计算机网络

[1. 应用层 1](#_147)

[1.1. HTTP 1](#_102)

[1.1.1. 请求/响应 1](#_106)

[1.1.2. HTTP1.1 1](#_107)

[1.1.3. HTTP2.0 1](#_108)

[1.1.4. 协议升级 2](#_118)

[1.1.5. 缓存 2](#_209)

[1.1.6. HTTPS查询过程 3](#_119)

[1.1.7. HTTP状态码 4](#_125)

[1.1.8. Session和Cookie 5](#_131)

[1.1.9. 8种方法 5](#_157)

[1.2. DNS劫持 6](#_168)

[1.2.1. 本机DNS劫持 6](#_169)

[1.2.2. 路由DNS劫持 6](#_170)

[1.2.3. 攻击DNS服务器：DDoS 6](#_171)

[1.3. 常见应用层协议 6](#_190)

[2. 奇怪的问题 6](#_184)

[2.1. 局域网中出现相同mac地址的主机 7](#_185)

[2.2. NAT 7](#_186)

[2.3. 路由算法 7](#_187)

[2.3.1. 静态路由 7](#_188)

[2.3.2. 动态路由 8](#_189)

[2.4. DNS负载均衡 8](#_207)

[3. 传输层 8](#_146)

[3.1. 适用场景 8](#_132)

[3.1.1. TCP：文件传输（准确性要求高） 8](#_133)

[3.1.2. UDP：即时通讯（准确性要求低） 8](#_134)

[3.2. TCP 8](#_135)

[3.2.1. 差错控制 9](#_136)

[3.2.2. 拥塞控制 9](#_137)

[3.2.3. 流量控制 10](#_138)

[3.2.4. 其他特性 10](#_142)

[3.2.5. ubuntu TCP调优 16](#_179)

[3.3. UDP 17](#_182)

[3.3.1. udp调用connect作用 17](#_208)

[3.4. QUIC(基于UDP) 17](#_183)

[4. 网络层（IP） 17](#_150)

[4.1. 子网划分 17](#_151)

[4.2. 基于ARP 17](#_152)

[4.3. 支持协议有ICMP,IGMP 17](#_153)

[4.4. VPN 17](#_154)

[4.5. NAT映射 18](#_155)

[4.6. 校验和仅覆盖IP首部（弱） 18](#_156)

[5. 体系结构 18](#_101)

[5.1. OSI七层 18](#_103)

[5.2. TCP/IP四层 18](#_104)

[5.3. 五层 18](#_105)

# 应用层

## HTTP

### 请求/响应

主要包括四部分：行、头、空行和体。

### HTTP1.1

#### 缓存控制

http1.1引入了更多的缓存控制策略

#### 错误通知管理

增加了更多的错误响应码，如410表示服务器上某个资源被永久删除。

#### 带宽优化

引入range头域，允许只请求资源的某个部分。

#### host头处理

http1.0中认为每台服务器绑定一个唯一的IP地址，但是随着虚拟主机的出现（一台物理服务器上可能存在多个虚拟主机，并且共享IP地址）。http1.1请求和响应都应支持host头域，且请求消息中没有host头域会报告错误。

#### 长连接

### HTTP2.0

#### 二进制分帧

#### header压缩

在客户端和服务端之间使用首部表来跟踪和存储之前发送的键值对。

对于相同的数据，不再重新通过每次请求和响应发送，每个新的首部键值对要么追加到当前表的末尾，要么替换表中之前的值。首部表在HTTP2.0的链接存续期内始终存在，由服务端和客户端共同渐进的更新。

#### TCP多路复用

单个连接上同时进行多个业务单元数据的传输

##### 然而丢包会导致严重的堵塞问题

#### Server推送

### 协议升级

使用Connection和Upgrade的组合。

Connection：Upgrade

Upgrade：protocol-name[/protocol-version]

### 缓存

#### Pragma

Pragma：是HTTP1.0标准中定义的一个header属性。请求（字段）中包含Pragma的效果跟在头信息中定义Cache-Control: no-cache相同，但是HTTP的响应头没有明确定义这个属性，所以它不能拿来完全替代HTTP/1.1中定义的Cache-control头。

