第1章

第 1.1 章 课后习题

端到端有效吞吐率 = 实际传输大小/传输时间

实际传输时间 = RTT+ 传输大小/信道带宽

RTT：请求与回答时间

问题：信道带宽1Gbps，端到端时延τ= 10ms， TCP发送窗口65535字节。问可

能达到的最大吞吐率T ？信道利用率ρ？

解析： T= size/(2τ+size/BW)

= size\*BW/(2τ\*BW+size) bps

= 65535\*8\*109/(20\*109\*10-3s+65535\*8)bps

= 524280\*103/(20 \*106 +524280)Mbps

= 25.5 Mbps， ρ=25.5M/1000M=2.55%

**1. 一局域网最大距离为2 KM，对于100 BYTE的数据包，在带宽为多少时，传播时延等于传输时延？512 BYT的数据包？(速度为2\*108 m/s)**

解答：传播延迟等于：

2×103 米÷（2×108 米/秒）=10-5 秒=10 微秒

100 字节÷10 微秒=10M 字节/秒=80M 位/秒

512 字节÷10 微秒=51.2M 字节/秒=409.6M 位/秒

因此，带宽应分别等于80M 位/秒和409.6M 位/秒

1. **CIDR 地址的利用率分析**

划分前网络掩码为m位数，可划分地址位数：a+b= 32- m ；

a 是划分后的掩码增加位数；b 是划分后的主机位数；

划分前的主机地址数=2 ^a+b -2（减去全0 0 全1 1 网络号）

划分后的主机地址数 = 新子网×新子网中的主机数=(2 ^a)×( 2^b - 2)

 对B 类地址 ， 设 a=6 ,b= 10 ；则结果是多少 ？

 对B类地址 ， a+b= 16; 对C 类地址， a+b=8 ；

1. **假设在地球和月球间建立一点对点 100MbPS 的连接，地月距离为 385 ， 000KM ，数据传送速率为光速 ----3\*10E8 M/S**

（a） 计算该链路的最小RTT。

解：最小RTT 等于2×385000000 米÷（3×10 8 米/秒）=2.57 秒

（b） 使用RTT 作为延迟，计算该链路的“延迟×带宽”值。

解：“延迟×带宽”值等于2.57 秒×100M 位/秒=257M 位≈32M 字节

（c） 在（b）中计算的“延迟×带宽”值的含义是什么？ 解：它表示发送方在收到一个响应之前能够发送的数据量。

（d） 在月亮上用一个照相机拍取地球的相片，并把它们以数字形式保存到磁盘上。 假定在地球上的任务控制要下载25M 字节的最新图象，那么，从发出数据请求 到传送结束最少要花多少时间？

解：在图象可以开始到达地面之前，至少需要一个RTT。假定仅有带宽延迟，那么发送需要 的时间等于25M 字节÷100M 位/秒=200M 位÷100M 位/秒=2 秒。所以，直到最后一个图象位 到达地球，总共化的时间等于2.0+2.57=4.57 秒。

比特率Bit Rate： bps = bit/s

– 每秒内传输的比特数。

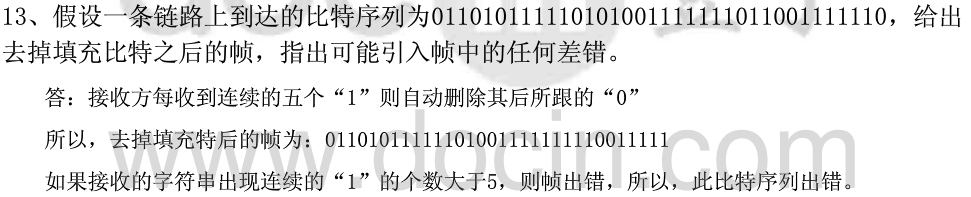
波德率Baud Rate:Buad

– 每秒内为表示某些比特而需要的信号单元数（或码元数）

当仅当一个信号单元表示一比特时，比特率才等于波德率。

Bit Rate = log2(Bit Units)×Baud Rate

**第 1.2 章 课后习题 第3题查48页表 第7题如下：**



第 1.3 章 课后习题

**第二章25. 假设你为卫星站的一个1Mbps点到点链路设计一个滑动窗口协议，卫星在3\*10^4KM的高度绕地球旋转。假设每帧携带1KB数据，在下述情况下，最少需要多少 比特？假设光速为3\*10^8m/s. (a)RWS=1 (b)RWS=SWS**

解： 单程的传播延时为3\*10^7/(3\*10^8)=0.1s.带宽为1Mbps，每帧携带1KB的数据，所以每秒可以传输1000000/（8\*1000）=125组，所以发送窗口大小为带宽\*往返延迟=125\*0.1\*2=25个分组。

(a) 当RWS=1时，必需的序号空间MaxSeqNum>=SWS+1=26,所以至少26个序号，需要5比特

(b) 当RWS=SWS时，SWS<( MaxSeqNum+1)/2,所以到50个序号，需要6比特

**第二章40. 当在以太网上的两个主机共享一个硬件地址时会出现何种类型的问题？描述会发生什么情况以及为什么那是一个问题？**

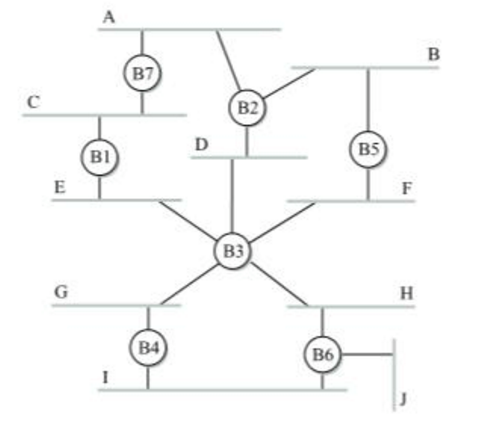
在以太网中，一个主机要和另一个主机进行直接通信，必须要知道目标主机的MAC地址。但这个目标MAC地址是如何获得的呢？它就是通过地址解析协议获得的。ARP协议用于将网络中的IP地址解析为的硬件地址（MAC地址），以保证通信的顺利进行。会出现MAC地址冲突导致网络使用异常

第三章

**13.指出下列哪些端口不会被生成树算法所选择。**

**B4,B5,B7不会被选择。**

**14. 对图中给出的扩展LAN，假设网桥B1遭遇灾难性的故障。指出当经过恢复过程并形成新树后，哪些端口不会被生成树算法所选择。（下图所示）**



答:因为B1节点故障，则B2节点为根,可以连接A、B、D；B3节点可以连接E、F、G、H；B4节点可以连接I；B5节点可以连接B、F但是不会被选择；B6节点可以连接J；B7节点则被用来连接C。

**第四章4. 假设一个TCP消息包包含2048字节数据和20字节的TCP首部，这个TCP 消息要传送给IP，传送通过因特网上的两个网络。第一个网络使用14字节的首部，MTU是1024，第二个网络使用8字节的首部，MTU是512。请给出传送到目的主机网络层的分段序列的尺寸和偏移量。假定所有IP首部的尺寸是20字节。**

先考虑第一个网络：1024-14-20=990字节，因为分段传输数据必须为8的倍数，所以每个分段可传输8 × ⌊990/8⌋ = 984 字节，共需传输2048+20=2068字节数据，则有984+984+100=2068,

对于第二个网络，512-8-20=484，每个分段最多可传输8 × ⌊484/8⌋ = 480 字节即将984字节分为480+480+24，而第一个网络中的100字节不需要分段

第四章5，

**第四章21 答案13545、22方法相同：**

目的地址和子网掩码相与，等于子网号则取下一跳，不等则取默认下一跳值。

**第四章40 一个单位有一个C类网络200.1.1。考虑到共有四个部门，准备划分子网。这四个部门内的主机数目分别是：A—72台， B—35台， C—20台， D—18台； 即共有145台主机。**

**（a） 给出一种可能的子网掩码安排来完成划分任务**

解答：每个部门分配一个子网，名义上部门A、B、C、D的子网大小分别是： 27（=128），26（=64），25（=32）和25（=32）

IP地址的最高位是0表示子网A，最高两位是10表示子网B，最高三位是110表示子网C，最高三位是111表示子网D。显然这里采用了可变长子网掩码，涉及3种子网掩码，分别是 255.255.255.128 255.255.255.192 255.255.255.224

