修士論文　　2005年度 (平成17年度)

# インターネットを用いた<br>複数ストリーム同期再生機構の設計

慶應義塾大学大学院 政策・メディア研究科

久松 剛

修士論文要旨 2005 年度 (平成 17 年度)

# インターネットを用いた複数ストリーム同期再生機構の設計

本研究では，インターネットを媒体とする映像配信環境において収録・再生機材，ネットワーク，映像フォーマット，圧縮方式などといった処理環境の違いによって発生する処理遅延に着目し，複数ストリームを同期伝送するための処理遅延吸収機構の構築を行った．現在利用されているインターネットを伝送媒体とするリアルタイムストリーミングによる映像配信・中継活動は，主に単一ストリームを 1 対 1，1 対多に伝送することを目的としている．しかし，時間軸同期が要求される複数ストリームの配信においては，個々のストリームで行われる処理遅延が環境によって変化するため，遅延吸収機構による同期機構が必要となる．

複数ストリーム同期再生機構を実現するうえで，処理要因によって発生する処理時間，すなわち遅延時間の分析を行った．1) 送信端末から受信端末までに至る入力部，2) コンテンツフォーマットのデコードやトランスコーディングなどの変換部，3) 受信端末出力から収録機器を経てユーザに対する出力までに至る出力部　の 3 状態による処理遅延を算出し，各ストリームの処理時間を求めた．

各処理時間によって生じる遅延時間は，用いるフォーマットや収録機器，ネットワーク環境によって異なる．処理時間の異なる複数ストリームの同期を行う上で，各々の遅延時間合計である再生遅延に対し，時間軸吸収機構を適応する事によって包括的同期を行った．

提案した機構に基づき 3 つのモジュールを作成した．1)Negotiation Module では端末間協調を行うことにより，受信したストリームのタイムスタンプに対する再生タイミングの同期を行う．2)Avarage Counter では伝送遅延時間の計算を行う．3)Buffering Module では収録・再生機器とは独立して動作をする帯域制御システムとの協調を行うことで入力部での遅延時間吸収を実現する．本研究の有用性を示すために DVTS に本機構を適用し，実装，評価を行った．本研究により，複数コンテンツフォーマットによる複数ストリームの統一的利用が可能となった．映像・音声データを対象として扱うため，放送中継，デジタルシネマ，マルチビジョンシステム，ザッピングなどのリアルタイムコンテンツ伝送にも応用可能である．

キーワード

1．実時間配信, 2．コンテンツフォーマット, 3．同期,
4．分散環境, 5. インターネット

慶應義塾大学大学院 政策・メディア研究科

久松 剛

Abstract of Master's Thesis

Academic Year 2005

# Design of synchronous reproduction system for multi streams on the Internet

In this research, absorption of the playback delay occured by environmented differences in recording, transporting, and receiving multiple synchronized contents stream via Internet is designed. Live video streaming and video relay activities using the Internet are becoming popular. Traditional real time streaming is either transporting singular streams in unicast or in multicast directions. Transporting multiple set of streams with real time relations, to designated locations is the new demand for streaming activities.

To analyze the playback delay occured by the stream transport applications, system was divided into 3 methods. 1) Input method from Sender terminal to the receiver terminal, 2) transformation method which devodes or transforms the contents format, and 3) Output method which sends receiver output to the user system whcih plabacks and/or records the streams. Each delay occured by transactions differs by the media format, recording equipment, and network environment.

To establish synchronization mechanism for multiple stream, absorption of each stream's delay consumed by transaction are summarized. Time line absorption mechanism is adapted to absorb synchronization. 3 module is developed to synchronize the multiple streams. 1) Negotiation module collaborates in terminal basis to synchronizes the playback timing by receiving the time stamp of the stream. 2) Average counter calculates the jitter within the transport delay. and 3) Buffering module absorbs the delay occured in input method by collaborating to the bandwidth control system dedicated to recording/ playback system. Mechnism is implemented into DVTS to evaluate the delay absorption.

This research realizes the multiple stream transport by using various contents format. By using video and audio format, this mechanism allows to realize real time contents streaming such as broadcast relay, digital cinema onstruction, multi-vision system and program zapping.

Keywords :
1. real-time streaing, 2. contents format, 3. synchronization,
4. distributed environment, 5. Internet

Keio University Graduate School of Media and Governance

Tsuyoshi Hisamatsu

# 目 次

**第 1 章　はじめに** **1**
1.1　背景：複数ストリームを用いたコミュニケーション形態 . . . . . 1
1.2　本研究の目的：複数ストリーミング同期システム . . . . . 2
1.3　論文の構成 . . . . . 3

**第 2 章　ストリーミング技術** **4**
2.1　収録・再生環境 . . . . . 4
2.2　メディアフォーマット . . . . . 4
2.2.1　DV . . . . . 5
2.2.2　HDV . . . . . 5
2.2.3　H.264 . . . . . 6
2.2.4　まとめ：メディアフォーマット . . . . . 7
2.3　IP ネットワーク上における映像・音声転送 . . . . . 7
2.3.1　リアルタイムストリーミング技術 . . . . . 7
2.4　同期許容時間 . . . . . 8
2.5　まとめ . . . . . 9

**第 3 章　複数ストリーム同期における課題** **10**
3.1　リアルタイムストリーミングにおける課題 . . . . . 10
3.1.1　伝送遅延時間の揺らぎ吸収技術と課題 . . . . . 10
3.2　複数ストリームの同期 . . . . . 12
3.2.1　再生遅延の発生 . . . . . 13
3.3　まとめ . . . . . 16

**第 4 章　複数ストリームの同期再生** **17**
4.1　課題へのアプローチ . . . . . 17
4.1.1　汎用性を確保した遅延時間の吸収 . . . . . 17
4.2　設計概要 . . . . . 18
4.3　Average Counter Module: 再生遅延時間揺らぎ吸収 . . . . . 19
4.4　Negotiation Module: 端末間における時間軸の共有 . . . . . 21
4.4.1　再生遅延時間差の吸収要件 . . . . . 21
4.4.2　絶対時間の共有 . . . . . 21
4.4.3　相対時間共有 . . . . . 23

4.4.4 時間軸共有機構 . . . . . 24
4.5 Buffering Module: 再生遅延時間の吸収 . . . . . 26
4.6 まとめ . . . . . 28

**第 5 章 複数ストリームにおける同期再生の実現 29**
5.1 実装概要 . . . . . 29
5.2 DVTS . . . . . 30
5.3 送信端末処理 . . . . . 30
5.4 受信端末処理 . . . . . 32
5.4.1 受信端末における実装項目 . . . . . 32
5.4.2 実装関数と処理概要 . . . . . 33
5.4.3 Average Counter Module . . . . . 35
5.4.4 Negotiation Module . . . . . 36
5.4.5 Buffering Module への接続 . . . . . 45
5.5 Buffering Module . . . . . 46
5.6 まとめ . . . . . 47

**第 6 章 複数ストリームにおける同期再生機構の評価 50**
6.1 本機構の実現した機能 . . . . . 50
6.2 評価環境 . . . . . 50
6.3 受信端末間における協調同期 . . . . . 51
6.4 送信端末間における自律分散同期 . . . . . 54
6.5 考察 . . . . . 58
6.6 まとめ . . . . . 59

**第 7 章 結論 60**
7.1 まとめ . . . . . 60
7.2 今後の課題 . . . . . 61
7.3 本機構の応用例 . . . . . 61
7.3.1 ザッピング . . . . . 61

**謝辞 64**

# 図 目 次


1.1 再生遅延構成要素 . . . . . 2

2.1 収録・再生環境 . . . . . 5
2.2 GOP 概要図 . . . . . 6
2.3 RTP フォーマット . . . . . 8

3.1 伝送遅延時間の揺らぎによるコンテンツ再生品質への影響 . . . . . 11
3.2 バッファリング . . . . . 11
3.3 バッファオーバーラン，アンダーラン . . . . . 12
3.4 再生遅延の発生要素 . . . . . 13
3.5 DV 伝送システムにおける再生処理遅延時間の計測 . . . . . 14
3.6 DV 伝送システムにおける再生処理遅延時間の計測 . . . . . 15
3.7 再生環境によって異なる再生遅延時間 . . . . . 16

4.1 DIL 層と DDL 層 . . . . . 18
4.2 設計概要 . . . . . 19
4.3 遅延時間算出フロー . . . . . 20
4.4 コンテンツ時間軸の差異と修正 . . . . . 22
4.5 NTP の持つ階層構造 . . . . . 23
4.6 The Time Synchronization Protocol for UNIX . . . . . 24
4.7 時間軸共有 . . . . . 25
4.8 STC を用いた時間軸付加 . . . . . 26
4.9 受信端末群内における最大遅延時間の共有 . . . . . 27

5.1 実装概要図 . . . . . 29
5.2 送信側におけるタイムスタンプ処理 . . . . . 31
5.3 受信側：DVTS への追加関数とその動作概要 . . . . . 34
5.4 Average Counter Module の呼び出し . . . . . 35
5.5 multi_stream_param 構造体 Average Counter Module 関係部分 . . . . . 36
5.6 Average Counter Module: 各時間の取得 . . . . . 37
5.7 Average Counter Module: 揺らぎの吸収 . . . . . 38
5.8 multi_stream_param 構造体 Negotiation Module 関係部分 . . . . . 39
5.9 Negotiation Module：パケット構造体 . . . . . 40
5.10 Negotiation Module:Master: ネゴシエーションパケット送信タイマ登録 . . 41

5.11 Negotiation Module: Master: 計算処理とネゴシエーションフラグ設定 . . . 42
5.12 Negotiation Module: Master: ネゴシエーションパケットの送信 . . . . . 43
5.13 Negotiation Module: ネゴシエーションパケットの受信 . . . . . 45
5.14 Negotiation Module: Master との時間軸差異計算 . . . . . 46
5.15 Negotiation Module: リセット要求の送信 . . . . . 47
5.16 Buffering Module の呼び出しタイマ登録 . . . . . 48
5.17 Buffering Module: Buffering Module の実行 . . . . . 49

6.1 処理遅延時間の計測 . . . . . 52
6.2 評価環境：受信側端末間協調同期 . . . . . 52
6.3 実験 1:DVTS 送受信 (伝送遅延時間無し) . . . . . 53
6.4 実験 2,3:DVTS 送受信 伝送遅延時間 100ms(左)，伝送遅延時間 200ms(右) . 53
6.5 実験 4:DVTS 送受信 (伝送遅延時間無し) . . . . . 54
6.6 実験 5:Buffering Module 適用前 (左)，適用後 (右)(遅延時間 100ms) . . . . . 54
6.7 実験 6:Buffering Module 適用前 (左)，適用後 (右)(遅延時間 200ms) . . . . . 55
6.8 評価環境：送信側自立分散同期 . . . . . 55
6.9 実験 7:複数送信機による DVTS 送受信 . . . . . 56
6.10 実験 8:複数送信機による DVTS 送受信 (伝送遅延時間 0ms) . . . . . 56
6.11 実験 9,10:複数送信機による DVTS 送受信 伝送遅延時間 100ms(左)，200ms(右) 57
6.12 実験 11:Buffering Module 適用前 (左)，適用後 (右)(遅延時間 0ms) . . . . . 57
6.13 実験 12:Buffering Module 適用前 (左)，適用後 (右)(遅延時間 100ms) . . . . 58
6.14 実験 13:本機構適用前 (左)，適用後 (右)(遅延時間 200ms) . . . . . 58

7.1 単一のフォーマットを用いた複数ストリーム転送例 (右) と複数のフォーマットを用いた複数ストリーム転送例 (左) . . . . . 62
7.2 ザッピング応用例 . . . . . 63

# 表 目 次

5.1 実装環境 . . . . . 30
5.2 実装コマンドオプション一覧 (受信側) . . . . . 33

6.1 評価に用いた計算機 . . . . . 51
6.2 参照した NTP サーバ . . . . . 51

# 第1章　はじめに

## 1.1　背景：複数ストリームを用いたコミュニケーション形態

ネットワークの広帯域化，FTTH や ADSL の普及に伴い，インターネットを用いた映像・音声配信サービスが一般的になり，BB!TV[1] に代表されるインターネット放送サービスが利用可能になった．これらのサービスは単一のストリームを用いた 1 対 1 通信，あるいは 1 対多通信によって構成される．今後のストリーミングサービスとして，複数のストリームが相互に関係するような形での多対 1，多対多通信を用いたコミュニケーション形態へとニーズは変化して行く．複数のストリームを相互利用したコミュニケーション形態として応用できる例として，放送中継と映画制作を挙げることができる．

JGN 札幌雪祭り中継 [2] に見られるような放送中継システムでは，複数に散らばった拠点からコンテンツを一カ所に集約し，編集する必要がある．デジタルシネマプロジェクト [3] に見られる映画制作システムでは，複数箇所に居るスタッフが協調できる環境が必要である．このようなコンテンツ収録環境では，多対 1，もしくは多対多通信を用いる必要がある．

複数ストリームを利用したサービスで問題になるのは，各ストリームが時間軸での関連を持つ場合，それぞれのストリームの同期を考慮する必要があることである．従来型の独立した単一ストリームでは，該当するストリーム以外は排他的に扱われる．しかし多対 1, 多対多ストリーミングでは，複数のストリームが関連を持つことで一つのサービスを形成する為，全てのストリームに対して統一的制御を行う必要がある．本研究ではこのように受信者が，1 つのコンテンツとして受信・再生する為に選択した複数のストリームをグループと呼ぶ．

本研究では，コンテンツの入力から再生機器によって再生されるまでの処理時間を，再生遅延時間と呼ぶ．再生遅延を構成する要素を図 1.1 に示す．複数のストリームを扱う場合，それぞれのストリームが伝送される途中経路の違いや，メディアフォーマットの違い，収録・再生機器の組み合わせといった遅延を発生させる要素が複合し，再生時のコンテンツ時間軸の差となる．本研究ではコンテンツフォーマットを含めた環境の組み合わせの違いを，収録・再生環境の差異と呼ぶ．収録・再生環境の差異により再生遅延時間の差が発生し，複数のストリームの協調が困難となる．