#### Expires

Expires：是HTTP/1.0设置缓存过期时间的，Expires响应字段包含了资源过期的日期和时间（相对于服务器时间）。如果Cache-Control中设置了max-age或s-max-age，那么Expires头被忽略。如果该字段与Pragma同时存在时，Pragma起作用。

#### Cache-Control

HTTP/1.1，用来控制缓存行为。通过它提供的不同的值来定义缓存策略。请求头和响应头都支持这个属性。

**1. max-age**

指定设置缓存最大的有效时间，定义的是时间长短。当浏览器向服务器发送请求后，在max-age这段时间里浏览器就不会再向服务器发送请求了。

**2. s-maxage**

同max-age，只用于共享缓存（比如CDN缓存）。比如，当s-maxage=60时，在这60秒中，即使更新了CDN的内容，浏览器也不会进行请求。也就是说max-age用于普通缓存，而s-maxage用于代理缓存。

**3. public**

指定响应会被缓存，并且在多用户间（多浏览器间）共享。也就是下图的意思。如果没有指定public还是private，则默认为public。

**4. private**

响应只作为私有的缓存（见下图），不能在用户间共享。如果要求HTTP认证，响应会自动设置为 private

**5. no-cache**

指定不缓存响应，表明资源不进行缓存，但是设置了 no-cache 之后并不代表浏览器不缓存，而是在获取缓存前要向服务器确认资源是否被更改。因此有的时候只设置 no-cache 防止缓存还是不够保险，还可以加上 private 指令，将过期时间设为过去的时间。

**6. no-store**

绝对禁止缓存，一看就知道如果用了这个命令当然就是不会进行缓存啦～每次请求资源都要从服务器重新获取。

**7. must-revalidate**

指定如果页面是过期的，则去服务器进行获取。这个指令并不常用，就不做过多的讨论了。

https://juejin.im/entry/6844903897476120584

### HTTPS查询过程

#### 1. DNS解析

先查找本地缓存（浏览器缓存，操作系统缓存，hosts文件），如果未命中，则交给DNS服务器。

响应时间= DNS域名解析时间 + TCP 连接建立时间 + HTTP交易时间，为了让响应时间尽量小，只能缩小DNS解析的时间，所以采用UDP去解析。

##### 递归查询

##### 迭代查询

#### 2. 三次连接

#### 3. SSL/TLS握手

C->S: Client Hello（Random1，加密套件，SSL Version等）

S->C: Server Hello（Random2，确定加密组件）

S->C: 证书(服务器信息和使用CA私钥加密后的信息摘要）

S->C: Server Hello Done通知Server Hello结束

C->S: Certificate Verify验证证书合法性，生成Random3，非对称加密得到PreMaster Key

C->S: Client Key Exchange把PreMaster Key传给服务器（至此，两端都有了三个随机数，然后通过相同的算法得到同样的密钥）

C->S: Change Cipher Spec通知后续消息都会用密钥进行加密

C->S: Client Finish将前面的握手信息生成摘要并加密

S->C: 通知客户端后续消息使用协商的密钥加密

S->C: Server Finish将握手信息生成摘要并加密，发送

PreMaster的存在是由于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pre master secret就有可能被猜出来，那么仅适用pre master secret作为密钥就不合适了，因此必须引入新的随机因素，那么客户端和服务器加上pre master secret三个随机数一同生成的密钥就不容易被猜出了，一个伪随机可能完全不随机，可是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。

#### 4. 得到资源，开始载入渲染

1. 发现css以及js文件，发起请求

2. 渲染，发现有图片就请求，但继续渲染

3. 发现script标签，执行

4. 图片下载完，重新渲染部分区域

### HTTP状态码

#### 1\*\*

表示服务器收到信息，需要请求者继续操作

如100表示目前为止都正常。

#### 2\*\*

操作被成功接收并处理，如200成功响应。

#### 3\*\*

重定向

如301表示永久性重定向，302表示临时重定向。

#### 4\*\*

客户端错误

400表示错误请求，404表示请求的资源不存在。

#### 5\*\*

服务端错误，500表示服务器执行错误。

### Session和Cookie

HTTP是无状态协议，但是有些请求需要状态，比如会话等。

当用户登录后，会在服务端建立一个会话Session，用来标识这个特定的用户。第一次创建Session的时候，服务端会在HTTP协议中告诉客户端，需要在 Cookie 里面记录一个Session ID，以后每次请求把这个会话ID发送到服务器。

