帯域予約型ネットワーク通信プロトコルの実装と評価*

木原 誠司
NTT 情報通信研究所
〒238-03 横須賀市武 1-2356
kihara@isl.ntt.co.jp

連続メディアに向いた，帯域予約型のリアルタイム通信プロトコルであるST-IIの実装について述べる．ST-IIの実装は現在までにいくつかされてきているが，それらはネットワークの使用帯域について考慮しているものの，プロトコル処理の実時間処理を考慮していない．そこで，オペレーティングシステムとしてReal-Time Machを使用し，その実時間処理により，プロトコルの実時間処理を実現する．Real-Time Machのリアルタイムスレッドを使用し，リアルタイムスケジューリングによりデータの送受信を行なう．本稿ではこのプロトコル処理系の設計、現在の実装状況について説明し，評価実験の結果を報告する．

1　はじめに

近年マルチメディア処理に向いたコンピュータシステムの研究開発が盛んである．様々なメディアの中で，最も処理が難しいのは，連続メディアと呼ばれる，時間軸方向に連続的な性質を持つストリーム状のメディアであり，デジタル化した音声，動画などに代表される．

連続メディアを扱うためには，システムはアプリケーションが要求するQOS(Quality Of Service)を保証しなければならない．アプリケーションから指定されるQOS，例えば動画の色数，ピクセル数，秒あたりのフレーム数などは，帯域幅・パケット長・遅延時間・ジッター・誤り率などの，通信レベルのQOSに変換される．ネットワークプロトコルにおいては，これらのQOSをエンドツーエンドで保証することが要求される．

加えて，動画のように帯域幅の広いメディアを転送するためにはプロトコル処理が高速である必要がある．音声と動画とでは帯域幅が大きく異なるため，様々な帯域幅に対応できることが必要となる．例えば移動ホストに対する対応などの要求から，異種混在のネットワークへの対応が望まれる．さらに，動画や音声などのアプリケーションには1対1ではなく複数のメンバからなるグループ間の通信を必要とし，動画や音声などの巨大になりやすいデータを転送するためには，マルチキャスト機能が必要である．

本稿では，以上のような性質を満たすリアルタイム通信プロトコルであるST-II[17]の実装について述べる．第2章にて関連研究について議論する．続いて，ST-IIプロトコル処理系の設計，実装および評価実験について述べる．最後に，結論と今後の課題について述べる．

2　関連研究

本説では，リアルタイム通信プロトコルについて，プロトコルの実装について，そして実装方法について，関連する研究を紹介する．

2.1　リアルタイム通信プロトコル

連続メディアをネットワークで扱うためには，ネットワークがその性質を保つため，従来の文字などの単なるデータ通信と異なり，時間制約を守った通信をサポートする必要がある．しかし，例えば現在一般的に使われている通信プロトコルであるTCP/IPには，このような機能がない．また，リアルタイム通信プロトコルには，高速，低遅延を目指したものがあるが，連続メディアをサポートするためには，ネットワークの帯域保証や，帯域制御が必要である．

*An Implementation and Evaluation of a Reservation-based Network Protocol, Seiji Kihara (NTT Information and Communications Systems Laboratories)

このようなプロトコルに，ST-II，CBSRP[16]，RMTP/RTIP[19]，RSVP[20] などがある．

これらのうち，ST-II は帯域の予約，遅延時間の保証以外に，ヘッダが小さい，マルチキャストのサポート，プロトコル仕様が公開されているなどの特徴を持つ．そこで，著者は，まず ST-II プロトコル処理系のプロトタイプを Real-Time Mach の上に実装し，改良していく方針をとっている．

2.2 プロトコルの実装

x-kernel[5, 13] は開発当初はオペレーティングシステムのカーネルとして開発されたが，最近はプロトコル処理系のための環境として SunOS 4.1 のユーザプロセスまたは Mach 3.0 のタスクとして実装されている．オブジェクト指向のマルチスレッドの構造を取っていて，セッションはプロトコルオブジェクトのインスタンスとして生成される．統一的なプロトコルインタフェースを提供し，メッセージ，メモリテーブル，イベントなどの管理を実現している．