C类网络默认子网掩码255.255.255.0

2^N-2>=72 N最小值是7 那么子网掩码就是255.255.255.128

主机地址就 200.1.1.1到200.1.1.127 同样

B部门35台，255.255.255.192. 主机地址 200.1.1.129到191

C部门20台，255.255.255.224 主机地址 200.1.1.193到200.1.1.223

D部门18台，255.255.255.224. 主机地址 200.1.1.225到200.1.1.254

**（b） 如果部门D的主机数目增长到34台，那么该单位又该怎么做？**

解答： 给部门A分配两个子网01和001，名义上分别是64个地址和32个地址，共96个地址；

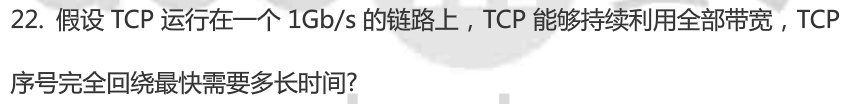
部门B不变，仍然是10，名义上大小为64个地址；

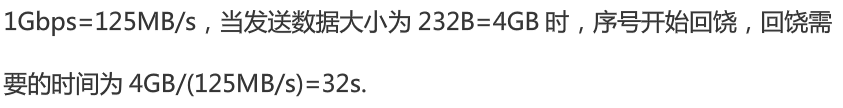
部门C改为000，名义上大小是32个地址

部门D改为11，名义上大小是64个地址。

**第四章 46 答案：FBEADC**

**第五章 12、13题如下图 第二问：t\*1000？**



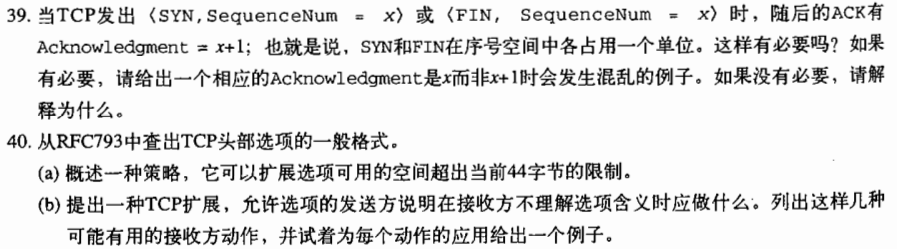


如果存在一个速率为1Gbps的网络，TCP发送端1秒会发送125MB的数据，32位的序列号空间能传输2的32次方个字节，也就是说32秒左右就会发生回绕，我们知道这个值远小于MSL值，因此会发生的。

**第二问：如果在上面算出的回绕时间内，一个附加的32位时间戳字段增量1000次。这个时段回绕需要多长时间？**

上问回绕需要时间为32秒，则增量一次需要32s/1000=32ms，则需要32×4×109ms

**第五章**



**39.** Ack号码的递增对于FIN来说必不可少，因为FIN的发送者可由此确定前面的数据和FIN都已经被接收。对于一个SYN消息，所有后续数据的确认号 ACK会逐步递增，这样ACK也隐式确认接收了SYN（连接建立后才能确认数据被接收）。因此，递增的序列号更是出于一种约定和一致性的考虑而非设计上的必要**。**

40.

(a)有一种方法是让TCP数据的前n个字节代表某几个具体的选项。

(b) 当一个TCP端点接收到一个未知的选项

关闭/中止连接。如果不知道选项的话，连接已经失去意义。

忽略选项但保留TCP数据。这是目前RFC 1122要求。

回信“我不明白”。这仅仅是对之前的响应的一种显式表达。一种改进方法是发送回主机可以理解的类似选项列表的消息。

丢弃伴随的TCP数据。当数据段被加密，或者数据段在选项指定的格式中时可能会用到。对于这个有意义的序列号的理解是必要的；如果整个TCP数据段是一个扩展选项块，那么序列号完全不应该被增加。

丢弃前N个字节的TCP数据。这是之前的策略的一种延伸，用来处理当TCP数据的前n个字节被指定为扩展选项块的情况。当接收者理解这n个字节的意义，却不明白它所代表的选项时，就会有点迷茫了。

第2章

**流控与拥控之差别:**

**–流控:防止发送超过接收者能力（速/量） , 是端到端的发送**

**–拥控:防止过多数据注入到网络中,从而引起交换机或链路超**

**载,拥控是关于主机到网络的发送**

**1. TCP拥塞控制原理。**

答： TCP的拥塞控制由4个核心算法组成：慢启动、拥塞避免、快速重传、快速恢复。发送方维持一个叫做拥塞窗口cwnd的状态变量。拥塞窗口的大小取决于网络的拥塞程度，并且动态地在变化。发送方让自己的发送窗口等于拥塞窗口，另外考虑到接受方的接收能力，发送窗口可能小于拥塞窗口。

慢开始算法的思路就是，不要一开始就发送大量的数据，先探测一下网络的拥塞程度，也就是说由小到大逐渐增加拥塞窗口的大小。当新建连接时，cwnd初始化为1个最大报文段(MSS)大小，发送端开始按照拥塞窗口大小发送数据，每当有一个报文段被确认，cwnd就增加1个MSS大小。这样cwnd的值就随着网络往返时间 (Round Trip Time,RTT)呈指数级增长。

为了防止cwnd增长过大引起网络拥塞，还需设置一个慢开始门限ssthresh状态变量。ssthresh的用法如下：

当cwnd<ssthresh时，使用慢开始算法。

当cwnd>ssthresh时，改用拥塞避免算法。

当cwnd=ssthresh时，慢开始与拥塞避免算法任意。

拥塞避免算法让拥塞窗口缓慢增长，即每经过一个往返时间RTT就把发送方的拥塞窗口cwnd加1，而不是加倍。这样拥塞窗口按线性规律缓慢增长。无论是在慢开始阶段还是在拥塞避免阶段，只要发送方判断网络出现拥塞（其根据就是没有收到确认，虽然没有收到确认可能是其他原因的分组丢失，但是因为无法判定，所以都当做拥塞来处理），就把慢开始门限设置为出现拥塞时的发送窗口大小的一半。然后把拥塞窗口设置为1，执行慢开始算法。

快重传要求接收方在收到一个失序的报文段后就立即发出重复确认（为的是使发送方及早知道有报文段没有到达对方）而不要等到自己发送数据时捎带确认。快重传算法规定，发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段，而不必继续等待设置的重传计时器时间到期。

快重传配合使用的还有快恢复算法，有以下两个要点:

①当发送方连续收到三个重复确认时，就执行“乘法减小”算法，把ssthresh门限减半。但是接下去并不执行慢开始算法。

②考虑到如果网络出现拥塞的话就不会收到好几个重复的确认，所以发送方现在认为网络可能没有出现拥塞。所以此时不执行慢开始算法，而是将cwnd设置为ssthresh的大小，然后执行拥塞避免算法。

**2. RED主动队列管理。**

以上的拥塞避免算法并没有和网络层联系起来，实际上网络层的策略对拥塞避免算法影响最大的就是路由器的丢弃策略。在简单的情况下路由器通常按照先进先出的策略处理到来的分组。当路由器的缓存装不下分组的时候就丢弃到来的分组，这叫做尾部丢弃策略。这样就会导致分组丢失，发送方认为网络产生拥塞。更为严重的是网络中存在很多的TCP连接，这些连接中的报文段通常是复用路由路径。若发生路由器的尾部丢弃，可能影响到很多条TCP连接，结果就是这许多的TCP连接在同一时间进入慢开始状态。这在术语中称为全局同步。全局同步会使得网络的通信量突然下降很多，而在网络恢复正常之后，其通信量又突然增大很多。

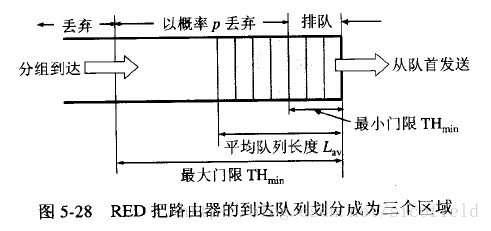
为避免发生网路中的全局同步现象，路由器采用随机早期检测(RED:randomearly detection)。该算法要点如下：

使路由器的队列维持两个参数，即队列长队最小门限min和最大门限max，每当一个分组到达的时候，RED就计算平均队列长度。然后分情况对待到来的分组：

①平均队列长度小于最小门限——把新到达的分组放入队列排队。

②平均队列长度在最小门限与最大门限之间——则按照某一概率将分组丢弃。

③平均队列长度大于最大门限——丢弃新到达的分组。



以概率p随机丢弃分组，让拥塞控制只在个别的TCP连接上执行，因而避免全局性的拥塞控制。 RED的关键就是选择三个参数最小门限、最大门限、丢弃概率和计算平均队列长度。平均队列长度采用加权平均的方法计算平均队列长度，这和往返时间（RTT）的计算策略是一样的。

