Langeldep的专栏

个人资料

langeldep

访问： 630707次
积分： 6940
等级： BLOG 6
排名： 第990名

原创： 112篇 转载： 44篇
译文： 2篇 评论： 457条

文章搜索

文章分类

Linux操作与使用 (37)
人生感悟---程序与人生 (4)
学习C/C++经验 (26)
嵌入式开发 (14)
数据库开发 (5)
流媒体开发 (12)

文章存档

2014年09月 (1)
2014年08月 (7)
2014年04月 (2)
2014年02月 (1)
2013年12月 (1)

展开

IP组播技术介绍及实现例子

2011-01-27 18:47 10378人阅读 评论

路由器 网络 internet socket struct interface

引 言

近年来，随着Internet的迅速普及和爆炸性发展，在Internet上产生了许多新的应用，其中不少是高带宽的多媒体应用，譬如网 络视频会议、网络音频/视频广播、AOD/VOD、股市行情发布、多媒体远程教育、CSCW协同计算、远程会诊。这就带来了带宽的急剧消耗和网络拥挤问 题。为了缓解网络瓶颈，人们提出各种方案，归纳起来，主要包括以下四种：

●增加互连带宽；

●服务器的分散与集群，以改变网络流量结构，减轻主干网的瓶颈；

●应用QoS机制，把带宽分配给一部分应用；

●采用IP Multicast(译为组播、多播或多路广播，下文不加区分)技术。

比较而言，IP组播技术有其独特的优越性——在组播网络中，即使用户数量成倍增长，主干带宽不需要随之增加。这个优点使它成为当前网络技术中的研究热点之一。

本文简单介绍了组播的发展、分析了组播网络的体系结构、算法和协议，讨论了组播技术的应用，总结了组播技术的难点，希望通过本文能使读者对组播技术有总体的了解。

一、IP组播发展简史

20 世纪80年代中期，斯坦福大学的博士生S. E. Deering发表Host group: A multicast extension to the Internet Protocol (RFC0966) 和Host extensions for IP Multicasting (RFC0988) 两篇论文。他总结出：“OSPF的链路状态机制完全能被扩展用来支持组播……，RIP的基本机制能被用来作为一种新的距离向量的组播路由协议的基础。”这 些论断提出了IP组播的可能性。

1988年，D. Waltzman, C. Portridge, S. E. Deering发表题为《距离向量组播路由协议》的文章（RFC1075），它是组播路由协议的首次实践；

1991年12月，S. E. Deering发表了他的博士论文《数据报互连网络中的组播路由》（RFC1112）。它奠定了组播网络体系结构和路由协议的基础。该文也成为Internet组管理协议（IGMP）的原型；

1994年3月，形成了对OSPF协议的扩展协议MOSPF（RFC1584）；

1996年11月，出现了对于基于UNI3.0/3.1的ATM组播网络支持协议（RFC2022）；

1997年9月，有核树（CBTv2）组播路由体系结构形成（RFC2189）；

1997年11月，组管理协议IGMPv2得到IETF的批准，成为标准（RFC2336）；

1998年6月，评估可靠组播传输协议RMTP的IETF标准出台（RFC2357）；

1998年7月，在制定IPv6地址体系标准时，确定IPv6组播地址分配方案（RFC2373），这为组播技术在下一代Internet上的应用做出了必要的准备；

1999年10月，Cisco、AT&T、Microsoft制定组播地址动态客户分配协议MADCAP（RFC2730）；

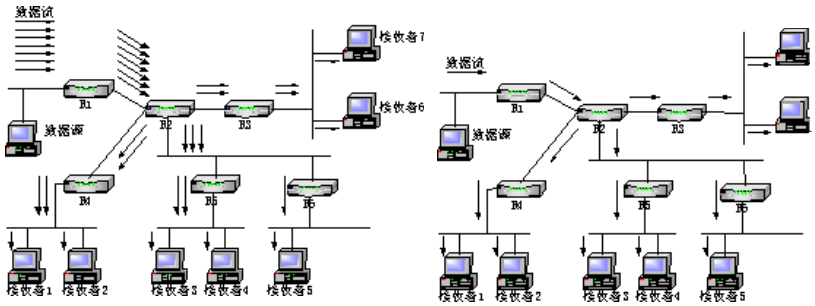
2000年底2001年初，人们着手制定各种组播MIB库，这标志组播技术正向可管理、可控制方向发展。

二、组播网络的体系结构

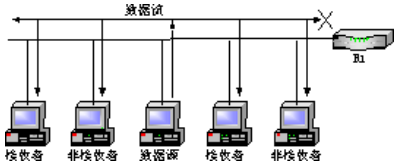
组播网络体系结构包括：组播的基本工作原理、实现组播的条件、组播的地址分配方案及与MAC地址映射、Internet组管理协议。

2.1组播的工作原理

组 播是一种允许一个或多个发送者（组播源）发送单一的数据包到多个接收者（一次的，同时的）的网络技术。组播源把数据包发送到特定组播组，而只有属于该组播 组的地址才能接收到数据包。组播可以大大的节省网络带宽，因为无论有多少个目标地址，在整个网络的任何一条链路上只传送单一的数据包。图1 为基于三种通讯方式的网络结构和数据传递过程。