NPS[11] は x-kernel 同様，ユーザレベルサーバによりプロトコル処理系を作るためのツールキットであり，サービスインタフェース，ネットワークデバイスインタフェース，プロトコルサーバとアプリケーション間のメッセージ共有機構が実現されている．優先度に応じたワーカースレッドにより，リアルタイム通信プロトコルの実装に対応している．

ST-II の実装はすでにいくつか行なわれている．スウェーデン国立計算機科学研究所は SunOS の上に ST-II を実装し [12]，IBM ヨーロッパネットワークセンタでは OS/2 および AIX で動作する Heiderberg Transport System[3] に ST-II を採用し改良している [4] が，いずれも CPU やメモリといったローカルな資源の予約については議論していない．

2.3 実装の位置

Real-Time Mach 3.0 のようなマイクロカーネルベースのオペレーティングシステムにサービスを実装する場合，[15] に示されているように，以下の 3 つのレベルで行なうことが可能である．

- カーネル内
- ユーザレベルサーバ
- ライブラリ

カーネル内の実装は，アプリケーションの空間でカーネルモードに遷移するだけで済むため，コンテクスト切替やデータの複写が不要となり，高速であるという利点がある．しかし，プログラミングや保守が難しくなること，カーネルが肥大化するといった欠点がある．

ユーザレベルのサーバとしての実装は，アプリケーションとの間でコンテクスト切替が起こり，またデータの共有が複雑になるため，速度面が不利になるという欠点がある．しかし，プログラミングや保守が容易であり，不要な場合には取り除くことができる．

ライブラリによる実現では，アプリケーションによりサービスが実行されるため高速であり，またユーザレベルであるためプログラミングや保守がしやすい．しかし，アプリケーションによる低いレベルのアクセスを許す必要があるため，セキュリティが問題となる．また，サービス全体で情報を持つことができない．

これら 3 つを組み合わせることも可能である．例えば，セキュリティが必要な部分や全体的な処理をユーザレベルサーバで，性能が問題になる部分をライブラリで実現するといったことが考えられる．例えば，CMU はユーザレベルライブラリと UNIX サーバの組み合わせによる TCP/IP を Mach 3.0 MK83+UX42 に実装している [7]．性能が重視されるデータ送受信はユーザレベルライブラリのみで直接ネットワークデバイスドライバをアクセスして通信を行ない，ソケットインタフェースはセキュリティや情報共有などのためユーザレベルサーバである UNIX サーバとの通信を伴う．また，ワシントン大学も同様の構成でプロトコルライブラリを実現している [14]．この方式により，ユーザレベルサーバによる実現とライブラリによる実現の両方の利点を得ることができ，カーネル内実装と同等の性能を得られる．CMU の結果 [7] によると，ユーザレベルサーバのみによる実装がカーネル内実装に比べて 3 割程度スループットが小さくなっていたのに対し，ユーザレベルサーバとユーザレベルライブラリの組合せによりカーネル内実装と遜色のないスループットが得られている．

3 設計

本節では，プロトコルサーバの構成と，設計上考慮した手法について述べる．

3.1 ST-II プロトコルサーバの構成

カーネル内実装では，カーネルが肥大化し，プログラミングや保守が困難になるという問題点がある．また，ユーザレベルライブラリによる実装では，高速な処理が期待できるが，例えばアプリケーションのバグにより，予定していたより速いレートでデータの送出が行なわれると，予約より多くの資源を使用することになり，問題が起こる．これらに加え，ST Control Message Protocol (SCMP，ST-II の制御プロトコル) が複雑で状態を持つ必要があることや，データ送受信のレート制御，CPU の予約を行なうため，ユーザレベルのサーバとして設計を行なった．以後、ユーザレベルサーバによる ST-II プロトコル処理系を，ST-II プロトコルサーバと呼ぶ．

ST-II プロトコルサーバは，図 1に示すように，次の 7 つのスレッド群によって構成される．

メインスレッド　ST-II サーバで最初に起動されるスレッドで，SCMP サーバスレッド，ディスパッチャスレッドを作成する．また，アプリケーションと通信し，API スレッドを作って通信用のポートを返す．