3.问题

第2章 问题

• 某台计算机有一输出文件F Bits，该计算机的输出链路速率是X bps，但受到其接入网关的令牌桶交通管制；令牌桶始终保持Y bps 的填充速率；令牌桶大小是P Bits。假设发送开始时令牌桶已经充满，试求？

a) 计算机以完全速率发送的持续时间t之表达式？

b) 若X＞Y， F＞P，求文件全部输出完毕所需要的时间T是多少

c) 若X =6Mbps、 Y=1Mbps、 F=10.6MBits 、 P=8MBits求t= ?、 T=？

• 解答

a) t时间内计算机输出数据 = 网关输入数据， tX=tY +P ， t=P/(X-Y)

b) T = t+(F-Xt)/Y

c)t=8MBits/(6Mbps-1Mbps)=1.6s;

T=1.6s+(10.6MBits-1.6s\*6Mbps)/1Mbps=2.6s

第3章

**1．802.11不采用冲突检测？**

由于接收信号的衰减，发送时难于接收（感知冲突），不能感知任何情况下的所有冲突：隐藏终端、衰减，目的是为了避免冲突。

**2. 为什么RTS/CTS不能解决暴露站问题？**

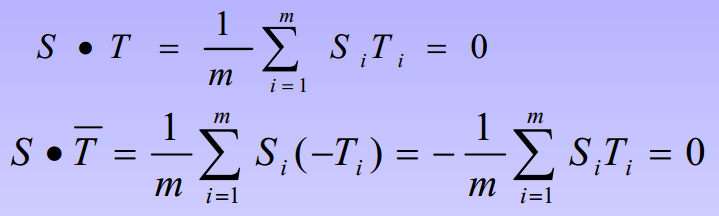
Ad Hoc网络基于IEEE 802．11的协议采用了物理载波侦听和虚拟载波侦听相结合的方式来预约信道。在发送数据之前，发送节点S先物理载波侦听物理信道上是否有信号正在传送．如信道空闲则采用虚拟载波侦听的方式．先发送RTS给接收节点R．R收到后回送一个CTS给S。S和R的邻居节点接收到RTS或CTS后．分别设置自己的NAV(Network Allo．cation Vector)值延迟自己的发送。由于NAV时长为S和R的DATA/ACK发送完成时间。因此S和R就可以安全地发送完DATA／ACK。而邻居的NAV定时器到时后，各节点进入信道竞争窗口，重新竞争信道，发起新一轮的通信。

该协议设置的NAV有效地抑制了隐藏终端问题，能使收发节点对在没有冲突的情况下完成数据交换，但所有监听到RTS／CTS的节点均通过设置NAV值来阻塞自己的发送或接收．因此并没有解决暴露终端问题，仍然浪费了系统带宽。

**3. 为什么在无线网上发送数据帧后要对方必须回确认帧，而以太网则不需要对方发回确认帧？**

无线局域网可能出现检测错误的情况：检测到信道空闲，其实并不空闲，而检测到信道忙，其实并不忙，因此需要接收方发回确认帧来确定信道是否空闲。

**4. 求证CDMA码片序列的正交特性，即若ST=0,证明S(-T） =0？**

****

**\**

**5.** **考虑另一种检测CDMA码片序列正交性的方法。两个序列中的每个元素可以匹配、也可以不匹配。借助于匹配和不匹配来表示码片序列的正交性？**

当两个码片匹配时，它们的乘积为+1.当它们不匹配时，乘积为-1.为使总和为0，匹配的数量与不匹配的数量应该相等。因此，如果一半匹配而另一半不匹配，那么这两个码片序列就是正交的。

**6. 假定A、 B、 C三站都使用CDMA系统同时发送比特“0”，它们的码片序列分别依次如下： A(-1-1-1+1+1-1+1+1);B(-1-1+1-1+1+1+1-1); C:(-1+1-1+1+1+1-1-1)。求发送结果产生的码片序列是什么？**

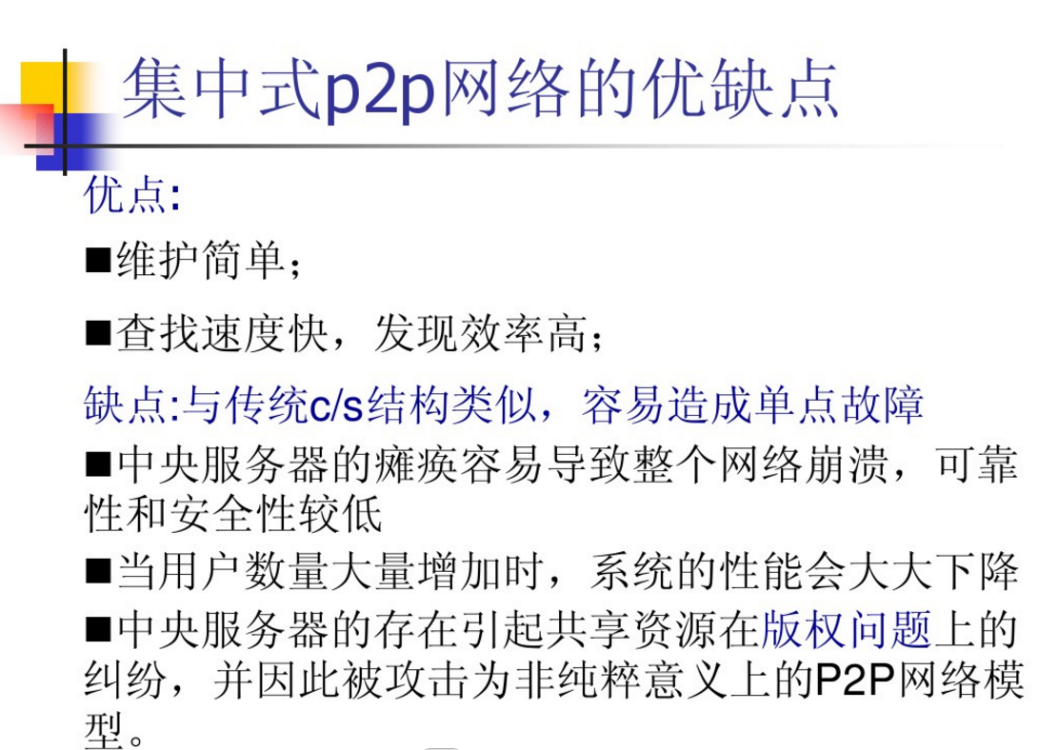
解答：发送比特1时 ＝ +1，发送比特0时 ＝ -1，发送无效=0

-1(-1-1-1+1+1-1+1+1)-1(-1-1+1-1+1+1+1-1)-1(-1+1-1+1+1+1-1-1)=（+3+1+1+1-3-1-1+1）即为产生的码片序列。

第4章

**1. 三代P2P网络之间的定义和差异是什么？**

第一代p2p网络：基于中央控制的集中式p2p网络。中央目录服务器存储的是对等节点的信息资源的目录索引信息。



第二代p2p网络：非结构化p2p网络拓扑

纯P2P网络模型，即没有集中的中央目录服务器，结点之间的内容查询和共享都是直接通过相邻结点广播接力传递，同时每个结点还会记录搜索轨迹，以防止搜索环路的发生。

Gnutella模型是现在应用最广泛的纯P2P非结构化拓扑结构(见图2)，它解决了网络结构中心化的问题，扩展性和容错性较好，但是Gnutella网络中的搜索算法以泛洪的方式进行，控制信息的泛滥消耗了大量带宽并很快造成网络拥塞甚至网络的不稳定。同时，局部性能较差的节点可能会导致Gnutella网络被分片，从而导致整个网络的可用性较差，另外这类系统更容易受到垃圾信息，甚至是病毒的恶意攻击。

混合式网络模型 。Kazaa模型是P2P混合模型的典型代表，它在纯P2P分布式模型基础上引入了超级节点的概念，综合了集中式P2P快速查找和纯P2P去中心化的优势。总的来说，基于超级节点的混合式P2P网络结构比以往有较大程度的改进。 然而，由于超级节点本身的脆弱性也可能导致其簇内的结点处于孤立状态，因此这种局部索引的方法仍然存在一定的局限性。这导致了结构化的P2P网络模型的出现。

