http://www.ruanyifeng.com/blog/2014/02/ssl_tls.html

前言

作为一名程序员，不可能不与网络打交道。不夸张地说，我们的手机和电脑离开了网络就是一块「废铁」，它们的作用将大打折扣.。本文要是针对非网络专业开发的人员准备的，以「最短的时间，了解计网最多的知识」为前提而写。

概述

先来了解下各种我们知道，但是不太了解的专业名词的意思

因特网

因特网是当今世界上最大的网络，是”网络的网络”. 即因特网是所有网络互连起来的一个巨型网络.

因特网的组成 ：

- 边缘部分 ：主机
- 核心部分 ：大量网络和连接这些网络的路由器(此路由器不是我们家用的路由器)

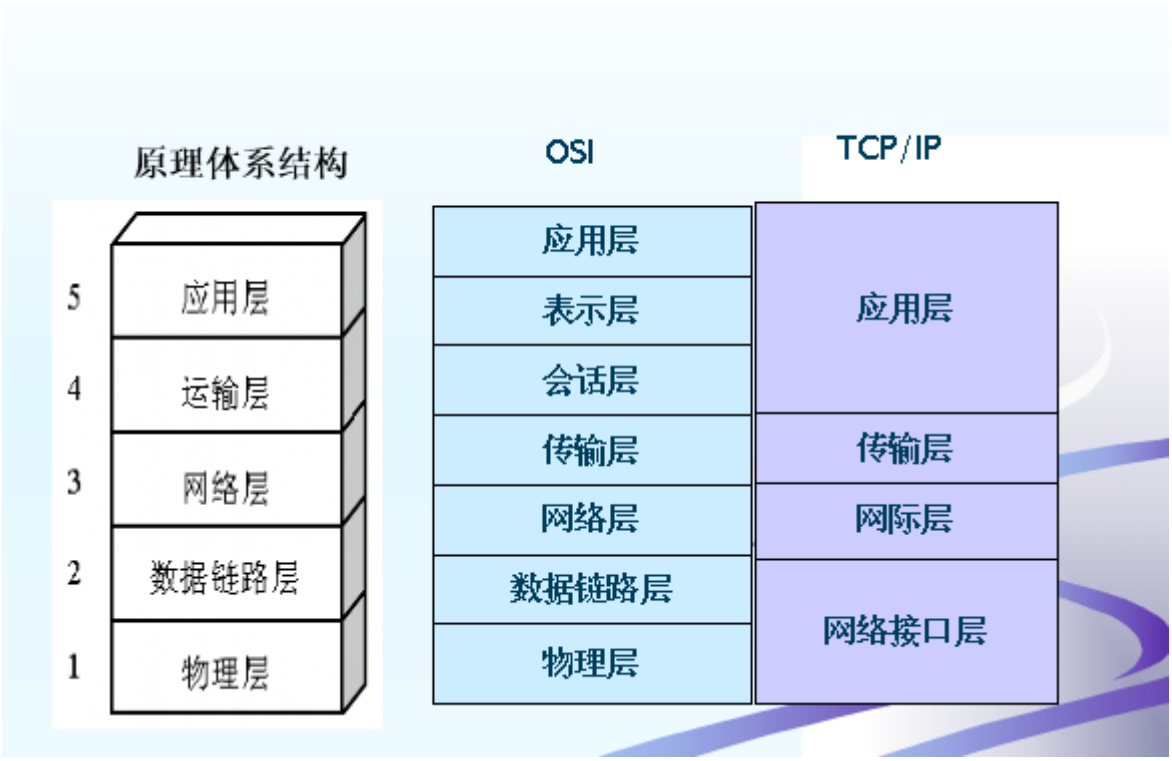
以太网

以太网是现在最常用的局域网通信协议，以太网上传输的是 MAC 帧. 由于以太网同一时间只允许一台计算机发送数据，所以必须有一套检测机制，那就是 CSMA/CD 协议 ：

1. 多点接入 ：多台计算机以多点接入的方式连接在一根总线上
2. 载波监听 ：不管是否正在发送，每个站都必须不停地检测信道
3. 碰撞检测 ：边发送边监听

OSI

开放系统互连基本参考模型，只要遵守这个 OSI 标准，任何两个系统都能进行通信. OSI 是七层协议体系结构，而 TCP/IP 是一个四层协议体系结构，于是我们采取折中的方法，学习计算机网络原理的时候往往用的是五层协议的体系结构 ：物理层，数据链路层，网络层，传输层和应用层



协议体系结构

物理层

计算机的世界里只有 0 和 1，正如你现在所看这篇文章的文字，存储在计算机中也是一大串 0 和 1 的组合. 但是这些数字不能在真实的物理介质中传输的，而需要把它转换为光信号或者电信号，所以这一层负责将这些**比特流**(0101)与光电信号进行转换.

如果没有物理层，那么也就不存在互联网，不存在数据的共享，因为数据无法在网络中流动.

数据链路层

数据在这一层不再是以比特流的形式传输，而是分割成一个一个的**帧**再进行传输.

MAC 地址

又称计算机的硬件地址，被固化在适配器(网卡)ROM 上的占 48 位的地址. MAC 地址可以用来唯一区别一台计算机，因为它在全球是独一无二的

分组交换

由于数据在这次曾要被分割成一个一个的帧，由于不同的链路规定了不同的最大帧长，即 MTU(最大传输单元)，凡是超出这个 MTU 的帧都必须被分块. 例如一台货车一次能运输 5 吨的货物，而有条公路限载重 2 吨，那么你只好分 3 次运输.

网桥

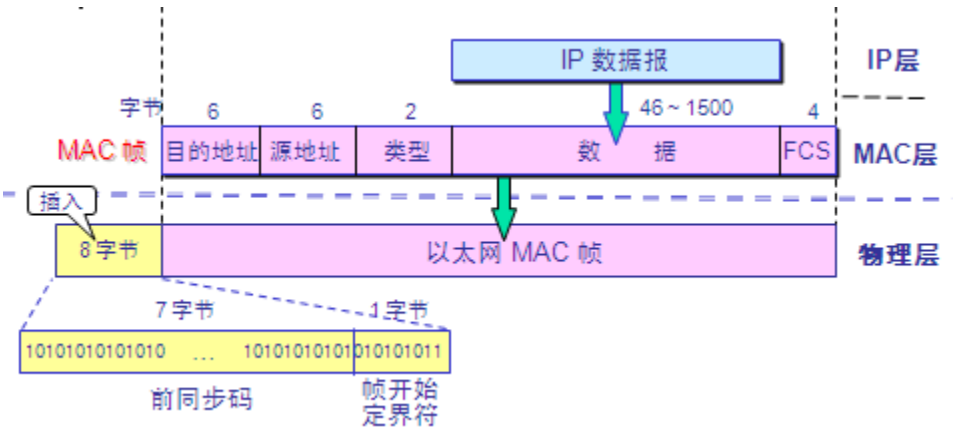
网桥工作在数据链路层，根据 MAC 帧的目的地址对收到的帧进行转发和过滤.

