网络

## UDP

把数据打包

数据大小有限制（64k）

不建立连接

速度快，但可靠性低

为什么不直接使用IP协议而要额外增加一个UDP协议呢？ 一个重要的原因是IP协议中并没有端口(port)的概念。IP协议进行的是IP地址到IP地址的传输

UDP头：源端口，目的端口，包长度，校验码，共8字节

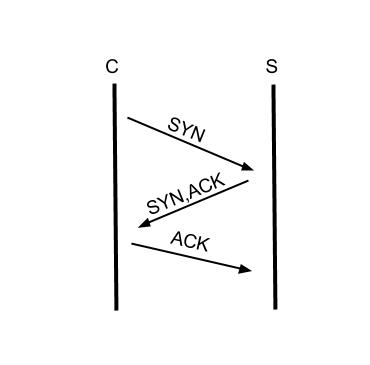
## TCP

以太网数据包（packet）的大小是固定的，最初是1518字节，后来增加到1522字节。其中， 1500 字节是负载（payload），22字节是头信息（head）。

IP 数据包在以太网数据包的负载里面，它也有自己的头信息，最少需要20字节，所以 IP 数据包的负载最多为1480字节。TCP 数据包在 IP 数据包的负载里面。它的头信息最少也需要20字节，因此 TCP 数据包的最大负载是 1480 - 20 = 1460 字节。由于 IP 和 TCP 协议往往有额外的头信息，所以 TCP 负载实际为1400字节左右。

建立连接，三次握手：

第一个报文SYN从发起方发出，第二个报文SYN,ACK是从被连接方发出，第三个报文ACK确认对方的SYN，ACK已经收到，



断开连接，四次握手

客户端发送FIN，服务端返回ACK，服务端发送FIN，客户端返回ACK，断开。

由于TCP连接时全双工的，因此，每个方向都必须要单独进行关闭，这一原则是当一方完成数据发送任务后，发送一个FIN来终止这一方向的连接，收到一个FIN只是意味着这一方向上没有数据流动了，即不会再收到数据了，但是在这个TCP连接上仍然能够发送数据，直到这一方向也发送了FIN。首先进行关闭的一方将执行主动关闭，而另一方则执行被动关闭。

为什么建立连接是三次握手，而关闭连接却是四次挥手呢？

这是因为服务端在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，仅仅表示对方不再发送数据了但是还能接收数据，己方也未必全部数据都发送给对方了，所以己方可以立即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，己方ACK和FIN一般都会分开发送。

为什么TIME\_WAIT状态需要经过2MSL(最大报文段生存时间)才能返回到CLOSE状态？

原因有二：

一、保证TCP协议的全双工连接能够可靠关闭

二、保证这次连接的重复数据段从网络中消失

先说第一点，如果Client直接CLOSED了，那么由于IP协议的不可靠性或者是其它网络原因，导致Server没有收到Client最后回复的ACK。那么Server就会在超时之后继续发送FIN，此时由于Client已经CLOSED了，就找不到与重发的FIN对应的连接，最后Server就会收到RST而不是ACK，Server就会以为是连接错误把问题报告给高层。这样的情况虽然不会造成数据丢失，但是却导致TCP协议不符合可靠连接的要求。所以，Client不是直接进入CLOSED，而是要保持TIME\_WAIT，当再次收到FIN的时候，能够保证对方收到ACK，最后正确的关闭连接。

再说第二点，如果Client直接CLOSED，然后又再向Server发起一个新连接，我们不能保证这个新连接与刚关闭的连接的端口号是不同的。也就是说有可能新连接和老连接的端口号是相同的。一般来说不会发生什么问题，但是还是有特殊情况出现：假设新连接和已经关闭的老连接端口号是一样的，如果前一次连接的某些数据仍然滞留在网络中，这些延迟数据在建立新连接之后才到达Server，由于新连接和老连接的端口号是一样的，又因为TCP协议判断不同连接的依据是socket pair，于是，TCP协议就认为那个延迟的数据是属于新连接的，这样就和真正的新连接的数据包发生混淆了。所以TCP连接还要在TIME\_WAIT状态等待2倍MSL，这样可以保证本次连接的所有数据都从网络中消失。