A: 单播方式实现组播功能　　B: 组播方式



C: 广播实现组播功能

图 1 三种通讯方式数据传递过程比较

从图1的工作方式可以看出：

单播（Unicast）传输：在发送者和每一接收者之间需要单独的数据信道。如果一台主机同时给很少量的接收者传输数据，一般没有什么问题。但如果有大量主机希望获得数据包的同一份拷贝时却很难实现。这将导致发送者负担沉重、延迟长、网络拥塞；为保证一定的服务 质量需增加硬件和带宽。

组播（Multicast）传输：它提高了数据传送效率。减少了主干网出现拥塞的可能性。组播组中的主机可以是在同一个物理网络，也可以来自不同的物理网络（如果有组播路由器的支持）。

广 播（Broadcast）传输：是指在IP子网内广播数据包，所有在子网内部的主机都将收到这些数据包。广播意味着网络向子网主机都投递一份数据包，不论 这些主机是否乐于接收该数据包。然而广播的使用范围非常小，只在本地子网内有效，因为路由器会封锁广播通信。广播传输增加非接收者的开销。

2.2 实现IP组播的前提条件

实现IP组播传输，则组播源和接收者以及两者之间的下层网络都必须支持组播。这包括以下几方面：

●主机的TCP/IP实现支持发送和接收IP组播；

●主机的网络接口支持组播；

●有一套用于加入、离开、查询的组管理协议，即IGMP（v1,v2）；

●有一套IP地址分配策略，并能将第三层IP组播地址映射到第二层MAC地址；

●支持IP组播的应用软件；

●所有介于组播源和接收者之间的路由器、集线器、交换机、TCP/IP栈、防火墙均需支持组播；

目 前，IP组播技术得到硬件、软件厂商的广泛支持，比如，新生产的以太网卡几乎都支持组播；Cisco的路由

器不仅支持DVMRP、PIM路由协议、 IGMP组管理协议，而且支持Cisco专有Cisco组管理协议CGMP，再如微软的Windows 95支持IP组播和IGMPv1，而Windows 98还支持IGMPv2。对于不支持IP组播传输的中间路由器采用IP隧道（Tunneling）技术作为过渡方案。这些说明IP组播技术的应用环境已基 本具备。

2.3 组播地址分配与MAC地址

在组播通信中，我们需要两种地址：一个IP组播地址和一个Ethernet组播地址。其 中，IP组播地址标识一个组播组。由于所有IP数据包都封装在Ethernet帧中，所以还需要一个组播Ethernet地址。为使组播正常工作，主机应 能同时接收单播和组播数据，这意味着主机需要多个IP和Ethernet地址。

IP地址方案专门为组播划出一个地址范围，在IPv4中为D类地 址，范围是224.0.0.0到239.255.255.255，并将D类地址划分为局部链接组播地址、预留组播地址、管理权限组播地址；在IPv6中为 组播地址提供了许多新的标识功能，图2为IPv4和IPv6的组播地址格式，其中IPv6中特殊域的定义见表1。

域	值	含义
flags	0000	永久组播地址
	0001	动态组播地址
scope	0001	本地结点
	0010	本地链路
	0101	本地网点
	1000	本地组织
	1110	全局组播地址
	其它	保留或未指定

图 2 IPv4 和 IPv6 的组播地址格式

表 1 IPv6 中特殊域的定义

局部链接地址：224.0.0.0～224.0.0.255，用于局域网，路由器不转发属于此范围的IP包；

预留组播地址：224.0.1.0～238.255.255.255，用于全球范围或网络协议；

管理权限地址：239.0.0.0～239.255.255.255，组织内部使用，用于限制组播范围；

2.3.1 以太网与FDDI组播MAC地址映射

IP 组播帧都使用以0X0100.5EXX.XXXX的24位前缀开始的MAC层地址，但只有其中的一半MAC地址可以被IP组播使用，剩下的MAC地址空间 的23位作为第三层IP组播地址进入第二层MAC地址的映射使用。由于第三层IP组播的28位地址不能映射到只有23位的可用MAC地址空间，造成有 32:1的地址不明确，所以主机CPU必须对收到的每一个组播数据包做出判断。这增加了主机CPU的开销。此外，还产生抑制第二层局域网交换的组播扩散问 题。

2.3.2 令牌环网组播MAC地址映射

令牌环网MAC地址格式与标准以太网MAC地址格式位序相反。令牌环网的缺点是其功能地 址位使得它对组播地址映射的不明确性高达228:1，这意味着令牌环网上的组播数据流将导致令牌环网点的CPU被环路上的每一个组播数据包中断，从这个角 度来说，令牌环网不适合于组播。

2.4 组播树

在单播模型中，数据包通过网络沿着单一路径从源主机向目标主机传递，但在组播模型中，组播源向某一组地址传递数据包，而这一地址却代表一个主机组。为了向所有接收者传递数据，一般采用组播分布树描述IP组播在网络里经过的路径。

组播分布树有四种基本类型：泛洪法、有源树、有核树和Steiner树。

2.4.1 洪泛法（Flooding）

这是最简单的向前传送组播路由算法，并不构造所谓的分布树。其基本原理如下：当组播路由器收到发往某个组播地址的数据包后，首先判断是否是首次收到该数据包，如果是首次收到，那么将其转发到所有接口上，以确保其最终能到达所有接收者；如果不是首次收到，则抛弃该数据包。