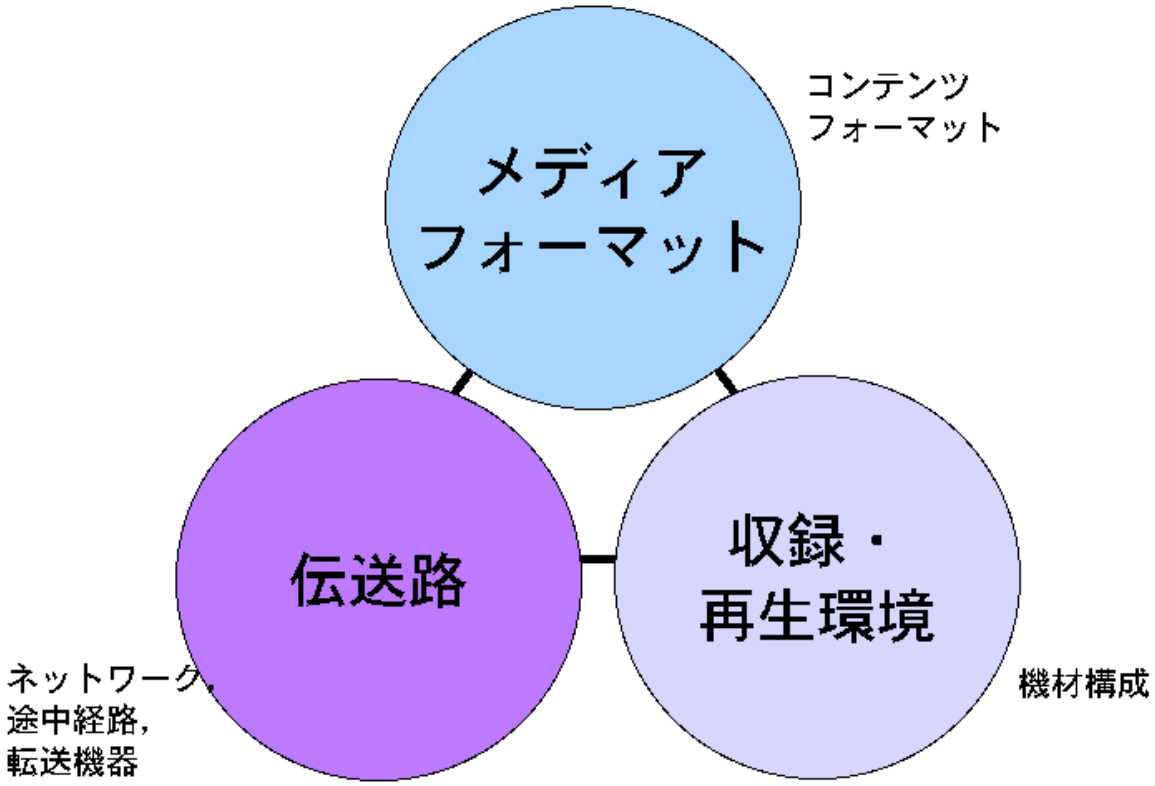


図 1.1: 再生遅延構成要素

## 1.2 本研究の目的：複数ストリーミング同期システム

本研究ではストリーミングシステムの構成要素の処理によって生じる再生遅延時間に着目し，汎用性の高い同期機構の構築を行う．ストリーミングシステムにおける処理時間の発生要因に対し，以下の 3 つのアプローチを行う．

1. 再生遅延発生部の分類
   収録・再生環境の組み合わせによって異なる再生遅延時間に対し，遅延時間発生箇所の分類を行い，多様化が進む収録・再生環境を汎用化し，再生遅延時間差の吸収に対する要求事項の整理を行う．

2. 収録・再生環境に対する汎用的な遅延時間吸収機構の構築
   再生遅延時間を吸収するためには，それぞれの環境に合わせた遅延時間の取得と吸収をしなければならず，既存のストリーミングシステムに対する変更が発生する．汎用性を確保する為には，再生遅延時間吸収のための既存機構の変更は最小限にしなければならない．

3. 様々な遅延発生要素に対する統一的な指標の導入
   収録・再生環境の組み合わせは多様であり，それぞれの遅延時間発生箇所から取得できる情報も様々である．汎用的な再生遅延時間吸収機構を実現する為には，コンテンツの収録端末への入力から，ユーザに対する出力に至るまでに共通した単位による再生遅延時間の計算をする必要がある．

## 1.3 論文の構成

本論文は 7 章から構成される．第 2 章では，本研究における要素技術として複数ストリームの転送を行うための要素技術に関して述べる．第 3 章では，複数ストリーム同期実現に関する同期手法の解説と本機構による応用例について述べる．インターネット上の異なる地点に存在する複数機器間の時間軸の共有，コンテンツに対する時間軸付加手法，複数ストリーム同期についての既存技術，時間軸差異吸収についての既存技術について述べ，それぞれの問題点を挙げた後，オンラインスタジオ実現のための必要事項の整理を行う．第 4 章においてその問題を解決する事のできる，本システムの設計について述べる．第 5 章で設計に基づいた実装に関して述べる．第 6 章で本システムの実装に対する評価を行い，既存の問題を解決したか否かを述べる．第 7 章で本研究のまとめ，及び今後の展望を述べる．

# 第2章　ストリーミング技術

本章では，複数ストリーム同期機構を構築するために前提となる技術について述べる．まず，現在のコンテンツ収録・再生環境の多様性について述べる．特に収録・再生環境の構成要素の中で，コンテンツフォーマットとその伝送技術に着目し．これを説明する．次に，複数ストリームの同期許容時間について述べる．

## 2.1　収録・再生環境

収録・再生環境の多様化が進んでいる．収録・再生環境の概要図を図 2.1 に示す．かつての収録・再生環境におけるコンテンツの流れは，放送を受信・録画する，ビデオに記録されたデータを再生する，といった限定的なものであった．しかし現在では，HDD に保存したものをネットワークを用いて再送する，ネットワークから取得したものをトランスコーディングした上でディスクに保存する，といったように，様々なコンテンツの流れが存在する．収録・再生環境は大きく以下の 3 つに分ける事ができる．

1. 入力処理
   収録機器からの入力や，ネットワーク伝送などの，コンテンツ入力の為の処理が含まれる．

2. 内部処理
   出力先に適した形に変更する為の処理が含まれる．トランスコーディングなどの他にディスクから読み取ったデータをネットワーク伝送する為に IP ヘッダを付加するなどのステップも含まれる．

3. 出力処理
   再生機器への出力や，HDD への書き出し，ネットワーク伝送といったコンテンツ出力の為の処理が含まれる．

このように収録・再生環境によってコンテンツが処理される手法やステップ数は多用である．今後もこうした収録・再生環境の多様性は広がりを続けるものと考えられる．

## 2.2　メディアフォーマット

本節では現在インターネット上で多く用いられている映像フォーマットについて述べ，その特徴を示す．

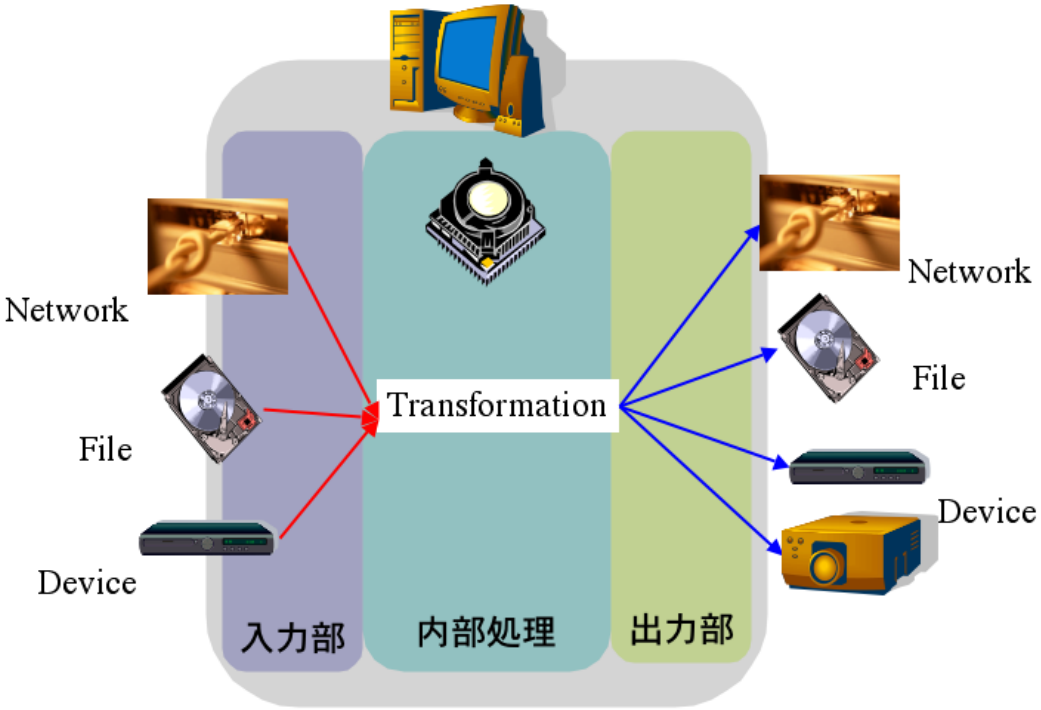


図 2.1: 収録・再生環境

### 2.2.1 DV

DV(Digital Video)[4][5] は，1999 年に HD デジタル VCR 協議会により規格化されたフレーム内圧縮方式を用いたフォーマットである．

画質はフルフレーム時で $720 \times 480 \times 19.97fps$ であり，圧縮率は一定である．DV を構成する最小単位は 80 バイトの DIF Block であり，150 DIF Block から構成される 1 DIF Sequesnce という単位がある．1 フレームは 10 DIF Sequence から構成される．

### 2.2.2 HDV

HDV(High Definition Video)[6] は 2003 年に Canon, Sharp, Sony, Victor により規格化されたフォーマットである．MPEG2[7] 形式で圧縮されたデータを DV, mini-DV テープに保存することができるようにしたものである．

MPEG2 フォーマットは，フレーム間圧縮方式を用いたフォーマットである．MPEG[8] におけるフレーム間圧縮の概要を図 2.2 に示す．MPEG では GOP(Group Of Picture) と呼ばれる圧縮・再生・編集の単位となる数フレームから構成される．GOP で扱われるフレームは I ピクチャ(Intra Picture)，P ピクチャ(Predictive Picture)，B ピクチャ(Bidirectionally Picture) から構成される．それぞれの概要を以下に示す．

- I ピクチャ
  基準となるピクチャであり，単体で画像を復元できる．

- P ピクチャ
  フレーム間順方向予測符号化画像とも呼ばれる．時間軸的に過去の I ピクチャ，P ピクチャからの相関情報で表現される．

- B ピクチャ
  双方向予測符号化画像と呼ばれる．時間軸的に前後する画面からの相関情報で表現される．

MPEG2 のビットレートは 2Mbps から 60Mbps 程度まで幅広く存在する．DV と同じ SD の画質を再現するために必要なビットレートは，4Mbps から 8Mbps 程度である．また，VBR(Variable Bit Rate) の適用も可能である．

このように MPEG フォーマットは前述した DV フォーマットに比べ，少ないビットレートで多くの画像情報を保存する事ができる．しかし前述した DV に比べてフレーム間の関係が複雑なものであるため，エンコード・デコードに掛かる処理は増加する．

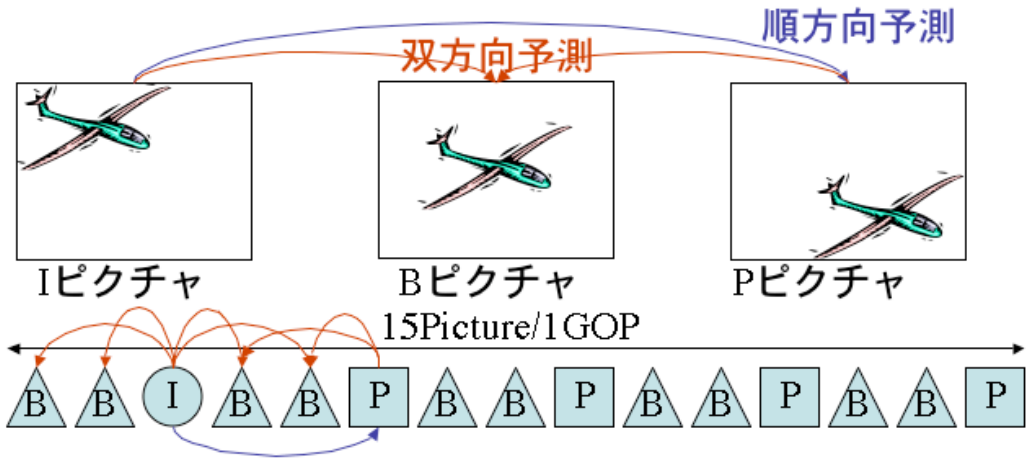


図 2.2: GOP 概要図

### 2.2.3 H.264

H.264[9] は ITU-T と ISO/IEC が共同で策定した国際標準規格であり，MPEG-4 Part 10 AVC とも呼ばれる．従来の MPEG4[10] に比べ，30%から最大 2 倍の圧縮率を実現する．DV と同じ SD の画質を再現するために必要なビットレートは 1Mbps 以下である．H.264 に含まれる主な技術を以下に述べる．

- 可変マクロブロックサイズ
  MPEG では，画面をマクロブロックという単位で分割する．MPEG2 では 16x16 ピクセル，MPEG4 では 8x8 ピクセルであるが，H.264 では 4x4〜16x16 ピクセルまでの様々なサイズを選択できる．エンコーダは映像の内容に合わせてマクロブロックの大きさを変更する．

- 遠隔参照フレーム
  MPEG2，MPEG4 では前後のフレームの差分のみを取っていたが，H.264 では最大 5 フレーム前のフレームの参照を行う．

- フレーム内予測
  I フレームに対し，類似するマクロブロックを探し，その差分を取って圧縮する．

- ループフィルタ
  従来の MPEG では，デコード後の画像に対し，フィルタ演算をすることでブロックノイズの削減を行なうことができた．H.264 では，エンコード時にフィルタ演算を踏まえたエンコードを行う．これにより，特に低ビットレート時のブロックノイズ減少に効果がある．

このように H.264 にはエンコード技術が数多く含まれている．そのため，MPEG2 以上にエンコード・デコード時間が発生する．

### 2.2.4 まとめ：メディアフォーマット

本節では現在ストリーミングアプリケーションで採用されている主流の映像フォーマットについて述べた．これまで述べて来たように同等の品質を再現する場合でも，そのビットレートやエンコード・デコードのための処理はフォーマットによって異なる．

