分布式多媒体应用级QoS协商算法*

张 怡, 陈福接

(国防科学技术大学 计算机学院 并行与分布处理重点实验室,湖南 长沙 410073)

E-mail: yizhang@nudt.edu.cn; zhangyii@china.com

摘要: 资源预约是保证分布式多媒体应用服务质量(QoS)的重要手段.提出一个应用级QoS协商算法,该算法有两个主要优点:一是局部参数全局化,使预约变得灵活,便于直接对用户所关心的QoS参数进行预约;二是全局预约局部化,使预约变得简单,有利于实现最优延时分配.

关键词: 分布式多媒体;QoS;应用级资源预约;最优延时分配

中图法分类号: TP37 **文献标识码**: A

QoS协商和资源预约是保证分布式多媒体应用QoS的重要手段[1~7].人们对资源预约的早期研究主要集中在网络一级,如RSVP(resource reservation protocol)协议[2],通过预约网络资源来保证数据传输的带宽和延时要求.随着ATM以及千兆以太网等高速网络技术的成熟和RSVP协议的使用,网络已不再是决定分布式多媒体应用QoS的唯一重要因素[1,3~8];分布式多媒体应用的QoS管理应该是完全端到端的过程[1,3],用户也希望参与QoS的协商.如何在应用级的层次上对端到端的资源进行协商和预约已成为一个新的研究热点[3~6].

目前,国际上提出了多种端到端的资源预约算法[2,4~8],对这些算法可以从以下几个角度进行分类:

(1) 两趟预约和三趟预约.两趟预约算法主要用于点到点和点到多点的应用[2,4],而三趟预约算法主要用于多点到多点的应用[7].

(2) 集中式预约和分布式预约.集中式预约在预约的过程中有一个总控节点[7],分布式预约则无总控节点[2,4].集中式预约的特点是控制简单,但总控节点使得系统较脆弱;分布式预约控制较复杂,但计算分布在各个节点上,系统稳定性好.

(3) 固定预约和协商预约.在固定预约中,一旦链路上某个节点无法满足QoS的要求,则预约失败[2,4].而当在协商预约中遇到这种情况时,QoS参数自动降级,使预约继续,由用户决定是否接受本次预约[7].

(4) 发送方发起的预约和接收方发起的预约.发送方发起的预约有利于计算资源的最优分配,但不适合多点到多点的应用[7].接收方发起的预约有利于多点到多点的应用,但往往较为复杂[2,8].

上述各种预约算法都只考虑了某一方面的好处,如或者只支持client-server模式[4],或者虽支持分布式结构却使用了总控节点[7]等.

基于以上考虑,本文提出了一种新的QoS协商算法.该算法既考虑了分布式多媒体应用拓扑结构的复杂性,又考虑了系统的稳定性,并能自动实现QoS降级.算法中首次提出了递归协商的思想,即协商本身是具有端到端意义的,但对于链路上的各节点来说却是局部的.它认为前趋节点就是应用的源,后继节点就是应用的宿,它所考虑的QoS就是全局的QoS,从而在全局范围内办商用户定义的局部有意义的QoS参数(如视频中的帧频、分辨率等).递归协商算法简化了协商过程,并能用较小的运算量实现最优延时分配.

本文首先为协商算法定义了一个基本应用模型,然后进一步阐述递归协商的思想,给出递归协商算法.然后通过分析指出,对于实现最优延时分配,递归协商算法优于非递归算法.最后,给出实验结果并总结全文.

* 收稿日期: 1999-03-16; 修改日期: 1999-09-13

基金项目: 国家杰出青年基金资助项目

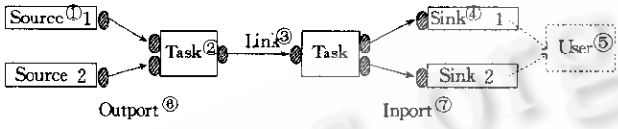
作者简介: 张怡(1973-),女,山西五台人,博士生,主要研究领域为分布式多媒体,实时技术;陈福接(1935-),男,福建安溪人,教授,博士生导师,主要研究领域为多媒体技术.

1 基本应用模型

定义 1. 分布式多媒体应用模型可用三元组(V,E,R)表示:V 为节点集,节点 $v_i \in V$ 称为任务(task)$(i=1,2,\ldots,n)$;E 为有向弧集,弧$(v_i,v_j) \in E(i \neq j)$称为链路(link);$R$ 为节点的属性集,定义 V 中任务的属性.