### 8种方法

#### GET

获取资源，是幂等操作。

#### POST

会向请求资源提交数据，请求服务器进行处理，如表单数据提交、文件上传等。POST是非幂等方法，这个请求可能会创建新的资源或修改现有资源。

#### HEAD

同GET方法，都是向服务器发出指定资源请求，但是服务器在响应HEAD请求时不会回传资源的内容部分。这样就可以在不传输全部内容的情况下，获取服务器的响应头信息。

#### PUT

会向指定资源位置上传最新内容，通过该方法客户端可以将指定资源的最新数据发送给服务端以取代旧数据。

#### PATCH

PATCH和PUT请求类似，但是有两点不同：

PATCH一般用于资源的部分更新，而PUT一般用于资源的整体更新

当资源不存在时，PATCH会创建一个新的资源，而PUT只会对存在的资源进行更新。

#### DELETE

用于请求服务器删除所请求URI所标识的资源。

这个方法也是幂等的，DELETE请求后指定资源会被删除。

#### OPTIONS

OPTIONS请求与HEAD类似，一般也是用于客户端查看服务器的性能。 这个方法会请求服务器返回该资源所支持的所有HTTP请求方法，该方法会用'\*'来代替资源名称，向服务器发送OPTIONS请求，可以测试服务器功能是否正常。JavaScript的XMLHttpRequest对象进行CORS跨域资源共享时，就是使用OPTIONS方法发送嗅探请求，以判断是否有对指定资源的访问权限

#### CONNECT

CONNECT方法是HTTP/1.1协议预留的，能够将连接改为管道方式的代理服务器。通常用于SSL加密服务器的链接与非加密的HTTP代理服务器的通信。

#### TRACE

TRACE请求服务器回显其收到的请求信息，该方法主要用于HTTP请求的测试或诊断。

## DNS劫持

### 本机DNS劫持

### 路由DNS劫持

### 攻击DNS服务器：DDoS

## 常见应用层协议

HTTP：TCP 80

HTTPS：TCP 443

DHCP：UDP 67/68

FTP：TCP 20/21

DNS：UDP/TCP 53

TELNET：TCP 23

# 奇怪的问题

## 局域网中出现相同mac地址的主机

前提条件：A(192.168.30.1), B(192.168.30.2), C(192.168.30.3)

A和B的mac地址相同

1. A和B之间不能互相访问。

2. C在访问A或B的时候，会产生大量随机性障碍

3. A或B访问C的时候都不受影响。

4. A或B都可以通过路由器上互联网（有点怀疑是NAT）。

5. A和B都跟C ping，C也可以ping通A和B。

其实路由器的本质是记录请求的主机的端口，在收到Web服务器响应的时候根据这个记录发送到指定的主机。

由于TCP重传的机制，所以还是能传输的。

## NAT

NAT实现方式有三种：

静态转换：将内部网络的私有IP地址转换为公有IP地址（一对一映射）。某个私有IP地址只能转换为某个公有IP，类似于负载均衡器。

动态转换：将内部网络的私有IP地址转化为公有IP地址，但是转化到什么公有IP是随机的。只要指定哪些内部地址可以进行转换，以及哪些合法地址可以作为外部地址时，就可以进行动态转换（也就是可以使用多个合法外部地址集）

端口多路复用：改变外出数据包的源端口并进行端口转换，即端口映射。内部网络所有主机均可共享一个外部IP地址实现对Internet的访问。

## 路由算法

### 静态路由

1. 洪泛

节点收到一个报文分组后，向所有可能的方向复制转发。保证了只要有一条信道能走通，那么就报文就可以到达。

2. 随机走动法

节点收到分组后，向所有与之相邻的节点中为分组随机选择一个节点转发出去。虽然简单，但不是最佳路由，效率低下。

3. 最短路径法

Dijkstra算法

### 动态路由

1. 距离矢量路由算法

每个路由器都要维护一张矢量表，这个矢量表中的每一行都记录了从当前位置能到达的最佳出口（接口）和距离（跳数）。

每隔一段时间路由器会向所有邻居节点发送自己的这个表，同时它也会接收每个邻居发来的它们的表，并进行对比更新。对比更新过程为路由器X离路由器Y的距离是m，如果收到邻居Y发来的“邻居Y距离路由器Z的距离是n”，那么可得到：路由器X到路由器Z的距离为m+n。