## 2.3 IP ネットワーク上における映像・音声転送

本節では IP ネットワーク上における映像・音声転送に用いられるストリーミング技術について述べる．まず始めに，ストリーミングアプリケーションにおける代表的なトランスポートプロトコルである RTP について述べる．次に，リアルタイムストリーミングを実現する際の課題として，特に伝送遅延時間の発生と揺らぎについて述べる．その上で伝送遅延時間揺らぎ解消技術としてバッファリングについて述べる．

### 2.3.1 リアルタイムストリーミング技術

映像・音声データ全体を受信してからローカルで再生する方式に対し，データの受信と再生を並行して行う方法がストリーミング再生である．リアルタイム性の高い映像・音声転送だけでなく，Real System[11] に代表される蓄積型映像・音声配信にも用いられる．

実時間通信を行う映像・音声転送では 2 つの特徴により，TCP(Transmission Control Protocol)[12] ではなく，UDP(User Datagram Protocol)[13] を用いる．

- TCP による再送制御が発生するとパケットの到着時間に遅れが生じる．実時間再生を行う受信側に対し，TCP による再送制御を行った場合，再送時間に対するパケットの到達時間が間に合わず，映像や音声が乱れる可能性がある．UDP を用いた通信は，TCP を用いた通信と異なり，トランスポートレイヤにおけるパケットの到達順序が異なる可能性がある．パケット到達順序の判別を行うにはアプリケーションでの対応が必要となる．到達順序の判別を行うための代表的な手法として RTP[14] がある．

- TCP による輻輳制御が発生するとパケットの到着時間に影響が生じる．輻輳制御が行われた場合，受信側で実時間に基づいた再生に対して間に合わず，映像や音声が乱れる可能性がある．そのため UDP を用いた通信では，RTP と RTCP(Real Time Transport Control Protocol)[14] を併用することでパケットロスを検知し，映像レート，フレーム数などを変化させることにより，転送量の調整を行うなどの対応がなされている．

RTP は end-to-end のネットワークにおいて映像・音声といった実時間性が必要とされるデータ転送に利用することを目的に，トランスポートレイヤとは独立して設計されているプロトコルである．RTP フォーマットを図 2.3 に示す．送信時に RTP によって Sequence Number の付加による送出パケットの順番を示し，Time Stamp に送出時の時間データを組み込むことで時間軸を示す．これにより受信側においてパケットの順序確認を行う．

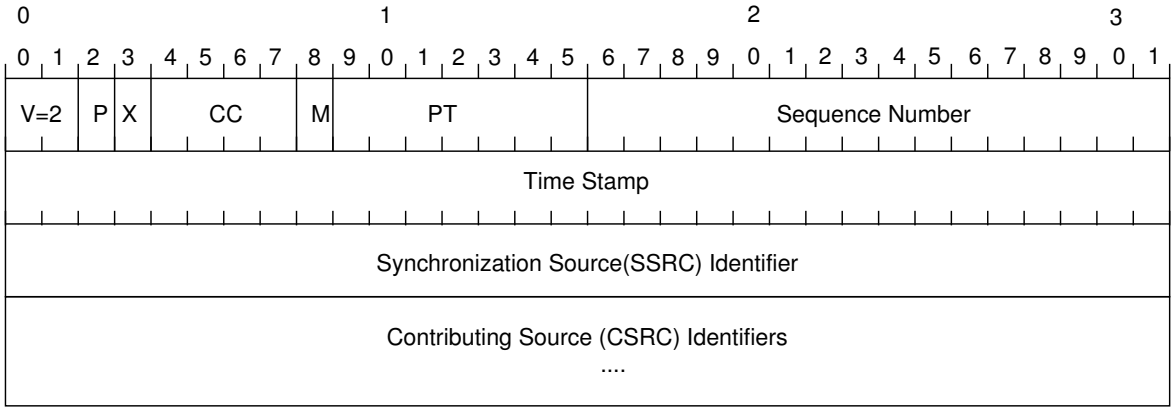


図 2.3: RTP フォーマット

## 2.4 同期許容時間

本節では複数ストリームの再生遅延を吸収し，同期再生を行う際の許容時間について述べる．本研究では，再生遅延吸収時の許容時間を同期許容時間と呼ぶ．

本研究では複数ストリーム同期許容時間の指標として，現在利用されているデジタルミキサで発生する遅延を基に設定した．デジタルミキサ内部には通常，フレームシンクロナイザを内蔵している．機種によって差はあるがNTSC の場合，1-2 フレーム分のビデオ入力バッファ[15] を備えている．このことより，各ストリームをミキサに対して出力した際に，それぞれのフレームが 30-60ms 以内の差でビデオ入力バッファに入力できた場合，デジタルミキサ内の処理によって同期できる．

本研究では，デジタルミキサのビデオ入力バッファの値を再生遅延時間吸収機構の同期精度の目標値とする．

## 2.5 まとめ

本章では，複数ストリーム同期機構を構築するために前提となる技術について述べた．まず始めに収録・再生環境の多様性について述べた．収録・再生環境の構成要素として，特にメディアフォーマットについてその多様性について述べた．次に伝送路としてインターネットを用いたリアルタイムストリーミング技術に付いて述べた．その上で，複数ストリーム同期を行う際の同期許容時間について 30 - 60ms を同期制度目標として設定した．次節では，まずリアルタイムストリーミングを実現する上での課題について述べる．その上で本章で述べた収録・再生環境の多様性を踏まえた複数ストリーム同期を実現するための課題について述べる．

# 第3章　複数ストリーム同期における課題

本章では，複数ストリームの同期機構の実現に必要な既存技術について述べる．まず始めにリアルタイムストリーミングにおける課題について述べる．次に複数ストリームの場合の問題点として同期について述べる．

## 3.1　リアルタイムストリーミングにおける課題

インターネットを媒体とした転送では，途中経路における伝送メディアの距離，ネットワークの帯域によって伝送遅延時間が生じる．また，パケット交換方式であるインターネットでは，他のトラフィックの増加や，途中経路に存在するルータのキュー溢れなどの要因によって伝送遅延時間の揺らぎ [16] やパケットロス [17] が発生する．

伝送遅延時間の揺らぎによって起きるコンテンツ再生品質への影響を図 3.1 に示す．図 3.1 内 A では伝送遅延時間の揺らぎが発生していないため，一定の伝送遅延時間で受信端末にパケットが伝送されている．そのため，受信端末では映像・音声の乱れが生じることなく再生ができる．しかし図 3.1 内 B の時点でパケットの到達に遅れが生じると，映像データの到達が間に合わず，映像が乱れるなどの悪影響が生じる．

ネットワーク伝送遅延時間の揺らぎに対する手法として，バッファリングがある．次節ではバッファリングについて述べる．

### 3.1.1　伝送遅延時間の揺らぎ吸収技術と課題

ストリーミング再生では，映像データは逐次消費されるため，映像データは必要とする時間 (デッドライン) までに受信端末に到着する必要がある．しかし伝送遅延時間の揺らぎが発生することで，パケットの伝送時間に開きが生じた場合，映像データはデッドラインまでに受信端末に到着することができない．このようにデッドラインまでに受信端末に到着することができなかった映像データは再生に利用されず，破棄される．そのため受信した映像・音声の乱れが生じる．

再生遅延を吸収するために，受信したデータを一時的にバッファに蓄積し，蓄積されたデータから再生を行うバッファリングと呼ばれる方法がある．バッファリングの概要を図 3.2 に示す．予め一定量の映像データをバッファリングし，逐次消費することでパケットの遅延や揺らぎ，順序の不整合を軽減することができる．

バッファサイズと，実ネットワークの動きの差によって生じる問題として，バッファオーバーラン，アンダーランの問題がある．バッファオーバーランとアンダーランの関係を図

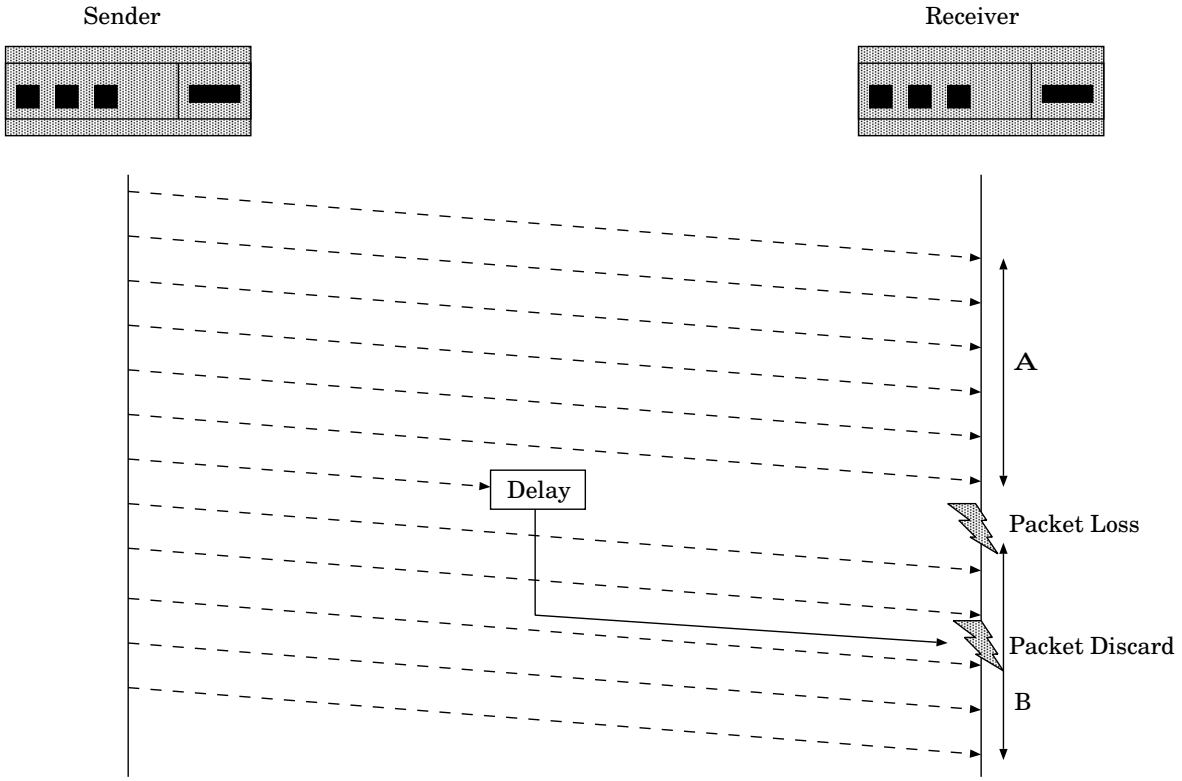


図 3.1: 伝送遅延時間の揺らぎによるコンテンツ再生品質への影響

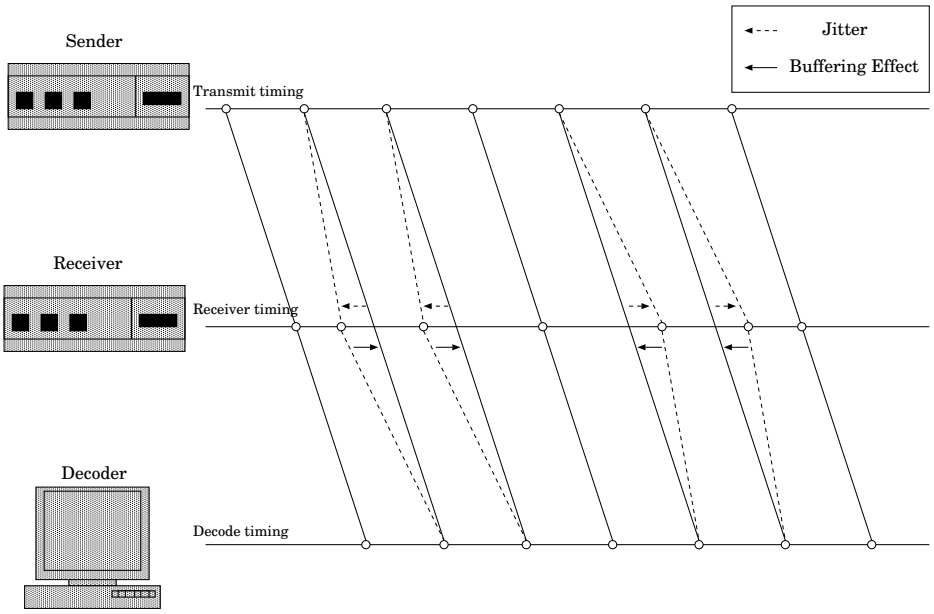


図 3.2: バッファリング

3.3 に示す．アプリケーションで用意されたバッファの下限を O(t) とし，送信側からのストリーム転送量を P(t) とする．t のタイミングでパケットロスなどの問題により受信側へのデータ到着が遅れる場合，t' の時点でバッファオーバーランが発生し，再生する映像に影響が発生する．また，用意したバッファサイズが極端に小さかった場合，バッファの上限値である buffer+O(t) とストリーム転送量 P(t) が接触することによりバッファーオーバーランが発生することも考えられる．これらのことにより，バッファサイズ P(t) の理想値は以下の数式で表される．

$$O(t) < P(t) < buffer + O(t)$$

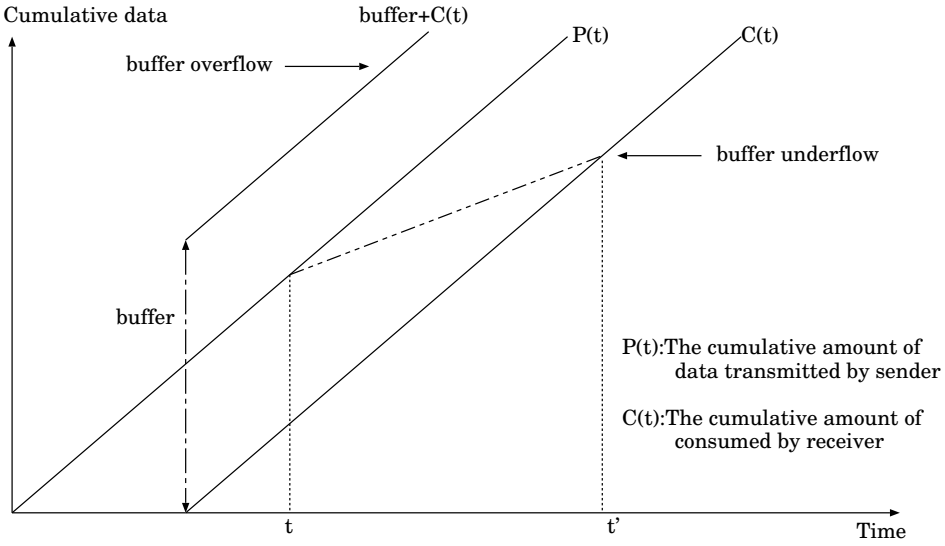


図 3.3: バッファオーバーラン，アンダーラン

リアルタイムストリーミングでは，コンテンツ時間軸の実時間性確保を行わなければならないため，ストリーミング開始前にバッファサイズの確定を行わなければならない．未知である未来の時間軸で発生する伝送遅延時間の揺らぎを吸収することを目的に，バッファサイズを余分に大きく確保する手法がある．しかし過大なバッファサイズの確保はデータ再生が遅れに繋がるため，リアルタイム性が損なわれる．また，実ネットワーク上における伝送遅延時間の揺らぎやパケットロスは偶発的なものが多い．[17]．そのため，過大なバッファを確保した場合であっても，過剰なバッファが有効利用される時間は短い．従って，バッファ量の算出を行う際に伝送遅延時間の揺らぎに対して閾値を設け，ネットワークの変化を偶発的なものと慢性的なものに切り分けることでリアルタイム性と伝送遅延時間の揺らぎ吸収を両立したストリーミングが実現できる．