以太网交换机

实际上就是一个多接口的网桥，以太网交换机的每个接口都直接与一个单个主机或另一个集线器相连，可以很容易实现 VLAN(虚拟局域网)

以太网的 MAC 帧

MAC 帧的格式为 ：



MAC 帧格式

- 目的地址 ： 接收方 48 位的 MAC 地址
- 源地址 ： 发送方 48 位的 MAC 地址

- 类型字段 ：标志上一层使用的是什么协议，0x0800 为 IP 数据报

网络层

如果只有数据链路层没有网络层，数据就只能在同一条链路上传输，不能跨链路传输. 有了网络层，数据便能跨域不同的数据链路传输.

IP 地址

IP 地址又称为软件地址，存储在计算机的存储器上，IPv4 地址为 32 位，IPv6 地址为 128 位

IP 地址和 MAC 地址

- 网络层以上使用 IP 地址，数据链路层以下使用 MAC 地址
- IP 地址是逻辑地址，MAC 地址是物理地址
- IP 分组中首部的源地址和目的地址在传输中不会改变，MAC 帧中首部的源地址和目的地址每到一个路由器会改变一次

IP 地址分类

IP 地址 = {<网络号>, <主机号>}

A 类地址 ： 0.0.0.0 ~ 127.0.0.0
B 类地址 ： 128.0.0.0 ~ 191.255.0.0
C 类地址 ： 192.0.0.0 ~ 223.255.255.0

划分子网之后的 IP 地址

IP 地址 = {<网络号>, <子网号>, <主机号>}

例如某单位拥有一个 B 类 IP 地址，145.13.0.0，但凡目的地址为 145.13.x.x 的数据报都会被送到这个网络上的路由器 R. 内部划分子网后变成 ：
145.13.3.0，145.13.7.0，145.13.21.0. 但是对外仍表现为一个网络，即 145.13.0.0. 这样路由器 R 收到报文后，再根据目的地址发到对应的子网上.

子网掩码

一般由一串 1 和一串 0 组成，不管网络有没有划分子网，将子网掩码和 IP 地址做按位与运算即可得出网络地址.

所有的网络都必须使用子网掩码，同时在路由表中必须有子网掩码这一栏. 如果一个网络**不划分子网**，那么该网络的子网掩码就是默认的子网掩码.
A 类地址的默认子网掩码为 255.0.0.0
B 类地址的默认子网掩码为 255.255.0.0
C 类地址的默认子网掩码为 255.255.255.0

尽管划分子网增加了灵活性，但是却减少了能够连接在网络上的主机总数.

构成超网的 IP 地址

IP 地址 = {<网络前缀>, <主机号>}

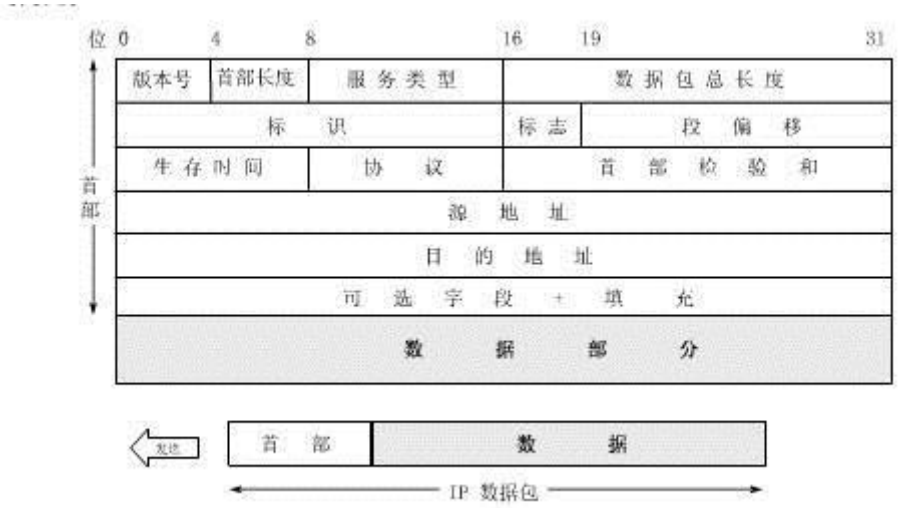
使用网络前缀，无分类域间路由选择 CIDR

例如，128.14.35.7/20，意思是前 20 位为网络前缀，后 12 位为主机号. 另外，CIDR 把网络前缀相同的连续的 IP 地址组成一个”CIDR 地址块”

地址掩码 : CIDR 使用 32 位的地址掩码，类似于子网掩码.

IP 数据报

在网络层，数据是以 IP 数据报(IP 分组)的形式传输的



IP 数据报的格式

首部前 20 字节为固定长度，是所有 IP 数据报必备的. 后 4 字节是可选字段，其长度可变.

IP 数据报首部固定的字段分析 :

- 版本号 :IP 协议的版本，IPv4 或 IPv6

- 首部长度 ：记录了首部的长度，最大为 1111，即 15 个 32 位字长，即 60 字节. 当首部长度不是 4 字节的整数倍时，需要使用最后的填充字段加以填充.
- 服务类型 ：一般无用
- 总长度 ：指首部和数据之和的长度. 最大为 2^{16}-1 = 65535 字节. 但是由于数据链路层规定每一帧的数据长度都有最大长度 MTU，以太网规定 MTU 为 1500 字节，所以超出范围的数据报就必须进行分片处理
- 标识 ：每产生一个 IP 数据报，计数器就+1，并将此值赋值给标识字段. 再以后需要分片的数据报中，标识相同说明是同一个数据报
- 标志 ：占 3 位，最低位记为 MF(More Fragment). MF = 1 说明还有分片; MF = 0 说明这已经是最后一个分片. 中间一位记为 DF(Don’t Fragment)，意思是不能分片. 只有当 DF = 0 时才允许分片.
- 段位移 ：又称片位移，相对于用户数据字段的起点，该片从何处开始. 片位移以 8 个字节为偏移单位. 所以，每个分片的长度一定是 8 字节的整数倍.
- 生存时间 ：TTL(Time To Live). 数据报能在因特网中经过路由器的最大次数为 255 次，每经过一个路由器则 TTL － 1，为 0 时丢弃该报文.
- 协议 ：记录该报文所携带的数据是使用何种协议.
- 首部检验和 ：只检验数据报的首部，不检验数据部分. 不为 0 则丢弃报文.
- 源地址和目的地址 ：不解释

IP 层转发分组的流程

每个路由器内部都维护一个路由表，路由表包含以下内容(目的网络地址，下一跳地址).

使用子网时分组转发时，路由表必须包含以下三项内容：目的网络地址，子网掩码和下一跳地址.

特定主机路由 ：对特定的目的地址指明一个路由

默认路由 ：不知道分组该发给哪个路由器时就发给默认路由. 当一个网络只有很少的对外连接时使用默认路由非常合适.

