マルチメディア通信と分散処理 78－3
（1996．9．27）

# Internetにおけるマルチメディア通信プラットホームの構築

普天間 智† 三嶋 正弘‡ 山口 英†

†奈良先端科学技術大学院大学 情報科学研究科
‡(株) 日本総合研究所 インフォメーション・テクノロジ事業部

**内容梗概**

Internet 上でのマルチメディア通信において今後一層重要になると考えられるのはアプリケーションの相互接続性である。

本論文では Internet におけるマルチメディア・アプリケーションの相互接続性を支援する、拡張性に優れたプラットフォーム PRISM(PRactical Inter-videoconference System Model) を提案する。PRISM を使用することで、様々な異種マルチメディア・アプリケーションが同時に 1 つのセッションに参加することが可能になる。PRISM では、アプリケーションのデータを構成する複数の PRISM サーバが相互に接続して、全体として PRISM ネットワークを形成する。

PRISM サーバはアプリケーションレベルの独立した複数のプログラム・モジュールの集合として実装した。モジュール間のインタフェースを定義することでモジュールを置き換え可能にし、新しいアプリケーションへの対応を容易にした。

PRISM サーバを一般的な能力のワークステーションで動かした結果、多少問題は残るが、実用的な性能を出し得ることが示された。

# Design and Implementation of a Platform for MultiMedia Communication on the Internet

Satoshi Futenma† Masahiro Mishima‡ Suguru Yamaguchi†

†Graduate School of Information Science,
Nara Institute of Science and Technology
‡The Japan Research Institute, Limited,
Department of Information Technology

**abstract**

Interoperability will be more important for multimedia in the near future, because networks are already interconnected and several standard protocols for realtime communication are now ready on the Internet.

This paper proposes PRISM(PRactical Inter-videoconference System Model) as a platform on which various multimedia applications can communicate each together even if their data formats and session protocols are imcompatible among them.

A PRISM server is implemented as a group of programs. It has an extensible flexibility so that modules in the PRISM architecture can be added, modified or replaced independently.

The result of experiment concludes that a PRISM server can work in resonable performance on workstations with moderate CPU power.

# 1 はじめに

近年、ネットワークの高速化[1]、計算機の性能向上、音声や画像の圧縮技術の確立[2]などにより、マルチメディア通信における技術の進歩は著しい。こうした状況の中でnv[3]、vat[4]、ivs[5]、vic[6]、CU-SeeMe[7]など多数のマルチメディア通信アプリケーションが開発され、Internet上で広く利用されている。しかしながら、これらのアプリケーションを使って、マルチメディア通信を行なうときに、考慮すべき問題がいくつかある。

まず、アプリケーションごとに使われているプロトコルやデータフォーマットが異なっているので、異なるアプリケーション同士を相互に接続できないことが多い。そのため、通信したい相手によって、アプリケーションを使い分けなければならないという問題が生じる。アプリケーションによっては、特定のプラットホームでしか動かないものもあり、異なるアプリケーションの相互接続を可能にするフレームワークが必要である。

また、IPマルチキャストで通信を行なう場合、Internet上に存在する多数のネットワークの帯域の差が問題となる。IPマルチキャストに依存した多点間通信では、広帯域のネットワークに合わせて大量のデータを送信すると、狭帯域のネットワークでは輻輳が生じてしまう。帯域に応じてデータ量をコントロールすることが必要である。

本研究では、マルチメディア通信におけるアプリケーションの違いやネットワークの帯域の違いを克服するためのプラットホームとなるPRISM(PRactical Inter-videoconfernce System Model)を提案し、実装を行った。PRISMを使用すると、同じアプリケーションと通信を行う場合と同じように、異なるアプリケーションとも通信を行うことができるようになる。PRISMでは、アプリケーションと直接通信する部分を独立したモジュールとして開発できるように設計を行った。また、共有メモリを利用することによって、モジュール間の通信のオーバヘッドを軽微に抑えた。

