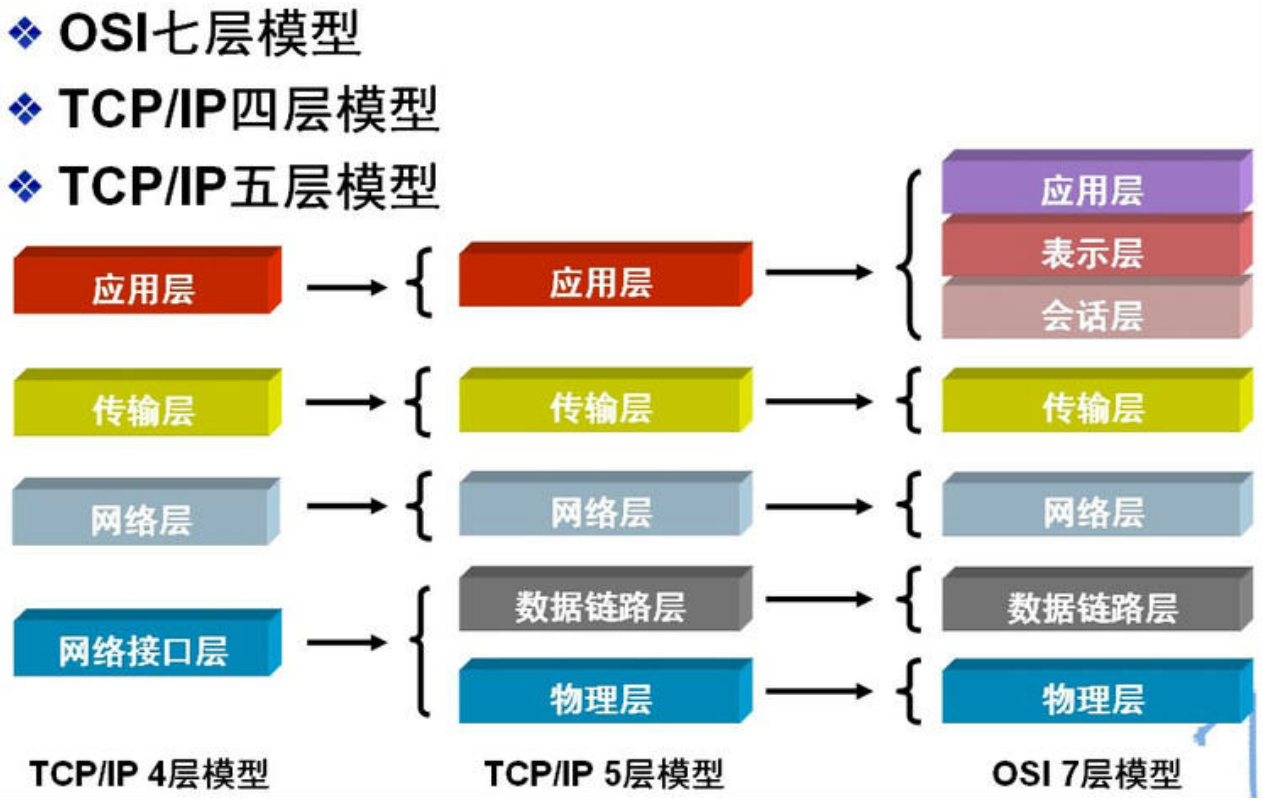
面试宝典-网络

计算机网络基础知识

网络层次划分

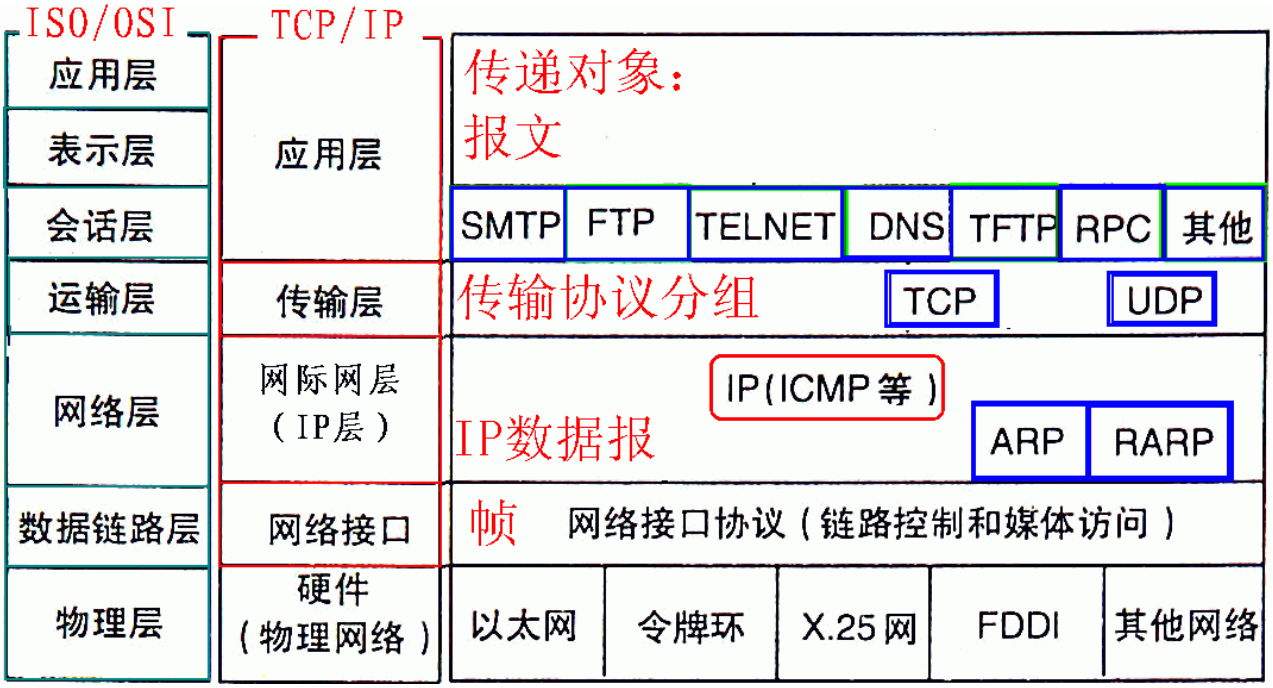
为了使不同计算机厂家生产的计算机能够相互通信，以便在更大的范围内建立计算机网络，国际标准化组织（ISO）在 1978 年提出了 “开放系统互联参考模型”，即著名的 OSI/RM 模型（Open System Interconnection/Reference Model）。它将计算机网络体系结构的通信协议划分为七层，自下而上依次为：物理层（Physics Layer）、数据链路层（Data Link Layer）、网络层（Network Layer）、传输层（Transport Layer）、会话层（Session Layer）、表示层（Presentation Layer）、应用层（Application Layer）。其中第四层完成数据传输服务，上面三层面向用户。

除了标准的 OSI 七层模型以外，常见的网络层划分还有 TCP/IP 四层协议以及 TCP/IP 五层协议，它们之间的对应关系如下图所示：



OSI 七层网络模型

TCP/IP 协议毫无疑问是互联网的基础协议，没有它就根本不可能上网，任何和互联网有关的操作都离不开 TCP/IP 协议。不管是 OSI 七层模型还是 TP/IP 的四层、五层模型，每一层中都要自己的专属协议，完成自己相应的工作以及与上下层级之间进行沟通。由于 OSI 七层模型为网络的标准层次划分，所以以 OSI 七层模型为例从下向上进行一一介绍。



TCP/IP 协议参考模型把所有的 TCP/IP 系列协议归类到四个抽象层中：

- 应用层：TETP、HTTP、SNMP、FTP、SMTP、DNS、Telnet 等。
- 传输层：TCP、UDP。
- 网络层：IP、ICMP、OSPF、EIGRP、IGMP。
- 数据链路层：SLIP、CSLIP、PPP、MTU。

会话层、表示层和应用层重点：

1. 数据传输基本单位为报文；
2. 包含的主要协议：FTP（文件传送协议）、Telnet（远程登录协议）、DNS（域名解析协议）、SMTP（邮件传送协议）、POP3（邮局协议）、HTTP 协议（Hyper Text Transfer Protocal）。

OSI 是一个理想的模型，一般的网络系统只涉及其中的几层，在七层模型中，每一层都提供一个特殊的网络功能，从网络功能角度观察：

- 下面 4 层（物理层、数据链路层、网络层和输出层）主要提供数据传输和交换功能，即以节点到节点之间的通信为主
- 第 4 层作为上下两部分的桥梁，是整个网络体系结构中最关键的部分
- 上 3 层（会话层、表示层和应用层）则以提供用户与应用程序之间的信息和数据处理功能为主。

简而言之，下 4 层主要完成通信子网的功能，上 3 层主要完成资源子网的功能。

物理层

设备之间的数据通信提供传输媒体及互连设备，为数据传输提供可靠的环境。激活、维持、关闭通信端点之间的机械特性、电气特性、功能特性以及过程特性。物理层确保原始的数据可在各种物理媒体上传输。

可以理解为网络传输的物理媒体部分，比如网卡、网线、集线器、中继器、调制解调器等！

在这一层，数据还没有被组织，仅作为原始的位流或电气电压处理，这一层的单位是：bit 比特。

数据链路层

数据链路层在物理层提供的服务的基础上向网络层提供服务，**其最基本的服务是将源自网络层来的数据可靠地传输到相邻节点的目标机网络层**。所以数据链路层在不可靠的物理介质上提供可靠的传输。

该层的作用包括：物理地址寻址、数据的成帧（帧是数据链路层的传送单位）、流量控制、数据的检错、重发等。

可以理解为数据通道，主要功能是如何在不可靠的物理线路上进行数据的可靠传递。

数据链路指的是：物理层要为终端设备间的数据通信提供传输媒体及其连接。媒体是长期的，连接是有生存期的。在连接生存期内，收发两端可以进行不等的一次或多次数据通信。每次通信都要经过建立通信联络和拆除通信联络两过程！这种建立起来的数据收发关系。

该层的设备有：网卡、网桥、网路交换机，另外这层的单位为：帧。

1. 数据链路层为网络层提供可靠的数据传输；
2. 基本数据单位为帧；
3. 主要的协议：以太网协议；
4. 两个重要设备名称：网桥和交换机；

网络层

网络层的目的是将网络地址翻译成对应的物理地址，并决定如何将数据从发送方路由到接收方，具体功能包括寻址和路由选择以及连接的建立、保持和终止等。它提供的服务使传输层不需要了解网络中的数据传输和交换技术。简单来说网络层，那就是： 路径选择、路由及逻辑寻址 ，建立网络连接和为上层提供服务。

另外 IP 协议就在这一层。网络层中设计众多的协议，其中包括最重要的协议，也是 TCP/IP 的核心协议 -- IP 协议。IP 协议非常简单，仅仅提供不可靠、无连接的传送服务。

IP 协议的主要功能有：无连接数据报传输、数据报路由选择和差错控制。

与 IP 协议配套使用实现其功能的还有地址解析协议 ARP、逆地址解析协议 RARP、因特网报文协议 ICMP、因特网组管理协议 IGMP。

1. 网络层负责对子网间的数据包进行路由选择。此外，网络层还可以实现拥塞控制、网际互连等功能；
2. 基本数据单位为 IP 数据包；
3. 包含的主要协议：
    - IP 协议（Internet Protocal，因特网互联协议）；
    - ICMP 协议（Internet Control Message Protocal，因特网控制报文协议）；
    - ARP 协议（Address Resolution Protocol，地址解析协议）；
    - RARP 协议（Reverse Address Resolution Protocal，逆地址解析协议）。
4. 重要的设备：路由器。

ARP 协议

地址解析协议，即 ARP（Address Resolution Protocol），是根据 IP 地址获取物理地址的一个 TCP/IP 协议。

主机发送消息时将包含目标 IP 地址的 ARP 请求广播到网络上的所有主机，并接收返回消息，以此确定目标的物理地址；收到返回消息后将该 IP 地址和物理地址存入本地 ARP 缓存中并保留一定时间，下次请求时直接查询 ARP 缓存以节约资源。

地址解析协议是建立在网络上各个主机互相信任的基础上的，网络上的主机可以自主发送 ARP 应答消息，其他主机收到应答报文时不会检测该报文的真实性就会将其计入本机 ARP 缓存；由此攻击者就可以向某一主机发送伪 ARP 应答报文，使其发送的消息无法达到预期的主机或达到错误的主机，这就构成了一个 ARP 欺骗。

ARP 命令可用于查询本机 ARP 缓存中 IP 地址和 MAC 地址的对应关系、添加或删除静态对应关系等。

RARP 协议

逆地址解析协议，即 RARP，功能和 ARP 协议相对，其将局域网中某个主机的物理地址转换为 IP 地址。

比如局域网中有一台主机只知道物理地址而不知道 IP 地址，那么可以通过 RARP 协议发出征求自身 IP 地址的广播请求，然后由 RARP 服务器负责回答。

传输层

第一个端到端，即主机到主机的层次。传输层负责将上层数据分段并提供端到端、可靠的或不可靠的传输。

此外，传输层还要处理端到端的差错控制和流量控制问题。传输层的任务是根据通信子网的特性，最佳的利用网络资源，为两个端系统的会话层之间，提供建立、维护和取消传输连接的功能，负责端到端的可靠数据传输。

在这一层，信息传送的协议单元称段或报文。网络层只是根据网络地址将源结点发出的数据包传送给到目的结点，而传输层则负责将数据可靠的传送到相应的端口。

向上面的应用层提供通信服务，面向通信部分的最高层，同时也是用户功能中的最底层。

接收会话层数据，在必要时将数据进行分割，并将这些数据交给网络层，并且保证这些数据段有效的到达对端。

1. 传输层负责将上层数据分段并提供端到端、可靠的或不可靠的传输以及端到端的差错控制和流量控制问题。
2. 包含的主要协议：TCP 协议（Transmission Control Protocol，传输控制协议）、UDP 协议（User Datagram Protocal，用户数据报协议）；
3. 重要设备：网关。
4. 传递的协议单元：段或报文。

TCP 协议

TCP（Transmission Control Protocol，传输控制协议）是一种面向连接的、可靠的、基于字节流的通信协议，通过三次握手建立连接，通讯完成时要断开连接，由于 TCP 是面向连接的所以只能用于端到端的通讯。建立连接的目的是保证 IP 地址、端口、物理链路等正确无误，为数据的传输开辟通道。

TCP 提供的是一种可靠的数据流服务，采用“带重传的肯定确认”技术来实现传输的可靠性。TCP 还采用一种称为“滑动窗口”的方式进行流量控制，所谓窗口实际表示接收能力，用以限制发送方的发送速度。