定义 2. 任务 v_i 表示完成部分应用功能的软件和/或硬件的集合.只有扇出弧的任务是源任务,只有扇入弧的任务是宿任务.源任务产生数据,宿任务消耗数据.既有扇出弧又有扇入弧的中间任务则对数据进行处理和转换.弧的起点定义为任务的输出端口,终点为输入端口,输入、输出端口表示数据的流入、流出点.

图 1 给出了一个典型的分布式多媒体应用模型.



①源任务,②任务,③链路,④宿任务,⑤用户,⑥输出端口,⑦输入端口.

Fig. 1 Distributed multimedia application model

图1 分布式多媒体应用模型

定义 3. 链路(v_i,v_j)是对任务 v_i 到 v_j 的单向数据连接的抽象,其方向表示数据流动方向.

定义 4. 任务 v_i 的属性 $r_i \in R$ 定义了任务与资源预约相关的属性参数及方法:

• Inport[INPORTNUM]:输入端口属性数组,INPORTNUM 为输入端口的个数,每个输入端口包括下列参数:

QoSConstraint:任务支持的此端口输入数据的 QoS 范围,其表示形式为序偶式二元组(QoS 参数,允许范围);

$t\vec{q}_{in}$:当前任务支持的此端口的最大输入 QoS,为 m 维向量,m 是 QoS 参数的个数;

• InportRelations:各个端口输入 QoS 之间的约束关系.$\vec{q}_{in}=\{\vec{q}_{in}^{1},\ldots,\vec{q}_{in}^{INPORTNUM}\}$,$\vec{q}_{in}^{i}$表示第 i 个输入端口的输入 QoS 要求,它们应满足 InportRelations.如一个将两条视频流混合成画中画视频流的任务可能要求小画面的帧频只需是主画面帧频的一半.

• Outport[OUTPORTNUM]:输出端口属性数组,OUTPORTNUM 为输出端口的个数,每个输出端口包括下列参数:

QoSConstraint:任务支持的此端口输出数据的 QoS 范围;

$t\vec{q}_{out}$:当前任务支持的此端口的最大输出 QoS,为 k 维向量,k 是 QoS 参数的个数;

• $R(\vec{q}_{in},\vec{q}_{out},\vec{r})$:表示输入 QoS 参数$\vec{q}_{in}$、输出 QoS 参数$\vec{q}_{out}$和资源量$\vec{r}$三者之间的关系.其中$\vec{r}=\{r_1,\ldots,r_t\}$,表示该任务使用的 t 种资源.$R(\vec{q}_{in},\vec{q}_{out},\vec{r})$可由应用开发者根据经验得出、统计得出或由函数式确定,模型本身对 R 的表示方法不作规定.文献[7]给出了一种简单而有效的分级映射表方法.

• TaskReserve(ReserveMsg[OUTPORTNUM]):任务处理预约请求消息(ReserveMsg)的方法;

• TaskRelax(RelaxMsg[INPORTNUM]):任务处理资源调整消息(RelaxMsg)的方法;

• TaskRelaxBack(CommitMsg):任务处理资源调整回溯消息(CommitMsg)的方法;

• TaskRefuse(RefuseMsg):任务处理协商失败消息(RefuseMsg)的方法.

2 递归协商算法

递归协商算法基于上述应用模型和递归的思想,就用户的 QoS 要求与可用资源量进行协商,并预约所需资源,从而在应用过程中为用户提供质量稳定的服务.

2.1 递归协商的基本过程

递归协商过程中使用了 4 种消息(见表 1).

递归协商算法的执行分为 3 个阶段(设 $v_i,v,v_j \in V$,$(v_i,v) \in E$,$(v,v_j) \in E$,$i=1,\ldots,q$,$j=1,\ldots,n$).

Table 1 Messages used in negotiation

表 1 协商过程中使用的消息

Message name①	Content②	Meanings③
ReserveMsg	$[\vec{q}_{min},\vec{q}_{max}]$	The QoS range that tasks receiving messages should guarantee when they output data④
RefuseMsg	Nothing⑤	Reservation failed, requiring tasks release reserved resources⑥
RelaxMsg	$\vec{q}_{max}$	QoS that could be guaranteed by sink sending this message⑦
CommitMsg	$\vec{q}$	Finally committed QoS⑧

①消息名称,②主要内容,③含义,④接收消息的任务在输出数据时应该保证的QoS范围,⑤无,⑥预约失败,要求任务释放已预约的资源,⑦发送此消息的宿能保证的QoS,⑧最终提交给用户的QoS.