API(Application Programming Interface) スレッド　アプリケーションとのインタフェースを行なう．アプリケーションからの要求を受け取ると，その内容に応じて制御スレッドや送信スレッドを生成したり，それらにメッセージを送る．またアプリケーションに結果を返す．図 1で四角が重なっているのは，要求に応じて API スレッドが複数作られることを意味する．

SCMP サーバスレッド　SCMP のうち個々のストリームではなく全体に関係する処理をする．

制御スレッド　SCMP のうち個々のストリームに関する処理をする．API スレッド，ディスパッチャスレッド，ネットワークデバイスドライバと通信する．非周期的スレッドとして実装する．ストリームごとにひとつ作られる．

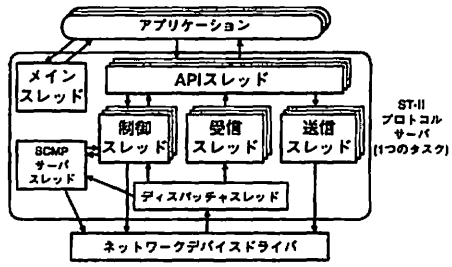


図 1: ST-II プロトコルサーバの構成

受信スレッド　ストリーム確立時に，中継エージェントとターゲットエージェントにストリームごとに作られる．受信パケットディスパッチャによってリングバッファに入れられたパケットを処理し，送信スレッドまたは API スレッドに渡す．周期的スレッドとして実装する．周期をストリームのレートに合わせることにより，受信のための CPU 資源の予約を行なう．

送信スレッド　ストリーム確立時に，送信エージェントと中継エージェントに送り側のホップごとに作られる．API スレッドまたは受信スレッドからデータパケットを受け取り，ネットワークデバイスドライバに渡す．周期的スレッドとして実装する．周期をストリームのレートに合わせることにより，送信のための CPU 資源の予約を行なう．

ディスパッチャスレッド　ネットワークデバイスドライバからパケットを受け取り，内容に応じて受信スレッドまたは制御スレッドに渡す．中継エージェントは受信スレッドが送信スレッドにデータパケットを渡すことにより実現される．複数のターゲットへの通信は，ひとつの受信スレッドから複数の送信スレッドにデータパケットを渡すことにより実現される．

3.2 高速化の手法

スループットは，全体のうち最も小さい部分に制限されるため，広帯域ネットワークにおいては，プロトコル処理のスループットを上げる必要がある．また，テレビ会議システムのように対話的なアプリケーションをサポートするためには，全体の遅延時間を小さくする必要があるため，プロトコル処理の

遅延時間も小さくする必要がある．つまり，「軽い」プロトコル処理が必要である．

連続メディアのデータパケットは一般に大きいため，プロトコル階層の間の複写を減らすことは，伝統的なデータ通信に比較して効果的である．そこで，サーバ内のスレッド間ではポインタ渡しによるデータの共有を，サーバとアプリケーションの間では仮想記憶ページを交換するプリミティブ[10]をもちいることにより，パケットの複写を避け，プロトコル処理の高速化を図る．

パケットの受け渡しには，リングバッファをもちいる[10]．連続メディア通信において，デッドラインを守れない古いメッセージは意味を持たない．そこで，意味を持たなくなった古いデータを捨て，新しいデータを入れることができる．このため，使用する記憶域の大きさは一定になり，コネクション設定時に予約することができる．

3.3 資源予約の手法

ST-IIでは，資源予約のためにFlowSpecという枠組を用意している．FlowSpecは，優先度，エラー率，最大遅延時間，パケットサイズ，パケットレート，最小帯域幅の希望値，最悪値などから構成される．この要求を保証するため，ネットワークの帯域幅のみではなく，メモリとCPU資源に対して予約を行なう．これはスケジューラ，リングバッファ管理により実現する．ネットワークの帯域予約は，ネットワーク自体が持つ機能を利用する．