TCP 报文首部格式：

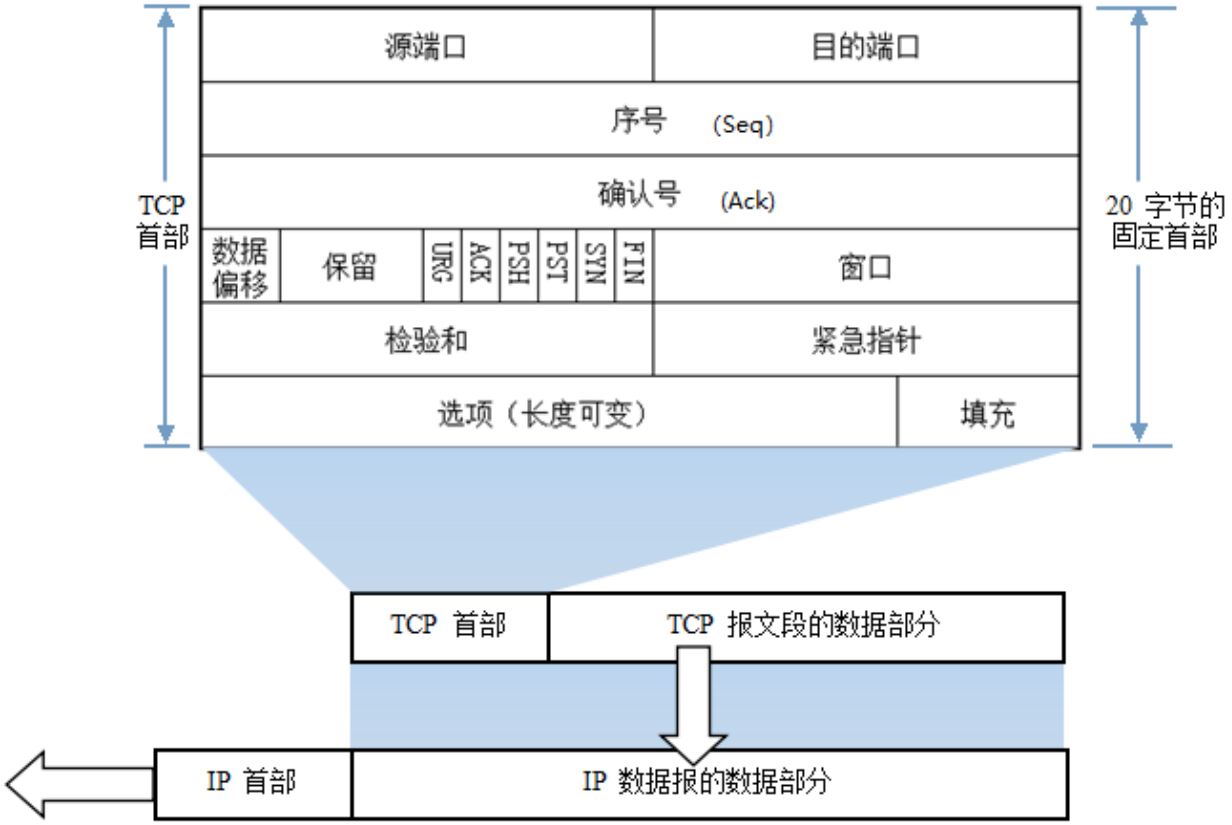
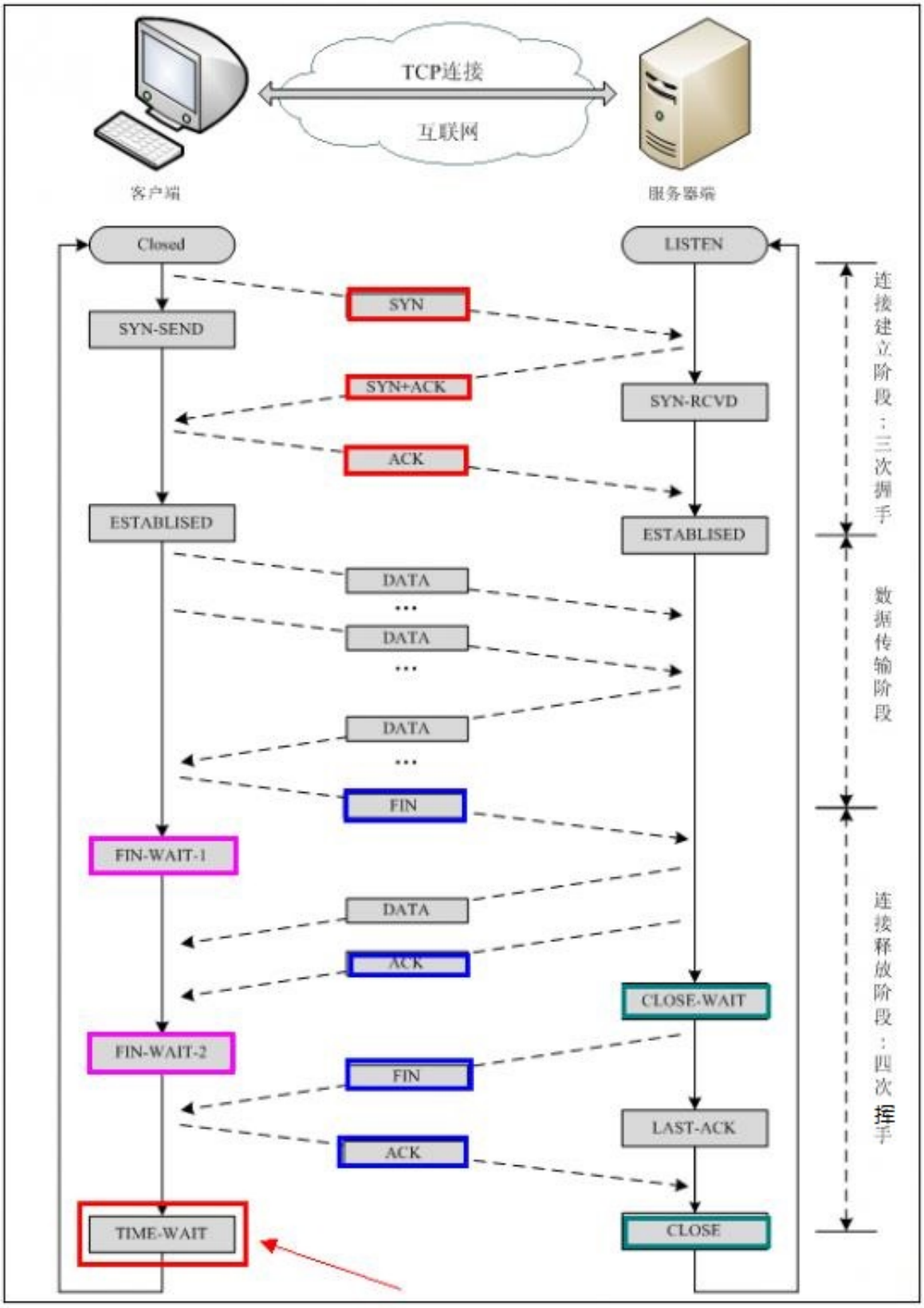


图 TCP 报文段的首部格式

TCP 协议的通信过程：



注：seq：“sequance” 序列号；ack："acknowledge" 确认号；SYN：“synchronize” 请求同步标志；ACK：“acknowledge” 确认标志；FIN："Finally" 结束标志。

使用 TCP 的协议：FTP （文件传输协议）、Telnet（远程登录协议）、SMTP（简单邮件传输协议）、POP3（和 SMTP 协议，用于接收邮件）、HTTP 协议等。

UDP 协议

UDP 用户数据报协议，是面向无连接的通讯协议，UDP 数据包括目的端口号和源端口号信息，由于通讯不需要连接，所以可以实现广播发送。

UDP 通讯时不需要接收方确认，属于不可靠的传输，可能会出现丢包现象，实际应用中要求程序员编码验证。

UDP （User Datagram Protocol）用户数据包协议，非连接的协议，传输数据之前源端和终端不建立连接，当它想传送时就简单地去抓取来自应用程序的数据，并尽可能快地把它扔到网络上。在发送端，UDP 传送数据地速度仅仅是受应用程序生成数据的速度、计算机的能力和传输带宽的限制；在接收端，UDP 把每个消息段放在队列中，应用程序每次从队列中读一个消息段。相比 TCP 就是无需建立连接，结构简单，但是无法保证正确性，容易丢包。

UDP 和 TCP 位于同一层，但它不管数据包的顺序、错误或重发。因此，UDP 不被应用于那些使用虚电路的面向连接的服务，UDP 主要用于那些面向查询-应答的服务，例如 NFS。相对于 FTP 或 Telent。这些服务需要交换的信息量较小。

每个 UDP 报文分 UDP 报头和 UDP 数据区两部分。报头由四个 16 位长（2 字节）字段组成，分别说明该报文的源端口、目的端口、报文长度以及校验值。UDP 报头由 4 个域组成，其中每个域个占用 2 个字节，具体如下：

1. 源端口号
2. 目的端口号
3. 数据报长度
4. 校验值

使用 UDP 协议包括：TETP（简单文件传输协议）、SNMP（简单网络管理协议）、DNS（域名解析协议）、NFS、BOOTP。

TCP 与 UDP 的区别：TCP 是面向连接的，可靠的字节流服务；UDP 是面向无连接的，不可靠的数据包服务。

会话层

会话层管理主机之间的会话进程，即负责在网络中的两节点之间建立、管理、终止进程之间的会话。建立通信链接，保持会话过程通信链接的畅通，同步两个节点之间的对话，决定通信是否被中断以及通信中断时决定从何处重新发送，即不同机器上的用户之间会话的建立及管理。会话层还利用在数据中插入校验点来实现数据的同步。

表示层

表示层对来自应用层的数据或信息进行解释，对各种语法赋予相应的含义，并按照一定的格式传送给会话层，以保证一个主机应用层信息可以被另一个主机的应用程序理解。表示层的数据转换包括数据的加密、压缩、格式转换等。

其主要功能是 “ 处理用户信息的表示问题，如编码、数据格式转换和加密解密、压缩解压缩 ” 等。

应用层

OSI 参考模型的最高层，为操作系统或网络应用程序提供访问网络服务的接口。它在其他 6 层工作的基础上，负责完成网络中应用程序与网络操作系统之间的联系，建立与结束使用者之间的联系，并完成网络用户提出的各种网络服务及应用所需的监督、管理和服务等各种协议。此外，该层还负责协调各个应用程序间的工作。

应用层为用户提供的服务和协议有：文件服务、目录服务、文件传输服务（FTP）、远程登录服务（Telnet）、电子邮件服务（E-mail）、打印服务、安全服务、网络管理服务、数据库服务等等。

DNS 协议

DNS 是域名系统（DomainNameSystem）的缩写，该系统用于命名组织到域层次结构中的计算机和网络服务，可以简单地理解为将 URL 转换为 IP 地址。

域名是由圆点分开一串单词或缩写组成地，每一个域名都对应一个唯一的 IP 地址，在 Internet 上域名与 IP 地址之间是一一对应的，DNS 就是进行域名解析的服务器。

DNS 命名用于 Internet 等 TCP/IP 网络中，通过用户友好的名称查找计算机和服务。

HTTP 协议

超文本传输协议（HTTP，HyperText Transfer Protocol）是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。

HTTP 协议包括的请求

1. GET：请求读取由 URL 所标志的信息。
2. POST：给服务器添加信息（如注释）。
3. PUT：在给定的 URL 下存储一个文档。
4. DELETE：删除给定的 URL 所标志的资源。

POST 与 GET 的区别

1. GET 是从服务器上获取数据，POST 是向服务器传输数据。
2. GET 是把参数数据队列加到提交表单的 Action 属性所指向的 URL 中，值和表单内各个字段一一对应，在 URL 中可以看到。
3. GET 传送的数据量小，不能大于 2 KB；POST 传送的数据量较大，一般被默认为不受限制。
4. 根据 HTTP 规范，GET 用于信息获取，而且应该是安全的和幂等的。

所谓安全的意味着该操作用于获取信息而非修改信息。换句话说，GET 请求一般不应产生副作用。就是说，它仅仅是获取资源信息，就像数据库查询一样，不会修改、增加数据，不会影响资源的状态。

幂等的意味着对同一 URL 的多个请求应该返回同样的结果。

DHCP 协议

DHCP 动态主机设置协议（Dynamic Host Configuration Protocol）是一个局域网的网络协议，使用 UDP 协议工作，主要有两个用途：给内部网络或网络服务供应商自动分配 IP 地址，给用户或者内部网络管理员作为对所有计算机作中央管理的手段。

IP 地址

为了实现网络中不同终端之间的通信，每个终端都必须有一个唯一的标识 -- IP 地址。

端口

1. 用于区分不同的应用程序
2. 端口号的范围为 0-65535，其中 0-1023 为系统的保留端口，程序尽可能别使用这些端口。
3. IP 地址和端口号组成了 Socket，Socket 是网络运行程序间双向通信链路的终结点，是 TCP 和 UDP 的基础。
4. 常用协议使用的端口：HTTP:80，FTP:21，TELNET：23。

网络地址

IP 地址由网络号（包括子网号）和主机号组成，网络地址的主机号为全 0，网络地址代表着整个网络。

广播地址

广播地址通常称为直接广播地址，是为了区分受限广播地址。

广播地址与网络地址的主机号正好相反，广播地址中，主机号为全 1。当向某个网络的广播地址发送消息时，该网络内的所有主机都能收到该广播消息。

组播地址

D 类地址就是组播地址。

A、B、C、D 类地址：

1. A 类地址以 0 开头，第一个字节作为网络号，地址范围为： 0.0.0.0 ~ 127.255.255.255；
2. B 类地址以 10 开头，前两个字节作为网络号，地址范围是：128.0.0.0 ~ 191.255.255.255；
3. C 类地址以 110 开头，前三个字节作为网络号，地址范围是：192.0.0.0 ~ 223.255.255.255；
4. D 类地址以 1110 开头，地址范围是 224.0.0.0 ~ 239.255.255.255，D 类地址作为组播地址（一对多的通信）；
5. E 类地址以 1111 开头，地址范围是 240.0.0.0 ~ 255.255.255.255，E 类地址为保留地址，供以后使用。

注：只有 A、B、C 有网络号和主机号之分，D 类地址和 E 类地址没有划分网络号和主机号。

255.255.255.255

该 IP 地址指的是受限的广播地址。受限广播地址与一般广播地址（直接广播地址）的区别在于，受限广播地址只能用于本地网络，路由器不会转发以受限广播地址为目的地址的分组；一般广播地址即可在本地广播，也可跨网段广播。例如：主机 192.168.1.1/30 发送一般广播数据包后，另外一个网段 192.168.1.5/30 也能收到该数据包；若发送受限广播数据包，则不能收到。

注：一般的广播地址（直接广播地址）能够通过某些路由器（当然不是所有的路由器），而受限的广播地址不能通过路由器。

0.0.0.0

常用于寻找自己的 IP 地址，例如在 RARP、BOOTP 和 DHCP 协议中，若某个未知 IP 地址的无盘机想要自己的 IP 地址，它就以 255.255.255.255 为目的地址，向本地范围（具体而言是被各个路由器屏蔽的范围内）的服务器发送 IP 请求分组。

回环地址

127.0.0.0/8 被作为回环地址，回环地址表示本机的地址，常用于对本机的测试，用的最多的是 127.0.0.1。

A、B、C 类私有地址

私有地址（private address）也叫专用地址，它们不会在全球使用，只具有本地意义。

A 类私有地址：10.0.0.0/8。范围是：10.0.0.0 ~ 10.255.255.255。

B 类私有地址：172.16.0.0/12，范围是：172.16.0.0 ~ 172.31.255.255。

C 类私有地址：192.168.0.0/16，范围是：192.168.0.0 ~ 192.168.255.255。

子网掩码及网络划分

随着互联网应用的不断扩大，原先的 IPv4 的弊端就逐渐暴露出来，即网络号占位太多，而主机号位太少，所以其能提供的主机地址也越来越稀缺，目前除了使用 NAT 在企业内部利用保留地址自行分配以外，通常都对一个高类别的 IP 地址进行再划分，以形成多个子网，提供给不同规模的用户群使用。

这里主要是为了在网络分段情况下有效地利用 IP 地址，通过对主机号的高位部分取作为子网号，从通常的网络位界限中扩展或压缩子网掩码，用来创建某类地址的更多子网。但创建更多的子网时，在每个子网上的可用主机地址数目会比原先减少。

什么是子网掩码

子网掩码是标志两个 IP 地址是否同属于一个子网的，也是 32 位二进制地址，其每一个为 1 代表该位的网络位，为 0 表示主机位。它和 IP 地址一样也是使用点式十进制来表示的。如果两个 IP 地址在子网掩码的按位与的计算下所得结果相同，即表明它们共属于同一子网中。

在计算子网掩码时，要注意 IP 地址中的保留地址，即 “0” 地址和广播地址，它们是指主机地址或网络地址全为 “0” 或 “1” 时的 IP 地址，它们代表着本地网络地址和广播地址，一般是不能被计算在内的。

子网掩码的计算

对于无须再划分成子网的 IP 地址来说，其子网掩码非常简单，即按照其定义即可写出：如果 B 类 IP 地址为 10.12.3.0，无需再分割子网，则该 IP 地址的子网掩码 255.255.0.0。如果它是一个 C 类地址，则其子网掩码为 255.255.255.0。其他类推。

路由选择协议

常见的路由选择协议有：RIP 协议、OSPF 协议。

RIP 协议：底层是贝尔曼福特算法，它选择路由的度量标准（metric）是跳数，最大跳数是 15 跳，如果大于 15 跳，它就会丢弃数据包。

OSPF 协议：Open Shortest Path First，开放式最短路径优化，底层是迪杰特拉算法，是链路状态路由选择协议，它选择路由的度量标准是带宽、延迟。

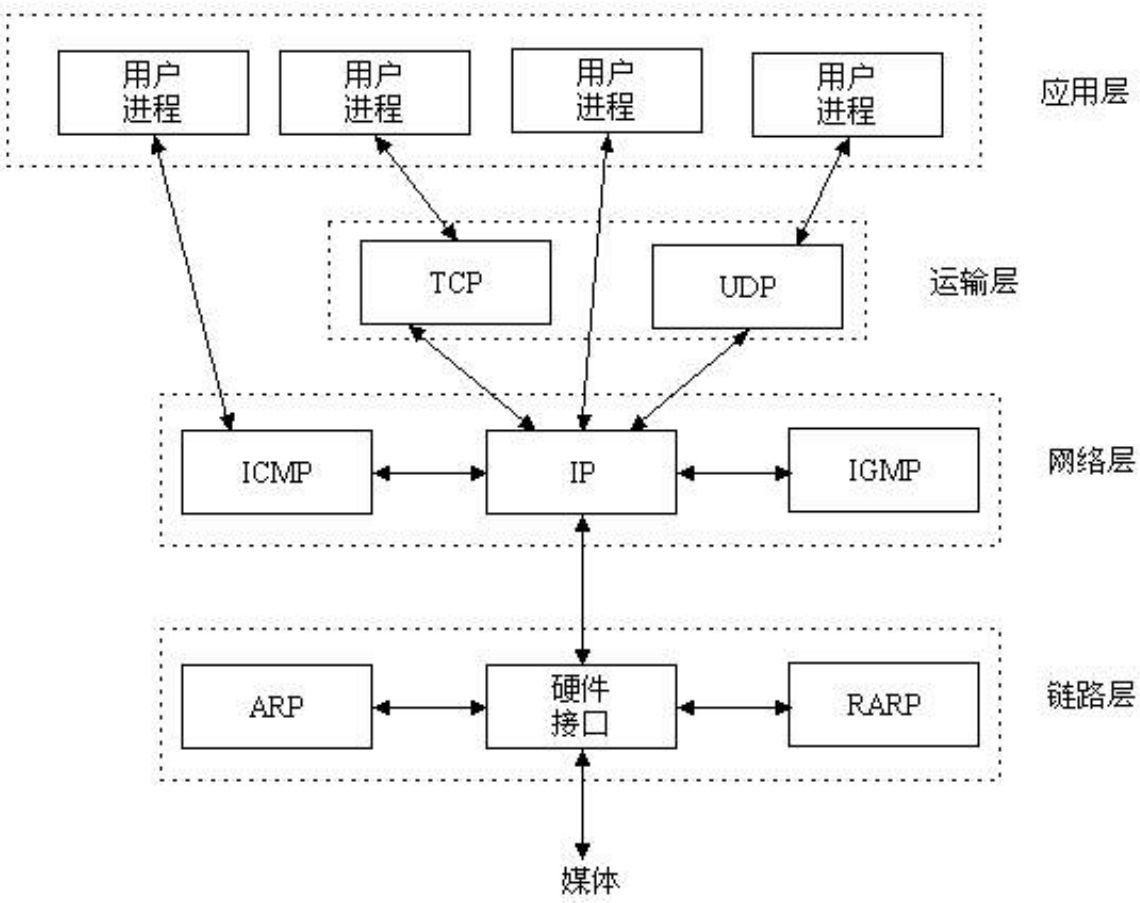
TCP/IP 协议

TCP/IP 协议是 Internet 最基本的协议、Internet 国际互联网络的基础，是一个协议簇，里面包含很多协议的。之所以命名为 TCP/IP 协议，因为 TCP 、IP 协议是两个很重要的协议，就用它两命名了。它是为广域网（WANs）设计的。