# 2 PRISMの機能

PRISMは、多点間通信、ネットワーク帯域を考慮したトラヒックの削減、アプリケーション間のプロトコルやデータフォーマットの変換、マルチメディアデータの加工を実現する。

## 2.1 多点間通信

PRISMでは、アプリケーションの接続したPRISMサーバが相互に通信することによって、アプリケーションに多点間通信の機能を提供する。PRISMサーバとアプリケーション間でもPRISMサーバ間でも、ユニキャスト、マルチキャストのどちらでも通信ができるようになっており、PRISMサーバ内ではユニキャストとマルチキャストの変換も行われる。これにより、マルチキャストに対応していないアプリケーションでも、PRISMを介して容易にMBONEに参加できることになる。

## 2.2 トラヒックの削減

PRISMでは、ネットワークの輻輳を防ぐために、各PRISMサーバで送信するデータ量を削減することができる。マルチメディアデータは既に圧縮されて送られていることが多い。圧縮されているデータ量を減らそうとすると、データの品質を下げなければならない。調整可能な品質としては、フレームレートと画質の2つが考えられる。

PRISMでは、フレームレートと画質の両方とも調整可能である。

## 2.3 プロトコル及びデータフォーマットの変換

異なるアプリケーション同士の接続には、プロトコル及びデータフォーマットの変換が必要である。

PRISMサーバで様々なフォーマットを、中間フォーマットを介して別のフォーマットに変換することで、複数のフォーマット間の相互変換を効率的に行う。中間フォーマットを用いることで、新たなフォーマットに対応する場合に、アプリケーション・フォーマットから中間フォーマットへ変換するデコーダとその逆を行うエンコーダを用意するだけですむ。

## 2.4 データの加工

PRISMでは、複数の音声や画像を合成して送ることができる。また、マルチメディア通信の経路上で画像データにおけるノイズの軽減や、音声データにおけるエコーの除去を行うことも可能である。

# 3 PRISMサーバの設計

この章では、PRISMネットワークを構成するPRISMサーバの設計について述べる。

PRISMでは、拡張性を重視したために、全てがモジュール構造になっている。モジュールの追加により、新たなアプリケーションへの対応、データ加工やトラヒック削減などの機能追加、置換が簡単にできる。

## 3.1 モジュール構成

PRISMサーバは、マネージャ、プロトコルモジュール、プロトコルアダプタ、メディアプロセッサの4種類のモジュールから構成される(図1)。

### 3.1.1 マネージャ

マネージャは、PRISMサーバにおける全てのモジュールの管理を行うPRISMサーバで核となる部分である。

マネージャは、必要なプロトコルモジュールやプロトコルアダプタを起動し、各モジュールの通信パラメータを初期化する。モジュール終了時には終了処理を行う。実際のデータ処理は、プロトコルモジュールやプロトコルアダプタが行うので、データ通信に関してマネージャは何も行わない。

### 3.1.2 プロトコルモジュール

プロトコルモジュールは、特定のアプリケーションのプロトコルを理解し、アプリケーションと直接通信を行うモジュールである。このモジュールは、アプリケーション1種類につき1つ用意される。

アプリケーションからデータパケットを受け取ったプロトコルモジュールは、ヘッダおよびデータを中間フォーマットに変換し、PRISMサーバ内の他のプロトコルモジュールへ転送する。プロトコルの中間フォーマットにはRTP及びRTCP[8]を、データの中間フォーマットには、音声の場合はPCM $\mu$-law形式を、画像の場合はYUV形式を用いる。他のプロトコルモジュールから転送されたデータを受け取ったプロトコルモジュールは、中間フォーマットを各々の理解するプロトコルやデータフォーマットに変換して、アプリケーションに送信する。必要があれば、ユニキャスト／マルチキャストの変換も行う。

