システムソフトウエアと
オペレーティング・システム　60－12
マルチメディア通信と分散処理　61－12
（１９９３．　７．　８）

分散マルチメディア環境を実現するAVサーバ 歌舞伎

永松 竜夫, 河上 達, 田辺 充, 田中 浩一

ソニー株式会社

本稿では、汎用ＵＮＩＸワークステーション上でマルチメディアデータを統合し、効率良く統一的に取り扱う環境をアプリケーションソフトウェアに提供するＡＶサーバ「歌舞伎」について述べる。ＡＶサーバ「歌舞伎」は、マルチメディアデータの同期、通信、圧縮伸張、リソース管理などを行なう。システムに対する要求と、それを実現するためのモデルと実装を提示し、ネットワーク環境での性能の測定結果と評価について述べる。

AVserver "KABUKI": Distributed Multimedia Environment

Tatsuo Nagamatsu, Itaru Kawakami, Mitsuru Tanabe, Koichi Tanaka
Sony Corporation

This paper presents the AVserver "KABUKI." It is developed to provide clients with an integrated environment to handle multimedia data on Unix workstation. It offers the functions of inter-media synchronization, data transfer, data compression/decompression and resource management. Requirement, architecture model and implementation of the system are described. Also, the results of performance measurements in a network environment and the evaluation are discussed.

1. はじめに

近年、パーソナルコンピュータやワークステーションなどの性能が格段に向上し、身近に普及しつつある。それらをテレビや電話と同じように日常的なコミュニケーションに利用したい、さらには、情報処理システムをより広範囲の用途に活用したいという要求が出てきている。これらの要求を実現するために、音声や動画などのマルチメディアデータを分散環境上でインタラクティブに扱えるようにする事が必要となっている。

しかしながら、動画や音声などのマルチメディアデータは、コンピュータがこれまで処理してきた数値データやテキストデータとは本質的に異なる性質を持っている。第一に、マルチメディアデータは単なるバイト列ではなく、明示的あるいは暗示的に時間の属性を持っている。第二に、本質的にデータ量が莫大かつ冗長で、ハードウェアの処理能力が向上したとしても、効率良く扱うためにはデータ圧縮をする必要がある。第三に、インタラクティブな処理を必要とされ、スループットだけでなくレスポンスや遅れなどの性能が重視される。

このため、これらを分散環境で扱うためには、メディア間同期、通信、処理、リーソス管理に新たな手法を導入する必要がある。AVサーバ「歌舞伎」は、これを実現するために開発した。

本稿では、2章においてAVサーバに要求される機能を挙げ、3章でAVサーバのモデル化とAPIを紹介し、4章で実装について述べる。5章でAVサーバの評価を行ない、6章で今後の課題、7章でまとめを述べる。

2. AVサーバの機能

今回の実装では、以下のような目標を考えた。

(1) いろいろな制約はあるが、標準のNEWS-OS (Unix) 上に構築し、既存のシステムとの親和性を保つこと。
(2) 音声や動画の通信や特有の処理は、サーバにまかせ、クライアントプログラムはそのコントロールだけを行うこと。
(3) メディアデータの属性と、クライアントプログラムの目的に応じて柔軟に対処できること。
(4) プロセッサやネットワークなどのリソースの負荷の変動に対して、対応できること。
(5) モデルが単純なこと。

これらの要求から、サーバは以下に述べるような機能を持つことが必要となる。

2.1 メディアタイプ

サーバで扱うメディアデータとして、基本的には音声と動画がある。これらには、単位時間あたりのデータ量や生成時刻などの時間的な属性をつけ、これをサーバでの処理に利用する。たとえば、データ幅8[bit]、サンプリング周波数8[KHz]の音声データの場合には、単位時間は1/8000[sec]、単位時間あたりのデータ量は、1[byte]という属性がつけられる。

2.2 サーバとメディアデバイス

サーバは、複数の入力デバイスからのデータを、時間的な同期を取りながら、出力デバイスに出力する。

入力デバイスとしては、
(1) オーディオインタフェースやビデオ入力インタフェースなどのハードウェアデバイス、
(2) サウンドファイルや動画ファイルやムービーファイルのようなマルチメディアデータファイル、
(3) マルチキャストアドレス、
(4) クライアントプロセス
などをサポートする。