第三代p2p网络：结构化p2p网络

所谓结构化与非结构化模型的根本区别在于每个节点所维护的邻居是否能够按照某种全局方式组织起来以利于快速查找。结构化P2P模式是一种采用纯分布式的消息传递机制和根据关键字进行查找的定位服务，目前的主流方法是采用分布式哈希表(DHT)技术，这也是目前扩展性最好的P2P路由方式之一。由于DHT各节点并不需要维护整个网络的信息，只在节点中存储其临近的后继节点信息，因此较少的路由信息就可以有效地实现到达目标节点，同时又取消了泛洪算法。该模型有效地减少了节点信息的发送数量，从而增强了P2P网络的扩展性。同时，出于冗余度以及延时的考虑，大部分DHT总是在节点的虚拟标识与关键字最接近的节点上复制备份冗余信息，这样也避免了单一节点失效的问题。

**2．请描述Kademlia协议是如何找到资源的？**

通过把资源信息与键进行映射，资源即可进行定位，哈希表是典型的用来映射的手段。由于以前的STORE消息，存储节点将会有对应STORE所存储的相关资源的信息。定位资源时，如果一个节点存有相应的资源的值的时候，它就返回该资源，搜索便结束了，除了该点以外，定位资源与定位离键最近的节点的过程相似。

考虑到节点未必都在线的情况，资源的值被存在多个节点上（节点中的K个），并且，为了提供冗余，还有可能在更多的节点上储存值。 储存值的节点将定期搜索网络中与储存值所对应的键接近的K个节点并且把值复制到这些节点上，这些节点可作为那些下线的节点的补充。另外，对于那些普遍流行的内容，可能有更多的请求需求，通过让那些访问值的节点 把值存储在附件的一些节点上（不在K个最近节点的范围之类）来减少存储值的那些节点的负载，这种新的存储技术就是缓存技术。通过这种技术，依赖于请求的数量，资源的值被存储在离键越来越远的那些节点上，这使得那些流行的搜索可以更快地找到资源的储存者。由于返回值的节点的NODE\_ID远离值所对应的关键字，网络中的“热点”区域存在的可能性也降低了。依据与键的距离，缓存的那些节点在一段时间以后将会删除所存储的缓存值。 DHT的某些实现（如Kad）即不提供冗余（复制）节点也不提供缓存，这主要是为了能够快速减少系统中的陈旧信息。在这种网络中，提供文件的那些节点将会周期性地更新网络上的信息（通过NODE\_LOOKUP消息和STORE消息）。当存有某个文件的所有节点都下线了，关于该文件的相关的值（源和关键字）的更新也就停止了，该文件的相关信息也就从网络上完全消失了。

**3. Chord和Kademlia的原理。**

**Chord原理：**

Chord通过把Node和Key映射到相同的空间而保证一致性哈希，为了保证哈希的非重复性，Chord选择SHA-1作为哈希函数，SHA-1会产生一个2160的空间，每项为一个16字节（160bit）的大整数。我们可以认为这些整数首尾相连形成一个环，称之为Chord环。整数在Chord环上按大小顺时针排列，Node（机器的IP地址和Port）与Key（资源标识）都被哈希到Chord环上，这样我们就假定了整个P2P网络的状态为一个虚拟的环，因此我们说Chord是结构化的P2P网络。

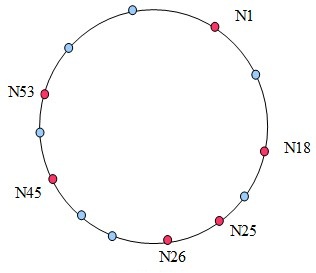
下面有几个定义：

我们称Chord环上的每个节点为标志符

如果某个Node映射到了某个标志符，则继续称该标准符为Node

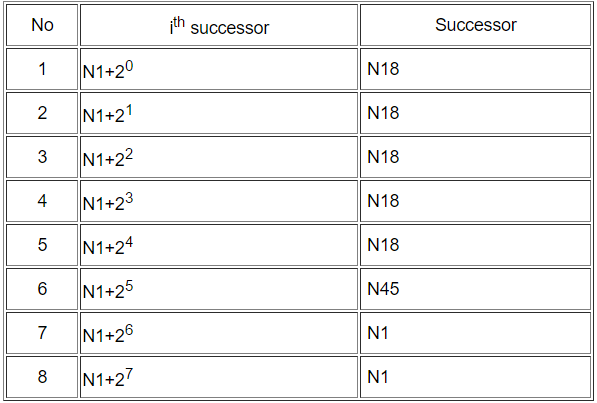
按顺时针，节点前面的成为前继(predecessor),节点后面的成为后继（successor)；同理，第一个predecessor称之为直接前继，第一个successor称之为直接后继

如图：



Chord环

红色点为Node，蓝色为标志符。上面只是部分节点和标志符，以节点N1为例说明其Finger表中的successor：



把Node和Key都映射到一个值域感觉是把狗和猫放在一起衡量，虽然有点怪，但这样可以保证一致性哈希，具体可以参考前文。

很显然，分布在Chord环上的Node数远远小于标志符数（2160是一个无法衡量的天文数字），这样Chord环上的Node就会很稀疏地分布在Chord环上，理论上应该是随机分布，但如前面一致性哈希的讨论，如果节点数量不多，分布肯定是不均匀的，可以考虑增加虚拟节点来增加其平衡性，如果在节点较多（比如大型的P2P网络有上百万的机器）就不必引入虚拟节点。

很显然，任何查找只要沿Chord环一圈结果肯定可以找到，这样的时间复杂度是O(N)，N为网络节点数，但对一个上百万节点，且节点经常加入、退出的P2P网络来说，O(N)是不可忍受的，因此Chord提出了下面非线性查找的算法：

1.每个节点都维护一个Finger表，该表长度为m（m就是位数，在Chord中为160），该表的第i项存放节点n的第(n+2i-1) mod 2m个successor(1<=i<=m)

2.每个节点都维护一个predecessor和successor列表，该列表的作用是能快速定位前继和后继，并能周期性检测前继和后继的健康状态

3.就是说存放的successor是按2的倍数等比递增，自所以取模是因为最后的节点的successor是开始的几个节点，比如最大的一个节点的下一个节点定义为第一个节点

4.资源Key存储在下面的Node上：沿Chord环，hash(Node)>=hash(key)的第一个Node，我们称这个Node为这个Key的successor

5.给定一个Key，按下面的步骤查找其对应的资源位于哪个节点，也就是查找该Key的successor：（假如查找是在节点n上进行）

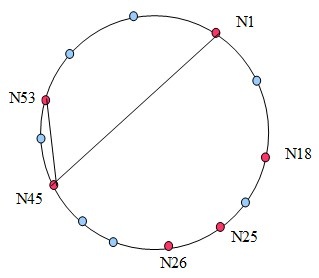
查看Key的哈希是否落在节点n和其直接successor之间，若是结束查找，n的successor即为所找

在n的Finger表中，找出与hash(Key)距离最近且<hash(Key)的n的successor，该节点也是Finger表中最接近Key的predecessor，把查找请求转发到该节点

继续上述过程，直至找到Key对应的节点

从直觉上来说，上次查找过程应该是指数收敛的，类似二分法的查找，收敛速度应该是很快的；反过来，查找时间或路由复杂度应该是对数即的，在下面我们会证明这一点。

下图表明了节点N1查找节点N53的过程，还是非常快的：



节点N1查找N53

**Kademlia的原理：**

Kad 是一种分布式哈希表（DHT）技术，不过和其他DHT 实现技术比较，如Chord、CAN、Pastry 等，Kad 通过独特的以异或算法（XOR）为距离度量基础，建立了一种全新的DHT拓扑结构，相比于其他算法，大大提高了路由查询速度。

一．节点状态

在Kad网络中，所有节点都被当作一颗二叉树的叶子，并且每一个节点的位置都由其ID值的最短前缀唯一的确定。

对于任意一个节点，都可以把这颗二叉树分解为一系列连续的，不包含自己的子树。最高层的子树，由整颗树不包含自己的树的另一半组成；下一层子树由剩下部分不包含自己的一半组成；依此类推，直到分割完整颗树。图1就展示了节点0011如何进行子树的划分：

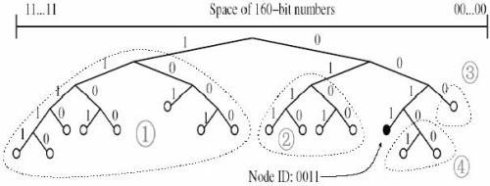


图1：节点0011的子树划分

虚线包含的部分就是各子树，由上到下各层的前缀分别为0，01，000，0010。

Kad协议确保每个节点知道其各子树的至少一个节点，只要这些子树非空。在这个前提下，每个节点都可以通过ID值来找到任何一个节点。这个路由的过程是通过所谓的XOR（异或）距离得到的。

