# 计算机网络

## TIPS

Host header就是 比如你三个域名解析到同一个ip上了, 就需要用host来区分具体是那个地址了

一个域名可以对应多个ip (比如给你来个源站最近的)

一个ip可以被多个域名绑定.

域名劫持 如果用户浏览器缓存中没有数据，浏览器会查找操作系统缓存中是否有这个域名对应的DNS解析结果。其实操作系统也有一个域名解析的过程，在Windows中可以通过C:\Windows\System32\drivers\etc\hosts文件来设置，在Linux中可以通过/etc/hosts文件来设置，

电路交换 这跟网线 如果被两台机器用来通信了,其他信息就无法通过这跟网线来传递了,就得等这两台机器的通信断了才能用这跟网线.

分组交换 就是把数据切成一个个小的数据包,包里面标记上要发给谁,然后通过这个网线发送,这样的话很多对机器就可以同时使用这跟网线.

中继器 物理层的路由器 网桥 数据链路层的路由器

交换机 数据链路层的路由器

路由器 ip层的延长网络的设备

以太网首部->ip包首部->tcp包首部->数据

在一个局域网(以太网)内,用mac就可以互相发数据,但是一旦要跨局域网mac就不行了,就得借助于ip了!!!

ARP(网络层协议)通过ip找mac

Mac地址是下一跳的地址老换

Ip地址是指示着我们前进的最终地址

一个网卡只有一个mac,却可以有多个ip

为了确保数据通信的安全，HTTPS已广泛应用于互联网，浏览器与服务器之间的HTTPS通信都是加密的。然而当浏览器需要通过代理服务器发起HTTPS请求时，由于请求的站点地址和端口号都是加密保存于HTTPS请求头中的，代理服务器是如何既确保通信是加密的（代理服务器自身也无法读取通信内容）又知道该往哪里发送请求呢？为了解决这个问题，浏览器需要先通过明文HTTP形式向代理服务器发送一个CONNECT请求告诉它目标站点地址及端口号。当代理服务器收到这个请求后，会在对应的端口上与目标站点建立一个TCP连接，连接建立成功后返回一个HTTP 200状态码告诉浏览器与该站点的加密通道已建成。接下来代理服务器仅仅是来回传输浏览器与该服务器之间的加密数据包，代理服务器并不需要解析这些内容以保证HTTPS的安全性。

什么时候会用到CONNECT方法？

只有当浏览器配置为使用代理服务器时才会用到CONNECT方法

所以说ping命令就是icmp协议,ip也是网络层的,但是这两个协议之前肯定会有一个arp的流程,获取一跳跳的mac地址.

IGMP协议

组管理协议 就是ip多播(组播)的组管理的,也是网络层的协议.

信令就用tcp,音视频啊数据包啊就用udp

Tcp的可靠性的保证 玩的就是序列号和ack机制

这样就会有超时重传机制.

三次握手

Client 我要连你了 协商点东西吧

Server 就这些信息 看看行不

Client 可以可以 明白了 通知你下

Client xxxx

Xxx

Xxx

Client发完第三次握手后就开始发数据了.

主机地址的那些位都是1的就是广播地址.

172开头的是B类地址,192开头的是C类地址

由于链路层有路径MTU的存在,所以ip层会存在分片.

v4 4个8比特 v6 8个16比特

滑动窗口(可以不收到ack还继续发送数据)

快速重传(收到了a4的ack,但是没有收到a3的ack,说明a3包丢了,我们马上重传a3的包)

由于滑动窗口 发送端要有个缓存区 缓存那些发了但是还没收到ack的消息

窗口大小动态变化

拥塞控制 分为慢启动(一开始发送包的速度是1248指数增长)和拥塞避免(到达一定数量后就由指数变成线性)

快速恢复就是一旦发现开始丢包了,把速率直接砍一半,然后不用慢启动了,直接用拥塞避免这种线性的.

捎带应答,就是发送端有请求,接收端也会有响应消息,正好我们把ack捎在响应消息里.