而 TCP 负责发现传输的问题，一有问题就发出信号，要求重新传输，直到所有数据安全正确地传输到目的地。 IP 是给因特网的每一台联网设备规定一个地址。

UDP （User Data Protocol，用户数据报协议）是与 TCP 相对应的协议。它也是属于 TCP/IP 协议族的一种。

下图表明了这些协议的关系：



TCP/IP 协议族包括运输层、网络层、链路层。

IP 层接收由更底层（网络接口层，例如以太网设备驱动程序）发来的数据包，并把该数据包发送到更高层 -TCP 或 UDP 层；相反，IP 层也把从 TCP 或 UDP 层接收来的数据包传送到更底层。IP 数据包是不可靠的，因为 IP 并没有做任何事情来确认数据包是否按顺序发送的或者有没有被破坏，IP 数据包中含有发送它的主机的地址（源地址）和接收他的主机的地址（目的地址）。

TCP/IP 提供点对点的连接机制，将数据应该如何封装、定址、传输、路由以及在目的地如何接收，都加以标准化。它将软件通信过程抽象化为四个抽象层，采取协议堆栈的方式，分别实现出不同通信协议。协议族下的各种协议，依其功能不同，被分别归属到这四个层次结构之中，常被视为是简化的七层 OSI 模型。它们之间好比送信的线路和驿站的作用，比如要建议送信驿站，必须要了解送信的各个细节。

NAT 协议

NAT 网络地址转换（Network Address Translation）属接入广域网（WAN）技术，是一种将私有（保留）地址转化为合法 IP 地址的转换技术，它被广泛应用于各种类型 Internet 接入方式和各种类型的网络中。原因很简单，NAT 不仅完美地解决了 IP 地址不足的问题，而且还能够有效地避免来自网络外部的攻击，隐藏并保护网络内部的计算机。

网络时延

RTT(Round-Trip Time)：往返时延，在计算机网络中它是一个重要的性能指标，它表示从发送端发送数据开始，到发送端收到来自接收端的确认（接收端收到数据后便立即发送确认），总共经历的时延。

通常，时延由发送时延、传播时延、排队时延、处理时延四个部分组成。

发送时延

发送时延是节点将数据分组发送到传播媒介所需要的时间，也就是从分组的第一个比特开始发送算起，到最后一个比特发送完毕所需要的时间。显然，发送时延与网络接口 / 信道的传输速率成反比，与数据分组的长度成正比。

传播时延

传播时延是电磁波在信道中传播一定距离所需要花费的时间，传播时延和信道的传输速率无关，而是取决于传输媒介的长度，以及某种物理形式的信号在传输媒介中的传播速度。

如电磁波在自由空间的传播速度是光速，即 3x105 km/s。电磁波在网络传输媒体中的传播速度比在自由空间中的传播速度要略低一些，在铜线中的传播速度约为 2.3x105 km/s，在光纤中的传播速度约为 2.0x105 km/s。

排队时延

排队时延是分组在所经过的网络结点的缓存队列所经历的时延，排队时延的长短主要取决于网络中当时的通信量，当网络的通信流量大时，排队时间就长，极端情况下，当网络发生阻塞导致分组丢失时，该结点的排队时延视为无穷大。

此外，在有优先级算法的网络中，排队时延还取决于数据的优先级和结点的队列调度算法。

处理时延

处理时延是分组在中间结点的存储转发过程中而进行的一些必要的处理所花费的时间，这些处理包括提取分部的首部，进行差错校验，为分组寻址和选路等。

网络端到端的时延是几种时延的总和，其计算公式是：总时延=传播时延+发送时延+排队时延+处理时延。

根据网络的不同情况，有时有些时延可以忽略不计，如在局域网中，传播时延很小可以忽略不计，当网络没有拥塞时，分组在各个结点的排队时延可以忽略不计。

往返时延（Round-Trip Time,RRT）是一个重要的性能指标，它表示从发送方发送数据开始，到发送方收到来自接收方的确认，总共经历的时延。对于负责的网络，往返时延要包括各中间结点的处理时延和转发数据时的发送时延。

在浏览器输入网址后执行的全部过程

现在假设如果在客户端浏览器中输入 http://www.baidu.com ，而 baidu.com 为要访问的服务器，下面详细分析客户端为了访问服务器而执行的一系列关于协议的操作：

1. 客户端浏览器通过 DNS 解析到 www.baidu.com 的 IP 地址 220.181.27.48，通过这个 IP 地址找到客户端到服务器的路径。客户端浏览器发起一个 HTTP 会话到 220.161.27.48，然后通过 TCP 进行封装数据包，输入到网络层。
2. 在客户端的传输层，将 HTTP 会话请求分成报文段，添加源和目的端口，如服务器使用 80 端口监听客户端的请求，客户端由系统随机选择一个端口如 5000，与服务器进行交换，服务器把相应的请求返回给客户端的 5000 端口，然后使用 IP 层的 IP 地址查找目的端。
3. 客户端的网络层不用关心应用层或者传输层的东西，主要做的是通过查找路由表确定如何达到服务器，期间可能经过多个路由器，这些都是由路由器来完成的工作，就是通过查找路由表决定通过哪个路径达到服务器。
4. 客户端的链路层，包通过链路层发送到路由器，先在本地 ARP 缓存中查找 IP 对应的 MAC 地址，如果没有则通过邻居协议查找给定 IP 地址的 MAC 地址，然后发送 ARP 请求查找目的地址，如果得到回应后（会将结果缓存一份），就可以使用 ARP 的请求应答交换的 IP 数据包进行传输了，然后发送 IP 数据到到达服务器的地址。

TCP 和 UDP

TCP 协议

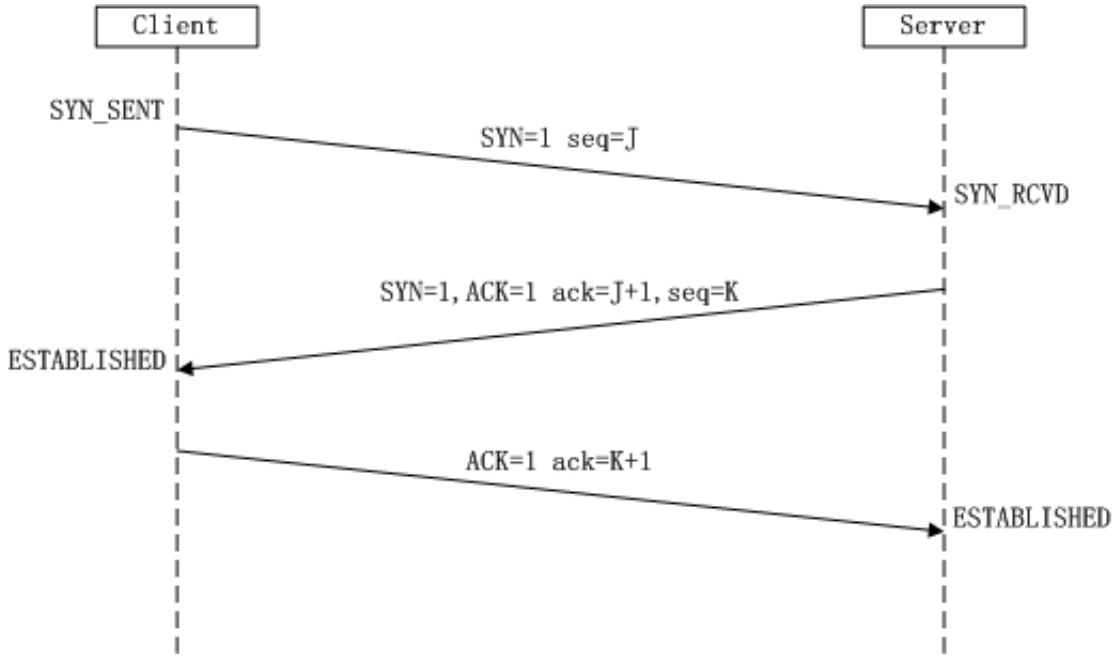
TCP （Transmission Control Protocal，传输控制协议）是面向连接、可靠的、基于字节流的通信协议，数据在传输前通过三次握手建立连接，传输完成后要断开连接，由于 TCP 是面向连接的所以只能用于端到端的通讯。

TCP 提供的是一种可靠的数据流服务，采用 “ 带重传的肯定确认 ” 技术来实现传输的可靠性，在收发时候时，都需要与对面建立可靠的连接。TCP 还采用一种称为 “ 滑动窗口 ” 的方式进行流量控制，所谓窗口实际表示接收能力，用以限制发送方的发送速度。

客户端在收发数据前要使用 connect() 函数和服务器建立连接。建立连接的目的是保证 IP 地址、端口、物理链路等正确无误，为数据的传输开辟通道。

三次握手

三次握手（Three-way Handshaking）：建立一个 TCP 连接时，需要客户端和服务端总共发送 3 个包以确认连接的建立，在 Socket 编程中，这一过程由客户端执行 connect 来发起，具体流程图如下：



1. 第一次握手：TCP 协议会组建一个数据包，并将标志位 SYN 置为 1，标识该数据包是用来建立同步连接的。同时生成一个随机数字 J，填充 “ 序号（Seq）” 字段 Seq= J，标识该数据包的序号。完成这些工作，将该数据包发送给服务器端，客户端就进入了 SYN_SEND 状态，等待 Server 确认。
2. 第二次握手：服务器收到数据包，检测到已经设置了 SYN 标志位，就知道这是客户端发来的建立连接的 “ 请求包 ”。服务器端也会组建一个数据包，并将标志位 SYN 和 ACK 置为 1，填充序号字段 Ack = J+1，随机产生一个值 Seq = K，并将数据包发送给 Client 已确认连接请求，Server 进入 SYN_RECV 状态，并为这次连接分配资源。

   SYN 标识该数据包用来建立连接，ACK 用来确认收到了刚才客户端发送的数据包。
3. 第三次握手：客户端收到数据包，检测到已经设置了 SYN 和 ACK 标志位，就知道这是服务器发来的 “ 确认包 ”。客户端会检测 “ 确认号（Ack）” 字段，看它的值是否为 J+1，如果是就说明连接建立成功，接着，客户端会继续组建数据包，并将标志位 ACK 设置为 1，表示客户端正确接收了服务器发来的 “ 确认包 ”。同时，将刚才服务器发来的数据包序号 K 加 1，用这个数字来填充 “确认号（Ack）” 字段。最后客户端将数据包发出，进入 ESTABLISED 状态，并分配资源，表示连接已经成功建立。

   服务器端收到数据包，检测到已经设置了 ACK 标志位，就知道这是客户端发来的 “ 确认包 ”。服务器会检测 “ 确认号（Ack）” 字段，看它的值是否为 K + 1，如果是就说明连接连接成功，服务器进入 ESTABLISED 状态。

至此，客户端和服务器都进入了 ESTABLISED 状态，完成了三次握手，连接建立成功，接下来客户端和服务器之间就可以传输数据了。

为什么要三次握手

三次握手就是为了防止已失效的连接请求报文段突然又传送到了服务端，因而产生错误。

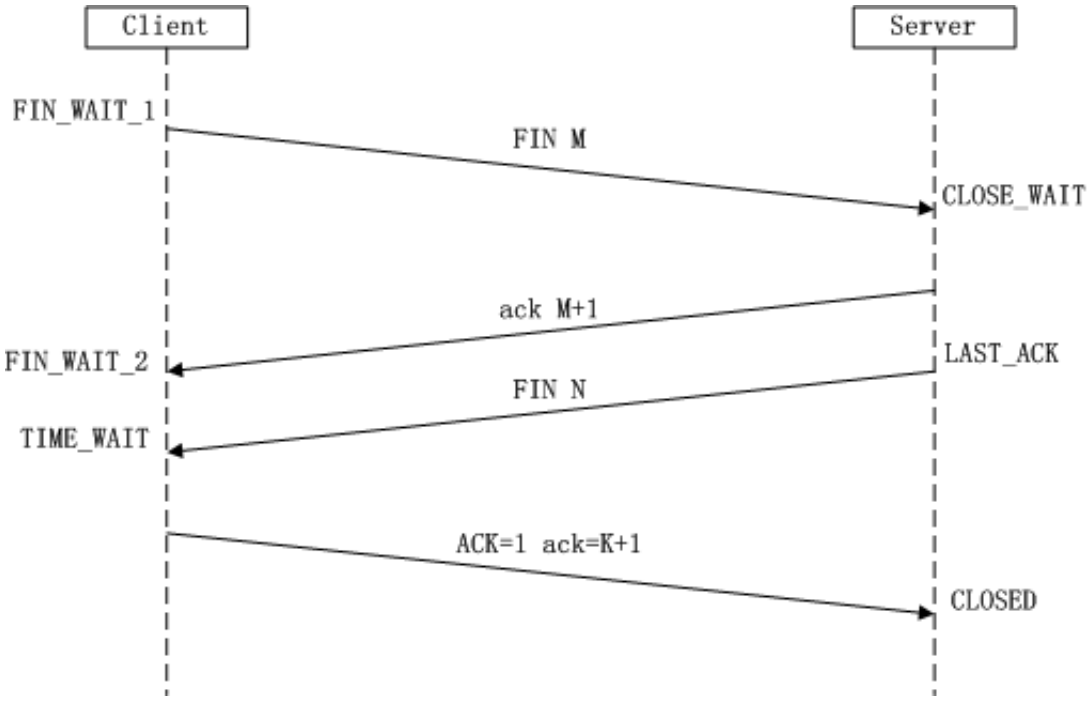
比如：client 发出的第一个连接请求报文段并没有丢失，而是在某个网络节点长时间的滞留了，以致延误到连接释放以后的某个时间才到达Server。本来这是一个早已失效的报文段，但是 server 收到此失效的连接请求报文段后，就误认为是 client 再次发出的一个新的连接请求，于是就向 client 发出确认报文段，同意建立连接。假设不采用 “ 三次握手 ”，那么只要 server 发出确认，新的连接就建立了，由于 client 并没有发出建立连接的请求，因此不理睬 server 的确认，也不会向 server 发送数据，但 server 却认为新的传输连接已经建立，并一直等待 client 发来数据。所以没有采用 “三次握手”，这种情

况下 server 的很多资源就白白浪费掉了。

四次挥手

建立连接非常重要，它是数据正确传输的前提；断开连接同样重要，它让计算机释放不再使用的资源。如果连接不能正常断开，不仅会造成数据传输错误，还会导致套接字不能关闭，持续占用资源，如果并发量高，服务器压力堪忧。

四次挥手：终止 TCP 连接，就是指断开一个 TCP 连接时，需要客户算和服务端总共发送 4 个包以确认连接的断开。在 Socket 编程中，这一过程由客户端或服务端任一方执行 close 来触发，具体流程图如下：



建立连接后，客户端和服务器都处于 ESTABLISED 状态。这时，假设客户端发起断开连接请求：

1. 第一次挥手：客户端向服务器发送 FIN 数据包，用来关闭客户端到服务器的数据发送，客户端进入 FIN_WAIT_1 状态。

   FIN 是 Finish 的缩写，表示完成任务需要断开连接。

2. 第二次挥手：服务器收到数据包后，检测到设置了 FIN 标志位，知道要断开连接，于是向客户端发送“确认包”，确认包设置标志位 ACK 为 1，确认序号为收到序号+1（与 SYN 相同，一个 FIN 占用一个序号），服务器进入 LAST_ACK 状态。

   注意：服务器收到请求后并不是立即断开连接，而是先向客户端发送“确认包”，告诉它知道了，需要准备一下才能断开连接。

   这时服务器还是可以继续向客户端发送数据的。

   客户端收到“确认包”后进入 FIN_WAIT_2 状态，等待服务器准备完毕后发送 FIN 数据包。

3. 第三次挥手：等待片刻后，服务器准备完毕，可以断开连接，于是再主动向客户端发送 FIN 包，用于关闭服务器到客户端的数据传送，然后进入 LAST_ACK 状态。

4. 第四次挥手：客户端收到服务器的 FIN 包后，会再向服务器发送 ACK 包，确认序号为收到序号+1，告诉服务器断开连接吧，客户端然后进入 TIME_WAIT 状态。

   服务器收到客户端的 ACK 包后，就断开连接，关闭套接字，进入 CLOSED 状态，完成四次挥手。

为什么要四次挥手

试想一下，加入现在客户端想断开跟 Server 的所有连接该怎么做？第一步，先停止向 Server 端发送数据，并等待 Server 的回复。但事情还没完，虽然 Client 不往 Server 发送数据了但是因为之前已经建立好平等的连接了，所以此时 Server 也有主动权向 Client 发送数据，故 Server 端还得终止主动向 Client 发送数据，并等待 Client 的确认。其实就是保证双方的一个合约的完整执行。

为什么连接是三次握手，而关闭连接却是四次挥手

因为服务器在 LISTEN 状态下，收到建立连接请求的 SYN 报文后，把 ACK 和 SYN 放在一个报文里发送给客户端。而关闭连接时，当收到对方的 FIN 报文时，仅仅标识对方不再发送数据了但是还能接收数据，服务器也未必全部数据都发送给对方了，所以服务器可以立即 close，也可以发送一些数据给对方后，再发送 FIN 报文给对方来表示同意现在关闭连接，因此，服务器 ACK 和 FIN 一般都会分开发送。

关于 TIME_WAIT 状态的说明

客户端最后一次发送 ACK 包后进入 TIME_WAIT 状态，而不是直接进入 CLOSED 状态关闭连接，这是为什么呢?