## HTTP

***Client头：***

Accept，作用： 浏览器端可以接受的媒体类型, 如：  Accept: text/html  代表浏览器可以接受服务器回发的类型为 text/html ，通配符 \* 代表任意类型， Accept: \*/\*  代表浏览器可以处理所有类型,(一般浏览器发给服务器都是发这个)

Accept-Encoding：浏览器申明自己接收的编码方法，通常指定压缩方法，是否支持压缩，支持什么压缩方法（gzip，deflate）

User-Agent：告诉HTTP服务器， 客户端使用的操作系统和浏览器的名称和版本.

Cookie: 最重要的header, 将cookie的值发送给HTTP 服务器

Content-Length：发送给HTTP服务器数据的长度。

Content-Type：如：Content-Type: application/x-www-form-urlencoded

Referer: 提供了Request的上下文信息的服务器，告诉服务器我是从哪个链接过来的

Connection：如：　Connection: keep-alive 当一个网页打开完成后，客户端和服务器之间用于传输HTTP数据的TCP连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连接。如：  Connection: close  代表一个Request完成后，客户端和服务器之间用于传输HTTP数据的TCP连接会关闭， 当客户端再次发送Request，需要重新建立TCP连接。

Host（发送请求时，该报头域是必需的）请求报头域主要用于指定被请求资源的Internet主机和端口号。如: 我们在浏览器中输入：<http://www.guet.edu.cn/index.html>。浏览器发送的请求消息中，就会包含Host请求报头域，如下：Host：http://www.guet.edu.cn此处使用缺省端口号80，若指定了端口号，则变成：Host：指定端口号

***Response头：***

Expires：浏览器会在指定过期时间内使用本地缓存

P3P：用于跨域设置Cookie, 这样可以解决iframe跨域访问cookie的问题

Set-Cookie：非常重要的header, 用于把cookie 发送到客户端浏览器， 每一个写入cookie都会生成一个Set-Cookie.

ETag：和If-None-Match 配合使用。

Last-Modified: 用于指示资源的最后修改日期和时间。

Content-Type：WEB服务器告诉浏览器自己响应的对象的类型和字符集, Content-Type: text/html; charset=utf-8

Content-Length：指明实体正文的长度

Content-Encoding: WEB服务器表明自己使用了什么压缩方法（gzip，deflate）压缩响应中的对象。

Server: 指明HTTP服务器的软件信息

Location：用于重定向一个新的位置, 包含新的URL地址

***KeepAlive实现原理：***

TCP 的 SO\_KEEPALIVE 确切的应该叫做 is\_keep\_alive?，它是一个探测服务，由内核完成的。当一端 recv(socket, ...) 时，如果对端一直没有消息返回，就会一直阻塞，本端根本不知道对端处于什么状态。所以本端可以周期发一个探测包，对端如果一定时间没有返回确认包，则认为对端已经断开连接， recv 会返回，不再继续阻塞。HTTP 的 keepalive 是双方通过 "Connection: keep-alive" 数据头来建立长连接，每次发送完请求不关闭套接字，而是继续下一次循环的 recv()，等待对端继续发送数据。如果设置了超时时间，那么 recv() 返回，关闭连接。TCP 的 SO\_KEEPALIVE，当网络突然中断时，用来及时探测对端断开，避免无限制阻塞 recv。HTTP 的 keepalive，是双方约定长连接、还是收完一次数据后立刻关闭套接字。