## 3.2 複数ストリームの同期

第 3.1 節では，単一のストリームに対する伝送遅延時間の処理について述べた．本節では，複数ストリームを行う際の問題点について述べる．特に複数ストリームの再生時の同

期という観点から，再生遅延時間の原因となる要素の分類を行う．

### 3.2.1 再生遅延の発生

第 2.1 節において，収録・再生環境の多様性について述べた．収録・再生環境の処理によって発生する再生遅延時間について，その発生箇所を分類したものを図 3.4 に示す．本研究では，再生遅延時間の発生要素を以下の 3 つに分類する．

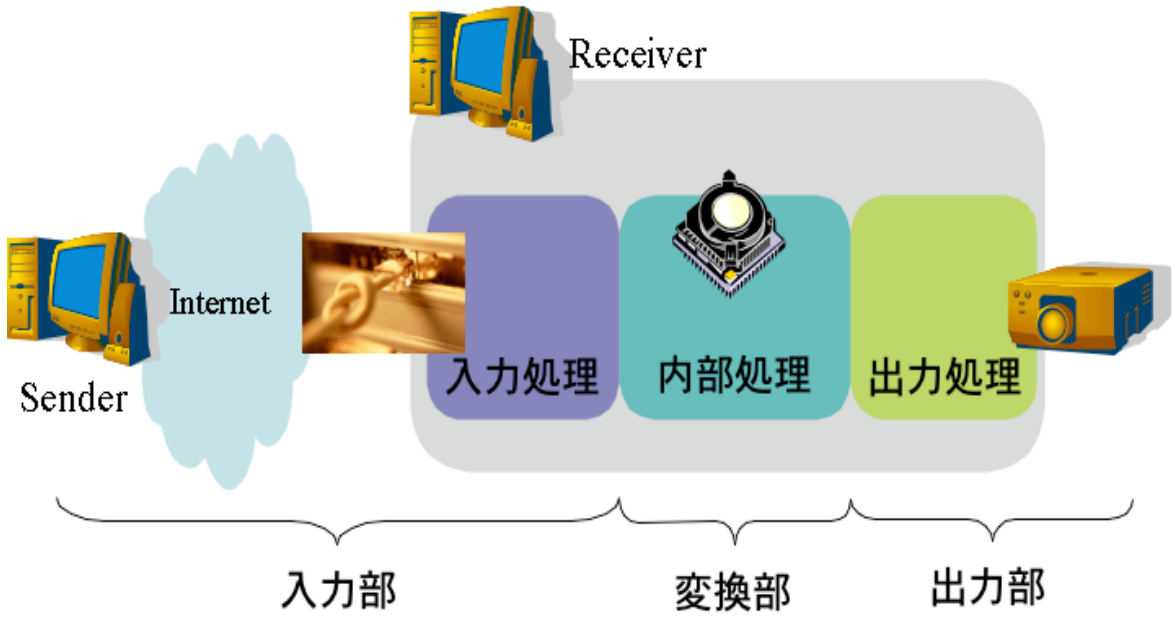


図 3.4: 再生遅延の発生要素

- 入力部
  内部処理が行われるまでの時間である．ネットワーク伝送遅延だけでなく，送信端末処理時間も含まれている．
- 変換部
  受信端末内で発生する処理時間である．ストリームのデコードや，出力部に接続された機器に対するフォーマット変更の時間が含まれている．
- 出力部
  コンテンツの出力までの時間である．受信端末に接続された収録端末内処理も含まれている．

収録・再生環境の差異と再生遅延時間の関係例として，IP を用いた DV 伝送を行う DVTS(Digital Video Transport System)[18][19] と，SONY ATM-IEEE1394 Link Unit

SEU-TL100[20] の比較実験を挙げる [21]．DV データを DIF ブロック単位に分割して IP ヘッダを付加する DVTS に対し，SEU-TL100 は DV データを ATM セルとして直接 ATM 網に送信を行う．

計測環境を図 3.6 に示す．実験では，フレームメモリを用いて白画面及び黒画面を 1 秒ずつ交互に表示し，DV/NTSC コンバータ及び DV 伝送システムによるエンコード・デコード処理により遅延を測定している．

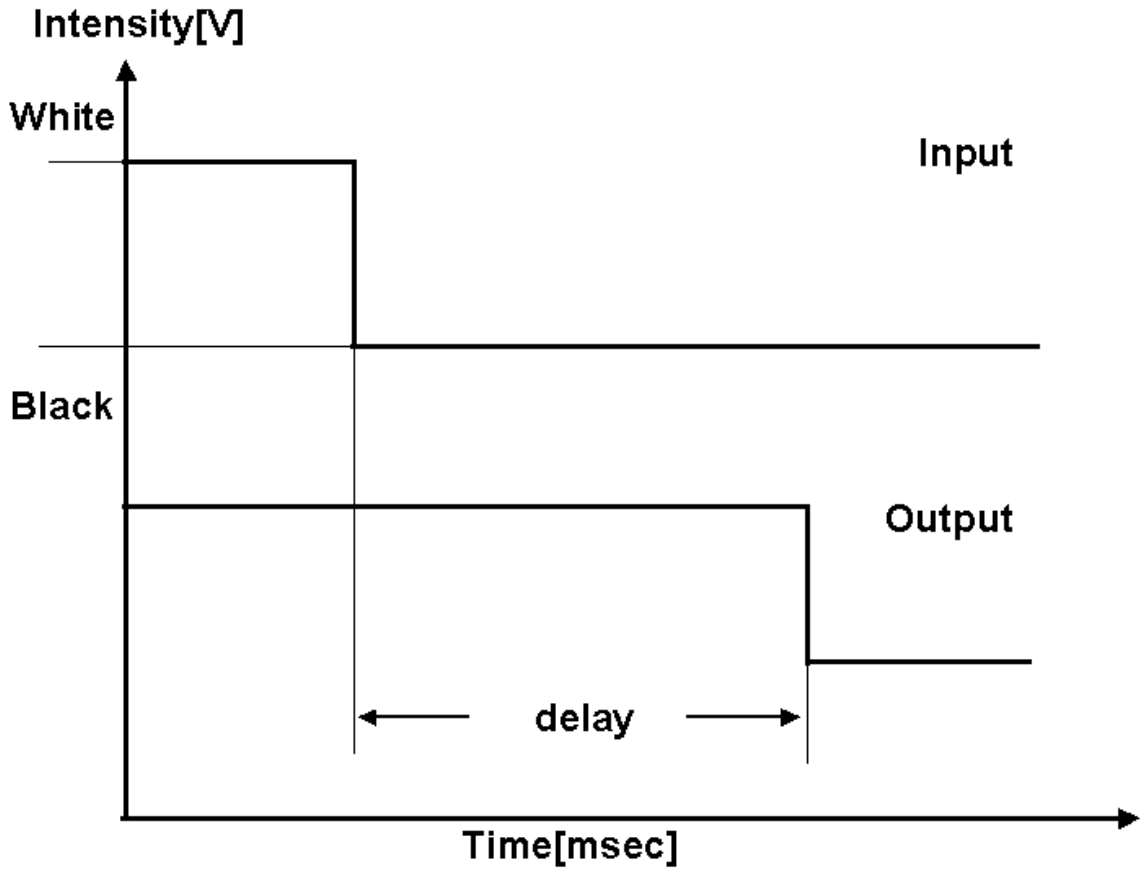


図 3.5: DV 伝送システムにおける再生処理遅延時間の計測

次に測定されたデータについて述べる．まず DV/NTSC コンバータのみを IEEE1394 ケーブルを用いて直結した場合，

$$d_{conv} + d_{conv'} = 70.0msec$$

が得られた．

次に DVTS と SEU-TL100 による処理遅延を計測した．この際，Fast Ethernet Swintching HUB を介しているが，処理遅延時間は 1msec 未満と小さいため，省略されて扱われている．計測の結果，

DVTS:

$$d_{conv} + d_{dvts} + d_{dvts'} + d_{conv'} = 222.7msec$$

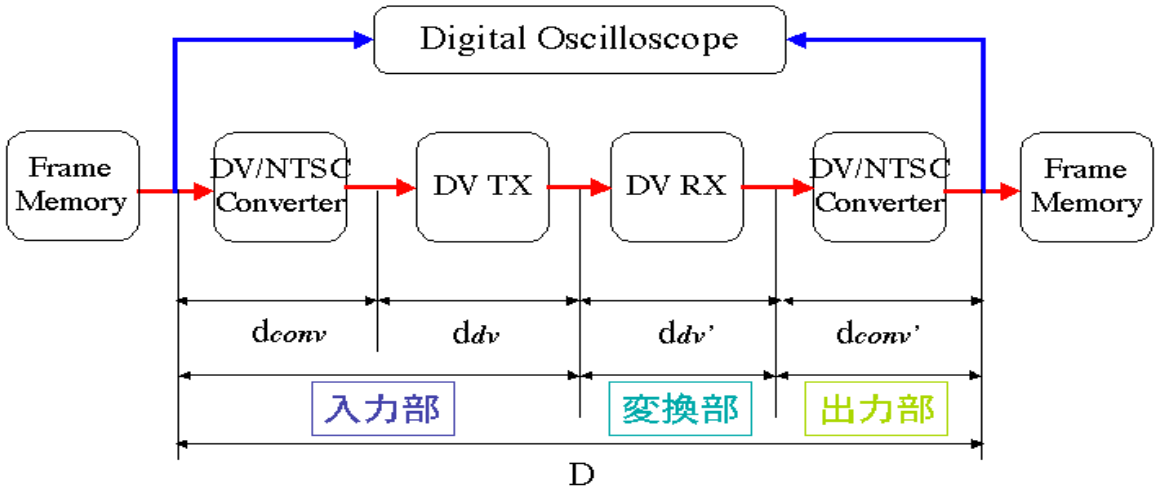


図 3.6: DV 伝送システムにおける再生処理遅延時間の計測

SEU-TL100:

$$d_{conv} + d_{tl100} + d_{tl100'} + d_{conv'} = 91.6msec$$

が得られた．これより DV/NTSC コンバータによる遅延を差し引くと
DVTS:

$$d_{dvts} + d_{dvts'} = 152.7msec$$

SEU-TL100:

$$d_{tl100} + d_{tl100'} = 21.6msec$$

となる．

この実験では，Fast Ethernet と ATM 網の特性の異なる伝送路を利用する異なるシステムを用いた計測を行った．その結果，同じコンテンツフォーマットを用いるシステムであっても，その処理時間は異なる．

入力部，変換部，出力部の 3 つの再生遅延時間発生要素を，異なる 3 つの収録・再生環境に適用したものを図 3.7 に示す．入力部に区分される遅延時間発生要素には，収録機器構成，エンコーディングと言った処理系に加え，ネットワーク環境がある．変換部に区分される遅延時間発生要素には，受信・再生処理の他，出力形式の違いや収録・再生機器の違いに寄るトランスコード処理がある．出力部に区分される遅延時間発生要素には，収録機器の構成がある．再生遅延時間 (d) は入力部処理時間 (i)，変換部処理時間 (t)，出力部処理時間 (o) より以下の式で表現することができる．

$$d = i + t + o$$

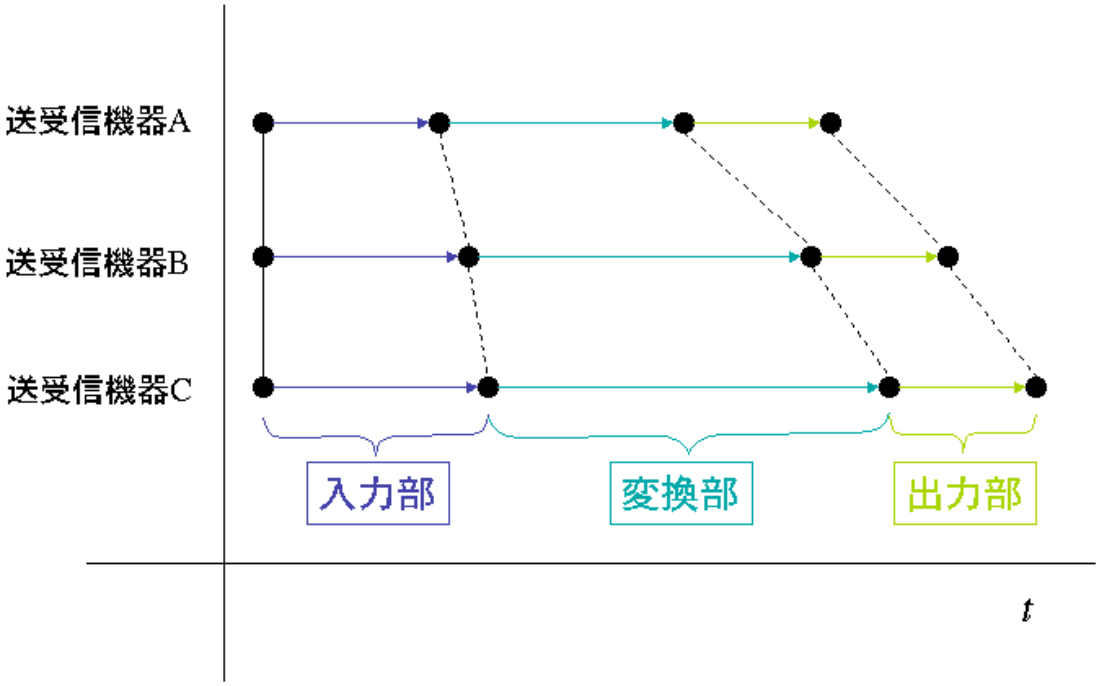


図 3.7: 再生環境によって異なる再生遅延時間

## 3.3 まとめ

本章では複数ストリーム同期にに関係する既存技術について述べ，検証を行った．
次章ではこれまで述べてきた事柄を元に，本モデルの設計について述べる．

# 第4章　複数ストリームの同期再生

第3章において複数ストリーム同期実現のための課題を述べた．本章ではまず，本研究でのアプローチについて述べる．その上で複数ストリームの再生における遅延時間の差を吸収する機構の設計について述べる．