TCP 是面向连接的传输方式，必须保证数据能够正确到达目标机器，不能丢失或出错，而网络是不稳定的，随时可能会毁坏数据，所以机器 A 每次向机器 B 发送数据包后，都要求机器 B “ 确认 ”，回传 ACK 包，告诉机器 A 收到了，这样机器 A 才能知道数据传送成功了。如果机器 B 没有回传 ACK 包，机器 A 会重新发送，直到机器 B 回传 ACK 包。

客户端最后一次向服务器回传 ACK 包时，有可能会因为网络问题导致服务器收不到，服务器会再次发送 FIN 包，如果这时客户端完全关闭了连接，那么服务器无论如何也收不到 ACK 包了，所以客户端需要等待片刻、确认对方收到 ACK 包后才能进入 CLOSED 状态，TCP 连接就关闭了。那么，要等待多久呢?

数据包在网络中是有生存时间的，超过这个时间还未到达目标主机就会被丢弃，并通知源主机。这称为报文最大生存时间（MSL，Maximum Segment Lifetime）。TIME_WAIT 要等待 2MSL 才会进入 CLOSED 状态。ACK 包到达服务器需要 MSL 时间，服务器重传 FIN 包也需要 MSL 时间，2MSL 是数据报往返的最大时间，如果 2MSL 后还未收到服务器重传的 FIN 包，就说明服务器已经收到了 ACK 包。

使用 TCP 的协议

使用 TCP 的协议：FTP （文件传输协议）、Telnet（远程登录协议）、SMTP（简单邮件传输协议）、POP3（和 SMTP 协议，用于接收邮件）、HTTP 协议等。

UDP 协议

UDP（User Datagram Protocol） 用户数据报协议，是面向无连接的通讯协议，传输数据之前源端和终端不建立连接，所以可以实现广播发送。当它想传送时就简单地去抓取来自应用程序的数据，并尽可能快地把它扔到网络上。

在发送端，UDP 传送数据地速度仅仅是受应用程序生成数据的速度、计算机的能力和传输带宽的限制，在接收端，UDP 把每个消息段放在队列中，应用程序每次从队列中读一个消息段。

相比 TCP，UDP 通讯时不需要接收方确认，也无需建立连接，属于不可靠的传输，结构简单，但是无法保证正确性，可能会出现丢包现象，实际应用中要求程序员编码验证。

UDP 和 TCP 位于同一层，但它不管数据包的顺序、错误或重发。因此，UDP 不被应用于那些使用虚电路的面向连接的服务，UDP 主要用于那些面向查询-应答的服务，例如 NFS，相对于 FTP 或 Telent，这些服务需要交换的信息量较小。

每个 UDP 报文分 UDP 报头和 UDP 数据区两部分。UDP 报头由 4 个域组成，其中每个域个占用 2 个字节（16 位），具体如下：

1. 源端口号
2. 目的端口号
3. 数据报长度
4. 校验值

使用 UDP 的协议

使用 UDP 协议包括：TETP（简单文件传输协议）、SNMP（简单网络管理协议）、DNS（域名解析协议）、NFS、BOOTP。

TCP 与 UDP 的区别

1. TCP 是面向连接的，可靠的字节流服务；
2. UDP 是面向无连接的，不可靠的数据报服务。

TCP 粘包

长连接与短连接

长连接：Client 方与 Server 方先建立通讯连接，连接建立后不断开，然后再进行报文发送和接收。

短连接：Client 方与 Server 进行一次报文收发交易时才进行通讯连接，交易完毕后立即断开连接。此种方式常用于一点对多点通讯，比如多个 Client 连接一个 Server。

什么时候需要考虑粘包问题

1. 如果利用 TCP 每次发送数据，就与对方建立连接，然后双方发送完一段数据后，就关闭连接，这样就不会出现粘包问题（因为只有一种包结构，类似于 HTTP 协议）。关闭连接主要要双方都发送 close 连接（参考 TCP 关闭连接）。

   如： A 需要发送一段字符串给 B，那么 A 与 B 建立连接，然后发送双方都默认好的协议字符如 “hello give me sth abour yourself”，然后 B 收到报文后，就将缓冲区数据接收，然后关闭连接，这样粘包问题不用考虑到，因为大家都知道是发送一段字符。
2. 如果发送数据无结构，如文件传输，这样发送方只管发送，接收方只管接收存储就 ok，也不用考虑粘包。
3. 如果双方建立连接，需要在连接后一段时间内发送不同结构数据，如连接后，有好几种结构：

   1. hello give me sth abour yourself
   2. Don't give me sth abour yourself

   那这样的话，如果发送方连续发送这两个包出去，接收方一次接收可能会是 "hello give me sth abour yourselfDon't give me sth abour yourself"，这样接收方就无法分辨了到底是要干嘛了，因为协议没有规定这样的字符串，所以要处理把它分包，怎么分也需要双方组织一个比较好的包结构，所以一般可能会在头加一个数据长度之类的包，以确保接收。

粘包出现的原因

在流传输中出现，UDP 不会出现粘包，因为它有消息边界。

1. 发送端需要等缓冲区满才发送出去，造成粘包。
2. 接收方不及时接收缓冲区的包，造成多个包接收。

解决方法

为了避免粘包问题，可采取以下几种措施：

1. 一是对于发送方引起的粘包现象，用户可通过编程设置来避免，TCP 提供了强制数据立即传送的操作指令 push，TCP 软件收到该操作指令后，就立即将本段数据发送出去，而不必等待发送缓冲区满；
2. 二是对于接收方引起的粘包，则可通过优化程序设计、精简接收进程工作量、提高接收进程优先级等措施，使其及时接收数据，从而尽量避免出现粘包现象；
3. 三是由接收方控制，将一包数据按结构字段，人为控制分多次接收，然后合并，通过这种手段来避免粘包。

以上提到的三种措施，都有不足之处：

1. 第一种编程设置方法虽然可以避免发送方引起的粘包，但它关闭了优化算法，降低了网络发送效率，影响应用程序的性能，一般不建议使用。
2. 第二种方法只能减少出现粘包的可能性，但并不能完全避免粘包，当发送频率较高时，或由于网络突发可能使某个时间段数据包到达接收方较快，接收方还是有可能来不及接收，还是导致粘包。
3. 第三种方法虽然避免了粘包，但应用程序的效率较低，对实时应用的场合不适合。

而对于解决粘包问题，基于 TCP 开发的通讯程序，有个很重要的问题需要解决，就是封包和拆包。

为什么基于 TCP 的通讯程序需要进行封包和拆包

TCP 是个 “ 流 ” 协议，所谓流，就是没有界限的一串数据，是连成一片的，其间是没有分界线的。但一般通讯程序开发是需要定义一个个相互独立的数据包的，比如用于登录的数据包、用于注销的数据包。由于 TCP " 流 " 的特性以及网络状况，在进行数据传输时会出现以下几种情况。

假设连续调用两次 send 分别发送两端数据 data1 和 data2，在接收端有以下几种接收情况：

1. 先接收到 data1，然后接收到 data2；
2. 先接收到 data1 的部分数据，然后接收到 data1 余下的部分以及 data2 的全部；
3. 先接收到了 data1 的全部数据和 data2 的部分数据，然后接收到了 data2 的余下的数据。
4. 一次性接收到了 data1 和 data2 的全部数据。

对于 2、3、4 的情况就是经常说的 “ 粘包 ”，就需要把接收到的数据进行拆包，拆成一个独立的数据包。为了拆包就必须在发送端进行封包。

另：对于 UDP 来说就不存在拆包的问题，因为 UDP 是个 “ 数据包 ” 协议，也就是两段数据间是有界限的，在接收端要么接收不到数据要么就是接收一个完整的一段数据，不会少接收也不会多接收。

为什么会出现 2、3、4 的情况

“ 粘包 ” 可能发生在发送端也可发生在接收端。

1. 由 Nagle 算法造成的发送端的粘包：Nagle 算法是一种改善网络传输效率的算法。简单的说，当提交一段数据给 TCP 发送时，TCP 并不立刻发送此段数据，而是等待一小段时间，看看在等待期间是否还有要发送的数据，若有则会一次把这两段数据发送出去。这是对 Nagle 算法一个简单的解释，像 3 和 4 的情况就有可能是 Nagle 算法造成的。
2. 接收端接收不及时造成的接收端粘包：TCP 会把接收到的数据存在自己的缓冲区中，然后通知应用层取数据。当应用层由于某些原因不能及时的把 TCP 的数据取出来，就会造成 TCP 缓冲区中存放了几段数据。

怎样封包和拆包

在遇到 “ 粘包 ” 的问题时，有一个笨办法就是：通过在两次 send 之间调用 sleep 来休眠一小段时间来解决。这个解决方法的缺点是显而易见的，使传输效率大大降低，而且也并不可靠。然后通过应答得方式来解决，尽管在大多数时候是可行的，但是不能解决像 2 的那种情况，而且采用应答方式增加了通讯量，加重了网络负荷。最后就是对数据包进行封包和拆包的操作。

封包

封包就是给一段数据加上包头，这样一来数据包就分为包头和包体两部分内容了。包头其实上是个大小固定的结构体，其中有个结构体成员变量表示包体的长度，这是个很重要的变量，其他的结构体成员可根据需要自己定义。

根据包头长度固定以及包头中含有包体长度的变量就能正确的拆分出一个完整的数据包。

拆包

对于拆包目前最常用的是以下两种方式：

动态缓冲区暂存方式

之所以说缓冲区是动态的是因为当需要缓冲的数据长度超出缓冲区的长度时会增大缓冲区长度。

大概过程描述如下：

1. 为每一个连接动态分配一个缓冲区，同时把此缓冲区和 SOCKET 关联，常用的是通过结构体关联。
2. 当接收到数据时首先把此段数据存放在缓存区中。
3. 判断缓冲区中的数据长度是否够一个包头的长度，如不够，则不进行拆包操作。
4. 根据包头数据解析出里面代表包体长度的变量。
5. 判断缓冲存区除包头外的数据长度是否够一个包体的长度，如不够，则不进行拆包操作。
6. 取出整个数据包。这里的 “ 取 ” 的意思是不光从缓冲区中拷贝出数据包，而且要把此数据包从缓冲区中删除掉。删除的办法就是把此包后面的数据移动到缓冲区的起始地址。

这个方法有两个缺点：1. 为每个连接动态分配一个缓冲区增大了内存的使用。2. 有三个地方需要拷贝数据，一个地方是把数据存放在缓冲区，一个地方是把完整的数据包从缓冲区取出来，一个地方是把数据包从缓冲区中删除。

对于这些缺点，可以采用环形缓冲来改进。但是这种改进方法还是不能解决第一个缺点以及第一个数据拷贝，只能解决第三个地方的数据拷贝（这个地方是拷贝数据最多的地方），第 2 种拆包方式会解决这两个问题。

环形缓冲实现方式是定义两个指针，分别指向有效数据的头和尾。在存放数据和删除数据时只是进行头尾指针的移动。

利用底层的缓冲区来进行拆包

由于 TCP 也维护了一个缓冲区，所以完全可以利用 TCP 的缓冲区来缓存数据，这样一来就不需要为每一个连接分配一个缓冲区了。另一方面 recv 或者 wsarecv 都有一个参数，用来表示要接收多少长度的数据，利用这两个条件就可以对第一种方法进行优化。

对于阻塞 SOCKET 来说，可以利用一个循环来接收包头长度的数据，然后解析出代表包体长度的那个变量，再用一个循环来接收包体长度的数据。

对于非阻塞的 SOCKET，比如完成端口，可以提交接收包头长度的数据的请求，然后判断接收的数据长度是否等于包头长度，若等于，则提交接收包体长度的数据的请求，若不等于则提交接收剩余数据的请求。当接收包体时，采用类似的方法。

一个包没有固定长度，以太网限制在 46-1500 字节，1500 就是以太网的 MTU，超过这个量，TCP 会为 IP 数据包设置偏移量进行分片传输，现在一般可允许应用层设置 8k （NTFS 系）的缓冲区，8k 的数据由底层分片，而应用看来只是一次发送。windows 的缓冲区经验值是 4k，Socket 本身分为两种，流（TCP）和数据报（UDP）。

1. 通信长度，这个是自己决定的，没有系统强迫要发多大的包，实际应该根据需求和网络状况来决定。

   对于 TCP ，这个长度可以大点，但要知道，Socket 内部默认的收发缓冲区大小大概是 8K，可以用 SetSockOpt 来改变。

   但对于 UDP 而言，就不要太大，一般在 1024 至 10K。

   注意一点，无论发多大的包，IP 层和链路层都会把包进行分片发送，一般局域网就是 1500 左右，广域网就只有几十字节。分片后的包将经过不同的路由到达接收方，对于 UDP 而言，要是其中一个分片丢失，那么接收方的 IP 层将把这个发送包丢弃，这就形成丢包。

   显然，要是一个 UDP 发包很大，它被分片后，链路层丢失分片的几率就很大，这个 UDP 包，就很容易丢失，但是太小又影响效率。

   配置这个值，最好可以根据不同的环境来调整到最佳状态。

   send() 函数返回了实际发送的长度，在网络不断的情况下，它绝不会返回（发送失败的）错误，最多就是返回 0。

   对于 TCP 可以自己写一个循环发送。当 send 函数返回 SOCKET_ERROR 时，才标志着有错误。

   但对于 UDP ，不要写循环发送，否则将给接收带来极大的麻烦。所以 UDP 需要用 SetSockOpt 来改变 Socket 内部 Buffer 的大小，以能容纳发包。

   明确一点，TCP 作为流，发包是不会整包到达的，而是源源不断的到，那接收方就必须组包。

   而 UDP 作为消息或数据报，它一定是整包到达接收方。

2. 关于接收，一般的发包都有包边界，首要的就是这个包的长度要让接收方知道，于是就有个包头信息。

   对于 TCP，接收方先收这个包头信息，然后再收包数据。一次收齐整个包也可以，可要对结果是否收齐进行验证。这也就完成了组包过程。UDP 只能整包接收。

   要是提供的接收 Buffer 过小，TCP 将返回实际接收的长度，余下的还可以收，而 UDP 不同的是，余下的数据被丢弃并返回 WSAEMSGSIZE 错误。

注意 TCP，要是提供的 Buffer 很大，那么可能收到的就是多个发包，必须分离它们，还有就是当 Buffer 太小，而一次收不完 Socket 内部的数据，那么 Socket 接收事件（OnReceive），可能不会再触发。使用事件方式进行接收时，密切注意这点。这些特性就是体现了流和数据报的区别。

TCP 的可靠传输机制

TCP 的报文是交给 IP 层传送的，但是 IP 层只能提供尽最大努力交付的服务，也就是说，TCP 下面的网络所提供的是不可靠传输，其实就是传输信道是不可靠的（所谓的信道，就是指连接信号发送方和接收方的传输线路，包括双绞铜线、同轴电缆、光纤、陆地无线电或者卫星无线电等物理媒体）这时，传输层的可靠传输机制就显得特别重要。

可靠传输的要求

所谓的可靠，就是能保证数据的正确性，无差错、不丢失、不重复、并且按序达到。

TCP 可靠传输的机制原理

TCP 首先采用三次握手来建立连接、四次挥手来释放连接。其次 TCP 采用了连续 ARQ 协议，即自动重传请求（Automatic Repeat-reQuest）来保证数据传输的正确性，使用滑动窗口协议来保证接收方能够及时处理接收到的数据，进行流量控制。最后 TCP 使用慢开始、拥塞避免、快重传、快恢复来进行拥塞控制，避免网络拥堵。下面我们将分别介绍三次握手、四次挥手、连续ARQ、拥塞控制。

三次握手的重要性

假如只进行一次握手，客户端发送连接请求后，没有收到服务端的应答，是没法判断连接是否成功的。

假如只进行两次握手，客户端发送连接请求后，会等待服务器端的应答。但是会出现的问题是，假如客户端的SYN迟迟没有到达服务器端，此时客户端超时后，会重新发送一次连接，假如重发的这次服务器端收到了，且应答客户端了，连接建立了。但是建立后，第一个 SYN 也到达服务端了，这时服务端会认为这是一个新连接，会再给客户端发送一个 ACK，这个 ACK 当然会被客户端丢弃。但是此时服务器端已经为这个连接分配资源了，而且服务器端会一直维持着这个资源，会造成浪费。

三次握手，两次握手的问题在于服务器端不知道 SYN 的有效性，所以如果是三次握手，服务器端会等待客户端的第三次握手，如果第三次握手迟迟不来，服务器端就会释放相关资源。但是有人会问，假如第三次握手没有到达服务器端呢？但是这时客户端认为连接已经建立了。但是其实这种情况下，只要客户端向服务器端写数据，就会收到服务器端的 RST 应答，这时客户端就能知道出现问题了。

RST：在 TCP 协议中，rst 段标识复位，用来异常的关闭连接。在 TCP 的设计中它是不可或缺的，发送 rst 段关闭连接时，不必等缓冲区的数据都发送出去，直接丢弃缓冲区中的数据。而接收端收到 rst 段后，也不必发送 ack 来确认。

四次挥手的原因

其实是客户端和服务端的两次挥手，也就是客户端和服务端分别释放连接的过程。可以看到，客户端在发送完最后一次确认之后，还要等待2MSL的时间。主要有两个原因，一个是为了**让B能够按照正常步骤进入CLOSED状态**，二是**为了防止已经失效的请求连接报文出现在下次连接中**。

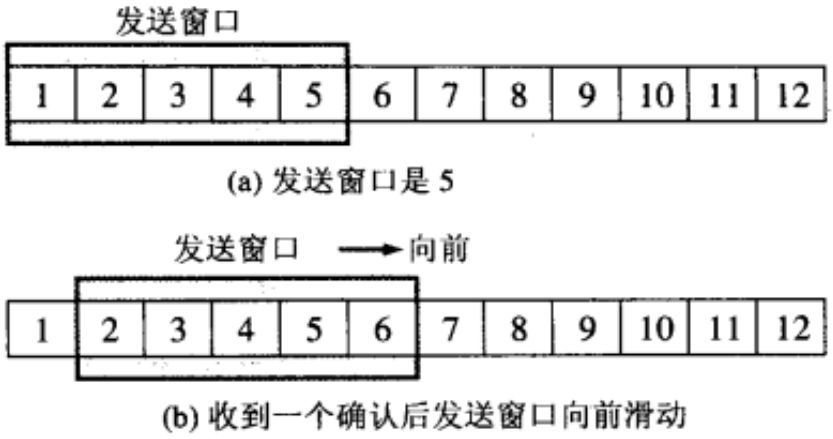
解释：

1）、由于客户端最后一个 ACK 可能会丢失，这样 B 就无法正常进入 CLOSED 状态。于是 B 会重传请求释放的报文，而此时 A 如果已经关闭了，那就收不到 B 的重传请求，就会导致 B 不能正常释放。而如果 A 还在等待时间内，就会收到 B 的重传，然后进行应答，这样 B 就可以进入 CLOSED 状态了。