不必要なデータ転送やフォーマット変換のオーバヘッドを避けるために、アプリケーションが1つも接続されていないプロトコルモジュールへはデータの転送は行われない。同じアプリケーションに対しては、変換を行わずにそのままデータが転送される。

PRISMサーバ間でのデータ転送が生じたときは、プロトコルモジュールは、アプリケーションから受け取ったパケットをそのまま、アプリケーションプロトコルを使って、他のPRISMサーバの同種プロトコルモジュールへ転送する。プロトコルやデータフォーマットの変換は、アプリケーションが直接接続しているPRISMサーバ側で行われる。PRISMサーバ相互に同種のプロトコルモジュールがない場合には、次節で述べるプロトコルアダプタを使って通信を行う。プロトコルオブジェクトは、データを中間フォーマットに変換して、プロトコルアダプタに転送する。

### 3.1.3 プロトコルアダプタ

プロトコルアダプタは、プロトコルモジュールの機能を補完するモジュールである。通信を行いたいPRISMサーバ間に同種のプロトコルモジュールが存在しないこともある。このような場合に、プロトコルモジュールに代わって、通信を行うのがプロトコルアダプタである。プロトコルアダプタ間の通信は、中間フォーマットを使って行われる。

### 3.1.4 メディアプロセッサ

メディアプロセッサは、データの加工を行うモジュールである。このモジュールは、プロトコルモジュールやプロトコルアダプタからの要求に基づき、中間フォーマットのデータに対して特定の加工を行う。1つのメディアプロセッサが行う加工の種類は原則として1つである。

## 3.2 モジュール間通信

各モジュールは、同じインターフェイスを持つ新しいモジュールと交換可能でなければならない。特に、プロトコルモジュールはアプリケーションに依存した実装になるため、他の部分とは独立して開発できなければならない。したがって、モジュール間プロトコルを明確に定めておく必要がある。

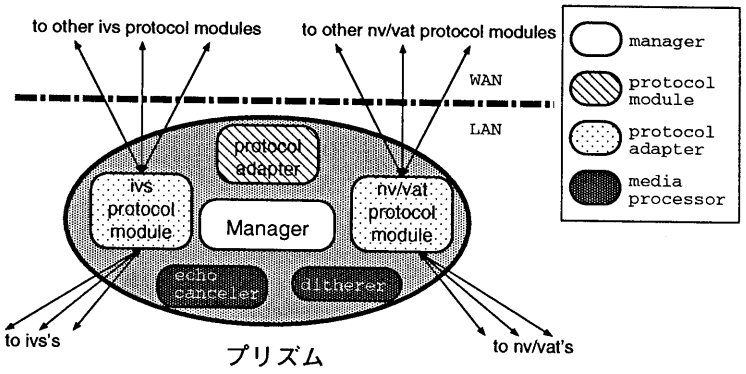


図 1: PRISM サーバの構成

#### 3.2.1 モジュール間通信の初期化

マネージャの通信パラメータは、各モジュールが起動されるときに、引数として渡される。

その他のモジュールの通信パラメータは、各モジュールから、まずマネージャに報告され、全てのモジュールから報告があった後でマネージャを介して各モジュールへ渡される。

#### 3.2.2 モジュール間通信の終了

各モジュールは実行を終了するときに、他の全てのモジュールに終了通知を出す。マネージャ以外のモジュールが終了通知を行った場合、他のモジュールは、そのモジュールとの通信を終了する。マネージャが終了通知を行った場合は、全てのモジュールは直ちに終了しなければならない。この場合、新たな終了通知は行われない。

#### 3.2.3 メディアデータの送受

メディアデータやメディア制御情報など、モジュール間で送受されるデータは、共通ヘッダを付けて送られる。共通ヘッダには、データタイプ（メディアデータ又は制御情報）、メディアタイプ（音声又は画像）、データ長の情報が含まれている。

## 4 PRISM の実装

SGI 社の INDY ワークステーション上の IRIX 5.3 で開発を行い、動作確認を行った。共有メモリとセマフォに関しては、System V 系のシステムコールに依存している。