图2就演示了节点0011如何通过连续查询来找到节点1110的。节点0011通过在逐步底层的子树间不断学习并查询最佳节点，获得了越来越接近的节点，最终收敛到目标节点上。

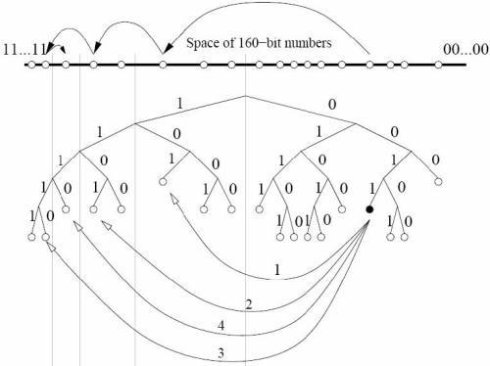


图2：通过ID值定位目标节点

需要说明的是，只有第一步查询的节点101，是节点0011已经知道的，后面各步查询的节点，都是由上一步查询返回的更接近目标的节点，这是一个递归操作的过程。

二、节点间距离

Kad 网络中每个节点都有一个160bit的ID值作为标志符，Key也是一个160bit的标志符，每一个加入Kad网络的计算机都会在160bit的key 空间被分配一个节点ID（node ID）值（可以认为ID是随机产生的），<key,value>对的数据就存放在ID值“最”接近key值的节点上。

判断两个节点x,y的距离远近是基于数学上的异或的二进制运算，d(x,y) = x？y，既对应位相同时结果为0，不同时结果为1。例如：

010101

XOR 110001

-------------

100100

则这两个节点的距离为32＋4＝36。

显然，高位上数值的差异对结果的影响更大。

对于异或操作，有如下一些数学性质：

? d(x, x) = 0

? d(x, y) > 0， if x ≠ y

? ∨x, y : d(x, y) = d(y, x)

? d(x, y) + d(y, z) ≧ d(x, z)

? d(x, y) ？ d(y, z) = d(x, z)

? ∨a≧ 0, b≧ 0, a + b≧ a ？ b

正 如Chord的顺时针旋转的度量一样，异或操作也是单向性的。对于任意给定的节点x和距离⊿≧0，总会存在一个精确的节点y，使得d(x,y)= ⊿。另外，单向性也确保了对于同一个key值的所有查询都会逐步收敛到同一个路径上，而不管查询的起始节点位置如何。这样，只要沿着查询路径上的节点都缓 存这个<key,value>对，就可以减轻存放热门key值节点的压力，同时也能够加快查询响应速度。

三、K桶

Kad的路由表是通过一些称之为K桶的表格构造起来的。这有点类似Tapestry技术，其路由表也是通过类似的方法构造的。

对 每一个0≦i≦160，每个节点都保存有一些和自己距离范围在区间 内的一些节点信息，这些信息由一些(IP address,UDP port,Node ID)数据列表构成（Kad网络是靠UDP协议交换信息的）。每一个这样的列表都称之为一个K桶，并且每个K桶内部信息存放位置是根据上次看到的时间顺序 排列，最近（least-recently）看到的放在头部，最后（most-recently）看到的放在尾部。每个桶都有不超过k个的数据项。

一个节点的全部K桶列表如表1所示：

I 距离 邻居

0 0 1[2 , 2) 0-1 (IP address,UDP port,Node ID) ......

0-k (IP address,UDP port,Node ID)

1 1 2[2 , 2) 1-1 (IP address,UDP port,Node ID) ......

1-k (IP address,UDP port,Node ID)

2 2 3[2 , 2) (IP address,UDP port,Node ID) ...... 2-1

(IP address,UDP port,Node ID) 2-k

……

i 1[2 , 2) i i+ i-1 (IP address,UDP port,Node ID) ......

i-k (IP address,UDP port,Node ID)

……

表1：K桶结构

不过通常来说当i值很小时，K桶通常是空的（也就是说没有足够多的节点，比如当i＝0时，就最多可能只有1项）；而当i值很 大时，其对应K桶的项数又很可能会超过k个（当然，覆盖距离范围越广，存在较多节点的可能性也就越大），这里k是为平衡系统性能和网络负载而设置的一个常 数，但必须是偶数，比如k＝20。在BitTorrent的实现中，取值为k＝8。

由于每个K桶覆盖距离的范围呈指数关系增长，这就形成了离自己 近的节点的信息多，离自己远的节点的信息少，从而可以保证路由查询过程是收敛。因为是用指数方式划分区间，经过证明，对于一个有N个节点的Kad网络，最 多只需要经过logN步查询，就可以准确定位到目标节点。这个特性和Chord网络上节点的finger table划分距离空间的原理类似。

当节点x收到一个PRC消息时，发送者y的IP地址就被用来更新对应的K桶，具体步骤如下：

1．计算自己和发送者的距离：d(x,y) = x？y，注意：x和y是ID值，不是IP地址

2．通过距离d选择对应的K桶进行更新操作。

3．如果y的IP地址已经存在于这个K桶中，则把对应项移到该该K桶的尾部

4．如果y的IP地址没有记录在该K桶中

⑴如果该K桶的记录项小于k个，则直接把y的(IP address,UDP port,Node ID)信息插入队列尾部

⑵如果该K桶的记录项大于k个，则选择头部的记录项（假如是节点z）进行RPC\_PING操作

①如果z没有响应，则从K桶中移除z的信息，并把y的信息插入队列尾部

②如果z有响应，则把z的信息移到队列尾部，同时忽略y的信息。

K桶的更新机制非常高效的实现了一种把最近看到的节点更新的策略，除非在线节点一直未从K桶中移出过。也就是说在线时间长的节点具有较高的可能性继续保留在K桶列表中。

采用这种机制是基于对Gnutella网络上大量用户行为习惯的研究结果，既节点的失效概率和在线时长成反比关系，如图3（横坐标为分钟，纵坐标为概率）：

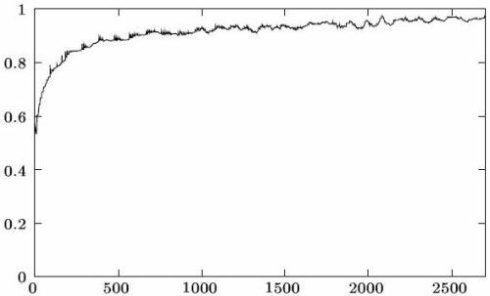


图3：Gnutella网络中在线时长和继续在线的概率关系

可以明显看出，用户在线时间越长，他在下一时段继续在线的可能性就越高。

所以，通过把在线时间长的节点留在K桶里，Kad就明显增加K桶中的节点在下一时间段仍然在线的概率，这对应Kad网络的稳定性和减少网络维护成本（不需要频繁构建节点的路由表）带来很大好处。

这种机制的另一个好处是能在一定程度上防御DOS攻击，因为只有当老节点失效后，Kad才会更新K桶的信息，这就避免了通过新节点的加入来泛洪路由信息。

为了防止K桶老化，所有在一定时间之内无更新操作的K桶，都会分别从自己的K桶中随机选择一些节点执行RPC\_PING操作。

上述这些K桶机制使Kad缓和了流量瓶颈（所有节点不会同时进行大量的更新操作），同时也能对节点的失效进行迅速响应。

四、Kademlia协议操作类型

Kademlia协议包括四种远程操作：PING、STORE、FIND\_NODE、FIND\_VALUE。

1、PING操作的作用是探测一个节点，用以判断其是否仍然在线。

2、STORE操作的作用是通知一个节点存储一个<key,value>对，以便以后查询需要。

3、FIND\_NODE操作使用一个160bit的ID作为参数。本操作的接受者返回它所知道的更接近目标ID的K个节点的(IP address,UDP port,Node ID)信息。

这些节点的信息可以是从一个单独的K桶获得，也可以从多个K桶获得（如果最接近目标ID的K桶未满）。不管是哪种情况，接受者都将返回K个节点的信息给操作发起者。但如果接受者所有K桶的节点信息加起来也没有K个，则它会返回全部节点的信息给发起者。

4、FIND\_VALUE操作和FIND\_NODE操作类似，不同的是它只需要返回一个节点的(IP address,UDP port,Node ID)信息。如果本操作的接受者收到同一个key的STORE操作，则会直接返回存储的value值。

注：在Kad网络中，系统存储的数据以<key,value>对形式存放。根据笔者的分析，在BitSpirit的DHT实现中，其key值为torrent文件的info\_hash串，其value值则和torrent文件有密切关系。