出力デバイスとしては、
(1) オーディオインタフェースやウィンドウなどのデバイス、
(2) マルチメディアファイル、
(3) マルチキャストアドレス、
(4) クライアントプロセス、
などをサポートする。

あるホストにはサーバはただ一つだけ存在し、それが直接に取り扱うデバイスは、そのホスト上にあることが必要である。入力デバイスと出力デバイスが別々のホストにあることを必要とする場合には、クライアントがそれぞれのホスト上のそれぞれのサーバにアクセスしてサーバ同志を接続させる。

2.3 メディアデータの転送と同期

マルチメディアシステムとしてユーザに提供する物のQOS (Quality-of-Service) の評価規準として以下

のものが考えられる。

(1) 転送の遅れの許容限度、
(2) メディア間同期の許容限度、
(3) スループット、
(4) データの欠損が許される場合と許されない場合

与えられた転送路で満足できる QOS を得るために、データ量、圧縮方式、プロトコルやパケットサイズなどの転送の際の種々のパラメタをコントロールする。サーバは QOS に従って、メディア間同期を行う。

2.4 ネットワークプロトコール

サーバが利用するマルチメディアデータのネットワークプロトコールは、 QOS を考慮しながらフローなどを動的にコントロールできる必要がある。現状のネットワーク環境の Ethernet や FDDI などネットワークデバイスは、分散環境上で個々に資源を取り合って共有しているため、あらかじめネットワーク資源を確保するようなサービスが難しい。しかし、現状のネットワーク環境との親和性を考えると IP を利用する必要があり、今回はネットワークプロトコールとして TCP と UDP を利用した。

2.5 データの圧縮伸張

サーバは、ソフトウェアまたはハードウェアによる、音声データや動画データの圧縮伸張機能を持つ。音声の圧縮方式としては G.711, G.721, G.722, G.728 などをサポートする。動画の圧縮方式としては JPEG、H.261、 MPEG などがある。圧縮方式ごとに特徴があるので、用途によって使い分ける必要がある。

2.6 リソースのコントロール

サーバは、複数のクライアントプログラムからの入出力要求を処理する機能をもつ。たとえば、音声入力デバイスは一つしかないのに複数のクライアントから音声入力要求があった場合には、以下のような処理方法が考えられる。 (1) すべての要求元にコピーして配る、 (2) 先着の要求を優先し、後着の要求を拒絶する、 (3) クライアントを順次切り替える、この機能は、ウィンドウシステムでのウィンドウマネージャに相当するマルチメディアマネージャなどのプログラムが利用する。

2.7 物理デバイスのコントロール

音声や動画などのマルチメディアデータを扱う場合には、ビデオカメラやビデオデッキのような AV 機器を接続することが必要になる。サーバは、これらのコントロールのために、 AV 機器制御用の VISCA プロトコールをサポートする。

3. AV サーバのモデルと API

3.1 AV サーバのモデル

AV サーバはクライアントそれぞれに対し 1 つの実行制御単位 (AVobj) を生成する。クライアントがマルチメディアデータの入出力を行いたい場合には、次のような手順で AV サーバに要求をだす。まず、 AVobj において仮想的なメディアデバイス (AVdev) をオープンする。 AVdev は物理的なデバイスではないため、排他制御や複数からのオープンなどが実現出来る。

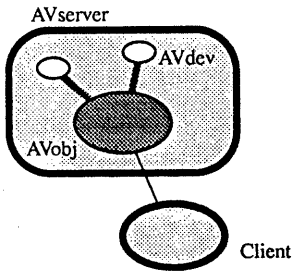

Fig 2. AVserver model

入力用に AVdev をオープンした AVobj と出力用に AVdev をオープンした AVobj とを接続することによりマルチメディアデータの転送路が確保される。同一の AVobj でオープンされているデバイス間のメディアの同期は保証される。