2）、在这 2MSL 等待时间里面，本次连接的所有的报文都已经从网络中消失，从而不会出现在下次连接中。

连接 ARQ 协议

在介绍连续ARQ协议之前，我们要先说一下停止等待协议。停止等待协议的优点就是比较简单，好实现。基本原理是A发送一个分组，发送完就暂停发送，然后一直等待接收方B的应答。A在收到B的应答之后，再发送下一个分组。如果中途B发现包有错或者就没有收到包，那么也就不会向A发送应答包。此时A规定时间内要是没有等到应答包，就会重新传送刚才的分组，也就是所谓的**超时重传。**从这个过程也可以看出，假如A发送分组的时间为Ta，B发送确认的时间为Tb，往返时间为Tr，那么完整发送一个分组的时间为Ta+Tb+Tr。过了这个时间后，第二个分组才开始发送。这样子的信道利用率和传输效率就比较低了。这时就引入**连续ARQ**了。



从上面的图来看，图（a）是发送方维持的发送窗口，它的意义是：位于发送窗口内的5个分组都可以连续发送出去，而不需要等待对方的确认，这样就提高了信道利用率。

连续ARQ协议规定，发送方没法送一个确认，就把发送窗口向前滑动一个分组的位置。例如上面的图（b），当发送方收到第一个分组的确认，就把发送窗口向前移动一个分组的位置。如果已经发送了前5个分组，则现在可以发送窗口内的第6个分组。

接收方一般都是采用**累积确认**的方式。也就是说接收方不必对收到的分组逐个发送确认。而是在收到几个分组后，对按序到达的最后一个分组发送确认。如果收到了这个分组确认信息，则表示到这个分组为止的所有分组都已经正确接收到了。

累积确认的优点就不用多说了，缺点是，不能正确的反映收到的分组，比如发送方发送了前5个分组，而中间的第3个分组丢失了，这时候接收方只能对前2个发出确认。而不知道后面3个分组的下落，因此只能把后面的3个分组都重传一次，这种机制叫**Go-back-N(回退N)**，表示需要再退回来重传已发送过的N个分组。

拥塞控制

1）TCP流量控制

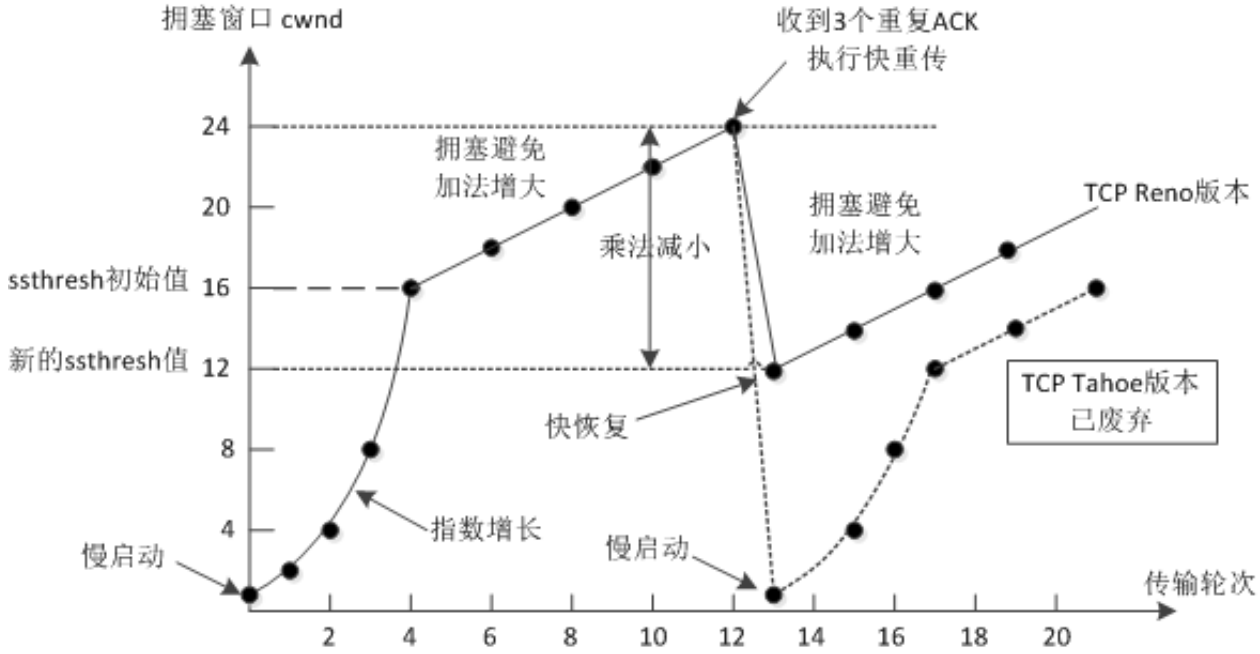
拥塞控制就是防止过多的数据注入到网络中，这样可以使网路中的路由器或链路不致于过载。而流量控制往往是指点对点通信量的控制，是个端到端的问题。流量控制所要做的就是控制发送端发送速率，以便使接收端来得及接受。利用滑动窗口机制就可以实现流量的控制。原理就是运用TCP报文中的窗口大小字段来控制，发送的发送窗口不可以大于接收方的接收窗口大小。如果接收方没有足够的缓存来接收数据，发送方就会收到一个零窗口的通知。此时发送方停止发送。并且定时发送一个窗口探测报文来探测接收方的接收能力。

2）TCP拥塞控制

在某段时间内，若对网络中的某一资源的需求超过了该资源所能提供的可用部分，网络的性能就要变差，这种叫做拥塞。

拥塞控制就是防止过多的数据注入到网络中，这样可以使网路中的路由器或链路不致于过载。RFC2581定义了四种拥塞控制算法，即**慢开始，拥塞避免、快重传、快恢复**。

下图是慢启动和拥塞避免的一个可视化描述。我们以段为单位来显示cwnd和ssthresh，但它们实际上都是以字节为单位进行维护的。



拥塞窗口的概念：

发送报文段速率的确定，纪要根据接收端的能力，又要从全局考虑不要使网路发生拥塞，这由接收窗口和拥塞窗口两个状态量来决定。**接收窗口**又称通知窗口，是**接收端**根据目前接收缓存大小所许诺的最新窗口值，是来自接收端的流量控制。**拥塞窗口cwnd**是发送端根据自己估计的网络拥塞程度而设置的窗口值，是来自**发送端**的流量控制。

慢启动原理：

当主机开始发送数据时，如果立即将较大的发送窗口的全部数据注入网路中，那么由于不清楚网络的情况，有可能引起拥塞。比较好的方式是试探一下，即从小到大逐渐增大发送端的拥塞控制窗口数值。当窗口值逐渐增大时，为了防止cwind的增长导致网络拥塞，还需要一个变量--慢开始门限ssthresh

拥塞控制

具体过程为：

TCP连接初始化，将拥塞窗口设置为1。接下来执行**慢开始算法，**当cwind==ssthresh时，开始执行**拥塞避免算法**：cwnd按照现行规律增长。当网络发生拥塞时，吧ssthresh值更新为拥塞前ssthresh的一半，cwnd重新设置为1。重复执行上个步骤。

快重传和快恢复

一条TCP连接有时会因等待重传计时器的超时而空闲较长的时间，慢开始和拥塞避免无法很好的解决这类问题，因此提出了快重传和快恢复的拥塞控制方法。

快重传算法并非取消了**重传机制**，只是在某些情况下更早的重传丢失的报文段（如果当发送端接收到三个重复的确认ACK时，则断定分组丢失，立即重传丢失的报文段，而不必等待重传计时器超时）。

例如：M1，M2，M3 -----> M1,M3,缺失M2，则接收方向发送方持续发送M2重复确认，当发送方收到M2的三次重复确认，则认为M2报文丢失，启动快重传机制，重传数据，其他数据发送数据放入队列，待快重传结束后再正常传输。

快恢复算法有以下两个要点：

1）当发送方连续收到接收方发来的三个重复确认时，就执行“乘法减小”算法，把慢开始门限减半，这是为了预防网络发生拥塞。

2）由于发送方现在认为网络很可能没有发生拥塞，因此现在不执行慢开始算法，而是把**cwnd(拥塞窗口)**值设置为慢开始门限减半后的值，然后开始执行拥塞避免算法，**使拥塞窗口的线性增大**。

#

* Socket 知识

Socket 是应用层与 TCP/IP 协议族通信的中间软件抽象层，它是一组接口。在设计模式中，Socket 其实就一个门面模式，它把复杂的 TCP/IP 协议族隐藏在 Socket 接口后面，对用户来说，一组简单的接口就是全部，让 Socket 去组织数据，以符合指定的协议。

* Socket 常用函数接口详解

HTTP

HTTP 协议是 Hyper Text Transfer Protocol (超文本传输协议) 的缩写。基于 TCP/IP 通信协议来传输数据（HTML 文件，图片文件，查询结果等）。HTTP 不仅能传送文本跳转所必须的信息，而且也能传输任何能在互联网上得到的信息。

HTTP 是面向事务的应用层协议。所谓面向事务就是指一系列的信息交换，这些交换的信息是一个整体，要么这些信息全部交换，要么就不交换。

HTTP 协议主要就是用来进行客户端和服务器端之间进行通信的标准协议。HTTP 主要规定了客户端如何与服务器端建立链接，客户端如何从服务器请求数据，服务器如何响应请求，以及最后连接如何关闭。

HTTP 协议用于从万维网（WWW:World Wide Web）服务器传输超文本到本地浏览器的传送协议。每个万维网网点都有一个服务器进程，用来监视 TCP 的端口 80，以便发现是否有浏览器向它发出建立连接请求，一旦监听到需要建立 TCP 连接，浏览器就会向万维网发出浏览某个页面的请求，而万维网对这个请求的页面做出响应，最后，TCP 连接释放。在浏览器和服务器之间的请求与响应的交互，必须按照一定的格式和规则，这些格式和规则就是超文本传输协议 HTTP。

HTTP 工作原理

HTTP 协议工作于客户端 - 服务端架构上。浏览器作为 HTTP 客户端通过 URL 向 HTTP 服务端发送所有请求。

Web 服务器有：Apache 服务器，IIS 服务器（Internet Information Service）等。

Web 服务器根据接收到的请求后，向客户端发送响应信息。

HTTP 默认端口号为 80，也可以改为 8080 或者其他端口。

一次网络请求的过程

1. 浏览器向 DNS 服务器请求解析该 URL 中的域名所对应的 IP 地址。
2. 解析出 IP 地址后，根据该 IP 地址和默认端口 80，和服务器建立 TCP 连接。
3. 浏览器发出读取文件（URL 中域名后面部分对应的文件）的 HTTP 请求，该请求报文作为 TCP 三次握手的第三个报文的数据发送给服务器。
4. 服务器对浏览器请求作出响应，并把对应的 html 文本发送给浏览器。
5. 释放 TCP 连接。
6. 浏览器将该 html 文本显示内容。

HTTP 三点注意事项

1. **HTTP 是无连接**：无连接的含义是限制每次连接只处理一个请求。服务端处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。虽然 HTTP 需要用到 TCP 作为传输层协议，但是在通信双方交换 HTTP 报文之前不需要建立 HTTP 连接。
2. **HTTP 是媒体独立的**：这意味着，只要客户端和服务端知道如何处理的数据内容，任何类型的数据都可以通过 HTTP 发送。客户端以及服务端指定使用合适的 MIME-type 内容类型。
3. **HTTP 是无状态**：HTTP 协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

浏览器显示的内容都有 HTML、XML、GIF、FLASH 等，浏览器是通过 MIME Type 区分它们，决定用什么内容什么形式来显示。

MIME Type 是该资源的媒体类型，MIME Type 不是个人指定的，是通过互联网（ IETF ）组织协商，以 RFC（ 是一系列以编号排定的文件，几乎所有的互联网标准都有收录在其中 ） 的形式作为建议的标准发布在网上的，大多数的 Web 服务器和用户代理都会支持这一规范（ 顺便说一句，Email 附件的类型也是通过 MIME Type 指定的 ）。

媒体类型通常通过 HTTP 协议，由 Web 服务器告知浏览器的，更准确的说，是通过 Content-Type 来表示的。

如果是某个客户端自己定义的格式，MIME Type 一点只能以 application/x- 开头。

HTTP 的版本

HTTP/1.0 的主要缺点：每当请求一个文档，就需要两倍的 RTT （ Round-Trip Time，往返时延，是指数据从网络一端传到另一端所需的时间) 的开销（一个 RTT 时用来 TCP 的连接，另一个 RTT 用来请求和接收万维网文档），如果有很多个对象需要建立连接，那么每一次连接都需要 2* RTT 的时间开销。另一方面，每一次请求都需要建立 TCP 连接，并且万维网通常都服务于大量的请求，所以这种非持续性的连接会导致万维网的负荷过大。HTTP/1.0 中浏览器与服务器只保持短暂的连接，连接无法复用。也就

说每个 TCP 连接只能发送一个请求。发送数据完毕，连接就关闭，如果还要请求其他资源，就必须再新建一个连接。

HTTP/1.1 采用的是持续连接，就是在建立连接后的一段时间内仍然保持连接，使用同一个客户和服务器传送后续的请求和响应。客户端和服务器发现对方一段时间没有活动，就可以主动关闭连接。或者客户端在最后一个请求时，主动告诉服务端要关闭连接。HTTP/1.1 版还引入了管道机制（pipelining），即在同一个 TCP 连接里面，客户端可以同时发送多个请求。但是对于执行者来说，还是需要按照顺序，先执行完一件事以后再执行另外一件事。

HTTP/2 采用了多路复用，即在一个连接里，客户端和浏览器都可以同时发送多个请求或回应，而且不用按照顺序一一对应。能这样做有一个前提，就是 HTTP/2 进行了二进制分帧，即 HTTP/2 会将所有传输的信息分割为更小的消息和帧（frame），并对它们采用二进制格式的编码。除此之外，还有一些其他的优化，比如做 Header 压缩、服务端推送等。

持续连接有两种工作方式，非流水线和流水线。

非流水线：客户在收到上一个响应才能发出下一个请求。缺点：服务器在发送了一个对象后，TCP 处于空闲状态，浪费资源。

流水线：客户在收到 HTTP 的响应报文之前，就能够接着发送新的请求。

HTTP 消息结构

HTTP 是基于客户端/服务端（C/S）的架构模型，通过一个可靠的链接来交换信息，是一个无状态的请求 / 响应协议。

一个 HTTP " 客户端 "是一个应用程序（ Web 浏览器或其他任何客户端 ），通过连接服务器达到发送一个或多个 HTTP 的请求的目的。

一个 HTTP “ 服务器 ” 同样也是一个应用程序（通常是一个 Web 服务，如 Apache Web 服务器或 IIS 服务器等），通过接收客户端的请求并向客户端发送 HTTP 响应数据。

HTTP 使用统一资源标识符（Uniform Resource Identifiers,URL） 来传输数据和建立连接。

一旦建立连接后，数据消息就通过类似 Internet 邮件所适用的格式和多用途 Internet 邮件扩展（MIME）来传送。

客户端请求消息

客户端发送一个 HTTP 请求到服务器的请求消息包括以下格式：**请求行**（request line）（ 方法 + URL + 版本 ）、**请求头部**（ header ）、**空行**和**请求数据**四个部分组成，下图给出了请求报文的一般格式：