路由器的分组转发算法

1. 从数据报中拿到目的 IP 地址 D，得出目的网络地址 N
2. 若 N 就是与此路由器直接相连的某个网络地址，则直接交付(不需要再交给其他路由器转发，直接找到该目的主机交付)，否则 -> (3)
3. 若路由表中有目的地址为 D 的特定主机路由，则把数据报传给该路由器，否则 -> (4)
4. 若路由表中有到达网络 N 的路由，则把数据报传给该路由器，否则 -> (5)
5. 若路由表中有默认路由，则交给该路由器，否则 -> (6)
6. 报告转发分组出错

虚拟专用网 VPN

因特网中的所有路由器对该目的地址是专用地址的数据报一律不转发，下面有 3 种专用地址(虚拟 IP 地址)

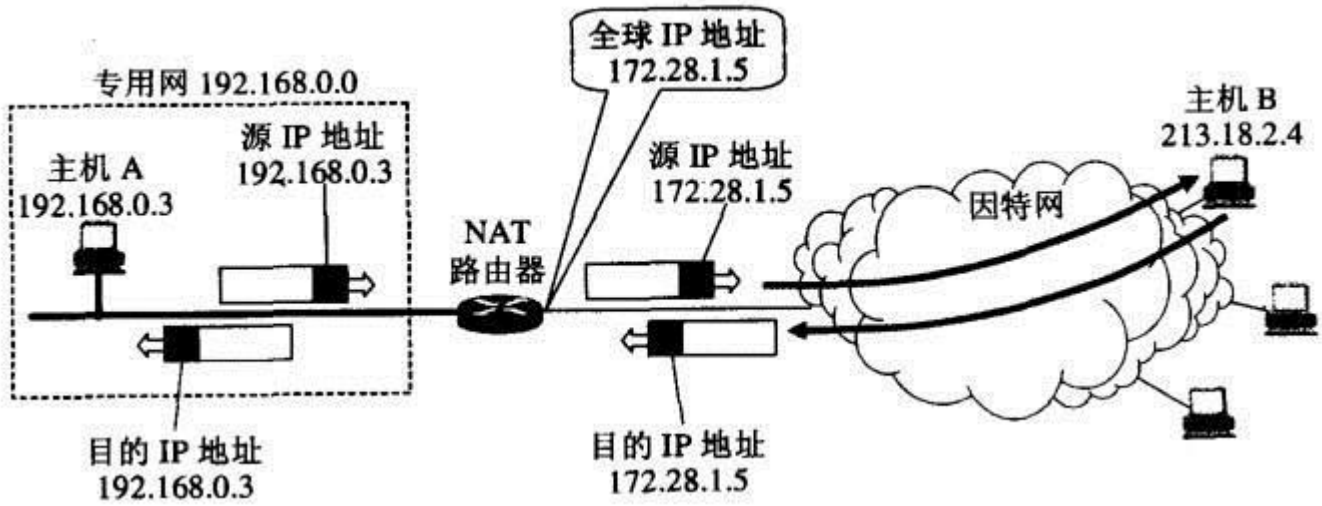
- 10.0.0.0 ~ 10.255.255.255
- 172.16.0.0 ~ 172.31.255.255
- 192.168.0.0 ~ 192.168.255.255

假设现在公司 A 有一个部门在广州和另一个在上海，而他们在当地都有自己的专用网. 那么怎么将这两个专用网连接起来呢?

1. 租用电信的通信线路为本机构专用，但是太贵了
2. 利用公用的因特网当做通信载体，这就是虚拟专用网 VPN

网络地址转换 NAT

多个专用网内部的主机公用一个 NAT 路由器的 IP 地址，在主机发送和接收 IP 数据报时必须先通过 NAT 路由器进行网络地址转换.



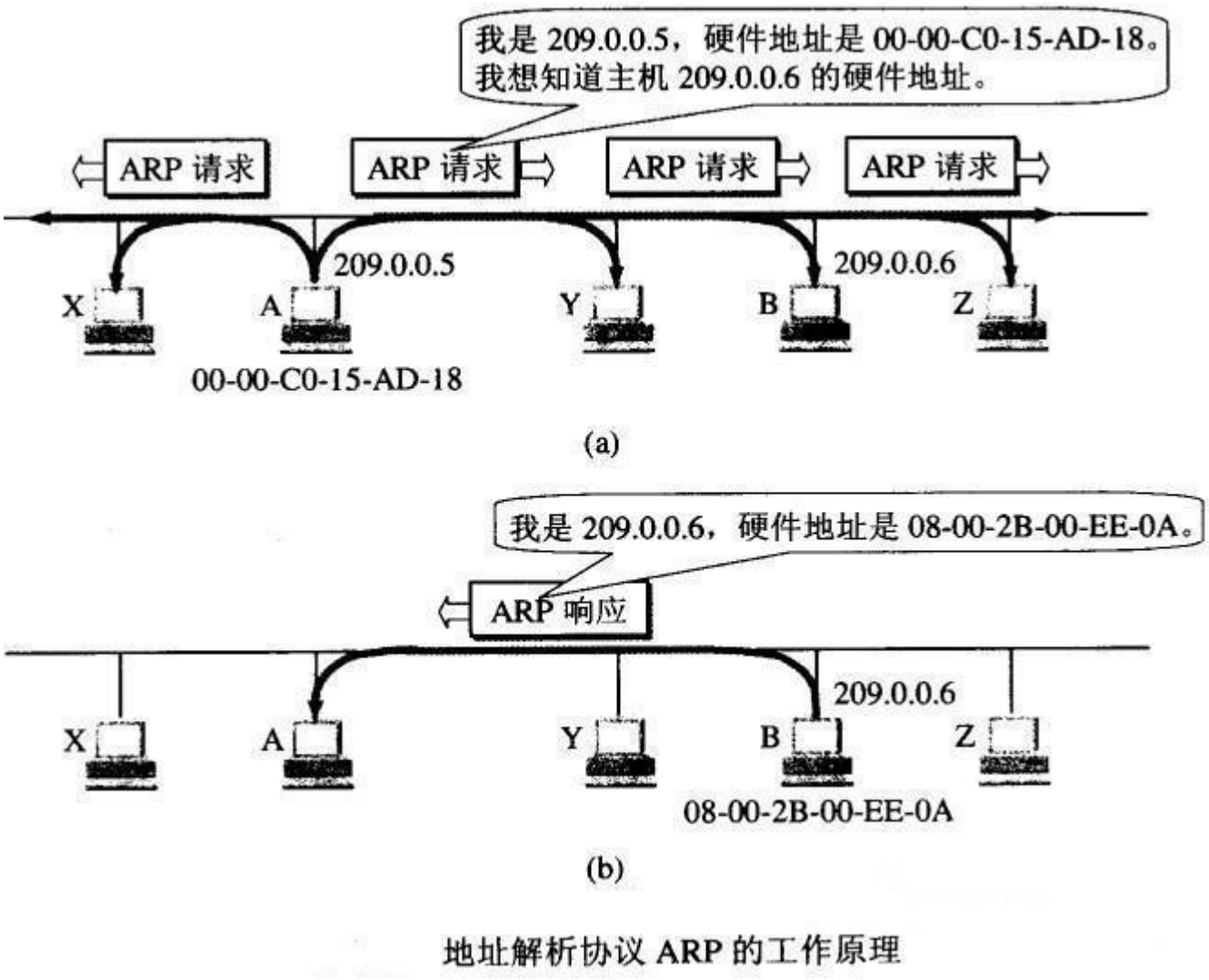
NAT 路由器的工作原理

不仅如此，NAT 还能使用端口号，摇身一变成为网络地址和端口转换 NAPT

ARP 协议

ARP 是解决同一个局域网上的主机或路由器的 IP 地址和 MAC 地址的映射问题，即 IP 地址 -> ARP -> MAC 地址

每一个主机都有一个 ARP 高速缓存，里面有本局域网上的各主机和路由器的 IP 地址到 MAC 地址的映射表. 以下是 ARP 的工作原理 ：



ARP 的工作原理

那如果是跨网络使用 ARP 呢?