继续类推，要不了多久，每个路由器就可以将网络中所有路由节点和子网线路都汇聚起来。

优点：非常简单清晰，且任何加入到网络中的新节点都能很快的与其它节点建立起联系获得补充信息。

缺点：首先就是每次发送信息的时候，要发送整个全局路由表，太大了，因为每个路由器需要在矢量表中记录下整个网络的信息，导致需要较大存储、CPU、网络开销，对资源的要求越来越高。还有一个问题就是收敛时间太慢，也就是路由器共享路由信息并使各台路由器掌握的网络情况达到一致所需的时间比较久，收敛速度慢会导致有些路由器的表更新慢，从而造成路由环路的问题

2. 链路状态路由

基于Dijkstra算法。

采用动态拼接地图的方式，每一个路由器首先会发现自己身边的邻居节点，然后将自己与邻居节点之间的链路状态包广播出去。别的路由器收到后，会进行拼装，从而形成全局网络视图。

## DNS负载均衡

DNS将域名解析为一组服务器，然后轮询，返回各服务器的IP

实际场景下，这一组服务器都是负载均衡服务器。

# 传输层

## 适用场景

### TCP：文件传输（准确性要求高）

### UDP：即时通讯（准确性要求低）

## TCP

### 差错控制

#### 校验和

覆盖TCP首部和数据部分，比较弱。

#### 超时

#### 确认

### 拥塞控制

#### CUBIC

经典的TCP拥塞控制算法，分为下列四部分：

慢开始：2倍增加

拥塞控制：线性增加

快重传：3个连续ACK

快恢复：门限减半，窗口设置为门限。

#### BBR

##### 最优工作点

1. 当网络中数据包不多，没有填满链路带宽时，随着网络中数据包增加，有效带宽不断上升，往返时延不发生变化。当数据包刚好填满管道，此时达到最优工作点，带宽时延积BDP=bestBw \* rtt。