洪 泛法的实现关键是“首次收到”的检测。这需要维护一个最近通过的数据包列表，但无需维护路由表。它适合于对组播需求比较高的场合，并且能做到即使传输出现 错误，只要还存在一条到接收者的链路，则所有接收者都能接收到组播数据包。然而，洪泛法不适合用于Internet，因为它不考虑链路状态，并产生大量的 拷贝数据包。此外，对于高速网络而言，“首次收到”列表将会很长，占用相当大的内存；尽管它能保证不对相同的数据包进行二次转发，但不能保证对相同数据包 只接收一次。

2.4.2 有源树

有源树也称为基于信源的树或最短路径树（Shortest Path Tree：SPT）。它是以组播源为根构造的从根到所有接收者路径都最短的分布树。如果组中有多个组播源，则必须为每个组播源构造一棵组播树。由于不同组

播源发出的数据包被分散到各自分离的组播树上，因此采用SPT有利于网络中数据流量的均衡。同时，因为从组播源到每个接收者的路径最短，所以端到端 （end-to-end）的时延性能较好，有利于流量大、时延性能要求较高的实时媒体应用。SPT的缺点是：要为每个组播源构造各自的分布树，当数据流量 不大时，构造SPT的开销相对较大。

2.4.3 共享树

共享树也称RP树（RPT），是指为每个组播组选定一个共用根（汇合点RP或 核心），以RP为根建立的组播树。同一组播组的组播源将所要组播的数据单播到RP，再由RP向其它成员转发。目前，讨论最多同时也是最具代表性的两种共享 树是Steiner树和有核树（CBT）。

Steiner树是总代价最小的分布树，它使连接特定图（graph）中的特定组成员所需的链路数最 少。若考虑资源总量被大量的组使用的情况，那么使用资源较少最终就会减少产生拥塞的风险。Steiner树相当不稳定，树的形状随组中成员关系的改变而改 变，且对大型网络缺少通用的解决方案。所以Steiner树只是一种理论模型，而非实用工具。目前，出现了许多Steiner树的次优启发式生成算法。

有 核树是由根到所有组成员的最短路径合并而成的树。A. Ballardie在1997年9月的《基于核的组播路由结构》（Core Based Trees (CBT) Multicast Routing Architecture）(RFC2189和RFC2201)中介绍了有核树。关于它的进一步讨论见下文。

共享树在路由器所需存储的状态信息的数量和路由树的总代价两个方面具有较好的性能。当组的规模较大，而每个成员的数据发送率较低时，使用共享树比较适合。但当通信量大时，使用共享树将导致流量集中及根（RP）附近的瓶颈。

2.5 组管理协议IGMP

主机使用IGMP通知子网组播路由器，希望加入组播组；路由器使用IGMP查询本地子网中是否有属于某个组播组的主机。

●加入组播组

当 某个主机加入某一个组播组时，它通过“成员资格报告”消息通知它所在的IP子网的组播路由器，同时将自己的IP模块做相应的准备，以便开始接收来自该组播 组传来的数据。如果这台主机是它所在的IP子网中第一台加入该组播组的主机，通过路由信息的交换，组播路由器加入组播分布树。

●退出组播组

在IGMP v1中，当主机离开某一个组播组时，它将自行退出。组播路由器定时(如120秒) 使用“成员资格查询”消息向IP子网中的所有主机的组地址（224.0.0.1）查询，如果某一组播组在IP子网中已经没有任何成员，那么组播路由器在确认这一事件后，将不再 在子网中转发该组播组的数据。与此同时，通过路由信息交换，从特定的组播组分布树中删除相应的组播路由器。这种不通知任何人而悄悄离开的方法，使得组播路 由器知道IP子网中已经没有任何成员的事件延时了一段时间，所以在IGMP v2.0中，当每一个主机离开某一个组播组时，需要通知子网组播路由器，组播路由器立即向IP子网中的所有组播组询问，从而减少了系统处理停止组播的延时。

三、组播转发

由于组播源是向组播组发送数据包而非单播模型中的具体目标主机，所以组播路由器不能依靠IP包中的目标地址来决定如何转发数 据包，而必须将组播数据包转发到多个外部接口上，以便同一组播组的成员都能接收到数据包。这使组播转发比单播转发更加复杂。大多数现有组播路由协议使用逆 向路径转发（RPF）机制作为组播转发的基础。

3.1 逆向路径转发(Reverse Path Forward: RPF)

当组播数据包到达路由器时，路由器作RPF检查，以决定是否转发或抛弃该数据包，若成功则转发，否则抛弃。

RPF检查过程如下：

●检查数据包的源地址，以确定该数据包经过的接口，是否在从源到此的路径上；

●若数据包是从可返回源主机的接口上到达，则RPF检查成功，转发该数据包到输出接口表上的所有接口，否则RPF检查失败，抛弃该数据包。

3.2 组播转发缓存

对于每一个输入组播数据包进行RPF检查会导致较大的路由器性能损失。因此，建立组播转发缓存时，通常由组播路由确定RPF接口。然后将RPF接口变成组播转发缓存项的输入接口。一旦RPF检查程序使用的路由表发

生变化，必须重新计算RPF接口；并更新组播转发缓存项。

3.3 TTL阈值

每 当路由器转发组播数据包，IP包中的TTL（Time To Live）值都减1。若数据包的TTL减少到0，则路由器将抛弃该数据包。TTL阈值可用于组播路由器的各个接口，以防止在该接口上转发低于TTL阈值的 组播数据包。这样可对组播的范围加以控制。表2给出典型的初始TTL值和作为不同TTL边界的路由器接口TTL阈值。