TCP是有校验和的,可以判断包在传输过程中有没有丢失.

http无状态无上下文,之所以做了这么简单是为了更大的并发量,如果想保存些上下文就需要用到cookie和session.

http长连接可以避免大量的tcp连接创建断开问题.

所谓通信中的隧道就是通过加密啊或者加头啊,封装一层外面看不懂的东西,对外面就是透明的就是看不懂的字节流,那么两个节点的通信就像是有一个隧道一样.

URI就是唯一资源标识符,URL是URI的一种实现方式.

## HTTPS

非对称加密 私钥加密 公钥可以解密 公钥加密 私钥也可以解密。

一般两端需要交互，都是先把自己的公钥给对方（公钥可以给很多人），私钥只有自己有，别人拿到了你的公钥，给你发消息时就会用公钥加密发给你，加密后的消息别人截获了也没有，别人没有你的私钥也解不了。

一般都是服务器把公钥发给用户，私钥服务端自己保存，用户收到服务器的公钥后把自己的公钥用服务器的公钥加密，发给服务器，服务器用自己的私钥解开后就拿到了用户的公钥，这样两端都保存了对方的公钥，发的消息都用对方的公钥加密了，通信就是安全的了。

问题就是第一步服务器把自己的公钥发给用户的时候还是不安全的。

解决方案就是服务端把自己的公钥告诉一个第三方机构 CA机构，CA把自己的信息啥的和服务器的公钥用自己的私钥加密成一个证书发给服务器，服务器把这个证书发给客户端，因为浏览器维护了大部分CA机构的名称地址和公钥信息，就可以验证这个证书是不是合法的对的，如果是真的，就用对应的CA机构的公钥解密这个证书拿到服务器的公钥就可以继续后面的流程了。

## 三次握手与四次挥手

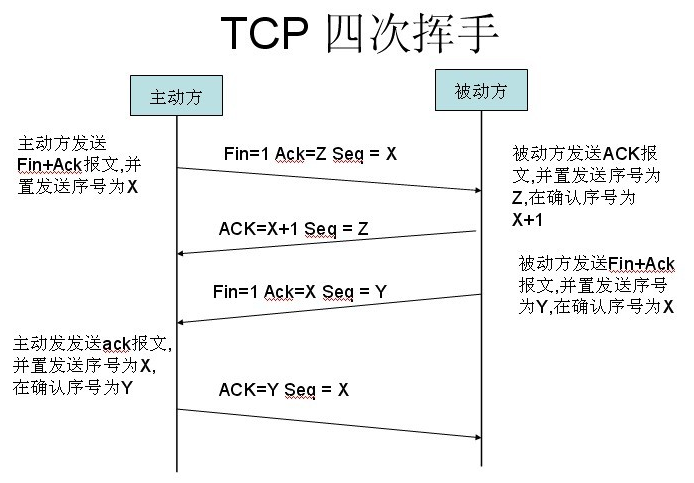


SYN叫做同步位,ACK叫做响应位,这俩大写的,每次都是1,是不会变的.

seq小写的这个才是序列号,注意client和server都会维护自己的seq序列号的,client和server也都是会给对面回复ack(小写)的时候ack=对面seq+1的.

为啥要三次握手呢,tcp连接是全双工的,也就是A给B发消息的同时B也能给A发消息.

那么A给B发完消息后,需要收到B的ack才能确定A->B的通路打通,如果只有两次握手的话client->server的通路可以确定通了,但是server->client的还没有通,因为server给client发的消息(第二次握手)没有收到ack,所以要有第三次握手.



四次挥手两边谁都可以触发,顺序并不一定是上面那样,也就是谁确定自己不给对面发消息了,就给对面发个FIN,但是对面可能还给你发,所以在四次挥手的时候会有个半关闭的概念.

四次挥手就是你给我发FIN,我给你ack,我给你发FIN,你给我回ACK.