2. 继续增加网络中的数据包，超出BDP，此时部分数据包需要排队，rtt增加。

3. 继续增加数据包，缓冲区被填满，从而造成丢包。

##### 基本思想

不考虑丢包，估计最优工作点。

交替测量带宽和延迟，用一段时间内的带宽极大值和延迟极小值作为估计值。

##### 具体过程

1. 启动：当连接建立时，指数增加发送速率，经过3次有效带宽不增长，开始进入下一阶段。延迟最小值就是初始估计值，带宽最大值则时启动结束的估计值。

2. 排空：指数降低发送速率，延迟慢慢降低，当延迟不再降低时，结束。

3. 带宽探测：会以至少8个往返时延为周期

在第1个往返时间，BBR尝试以估计带宽的5/4发送，如果有效带宽增加了，那么继续增加。

在下一个往返时间里，需要排空，会以3/4的速率发送，如果延迟有下降，就把3/4带宽作为新的估计带宽。

剩下6个往返时间，BBR使用估计的带宽发包。

如果带宽探测阶段连续10秒估计延时都没有发现更小的，那么就进入延迟探测，在200ms内发送4个包，以这段时间测得的最小延迟作为新的延迟估计。

(1)和(2)体现的思想是：激进加，保守减。

4. 延迟探测：如果带宽探测阶段连续10秒估计延时都没有发现更小的，那么就进入延迟探测，在200ms内发送4个包，以这段时间测得的最小延迟作为新的延迟估计。

##### 问题

对带宽不敏感，竞争不过CUBIC。

如果网络上同时存在BBR和CUBIC，那么永远也排空不了链路缓冲区。

### 流量控制

主要通过接收窗口控制。

### 其他特性

#### RTT计算：时间戳字段

可以使用时间戳字段更加精确地计算RTT。

首先RTT的计算肯定要动态变化，基本公式如下：

RTTs = （1 -a）\*旧RTTs +a \* 新RTT

#### RTO动态变化

1. 第一次测量时RTTd为RTT的一半，后续RTTd = （1- b）\* 旧RTTd + b \* | RTTs - 新RTT|

2. RTO = RTTs + 4 \* RTTd

#### 接收窗口放缩

#### MSS分片

MSS存在的意义主要是IP分片丢失的代价太高，需要重传整个IP包。

通过MSS，能够将重传的代价变小。

#### 发送报文时机

1. 缓冲区数据达到MSS

2. 由发送端的应用程序指明要马上发送，即TCP推送

3. 发送计时器达到期限

#### TIME\_WAIT

保证全双工的TCP连接正常终止。

保证网络中迷失的数据包正常过期。

#### 粘包/拆包

**拆包**

应用程序写入的数据大于套接字缓冲区大小

写入数据大于MSS

**粘包**

应用程序多次写入的数据都比较小，将会合成一个报文发送

接收端不及时读，也会发生粘包。

#### 标志位

##### RST报文

触发时机：

connect一个不存在的端口，服务器端会回复一个RST，客户端收到后，会显示Connection refused

向一个已经崩溃的对端发送数据，会受到对端发来的RST

close(sockfd)时，直接丢弃缓冲区中未读取的数据，并给对方发送RST。

A重启，收到B的保活探针，A发RST，通知B。

##### 推送PSH

两个进程在进行交互式通信时，一个进程键入一个命令希望另一个进程立即收到该进程的响应，将PSH置为1，TCP使用推送操作，发送方发送一个报文段，接收方收到TCP推送的报文段时，立即向前交付接受应用程序，不等整个缓冲区满了才向上交付。如SSH过程等

##### 紧急URG

当 URG = 1 时，发送应用进程就告诉发送方 TCP 有紧急数据要传送，于是发送方 TCP 就把紧急数据插入到报文段的最前面，而紧急数据后面的数据仍然是普通数据。紧急指针（Urgent Pointer）就是表示本报文段中紧急数据的字节数（紧急数据结束后就是普通数据），即紧急数据末尾在报文段中的位置。当所有紧急数据都处理完，TCP 就告诉应用层序恢复到正常操作。值得注意的是，即使窗口为 0，也可发紧急数据。

#### 三次握手

建立一个TCP连接时，需要客户端和服务器总共发送3个包。进行三次握手的主要作用是为了确认双方的接收能力和发送能力是否正常，指定自己的初始序列号为后面的可靠传输做准备。

实质上是连接服务器指定端口，建立TCP连接，并同步连接双方的序列号和确认号，交换TCP窗口大小信息。

1. 防止过期的SYN包到达服务端，使服务端误认为新连接。

2. 确认双方的接收和发送能力。

3. 确认序列号和对方的接收窗口等。

##### 两次握手

只有通过三次握手才能确认双发的发送和接收能力是正常的。

当使用两次握手时，会出现以下情况：

如客户端发出连接请求，但因连接请求报文丢失而未收到确认，于是客户端再重传一次连接请求。后来收到了确认，建立了连接。数据传输完毕后，就释放了连接，客户端共发出了两个连接请求报文段，其中第一个丢失，第二个到达了服务端，但是第一个丢失的报文段只是在某些网络结点长时间滞留了，延误到连接释放以后的某个时间才到达服务端，此时服务端误认为客户端又发出一次新的连接请求，于是就向客户端发出确认报文段，同意建立连接，不采用三次握手，只要服务端发出确认，就建立新的连接了，此时客户端忽略服务端发来的确认，也不发送数据，则服务端一直等待客户端发送数据，浪费资源。

##### 半连接队列

**半连接队列**

服务器第一次收到客户端的SYN之后，就会处于SYN\_RCVD状态，此时双方还未建立连接，服务器会把此种状态下的请求连接放在一个队列里。

关于SYN-ACK 重传次数的问题：服务器发送完SYN-ACK包，如果未收到客户确认包，服务器进行首次重传，等待一段时间仍未收到客户确认包，进行第二次重传。如果重传次数超过系统规定的最大重传次数，系统将该连接信息从半连接队列中删除。注意，每次重传等待的时间不一定相同，一般会是指数增长，例如间隔时间为 200ms, 400ms，800ms...