表2 典型的TTL边界值

范 围	初始TTL值	TTL阈值
本地网	1	N/A
区域	15	16
地区	63	64
全球	127	128

3.4 管理权限边界

除TTL阈值外，组播提供另一种称为管理权限的地址机制作为边界，以限制组播信息转发到域外。管理权限的组播地址是 从239.0.0.0到239.255.255.255，这段地址被认为是本地分配(类似于单播中的192.168.xx.xx)，不能用于 Internet。这种机制使得在Intranet内部可重复使用组播地址，提高组播地址空间的利用率。

四、组播路由协议

要想在一个实际网络中实现组播数据包的转发，必须在各个互连设备上运行可互操作的 组播路由协议。组播路由协议可分为三类：密集模式协议（如DVMRP，PIM-DM）、稀疏模式协议（如PIM-SM，CBT）和链路状态协议 （MOSPF），下面分别介绍各个协议的工作原理。

4.1 距离向量组播路由协议（Distance Vector Multicast Routing Protocol：DVMRP）

DVMRP 由单播路由协议RIP扩展而来，两者都使用距离向量算法得到网络的拓扑信息，不同之处在于RIP根据路由表前向转发数据，而DVMRP则是基于RPF。为 了使新加入的组播成员能及时收到组播数据，DVMPR采用定时发送数据包给所有的LAN的方法，然而这种方法导致大量路由控制数据包的扩散，这部分开销限 制了网络规模的扩大。另一方面，DVMRP使用跳数作为计量尺度，其上限为32跳，这对网络规模也是一个限制。目前提出了分层DVMRP，即对组播网络划 分区域，在区域内的组播可以按照任何协议进行，而对于跨区域的组播则由边界路由器在DVMRP协议下进行，这样可大大减少路由开销。

4.2 开放式组播最短路径优先协议（Multicast Open Shortest Path First：MOSPF）

MOSPF是一种基于链路状态的路由协议，是对单播OSPF协议的扩展。

同OSPF类似，MOSPF定义了三种级别的路由：

●MOSPF区域内组播路由：用于了解各网段中的组播成员，构造（源网络S，组G）对的SPT；

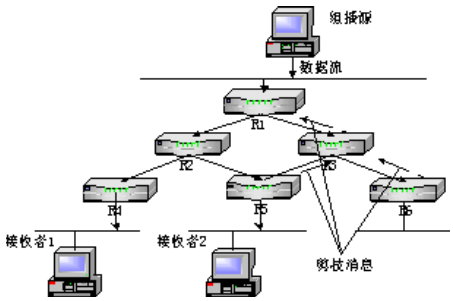
●MOSPF区域间组播路由：用于汇总区域内成员关系，并在自治系统（AS）主干网（区域0）上发布组成员关系记录通告，实现区域间组播包的转发。

●MOSPF AS 间组播路由：用于跨AS的组播包转发。

区 域内MOFPF利用了链路状态数据库，对单播OSPF数据格式进行扩充，定义了新的链路状态通告（Link State Advertisement：LSA），使得MOSPF路由器了解哪些组播组在哪些网络上。路由器使用Dijkstra算法构造（源网络S，组G）对的 SPT。MOSPF与DVMRP相比，路由开销较小，链路利用率高，然而Dijkstra算法计算量很大，为了减少路由器的计算量，MOSPF执行一种按 需计算方案，即只有当路由器收到组播源的第一个组播数据包后，才对（S，G）SPT计算，否则利用转发缓存（cache）中的（S，G）SPT。
MOSPF 继承了OSPF对网络拓扑的变化响应速度快的优点，但拓扑变动使所有路由器的缓存失效重新计算SPT，因而消耗大量路由器CPU资源。这就决定了 MOSPF不适合高动态性网络（组成员关系变化大、链路不稳定），而适用于网络连接状态比较稳定的环境。另外，对于有大量组播源子网络的网络而 言，MOSPF的扩展性问题引起了人们的关注，有待于进一步研究。

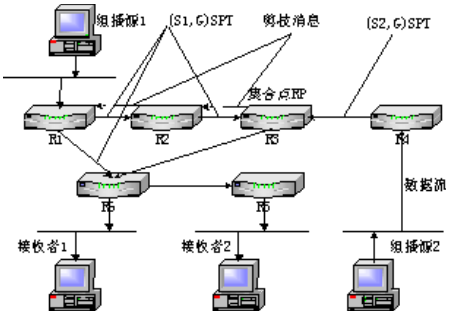
4.3 协议无关组播（Protocol Independent Multicast：PIM）

PIM 由IDMR（域间组播路由）工作组设计，顾名思义，PIM不依赖于某一特定单播路由协议，它[illegible]
单播路由协议建立的单播路由表完成RPF检查功能， 而不是维护一个分离的组播路由表实现组播转发。由于PIM无需收发组播路由更新，所以与其它组播协议相比，PIM开销降低了许多。PIM的设计出发点
范围内同时支持SPT和共享树，并使两者之间灵活转换，因而集中了它们的优点提高了组播效率。PIM定义了两种模式：密集模式 (Dense-Mode)和稀疏模式（Sparse-Mode）
4.3.1 PIM-DM
PIM-DM与DVMRP很相似，都属于密集模式协议，都采用了“扩散/剪枝”机制。同时，假定带宽不受限制，每个路由器都想接收组播数据包。主要不同之处在于DVMRP使用内建的组播路由协议，而PIM-DM采用RPF动态建立SPT。