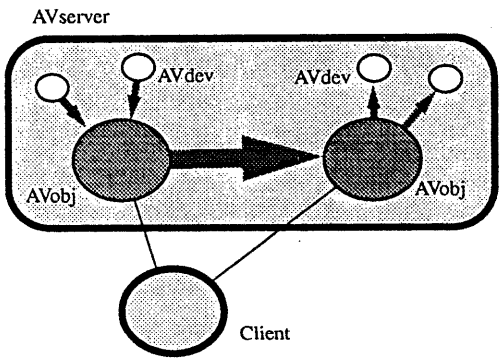

Fig 3. Media Transfer in a Local Host

また、図 4 のようにネットワークでつながれた異なるホスト上の AV サーバにおいて AVobj を生成し、接続することにより分散環境上のワークステーション

においてマルチメディアデータの転送が行われる。

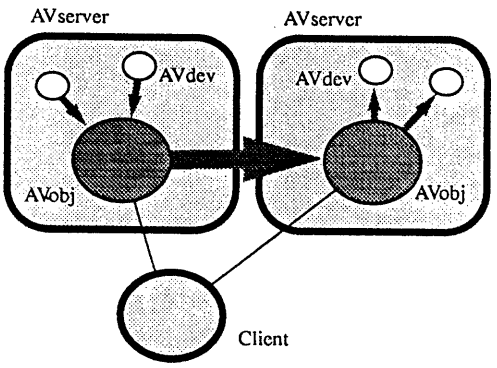

Fig 4. Media Transfer between Remote Host

送信、受信側の AVobj がオープンするデバイス AVdev としてサウンドデバイスを用いると電話が実現できる。さらにビデオデバイスをオープンするとテレビ電話が実現できる。また、入力デバイスとして Movie ファイルを指定し、出力デバイスにサウンドデバイスとビデオデバイスをオープンすると Movie プレーヤーとなる。このように入出力のデバイスを組み替えることにより各種マルチメディアアプリケーションを容易に作成することが可能となる。

3.2 AV サーバの API

AV Server のライブラリには次のものが用意されている。

- int avs_new(char *hostname);
  ホスト名 hostname 上で起動されている AV Server において AVobj を生成する。エラーが発生した場合には NULL が返される。正常終了した場合には AVobj の ID が返される。 AVobj に対する命令はすべてこの AVobj の ID を用いて行なわれる。

- int avs_open(int net, char *devname, int mode);
  デバイス AVdev をオープンする。引数は AVobj の ID、デバイス名、モードである。

- int avs_connect(int net1, int net2);
  2つの AVobj を point-to-point 接続する。これによって接続した AVobj の一方が以下に述べる avs_transfer によって転送状態になると、もう一方の AVobj がそのデータを受けとれるようになり自動的に受けとったデータを処理する。ひとつの送信 AVobj に対して複数の受信 AVobj を接続することが可能なため、1対多のデータ転送を。

- int avs_transfer(int net, int dev, int length);
  AVobj の送信を制御する。デバイス ID として 0 を指定すると、 AVobj がオープンしたすべてのデバイスに対して有効となる。 length に正の数を指定するとその長さだけデータ転送が行なわれる。（単位は msec）ゼロを指定すると、次の av_transfer が与えられるまで転送します。負の数を指定すると即座に停止する。

- int avs_destroy(int net);
  av_new によって生成した AVobj を解放する。

- int avs_interval(int net, int dev, int interval);
  転送インターバルの設定を行なう。単位は msec。デバイス ID としてゼロを設定すると、その AVobj でオープンされたすべてのデバイスに対して適応される。

- int avs_resize(int net, int dev, int width, int height);
  ビデオデバイスに対してサイズ変更を要求する。サイズの単位は pixel。デバイス ID としてゼロを設定すると、その AVobj でオープンされたすべてのビデオデバイスに対して適応される。

- int avs_nettype(int net, char *type);
  AVobj 同志を接続するネットワークのタイプを設定する。現在のところ "TCP"と "UDP"がサポートされている。これは avs_connect を実行する以前に行なわなければならない。接続を確立した後のタイプ変更は現在サポートされていない。

- int avs_fd(int net);
  クライアントと AVobj とのコントロール接続コネクションのファイル記述子を返す。

- int avs_codec(int net, char *type, int quality);
  メディアデータの圧縮方式を指定する。現在のところ、画像データに対して JPEG の圧縮伸張のみがサポートされている。

あと、デバイス AVdev に対し直接メディアデータにアクセスするためのライブラリとして次のものがある。

- int avs_read(int net, int dev, int shmid, int size);
- int avs_write(int net, int dev, int shmid, int size);
- int avs_ioctl(int net, int dev, int request, int shmi);