再细致一点，半连接队列又可称为syns queue（即当服务端处于syn\_recv状态时)

##### 全连接队列

已经完成三次握手，建立起连接的就会放在全连接队列中。如果队列满了就有可能会出现丢包现象。

简单来说TCP三次握手后有个accept队列，进到这个队列才能从Listen变成accept，默认backlog 值是50，很容易就满了。满了之后握手第三步的时候server会按照tcp\_abort\_on\_overflow参数采取相应的措施。

tcp\_abort\_on\_overflow:

0：表示三次握手的第三步时，如果全连接队列满了，那么server扔掉client发过来的ack。

1：表示全连接满的时候就发送一个reset包给client。

##### ISN(Initial Sequence Number)

三次握手的其中一个重要功能是客户端和服务端交换 ISN(Initial Sequence Number)，以便让对方知道接下来接收数据的时候如何按序列号组装数据。如果 ISN 是固定的，攻击者很容易猜出后续的确认号，因此 ISN 是动态生成的。

##### 携带数据

第一次握手不可以放数据，其中一个简单的原因就是会让服务器更加容易受到攻击了。而对于第三次的话，此时客户端已经处于 ESTABLISHED 状态。对于客户端来说，他已经建立起连接了，并且也已经知道服务器的接收、发送能力是正常的了，所以能携带数据也没啥毛病。

#### 攻击

##### SYN攻击

SYN攻击属于DOS攻击的一种，它利用TCP协议缺陷，通过发送大量的半连接请求，耗费CPU和内存资源。SYN攻击除了能影响主机外，还可以危害路由器、防火墙等网络系统，事实上SYN攻击并不管目标是什么系统，只要这些系统打开TCP服务就可以实施。从上图可看到，服务器接收到连接请求（syn=j），将此信息加入未连接队列，并发送请求包给客户（syn=k,ack=j+1），此时进入SYN\_RECV状态。当服务器未收到客户端的确认包时，重发请求包，一直到超时，才将此条目从未连接队列删除。配合IP欺骗，SYN攻击能达到很好的效果，通常，客户端在短时间内伪造大量不存在的IP地址，向服务器不断地发送syn包，服务器回复确认包，并等待客户的确认，由于源地址是不存在的，服务器需要不断的重发直至超时，这些伪造的SYN包将长时间占用未连接队列，正常的SYN请求被丢弃，目标系统运行缓慢，严重者引起网络堵塞甚至系统瘫痪。

###### 过滤网关防护

过滤网关主要指防火墙（当然路由器也可以）：

**1. 网关超时设置**

防火墙设置SYN转发超时参数（状态检测的防火墙可在状态表里面设置），该参数远小于服务器的timeout时间。当客户端发送完SYN包，服务端发送确认包后（SYN＋ACK），防火墙如果在计数器到期时还未收到客户端的确认包（ACK），则往服务器发送RST包，以使服务器从队列中删去该半连接。值得注意的是，网关超时参数设置不宜过小也不宜过大，超时参数设置过小会影响正常的通讯，设置太大，又会影响防范SYN攻击的效果，必须根据所处的网络应用环境来设置此参数。

**2. SYN网关**

SYN网关收到客户端的SYN包时，直接转发给服务器；SYN网关收到服务器的SYN/ACK包后，将该包转发给客户端，同时以客户端的名义给服务器发ACK确认包。此时服务器由半连接状态进入连接状态。当客户端确认包到达时，如果有数据则转发，否则丢弃。事实上，服务器除了维持半连接队列外，还要有一个连接队列，如果发生SYN攻击时，将使连接队列数目增加，但一般服务器所能承受的连接数量比半连接数量大得多，所以这种方法能有效地减轻对服务器的攻击。 （这种方法一般般）

**3. SYN代理**

当客户端SYN包到达过滤网关时，SYN代理并不转发SYN包，而是以服务器的名义主动回复SYN/ACK包给客户，如果收到客户的ACK包，表明这是正常的访问，此时防火墙向服务器发送ACK包并完成三次握手。SYN代理事实上代替了服务器去处理SYN攻击，此时要求过滤网关自身具有很强的防范SYN攻击能力

###### 加固TCP/IP协议栈

防范SYN攻击的另一项主要技术是调整tcp/ip协议栈，修改tcp协议实现。主要方法有SynAttackProtect保护机制、SYN cookies技术、增加最大半连接和缩短超时时间等。tcp/ip协议栈的调整可能会引起某些功能的受限，管理员应该在进行充分了解和测试的前提下进行此项工作。