请求行：用来说明请求类型，要访问的资源以及所使用的 HTTP 版本。

请求头部：紧接着请求行之后的部分，用来说明服务器要使用的附加信息。

空行：请求头部后面的空行是必须的。

请求数据：也称为请求主体，可以添加任意的其他数据。

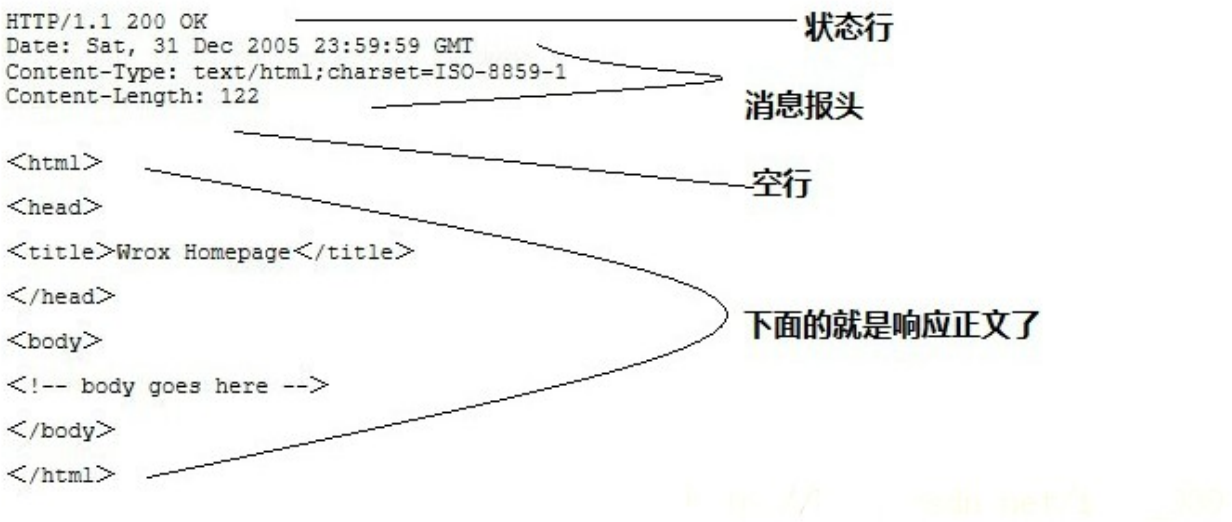
请求头部的首部字段名和含义

字段名	含义
Accept	浏览器可接受的 MIME 类型。
Accept-Charset	浏览器可接受的字符集。
Accept-Encoding	浏览器能够进行解码的数据编码方式，比如 gzip。Servlet 能够向支持 gzip 的浏览器返回经 gzip 编码的 HTML 页面。许多情形下这可以减少 5 到 10 倍的下载时间。
Accept-Language	浏览器所希望的语言种类，当服务器能够提供一种以上的语言版本时要用到。
Authorization	授权信息，通常出现在对服务器发送的 WWW-Authorizate 头的应答中。
Content-Length	表示请求消息正文的长度。
Host	客户机通过这个头告诉服务器，要访问的主机名。Host 头域指定请求资源的 Intenet 主机和端口号，必须表示请求 url 的原始服务器或网管的位置。HTTP/1.1 请求必须包含主机头域，否则系统会以 400 状态码返回。
If-Modified-Since	客户机通过这个头告诉服务器，资源的缓存时间。只有当所请求的内容在指定的时间后又经过修改才返回它，否则返回 304 “Not Modified” 应答。
Referer	客户机通过这个头告诉服务器，它是从哪个资源来访问服务器的（防盗链）。包含一个 URL，用户从该 URL 代表的页面出发访问当前请求的页面。
User-Agent	User-Agent 头域的内容包含发出请求的用户信息。浏览器类型，如果 Servlet 返回的内容与浏览器类型有关则该值非常有用。
Cookie	客户机通过这个头可以向服务器带数据，这是最重要的请求头信息之一。
Pragma	指定 “ no-cache ” 值表示服务器必须返回一个刷新后的文档，即使它是代理服务器而且已经有了页面的本地拷贝。
From	请求发送者的 email 地址，由一些特殊的 Web 客户程序使用，浏览器不会用到它。
Connection	处理完这次请求后是否断开连接还是继续保持连接。如果 Servlet 看到这里的值为 “Keep-Alive” ，或者看到请求使用的是 HTTP 1.1(HTTP 1.1 默认进行持久连接)，它就可以利用持久连接的优点，当页面包含多个元素时（例如 Applet、图片），显著地减少下载所需要的时间。要实现这一点，Servlet 需要在应答中发送一个 Content-Length 头，最简单的实现方法是：先把内容写入

	ByteArrayOutputStream，然后在正式写出内容之前计算它的大小。
Range	Range 头域可以请求实体的一个或者多个子范围，例如，表示头 500 个字节：bytes=0-499,表示第二个 500 字节：bytes=500-999,表示最后 500 个字节：bytes=-500，表示 500 字节以后的范围：bytes=500-，第一个和最后一个字节：bytes=0-0,-1，同时指定几个范围：bytes=500-600,601-999，但是服务器可以忽略此请求头，如果无条件 GET 包含 Range 请求头，响应会以状态码 206（PartialContent）返回而不是以 200（OK）。
UA-Pixels,UA-Color,UA-OS,UA-CPU	由某些版本的 IE 浏览器所发送的非标准的请求头，表示屏幕大小、屏幕深度、操作系统和 CPU 类型。

服务器响应消息

HTTP 响应也由四个部分组成，分别是：**状态行**（版本+状态+短语）、**消息报头**、**空行**和**响应正文**。



状态行：由 HTTP 协议版本号、状态码、状态消息三部分组成。

消息报头：用来说明客户端要使用的一些附加信息。

空行：消息报头后面的空行是必须的。

响应正文：服务器返回给客户端的文本信息。

下面实例是一些典型的使用 GET 来传递数据的实例：

客户端请求：

```
GET /hello.txt HTTP/1.1
User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
Host: www.example.com
Accept-Language: en, mi
```

服务端响应：

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 51
Vary: Accept-Encoding
Content-Type: text/plain
```

输出结果：

```
Hello World! My payload includes a trailing CRLF.
```

HTTP 请求方法

根据 HTTP 标准，HTTP 请求可以使用多种请求方法。

HTTP 1.0 定义了三种请求方法：**GET**、**POST** 和 **HEAD** 方法。

HTTP 1.1 新增了六种请求方法：**OPTIONS**、**PUT**、**PATCH**、**DELETE**、**TRACE** 和 **CONNECT** 方法。

方法	描述
GET（获取资源）	请求指定的页面信息，并返回实体主体。
HEAD（获取报文首部）	类似于 GET 请求，只不过返回的响应中没有具体的内容，用于获取报头。
POST（传输实体文本）	向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST 请求可能会导致新的资源的建立或已有资源的修改。
PUT（传输文件）	从客户端向服务器传送的数据取代指定的文档的内容。
DELETE（删除文件）	请求服务器删除指定的页面。
CONNECT（要求用隧道协议连接代理）	HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。要求在与代理服务器通信时建立隧道，实现用隧道协议进行 TCP 通信。主要使用 SSL (安全套接层) 和 TSL（ 传输层安全 ）协议把通信内容加密后经网络隧道传输。
OPTIONS（询问支持的方法）	允许客户端查看服务器的性能。用于查询针对请求 URL 指定资源支持的方法（客户端询问服务器可以提交哪些请求方法）。
TRACE（追踪路径）	回显服务器收到的请求，主要用于测试或诊断。
PATCH（补充）	是对 PUT 方法的补充，用来对已知资源进行局部更新。

GET 方法用来请求已被 URL 识别的资源。指定的资源经服务器端解析后返回响应内容（也就是说，如果请求的资源是文本，那就保持原样返回；如果是 CGI [通用网关接口] 那样的程序，则返回经过执行后的输出结果）。

POST 方法用来传输实体的主体。虽然用 GET 方法也可以传输实体的主体，但一般不用 GET 方法进行传输，而是用 POST 方法；虽然 GET 方法和 POST 方法很相似，但是 POST 的主要目的并不是获取相应的主体内容。

GET 方法和 POST 方法的区别

1. GET 方法用于信息获取，它是安全的（安全：指非修改信息，如数据库方面的信息），而 POST 方法是用于修改服务器上资源的请求；
2. GET 请求的数据会附在 URL 之后，而 POST 方法提交的数据则放置在 HTTP 报文实体的主体里，所以 POST 方法的安全性比 GET 方法要高；
3. GET 方法传输的数据量一般限制在 2 KB，其原因在于：GET 是通过 URL 提交数据，而 URL 本身对于数据没有限制，但是不同的浏览器对于 URL 是有限制的，比如 IE 浏览器对于 URL 的限制为 2 KB，而 chrome 、 FireFox 浏览器理论上对于 URL 是没有限制的，它真正的限制取决于操作系统本身；POST 方法对于数据大小是无限制的，真正影响到数据大小的是服务器处理程序的能力。

HEAD 方法与 GET 方法的区别：GET 方法有实体，HEAD 方法无实体。

HEAD 方法的主要用途：

1. 判断类型；
2. 查看响应中的状态码，看对象是否存在（响应：请求执行成功了，但是无数据返回）；
3. 测试资源是否被修改过。

HTTP 响应头信息

HTTP 响应头提供了关于请求、响应或者其他的发送实体的信息。

应答头	说明
Allow	服务器支持哪些请求方法（如 GET、POST 等）
Content-Encoding	文档的编码（Encode）方法。只有在解码之后才可以得到 Content-Type 头指定的内容类型。利用 gzip 压缩文档能够显著地减少 HTML 文档的下载时间。Java 的 GZIPOutputStream 可以很方便地进行 gzip 压缩，但只有 Unix 上的 Netscape 和 Windows 上的 IE 4、IE 5 才支持它。因此，Servlet 应该通过查看 Accept-Encoding 头（即 request.getHeader("Accept-Encoding")）检查浏览器是否支持 gzip，为支持 gzip 的浏览器返回经 gzip 压缩的 HTML 页面，为其他浏览器返回普通页面。
Content-Length	表示内容长度。只有当浏览器使用持久 HTTP 连接时才需要这个数据。 如果你想要利用持久连接的优势，可以把输出文档写入 ByteArrayOutputStream，完成后查看其大小，然后把该值放入 Content-Length 头，最后通过 byteArrayStream.write(response.getOutputStream))发送内容。
Content-Type	表示后面的文档属于什么 MIME 类型。Servlet 默认为 text/plain，但通常需要显式地指定为 text/html。由于经常要设置 Content-Type，因此 HttpServletResponse 提供了一个专用的方法 setContentType。
Date	当前的 GMT 时间。你可以用 setDateHeader 来设置这个头以避免转换时间格式的麻烦。
Expires	应该在什么时候认为文档已经过期，从而不再缓存它。
Last-Modified	文档的最后改动时间。客户可以通过 If - Modified - Since 请求头提供一个日期，该请求将被视为一个条件 GET，只有改动时间迟于指定时间的文档才会返回，否则返回一个 304（Not Modified）状态。Last - Modified 也可用 setDateHeader 方法来设置。
Location	表示客户应当到哪里去提取文档。Location 通常不是直接设置的，而是通过 HttpServletResponse 的 sendRedirect 方法，该方法同时设置状态代码为 302。
Refresh	表示浏览器应该在多少时间之后刷新文档，以秒计。除了刷新当前文档之外，你还可以通过 setHeader("Refresh","5;URL = http://host/path") 让浏览器读取指定的页面。注意这种功能通常是通过设置 HTML 页面 HEAD 区的 < META HTTP-EQUIV="Refresh" CONTENT="5;URL = http://host/path"> 实现，这是因为，自动刷新或重定向对于那些不能使用 CGI 或 Servlet 的 HTML 编写者十分重要。但是，对于 Servlet 来说，直接设置 Refresh 头更加方便。注意 Refresh 的意义是 “N 秒之后刷新本页面或访问指定页面”，而不是“每隔 N 秒刷新本页面或访问指定页面”。因此，连续刷新要求每次都发送一个 Refresh 头，而发送 204 状态代码则可以阻止浏览器继续刷新，不管是使用 Refresh 头还是 < META HTTP-EQUIV="Refrsh"... > 。注意 Refrsh 头不属于 HTTP 1.1 正式规范的一部分，而是一个扩展，但 Netscape 和 IE 都支持它。
Server	服务器名字。Servlet 一般不设置这个值，而是由 Web 服务器自己设置。
Set-Cookie	设置和页面关联的 Cookie。Servlet 不应使用 response.setHeader("Set-Cookie",...)，而是应使用 HttpServletResponse 提供的专用方法 addCookie。
WWW-Authenticate	客户应该在 Authorization 头中提供什么类型的授权信息？在包含 401 （Unauthorized）状态行的应答中这个头是必须的。例如，response.setHeader("WWW-Authenticate","BASIC realm="executives")。注意 Servlet 一般不进行这方面的处理，而是让 Web 服务器的专门机制来控制受密码保护页面的访问（例如 htaccess）。

HTTP 状态码

当浏览器访问一个网页时，浏览者的浏览器会向网页所在服务器发出请求。当浏览器接收并显示网页前，此网页所在的服务器会返回一个包含 HTTP 状态码的信息头（server header）用以响应浏览器的请求。

下面是常见的 HTTP 状态码：

- 200 - 请求成功
- 301 - 资源（网页等）被永久转义到其他 URL
- 404 - 请求的资源（网页等）不存在
- 500 - 内部服务器错误

HTTP 状态码分类

HTTP 状态码由三个十进制数字组成，第一个十进制数字定义了状态码的类型，后两个数字没有分类的作用。HTTP 状态码共分为 5 种类型：

HTTP 状态码分类：

分类	分类描述
1xx	信息，服务器收到请求，需要请求者继续执行操作
2xx	成功，操作被成功接收并处理
3xx	重定向，需要进一步的操作以完成请求
4xx	客户端错误，请求包含语法错误或无法完成请求
5xx	服务器错误，服务器在处理请求的过程中发生了错误

HTTP content-type

Content-Type (内容类型)，一般是指网页中存在的 Content-Type，用来定义网络文件的类型和网页的编码，决定浏览器将以什么形式、什么编码读取这个文件，这就是经常看到一些 PHP 网页点击的结果却是下载一个文件或一张图片的原因。

Content-Type 标头告诉客户端实际返回的内容的内容类型。

语法格式：