4．AV サーバの実装

システム全体における AV サーバの位置付けは次の図のようになる。

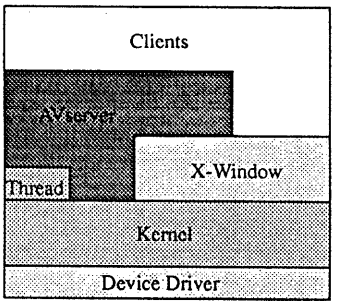

Fig 5. System Hierarchy

4.1 Thread の利用

thread を用いると、複数プロセスを用いて実現するよりもコンテキスト・スイッチの時間が短く、各 thread 間でメモリなどの環境を共有でき、プログラミングが容易になる。 AV サーバを実現するにあたって thread のモデルへの割り当て方法として次の 2 通りが考えられる。

・機能毎に thread にわりあてる
・データストリーム毎に thread を割り当てる

前者では画面入出力・音声入出力・ネットワークなどの機能毎に thread を割り当て、パイプラインを形成する方法であり、後者はメディアデータ毎の入力から出力までを行う処理に thread に割り当てる方法である。

機能毎にパイプラインを形成してもマルチプロセッサ環境下でないと利点はなく、複数のストリームにおいてストリーム単位のスケジュール・プライオリティ制御を行うには、後者が適しているため、今回の実装ではデータストリーム毎に thread を割り当てる方法も用いた。

4.2 TCP/IP の送信受信バッファと遅延

TCP/IP では高信頼性を実現するため、パケットの順序付け、チェックサム、タイムアウトそして再転送を行い、オーバヘッドが大きくデータの転送遅延が問題となる。

送信および受信のためにバッファが用意されている。 NEWS-OS 4.2 ではデフォルトで 8K となっている。このバッファにデータがたまることによって遅延が生じる。例えば、解像度が 160x120 深さ 16bit の画面ならば、一画面でおよそ 38K バイトとなり、送信側と受信側両方のバッファを合わせても 1 フレームもバッファリングされない。この場合にはバッファサイズを大きくすると転送効率は向上する。ところが、画像圧縮をかけて 1/10 程の 4K バイト程度の大きさにすると両方のバッファ合わせておよそ 4 フレームが溜まることとなる。1 秒間に 5 フレームのスピードで転送を行うならばこれだけで約 1 秒間の遅延となる。

しかし逆にあまりバッファを小さくすると、転送効率が悪くなるため、ここにトレードオフがある。この送信、受信バッファのサイズは setsockopt という関数で変更可能である。

5．AV サーバの評価

AV サーバを NEWS-OS 4.2.上に実装した。言語として AT&T の C++3.0.1 を用い、 DCE の pthread ライブラリで thread の機能を実現させた。ソースは注釈文を省くと C++ でおよそ 12200 行となった。

5.1 送受信バッファと転送遅延、効率

実際に送受信バッファのサイズと転送遅延、効率の測定をしてみた。同一ネットワークに接続された 2 台のワークステーションの間で動画像データの転送を行なった。ワークステーションとしてソニーの NEWS シリーズ（CPU R3000 20MHz)をイーサネットで接続し、解像度は 160x120、 16bit の深さの無圧縮と JPEG 圧縮画像で転送し測定した。 1 フレームのサイズは無圧縮でおよそ 38K バイト。JPEG 圧縮でおよそ 5K バイトである。

転送遅延は timed(8) を用いて時計を合わせ、画像取り込みから転送を行いリモート側で表示を行うまでの時間を測定した。また、転送効率としては最大転送フレームレートを測定した。図 5 に無圧縮、図 6 に JPEG 圧縮の画像の送受信バッファサイズと転送遅延、効率の関係を示す。