CPUの予約のため，パケットの送受信にReal-Time Machの周期的スレッドを使用する．この周期的スレッドは，ストリーム確立時に生成される．Real-Time Machのリアルタイムスケジューリングにより，遅延時間の上限が保証される．このとき，スケジューラビリティを予測し，スケジュールできない場合に，新しいスレッドを拒否するようなスケジューリングポリシーを使用することができれば，CPU帯域幅を予約できる．

4 実装

CPUにIntel 486を採用したIBM-PC互換機と，MIPS R3000Aを採用したDECstation 5000/125をターゲットとして実装を行なっている．オペレーティングシステムは双方ともReal-Time Mach 3.0 MK83である．以下において，APIおよび現在の実装の状況について説明する．

4.1 API

APIにはMIG(Mach Interface Generator)にリアルタイム通信向きの変更を加えられたRT-MIGを使用し，以下に示すものを用意している．

`st_create(mach_port_t, mach_port_t *)`
: 新しいSCMPスレッドを作ることを要求し，それとの通信用のポートを受け取る．

`st_bind(mach_port_t, struct sockaddr_st2 *, int)`
: ローカルなストリームのパラメータをサーバに通知する．

`st_connect(mach_port_t, struct sockaddr_st2 *, int)`
: 接続先のパラメータを渡し，CONNECTメッセージ送出を要求する．

`st_recv_accept(mach_port_t, struct st_param *, int *)`
: CONNECTに対するACCEPTメッセージに入っている，FlowSpecなどのパラメータを第二引数で受け取る．

`st_listen(mach_port_t, struct st_param *, int *)`
: CONNECTメッセージを待ち，その中のFlowSpecなどのパラメータを受け取る．

`st_accept(mach_port_t, struct st_param *, int *)`
: パラメータに入っている条件でCONNECTに対するACCEPTメッセージ送出を要求する．

`st_send(mach_port_t, char *, int)`
: データを送信する．

`st_recv(mach_port_t, char *, int *)`
: データを受信する．

`st_close(mach_port_t)`
: 通信を終了する．

4.2 実装状況

現在実装が完了しているプロトコル機能は，通信ができる最低限のSCMPと，データの送受信である．

エラー処理などは行なっていない．現在の実装ではFlowSpecを受け付けるが，実際に使っているのは最小のパケットレートのみであり，ネットワークの帯域予約などは行なっていない．これは，現在イーサネットのみ利用可能であり，イーサネットでは帯域予約が原理上不可能であるためである．パケット間の最小間隔を制御するために，送信スレッドを周期的スレッドとして実装しているが，その周期の決定は，最悪値をもちいて行なっている．デッドラインをミスした場合の処理は行なっていない．スケジューリングにはレートモノトニックスケジューリングポリシー[6]をもちいているため，できるだけ時間制約を守るようスケジューリングされるが，CPUの予約はできていない．また，前節で説明した，アプリケーションとサーバ間の仮想気おくべー字の交換プリミティブを利用していない．

なお、実装にあたってx-kernel, NPSは使用していない．x-kernelはリアルタイム通信をサポートしていない．また、NPSがいくつかのワーカースレッドを作ってそれを複数のセッションで共有するモデルを採用しているのに対して，本稿の実装では3.3節で述べているようにストリームごとに周期的スレッドを割り当てるため，構造が合わない．

5 評価

前節で説明した実装について，簡単な評価実験を行なった．実験の概要と結果について述べる．

5.1 実験環境

評価にもちいたハードウェアは，送信がIBM-PC互換機(CPUはIntel 486DX2，内部クロック66MHz)であり，受信が同様のIBM-PC互換機にAlpha Logic Inc.のSTAT!タイマーボード(以下STATと呼ぶ)を装着したものである．STATを使用することにより，最高250ナノ秒の精度が得られ，また一回あたり約30マイクロ秒で時刻が測定できる．したがって，受信側での時刻の測定には十分な精度であるといえる．一方，送信側であるが，送信側のパケット間の時間は，リアルタイムクロックの割り込みを契機に起動される周期的スレッドによって決められるため，同様に粒度の細かい時計が必要となる．ここにもSTAT