```
Content-Type: text/html; charset=utf-8
Content-Type: multipart/form-data; boundary=something
```

Session 与 Cookie

cookie 是什么

在网站中，http 请求是无状态的。也就是说即使第一次和服务器连接后并且登陆成功后，第二次请求服务器依然不能知道当前请求是哪个用户。cookie 的出现就是为了解决这个问题，第一次登陆后服务器返回一个数据（cookie）给浏览器，然后浏览器保存在本地，当该用户发送第二次请求的时候，就会自动的把上次请求存储的 cookie 数据自动的携带给服务器，服务器通过浏览器携带的数据就能判断当前用户是哪个了。cookie 存储的数据量有限，不同的浏览器有不同的存储大小，但一般不超过 4KB。因此使用 cookie 只能存储一些小量的数据。

session 是什么

session 和 cookie 的作用有点类似，都是为了存储用户相关的信息。不同的是，cookie 是存储在本地浏览器，而 session 存储在服务器。存储在服务器的数据会更加的安全，不容易被窃取。但存储在服务器也有一定的弊端，就是会占用服务器的资源。

session 的目的：弥补 HTTP 无状态特性，服务器可以利用 session 存储客户端在同一个会话期间的一些操作记录。

cookie 和 session 结合使用

在如今的市场或者企业里，一般有两种存储方式：

1. 存储在服务端：通过 cookie 存储一个 session_id，然后具体的数据则是保存在 session 中。如果用户已经登陆，则服务器会在 cookie 中保存一个 session_id，下次再次请求的时候，会把该 session_id 携带上来，服务器根据 session_id 在 session 库中获取用户的 session 数据，就能知道该用户到底是谁，以及之前保存的一些状态信息。这种专业术语叫做 server side session。
2. 将 session 数据加密，然后存储在 cookie 中。这种专业术语叫做 client side session。

session 的实现机制

1. 服务器如何判断客户端发送过来的请求术语同一个会话?

   用 session id 区分；session id 相同即认为是同一个会话。

   在 tomcat 中 session id 中用 JSESSIONID 来表示。

2. 服务器、客户端如何获取 sessionID ? SessionID 在期间是如何传输的?

服务器第一次接收到请求时，开辟了一块 Session 空间（创建了 Session 对象），同时生成一个 Session id，并通过响应头的 Set-Cookie："JSESSIONID=XXXXXXX" 命令，向客户端发送要求设置 cookie 的响应；客户端收到响应后，在本机客户端设置了一个 JSESSIONID=XXXXXXX 的 cookie 信息，该 cookie 的过期时间为浏览器会话结束。

接下来客户端每次向同一个网站发送请求时，请求头都会带上该 cookie 信息（包含 Session id）；然后，服务器通过读取请求头中的 Cookie 信息，获取名称为 JSESSIONID 的值，得到此次请求的 Session id。

注意：服务器只会在客户端第一次请求响应的时候，在响应头上添加 Set-Cookie : " JSESSIONID=XXXXXXX " 信息，接下来在同一个会话的第二第三次响应头里，是不会添加 Set-Cookie : " JSESSIONID=XXXXXXX " 信息的；而客户端是会在每次请求头的 cookie 中带上 JSESSION 信息。

cookie 与 session 的区别

1. cookie 以文本文件格式存储在浏览器中，而 session 存储在服务端。
2. cookie 的存储限制了数据量，只允许 4kb，而 session 是无限量的。
3. 可以轻松访问 cookie 值，但无法轻松访问 session，因为 session 更安全。
4. 设置 cookie 时间可以使 cookie 过期，但是使用 session-destory()，将会销毁会话。

如果需要经常登陆一个站点，最好用 cookie 来保存信息，不然每次登陆会特别麻烦，但是对于需要安全性高的站点以及控制数据的能力时需要用 session 效果更佳。

* HTTPS

HTTP 的特点

1. 无状态：协议对客户端没有状态存储，对事物处理没有“记忆”能力，比如访问一个网站需要反复进行登录操作。
2. 无连接：HTTP/1.1 之前，由于无状态特点，每次请求需要通过 TCP 三次握手四次挥手和服务器重新建立连接。比如某个客户机在短时间多次请求同一个资源，服务器并不能区别是否已经响应过用户的请求，所以每次需要重新响应请求，需要耗费不必要的时间和流量。
3. 基于请求和响应：基本的特性，由客户端发起请求，服务端响应。
4. 简单快速、灵活。
5. 通信使用明文，请求和响应不会对通信方进行确认、无法保护数据的完整性。

针对无状态的一些解决策略：

场景：逛电商商场用户需要使用的时间比较长，需要对用户一段时间的 HTTP 通信状态进行保存，比如执行一次登录操作，在 30 分钟内所有的请求都不需要再次登录。

1. 通过 Cookie/Session 技术。
2. HTTP/1.1 持久连接（HTTP keep-alive）方法，只要任意一端没有明确提出断开连接，则保持 TCP 连接状态，在请求首部字段中的 Connection:keep-alive 即为表明使用了持久连接。

HTTPS

https 协议（Hypertext Transfer Protocol Secure）：超文本传输安全协议。缩写为：HTTPS，常称为 HTTP over TLS，HTTP over SSL 或 HTTP source，是一种通过计算机网络进行安全通信的传输协议。这个协议由网景公司（Netscapt）在 1994 年首次题出，随后扩展到互联网上，随后扩展到互联网上。

在计算机网络上，HTTPS 经由 HTTP 进行通信，但利用 SSL/TLS 来加密数据包。HTTPS 开发的主要目的，是提供对网站服务器的身份认证，保护交换数据的隐私与完整性。

HTTP 协议传输的数据都是未加密的，也就是明文，因此使用 HTTP 协议传输隐私信息非常不安全。HTTP 使用 80 端口通讯，而 HTTPS 占用 443 端口通讯。

简单来说，HTTPS 是 HTTP 的安全版，是使用 SSL/TLS 加密的 HTTP 协议。通过 TSL/SSL 协议的身份验证、信息加密和完整性校验的功能，从而避免信息窃听、信息篡改和信息劫持的风险。

HTTPS 提供了加密（Encryption）、认证（Verification）、鉴定（Identification）三种功能。

HTTPS 就是在应用层和传输层中间加了一道验证的门槛以保证数据安全。

HTTPS 特点

HTTPS 协议基于 HTTP 协议，通过 SSL 或 TLS 提供加密处理数据、验证双方身份以及数据完整性保护。

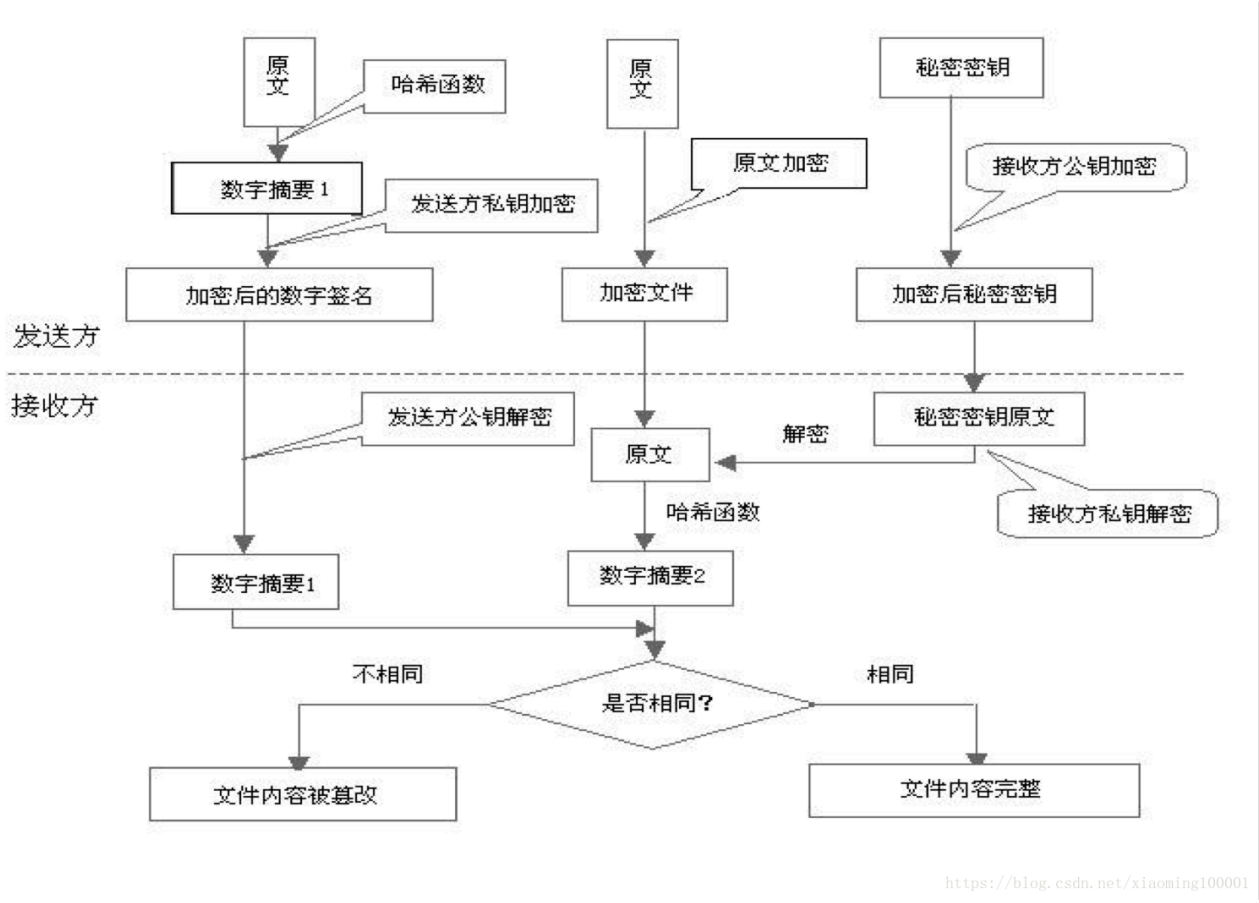
HTTPS 不是明文传递，而且 HTTPS 有如下特点：

1. 私密性（Confidentiality/Privacy）：内容加密，也就是提供信息加密，保证数据传输的安全。采用混合加密技术，中间者无法直接查看明文内容。
2. 可信性（Authentication）：身份验证，主要是服务器端的，确认网站的真实性，有些银行也会对客户端进行认证。通过证书认证客户端访问的是自己的服务器。
3. 完整性（Message Integrity）：保证信息传输过程中的完整性，防止传输的内容被中间人冒充或者篡改。
    - 收方能够证实发送方的真实身份；
    - 发送方事后不能否认所发送过的报文；
    - 收方或非法者不能伪造、篡改报文。

混合加密：结合非对称加密和对称加密技术。客户端使用对称加密生成密钥对传输数据进行加密，然后使用非对称加密的公钥再对密钥进行加密，所以网络上传输的数据是被密钥加密的密文和用公钥加密后的秘密密钥，因此即使被黑客截取，由于没有私钥，无法获取到加密明文的密钥，便无法获取到明文数据。

数字摘要：通过单向 hash 函数对原文进行哈希，将需加密的明文 “ 摘要 ” 成一串固定长度（如 128bit）的密文，不同的明文摘要成的密文其结果总是不相同，同样的明文其摘要必定一致，并且即使知道了摘要也不能反推出明文。

数字签名技术：数字签名建立在公钥加密机制基础上，是公钥加密技术的另一类应用。它把公钥加密技术和数字摘要结合起来，形成了实用的数字签名技术。



非对称加密过程需要用到公钥进行加密，而公钥被包含在数字证书中，数字证书通常来说是由受信任的数字证书颁发机构 CA，在验证服务器身份后颁发，证书中包含了一个密钥对（公钥和私钥）和所有者的识别信息。数字证书被放到服务端，具有服务端身份验证和数据传输加密功能。

SSL/TLS 协议

SSL（Secure Socket Layer）安全套接层。

TLS（Transport Layer Security）传输层安全。

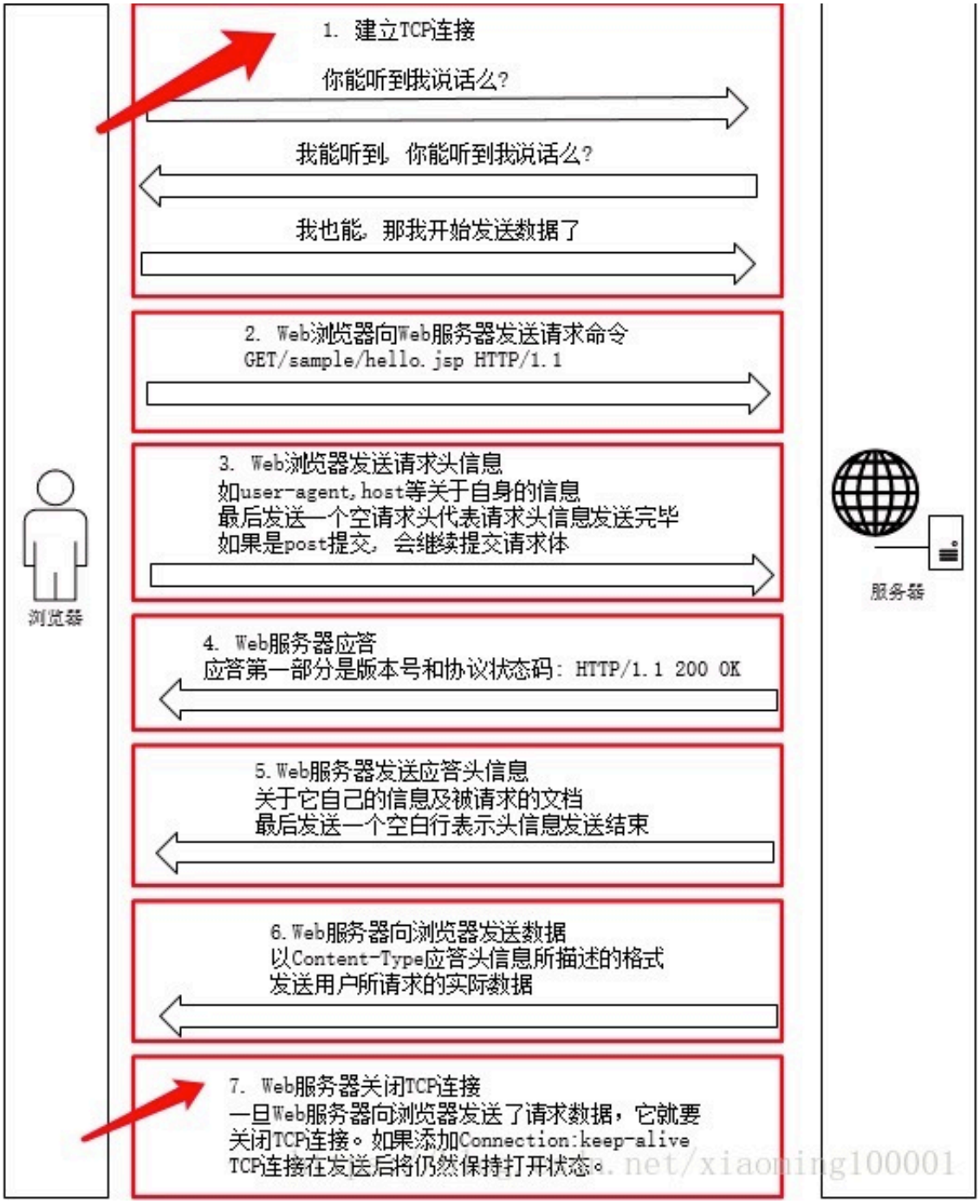
TLS 是传输层加密协议，前身是 SSL 协议，由网景公司 1995 年发布，有时候两者不区分。

SSL 及其继任者 TLS 是为网络通信提供安全及数据完整性的一种安全协议。TLS 与 SSL 在传输层对网络连接进行加密。

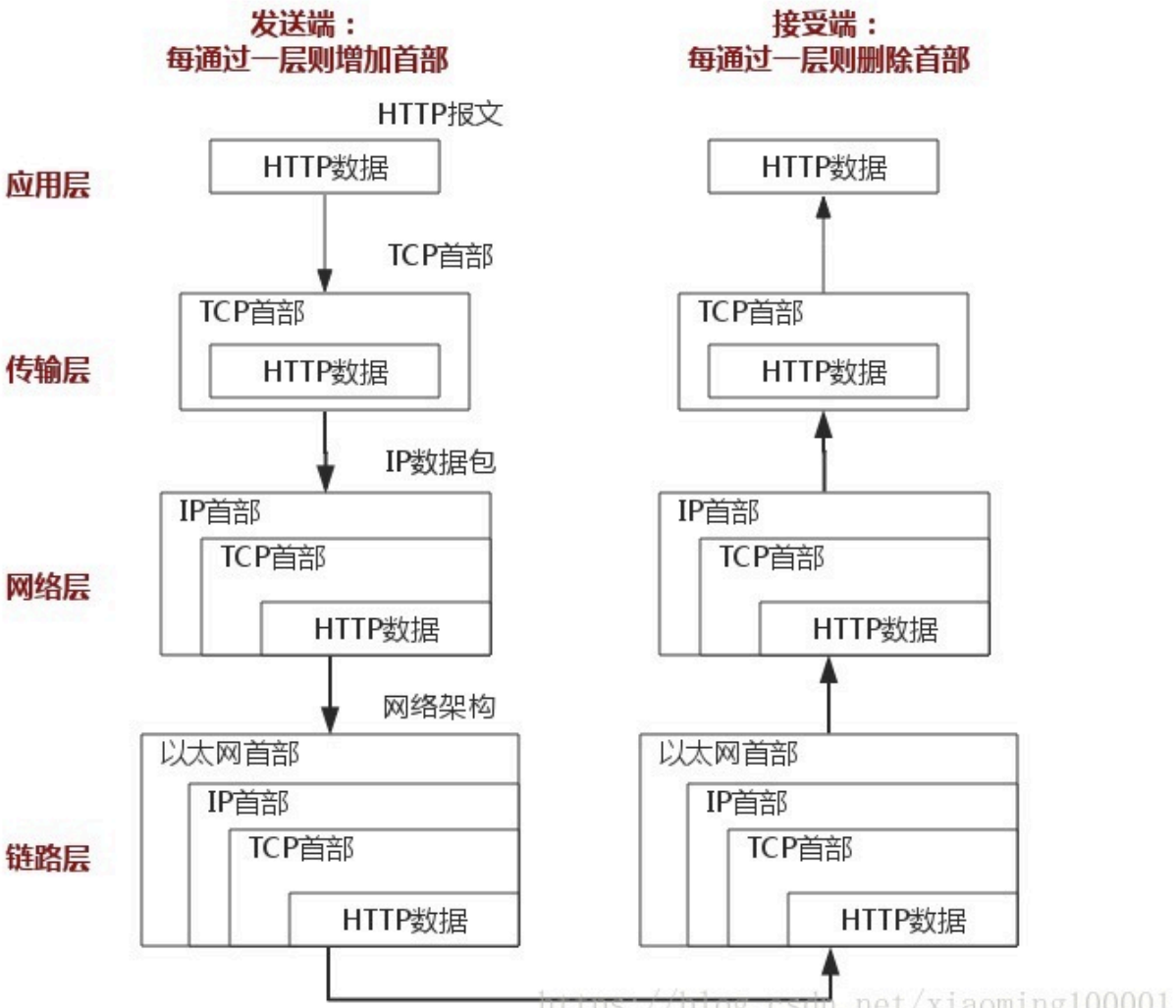
SSL 协议主要服务

1. 认证用户和服务器，确保数据发送到正确的客户机和服务器。
2. 加密数据以防止数据中途被窃取。
3. 维护数据的完整性，确保数据在传输过程中不被改变。

HTTP 通信传输

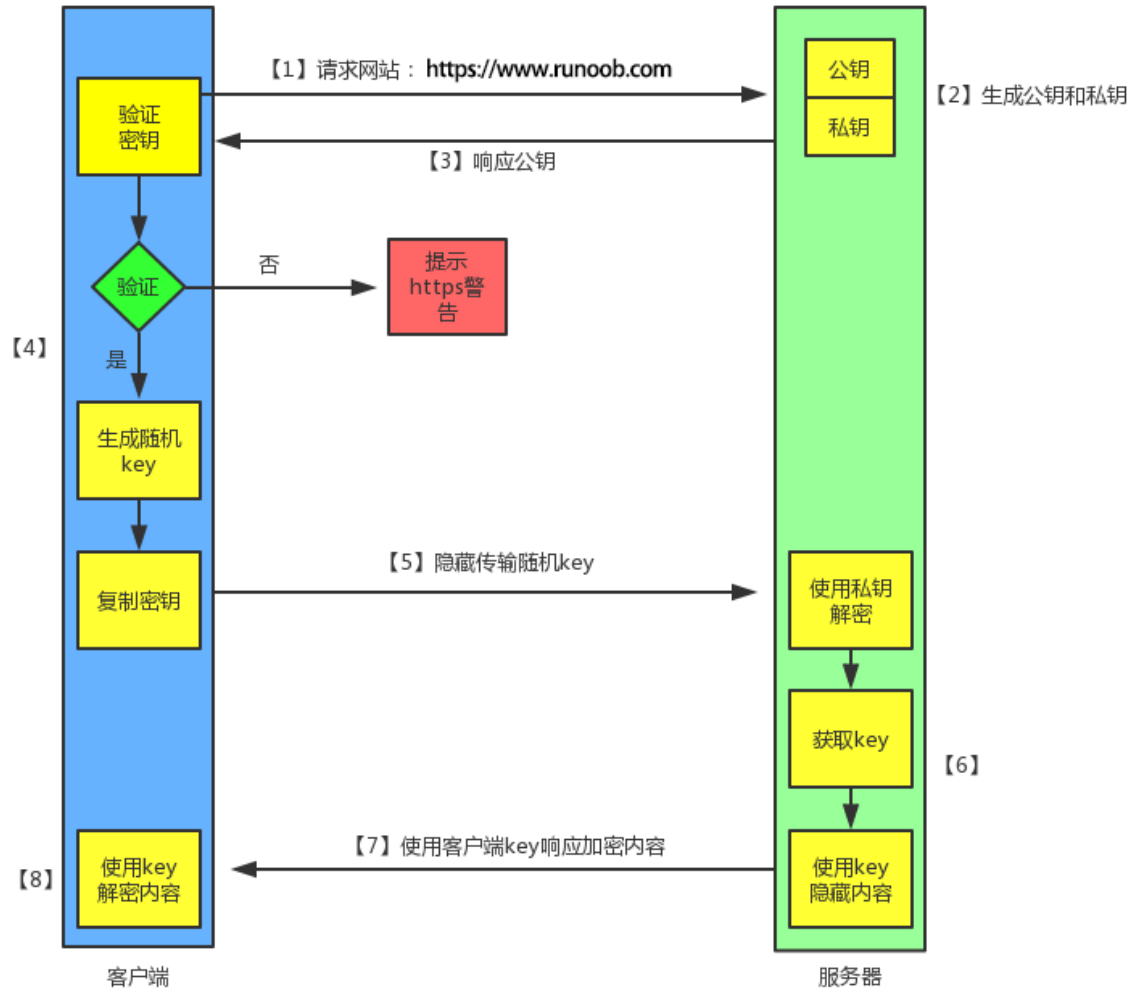


客户端输入 URL 回车，DNS 解析域名得到服务器的 IP 地址，服务器在 80 端口监听客户端请求，端口通过 TCP/IP 协议（可以通过 Sokcet 实现）建立连接。HTTP 属于 TCP/IP 模型中的应用层协议，所以通信的过程其实是对应数据的入栈和出栈。



HTTPS 通信传输

HTTPS 能够加密信息，以免敏感信息被第三方获取，所以很多银行网站或电子邮箱等等安全级别较高的服务都会采用 HTTPS 协议。



1. 客户端发送 HTTPS 请求

   就是用户在浏览器里输入一个 https 网址，然后连接到 server 的 443 端口。

2. 服务端的配置

   采用 HTTPS 协议的服务器必须要有一套数字证书，可以自己制作，也可以向组织申请，区别就是自己颁发的证书需要客户端验证通过，才可以继续访问，而使用受信任的公司申请的证书申请的证书则不会弹出提示页面。

   这套证书其实就是一对公钥和私钥。

3. 传送证书

   这个证书其实就是公钥，只是包含了很多信息，如证书的颁发机构、过期时间等等。

4. 客户端解析证书

   这部分工作是由客户端的 TLS 来完成的，首先会验证公钥是否有效，比如颁发机构、过期时间等等，如果发现异常，则会弹出一个警告框，提示证书存在问题。

   如果证书没有问题，那么就生成一个随机值，然后用证书对该随机值进行加密。

5. 传送加密信息

   这部分传送的是用证书加密后的随机值，目的就是让服务端得到这个随机值，以后客户端和服务端的通信就可以通过这个随机值进行加密解密了。

6. 服务端解密信息

   服务端用私钥解密后，得到了客户端传过来的随机值（私用），然后把内容通过该值进行对称加密，所谓对称加密就是，将信息和私钥通过某种算法混合在一起，这样除非知道私钥，不然无法获取内容，而正好客户端和服务端都知道这个私钥，所以只要加密算法够彪悍，私钥够复杂，数据就够安全。

7. 传输加密后的信息

   这部分信息是服务端用私钥加密后的信息，可以在客户端被还原。

8. 客户端解密信息

   客户端用之前生成的私钥解密服务端传过来的信息，于是获取了解密后的内容，整个过程第三方即使监听了数据，也束手无策。

基本的运行过程

HTTPS 默认工作在 TCP 协议 443 端口，它的工作流程一般如以下方式：

1. TCP 三次同步握手
2. 客户端验证服务器数字证书
3. DH 算法协商对称加密算法的密钥、hash 算法的密钥
4. SSL 安全加密隧道协商完成
5. 网页以加密的方式传输，用协商的对称加密算法和密钥加密，保证数据机密性；用协商的 hash 算法进行数据完整性保护，保证数据不被篡改。

SSL/TLS 协议的基本思路是采用公钥加密法，也就是说，客户端先向服务器索要公钥，然后用公钥加密信息，服务器收到密文件后，用自己的私钥解密。

1. 如何保证公钥不被篡改?

   解决方法：将公钥放在数字证书中。只要证书是可信的，公钥就是可信的。

2. 公钥加密计算量太大，如何减少耗用的时间?

   解决方法：每一次对话（seesion），客户端和服务器端都生成一个 “对话密钥”（session key），用它来加密信息。由于 “ 对话密钥 ” 是对称加密，所以运算速度非常快，而服务器公钥只用于加密 “ 对话密钥 ” 本身，这样就减少了加密运算的消耗时间。

SSL/TSL 协议的基本过程

SSL/TLS 协议的基本过程是这样的：

1. 客户端向服务器端索要并验证公钥。
2. 双方协商生成 “ 对话密钥 ”。
3. 双方采用 “ 对话密钥 ” 进行加密通信。

前两步，又称为 “ 握手阶段 ”（handshake）。

SSL、TLS 的握手过程