这是因为服务端的LISTEN状态下的SOCKET当收到SYN报文的连接请求后，它可以把ACK和SYN(ACK起应答作用，而SYN起同步作用)放在一个报文里来发送。但关闭连接时，当收到对方的FIN报文通知时，它仅仅表示对方没有数据发送给你了；但未必你所有的数据都全部发送给对方了，所以你可能未必会马上会关闭SOCKET,也即你可能还需要发送一些数据给对方之后，再发送FIN报文给对方来表示你同意现在可以关闭连接了，所以它这里的ACK报文和FIN报文多数情况下都是分开发送的。

## DNS

浏览器缓存dns, 和host文件都找不到的时候，就要用到我们网络配置中的”DNS服务器地址”了。

操作系统会把这个域名发送给这个LDNS，也就是本地区的域名服务器。这个DNS通常都提供给用户本地互联网接入的一个DNS解析服务，例如用户是在学校接入互联网，那么用户的DNS服务器肯定在学校；

如果用户是在小区接入互联网，那么用户的DNS就是再提供接入互联网的应用提供商，即电信或联通，也就是通常说的SPA，那么这个DNS通常也会在用户所在城市的某个角落，不会很远。Windows环境下通过命令行输入ipconfig，Linux环境下通过cat /etc/resolv.conf就可以查询配置的DNS服务器了。这个专门的域名解析服务器性能都会很好，它们一般都会缓存域名解析结果，当然缓存时间是受到域名的失效时间控制的。大约80%的域名解析到这里就结束了，所以LDNS主要承担了域名的解析工作。

4. 如果LDNS仍然没有命中，就直接到Root Server域名服务器请求解析，跟域名服务器全球只有13台。

5、根域名服务器返回给本地域名服务器一个所查询的主域名服务器（gTLD Server）地址。gTLD是国际顶级域名服务器，如.com、.cn、.org等。

6、本地域名服务器LDNS再向上一步返回的gTLD服务器发送请求

7、接受请求的*gTLD服务器查找并返回此域名对应的Name Server域名服务器的地址，这个Name Server通常就是用户注册的域名服务器，例如用户在某个域名服务提供商申*请的域名，那么这个域名解析任务就由这个域名提供商的服务器来完成

什么阿里云 百度 万网都是域名服务商, 他们都被gTLD管理, 这个域名gTLD知道是那个域名服务商提供的.

8、Name Server域名服务器会查询存储的域名和IP的映射关系表，在正常情况下都根据域名得到目标IP地址，连同一个TTL值返回给DNS Server域名服务器

9、返回该域名对应的IP和TTL值，LDNS会缓存这个域名和IP的对应关系，缓存时间由TTL值控制

10、把解析的结果返回给用户，用户根据TTL值缓存在本地系统缓存中，域名解析过程结束

在实际的DNS解析过程中，可能还不止这10步，如Name Server可能有很多级，或者有一个GTM来负载均衡控制，这都有可能会影响域名解析过程 [2]。

## A记录

域名 www.xx.com → 111.111.111.111,也就是当你访问这些域名或者主机名的时候，DNS服务器上会通过A记录会帮你解析出相应的IP地址，以达到后续访问目的。所以A记录是IP解析，直接将域名或主机名指向某个IP。

## CNAME

CNAME记录，也叫别名记录，相当于给A记录中的域名起个小名儿，比如www.xx.com的小名儿就叫www.yy.com好了，然后CNAME记录也和A记录一样，是一种指向关系，把小名儿www.yy.com指向了www.xx.com，然后通过A记录，www.xx.com又指向了对应的IP：

这时候有人问：这不多了一步嘛，不嫌麻烦？

假如这个时候我又想给原域名取几个小名儿，分别叫www.cc.com和www.kk.com那么存在下列指向关系：

www.yy.com → www.xx.com → 111.111.111.111  
www.cc.com → www.xx.com → 111.111.111.111  
www.kk.com → www.xx.com → 111.111.111.111  
突然服务器的IP地址因为一些不可描述的原因要换了，不再是111.111.111.111了，换成了333.333.333.333，这时候你发现，只要把www.xx.com的指向修改一下即可：

域名 www.xx.com → 333.333.333.333  
这时候你又发现了，原来他的小名儿不需要做更改，直接就能访问服务器，因为他们都只指向了www.xx.com，服务器IP改没改它们不管。