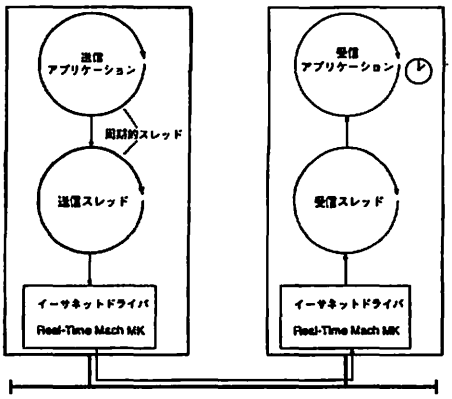


図 2: 評価実験の概要

を付加したIBM-PC互換機が使用できれば良いのであるが，用意することができなかったため，内蔵のシステム時計を使用した．

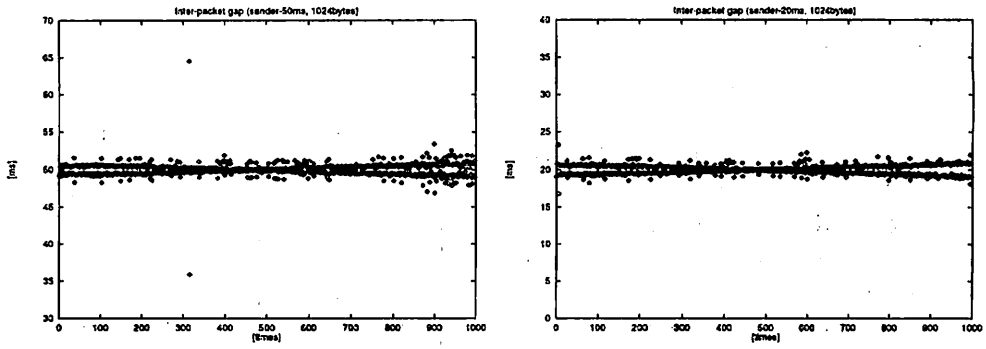
5.2 実験方法

実験の概要を図2に示す．アプリケーションがST-IIを使って同じ大きさのパケットを一定の間隔で送出し，受信側でパケットの到着間隔を測定した．送り出し側のアプリケーションは，最低データレートに周期を合わせた周期的スレッドとした．したがって，ST-IIサーバ，アプリケーションとも同じ周期で動作する．現在受信側は周期的スレッドを使っていないため，サーバはMach Packet Filter[18]によってパケットが渡されるのを待ち，受け取ると即座に受信待ちのアプリケーションのスレッドに渡す．この時点でアプリケーションはタイマーボードから時刻を読み，記録する．最後に，記録した時刻の差を出力する．

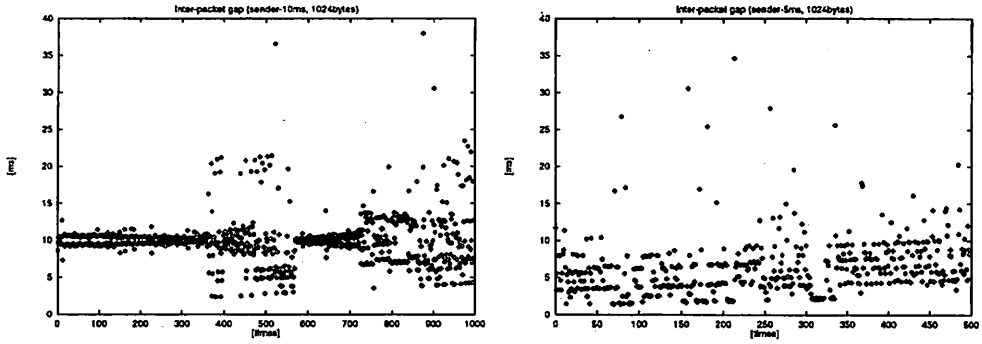
5.3 結果と考察

パケットの周期を50ミリ秒，20ミリ秒，10ミリ秒，5ミリ秒にした場合の測定結果を図3に示す．

これらから，周期20ミリ秒程度であれば，パケットの到着間隔のゆらぎが約2ミリ秒以内に抑えられていて，問題なく制御できているが，周期10ミリ秒であると乱れが生じ，周期5ミリ秒の場合はばらつきが大きくなることが観察された．また，周期10ミリ秒，5ミリ秒では場合によって途中でシステムが



(a) 送信間隔 50 ミリ秒　　(b) 送信間隔 20 ミリ秒



(c) 送信間隔 10 ミリ秒　　(d) 送信間隔 5 ミリ秒

図 3: パケット受信時刻の差

ハングアップしてパケットの送出ができなくなるという現象が見られた.