SSL 协议在握手阶段使用的是非对称加密，在传输节点使用的是对称加密，也就是说在 SSL 上传送的数据是使用对称密钥加密的。

因为非对称加密的速度缓慢，浪费资源。其实当客户端和主机使用非对称加密方式建立连接后，客户端和主机已经决定好了在传输过程使用的对称加密算法和关键的对称加密密钥，由于这个过程本身是安全可靠的，也即对称加密密钥是不可能被窃取盗用的，因此，保证了在传输过程中对数据进行对称加密也是安全可靠的，因为除了客户端和自己之外，不可能有第三方窃听并解密出对称加密密钥。如果有人窃听通信，它可以知道双方选择的加密方法，以及三个随机数中的两个。整个通话的安全，只取决于第三个随机数（pre-master secret）能不能被破解。

第三个随机数 c，又称 “ pre-master key ”。有了它以后，客户端和服务器就同时有了三个随机数，接着双方就用实现商定的加密方法，各自生成本次会话所用的同一把 " session key（会话密钥）"。

为什么一定要用三个随机数，来生成 “ 会话密钥 ”，SSL 协议不信任每个主机都能产生完全随机的随机数，如果随机数不随机，那么会话密钥就有可能被猜出来，一个伪随机可能完全不随机，可是三个伪随机就十分接近随机了，每增加一个自由度，随机性增加的可不是一。

客户端收到服务器第一次的回应以后，首先验证服务器证书。如果证书不是可信机构颁布、或者证书中的域名与实际域名不一致、或者证书已经过期，就会向访问者显示一个警告，由其选择是否还要继续通信。

服务器收到客户端的三个随机数之后，计算生成本次会话所用的 “ 会话密钥 ”。然后，向客户端最后发送下面信息：

1. 编码改变通知，表示随后的信息都将用双方商定的加密方法和密钥发送。
2. 服务器握手结束通知，表示服务器的握手阶段已经结束，这一项同时也是前面发送的所有内容的 hash 值，用于供客户端校验。

至此，整个握手阶段全部结束。接下来，客户端与服务器进入加密通信，就完全是使用普通的 HTTP 协议，只不过用 “ 会话密钥 ” 加密内容。

HTTPS 在传输数据之前需要客户端（浏览器）与服务端（网站）之前进行一次握手，在握手过程中将确立双方加密传输数据的密码信息。TLS/SSL 中使用了非对称加密、对称加密以及 HASH 算法。

数字证书

数字证书内容包括了加密后服务器的公钥、权威机构的信息、服务器域名，还有经过 CA 私钥签名之后的证书内容（经过先通过 Hash 函数计算得到证书数字摘要，然后用权威机构私钥加密数字摘要得到数字签名）、签名计算方法以及证书对应的域名。

验证证书安全性过程

1. 当客户端收到这个证书之后，使用本地配置的权威机构的公钥对证书进行解密得到服务器的公钥和证书的数字签名，数字签名经过 CA 公钥解密得到证书信息摘要。
2. 然后证书签名的方法计算一下当前证书的信息摘要，与收到的信息摘要作对比，如果一样，表示证书一定是服务器下发的，没有被中间人篡改过，因为中间人虽然有权威机构的公钥，能够解析证书内容并篡改，但是篡改完成之后中间人需要将证书重新加密，但是中间人没有权威机构的私钥，无法加密，强行加密只会导致客户端无法解密，如果中间人强行乱修改证书，就会导致证书内容和证书签名不匹配。

那第三方攻击者能否让自己的证书显示出来的信息也是服务端呢？（伪装服务端一样的配置）显然这个是不行的，因为当第三方攻击者去 CA 那边寻求认证的时候 CA 会要求其提供域名的 whois 信息、域名管理邮箱等证明是服务端域名的拥有者，而第三方攻击者是无法提供这些信息所以就无法骗 CA 拥有属于服务端的域名。

对客户端的验证

对于非常重要的保密数据，服务端还需要对客户端进行验证，以保证数据传送给了安全的合法的客户端。服务端可以向客户端发出 Cerficate Request 消息，要求客户端发送证书给客户端的合法性进行验证。比如，金融机构往往只允许认证客户连入自己的网络，就会向正式客户提供 USB 密钥，里面就包含了一张客户端证书。

Session 的恢复

有两个方法可以恢复原来的 session：一种叫做 session ID，另一种叫做 session ticket。

session ID

session ID 的思想很简单，就是每一次对话都有一个编号（session ID）。如果对话中断，下次重连的时候，只要客户端给出这个编号，且服务器有这个编号的记录，双方就可以重新使用已有的 “ 对话密钥 ”，而不必重新生成一把。

session ID 是目前所有浏览器都支持的方法，但是它的缺点在于 session ID 往往只保留在一台服务器上。所以，如果客户端的请求发到另一台服务器，就无法恢复对话。

session ticket

客户端发送一个服务器在上一次对话中发送过来的 session ticket。这个 session ticket 是加密的，只有服务器才能解密，其中包括本次对话的主要信息，比如对话密钥和加密方式。当服务器收到 session ticket 以后，解密后就不必重新生成对话密钥了。目前只有 Firefox 和 Chrome 浏览器支持。

HTTPS 协议和 HTTP 协议的区别

1. 使用 HTTPS 协议需要到 ca （Certificate Authority，数字证书认证机构）申请证书，一般免费证书很少，因此需要一定费用。证书颁发机构如：Symantec、Comodo、GoDaddy 和 GlobalSign 等。
2. http 是超文本传输协议，信息是明文传输，数据都是未加密的，安全性较差，连接很简单，是无状态的；https 协议是由 SSL+HTTP 协议构建的具有安全性的 ssl 加密、身份认证的传输协议，数据传输过程是加密的，安全性较好。
3. http 和 https 使用的是完全不同的连接方式，用的端口也不一样，前者是 80，后者是 443。
4. HTTP 页面响应速度比 HTTPS 快，主要是因为 HTTP 使用 TCP 三次握手建立连接，客户端和服务器需要交换 3 个包，而 HTTPS 除了 TCP 的三个包，还要加上 ssl 握手需要的 9 个包，所以一共是 12 个包。
5. HTTPS 其实就是建构在 SSL/TLS 之上的 HTTP 协议，所以，HTTPS 要比 HTTP 更耗费服务器资源。

* IP 协议

IP 是 Internet Protocol（网络互联协议）的缩写，是 TCP/IP 体系中的网络层协议。设计 IP 的目的是提高网络的可扩展性：一是解决互联网问题，实现大规模、异构网络的互联互通；二是分割顶层网络应用和底层网络技术之间的耦合关系，以利于两者的独立发展。根据端对端的设计原则，IP 只为主机提供一种无连接、不可靠的、尽力而为的数据报传输服务。

* IPv4 和 IPv6

IP 地址有两个标准：IP 版本 4 （IPv4）和 IP 版本 6（IPv6）。所有有 IP 地址的计算机都有 IPv4 地址，许多计算机也开始使用新的 IPv6 地址系统。

电脑如何得到它的 IP 地址

IP 地址可以是动态的，也可以是静态的。静态地址是通过编辑计算机的网络设置来配置自己的地址。这种类型的地址很少见，如果在不了解 TCP/IP 的情况下使用它，可能会造成网络问题。动态地址是最常见的。它们由运行在网络上的服务动态主机配置协议（DHCP）分配。DHCP 通常运行在网络硬件上，如路由器或专用的 DHCP 服务器。

动态 IP 地址是使用租赁系统发出的，这意味着 IP 地址只在有限的时间内有效。如果租约到期，计算机将自动请求新的租约。有时，这意味着计算机也将获得一个新的 IP 地址，特别是在计算机在租约之间断开网络连接的情况下。这个过程通常对用户是透明的，除非计算机警告网络上的 IP 地址冲突（两台具有相同 IP 地址的计算机）。地址冲突很少见，而今天的技术通常会自动修复问题。

常见的网络攻击类型

网络攻击的方式主要分为四类：

1. 第一类是服务拒绝攻击，包括死亡之 ping（ping of death）、泪滴（teardrop）、UDP 洪水（UDP flood）、SYN 洪水（SYN flood）、Land 攻击、Smurf 攻击、Fraggle 攻击、电子邮件炸弹、畸形消息攻击等。
2. 第二类是利用型攻击，包括口令猜测、特洛伊木马、缓冲区溢出。
3. 第三类是信息收集型攻击，包括地址扫描、端口扫描、慢速扫描、体系结构探测、DNS 域转换、Finger 服务、LDAP 服务等。
4. 第四类是假消息攻击，主要包括：DNS 高速缓存污染、伪造电子邮件。

DOS 攻击

攻击描述

DOS 攻击通过协议方式或抓住系统漏洞，借助代理服务器模拟多个用户不停的对网站进行访问请求，集中对目标进行网络攻击，让目标计算机或网络无法提供正常的服务或资源访问，使目标系统服务系统停止响应甚至崩溃，例如疯狂 Ping 攻击。

危害说明

服务器资源耗尽，停止响应；技术门槛较低，效果明显。

处理方法

1. 扩展访问列表是防止 DOS 攻击的有效工具，例如 Show IP access-list。
2. 让路由器具备 TCP 拦截功能，在对方发送数据流时可以很好的监控和拦截。
3. 防止 DOS 攻击的根本是利用设备规则来合理的屏蔽持续的、高频度的数据冲击。
4. 对用户操作进行记录，高于一定频率则禁封访问 IP。

ARP 攻击

攻击描述

通过伪造 IP 地址和 MAC 地址实现 ARP 欺骗，能够在网络中产生大量的 ARP 通信量使网络阻塞，能更改目标主机 ARP 缓存中的 IP-MAC 条目，造成网络中断或中间人攻击。

危害说明

攻击者计算机不堪重负，网段中其他计算机联网时断时续（因为有时能收到真实的网关 ARP 信息）。网段所属的计算机不堪重负，其他计算机完全无法上网。

处理方法

1. 安装 ARP 防火墙：360 安全卫士（内置）、金山贝壳 ARP 专杀、金山卫士。
2. 安装专门的杀毒软件：利用局域网 ARP 欺骗检测工具来确定 ARP 攻击源，然后利用 ARP 专杀工具进行杀毒。
3. 通过 “ 网络参数 ” - “ LAN 口参数 ” 来查找路由器的 MAC 地址和 IP 地址，在局域网中的每台电脑中实现静态 ARP 绑定。

XSS 攻击

攻击描述

攻击者通过在链接中插入恶意代码，用户一旦点开链接，攻击者能够盗取用户信息。攻击者通常会用十六进制链接编码，提供可信度。网站在接收到包含恶意代码的请求之后会产生一个看似合法实则包含恶意代码的页面。

危害说明

攻击者通常会在有漏洞的程序中插入 JavaScript、VBScript、ActiveX 或 Flash 以欺骗用户。一旦得手，他们可以盗取用户名，修改用户设置，盗取/污染 cookie，做虚假广告等。

处理方法

1. 网站开发者：验证所有输入数据，检测攻击；对所有输出数据进行适当的编码。
2. 用户：在浏览器设置中关闭 JavaScript。如果使用 IE 浏览器，将安全级别设置到 “高”。

在防护时应在客户端、服务器均做防护，因为客户端很容易绕过，攻击者找到后台接口之后仍然可以进行 XSS 注入。防护时要么对 <、>、script、div 等等字符直接屏蔽，要么对其进行编码转换。

SQL 注入

攻击描述

通过把 SQL 命令插入到 Web 表单提交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的 SQL 命令，“#”、“ ’ ”、"-"、" ' or 1 = 1' " 等 sql 注入最常见。

危害说明

数据库入侵，用户信息泄露，数据表被篡改，数据库被篡改比网页文件被篡改危害大得多。

处理方式

1. 在客户端、服务器、数据库均进行 SQL 敏感字符过滤。
2. 限制 Web 应用程序所用的数据库访问账号权限。

在做防护时同样可以有直接屏蔽和 sql 转码两种方式，要么直接屏蔽掉含有 sql 敏感字符的传入并予以警告，要么对其 sql 敏感字符进行转码，用 &+ 自定义字母等字符进行替换。

域名攻击

攻击描述

通过攻击域名解析服务器（DNS）或伪造域名解析服务器（DNS）的方法，把目标网站域名解析到错误的地址，使得域名被盗或 DNS 域名劫持，主要用来阻止用户访问某些特定的网站或者是将用户引导到广告页面。

危害说明

失去域名控制权，域名会被绑定解析到黑客网站，被反解析权重会分散，引起搜索引擎、安全平台不信任从而降权标黑。

处理方式

1. 选择大型知名域名注册商，填写真实信息，锁定域名禁止转移。
2. 保证域名注册邮箱安全；
3. 选择大型稳定域名解析商：锁定解析。

这种攻击如果是选用亚马逊、阿里云、腾讯云等知名云计算服务平台的云主机一般不会出现域名攻击，因为这些公司都把这些安全措施做好了，但是如果是自己搭建云服务器那就得注意了，特别是学校、企业等。

嗅探扫描

攻击描述

网络嗅探也叫网络监听，是将网络上传输的数据未经用户许可进行捕获并进行分析的一种行为。许多网络入侵往往伴随着网络嗅探行为，许多网络攻击也都借助于网络嗅探，如著名的会话劫持。

危害说明

攻击者窃取数据包，而数据包中一般会包含很多重要的隐私信息。这些信息主要包括账号名称、口令密码信息、捕获专用的或者机密的信息、私人信息数据等。

处理方式

1. 探测网卡是否处于混杂模式；通过防火墙，实时观看目前网络带宽的分布情况。
2. 数据加密技术：对账号、口令和那些敏感数据进行加密传输，网站中使用 HTTPS 最好。
3. 使用安全拓扑结构，但开销很大。

病毒攻击

攻击描述

黑客向宿主计算机中插入病毒，病毒通过复制对系统进行破坏，计算机病毒有许多感染方式，可以通过文件（宏病毒）、硬盘和网络等。

危害说明

被攻击计算机直接被病毒侵害，系统无法正常运行甚至直接宕机。如果中了蠕虫病毒危害将会更大，同一个域的计算机或与被攻击计算机有数据交易的计算机都将可能被入侵，并且传播迅速不可控。

处理方式

1. 开启网络防火墙；
2. 关闭不常用端口，只开启平时使用的端口，减少病毒攻击的可能；
3. 定时打补丁，修复计算机漏洞。