1. 在本网络上广播
2. 未找到该主机，则到路由器
3. 路由器帮忙转发(在另一网络上广播)
4. 找到了则完成 ARP 请求，未找到则返回(2)

传输层

这一层是重中之重，因为数据链路层，网络层这两层的数据传输都是**不可靠的，尽最大能力交付的**. 什么意思的? 就是它们不负责提交给你的就是正确的数据. 然而这一层的 TCP 协议将要提供可靠传输

这一层主要重点是两个协议 : UDP 和 TCP

用户数据报协议 UDP

UDP 主要特点 :

- 无连接
- 尽最大努力交付
- 面向报文 : 应用层交下来的报文直接加上 UDP 头部就往 IP 层扔，不合并也不拆分
- 没有拥塞控制
- 支持一对一，一对多，多对一和多对多的交互通信
- 首部开销小，只有 8 个字节

UDP 首部

UDP 首部格式

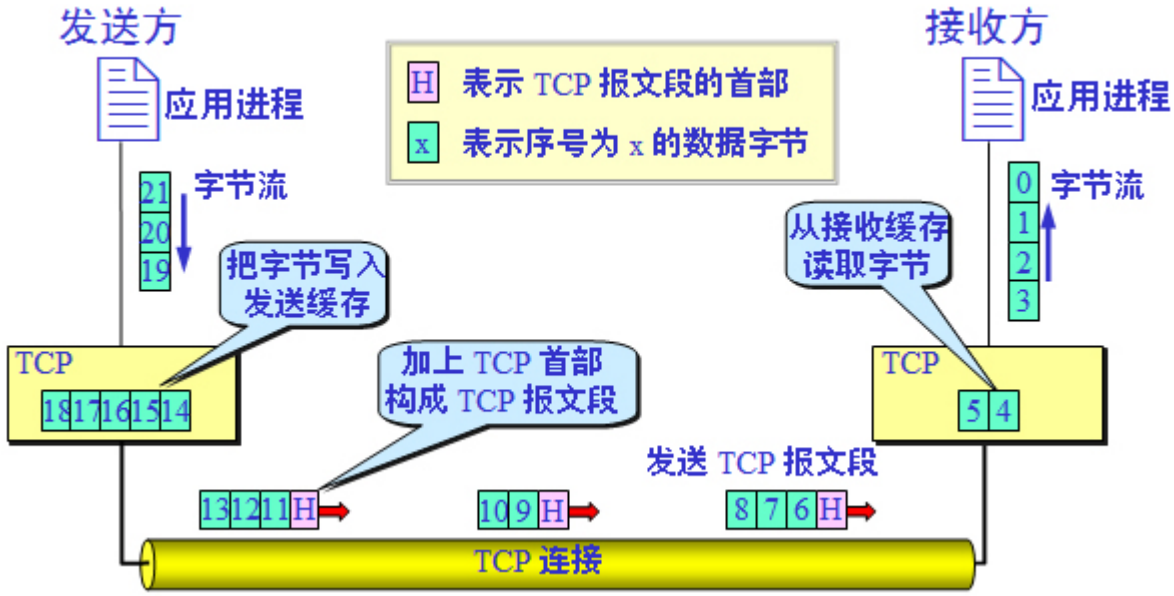
- 源端口 : 源端口号. 在需要对方回信时选用，不需要则全 0
- 目的端口 : 目的端口号. 这在终点交付报文时必须要使用到
- 长度 : UDP 数据报的长度，最小值为 8(仅有首部)
- 检验和 : 与 IP 数据报只检验首部不同的是，UDP 需要把首部和数据部分一起检验

传输控制协议 TCP

TCP 主要特点 :

- 面向连接的运输层协议
- 每一条 TCP 连接只能有 2 个端点，TCP 是点对点的
- 提供可靠交付
- 全双工通信
- 面向字节流

TCP 的工作流程



TCP 字节流

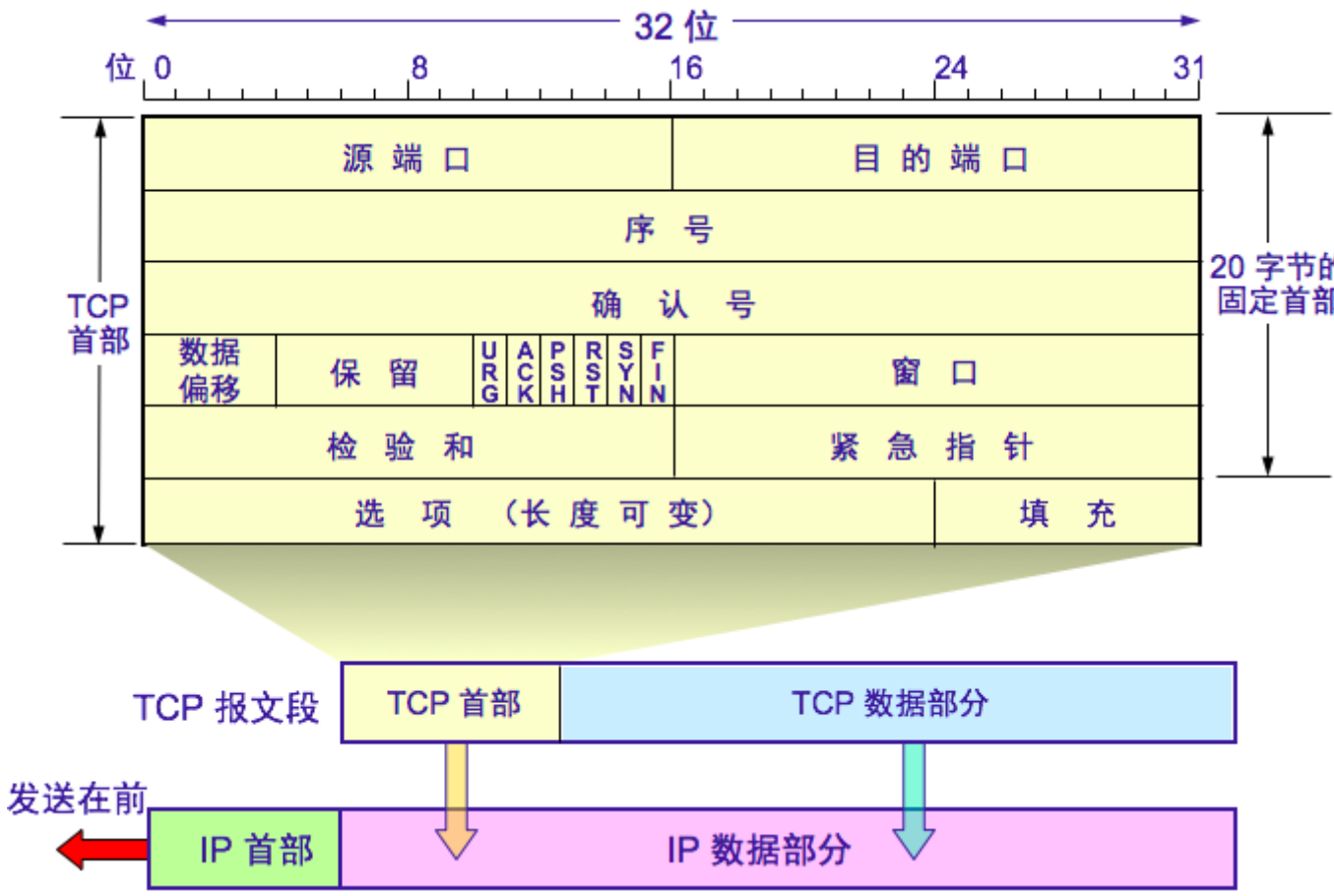
TCP 的连接

TCP 连接的端点叫套接字(socket)

socket = (IP 地址 : 端口号)