**tcp的keepalive**是侧重在保持客户端和服务端的连接，一方会不定期发送心跳包给另一方，当一方断掉的时候，没有断掉的定时发送几次心跳包，如果间隔发送几次，对方都返回的是RST，而不是ACK，那么就释放当前链接。

建立TCP连接时，就有定时器与之绑定，其中的一些定时器就用于处理keepalive过程。当keepalive定时器到0的时候，便会给对端发送一个不包含数据部分的keepalive探测包（probe packet），如果收到了keepalive探测包的回复消息，那就可以断定连接依然是OK的。如果我们没有收到对端keepalive探测包的回复消息，我们便可以断定连接已经不可用，进而采取一些措施。

KeepAlive并不是默认开启的，在Linux系统上没有一个全局的选项去开启TCP的KeepAlive。需要开启KeepAlive的应用必须在TCP的socket中单独开启。Linux Kernel有三个选项影响到KeepAlive的行为：

1.net.ipv4.tcpkeepaliveintvl = 75

2.net.ipv4.tcpkeepaliveprobes = 9

3.net.ipv4.tcpkeepalivetime = 7200

tcpkeepalivetime的单位是秒，表示TCP链接在多少秒之后没有数据报文传输启动探测报文; tcpkeepaliveintvl单位是也秒,表示前一个探测报文和后一个探测报文之间的时间间隔，tcpkeepaliveprobes表示探测的次数。

tcp的keepalive就是为了检测链接的可用性。主要调节的参数有三个：

* tcp\_keepalive\_time // 距离上次传送数据多少时间未收到判断为开始检测
* tcp\_keepalive\_intvl // 检测开始每多少时间发送心跳包
* tcp\_keepalive\_probes // 发送几次心跳包对方未响应则close连接

在客户端和服务端进行完三次握手之后，客户端和服务端都处在ESTABLISH状态，这个时候进行正常的PSH和ACK交互，但是一旦一方服务中断了，另一方在距离上次PSH时间tcp\_keepalive\_time发现对方未发送数据，则开始心跳检测。心跳检测实际就是发送一个PSH的空心跳包，这里说的空心跳包就是包的数据为空，但是TCP包的头部的数据和标识和正常包一样。如果这个包获取到的是RST返回的话，下面就会继续每隔tcp\_keepalive\_intval的时长发送一个空心跳包，如果tcp\_keepalive\_probes次心跳包对方都是返回RST而不是ACK，则心跳发起方就判断这个连接已经失效，主动CLOST这个连接。

[tcp层的keepalive会在两个场景下比较有用](http://www.tldp.org/HOWTO/html_single/TCP-Keepalive-HOWTO/)：

## 检测连接的一方是否断了

这里说的连接的一方是否断了包含几种情况：

* 连接一方服务中止
* 网络不好导致的服务长时间无响应
* 连接一方服务重启中

结合这三种方式就很好理解为什么会有 tcp\_keepalive\_time, tcp\_keepalive\_intval, tcp\_keepalive\_probes三种的设置了。如果是对方服务器进行重启的时候，我们不能根据一次的tcp返回重置信号就判定这个连接失效。相反的，重启之后，这个心跳包一旦正常，这个连接仍然可以继续使用。

## 防止因为长时间不用链接导致连接失效

这个往往在代理或者内网状况下会使用到。一般NAT网络为了资源，会和外网保持一定的资源连接数，而且采用的是淘汰机制，淘汰掉旧的，不用的连接，创建和使用新的连接。如果我们没有心跳检测机制，那么我们的连接在一段时间没有使用的时候，NAT对外的机制会判断对应的对外网络是无用的，淘汰掉旧的，即使这个时候客户端和服务端都还正常服务着，只是长时间未联络了而已。keepalive的机制由于有定时心跳包，自然就能解决这个问题了。