该模式适合于下述几种情况：高速网络；组播源和接收者比较靠近，发送者少，接收者多；组播数据流比较大且比较稳定。

4.3.2 PIM-SM
PIM-SM与基于“扩散/剪枝”模型的根本差别在于PIM-SM是基于显式加入模型，即接收者向RP发送加入消息，而路由器只在已加入某个组播组输出接口上转发那个组播组的数据包。
PIM- SM采用共享树进行组播数据包转发。每一个组有一个汇合点（Rendezvous Point: RP），组播源沿最短路径向RP发送数据，再由RP 沿最短路径将数据发送到各个接收端。这一点类似于CBT，但PIM-SM不使用核的概念。PIM-SM主要优势之一是它不局限于通过共享树接收组播信息， 还提供从共享树向SPT转换的机制。尽管从共享树向SPT转换减少了网络延迟以及在RP上可能出现的阻塞，但这种转换耗费了相当的路由器资源，所以它适用于有多对组播数据源和网络组数目较少的环境。



4.4 有核树组播路由协议（Core-Based Trees: CBT）

CBT 的基本目标是减少网络中路由器组播状态，以提供组播的可扩展性。为此，CBT被设计成稀疏模式（与PIM-SM相似）。CBT使用双向共享树，双向共享树 以某个核心路由器为根，允许组播信息在两个方向流动。这一点与PIM-SM不同（PIM-SM中共享树是单向的，在RP与组播源之间使用SPT将组播数据 转发到RP），所以CBT不能使用RPF检查，而使用IP包头的目标组地址作检查转发缓存。这就要求对CBT共享树的维护就需非常小心，以确保不会产生组 播路由循环。

从路由器创建的组播状态的数量来看， CBT比支持SPT的协议效率高，在具有大量组播源和组的网络中，CBT能把组播状态优化到组的数量级。

CBT为每个组播组建立一个生成树，所有组播源使用同一棵组播树。CBT工作过程大体如下：

●首先选择一个核，即网络中组播组的固定中心，来构造一棵CBT；

●主机向这个核发送join命令；

●所有中间路由器都接收到该命令，并把接收该命令的接口标记为属于这个组的树；

●如果接收到命令的路由器已是树中一个成员，那么只要再标记一次该接口属于该组；如果路由器join命令，那么它就向核的方向进一步转发该命令，路由器就需要为每个组保留一份状态信息；

●当组播数据到达一个在CBT树上的组播路由器时，路由器组播数据到树的核。以保证数据能够发送到组的所有成员。

CBT将组播扩张限制在接收者范围内，即使第一个数据包也无需在全网扩散，但CBT导致核周围的流量集中，网络性能下降。所以某些版本的CBT支持多个核心以平衡负载。

目前CBTv3草案已公布。该方案通过使用CBT边界路由器（BR）更好地处理域间组播的转发。CBTv3还引入新的状态及单向分支CBT概念。尽管CBT很有代表性，但至今却几乎没有已实现的CBT网络。

五、组播应用与编程

组播技术被认为是WWW技术推广之后出现的最激动人心的网络技术之一。 1992年出现支持IP组播的Mbone(组播主干网)和Mbone桌面工具；1993-1996年IP Multicast成为业界关注的焦点，然而因发展条件不成熟使得IP组播只为业界所关注；进入1999年以来，IP组播具备了发展的三个关键条件：支持 组播的路由协议；基于开放标准的可测试管理协议；因商业发展机遇而进入高速发展阶段。又一次掀起了组播实践的高潮，下面将有关组播应用作简单讨论：

5.1 组播主干网(Multicast Backbone：Mbone)

Mbone是一个由IETF开发的运行在Internet上的虚拟重叠网络。Mbone的初衷是创建一个半永久的IP组播测试床而不需要等到整个Internet都采用支持组播的路由器。

Mbone跨越几个洲，用户数大约在10000-30000之间。在IETF会议期间，大约有1000个不同的接收主机接入。它成为Internet上传送声音和视频信息的一个重要组成部分。

1992年，组播技术还处于实验阶段。当时提出以IP遂道（Tunneling）联结组播岛，组播岛是支持组播服务的区域，最小的组播岛是一个支持组播的LAN。

Mbone 使用DVMRP协议，而DVMRP在UNIX下是由标准守护进程mrouted得以实现，所以许多用户使用UNIX主机接入Mbone。由于UNIX主机 上的I/O处理能力、对IP遂道的处理能力、网络接口数量等方面都不及商用路由器，这都无形制约了Mbone的发展。

Mbone自从出现就不断发展。今天，从基于mrouted的UNIX主机到商用路由器的迁移已超过了50%；Mbone也采用剪枝、封装等技术。新的域间组播路由协议和转发算法、流量控制与管理、可靠组播也将对Mbone产生影响。