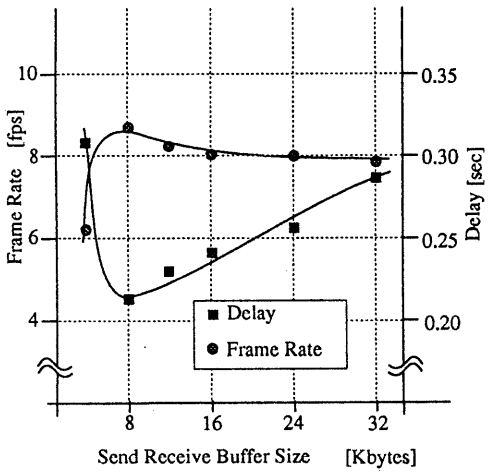

Fig 6. RAW data Performance and Buffer Size

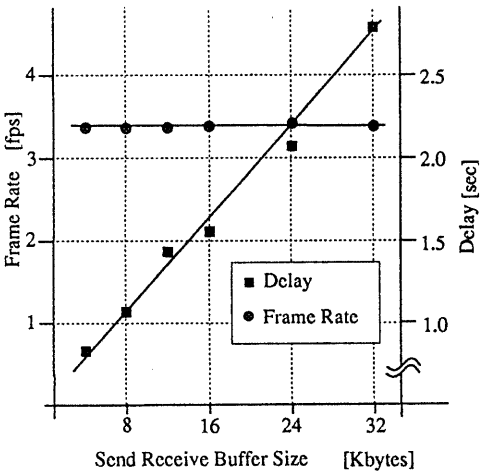

Fig 7. JPEG Performance and Buffer Size

無圧縮で１フレームの大きさが大きい場合には、送受信バッファに１フレームが入り切らないためバッファのサイズに対する転送遅延の差はほとんど変わらない。バッファサイズが 8K バイトの時に転送遅延、効率ともに良い結果が現れる。デフォルトの送受信バッファサイズが 8Kバイトということからも TCP/IPでの転送がその場合に一番効率が良くなるようになっている。バッファサイズが小さくなると極端に悪くなる。

それに比べ、 JPEG 圧縮を行なった画像を転送する場合には、ソフトウェアで JPEG の圧縮伸張を行うため最大フレームレートは少ない。また、送受信バッファに数フレームが溜まってしまうため、バッファのサイズが大きいほど遅延が増大する。そこで JPEG などの圧縮を行い画像データがネットワークのバンド幅より充分小さい場合には、送受信のバッファサイズを転送効率が下がらない程度に小さくすることにより、メディアデータの転送遅延を短縮できる。

5.2 画面転送スピード

同様の環境で、画面サイズとフレームレートおよび転送遅延を測定してみた。

まず、ネットワークとして Ethernet を用いた場合は、次の表の結果が得られた。表１に 16bit の深さの無圧縮画像、表２には JPEG 圧縮した画像の転送最大フレームレートと遅延の性能を示す。

FrameSize	Rate [fps]	Delay [msec]
80x60	18.8	90
160x120	8.6	210
320x240	2.2	440

Table 1. RAW data Performance over Ethrnet

FrameSize	Rate [fps]	Delay [msec]
80x60	5.8	310
160x120	3.4	850
320x240	1.0	4500

Table 2. JPEG Performance over Ethernet

次にネットワークとして ISDN の 1B(64K) を用いた場合の性能は表３のようになった。 ISDN を用いた場合には、画像データを圧縮しないと最小の画面サイズ (80x60) においてもフレームレートが 0.2[fps]となり、実用的でないため、深さ 16bit の無圧縮画像の転送性能の評価は省いた。

FrameSize	Rate [fps]	Delay [msec]
80x60	2.8	750
160x120	2.0	1050
320x240	0.9	2800

Table 3. JPEG Performance over ISDN

5.3 AV サーバのクライアント

AV サーバのクライアントとして avtalk を作成した。これは従来の talk(1) が文字を用いた対話であるのに対し、動画と音声を用いた対話が出来るものである。 AV サーバを用いることにより、画像と音声の取

り込みや圧縮伸張、またネットワークプログラミングを意識することなく容易にマルチメディアアプリケーションが作成可能となった。この avtalk のサンプル画面を図7に示す。

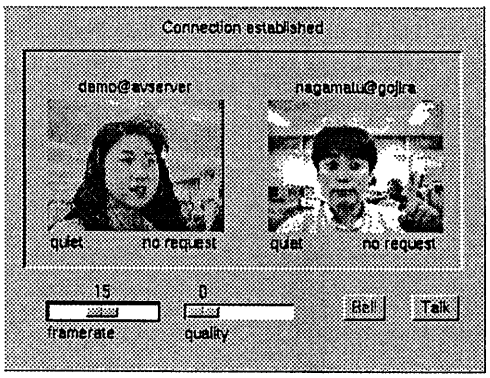

Fig 7. AVtalk screen sample

5.4 メディア間同期の実現

この AV サーバでは同期をとりメディアデータは単一のストリームにインターリーブして転送している。ネットワークプロトコルとして TCP/IP を用いているため、パケットの順序は保証される。複数のメディアデータの取り込みを同時に行い、インターリーブしてデータ送信時に同期を保証できれば、データ受取側でも複数のメディアデータの同期はとれているものと仮定できる。実際この方法で単一ネットワーク上にて転送した音声と動画のメディアデータの同期は満足のいく結果となった。しかし、ネットワークの外乱による Jitter やゲートウェイを通した転送については問題が残されている。