为了防止伪造地址，在所有RPC操作中，接受者都需要响应一个随机的160bit的ID值。另外，为了确信发送者的网络地址，PING操作还可以附带在接受者的RPC回复信息中。

五、路由查询机制

Kad技术的最大特点之一就是能够提供快速的节点查找机制，并且还可以通过参数进行查找速度的调节。

假如节点x要查找ID值为t的节点，Kad按照如下递归操作步骤进行路由查找：

1、 计算到t的距离：d(x,y) = x？y

2、 从x的第[㏒d]个K桶中取出α个节点的信息（“[”“]”是取整符号），同时进行FIND\_NODE操作。如果这个K桶中的信息少于α个，则从附近多个桶中选择距离最接近d的总共α个节点。

3、 对接受到查询操作的每个节点，如果发现自己就是t，则回答自己是最接近t的；否则测量自己和t的距离，并从自己对应的K桶中选择α个节点的信息给x。

4、 X对新接受到的每个节点都再次执行FIND\_NODE操作，此过程不断重复执行，直到每一个分支都有节点响应自己是最接近t的。

5、 通过上述查找操作，x得到了k个最接近t的节点信息。

注意：这里用“最接近”这个说法，是因为ID值为t的节点不一定存在网络中，也就是说t没有分配给任何一台电脑。

这里α也是为系统优化而设立的一个参数，就像K一样。在BitTorrent实现中，取值为α＝3。

当α＝1时，查询过程就类似于Chord的逐跳查询过程，如图4。

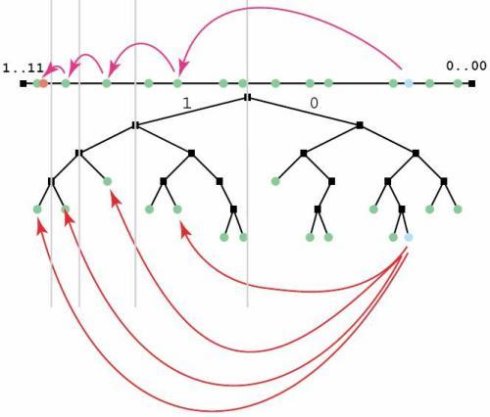
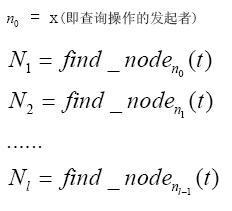


图4：α＝1时的查询过程

整个路由查询过程是递归操作的，其过程可用数学公式表示为：



这个递归过程一直持续到 Nl ＝t，或者 Nl 的路由表中没有任何关于t 的信息，即查询

失败。

由于每次查询都能从更接近t的K桶中获取信息，这样的机制保证了每一次递归操作都能够至少获得距离减半（或距离减少1bit）的效果，从而保证整个查询过程的收敛速度为O(logN)，这里N为网络全部节点的数量。

当节点x要查询<key,value>对时，和查找节点的操作类似，x选择k个ID值最接近key值的节点，执行FIND\_VALUE操作，并对每一个返回的新节点重复执行FIND\_VALUE操作，直到某个节点返回value值。

一 旦FIND\_VALUE操作成功执行，则<key,value>对数据会缓存在没有返回value值的最接近的节点上。这样下一次查询相同的 key时就会更加快速的得到结果。通过这样的方式，热门<key,value>对数据的缓存范围就逐步扩大，使系统具有极佳的响应速度，如图 5所示。

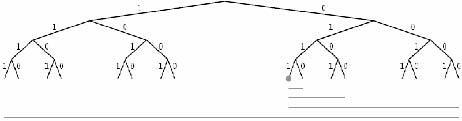


图5：缓存原则

六、数据存放

存放<key,value>对数据的过程为：

1、 发起者首先定位k个ID值最接近key的节点；

2、 发起者对这k个节点发起STORE操作

3、 执行STORE操作的k个节点每小时重发布自己所有的<key,value>对数据。

4、 为了限制失效信息，所有<key,value>对数据在初始发布24小时后过期。

另外，为了保证数据发布、搜寻的一致性，规定在任何时候，当节点w发现新节点u比w上的某些<key,value>对数据更接近，则w把这些<key,value>对数据复制到u上，但是并不会从w上删除。

七、节点加入和离开

如果节点u要想加入Kad网络，它必须要和一个已经在Kad网络的节点，比如w，取得联系。

u首先把w插入自己适当的K桶中，然后对自己的节点ID执行一次FIND\_NODE操作，然后根据接收到的信息更新自己的K桶内容。通过对自己邻近节点由近及远的逐步查询，u完成了仍然是空的K桶信息的构建，同时也把自己的信息发布到其他节点的K桶中。

在Kad网络中，每个节点的路由表都表示为一颗二叉树，叶子节点为K桶，K桶存放的是有相同ID前缀的节点信息，而这个前缀就是该K桶在二叉树中的位置。这样，每个K桶都覆盖了ID空间的一部分，全部K桶的信息加起来就覆盖了整个160bit的ID空间，而且没有重叠。

以节点u为例，其路由表的生成过程为：

1． 最初，u的路由表为一个单个的K桶，覆盖了整个160bitID空间，如图6最上面的路由表；

2． 当学习到新的节点信息后，则u会尝试把新节点的信息，根据其前缀值插入到对应的K桶中：

① 如果该K桶没有满，则新节点直接插入到这个K桶中；

② 如果该K桶已经满了，

⑴ 如果该K桶覆盖范围包含了节点u的ID，则把该K桶分裂为两个大小相同的新K桶，并对原K桶内的节点信息按照新的K桶前缀值进行重新分配

⑵ 如果该K桶覆盖范围没有包节点u的ID，则直接丢弃该新节点信息

3． 上述过程不断重复，最终会形成表1结构的路由表。达到距离近的节点的信息多，距离远的节点的信息少的结果，保证了路由查询过程能快速收敛。

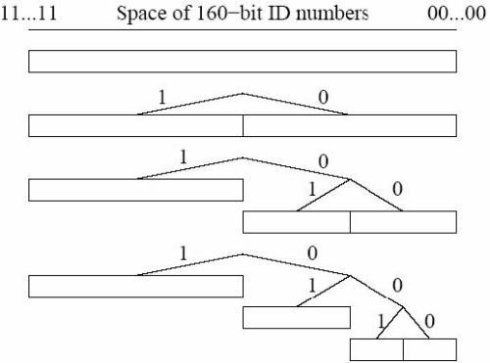


图6：节点000的路由表生成演化

在图7 中，演示了当覆盖范围包含自己ID 值的K 桶是如何逐步分裂的。

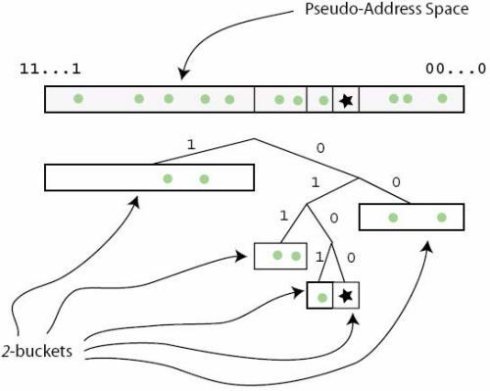


图7：节点0100的K桶分裂过程

当K桶010满了之后，由于其覆盖范围包含了节点0100的ID，故该K桶分裂为两个新的K桶：0101 和0100，原K桶010的信息会根据其其前缀值重新分布到这两个新的K桶中。注意，这里并没有使用160bit的ID值表示法，只是为了方便原理的演 示，实际Kad网络中的ID值都是160bit的。

节点离开Kad网络不需要发布任何信息，Kademlia协议的目标之一就是 能够弹性工作在任意节点随时失效的情况下。为此，Kad要求每个节点必须周期性的发布全部自己存放的<key,value>对数据，并把这些 数据缓存在自己的k个最近邻居处，这样存放在失效节点的数据会很快被更新到其他新节点上。

**4. 论述Bitcoin原理。**

Bitcoin 为什么保值，BTC (Bitcoin 的货币简称）存在于一个庞大的 p2p 网络中。使用 Bitcoin 的群体公认了一种算法，这种算法在现今的条件下，每小时只会新产生大约 6 组新的 BTC ，目前一组是 50 个。也就是说，这个世界上，每个小时大约只会产生 300 个 BTC 。这个产量还会由网络自动调整难度来限制产量。你没办法通过修改所有人的 Client 的算法及参数（client 是开源的）来加快货币产量。伪造的货币会被网络丢弃（除非你可以控制大部分网络节点）。