每一条 TCP 连接唯一地被通信两端的两个端点(socket)所确定. 即 :
TCP 连接 ::= {socket1, socket2} = {(IP1 : port1), (IP2 : port2)}

TCP 报文段的首部



TCP 报文段的首部

- 源端口和目的端口 ： 同 UDP 端口作用
- 序号 ： 本报文段的数据的第一个字节的序号
- 确认号 ： 期望收到对方下一个报文段的第一个数据字节的序号
  若确认号 = N，则表明 ： 到序号 N-1 为止的所有数据都已正常收到
- 数据偏移 ：TCP 报文段的首部长度
- 保留 ： 以后用，目前为 0
- 紧急 URG： 若 URG = 1 时，说明紧急指针字段有效，告诉系统这是紧急数据，应尽快传送. 例如突然要中断传送
- 确认 ACK : ACK = 1 时确认号才有效，ACK = 0 时确认号无效. TCP 规定，连接建立后所有传送的报文段都必须把 ACK 置 1
- 推送 PSH： 若 PSH = 1，则接收方收到报文段之后不再等到整个缓存满而是直接向上交付
- 复位 RST： 当 RST = 1，说明 TCP 连接有严重错误，必须释放连接再重连
- 同步 SYN： 在连接建立时用来同步序号. 当 SYN = 1，ACK = 0 时表明这是一个连接请求报文段，对方若同意建立连接，则在响应的报文段中置 SYN = 1，ACK = 1
- 终止 FIN： 当 FIN = 1，表明此报文段的发送方数据已发送完毕，并要求释放连接
- 窗口 ： 告诉对方 ： 从本报文段首部中的确认号算起，接收方目前允许对方发送的数据量. 这是作为接收方让发送方设置其发送窗口的依据
- 检验和 ： 同 UDP，检验首部和数据部分

- 紧急指针 ： 当 URG = 1 时有效，指出紧急数据的末尾在报文段的位置
- 选项 ： 最大可 40 字节，没有则为 0
  最大报文段长度 MSS(Maximum Segment Size) ： 每一个 TCP 报文段中数据字段的最大长度，若不填写则为默认的 536 字节.

窗口

TCP 中很重要的一个概念，那就是窗口(发送窗口和接收窗口)

窗口

由于停止等待协议非常低效，于是衍生出窗口这一概念. 上图为发送方维持的发送窗口，位于发送窗口的 5 个分组都可以连续发送出去而不需要等待对方的确认. 每收到一个确认，就把发送窗口前移一个分组的位置. 这大大提高了信道利用率!

接收方不必发送每个分组的确认报文，而是采用累积确认的方式. 也就是说，对按序到达的最后一个分组发送确认报文.

超时重传

如果发送方等待一段时间后，还是没收到 ACK 确认报文，就会启动超时重传. 这个等待的时间为重传超时时间(RTO, Retransmission TimeOut).

然而，RTO 的值不是固定的，这个时间总是略大于连接往返时间(RTT，Round Trip Time). 假设报文发送过去需要 5 秒，对方收到后发送确认报文回来也需要 5 秒，那么 RTT 就为 10 秒，那这 RTO 就要比 10 秒要略大一些. 那么超过 RTO 之后还没有收到确认报文就认为报文丢失了，就要重传.

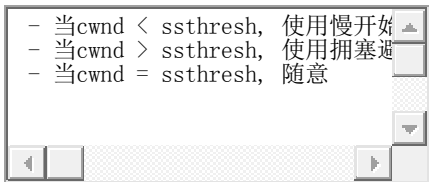
流量控制

利用滑动窗口和报文段的发送时机来进行流量控制.

拥塞控制

发送方维持一个拥塞窗口 cwnd，发送窗口 = 拥塞窗口.

慢开始 ： cwnd = 1，然后每经过一个传输轮次就翻倍
拥塞避免 ： 让 cwnd 缓慢增大，每经过一个传输轮次就+1
慢开始门限 ssthresh ：



1 - 当 cwnd < ssthresh, 使用慢开始算法
2 - 当 cwnd > ssthresh, 使用拥塞避免算法
3 - 当 cwnd = ssthresh, 随意