5.2 组播应用程序接口与编程

RFC1112推荐了一些支持组播的应用程序接口：

●加入一个组播组；

●离开一个组播组；

●为调整范围对一个组播数据的IP TTL值进行设定；
●为组播传输和接收设定本地的接口；
●禁止输出的组播数据回送。
现在，许多TCP/IP实现都支持RFC1112所提到的要求，下面简要介绍UNIX(Berkeley Socket)和Windows(Winsock) API。
5.2.1 Berkeley Socket组播API
所 有Berkeley Socket API都采用setsockopt()的“套接字选项”功能来设置（对于某些选项，getsockopt()功能可用来获得当前的设置）。表3描述了 Berkeley BSD的set sockopt()/getsockopt()组播命令。
表3 BSD setsockopt()/getsockopt()组播命令的说明

setsockopt()/getsockopt() 组播命令	命令说明
IP_MULTICAST_TTL	设置输出组播数据的 TTL 值
IP_ADD_MEMBERSHIP	在指定接口上加入组播组
IP_DROP_MEMBERSHIP	退出组播组（在 IGMPv2 中实现）
IP_MULTICAST_IF	获取默认接口或设置接口
IP_MULTICAST_LOOP	禁止组播数据回送

对于套接字编程，首先要使用函数socket()建立一个数据包套接字，然后用bind()函数将套接字与一个地址和端口号连接起来。
为了发送一个组播数据包，需要在sendto()调用中指定一个组播地址作为目的地址（所有IP地址都使用网络字节顺序）。
为了接收一个组播数据包，需要在recvfrom()调用中指定所要接收的组播地址。
IP_MULTICAST_TTL允许将随后的组播数据的TTL设定成从0到255之间的任何值，例如：

```
u_char ttl;
setsockopt(sock,IPPROTO_IP,IP_MULTICAST_TTL,&ttl,sizeof(ttl));
```

关于TTL的讨论见上文。
通过IP_MULTICAST_IF,系统管理员可在安装的时候为组播创建默认的接口（为从一个给定的网络接口并发传送，一个网络接口会忽略这个默认值）。例如：

```
struct in_addr addr;
setsockopt(sock,IPPROTO_IP,IP_MULTICAST_IF,&addr,sizeof(addr));
```

在这里，addr是希望输出接口的本地IP地址，可使用一个INADDR_ANY地址来回送到默认的接口。
当组播组中的一台主机发送组播数据到输出接口时，默认的IP层将为本地回送数据的拷贝。
IP_MULTICAST_LOOP网络参数控制IP层是否回送所送的数据。例如：

```
u_char loop;
setsockopt(sock,IPPROTO_IP,IP_MULTICAST_LOOP,&loop,sizeof(loop));
```

将loop设置为0则禁止回送，设置为1则允许回送。
为了能够接收IP组播数据，主机必须加入某个或多个组播组，程序通过使用IP_ADD_MEMBERSHIP网络接口参数向主机提出加入组播组的申请。例如：

```
struct ip_mreq
{struct in_addr imn_multiaddr; /* multicast group to join */
struct in_addr imr_interface; /* interface to join on */
}mreq;
```

setsockopt(sock,IPPROTO_IP,IP_ADD_MEMBERSHIP,&mreq,sizeof(mreq));
一个组的成员是与一个单一的网络接口相联系；主机可在不止一个网络接口上加入相同的组。若选择默认组播接口，要将imr_interface设置为INADDR_ANY;若选择主机其中一个本地地址，要将imr_interface设置为特定的组播接口。
若撤消一个成员资格，使用IP_DROP_MEMBERSHIP
struct ip_mreq mreq;
setsockopt(sock,IPPROTP_IP,IP_DROP_MEMBERSHIP,&mreq,sizeof(sreq));
其中mreq包含了在IP_ADD_MEMBERSHIP命令中相同的值。
5.2.2 Windows Socket组播API
基于Winsock1.1的组播编程与Berkeley Socket类似，这里不再赘述。Winsock2是Winsock1.1的扩展，除兼容Berkeley Sockets组播API外，它还定义了一套支持IP组播的协议独立API，如表4所示：
表4 WinSock 2的协议独立组播API说明

WSAEnum Protocol()	获得协议信息结构 (WASPROTOCOL_INFO)
WSASocket()	设置组播类型
WSAJoinLeaf	加入组播组并指定角色 (发送者 / 接收者)
WSAIoctl(…SIO_MULTICAST_SCOPE…)	设置 IP TTL
WSAIoctl(…SIO_MULTICAST_LOOPBACK…)	禁止组播数据回送

在 Winsock2中，定义了“数据平面”（Data Plane）和“控制平面”(Control Plane)的概念，其中，数据平面决定在不同的网络成员之间数据如何传送；控制平面定义网络成员的组织方式； 这两方面的特征既可以是“有根的”（Rooted）,也可以是“无根的”（Nonrooted）。在“有根的”控制平面内，存在一个特殊的组播组成员，称 作C_root（根节点），其余的组成员称作C_leaf(叶节点)。对“无根的”控制平面而言，只存在叶节点。
在“有根的”平面中，根节点负责组播的建立，以及同任意数量叶节点的连接。叶节点可申请加入一个特定的组播组。数据传送只能在根节点和叶节点之间进行，根节点将数据组播到每个叶节点。
在“无根的”平面中，只存在叶节点，它们可以任意加入一个组播组。从叶节点发送的数据会组播到每一个叶节点。
由于篇幅所限，有关Winsock API的进一步讨论，请参阅参考文献<3>、<5>和MSDN。