**http的keep-alive**是一旦客户端发送connection:keep-alive头给服务端，且服务端也接受这个keep-alive的话，两边对上暗号，这个连接就可以复用了，一个http处理完之后，另外一个http数据直接从这个连接走了。http层有个keep-alive, 它主要是用于客户端告诉服务端，这个连接我还会继续使用，在使用完之后不要关闭。这个设置首先会在性能上对客户端和服务器端性能上有一定的提升。很好理解的是少了TCP的三次握手和四次挥手，第二次传递数据就可以通过前一个连接直接进行数据交互了。当然会提升服务性能了。由于HTTP服务的发起方一般都是浏览器，即客户端。但是先执行完逻辑，传输完数据的一定是服务端。那么一旦没有keep-alive机制，服务端在传送完数据之后会率先发起连接断开的操作。由于TCP的四次挥手机制，先发起连接断开的一方会在连接断开之后进入到TIME\_WAIT的状态达到2MSL之久。

什么是 MIME TYPE？资源的媒体类型。 HTTP 协议，通过 Content-Type 来表示的，例如:Content-Type: text/HTML

一个HTTP请求报文由请求行（request line）、请求头（header）、空行和请求数据4个部分组成。

HTTP1.0定义了三种请求方法： GET, POST 和 HEAD方法。

HTTP1.1新增了五种请求方法：OPTIONS, PUT, DELETE, TRACE 和 CONNECT 方法。

GET请求：由于不同的浏览器对地址的字符限制也有所不同，一般最多只能识别2083个字符。

POST方式对传送的数据大小没有限制（限制在web服务器端，如tomcat默认为2m）

为什么不能用post代替所有的get：

1、多数浏览器对于POST采用两阶段发送数据的，先发送请求头，再发送请求体，即使参数再少再短，也会被分成两个步骤来发送（相对于GET），也就是第一步发送header数据，第二步再发送body部分。HTTP是应用层的协议，而在传输层有些情况TCP会出现两次连结的过程，HTTP协议本身不保存状态信息，一次请求一次响应。对于TCP而言，通信次数越多反而靠性越低，能在一次连结中传输完需要的消息是最可靠的，尽量使用GET请求来减少网络耗时。如果通信时间增加，这段时间客户端与服务器端一直保持连接状态，在服务器侧负载可能会增加，可靠性会下降。  
2、GET请求能够被cache，GET请求能够被保存在浏览器的浏览历史里面（密码等重要数据GET提交，别人查看历史记录，就可以直接看到这些私密数据）POST不进行缓存。  
3、GET参数是带在URL后面，传统IE中URL的最大可用长度为2048字符，其他浏览器对URL长度限制实现上有所不同。POST请求无长度限制（目前理论上是这样的）。  
4、GET提交的数据大小，不同浏览器的限制不同，一般在2k-8K之间，POST提交数据比较大，大小靠服务器的设定值限制。  
5、全部用POST不是十分合理，最好先把请求按功能和场景分下类，对数据请求频繁，数据不敏感且数据量在普通浏览器最小限定的2k范围内，这样的情况使用GET。其他地方使用POST。

## Yahoo高性能网站

使用CDN

为文件头指定Expires或Cache-Control：

对于静态内容：设置文件头过期时间Expires的值为“Never expire”（永不过期）

对于动态内容：使用恰当的Cache-Control文件头来帮助浏览器进行有条件的请求

Gzip压缩文件内容

从HTTP/1.1开始，web客户端都默认支持HTTP请求中有Accept-Encoding文件头的压缩格式：Accept-Encoding: gzip, deflate

Web服务器把压缩方式通过响应文件头中的Content-Encoding来返回给浏览器。

配置ETag

Etag是一个识别内容版本号的唯一字符串。增加ETag为实体的验证提供了一个比使用“last-modified date（上次编辑时间）”更加灵活的机制

尽早刷新输出缓冲

在PHP中，你可以使用flush()方法，它允许你把已经编译的好的部分HTML响应文件先发送给浏览器

使用GET来完成AJAX请求

当使用XMLHttpRequest时，浏览器中的POST方法是一个“两步走”的过程：首先发送文件头，然后才发送数据。因此使用GET最为恰当，因为它只需发送一个TCP包（除非你有很多cookie）。