预约阶段. 此阶段是由宿向源反向搜索(V,E,R)的过程. 宿发出ReserveMsg消息,$\vec{q}_{max}$为用户期望的QoS,$\vec{q}_{min}$为用户可容忍的最坏QoS. v收到v_j的ReserveMsg后,在本地预约资源,并创建新的ReserveMsg消息发往v_i. 若v预约资源失败,则向宿方向发失败消息RefuseMsg,通知已预约的任务释放预约的资源,协商过程结束;否则,当所有源预约成功后预约阶段结束,进入下面的调整释放阶段.

释放阶段. 源向宿方向发调整释放消息RelaxMsg,v收到来自v_i的RelaxMsg后,根据$\vec{q}_{max}$尽可能释放过多预约的资源,创建新的RelaxMsg发往v_j. 当每个宿都收到RelaxMsg后,释放阶段结束,进入下面的提交阶段.

提交阶段. 宿向源方向发提交消息CommitMsg,其中$\vec{q}=\min\{\vec{q}_{max,i},i=1,2,\ldots,n\}$即为所有$v_j$的RelaxMsg中最小的$\vec{q}_{max}$,$v$根据$\vec{q}_{max}$进一步释放可能过多预约的资源,创建新的CommitMsg发往v_i. 当每个源都收到CommitMsg后,整个协商过程结束.

2.2 递归协商的特点

算法的特点是局部QoS参数全局化及全局预约局部化.

局部QoS参数全局化. 局部QoS参数全局化就是将只在局部有意义的QoS参数放到端到端的协商过程中. 这样,不仅可以协商全局有意义的参数(如延时),用户所关心的只在局部有意义的QoS参数(如帧频和分辨率)也可以在端到端的意义上进行协商,这就真正符合了完全端到端的意义[3].

局部QoS参数可以全局化是因为在递归协商算法中,每个任务都把其前趋任务作为源. 例如,用户认为本地的视频播放器就是向他提供画面的源,他可以提出帧频和分辨率的要求,因为这时帧频和分辨率是具有全局意义的. 至于服务器如何向视频播放器提供播放内容,以及网络如何向服务器提供足够的带宽则不是用户所关心的问题.

为了使局部参数全局化,每个任务上的关系$R(\vec{q}_{in},\vec{q}_{out},\vec{r})$提供输入、输出QoS之间的转换. 因此,只要在相邻两个任务间有意义的QoS参数都可以在资源预约时使用.

全局协商局部化. 协商必须是端到端的,即具有全局意义. 协商局部化并不是否定其全局意义,而是通过递归的方法将协商过程分解,使得任务行为只受其相邻任务的影响,从而简化了预约过程.

为了屏蔽非相邻任务的影响,在采用递归策略时可能还需要一系列的合并过程,详细分析见第3节.

2.3 递归协商算法

下面给出任务在预约和释放阶段的算法,由于任务对RelaxMsg和CommitMsg消息的处理大致相同(仅消息传递的方向相反),对RefuseMsg消息的处理也较简单,因此省略了处理CommitMsg和RefuseMsg的算法.

算法1. 预约阶段处理算法——TaskReserve(ReserveMsg[OUTPORTNUM]).

· 将每个输出端口i收到的ReserveMsg[i]的$[\vec{q}_{min},\vec{q}_{max}]$与此端口的QoSConstraint取交集,若交集为空,则向每个输出端口发RefuseMsg后终止算法;

· 合并所有交集得到对此任务的输出QoS要求$[\vec{q}_{out,min},\vec{q}_{out,max}]$;

· 由各个输入端口的QoSConstraint及InportRelations确定$[\vec{q}_{in,min},\vec{q}_{in,max}]$;

· 根据当前可用资源$\vec{r}_{now}$和$R(\vec{q}_{in},\vec{q}_{out},\vec{r})$确定集合$S$:

$$S=\{(\vec{q}_{in},\vec{q}_{out},\vec{r})\mid R(\vec{q}_{in},\vec{q}_{out},\vec{r})\&(\vec{q}_{out}\in[\vec{q}_{out,min},\vec{q}_{out,max}])\&(\vec{q}_{in}\in[\vec{q}_{in,min},\vec{q}_{in,max}])\&(\vec{r}\leqslant\vec{r}_{now})\};$$