六、**IP**组播中存在的问题与发展

6.1 组播的可靠性
IP组播数据包典型使用用户数据报协议（UDP），而UDP是一种“尽力而为”（Best-effort）协议。因此，IP组播应用必定会遇到数据包丢失和乱序问题。
从端到端传输延迟和可靠性方面考虑，组播应用可大致分为三类：
1）实时交互应用，如视频会议系统，这类应用对可靠性要求相对较低，但对端到端传输延迟和网络抖动的要求很高。
2）实时非交互型应用，如数据广播，这类应用传输延迟要求相对前一类应用较低，但在一定延迟范围内，却对可靠性提出更高要求。
3）非实时应用，如软件分发，这类应用中，可靠性是最基本的要求，在满足可靠性要求的前提下，必须保证传输延迟在可接受的范围之内。
对于不同类型的应用必须在确认方式（肯定确认ACK和否定确认NACK），集中确认与分布确认、重传机制、重传范围、流量控制、拥塞控制、end-to-end延迟和广播延迟、网络抖动、可伸缩性与网络的异构性等方面做

出综合考虑，提出相应的解决办法。

迄 今为止，尽管在广域网环境中已经存在许多可靠组播协议，包括可靠组播协议RMP（Reliable Multicast Protocol）、可扩展可靠组播SRM(Scalable Reliable Multicast)、基于日志的可靠组播LBRM（Log-Based Reliable Multicast）和可靠组播传输协议RMTP(Reliable Multicast Transport Protocol)，但组播的可靠性研究仍然是国际上的重点研究课题之一。

6.2 组播的安全性

安全组播就是只有注册的发送者才可以向组发送数据；只有注册的接收者才可以接收组播数据。然而IP组播很难保证这一点。

首 先，IP组播使用UDP，任何主机都可以向某个组播地址发送UDP包，并且低层组播机构将传送这些UDP包到所有组成员。其次，Internet缺少对于 网络层的访问控制。第三，组成员可以随时加入/退出组播组。这几点使组播安全性问题同组播的可靠性问题一样难以解决。

总的来说，安全组播可分为 集中式和分布（分层）式密钥管理体系。目前，对于组播安全性问题[一]一[一]钥管理、Iolus、Nortel框架和SRM（Secure Reliable Multicast）等解决方案。Matthew J. Mayer等人在<29>中提出了安全组播评估标准，回顾并讨论了安全组播体系结构、组密钥管理和信源认证等问题。

决方案都不同 程度的存在不足，安全组播仍然是一个技术难点。

6.3 网络的异构性导致组播的复杂性

Internet是一个异构网络，这种 异构性表现在很多方面。第一，Internet的低层硬件平台千差万别，可以是Ethernet、ATM、FDDI、令牌环网、帧中继、串行链路 （PSTN、xDSL）、无线网络、卫星网络、移动网络等等。这些低层网络具有不同的带宽、硬件存取控制方式、时延特征。在多链路情况下，各链路的带宽与 代价也可能不同。另外，某些网络平台的数据链路具有非对称性，比如xDSL和卫星网络。第二，主机的硬件处理能力和操作系统各不相同。就操作系统而言，主 要的操作系统，如UNIX、Windows、MacOS、OS2有不同的变种和版本，对IP组播的支持程度、进程的调度与管理、TCP/IP的实现方式和 API都有差异。第三，互连设备的差异。路由器、交换机、网络服务器在背板能力、包转发率、支持的路由协议的互操作性。这些异构性都导致在实现IP组播网 络中的复杂性。

比如：网络中不同部分的带宽不同、接收者的处理要求和处理能力不同，而所有接收者都要与同一组播源交互，这就要求采取某些方法使 得每一个接收者接收到与其接收能力和从组播源到接收者之间带宽相适合的数据流（即公平性）。再比如：ATM面向连接的特点在IP组播传输中带来了新的问 题，这使IP组播与ATM组播具有不同特点。

所以，在设计IP组播网络时，必须充分考虑到网络的异构性。

七、结束语

本文从组播的产生和发展出发，介绍了组播网络的体系结构、算法和协议，讨论了组播技术的应用，总结了组播技术的难点。随着高宽带多媒体应用的迫切需求、ISP、ICP对IP组播网络的支持、设备提供商的投入、各种专业组织的介入，IP组播技术必然有着广阔的发展前景。

下面的代码为Sender.cpp 文件代码

```cpp
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <time.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#define HELLO_PORT 12345
#define HELLO_GROUP "225.0.0.37"
int main(int argc, char *argv[])
{
```

```
    struct sockaddr_in addr;
    int fd, cnt;
    struct ip_mreq mreq;
    char *message="Hello, World!";
    /* create what looks like an ordinary UDP socket */
    if ((fd=socket(AF_INET,SOCK_DGRAM,0)) < 0)
    {
        perror("socket");
        exit(1);
    }
    /* set up destination address */
    memset(&addr,0,sizeof(addr));
    addr.sin_family=AF_INET;
    addr.sin_addr.s_addr=inet_addr(HELLO_GROUP);
    addr.sin_port=htons(HELLO_PORT);
    /* now just sendto() our destination! */
    while (1)
    {
        if (sendto(fd,message, strlen(message), 0, (struct sockaddr *) &addr,
        {
            perror("sendto");
            exit(1);
        }
        sleep(1);
    }
    return 0;
}
```