## NAT（地址转换）

狭义上，NAT分为SNAT（原地址转换）和DNAT（目标地址转换）

NAP（p为port）T与NAT的区别在于，NAPT不仅转换IP包中的IP地址，还对IP包中TCP和UDP的Port进行转换。这使得多台私有网主机利用1个NAT公共IP就可以同时和公共网进行通信。（NAPT多了对TCP和UDP的端口号的转换）

DNAT，就是指数据包从网卡发送出去的时候，修改数据包中的目的IP，表现为如果你想访问A，可是因为网关做了DNAT，把所有访问A的数据包的目的IP全部修改为B，那么，你实际上访问的是B

ISP会把用户接入局域网，使得多个用户共享同一个公网IP，而每一个用户各分得一个局域网内网IP。而连接公网和局域网的这台路由器，称之为网关（gateway），NAT的过程就发生在这台网关路由器上。

在三层地址转换中，我们可以保证局域网内主机向公网发出的IP报文能顺利到达目的主机，但是从目的主机返回的IP报文却不能准确送至指定局域网主机（我们不能让网关把IP报文广播至全部局域网主机，因为这样必然会带来安全和性能问题）。为了解决这个问题，网关路由器需要借助传输层端口，通常情况下是TCP或UDP端口，由此来生成一张端口转换表。

让我们通过一个实例来说明端口转换表如何运作。  
假设局域网主机A192.168.1.100需要与公网上的目标主机B210.199.38.2:80进行一次TCP通信。其中A所在局域网的网关C的公网IP地址为210.177.63.2。步骤如下：

1. 局域网主机A192.168.1.100发出TCP连接请求，A上的TCP端口为系统分配的53600。该TCP握手包中，包含源地址和端口192.168.1.100:53600，目的地址和端口210.199.38.2:80。  
2. 网关C将该包的原地址和端口修改为210.177.63.2:63000，其中63000是网关分配的临时端口。  
3. 网关C在端口转换表中增加一条记录：

| **内网主机IP** | **内网主机端口** | **网关端口** | **目的主机IP** | **目的主机端口** |
| --- | --- | --- | --- | --- |
| 192.168.1.100 | 53600 | 63000 | 210.199.38.2 | 80 |

4. 网关C将修改后的TCP包发送至目的主机B。  
5. 目的主机B收到后，发送响应TCP包。该响应TCP包含有以下信息：源地址和端口210.199.38.2:80，目的地址和端口210.177.63.2:63000。  
6. 网关C收到这个来自B的响应包后，随即在端口转换表中查找记录。该记录须符合以下条件：目的主机IP==210.199.38.2，目的主机端口==80，网关端口==63000。  
7. 网关C搜索到这条记录，记录显示内网主机IP为192.168.1.100，内网主机端口为53600。  
8. 网关C将该包的目的地址和端口修改为192.168.1.100:53600。  
9. 网关C随即将该修改后的TCP包转发至192.168.1.100:53600，即局域网主机A。此时运输层数据的一次交换已完成。

***问题所在***

在网关C上，由于端口数量有限（0~65535），端口转换表的维护占用系统资源，因此不能无休止地向端口转换表中增加记录。对于过期的记录，网关需要将其删除。如何判断哪些是过期记录？网关认为，一段时间内无活动的连接是过期的，应定时检测转换表中的非活动连接，并将之丢弃。**而这个丢弃的过程，网关不会以任何的方式通告该连接的任何一端。**那么问题就来了：如果一个客户端应用程序由于业务需要，需要与服务端维持长连接（如TCP聊天程序），而如果在特别长的时间内（在博主的ISP环境下，该时间在3分钟左右），这个连接没有任何的数据交换，网关会认为这个连接过期并将这个连接从端口转换表中丢弃。该连接被丢弃时，客户端和服务端对此是完全无感知的。在连接被丢弃后，客户端将收不到服务端的数据推送，客户端发送的数据包也不能到达服务端。

## 路由器工作原理

路由表，NAT，TTL字段代表路由器跳数

ICMP

Ping

telnet

DNS

CDN

反向代理

RPC

Socket编程