## CNAME与CND

举个CDN的栗子 ：假如你是DD公司老板，你公司中的一台IP为1.1.1.1的服务器，注册了域名为www.dd.com，要对外提供客户访问。随着公司越做越大，访问量也越来越多，服务器顶不住了，你去找CDN提供商购买CDN加速服务，这个时候他们要求你的域名做个CNAME指向他们给你的一个域名叫www.xdd.com，当用户访问www.dd.com的时候，本地DNS会获得CDN提供的CNAME域名：[www.xdd.com，然后再次向DNS调度系统发出请求，通过DNS调度系统的智能分析，把这个www.xdd.com指向一个（离用户地理位置最近的）CDN提供商的服务器IP，让用户就近取到想要的资源（如访问网站），大大降低了延迟。](http://www.xdd.com，然后再次向DNS调度系统发出请求，通过DNS调度系统的智能分析，把这个www.xdd.com指向一个（离用户地理位置最近的）CDN提供商的服务器IP，让用户就近取到想要的资源（如访问网站），大大降低了延迟。)

## 接入网

所以b gw只能提供vip到域名的关系 外面的请求通过bddns获取到vip后就会到b gw, gw可以做些四层的安全, 把tcp攻击的那套给去掉, 然后还可以吧多个bfe代理成一个vip.bfe就类似与ngix是七层的,所以就可以根据http的一些url啊 域名啊 header继续分发.

## CDN

都需要进行CDN域名和服务域名的注册绑定

CDN回源

也就是baidu.com的dns 最后会走到baidu自建的一个dns里面,在这里面可以做各种cname啊 的操作和调度.因为dns是自己建的.

## http

### http1.x

HTTP/1.0传输数据时，每次都需要重新建立连接，增加延迟。

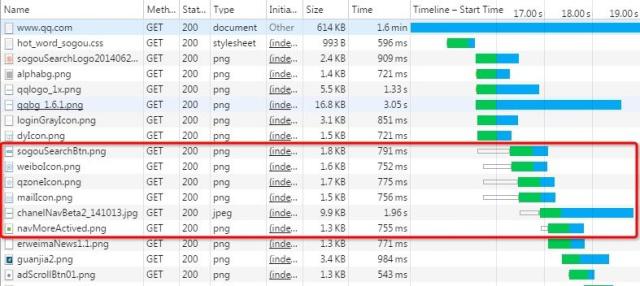
HTTP/1.1虽然加入keep-alive可以复用一部分连接等情况下仍然需要建立多个connection，耗费资源，给服务器带来性能压力。

一个浏览器对一个域名开的tcp连接数是有上限的,所以不能搞很多链接去并行

HOLB是指一系列包（package）因为第一个包被阻塞；当页面中需要请求很多资源的时候，HOLB（队头阻塞）会导致同一个域名的链接在达到最大数量时，剩余的资源需要等待其他资源请求完成后才能发起请求。

所有就会有队头阻塞这种东西.

HTTP 1.1：尝试使用 pipeling 来解决，即浏览器可以一次性发出多个请求（同个域名，同一条 TCP 链接）。但 pipeling 要求返回是按序的，那么前一个请求如果很耗时（比如处理大图片），那么后面的请求即使服务器已经处理完，仍会等待前面的请求处理完才开始按序返回。所以，pipeling 只部分解决了 HOLB



如上图所示，红色圈出来的请求就因域名链接数已超过限制，而被挂起等待了一段时间。

### http2.0

2015年，HTTP/2 发布。HTTP/2是现行HTTP协议（HTTP/1.x）的替代，但它不是重写，HTTP方法/状态码/语义都与HTTP/1.x一样。HTTP/2基于SPDY3，专注于性能，最大的一个目标是在用户和网站间只用一个连接（connection）。

二进制格式传输数据

流

消息

帧

多路复用

### http3.0

虽然 HTTP/2 解决了很多之前旧版本的问题，但是它还是存在一个巨大的问题，主要是底层支撑的 TCP 协议造成的。

上文提到 HTTP/2 使用了多路复用，一般来说同一域名下只需要使用一个 TCP 连接。但当这个连接中出现了丢包的情况，那就会导致 HTTP/2 的表现情况反倒不如 HTTP/1 了。