## 4.1　課題へのアプローチ

第3章ではリアルタイム複数ストリームの同期を実現する為の課題について述べた．本節では課題をまとめ，本研究でのアプローチを述べる．

### 4.1.1　汎用性を確保した遅延時間の吸収

第2章では，収録・再生環境の組み合わせの多様性について述べた．収録・再生環境への依存を最小にし，汎用的に扱うことのできる機構が必要である．

第3.2節では，収録・再生環境の組み合わせの違いによる再生遅延時間の差について述べた．再生遅延時間を算出する為には，収録・再生環境に対し統一的な指標を設ける必要がある．

本研究では，遅延時間吸収機構として収録・再生環境に対する依存性が高い機能をDDL(Device Dependent Layer)，低い機能をDIL(Device Independent Layer)として分類し，機能分担を行う．また，本研究では異なる収録・再生環境に対し，時間軸を指標とした統一的処理を実現し，個々の再生遅延時間の要素に対し管理・吸収を行う．DILとDDLの概念図4.1に示す．それぞれの概要を以下に述べる．

- DIL
  コンテンツフォーマット，アプリケーション，OS，収録機器など，収録・再生環境に依存しない機能を指す．バッファリング機構がDILとして含まれる．

- DDL
  コンテンツフォーマットやアプリケーションへの依存が避けられない機能を指す．データ受信時間や処理時間の計測などが含まれる．

- OS層
  クライアント接続と認証の管理，及びメモリ割当・解放ルーチンの提供を行う．ホストOSに依存する機能も含む．

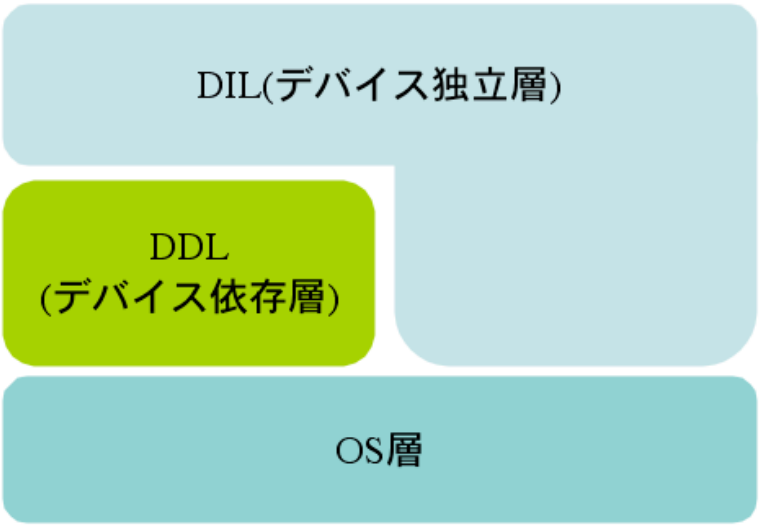


図 4.1: DIL 層と DDL 層

以下にリアルタイムでの複数ストリームの同期を実現する為の課題をまとめ，それぞれに対する解決方法を述べる．

- 伝送遅延時間の揺らぎ
  第 3.1 節では伝送遅延時間の揺らぎの発生と，それに伴うバッファ量決定の難しさについて述べた．本研究では，遅延時間の揺らぎに対して閾値を設け，偶発的なものと慢性的なものの 2 つに分類することでネットワークの変化を定義する．慢性的な遅延時間の揺らぎに対してのみに対応することで適切なバッファ量算出を行う．

- ストリームによって異なる再生遅延時間
  第 3.2 節では，収録・再生環境の違いに寄って再生遅延時間に差があることを述べた．異なる再生時間を同期させる為に，端末間ネゴシエーションを行う．

- 収録・再生環境の多様性
  収録・再生環境の多様性に対し，汎用的なバッファリング機構による同期が必要となる．本研究ではバッファリング機構を独立した計算機として持つ．

## 4.2 設計概要

第 4.1 節を受け，本研究の設計概要を図 4.2 に示す．本機構は以下の 3 つのモジュールより構成される．

- Average Counter Module(DDL)
  伝送遅延時間の算出を行う．パケット受信時刻を取得など，アプリケーションへの依存性が高い．ため，DDL に分類する．

- Negotiation Module(DDL)
  Average Counter Module から取得した伝送遅延時間と，他の受信端末の持つ伝送遅延時間との比較を行う．これにより，受信端末群内で最も遅いストリームを基準としたバッファ量算出を行う．アプリケーションレイヤでの動作を行うため，受信端末の下位レイヤに依存することから DDL に分類する．

- Buffering Module(DIL))
  Negotiation Module によって算出されたバッファ量を元にしたバッファリングを行う．独立した計算機として存在するため，DIL に分類する．

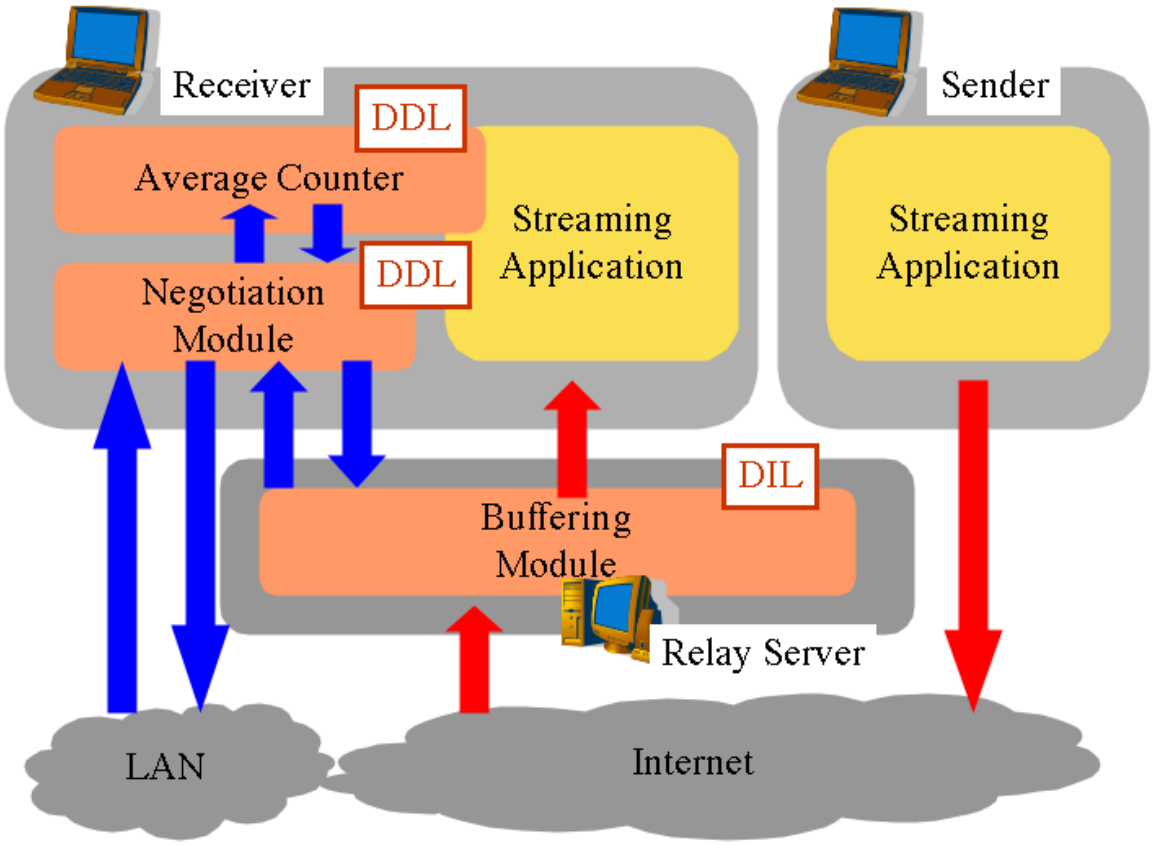


図 4.2: 設計概要

以下に各モジュールについて詳細を述べる．

## 4.3 Average Counter Module: 再生遅延時間揺らぎ吸収

第 2.3 では，インターネット上での伝送遅延時間の揺らぎには偶発的なものと慢性的なものがあることを述べた．その上でリアルタイム性と揺らぎ吸収の両立を行う為には，伝

送遅延時間の揺らぎに対して閾値を設け，偶発的な揺らぎは考慮せず，慢性的な揺らぎのみを取り出し，バッファ量計算を行う必要があることを述べた．

本研究では，伝送遅延時間の揺らぎに対し，偶発的であるものと慢性的であるものの閾値として，平均遅延時間に対する揺らぎ率 (%) と，揺らぎ回数を挙げる．

本機構におけるバッファ量算出フローを図 4.3 に示す．平均遅延時間 $a$(ms) に対し，設定された許容揺らぎ率より，許容揺らぎ幅 $m$(ms) が存在する場合，任意のパケット到達遅延時間の許容揺らぎ時間幅 $c$(ms) は以下の式で表すことができる．伝送遅延時間の揺らぎが許容範囲内であると判断された場合，平均遅延時間の更新は発生しない．

$$a - m > c > a + m$$

伝送遅延時間の許容範囲超過回数が $n$ 回を超過した場合，ネットワーク伝送遅延時間の揺らぎは慢性的なものであり，バッファ量を変更する必要があると判断し，平均遅延時間の更新を行う．

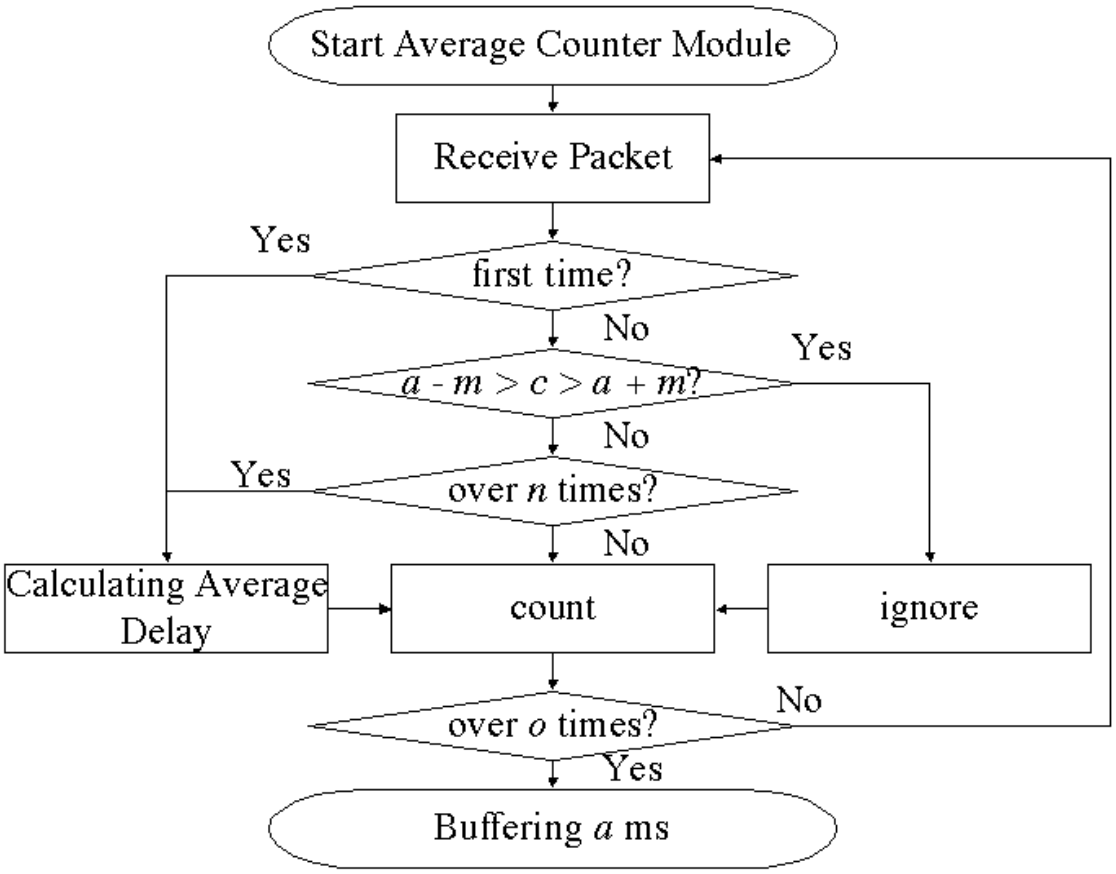


図 4.3: 遅延時間算出フロー

## 4.4 Negotiation Module: 端末間における時間軸の共有

複数ストリーム同期を実現するためには，それぞれのストリーム再生時に発生する再生遅延時間を求め，適切なバッファ量を設定しなければならない．本節ではまず，再生遅延時間算出のための要件について述べた後，算出に必要な端末間における時間軸の共有手法について述べる．

### 4.4.1 再生遅延時間差の吸収要件

再生遅延時間の差を構成する要因を図 4.4 に示す．再生遅延時間の差を吸収するための要件は以下の 2 つである．

1. コンテンツ時間軸の一致
   再生遅延時間の取得を行う必要がある．再生遅延時間 $p$(ms) は受信時刻 $r$(ms)，送信時時刻 $s$(ms)，収録・再生環境内で発生する処理時間 $d$(ms) より，以下の式で表される．

   $$d = (r - s) + e$$

   上記の式による計算を行うためには，送信時刻と受信時刻を一致させる必要がある．(図 4.4 内左上，左下)

2. 再生タイミングの一致
   再生遅延時間 $d$ はそれぞれの送受信機器が存在する収録・再生環境によって異なる．(図 4.4 内右上) 全ての受信機で再生遅延時間を一致させるためには，受信機間でネゴシエーションにより，最も再生遅延時間の大きい端末と同期を取る必要がある．(図 4.4 内右下)

次項では，まず時間軸同期に関する技術について述べた後，本機構での具体的な動作概要を述べる．

### 4.4.2 絶対時間の共有

UTC(Coordinated Universal Time) を基準とした時刻配布システムとして，衛星を伝送媒体とする GPS と，インターネットを伝送媒体とする NTP について述べる．

#### GPS

GPS(Global Positioning System) では，UTC における 1980 年 1 月 6 日を 00:00:00 秒とした時刻配布を行う [22]．また，GPS 衛星より配布された時間と UTC との誤差は 50ns 以下である [23].