· 若$S=\varnothing$,则向每个输出端口发RefuseMsg后终止算法;

· 令 $(\vec{q}_{in,min},\vec{q}_{out,min},\vec{r}_{min})=\min\{(\vec{q}_{in},\vec{q}_{out},\vec{r})|(\vec{q}_{in},\vec{q}_{out},\vec{r})\in S\}$;

$(\vec{q}_{in,max},\vec{q}_{out,max},\vec{r}_{max})=\max\{(\vec{q}_{in},\vec{q}_{out},\vec{r})|(\vec{q}_{in},\vec{q}_{out},\vec{r})\in S\}$;

· 对于每个输出端口,令 $t\vec{q}_{out}=\vec{q}^{t}_{out,max}$;

· 对于每个输入端口,令 $t\vec{q}_{in}=\vec{q}^{t}_{in,max}$,并以$[\vec{q}^{t}_{in,min},\vec{q}^{t}_{in,max}]$为参数向端口所连链路发 ReserveMsg;

· 请求系统资源管理部件预约资源$\vec{r}_{mex}$.

算法 2. 释放阶段处理算法——TaskRelax(RelaxMsg[INPORTNUM]).

· 若存在输入端口 $i(i\leqslant INPORTNUM)$满足 RelaxMsg[i].$\vec{q}_{max}<$Inport[i].$t\vec{q}_{in}$,则调整所有输入端口的$t\vec{q}_{in}$,使得它们满足 InportRelations 且 Inport[i].$t\vec{q}_{in}\leqslant$RelaxMsg[i].$\vec{q}_{max}$(对所有 $i\leqslant INPRTNUM$);

· 对于所有的输入端口,若其$t\vec{q}_{in}$被修改,则以此$t\vec{q}_{in}$为参数向该端口发 CommitMsg,并重新计算$t\vec{q}_{out}$;

· 释放多余资源;

· 对于所有输出端口以其$t\vec{q}_{out}$为参数向相连链路发 RelaxMsg.

3 最优延时分配

端到端延时的分配是协商和预约中的一个重要问题,它直接关系到应用的服务质量和资源的优化利用.设 D 表示用户的端到端延时要求,d_i 是在任务 $v_i(v\in V)$上的延时;C_i 是 v_i 上花费与延时的函数,在一定范围内,延时越小,需预约的资源就越多,花费也就越大.C_i 与资源使用量、任务的重要程度及资源的紧俏度有关.最优延时分配问题就是找出一组 $d_0,\ldots,d_n$,使得在约束条件 $\sum_{i=1}^{n}d_i<D$ 的情况下,$\sum_{i=1}^{n}$ 最小[8].下面,重点介绍在我们建立的 VoD(video on demand)原型系统[9,10]中实现最优延时分配的方法.

目前,在应用级对最优延时分配问题研究得较少,且从解决方案上看有两个不足之处:一是将 C_i 限制为分段线性单调递减的凸函数,忽略了 C_i 实际的复杂性;二是算法复杂度随任务数增加而增加.我们提出的实现方案只用很小的开销即可实现类最优的延时分配,其主要优点是算法简单、复杂度与任务数 n 无关,且 C_i 函数的形式不限.

改进方案的基础是递归协商算法.在递归协商中,每个任务将其前趋任务看做源,后继任务看做宿,因此每个任务只看到其前趋的 C_i 函数.为了使其后继看到它的 C_i 函数,在预约阶段本地将接收到的 C_{i-1} 与本地的 C_i 合并,然后将得到的新 C_i 函数(实际是使用了一个函数表示了 $C_1\sim C_i$)传给任务 $i+1$.在释放阶段,每个任务只须比较本地的 C_i 函数和前一个任务的 C_i 函数就可以得到本地最佳释放的延时值.方案的关键是 C_i 函数的合并过程,下面给出合并算法.