BTC 本身有什么价值？

BTC 的价值就是交易渠道本身。一组新制造出来的 BTC 提供了把旧的 BTC 从一个帐户转移到另一个帐户的数学保证。这个安全保证背后的代价是大量的计算力。生产这么一个安全通道是需要消耗大量能源的，所以整个 BTC 用户群体，奖励那个造币者（目前是 50 BTC)。

简单说，我的理解就是，现在世界上所有的 BTC 背后都是用运行计算机的能量产生出来的，它们的总价值，（到现在一共有大约 12w 组 BTC 被生产出来，每组 50 个，市场价格大约 7.3 美金一个），应该是少于消耗掉的能源的总市场价值的。不过我想，用于生产 BTC 的能源大都原本就是不用也被浪费掉的资源。

一个没有中心节点的“银行”是怎么让大家信任并工作起来的呢？

答案是，这个 p2p 网络上每个节点都记录了 BTC 诞生以来的每笔交易的详单，并从中可以推测出每个 BTC 唯一的属于谁。这样你接受一笔交易时，就能知道别人给你的钱是不是合法的。

从最基本的说起：

每个帐户其实就是一对公私匙，有私匙的人就是帐户的主人。如果 A 要给 B 转一笔钱，A 就把钱的数量加上 B 的公匙，用自己的钥匙签名。而 B 看到这个签名，就可以了解，的确是 A 转给了他如数的 BTC 。

那么这笔交易需要一个见证人，担保交易发生过。这样，以后 B 想用这笔钱的时候才是合法的。担保人就是整个使用 BTC 的网络。

A 在发起这笔交易的时候，必须把签过名的交易单尽量的广播到 p2p 网络上，最终会让每个节点都知道这件事。B 从 p2p 网络上不断的收到别人的确认信息。当它收到足够多的确认信息后，就认为 A 的确发出了这条交易单。这以后，B 就可以自由使用这笔钱了。

当 B 使用 A 转给它的钱给 C 时，也会广播给足够多（最终所有人都收到）的人让他们担保。每个担保人只有确信 B 有足够多的钱可以支付的时候才做确认。本质上，BTC 网络并没有记录每一块钱属于谁，它记录的是从诞生起到当前的每一笔交易，并推算出每个帐户里有多少钱。任何人试图确认一个交易单时，它需要确认的是转出帐号上有没有那么多钱。

Bitcoin 需要解决的核心问题是，如何避免一笔钱被花两次。

整个帐单序列是一环套一环的。每个人在完整的全局帐单上签上新的一笔的时候，都需要利用前面信息生成后面的。这个帐单序列被称为 chain of blocks 。每个 block 里面包含有若干条经过确认并 hash 签名 (难以伪造) 的交易记录。每个 block 都和全局表上的上一个 block 有关联。每条帐单都会通过 p2p 网络最终被转发给制造新 block 的节点上。

这个制造新 block 的过程被叫做 mining ，制造新 block 就是把最近收到的帐单打包在刚制造的 block 里。这个打包的过程即制作的过程，只有极其稀少的几率被制造成功。（你可以理解成把新收到的帐单合在一起，一次成型不可修改，如果制造失败就要再来一次）一旦制造成功，你就把新的 block （被认为是对老的全局 block 链的延续）广播出去。

因为是 p2p 网络，可能有许多人都在同时制造新的 block ，但有一个排序机制保证只有最优（最难，花费最大计算时间的）的那个新 block 被网络群体接受，挂在全局的 block 链上。重复一次，整个 BTC 网络只有一个全局帐单表，每个节点都完整的保存有一份。

这个全局帐单表会越来越大，block 链越来越长，在最新的部分，必然有许多分茬。这是因为 p2p 网络的 mining 过程是分开并行进行的，每条新帐单也不能立刻广播给所有的节点。每个 mining 的节点都有责任把他新收到的，在他认可的老的全局帐单上不存在的帐单，合在他准备制造的新 block 中。一旦新 block 被制造出来，就立刻广播出去，争取得到更多人的认可。主要是得到那些想 mining 的人的认可，这些人会在这个 block 的基础上制造新的 block 。

如果 p2p 网络过大，交易帐单不能尽量的迅速的广播到全网络。就会出来 p2p 的网络的局部保持有小群体共同认可的一份全局帐单。多个全局帐单的分支同时发展是有可能的。因为每个小群体都可能认为他们看见的那部分更长更有效。但是，只有有人发现另一条分支更长，它就会转换阵营。所以，有一定的可能性，你的帐单被一个小群体接受，但在一段时间后，被更大的阵营抛弃。

不过，算法参数决定了，新的 block 产生速度很慢，如果你的帐单被多达 6 个人确认，基本上就保证了它合并到的那份全局帐单，就是 p2p 网络全体认可的。

既然生成新 block 费时费力，制造出新 block 的几率好象买彩票中大奖，还有那么多人去执行程序计算出新 block 呢？答案是，每个制造出新 block 的人，都有权利构造一条帐单声明老天给了我 50 BTC 。这个规则是被所有 BTC 用户共同承认的。把制造 block 等同于成挖金矿 (mining) 只是一个形象上的比喻。实际上，没有人可以把金子挖出来囤积。每个新 block 必须包含全局表上的上一个 block 的 hash 值，BTC 网络自我调节难度，让每 10 分钟大约产生一个新 block 。如果你 10 分钟内没制造出新的 block ，差不多就是说你前面 10 分钟干的活白干了。从最新版的 block 继续演算。

所以更恰当的比喻是买彩票。一个每 10 分钟开一次的彩票。你不停的花钱买，10 分钟内开中了就是你的，开不中先买的都作废，然后下一轮。

数学上怎样保证 mining 的过程需要消耗大量的 CPU 时间？并只有很小的几率成功？

这里用到一个叫做 Hashcash 的系统。它最早是为了改善 email spam 的问题被发明出来的。

就是给一段特定信息（比如这封 email 是从谁发给谁）加一个特定的 hash 头。这个 hash 头需要大量的 CPU 时间计算出来。发 spam 的人没有那么多 CPU 时间为群发的每一封 email 计算一个符合要求的 hash 头，所以认为有这个合法 hash 头的 email 不太可能是 spam （花了 CPU 时间在上面）

这个算法就是，为你想保护的信息，找到一串数字，附加上去后，使用某种公认的 hash 算法，比如 SHA-2 ，算出一个 hash 值。如果 hash 值由一长串 0 打头（具体多少个决定了难度），那么就成功了。

为一段信息，找到这串数字，在目前来说，除了暴力尝试没有什么好的方法。也就是随机更换数字，换一次就 hash 一次比对。在一个可以预期的尝试次数后，一般都能找到想找的数字。

每个想 mining 赚 BTC 的人，不停的从 BTC 网络上监听信息。如果有人发布了新的合法的 block ，他就合并到本地的全局表里。并重置自己的计算过程，从新得到的 block 开始。如果有新发布的交易单，也记录下来。不断的把最新的 block 的 hash 值、新收到的交易单，自己获得 50 BTC 的那条奖励单合并在一起，计算 SHA-256 ，看看结果是否满足条件。一旦满足，就把这个新的 block 广播出去。

当足够的人认可它，（以它为基础计算后面的 block ），他也就获得了那 50 BTC 。

为了匹配 BTC 的经济规模。所有的 Bitcoin client 都被设置成，每 210000 个 block ，生产新 block 的人被认可凭空获得的 BTC 数量比之前的少一半（如果这个时候他还在包内写上自己获得 50 BTC ，其他人不会确认他的这个 block ）。这会让 BTC 的总量增速变缓。新的 block 产生的速度是由难度来调节的。这个难度会由 p2p 网络根据最近生产 block 的速度自动调节。所以即使日后计算能力增加，也能保证大约 10 分钟一个的速度。

而且，随着生产新 block 的收益减少，愿意贡献自己的 CPU 来 mining 的节点也会变少。（如果减少太多，只需要减少难度即可）

最终，p2p 网络不再凭空制造出新的 BTC ，这个时候制造新的 block 的动力是什么呢？那就是交易税。因为没有什么人愿意生产新的 block ，发起交易就变的困难。（没有 block 可以容纳交易单）希望交易被确认的人可以声明，如果有人制造出新的 block 接纳他的交易单，他会支付一小笔交易税给他。当许多人都这么做的时候，制造 block 又变的有利可图了。只不过，直接上不再有新的 BTC 诞生，只是在这些 BTC 用户之间流通。