### 4.1 モジュール

マネージャ、ivs プロトコルモジュール、nv/vat プロトコルモジュールの 3 つのモジュールを実装した。各モジュールは別々のプログラムとして実装した。モジュール間通信は、ソケットを使って実装した。データサイズの大きい画像データの中間フォーマットは共有メモリに置き、その ID を渡すようにして、モジュール間通信におけるオーバヘッドを回避した。

#### 4.1.1 マネージャ

今回マネージャに実装された機能は、全てのプロトコルモジュールの起動、全てのプロトコルモジュールに各モジュールのソケットアドレスなどの情報の送信、終了時における全てのプロトコルモジュールへの終了通知の 3 つである。

#### 4.1.2 プロトコルモジュール

アプリケーションからパケットを受け取ったときに処理を行うデコードルーチンと他のプロトコルモジュールからパケットを受け取ったときに処理を行うエンコードルーチンを実装した。これらは、nv 及び ivs のソースに変更を加えることにより実装した。アプリケーションからデータを受け取った場合は、入力ポートによって、他のプロトコルモジュールからデータを受け取った場合は、モ

ジュール間通信のヘッダにより、適切なルーチンが呼ばれる。

***_decode()**
アプリケーションから受け取ったデータパケットを中間フォーマットに変換する。プロトコルヘッダとデータ部の両方の変換が行われる。**video_decode** ルーチンと **audio_decode** ルーチンがある。

***_encode()**
他のモジュールから受け取った中間フォーマットのデータをアプリケーションに合わせたフォーマットに変換する。**video_encode** ルーチンと **audio_encode** ルーチンがある。

***_decode_ctrl()**
アプリケーションから受け取った制御パケットを中間フォーマットである RTCP パケットに変換する。**video_decode_ctrl** ルーチンと **audio_decode_ctrl** ルーチンがある。

***_encode_ctrl()**
他のモジュールから受け取った RTCP パケットをアプリケーションに合わせた制御パケットに変換する。**video_encode_ctrl** ルーチンと **audio_encode_ctrl** ルーチンがある。

どのルーチンも、アプリケーションパケットへのポインタ、中間フォーマットへのポインタを引数として取る。video_decode ルーチン、video_encode ルーチンでは、中間フォーマットを共有メモリ上に格納したので、ポインタとして共有メモリの ID が渡される。

プロトコルモジュールにおいて、以上の 8 つのルーチンがアプリケーションに依存した部分になる。新たなアプリケーションへ対応するには、これらのルーチンを用意すればよい。

**ivs プロトコルモジュール**

ivs は、送信者が複数あると、プログラムが fork() して送信者ごとにデコーダを割り当てるという実装になっている。そのため、ivs プロトコルモジュールでは、送信者ごとにデコーダプロセスを起動する実装になっており、ivs プロトコルモジュール本体とデコーダプロセスとの間でプロセス間通信が生じている。このプロセス間通信はパイプで実装した。

**nv/vat プロトコルモジュール**

nv と vat は、別々のアプリケーションであるから、本来なら別々のプロトコルモジュールで対応すべきだが、今回は 1 つのプロトコルモジュールとして実装した。

### 4.2 性能測定

PRISM を介して接続された ivs と nv/vat との間の通信を行い、PRISM サーバ内でのデータの遅延を測定した (図 2)。サーバとクライアントには、SGI INDY IRIX 5.3(R4600、100MHz、32MB) を、ネットワークには FDDI を使った。

測定結果、以下のことがわかった。

- 音声の遅延は、30msec 以内に収まっており、十分に実用的である。
- 画像において、ivs から nv への遅延は十分に小さかったが、ivs から nv への遅延は極端に大きいものになっている。これは、ivs プロトコルモジュールで、デコーダを別プロセスとして実装したためだと思われる。