定义 5. v_i 的开销函数 C_i 为二元组(d_i,c_i).d_i 为 v_i 能保证的最小延时,c_i 为 $n+1$ 维数组,$n=\left\lceil\frac{D-d_i}{\Delta t}\right\rceil$,$\Delta t$ 为 v_i 上可预约延时的最小单位.$c_i[k]=C_i(d_i+k\cdot\Delta t)$,$k=0,1,\ldots,n$.

v_i 收到预约消息后,对消息中携带的 C_0 和本地的 C_i 进行合并,产生新的 $C_0^{new}=(d_0^{new},v_0^{new}[m])$及数组 $x[m]$,其中 $m=\left\lceil\frac{D-d_0-d_i}{\Delta t}\right\rceil$.$C_0^{new}$ 随预约消息传送到后继任务,数组 $x[m]$保留在本地,记录本地释放延时的开销,供释放过程使用.

算法 3. 合并算法.

(1) $m=\left\lceil\frac{D-d_0-d_i}{\Delta t}\right\rceil$,$d_0^{new}=d_0+d_i$.若 $d_0^{new}>D$,则预约失败,算法结束;

(2) 循环 1:p 从 0 到 m do $c_0^{new}[p]=c_0[0]+c_i[0]$,$x[p]=0$;

循环 2:q 从 0 到 p do

若$(c_0[q]+c_i[p-q]<c_0^{new}[p])$,则 $c_0^{new}[p]=c_0[q]+c_i[p-q]$,$x[p]=p-q$,

循环 2 结束;

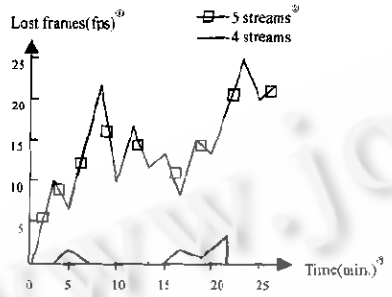
循环 1 结束.

4 实验结果

我们在自己研制的 VoD 原型系统[9,10]中进行了实验. 系统中视频服务器采用 PII333M 个人机,点播端均为 Pentium166M 个人机,网络为共享式 10M 以太网. 分别在两种情况下进行了测试.

4.1 无竞争情况

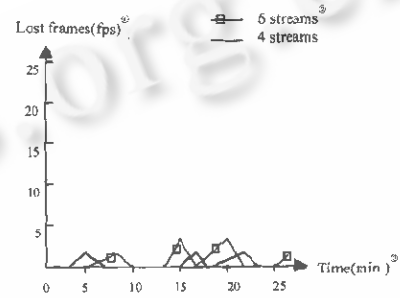
整个系统只传输播放 MPEG-1 流,实验结果如图 2 所示. 由图 2(a)可以看出,在不采用协商的情况下,系统最多只能支持 4 条流,用户无法提出自己的 QoS 要求;而在采用递归协商算法时(如图 2(b)所示),通过 QoS 自适应(假设用户最低要求为 15 帧/s),最多能支持 6 条流,且质量相当稳定.



①丢帧率(帧/秒),②5 条流,4 条流,③时间(分).

(a) No negotiation case

(a) 无协商情况



①丢帧率(帧/秒),②5 条流,4 条流,③时间(分).

(b) Negotiation case

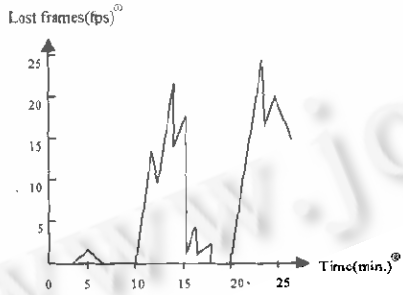
(b) 有协商情况

Fig. 2

图 2

4.2 有竞争的情况

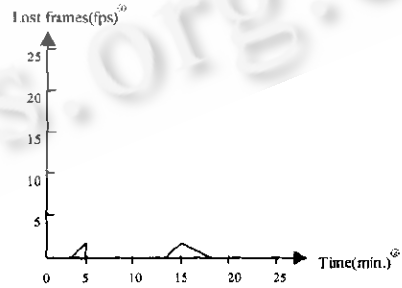
系统同时播放 3 条 MPEG-1 流,每隔 10 分钟从服务器向某用户发送一个平均速率为 2M/s 的文件,结果如图 3 所示. 从图中可以看出,未采用协商协议时,文件传输的介入极大地影响了流的播放质量,几乎无法观看;而在采用协商协议时,文件传输对流质量基本无影响.



①丢帧率(帧/秒),②时间(分).

(a) No negotiation case

(a) 无协商情况



①丢帧率(帧/秒),②时间(分).