下面的代码为 Recver.cpp代码

```
[cpp]

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <time.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#define HELLO_PORT  12345
#define HELLO_GROUP "225.0.0.37"
#define MSGBUFSIZE 256
int main(int argc, char *argv[])
{
    struct sockaddr_in addr;
    int fd, nbytes,addrlen;
    struct ip_mreq mreq;
    char msgbuf[MSGBUFSIZE];
    u_int yes=1; /*** MODIFICATION TO ORIGINAL */
    /* create what looks like an ordinary UDP socket */
    if ((fd=socket(AF_INET,SOCK_DGRAM,0)) < 0)
    {
        perror("socket");
        exit(1);
    }
    /**** MODIFICATION TO ORIGINAL */
    /* allow multiple sockets to use the same PORT number */
    if (setsockopt(fd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(yes)) < 0)
    {
        perror("Reusing ADDR failed");
        exit(1);
    }
    /*** END OF MODIFICATION TO ORIGINAL */
    /* set up destination address */
    memset(&addr,0,sizeof(addr));
```

```
    addr.sin_family=AF_INET;
    addr.sin_addr.s_addr=htonl(INADDR_ANY); /* N.B.: differs from sender */
    addr.sin_port=htons(HELLO_PORT);
    /* bind to receive address */
    if (bind(fd,(struct sockaddr *) &addr,sizeof(addr)) < 0)
    {
        perror("bind");
        exit(1);
    }
    /* use setsockopt() to request that the kernel join a multicast group */
    mreq.imr_multiaddr.s_addr=inet_addr(HELLO_GROUP);
    mreq.imr_interface.s_addr=htonl(INADDR_ANY);
    if (setsockopt(fd,IPPROTO_IP,IP_ADD_MEMBERSHIP,&mreq,sizeof(mreq)) < 0)
    {
        perror("setsockopt");
        exit(1);
    }
    /* now just enter a read-print loop */
    while (1)
    {
        //ssize_t recvfrom(int s, void *buf, size_t len, int flags, struct so
        addrlen=sizeof(addr);
        if ((nbytes=recvfrom(fd, msgbuf, MSGBUFSIZE, 0, (struct sockaddr *) &addr, (sockle
        {
            perror("recvfrom");
            exit(1);
        }
        puts(msgbuf);
    }
    return 0;
}
```

下面为 编译 sender 和 recver 的Makefile 文件

```
[c-sharp]

CC = gcc
CXX = g++
CFLAGS = -Wall -pipe -D_DEBUG -DDEBUG -g -O0
LDFLAGS = -lstdc++
RM = /bin/rm -f
MODULE_INC = -I../curl-7.21.3/include -I../boost_1_45_0 -I./
MODULE_LIB = -L../boost_1_45_0/stage/lib
CFLAGS  += $(MODULE_INC)
LDFLAGS += $(MODULE_LIB)
LIBOBJS = Sender.o Recver.o
TARGET = sender recver
all: $(TARGET)
sender: Sender.o
    $(CXX) -o $@ $^ $(LDFLAGS)
recver: Recver.o
    $(CXX) -o $@ $^ $(LDFLAGS)
clean:
    rm -f *.o
    rm -f $(TARGET)
# make rule
%.o : %.c
    $(CC) $(CFLAGS) -c $^ -o $@
%.o : %.cpp
    $(CC) $(CFLAGS) -c $^ -o $@
install:
    cp -f $(TARGET) ../bin/
```

编译好后就可以执行 ./sender 和 ./recver 查看执行的结果。

Time-To-Live (TTL) for Multicast Packets

The IP multicast routing protocol uses the **Time To Live (TTL)** field of IP datagrams to decide how "far" from a sending host a given multicast packet should be forwarded. The default TTL for multicast datagrams is 1, which will result in multicast packets going only to other hosts on the local network. A **setsockopt** (2) call may be used to change the TTL:

```
unsigned char ttl;
```

```
setsockopt(sock,IPPROTO_IP,IP_MULTICAST_TTL,&ttl,sizeof(ttl));
```

As the values of the TTL field increase, routers will expand the number of hops they will forward a multicast packet. To provide meaningful scope control, multicast routers enforce the following "thresholds" on forwarding based on the TTL field:

0
: restricted to the same host

1
: restricted to the same subnet

32
: restricted to the same site

64
: restricted to the same region

128
: restricted to the same continent

255
: unrestricted

上一篇 shell 脚本监控程序是否正在执行， 如果没有执行， 则自动启动该进程
下一篇 Mysql 下载安装及链接错误处理

查看评论

6楼 haicai1989 2013-12-24 09:58发表

在同一个ip上运行ser和cli是可以的，不同的ip，cli就收不到组播数据了

5楼 gqqnb 2013-10-23 03:46发表

谢谢作者

4楼 csgo 2013-08-29 09:20发表

这个头像上的小孩长得很像李二白！

3楼 WUDAIJUN 2013-07-15 23:37发表

花了一天的时间阅读，结合《计算机网络(第五版)》，重新复习了很多知识，是不可多得的汇总。

2楼 huatian008 2013-02-22 00:37发表

介绍的很清楚，学习了

1楼 lmh320 2012-04-17 20:48发表

很好 谢谢

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack FTC
coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo
Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
Angular Cloud Foundry Redis Scala Django Bootstrap