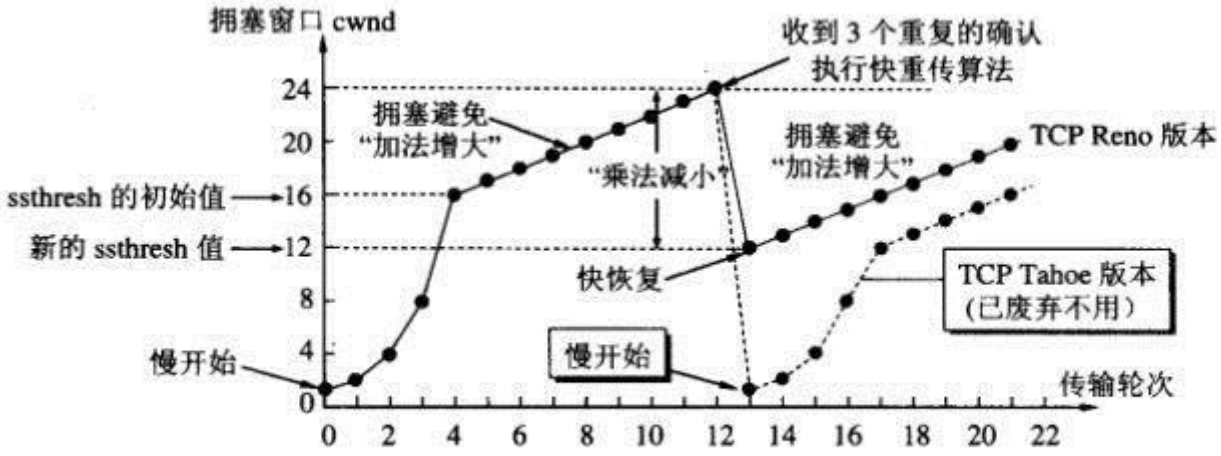


图 5-27　从连续收到三个重复的确认转入拥塞避免

拥塞控制

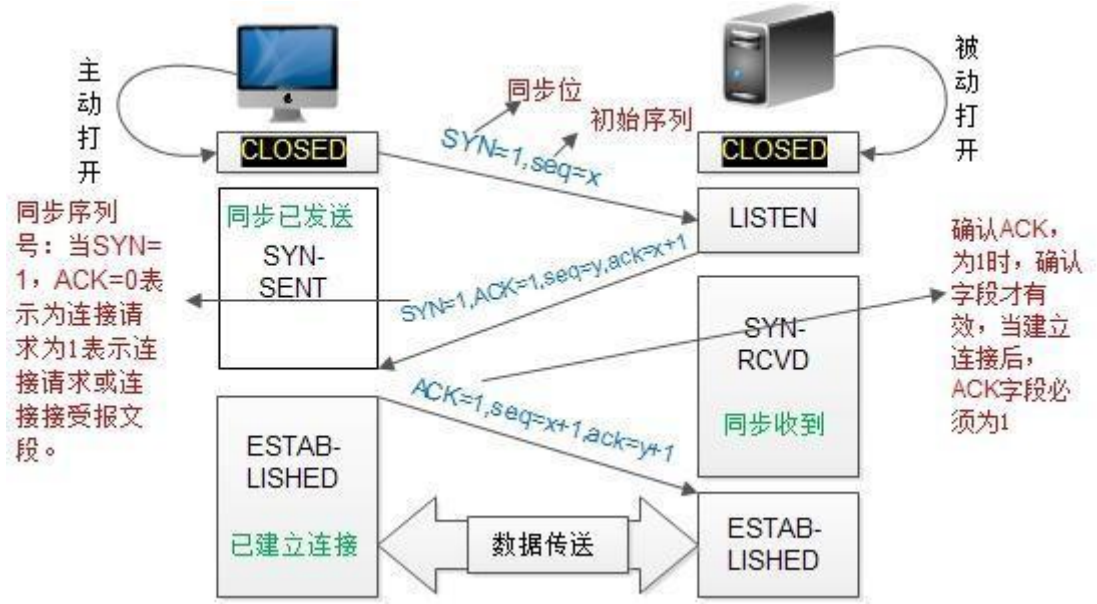
只要判断网络出现拥塞，把 ssthresh 设为当前发送拥塞窗口的一半(不能小于 2)，并把 cwnd 设为 1，重新执行慢开始算法.

除了慢开始和拥塞避免算法外，还有一组快重传和快恢复算法 ：

快重传 ： 接收方及时发送确认，而发送方只要一连收到三个重复确认，马上重传
快恢复 ： 当发送方一连收到三个重复确认时，ssthresh 减半，cwnd 设为 ssthresh.

TCP 三次握手

TCP 三次握手建立连接和四次挥手断开连接是面试爱问的知识点.

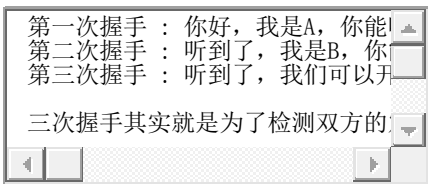


TCP 三次握手

Q ： 为什么要三次握手，两次不可以吗?

A： 试想一下，A 第一次发送请求连接，但是在网络某节点滞留了，A 超时重传，然后这一次一切正常，A 跟 B 就愉快地进行数据传输了. 等到连接释放了以后，那个迷失了的连接请求突然到了 B 那，如果是两次握手的话，B 发送确认，它们就算是建立起了连接了. 事实上 A 并不会理会这个确认，因为我压根没有要传数据啊. 但是 B 却傻傻地以为有数据要来，苦苦等待. 结果就是造成资源的浪费.

更加接地气的解释就是 ： A 打电话给 B



1 第一次握手 ： 你好，我是 A，你能听到我说话吗
2 第二次握手 ： 听到了，我是 B，你能听到我说话吗
3 第三次握手 ： 听到了，我们可以开始聊天了
4

TCP 四次挥手

TCP 四次挥手

Q ： 为什么要四次挥手，而不是两次，三次?

A ：
首先，由于 TCP 的全双工通信，双方都能作为数据发送方. A 想要关闭连接，必须要等数据都发送完毕，才发送 FIN 给 B. (此时 A 处于半关闭状态)
然后，B 发送确认 ACK，并且 B 此时如果要发送数据，就发送(例如做一些释放前的处理)
再者，B 发送完数据之后，发送 FIN 给 A. (此时 B 处于半关闭状态)
然后，A 发送 ACK，进入 TIME-WAIT 状态
最后，经过 2MSL 时间后没有收到 B 传来的报文，则确定 B 收到了 ACK 了.(此时 A，B 才算是处于完全关闭状态)

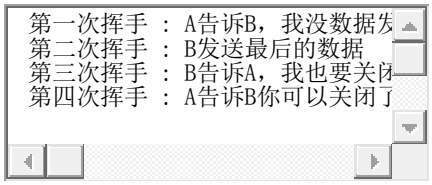
PS ： 仔细分析以上步骤就知道为什么不能少于四次挥手了.

Q ： 为什么要等待 2MSL(Maximum Segment Lifetime)时间，才从 TIME_WAIT 到 CLOSED?

A ： 在 Client 发送出最后的 ACK 回复，但该 ACK 可能丢失。Server 如果没有收到 ACK,将不断重复发送 FIN 片段。所以 Client 不能立即关闭,它必须确认 Server 接收到了该 ACK。Client 会在发送出 ACK 之后进入到 TIME_WAIT 状态。Client 会设置一个计时器,等待 2MSL 的时间。如果在该时间内再次收到 FIN,那么 Client

会重发 ACK 并再次等待 2MSL。MSL 指一个片段在网络中最大的存活时间，2MSL 就是一个发送和一个回复所需的最大时间。如果直到 2MSL，Client 都没有再次收到 FIN，那么 Client 推断 ACK 已经被成功接收，则结束 TCP 连接。

更加接地气的解释 ：



1 第一次挥手 :A 告诉 B，我没数据发了，准备关闭连接了，你要发送数据吗
2 第二次挥手 :B 发送最后的数据
3 第三次挥手 :B 告诉 A，我也要关闭连接了
4 第四次挥手 :A 告诉 B 你可以关闭了，我这边也关闭了