**1. SynAttackProtect机制**

SynAttackProtect机制是通过关闭某些socket选项，增加额外的连接指示和减少超时时间，使系统能处理更多的SYN连接，以达到防范SYN攻击的目的。

**2. SYN cookies技术**

在TCP实现中，当收到客户端的SYN请求时，服务器需要回复SYN＋ACK包给客户端，客户端也要发送确认包给服务器。通常，服务器的初始序列号由服务器按照一定的规律计算得到或采用随机数，但在SYN cookies中，服务器的初始序列号是通过对客户端IP地址、客户端端囗、服务器IP地址和服务器端囗以及其他一些安全数值等要素进行hash运算，加密得到的，称之为cookie。当服务器遭受SYN攻击使得backlog队列满时，服务器并不拒绝新的SYN请求，而是回复cookie（回复包的SYN序列号）给客户端， 如果收到客户端的ACK包，服务器将客户端的ACK序列号减去1得到cookie比较值，并将上述要素进行一次hash运算，看看是否等于此cookie。如果相等，直接完成三次握手（注意：此时并不用查看此连接是否属于backlog队列）。

##### 子主题

### ubuntu TCP调优

/proc/sys/net目录下的所有文件都对应着各自的TCP参数。

**有几个重要的参数**：

tcp\_keepalive\_time：7200（即2个小时）

tcp\_tw\_reuse：0（即不允许将TIME\_WAIT状态的socket用于新的tcp连接）

tcp\_fin\_timeout：60（对于本端断开的socket连接，TCP保持在FIN\_WAIT2状态的时间)

tcp\_timeout\_close\_wait：60（对于被动断开，维持在ClOSE\_WAIT的时间）

tcp\_timeout\_time\_wait：120（对于主动断开的socket连接，维持在TIME\_WAIT状态的时间）

**关于重传**：

tcp\_syn\_retries：负责建立连接时候的重传，默认为6.

tcp\_retries1：当重传超时后，将更新路由缓存，默认3.

tcp\_retries2：当重传超时后，会关闭TCP流，默认15.

注意：各个参数的值并不是重传次数，而是根据rto\_base（初始RTO，为200ms）和参数值，计算得到timeout，超过这个timeout，则就认为超过了阈值。另外，关于重传时间具体增加细节为，前9次呈2次方增加，后面保持在120s。

总的来说，在Linux3.10中，如果tcp\_retres2设置为15。总重传超时周期应该在如下范围内[924.6s, 1044.6s)

可通过vim /etc/sysctl.conf更改参数。

## UDP

### udp调用connect作用

1. udp调用connect并不会触发三次握手，仅仅是把对端的ip&port记录下来。

2. udp可以多次调用connect，用于指定新连接或是断开之前的连接。

3. udp使用connect可以提高效率，不使用connect每次使用sendTo，将先连接，发送数据包，再断开。而使用了connect，则无需重复连接和断开连接的步骤，并且可以使用send()/revc()，read()/write();

4. UDP中使用connect的好处:1:会提升效率.前面已经描述了.2:高并发服务中会增加系统稳定性.原因:假设client A 通过非connect的UDP与server B,C通信.B,C提供相同服务.为了负载均衡,我们让A与B,C交替通信.A与 B通信IPa:PORTa <----> IPb:PORTbA与 C通信IPa:PORTa' <---->IPc:PORTc

假设PORTa与 PORTa'相同了(在大并发情况下会发生这种情况),那么就有可能出现A等待B的报文,却收到了C的报文.导致收报错误.解决方法内就是采用connect的UDP通信方式.在A中创建两个udp,然后分别connect到B,C.

## QUIC(基于UDP)

QUIC的主要机制：

自定义连接机制：QUIC使用一个64位的随机数作为ID来标识。

自定义重传机制：通过递增的序列号加上stream offset来解决。

通过使用QUIC，可以解决HTTP2.0的多路复用问题。

# 网络层（IP）

## 子网划分

主要有两种划分方式：

1. A、B、C类网。

2. 根据前缀划分。

## 基于ARP

## 支持协议有ICMP,IGMP

## VPN

## NAT映射

## 校验和仅覆盖IP首部（弱）

# 体系结构

## OSI七层

## TCP/IP四层

## 五层