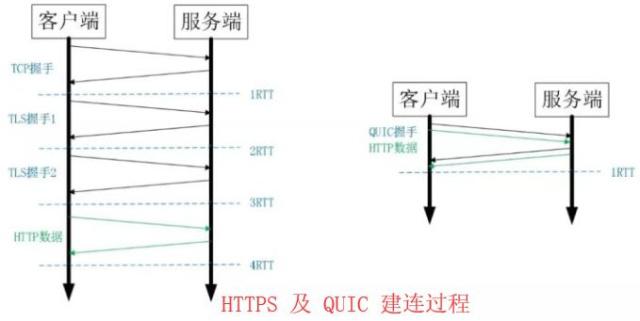
因为在出现丢包的情况下，整个 TCP 都要开始等待重传，也就导致了后面的所有数据都被阻塞了。但是对于 HTTP/1.1 来说，可以开启多个 TCP 连接，出现这种情况反到只会影响其中一个连接，剩余的 TCP 连接还可以正常传输数据。

那么可能就会有人考虑到去修改 TCP 协议，其实这已经是一件不可能完成的任务了。因为 TCP 存在的时间实在太长，已经充斥在各种设备中，并且这个协议是由操作系统实现的，更新起来不大现实。

基于这个原因，Google 就更起炉灶搞了一个基于 UDP 协议的 QUIC 协议，并且使用在了 HTTP/3 上，HTTP/3 之前名为 HTTP-over-QUIC，从这个名字中我们也可以发现，HTTP/3 最大的改造就是使用了 QUIC。

QUIC 虽然基于 UDP，但是在原本的基础上新增了很多功能，接下来我们重点介绍几个QUIC新功能。

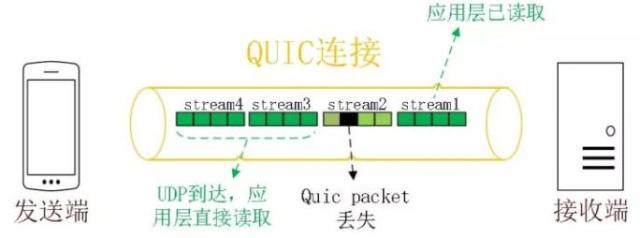
0RTT



多路复用

虽然 HTTP/2 支持了多路复用，但是 TCP 协议终究是没有这个功能的。QUIC 原生就实现了这个功能，并且传输的单个数据流可以保证有序交付且不会影响其他的数据流，这样的技术就解决了之前 TCP 存在的问题。

同HTTP2.0一样，同一条 QUIC连接上可以创建多个stream，来发送多个HTTP请求，但是，QUIC是基于UDP的，一个连接上的多个stream之间没有依赖。比如下图中stream2丢了一个UDP包，不会影响后面跟着 Stream3 和 Stream4，不存在 TCP 队头阻塞。虽然stream2的那个包需要重新传，但是stream3、stream4的包无需等待，就可以发给用户。



另外QUIC 在移动端的表现也会比 TCP 好。因为 TCP 是基于 IP 和端口去识别连接的，这种方式在多变的移动端网络环境下是很脆弱的。但是 QUIC 是通过 ID 的方式去识别一个连接，不管你网络环境如何变化，只要 ID 不变，就能迅速重连上。

## Websocket

server核心代码,需要先升级http的request和writer到websocketconn

//服务升级，对于来到的http连接进行服务升级，升级到ws

cn, err := upgrader.Upgrade(c.Writer, c.Request, nil)

defer cn.Close()

if err != nil {

fmt.Println(err)

return

}

for {

//messageType int, p []byte, err error

mt, message, err := cn.ReadMessage()

if err != nil {

log.Println("read:", err)

break

}

let url ='ws://'+ this.baseUrl.split('//')[1] +'/TotalEmergeProgressWebsocket'

websocker的协议名字就是这样的