これは，1 周期あたりにかかる処理時間が大き過ぎるといった要因によって，送信スレッドまたは送信アプリケーションのスケジュールが不可能になっていると考えられる．これを防ぐには，第 3.2節で提案している手法を実装することによって処理を高速化することと，デッドラインミスの監視が必要であると考えられる．しかし，分散マルチメディアを扱うシステムという前提で考えると，動画の表示処理などがプロトコル処理より CPU 資源を多く消費すると考えられる．したがって，評価に用いたシステムよりも高性能なものが必要である．

別の要因として，システムの時計の粒度が挙げられる．現在は Real-Time Mach のプリミティブにより時計の粒度を 0.5 ミリ秒に指定しているが，PC の内部時計の精度が良くないため，10 ミリ秒，5 ミリ秒といった細かな制御には向かないと考えられる．これは，STAT のような精度の良い時計を入手することにより解決可能と考えられるが，今回使用した PC に限らず，現在一般に使われているワークステーションに内蔵のシステム時計の粒度は荒く，本稿で議論しているようなプロトコルの実装には向かない．一方，時計の粒度を細かくしてそれを元にスケジューリングしようとすると，頻繁に時計割り込みが起こることになり，システムに対する負担が大きい．時計割り込みの粒度を決定するときには，これらのバランスを考える必要がある．

ひとつの解決法に，ハードウェア化がある．例えば，現在の多くの ATM インタフェースボードのように，デバイスドライバ，ボードへの転送までは大きな単位で渡し，ボードで細かな単位に分け，一定間隔で送信するという方法も考えられる．この方法の問題点は，遅延が大きくなること，ソフトウェア

による柔軟な処理が難しいことである．ハードウェア化にもバランスが必要である．

6 結論と今後の課題

本稿では，ST-II を Real-Time Mach のユーザレベルサーバで実現する上での議論を行ない，現時点での実装について評価を行なった．パケット間の時間間隔が 20 ミリ秒までであればそのゆらぎを 2 ミリ秒以内に抑えられることが示せた．また，現在利用しているハードウェアに不足している点を指摘した．

今後は，まず本稿で議論した手法をすべて実装した上で，評価を行なう予定である．デッドラインをミスしているか監視し，ミスが多ければサービス品質を落とし，少なければ品質を上げる手法であるセルフスタビライゼーション [2] や，プロセッサ予約 [9] などを検討していく予定である．

次に，ネットワークの帯域予約には，ATM-LAN の利用を予定している．現在，情報処理振興事業協会・創造的ソフトウェア育成事業「次世代マイクロカーネルプロジェクト」において，Real-Time Mach 用の ATM のデバイスドライバを開発中であり，これが完成し次第使用する予定である．また，NTT のマルチメディア共同利用実験の一貫として慶應義塾大学と NTT とで慶應情報スーパーハイウェイ [8] を構築しているため，ATM-LAN だけではなく，ATM-WAN 環境での評価を行なうことが可能である．

また，ST-II の新しい仕様である ST2+[1] への対応を予定している．

最後に，ST-II の実装を評価する方法も今後考えて行く必要がある．本稿ではパケットの到着間隔を測定することによったが，今後は帯域予約型プロトコルに適した手法を検討する．

謝辞

本研究を行なうにあたり，貴重なご討論をいただいている，慶應義塾大学環境情報学部の徳田英幸教授はじめとする次世代マイクロカーネル研究プロジェクトの皆様，NTT 情報通信研究所の皆様に感謝いたします。

なお，本研究の一部は，情報処理振興事業協会・開放型基盤ソフトウェア研究開発評価事業「マルチメディア統合環境基盤ソフトウェアの研究開発」プロジェクトへの研究協力として行なわれました．

参考文献

[1] Delgrossi, L. and Berger, L.: Internet Stream Protocol Version 2 (ST2) Protocol Specification - Version ST2+ (1995), RFC 1819.