总有一些 BTC 会消失，主要是那些帐号的私匙丢失了，没有任何人可以转移走帐户上的钱。不能流通的货币就不是货币了。但最终 BTC 总体会达到一个比较大的规模，准确说是两千一百万个。但 BTC 本身是可以切割的，比如你可以支付给别人 0.01 个 BTC 。所以 BTC 本身会升值，总数也一直够用。

第5章

**1. IPv4和IPv6差异点？**

1.扩大地址空间，路由结构更层次化

2.报头格式大简化，便于硬件处理

3.网络管理更加简单

4.安全性支持

以及QoS能力，多播寻址，新的集群通信方式，可移植性等

**2. IPv6地址分类和特点？**

IPv6的地址类型分为三类

单播－unicast：目的地址指明一个单一的计算机（single

interface）,可是主机或路由器，发送到unicast的包将选择一条

最短的路径到达目的站

近播－anycast（集群…）:目的地址是共享一个地址前缀的计算

机集合（a set of interfaces）,典型的情况是在同一物理网络上

的不同节点，发送到anycast地址的包将选择一条最近路径到达

该集群（路由度量距离最近的节点）中一个。

组播-multicast:目的地址是一组计算机（a set of interfaces）

，典型情况是属于不同网路的不同节点，发送到一个multicast地

址的包将通过硬件或广播投递给组中的每个成员。IPv6中没有广

播地址，其功能可由组播取代

特点：

1.所有IPv6地址都是分配给interface而不是node的，所有接口都必须有至少一个link-local unicast，一个单接口可分配任何一种类型的多重地址（uni/any/multicast）或地址范围

2.子网前缀仅与一条链路相连（继承v4），但多重子网前缀可分配给同一条链路

**3．IPv6地址自动配置过程**

无状态自动配置过程要求节点采用如下步骤：

1 进行自动配置的节点必须确定自己的链路本地地址；

2 必须验证该链路本地地址在链路上的唯一性；

3 节点必须确定需要配置的信息。该信息可能是节点的IP地址，或者是其他配置信息，或者两者皆有。如果需要IP地址，节点必须确定是使用无状态自动配置过程还是使用状态自动配置过程来获得。

具体地说，在无状态自动配置过程中，主机首先通过将它的网卡MAC地址附加在链接本地地址前缀1111111010之后，产生一个链接本地单播地址（IEEE已经将网卡MAC地址由48位改为了64位。如果主机采用的网卡的MAC地址依然是48位，那么IPv6网卡驱动程序会根据IEEE的一个公式将48位MAC地址转换为64位MAC地址）。接着主机向该地址发出一个被称为邻居发现（neighbor discovrey）的请求，以验证地址的唯一性。如果请求没有得到响应，则表明主机自我设置的链接本地单点广播地址是唯一的。否则，主机将使用一个随机产生的接口ID组成一个新的链接本地单点广播地址。然后，以该地址为源地址，主机向本地链接中所有路由器多点广播一个被称为路由器请求（router solicitation）的配置信息请求，路由器以一个包含一个可聚集全局单点广播地址前缀和其它相关配置信息的路由器公告响应该请求。主机用它从路由器得到的全局地址前缀加上自己的接口ID，自动配置全局地址，然后就可以与Internet中的其它主机通信了。

如果没有路由器为网络上的节点服务，也即本地网络孤立于其他网络，有两种方式可以完成配置：

1 节点寻找配置服务器来完成其配置；

2 节点侦听路由器通告报文，这些报文周期性地发往所有主机的组播地址，以指明诸如网络地址和子网地址等配置信息。节点可以等待路由器的通告，也可以通过发送组播请求给所有路由器的组播地址来请求路由器发送通告。一旦收到路由器的响应，节点就可以使用响应的信息来完成自动配置。

使用无状态自动配置，无需手动干预就能够改变网络中所有主机的IP地址。例如，当企业更换了联入Internet的ISP时，将从新ISP处得到一个新的可聚集全局地址前缀。ISP把这个地址前缀从它的路由器上传送到企业路由器上。由于企业路由器将周期性地向本地链接中的所有主机多点广播路由器公告，因此企业网络中所有主机都将通过路由器公告收到新的地址前缀，此后，它们就会自动产生新的IP地址并覆盖旧的IP地址。

**首先主机向路由器发送RS（路由器请求报文），路由器收到后回复RA（路由器通告报文）；当主机收到RA消息后，将子网前缀和本机的网络接口标识结合在一起，就生成了一个地址。例如RA消息中提供的子网前缀是 3ffe:3216:2101:2106::/64，网络接口的48位链路层地址是02:E0:4C:30:0A:D5，则首先将链路层地址根据 EUI64编码规则变换成64位的接口标识02:e0:4c:ff:fe:30:0a:d5，然后将接口标识附加到子网前缀后，就构成了一个新的网络地址3ffe:3216:2101:2106:02e0:4cff:fe30:0ad5。**

**4. 已知华中科技大学(总面积4517542平方米=约450万平方米=4.5**

**平方公里)分配到的IPv6地址是 2001： 0250： 4000::/48**

**请问这是一个何种类型的 Unicast / Multicast地址？其相应**

**FP/TLA ID/RES/NLA ID/SLA ID/接口ID分别是多少？ HUST是一个TLA ID或 NLA ID或SLA ID机构？HUST所分得的地址空间相当于v4的多好个A类地址？该地址占整个IPv6地址空间的比例是多大？**

解答

• 0010 0000 0000 0001： 0000 0010 0101 0000： 0100 0000 0000 0000：： /48

• 2128－48＝280 ≈(210)8=(103)8 =1024/450万m2=2.22\*1017/m2

• 2.22\*1017 / 6.02×1023 ≈ 0.368×10-6 = 百万分之一的3分之一个摩尔数

• 280 / 224 = 256 ≈(103)5.6=1016.8个A类

• 280 / 2128 ＝ 1/ 248 ≈ 1/1014.4 = 百万亿分之一

第6章

**1. Openflow的定义？**

OpenFlow一种网络协议，具有四个特点：

第一，设备具备商用的高性能和低价格的特点；

第二，设备必须能支持各种不同的研究范围；

第三，设备必须能隔绝实验流量和运行流量；

第四，设备必须满足设备制造商封闭平台的要求。

OpenFlow网络由OpenFlow交换机，FlowVisor和Controller三部分组成。OpenFlow交换机进行数据层的转发；FlowVisor网络进行虚拟化；Controller对网络进行集中控制，实现控制差点功能。

**2. Openflow和SDN的差异？**

SDN （Software Defined Networking）

SDN不等于OpenFlow,其关系类似于互联网不等于IP协议，PSTN不等于7号信令，IMS不等于SIP,WEB体系不等于HTTP协议一样。

OpenFlow仅仅是SDN中控制器控制转发面设备的协议而已，控制器本身的架构、网络拓扑算法、运行环境、编程工具，以及上层应用的集成技术都是SDN的一部分，并且是架构上更为核心的部分。

**3. NV和NFV的定义**

NV（网络虚拟化）是指为了实现应用或租户的隔离，而对网络资源或途径进行虚拟化。---Self

网络虚拟化其实是通过在流量层面逻辑地划分网络，以实现在现有的网络中创建逻辑网段，这类似于将硬盘驱动器分区。

NV是一种覆盖，也是一个通道。NV是在现有的网络中创建一个通道以连接两个域，而不是在一个网络中将两个域进行物理连接。NV很有价值，因为管理员不再需要物理地接通每个新的域，尤其是那些创建的虚拟机的连接。管理员不需要再改变他们之前已经做好的工作，而是得到了一种新的方式来虚拟化其基础设施，并更改现有的基础设施。

NFV（网络功能虚拟化）网络功能虚拟化旨在对四到七层功能进行虚拟化处理，其中包括防火墙、IDPS甚至负载平衡机制(应用程序交付控制器)。

管理员可以通过简单的点击来设置虚拟机，但他们却不能以同样的方式打开防火墙或IDS/IPS，而这正是NFV可以实现的功能。

网络功能虚拟化利用最佳实践方式作为基础策略，针对不同网络元素加以配置。如果大家已经依靠网络虚拟化建立起一条贯通整套基础设施的特殊通道，则完全可以将防火墙或者IDS/IPS添加进去。PLUMgrid、Embrane等公司就提供这种防火墙或IDS/IPS功能。

企业的网络团队经常会面临铺天盖地的配置请求，而处理这些配置请求可能需要几天或几个星期的时间。所幸的是，这些网络团队可以利用一些新出现的技术来帮助他们解决这些问题，并提高网络的灵活行。在这些方法中，最主要的是网络虚拟化[注]（NV）、网络功能虚拟化[注]（NFV[注]）和软件定义网络[注]（SDN[注]）。