(b) Negotiation case

(b) 有协商情况

Fig. 3

图 3

递归协商算法的复杂性主要由两个因素决定:应用中节点的个数和每个节点进行本地资源协商的复杂性. 对于后者,我们研究了相应的解决方法[9];而对于前者,则需要应用设计者在应用的模块化程度和协商复杂性之间进行权衡:划分的模块越多,每块的功能就越单纯、清晰,但协商的时间开销就越大;反之,模块越少,时间开销就越小,但模块功能就越复杂,不利于应用的维护.

与目前功能较为完善的 XNRP(extended negotiation and reservation protocol)[7]协议相比较,二者均能支持结构复杂的分布式多媒体应用和 QoS 降级.递归协商的优越性在于取消了总控节点,增强了系统的强壮性,节省了每个节点与总控节点通信的开销,且接收方发起的协议能更灵活地支持多目通信[2].

只有一个好的协商算法是不够的,它还需要本地资源对预约的更好支持.在递归协商算法中,如何更简单、更灵活地定义关系 $R(\vec{q}_{in},\vec{q}_{out},\vec{r})$,在延时最优分配中,本地如何提供开销函数 C 都是我们要继续研究的问题.

References:

[1] Campbell, A., Coulson, G., Hutchison, D. A quality of service architecture. ACM Computer Communication Review, 1994,24(2):6~27.

[2] Zhang, L. X., Deering, S., Estrin, D., *et al*. RSVP: a new resource reservation protocol. IEEE Network, 1993,7(5):8~18.

[3] Wang, Xing-wei, Zhang, Ying-hui, Liu, Ji-ren, *et al*. Research of QoS management mechanisms in distributed multimedia systems. Journal of Software, 1998,9(2):86~90 (in Chinese).

[4] Nahrstedt, K., Smith, J. The QoS broker. IEEE Multimedia, 1995,1(1):34~45.

[5] Hafid, A., Abeni, L., Anna, S. Models for QoS negotiation in distributed multimedia applications. In: Dssoulf, R ed. Proceedings of the 2nd International Workshop on Protocols for Multimedia Systems. Austria: Salzbulg Press, 1995. 60~78.

[6] Dermler, G., Fiederer, W., Barthi, R. A negotiation and resource reservation protocol for distributed multimedia applications. In: Kim, M ed. Proceedings of the IEEE International Conference on Multimedia Computing and Systems. Tokyo: Osaka Press, 1995. 45~57.

[7] Kurt, R., Dermler, G. QoS negotiation and resource reservation for distributed multimedia applications [Ph. D. Theisis]. University of Stuttgart, 1996.

[8] Anderson, D. P. Metascheduling for continuous media. ACM Transactions on Computer Systems, 1993,11(3),226~252.

[9] Wu, Fei, Chen, Fu-jie, Ren, Hong. A multimedia service system supporting parallel streams. Journal of Software, 1997,8(5):327~333 (in Chinese).

[10] Liu, Heng-zhu, Chen, Fu-jie, Chen, Xu-can. Study of algorithms about storage allocation of video server. Chinese Journal of Computers, 1998,21(4):289~295 (in Chinese).

附中文参考文献:

[3] 王兴伟,张应辉,刘积仁,等.分布式多媒体系统服务质量管理机制的研究.软件学报,1998,9(2):86~90.

[9] 吴飞,陈福接,任鸿.一个支持并行流的多媒体服务器系统.软件学报,1997,8(5):327~333.

[10] 刘衡竹,陈福接,陈旭灿.视频服务器中视频流的存储分配算法的研究.计算机学报,1993,21(4):289~295.

An Application Level QoS Negotiation Protocol in Distributed Multimedia Applications

ZHANG Yi, CHEN Fu-jie

(*Key Laboratory of Parallel and Distributed Processing, School of Computer, National University of Defence Technology, Changsha* 410073, *China*)

E-mail: yizhang@nudt.edu.cn; zhangyii@china.com

Received March 16, 1999; accepted September 13, 1999

Abstract: Resource reservation is an important means to guarantee the QoS of distributed multimedia applications. In this paper, an application level QoS negotiation protocol is proposed. It has two merits. One is globalizing the local QoS parameters so that the reservation becomes more flexible and users can reserve resources using the parameters they concern. The other is localizing the global reservation to make reservation simpler and to facilitate the implementation of optimal delay allocation.

Key words: distributed multimedia; QoS; application level resource reservation; optimal delay allocation