しかし，GPS を利用した時間軸同期を行うためには GPS 受信機材が必要となる．

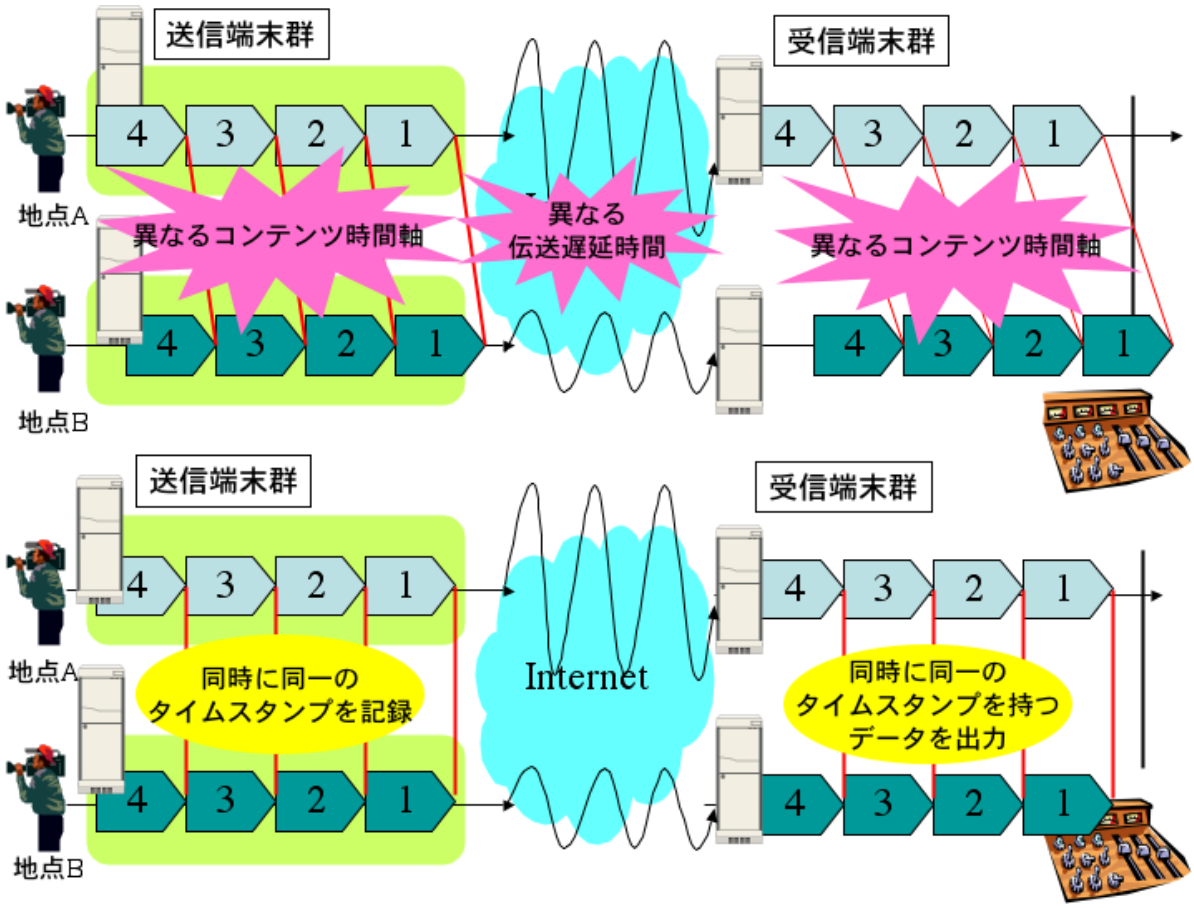


図 4.4: コンテンツ時間軸の差異と修正

**NTP**

収録端末の時間を統一する手法として外部の時刻同期用サーバを参照する方法がある．例として NTP(Network Time Protocol)[24] を挙げる．

NTP の階層構造を図 4.5 に示す．NTP は GPS や原子時計に直結されたサーバである Stratum1 を最上位とした，Stratum16 までの階層構造を持つ時刻同期システムである．Stratum の数が小さいほど，基準時間となる GPS や原子時計に近いため，精度が高い．NTP の特性として複数の異なる Stratum を持つ NTP サーバを選択した場合，より上位の Stratum が優先して選択される．また，同階層の Stratum を選択した場合，精度と分散を計算し，もっとも安定していると思われるサーバを基に時刻合わせが行われる．

NTP の問題点として，複数の異なる地点に存在する収録端末が参照する時刻に対する信頼性が挙げられる．地点 A では NTP サーバ A を，地点 B では NTP サーバ B を参照した場合，NTP サーバ A と B の示す時間が異なる可能性がある．また，地点 A と地点 B において同一の NTP サーバ A を参照した場合，NTP サーバ A までの経路によっては地点 A と B で得られる時間が異なる [25]．

また NTP は伝送遅延時間が往復の経路で同じであることが前提となっているため，往

復の経路が異なるなどの状況の場合，正確な時間を取得することができない [26].

これらの問題の解決策の一つとして，複数の同一 Stratum に所属する NTP サーバを参照し，最も確からしい時間を配布しているサーバを選出する方法の採用が挙げられる [26].

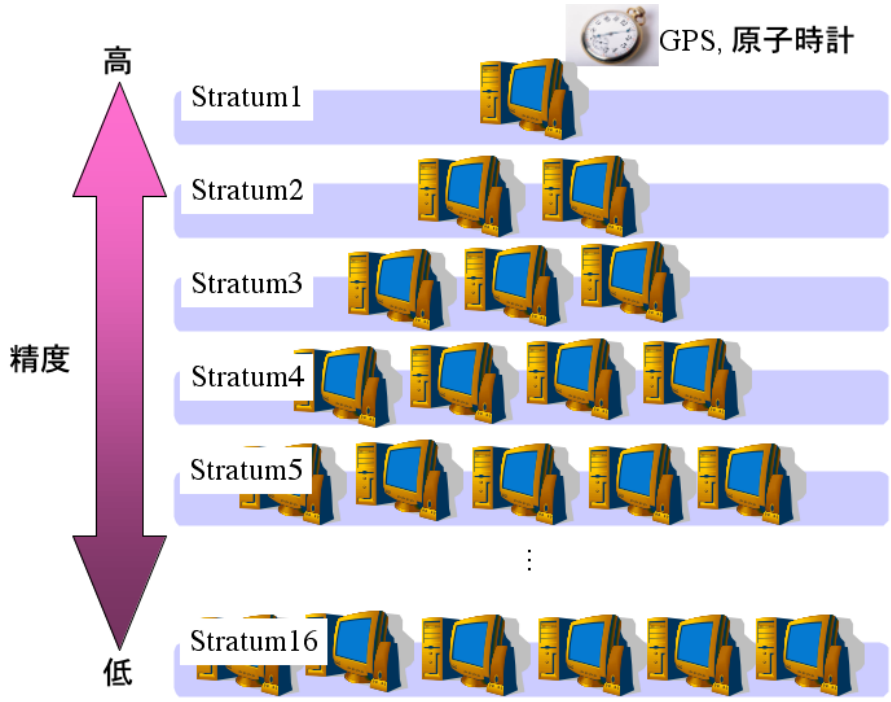


図 4.5: NTP の持つ階層構造

### 4.4.3 相対時間共有

時間軸を共有する必要のあるグループ内のみで完結した時刻を用いた相互通信することにより，時間軸同期を行うシステムの例として，TSP を挙げる.

#### TSP

TSP(Time Synchronization Protocol)[27] のシステム概要を図 4.6 に示す．TSP とは，ローカルエリアネットワーク内に存在するマシンの時刻を平均化するものである．時刻同期精度はマスタのクロック精度によって決められてしまうが，安価に時刻同期が可能である.

timed が実行されたマシンは，マスタとして設定されたサーバに時刻の問い合わせ，その時刻に合わせる．その後，マスタが定期的に送る同期メッセージを受け取り，時刻の補正を行う．この際，ICMP タイムスタンプリクエストによる時間差計測が行われる.

timed はローカルエリア内機器の時間を同期するために用いられる．そのため，機器上で使われるコンテンツの時間には関与しないため，コンテンツの開始時間などについては

他の手法と併用する必要がある．また，ローカルエリアネットワーク外にあるマシンとの同期は考慮されていない．

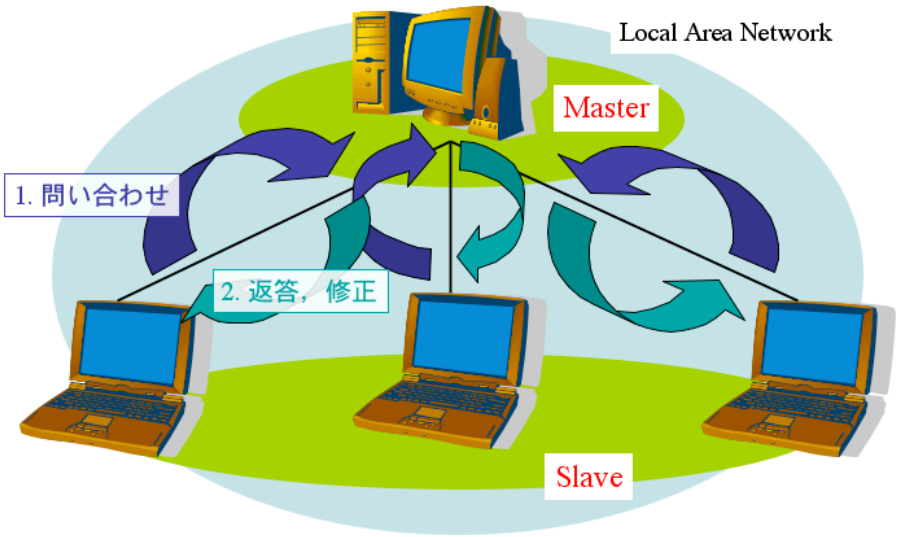


図 4.6: The Time Synchronization Protocol for UNIX

### 4.4.4 時間軸共有機構

時間軸共有機構について，時間軸共有をするための基準時刻の観点より，絶対時間と相対時間に大別し，解説を行った．

一般的に絶対時間を基準として時間軸共有を行うシステムは，地理的に異なる地点に存在する複数ノード間での時刻同期を視野に入れたものが多い．相対時間を地理的に異なる地点での時間軸共有に用いる場合，ネットワーク伝送遅延時間や遅延時間の揺らぎを別途考慮する必要がある．

一方で第 4.4.1 節で述べた送受信端末の双方で実現する同期手法では，受信端末群内でのコンテンツ再生タイミングの共有が必要となる．この課題に対し，受信端末群内でそれぞれの再生遅延時間を交換し，最も再生遅延時間の大きい端末と同期を取る．

本研究の時間軸共有に関する動作概要を図 4.7 に示し，詳細を以下に述べる．

#### 送信機器群: 自律分散型同期

第 4.4.1 節で述べたストリーム同期要件を満たすためには，受信したストリーミングデータの時間軸別の比較が必要となる．比較の基準となるコンテンツに対する時間軸の，付加手法について述べる．

時間軸付加手法の例として，MPEG における STC(System Time Clock) を挙げる．STC を用いた同期概要を図 4.8 に示す．MPEG2 において映像や音声などのデータは，アクセ

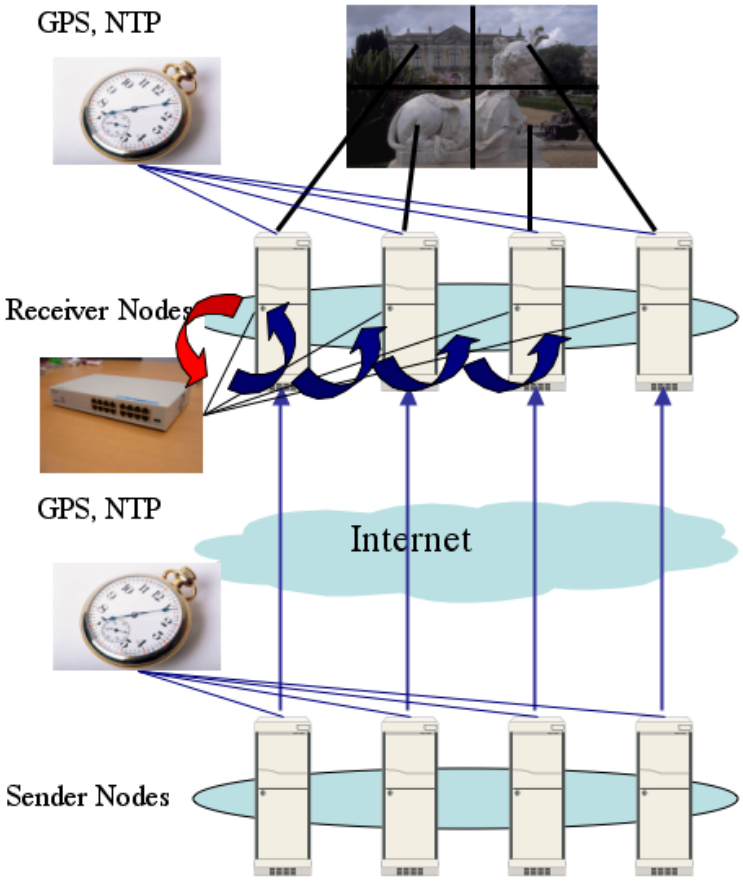


図 4.7: 時間軸共有

スユニットと呼ばれる復号・再生の単位で扱われる．各アクセスユニットに対し，STC におけるどのタイミングで復号し，再生するかという時間情報を付加することにより，STC を基準とした映像と音声の同期が行われている．