[2] 松渡，徳田: Real-Time Mach 3.0 における連続メディアサーバの実験 — QOS 制御を採り入れた Quick Time Player の評価 —，情報処理学会研究報告，93，58，75–82 (1993).

[3] Hehmann, D., Herrtwich, R. G., Schulz, W., Schutt, T., and Steinmetz, R.: Implementing HeiTS: Architecture and Implementation Strategy of the Heidelberg High-Speed Transport System, in *Proceedings of 2nd International Workshop on Network and Operating System Support for Digital Audio and Video*, Springer-Verlag (1991).

[4] Herrtwich, R. G. and Delgrossi, L.: Beyond ST-II: Fulfilling the Requirements of Multimedia Communication, in *Proceedings of 3rd International Workshop on Network and Operating System Support for Digital Audio and Video*, Springer-Verlag (1992).

[5] Hutchinson, N. C. and Peterson, L. L.: Design of the *x*-Kernel, in *Proceedings of SIGCOMM 88*, ACM Press (1988).

[6] Lehoczky, J., Sha, L., and Ding, Y.: The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior, Technical report, Department of Statistics, Carnegie Mellon University (1987).

[7] Maeda, C. and Bershad, B. N.: Protocol Service Decomposition for High-Performance Networking, in *Proceedings of the 14th ACM Symposium on Operating System Principles*, 244–255 (1993).

[8] 松井，内海，松渡，中村，成田，細川，徳田: ATM-LAN 環境と ATM-WAN 環境の転送特性の比較，情報処理学会研究報告，95，73，99–104 (1995).

[9] Mercer, C. W., Savage, S., and Tokuda, H.: Processor Capacity Reserves for Multimedia Operating Systems, Technical Report CMU-CS-93-157, Carnegie Mellon University (1993).

[10] 盛合，木原，南部: 連続メディアオブジェクトに対応したメモリ管理機構について，情報処理学会研究報告，94，39，73–78 (1994).

[11] 中島，徳田: 予測可能な通信を提供するユーザレベル・ネットワークシステム，日本ソフトウェア科学会第 10 回大会論文集 (1993).

[12] Partridge, C. and Pink, S.: An Implementation of Revised Internet Stream Protocol(ST-II), *Internetworking: Research and Experience*, 3, 1 (1992).

[13] Peterson, L., Hutchinson, N., O'Malley, S., and Rao, H.: The x-kernel: A Platform for Accessing Internet Resources, *IEEE COMPUTER* (1990).

[14] Thekkath, C. A., Nguyen, T. D., Moy, E., and Lazowska, E. D.: Implementing Network Protocols at User Level, in *Proceedings of SIGCOMM 93*, 64–73, ACM Press (1993).

[15] 徳田: 分散リアルタイムシステムのための OS アーキテクチャ, 情報処理, 35, 1, 18–25 (1994).

[16] Tokuda, H., Tobe, Y., Chou, S. T.-C., and Moura, J. M. F.: Continuous Media Communication with Dynamic QOS Control Using ARTS with an FDDI Network, in *Proceedings of SIGCOMM 92*, ACM Press (1992).

[17] Topolcic, C., Casner, S., Lynn, C., Park, P., and Schroder, K.: Experimental Internet Stream Protocol, Version 2 (ST-II) (1988), RFC 1190.

[18] Yuhara, M., Bershad, B. N., Maeda, C., and Moss, J. E. B.: Efficient Packet Demultiplexing for Multiple Endpoints and Large Messages, in *USENIX Winter Conference*, USENIX Association (1994).

[19] Zhang, H. and Fisher, T.: Preliminary Measurement of the RMTP/RTIP, in *Proceedings of 3rd International Workshop on Network and Operating System Support for Digital Audio and Video*, Springer-Verlag (1992).

[20] Zhang, L., Deering, S., Estrin, D., Shenker, S., and Zappala, D.: RSVP: A New Resource ReSerVation Protocol, *IEEE Network*, 8–18 (1993).