## 5 考察、今後の課題

本研究では、各モジュールの独立性や拡張性を重視して開発を行ったため、プロセスのコンテキスト切替えやプロセス間通信がオーバヘッドになることや、実行を制御するスケジューリングが OS に任されるなど、実行効率に問題を残した。また、性能についても、詳細な測定が必要であろう。

PRISM は設計上、同一セグメント上に PRISM サーバ又は異なるアプリケーションがある場合や、同種のアプリケーションでもそれぞれが 1 対 1 で PRISM サーバと通信している場合には、トラヒックの増加を招いてしまう。PRISM は、別セグメントにある異種のアプリケーションの相互接続を行うプラットホームとして、利用するのがよいと思われる。

今後の課題としては、設計の抽象化が考えられる。現状では、実装に依存した部分も多い。特に、画像通信では、共有メモリを使用しているが、共有メモリも普通のメモリも統一的なインターフェイスで扱えるのが理想的である。

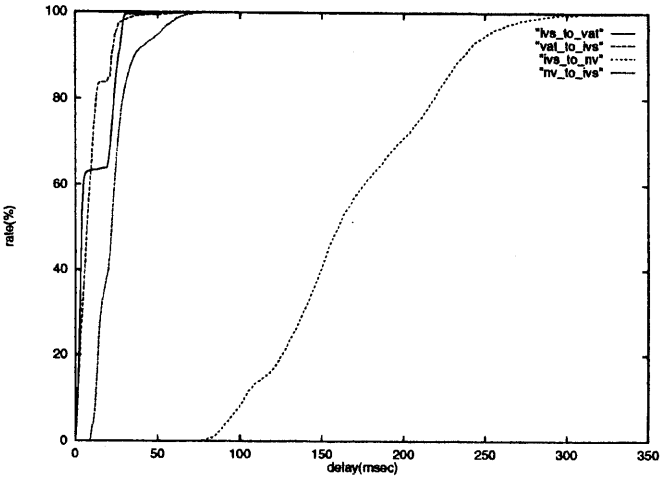


図 2: PRISM サーバ内でのデータ遅延

## 6 まとめ

本研究では、Internet のマルチメディア通信において、異種アプリケーション間の相互接続性を提供し、拡張性に富んだプラットホーム PRISM の提案、実装、評価を行った。

PRISM の実装では、IP マルチキャストに依存しない多点間通信、ネットワークの帯域に応じたデータ量の削減、H.261 形式と nv 形式間の画像フォーマットの相互変換、vat と RTP 間のプロトコルの相互変換を実現した。

今後の課題として、設計及び実装の改善、設計の抽象化を行う必要があることを述べた。

## 謝辞

本研究を行う機会を与えて下さり、御指導くださった、奈良先端科学技術大学院大学情報科学研究科の山口英 助教授に深く感謝いたします。

また、研究に協力して頂いた (株) 日本総合研究所インフォメーション・テクノロジ事業部の三嶋正弘氏に感謝いたします。

## 参考文献

[1] Craig Partridge. *Gigabit Networking*. Adison Wesley, 1993.

[2] 藤原 洋. 最新 *MPEG* 教科書. アスキー出版局, 1994.

[3] Ron Frederick. Experiences with real-time software video compression. *ftp://parcftp.xerox.com/pub/net-research/nv-paper.ps*, July 1994.

[4] Van jacobson and Steven McCanne. vat - LBNL audio conferencing tool.

[5] Thierry Turletti. The INRIA videoconferencing system (ivs). *ConneXions - The Interoperability Report*, 8(10), October 1994.

[6] Steven McCanne and Van Jacobson. vic: A flexible framework for packet video. In *Proceedings of ACM Multimedia '95, San Fracisco, CA USA*, November 1995.

[7] Tim Dorcey. CU-SeeMe desktop videoconferencing software. *ConneXions - The Interoperability Report*, 9(3), March 1995.

[8] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time application. *RFC1889*, January 1996.