应用层

应用层协议最著名的就是 HTTP，FTP 了，还有一个重要的 DNS

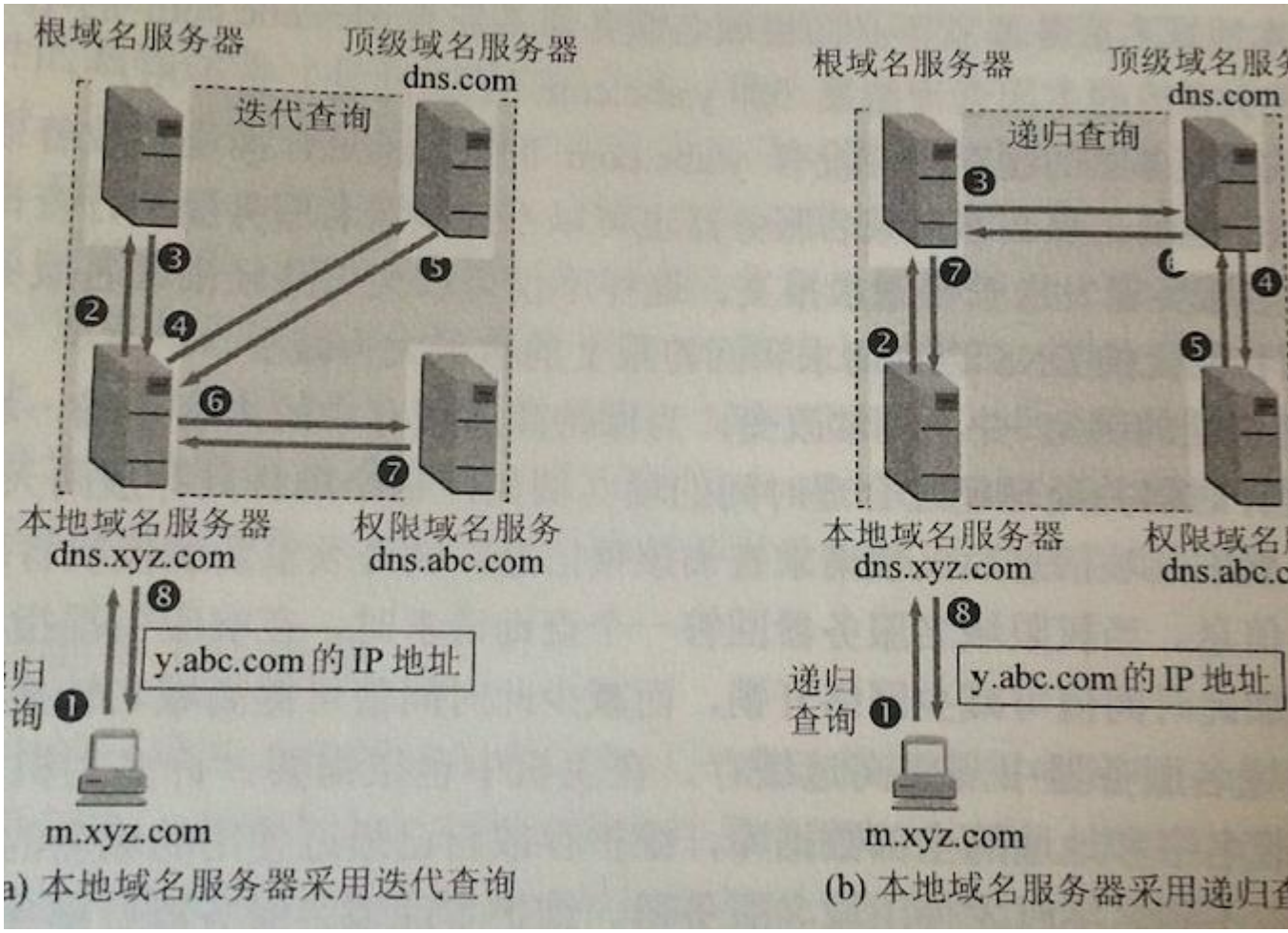
域名系统(DNS，Domain Name System)

DNS 能将域名(例如，www.jobbole.com)解析成 IP 地址.

域名服务器分类

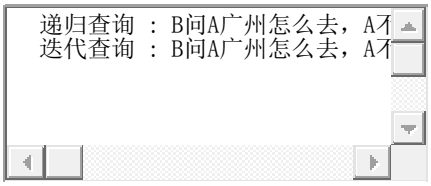
- 根域名服务器 ： 最高层次的域名服务器
- 顶级域名服务器 ： 如其名
- 权限域名服务器 ： 负责一个区的应服务器
- 本地域名服务器 ： 主机发送 DNS 查询请求就是发给它

DNS 查询



DNS 查询

1. 主机向本地域名服务器的查询一般都是采用**递归查询**
2. 本地域名服务器向根域名服务器的查询通常是采用**迭代查询**



递归查询 :B 问 A 广州怎么去，A 不知道，A 就问 C，C 不知道就问 D...直到知道了 1 再一层一层转告直到 A 告诉 B.

2 迭代查询 :B 问 A 广州怎么去，A 不知道，A 就告诉你可以去问 C，然后 B 就去问 C，C 不知道，C 就告诉你可以去问 D，然后 B 就去问 D...直到 B 知道为止

DNS 查询例子 ： 域名为 x.tom.com 的主机想知道 y.jerry.com 的 IP 地址

1. 主机 x.tom.com 先向本地域名服务器 dns.tom.com 进行递归查询

2. 本地域名服务器采用迭代查询. 它先问一个根域名服务器
3. 根域名服务器告诉它，你去问顶级域名服务器 dns.com
4. 本地域名服务器问顶级域名服务器 dns.com
5. 顶级域名服务器告诉它，你去问权限域名服务器 dns.jerry.com
6. 本地域名服务器问权限域名服务器 dns.jerry.com
7. 权限域名服务器 dns.jerry.com 告诉它所查询的主机的 IP 地址
8. 本地域名服务器把查询结果告诉主机 x.tom.com

1 PS： 该查询使用 UDP，并且为了提高 DNS 查询效率，每个域名服务器都使用高速缓存.

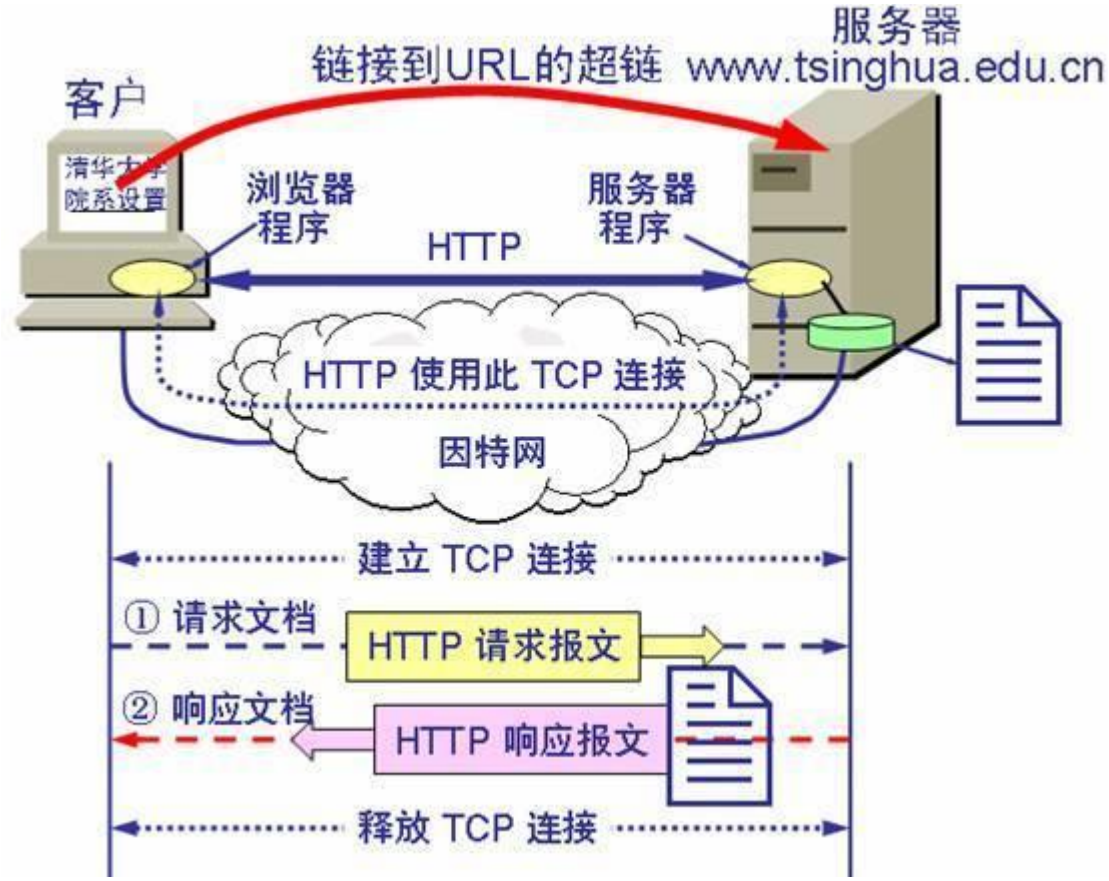
URL

URL 的格式 ： <协议>://<主机>:<端口>/<路径>, 端口和路径有时可省略.