STC のような基準時刻に対し，コンテンツデータをマッピングすることで，フォーマットに依存しない複数ストリーム同期が可能となる．

本研究では，まず始めに各端末による GPS, または複数の NTP サーバ参照により，基準となる絶対時刻を統一する．同期された時刻を RTP ヘッダ内の TS フィールドに挿入する．これにより，絶対時間を基準としたコンテンツデータとの関連付けを行う．

#### 受信機器群: 端末間における協調同期

第 4.4.1 節で述べたストリーム同期要件を満たすためには，受信機器群内でデータ再生タイミングの共有が必要となる．コンテンツ時間軸の処理概要を図 4.9 に示す．受信機器群を結ぶ LAN を形成し，LAN 内ブロードキャストによって再生タイミングの交換を行う．以下に端末間における協調同期の詳細を示す．

LAN 内ブロードキャストによる協調同期の概念図を図 4.9 に示す．ここでは同期を行う

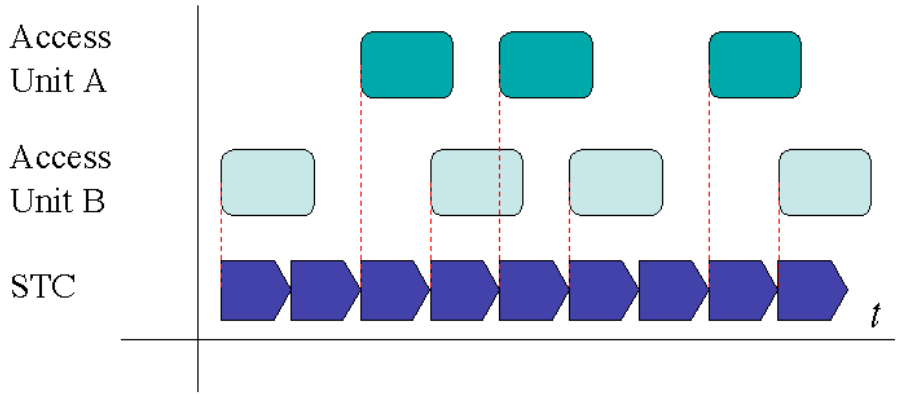


図 4.8: STC を用いた時間軸付加

ための指標として，個々の受信端末における再生遅延時間を用いる．協調同期の流れを以下に示す．

1. ユーザは同期を行う対象となる受信機群において一意の Master となるマシンの決定をする
2. Master は自端末のデータ受信時間と収録・再生環境による再生遅延時間を LAN 内ブロードキャストする
3. Master によるブロードキャストに対し，全ての受信機は各端末の再生遅延時間との照合を行い，個々の受信端末における必要バッファ量を算出する

ブロードキャストされた再生遅延時間に対し，任意の受信端末における再生遅延時間が大きかった場合，受信処理不可時間として Master に対しリセット要求を行う．こうして決められた各受信ノードの待機時間を後述する Buffering Module に通知する．

## 4.5 Buffering Module: 再生遅延時間の吸収

第 4.1.1 節では，収録・再生環境の差異隠蔽を実現するために時間軸を指標とした統一的な同期を実現することが必要であることを述べた．同期を目的としたバッファリング実行部としてアプリケーション，OS，ネットワーク，機材の 3 つが挙げられる．以下にそれぞれの部でバッファリングを行う特徴について述べる．

- アプリケーション
  アプリケーションとしてバッファ機構を提供するため，ユーザ利用コストが最も低い．しかしバッファリング機構としてアプリケーションに組み込む必要があるため，多様化の進むフォーマットに対し個別に開発を行わなければならず，開発コストが高い．加えて転送機器だけでは取得できない収録・再生用機器内の処理時間差を何らかの方法によって取得する必要がある．

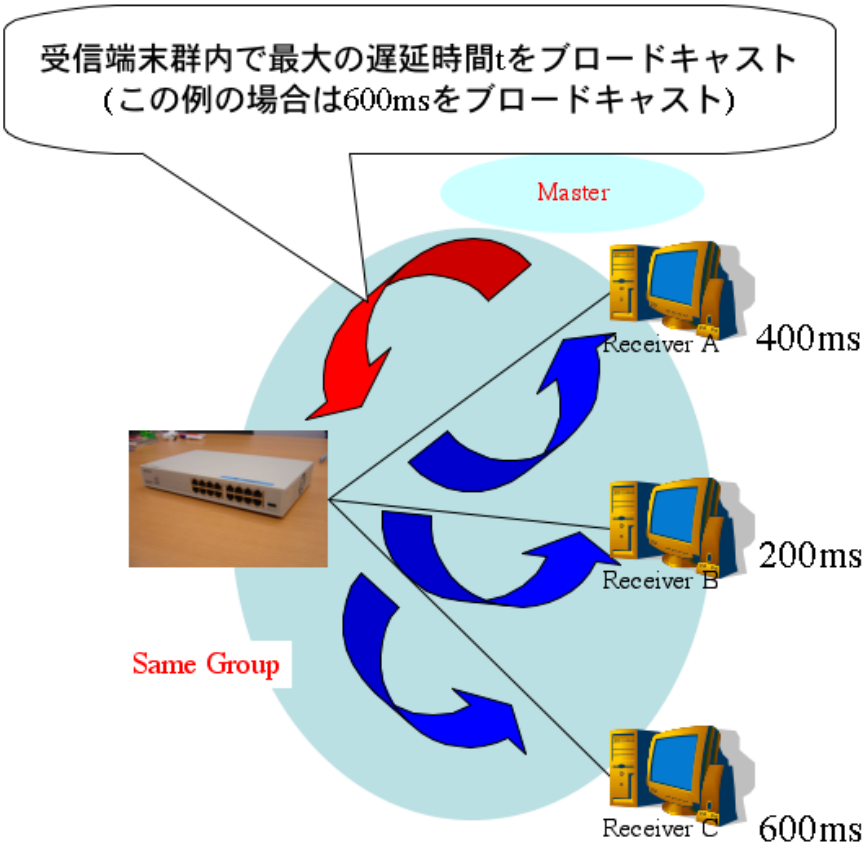


図 4.9: 受信端末群内における最大遅延時間の共有

- OS
  Kernel レベルでのバッファリングである．様々なアプリケーションでの応用利用が可能であるというメリットがある．しかし OS 依存になってしまうというデメリットがある．また，転送機器だけでは取得できない収録・再生用機器内の処理時間差を何らかの方法によって取得する必要がある．

- ネットワーク
  帯域制御システムを利用したバッファリングである．アプリケーションに最小限の手を加えるだけでシステムの適用が可能であるというメリットがある．またフォーマットの種類や OS を選ばない．しかしアプリケーションや OS と同様に，転送機器だけでは取得できない収録・再生用機器内の処理時間差を何らかの方法によって取得する必要がある．

- 機材
  長野オリンピック開会式に用いられたタイムラグアジャスタ [28] に見られるバッファリング用専用機材を用いる手法が考えられる．映像出力の最終段に用いる事により，収録・再生環境の差異を絶対的に吸収できるというメリットがある．しかし，バッ

ファリング用専用機材が対応したフォーマットに変換する必要がある．

本研究では，収録・再生環境の組み合わせに対し，汎用的に適応させるために，最も柔軟性が高いネットワークによるバッファリングを採用する．

第 4.4.4 節に述べた受信機器群における協調同期によって導かれた必要バッファ量を帯域制御システムに通知し，ストリーム単位でのバッファリングを行う．

## 4.6 まとめ

本章では第 3 章で述べた複数ストリームの同期要件に基づいた設計について述べた．Negotiation Module, Buffering Module, Average Module の 3 つのモジュールについて設計を行った．次章では本章で述べた設計を基に行った実装について述べる．

# 第5章　複数ストリームにおける同期再生の実現

## 5.1　実装概要

本章では，第 4 章の設計に基づいた実装に関して述べる．

本研究の実装概要図を図 5.1 に示す．始めに本機構の有用性を評価するためのアプリケーションとして選択をした DVTS について述べる．送信端末の処理について述べた後に，受信端末の処理として Average Counter Module, Negotiation Module について述べる．最後に Buffering Module として用いた dummynet について述べる．

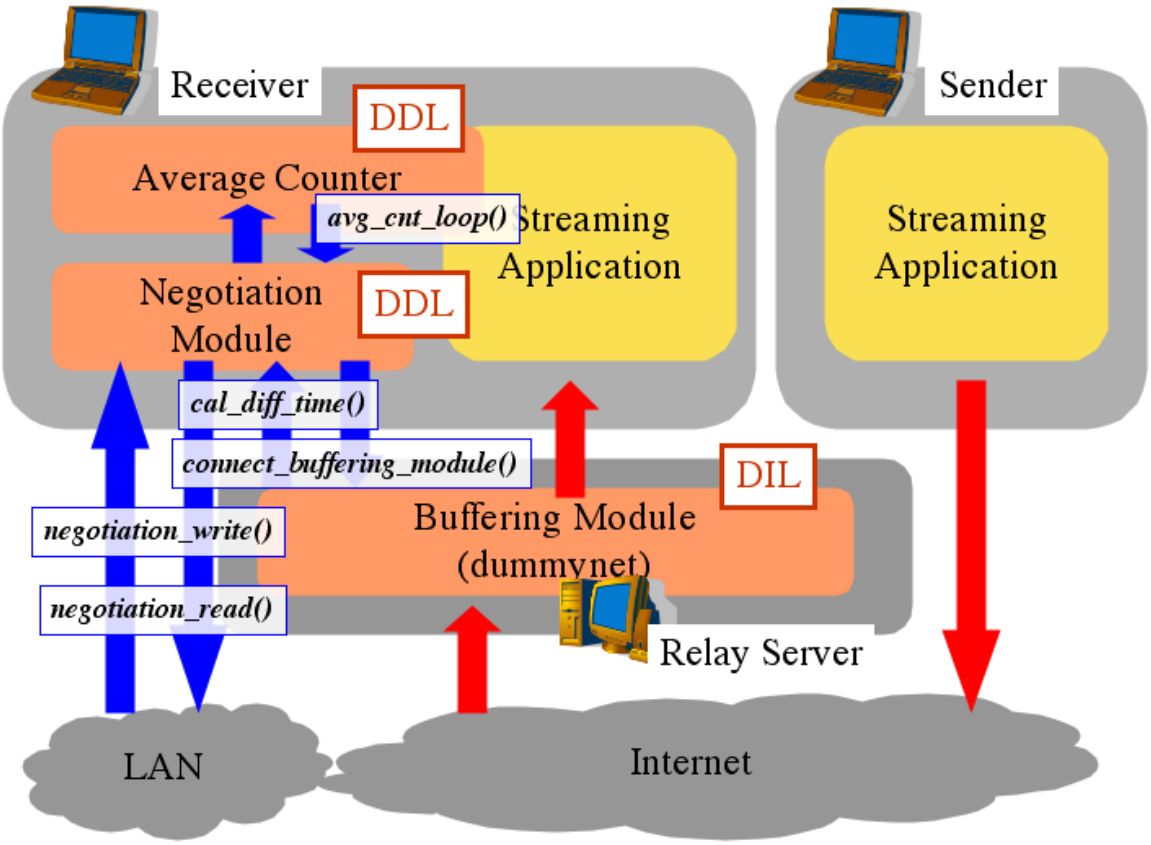


図 5.1: 実装概要図

本研究の実装環境を表 5.1 に示す.

表 5.1: 実装環境

| | |
|---|---|
| OS | debian GNU Linux Kernel 2.6.8-3-686 |
| コンパイラ | gcc 3.3.5 |
| プログラミング環境 | C 言語 |

## 5.2 DVTS

本研究では有用性を評価する為のアプリケーションとして，DVTS を選択した．以下に DVTS の概要，実装に用いるメリットを述べる.

- IEEE1394 からの映像・音声データの入出力に Isochronous 転送を用いる
- 音声は IEEE1394 からの入力を非圧縮で転送する
- IPv4/IPv6 に対応する
- RTP を利用した通信を行う
- Unix 用ソフトウェアとして，Open Source で提供されている
- IEEE1394 出力だけでなく，ディスプレイ出力にも対応する
- DV だけでなく，HDV 転送に向けての開発が始まっている

これらの特徴より，本研究はリアルタイム性が高く，ソースコードが公開されているため開発の行いやすい DVTS を選択し，実装を行った.

## 5.3 送信端末処理

第 4.4.1 項で述べた複数ストリームの同期手法より，送信端末において必要な事柄は，以下の 2 点である.