5.5 遅延とゲートウェイの関係

今回の実装ではネットワークプロトコルとして

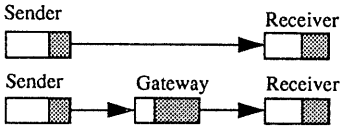

Fig 8. Gateway and Buffering

UNIX で標準的な TCP/IP を用いた。メディアデータの転送にゲートウェイを通さず、直接送ることの出来る環境では送信側と受信側のバッファの制御を AV サーバが行うことができ、細かな流量制御が行えた。ところが、メディアデータ転送経路の途中にゲートウェイが存在すると経路全体の送信、受信バッファのサイズは増大し遅延時間は増える。さらに AV サーバが乗っているホストマシンの送信、受信バッファは転送するメディアデータの転送量に合わせてそのサイズを変更できるが、ゲートウェイの送信、受信バッファのサイズまでは変更できない。

6. 今後の課題

6.1 リアルタイムプロトコル

今回の実装ではネットワークプロトコルとして UNIX で標準的な TCP/IP を用いた。マルチメディアデータの転送にゲートウェイを通さず、直接送ることの出来る環境では送信側と受信側のバッファの制御を AV サーバが行うことが可能であり、細かな遅延制御が行えるが、ゲートウェイを通すと、そこで生じる遅延を AV サーバは制御することが出来ない。そのため、このような環境下ではメディアデータ専用のルータもしくはリアルタイムプロトコルが必要とされる。

6.2 モジュール化による拡張性

Apple の QuickTime では、メディアデータ取り込みのためのルーチン、圧縮伸張 CODEC のルーチンなど、すべてがコンポーネントとして表現され、それらは自由に切り替えることが出来る。今回の実装ではデバイスクラスを追加することにより拡張可能になっているが、動的に追加、変更する機能はサポートしていない。今後、JPEG 以外の CODEC などを組み込む場合には、このように動的に拡張できる機能が必要であろう。

6.3 QOS コントロール

AV サーバのライブラリ avs_interval および avs_resize, avs_codec を用いることによってサービスの質 (QOS) をクライアントから制御することは可能である。ところが、このような QOS の制御を行うにはクライアント側からすると情報不足であり、煩わしい作業であるため、クライアントの起動時に QOS のポリシーを設定するだけであとはすべて AV サーバ側が行うことが望ましい。

現在の実装方法では計算機・ネットワーク資源がなくなるとデータの取りこぼしやブロッキングが起こ

り、転送できるフレームレートが制限されるのでに時間的解像度は自動的に制御される。しかし資源を使い切り、他のプロセス、ユーザに迷惑をかける以前に資源の予約を行い、確保した資源の中で実行できる環境が望まれる。

7. まとめ

分散環境においてサウンドやビデオなどのマルチメディアデータを扱う AV サーバを実現した。これは複数メディアの同期・ネットワークによる通信・データの圧縮伸張を行う。

参考文献

1. Apple Computer, Inc., "QuickTime Developer's Guide, " Draft Developer Technical Publications 1991

2. David P. Anderson and George Homsy, "A Continuous Media I/O Server and Its Synchronization Mechanism," COMPUTER, October 1991, pp 51-57.

3. Earl Rennison, Rusti Baker, Doohyun David Kim, Young-Hwan Lim, "MuX: An X Co-Existent, Time--Based Multimedia I/O Server," THE X RESOURCE 1, Winter 1992, pp 213-233

4. Ralf Guido Herrtwich, "Beyond ST-II: Fulfilling the Requirements of Multimedia Communication ," Third International Workshop on Network and Operating System Support for DIgital Audio and Video November1992, pp 23 - 29

5. Stephen T. C. Chou and Hideyuki Tokuda, "System Support for Dynamic QOS Control of Continuous Media Communication," Third International Workshop on Network and Operating Systems Support for Digital Audio and VideoNovenber 1992, pp 322 - 327