使用 HTTP 协议的 URL ： http://<主机>:<端口>/<路径>, HTTP 默认端口号是 80

HTTP 协议

HTTP 是面向事务的，即它传输的数据是一个整体，要么全部收到，要么全部收不到.

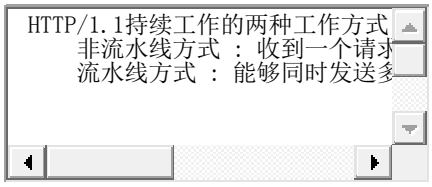


每一次 HTTP 请求就需要建立一次 TCP 连接和释放 TCP 连接.

HTTP 是无连接，无状态的. 每一次请求都是作为一次新请求.

HTTP/1.0 缺点 ： 无连接，每一次请求都要重新建立 TCP 连接，所以每一次 HTTP 请求都要花费 2 倍 RTT 时间(一次 TCP 请求，一次 HTTP 请求)

HTTP/1.1 ： 使用持续连接，即保持 TCP 连接一段时间.



1 HTTP/1.1 持续工作的两种工作方式 ： 非流水线方式和流水线方式
2 非流水线方式 ： 收到一个请求的响应再发下一个请求，效率低，浪费资源
3 流水线方式 ： 能够同时发送多个请求，效率高

HTTP 的 GET 和 POST

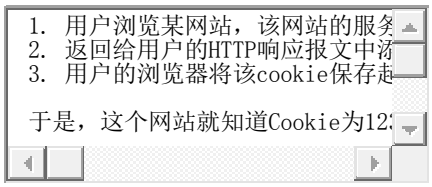
GET 请求通常用于查询、获取数据，而 POST 请求则用于发送数据
GET 请求的参数在 URL 中，因此绝不能用 GET 请求传输敏感数据，而 POST 请求的参数在请求头中，安全性略高于 GET 请求

1 ps : POST 请求的数据也是以明文的形式存放在请求头中，因此也不安全

Cookie

万维网使用 Cookie 来跟踪用户，表示 HTTP 服务器和用户之间传递的状态信息.

Cookie 工作原理 ：



1. 用户浏览某网站，该网站的服务器为用户产生一个唯一的识别码，并以此为索引在服务器后端数据库中产生一个项目
2. 返回给用户的 HTTP 响应报文中添加一条 "Set-cookie"，值为该识别码，如 123
3. 用户的浏览器将该 cookie 保存起来，在用于继续浏览该网站时发送的每一个 HTTP 请求都会有一行 Cookie: 123

于是，这个网站就知道 Cookie 为 123 的这个用户做了什么，为这个用户维护一个独立的列表(如购物车)

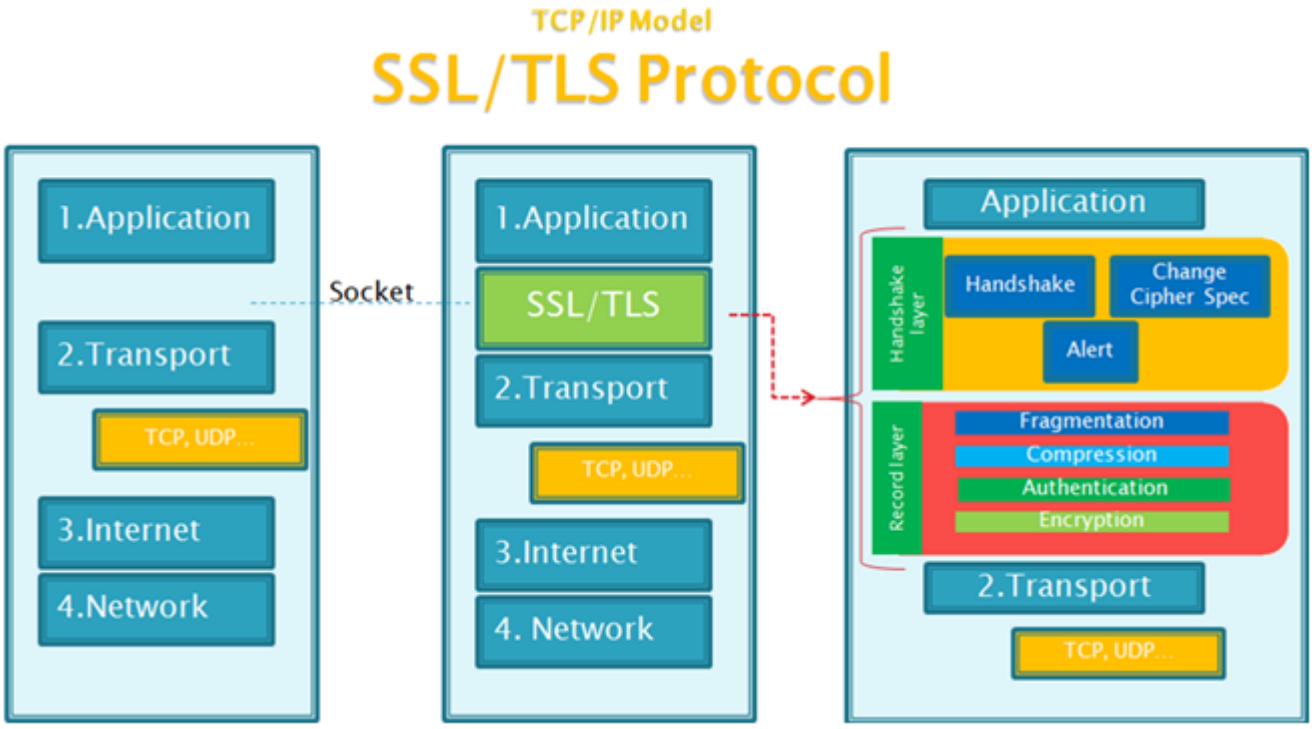
当然，Cookie 是把双刃剑，方便的同时也带有危险性，例如隐私泄露等，用户可以自行决定是否使用 Cookie

Session

Cookie 是保存在客户端上的，而 Session 是保存在服务器中. 当服务器收到用户发出的 Cookie 时，会根据 Cookie 中的 SessionID 来查找对应的 Session，如没有则会生成一个新的 SessionID 返回给用户

总而言之，Cookie 和 Session 就是同一样东西存放地方不同而已.

HTTPS



HTTPS 协议在 HTTP 协议的基础上，在 HTTP 和 TCP 中间加入了一层 SSL/TLS 加密层，解决了 HTTP 不安全的问题：冒充，篡改，窃听三大风险.