- 送信機器群における絶対時間の同期
- フレーム単位での絶対時間の挿入

本機構では，利用する全ての送受信機器は第 4.4.2 項で述べたように，3 つ以上の NTP サーバを参照した同期が実現されていると仮定する．

絶対時間の挿入の流れを図 5.2 に示す．DVTS の実装では，入力された DV データは header, subcode, video aux, audio, video といったセクタ単位で扱われている．加えて，video セクタの終了では，*_dvdif_flush()* が呼び出される．絶対時間の取得は*_dvdif_flush()* 関数内において行う．前フレームの送信後に *gettimeofday()* を発行し，取得された時間を次フレームを構成する RTP ヘッダ内 TS フィールドに挿入する．

```
/*** 1 フレームの最後に呼び出される関数 ***/
static void
_dvdif_flush(struct dvsend_param *dvsend_param)
{
(省略)
/* パケットの送信 */
send_pkt(dvsend_param, outbuf, outbuf_len + sizeof(rtp_hdr_t));
outbuf_len = 0;
/* 1 フレーム送信完了 */
(省略)
/** 次フレームに向けたシーケンス番号，タイムスタンプの設定 **/
/* RTP sequence number will increase every time a RTP packet is sent */
rtphdr->seq = htons(ntohs(rtphdr->seq)+1);

/* RTP timestamp will increase every by frame. */
gettimeofday(&tv, NULL);
rtphdr->ts = htonl((tv.tv_sec%60)*1000+tv.tv_usec/1000);

/* The variable [frame] counts the number of frames read */
/* from the IEEE 1394 device */
frame++;
(省略)
return;
}
```

図 5.2: 送信側におけるタイムスタンプ処理

## 5.4 受信端末処理

第 4.4.1 項で述べた複数ストリーム同期手法より，受信端末において必要な事柄は，以下の 4 点である．

- 受信端末群における絶対時間の同期
- 受信したフレームのシーケンス番号と受信時間の照合による受信遅延時間の算出 (Average Counter Module)
- 受信機器間による最大再生遅延時間の共有 (Negotiation Module)
- 指定時間経過後，バッファリングを開始 (Buffering Module の呼び出し)

次項では上記の用件を基に行った実装について述べる．

### 5.4.1 受信端末における実装項目

DVTS に追加したコマンドを表 5.2 に示す．以下に Module 別にそれぞれの値についての説明を行う．

#### Buffering Module 用コマンドオプション

全ての受信端末に大して必要な項目である．Buffering Module として利用する帯域制御ノードの IP アドレスを設定する必要がある．

#### Average Counter Module 用コマンドオプション

再生遅延時間の揺らぎに対し，第 4.3 節に基づいた値設定を行う．許容揺らぎ幅 (%) と許容揺らぎパケット数の設定ができる．非設定時には許容揺らぎ幅は 10%，許容揺らぎパケット数は 10 に設定されている．また，Negotiation Module に引き渡すための平均受信遅延時間を求めるために，サンプルとなるコンテンツデータ受信ループ回数の設定ができる．非設定時は 300 回が設定される．

#### Negotiation Module 用コマンドオプション

第 4.4.4 項にて述べた手法に従い，受信端末群における最大再生遅延時間を求めるための値設定を行う．実行受信端末が Master に指定された場合，時間軸共有を目的としたネゴシエーションパケット送信間隔の設定ができる．また，ネゴシエーションパケットに挿入する再生遅延時間算出のために遅延追加要求係数とネゴシエーション時間の 2 つの値が設定可能である．遅延追加要求係数とは，Master が送信した予定時間に，指定されたフレームを受信できない端末が存在した場合，該当端末から送られた追加時間要求にか

ける係数である．遅延追加要求係数をかけることにより，受信時間の揺らぎを柔軟に吸収することができる．遅延追加要求係数の初期値は 1.2 である．ネゴシエーション時間とは Average Counter Module によって平均遅延時間が算出された後から Buffering Module に接続されるまでを指す．初期値として 20 秒が挿入されている．

Master のネゴシエーションパケット送信，Slave の追加時間要求を送信するため，それぞれの端末でブロードキャストアドレスを設定する必要がある．

ネゴシエーションの終了，Buffering Module の呼び出しのために，タイマ設定を行う．

加えて，主に再生・収録環境によって発生する遅延時間吸収のため，-A オプションによって遅延時間の加算ができる．

表 5.2: 実装コマンドオプション一覧 (受信側)

| コマンドオプション | 動作概要 |
|---|---|
| 複数受信機協調モード | |
| To: Buffering Module | |
| -a | dummynet の IP アドレス |
| -w | dummynet pipe ID |
| Average Counter Module | |
| -x | イニシャライズ用ループ（回） |
| -y | 許容揺らぎ幅 (%) |
| -z | 許容揺らぎパケット数 (回) |
| Negotiation Module | |
| -m | master(非設定時は slave) |
| -r | 遅延追加要求係数 |
| -t | 計算フレーム間隔 |
| -T | Negotiation パケット送出間隔 (秒) |
| -b | broadcast address |
| -c | broadcast 送受信用ポート番号 |
| -B | イニシャライズ用ループ (秒) |
| -A | 再生・収録機器用追加遅延時間 |

次項では，まず実際の実装を基にした処理概要について述べる．

### 5.4.2 実装関数と処理概要

DVTS に追加した関数とその動作概要をを図 5.3 に示す．通常の DVTS では，*fork()* により *dvrtp_loop()*(パケット受信) プロセスと *ieee1394dv_write_loop()*(データ書き込み) プロセスが発生する．本システムでは，DDL として Average Counter Module と Negotiation Module が存在する．Average Counter Module と Negotiation Module に必要な情報として，受信フレームのシーケンス番号と受信時のタイムスタンプがある．そのため，*dvrtp_loop()*

内に Average Counter Module のインターフェースである *avg_cnt_loop()* と Negotiation Module のインターフェースである *negotiation_loop()* の組み込みを行った．

本システムでは，*fork()* によって *dvrtp_loop()* プロセスと *ieee1394dv_write_loop()* プロセス派生後の動作関して，テスト動作期間と本動作期間に区分される．テスト動作期間では，*dvrtp_loop()* 内で実際のパケット受信・再生が開始された後，*avg_cnt_loop()* が平行動作を始める．これにより，個々の受信端末内におけるパケット到達の絶対時間監視が行われる．設定されたイニシャライズループ回数の完了後，*negotiation_loop* の呼び出しが発生する．これにより，第 4.4.4 節にて述べた LAN 内ブロードキャストによるネゴシエーションが開始される．これにより，個々の受信端末で必要なバッファリング時間を求められる．設定されたイニシャライズ時間後，必要バッファリング時間に設定された遅延時間加算分が加えられた後，Buffering Module へのバッファ時間の通知がなされる．

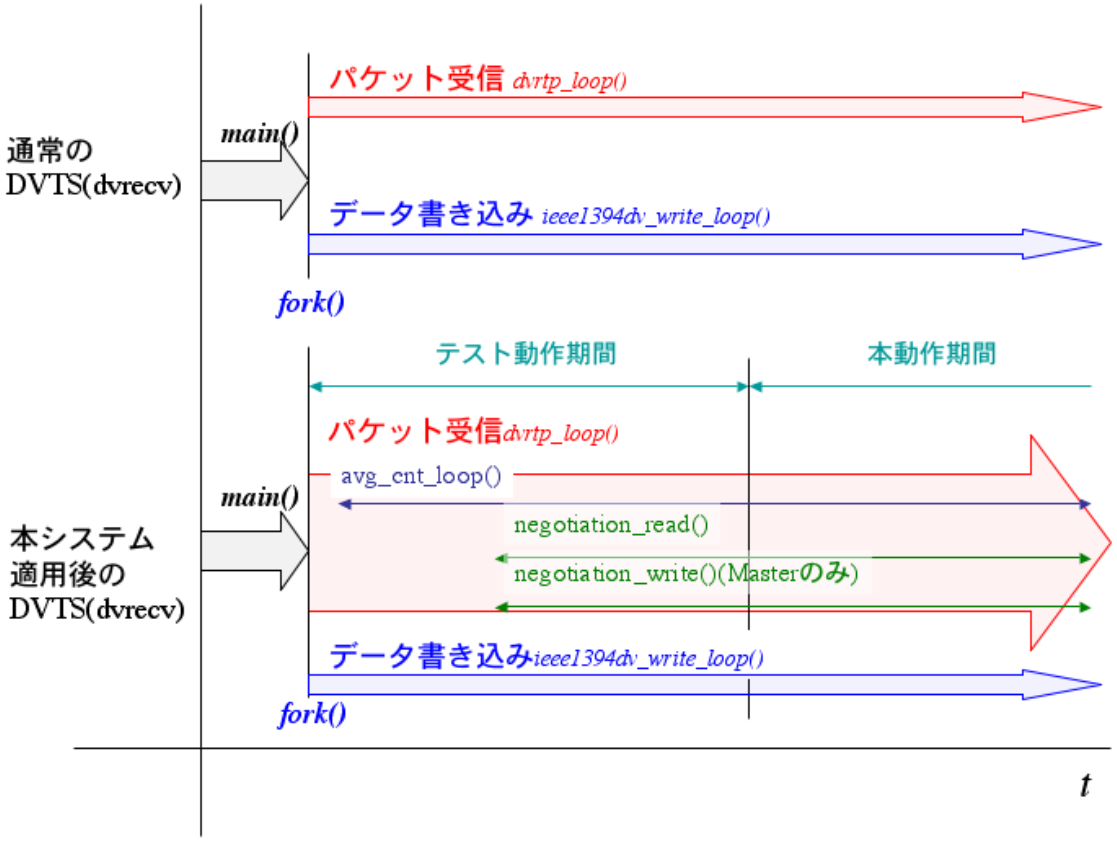


図 5.3: 受信側：DVTS への追加関数とその動作概要

本システムの一連のパラメータは，ストリーミングアプリケーション本体とは独立した構造体に格納した．全てのモジュールは *multi_stream_param()* を介したデータ共有を行う．

次項ではそれぞれの Module 内での実装について *multi_stream_param()* 構造体を参照し，述べる．

### 5.4.3 Average Counter Module

Average Counter Module の呼び出し処理を図 5.4 に示す．Average Counter Module は *dvrtp_loop()* 内で呼び出される．第 5.3 節で述べたように，DVTS では送信側において，RTP ヘッダ内のタイムスタンプを 1 フレームごとに更新する．これより，Average Counter Module は RTP ヘッダのタイムスタンプ更新をトリガとしてフレームの更新を認知し，時間計算処理を実行する．

```
void
dvrtp_read_loop (struct dvrecv_param *dvrecv_param){
  (省略)
   /** パケット受信ループの開始 **/
   while (1) {
  (省略)
      seq  = (ntohl(recvbuf[(sizeof(rtp_hdr_t)+80*i)/4]) >> 24) & 0xf;
  (省略)
      else if (rtphdr->ts != rtp_ts_prev) {/*1 フレームの先頭*/
        /** フレームシーケンス番号の登録 **/
        multi_stream_param.own_seq = seq;
        /** Average Counter Module の呼び出し **/
    avg_cnt_loop(&multi_stream_param);
  (省略)
    rtp_ts_prev = rtphdr->ts;
  (省略)
```

図 5.4: Average Counter Module の呼び出し

主に Average Counter Module にて利用される *multi_stream_param()* 構造体の要素を図 5.5 に示す．この構造体を踏まえ，Average Counter Module における平均遅延時間 (*own_avg_delay*) の算出について以下に述べる．

Average Counter Module ではまず始めにパケット受信時間，平均受信遅延時間，許容揺らぎ幅の算出を行う．各時間の算出処理を図 5.6 に示す．算出処理は，第 4.3 節に述べた遅延時間計算フローに従って構成される．

*dvrtp_loop()* 内でシーケンス番号の更新と共に発行された *avg_cnt_loop()* は，*gettimeofday()* を用いた絶対時間の取得を行う．これにユーザによって設定された追加遅延時間 (*additional_delay*) を加えたものを再生遅延時間とする．次に平均遅延時間 (*own_avg_delay*) を算出する．前フレームシーケンス番号受信時間 (*recv_time_prev*) と現在のフレームシーケンス番号受信時間 (*recv_time*) を基に，以下の式を用いて算出する．

$$own\_avg\_delay = recv\_time\_prev - recv\_time$$

```
struct multi_stream_param{
(省略)
  /********** Average Counter **********/
  int avg_init_time; /* イニシャライズ回数 */
  int avg_accept_jitter_percent; /* 許容揺らぎ幅 (%) */
  int avg_accept_packet; /*許容揺らぎ回数*/
  int init_loop_count;/*loop 回数*/
  int threshold_shift_count;/*揺らぎ回数*/
  int tmp_time; /* 仮平均時間 */
  int own_avg_delay;  /* 平均遅延時間 */
  int avg_loop_seq_term; /*ループ回数*/
(省略)
}
```

図 5.5: multi_stream_param 構造体 Average Counter Module 関係部分

遅延時間の揺らぎを偶発的なものと慢性的なものを切り分けるための許容揺らぎ幅 (*tmp_threshold*) を平均遅延時間 (*own_avg_time*) と許容揺らぎ幅 (%)(*avg_accept_jitter_percent*) を用いて算出する．算出には以下の式を用いる．

$$tmp\_threshold = own\_avg\_tiime * avg\_accept\_jitter\_percent$$

これらの値を基にした偶発的な揺らぎと慢性的な揺らぎを切り分けた平均遅延時間の算出処理を図 5.6 に示す．揺らぎを偶発的なものと慢性的なものに切り分けるための指標として，許容揺らぎ幅超過回数のカウント (*threshold_shift_count*) がある．受信遅延時間がそれまでの受信平均遅延時間に対し，許容揺らぎ幅以内であった場合，超過回数カウントは 0 に設定される．受信遅延時間が許容揺らぎ幅を超えた場合，超過回数がカウントが開始する．超過回数カウント中は，平均遅延時間に対する変更は行われず，仮平均時間 (*tmp_time*) として算出される．許容揺らぎ回数を超過した場合，平均遅延時間は仮平均時間の値に更新される．許容揺らぎ回数内で揺らぎの収束が発生した場合，仮平均時間は破棄される．

設定されたループ回数 (*avg_loop_seq_term*) 後，そのタイミングでの平均遅延時間を基に，Negotiation Module による受信端末群協調動機が開始される．次項では，Negotiation Module の詳細な動作について述べる．

### 5.4.4 Negotiation Module

Negotiation Module は Average Counter からの呼び出しを受けて実行される．主に Negotiation Module にて利用される *multi_stream_param()* 構造体の要素を図 5.8 に示す．Ne-

```
int avg_cnt_loop(struct multi_stream_param *multi_stream_param){
(省略)
/***** 受信時刻の取得 *****/
gettimeofday(&tv,0);
multi_stream_param->recv_time =
  (tv.tv_sec%60) * 1000 + tv.tv_usec / 1000;
(省略)
/***** 受信遅延時間算出 ****/
multi_stream_param->own_avg_delay =
  multi_stream_param->recv_time_prev -
  multi_stream_param->recv_time;
(省略)
/* 許容揺らぎ幅算出 */
tmp_threshold = multi_stream_param->own_avg_time *
  (multi_stream_param->avg_accept_jitter_percent/100);
(省略)
}
```

図 5.6: Average Counter Module: 各時間の取得

gotiation Module の実装に関して，Master と Slave の大別を行い，それぞれの詳細について述べる．

### Master 処理

Master は第 4.4.4 項で述べた端末間における協調同期を実現するための定間隔なメッセージング発行を行う．Negotiation Module で使用するパケット構造体を図 5.9 に示す．プロトコルバージョン (*version*) の後に，フラグ (*f*) が挿入される．また，ネゴシエーションに用いるための再生遅延時間 (*ts*) が挿入される．Slave で，Master の指定した時間に該当するシーケンス番号の受信が不可能と判断された場合，処理不可時間 (*rts*) の挿入が行われる．

Master に関する処理の流れは，以下の 4 つに大別される．

1. 定期的なメッセージングを行うための送信関数のタイマ登録
2. ネゴシエーションパケットへのデータ設定，送信
3. Slave よりリセット要求を受信した際の再計算処理とフラグ設定

次にそれぞれの詳細について述べる．

```
int avg_cnt_loop(struct multi_stream_param *multi_stream_param){
(省略)
/** 許容揺らぎ幅・回数内処理 **/
if(tmp_min_threshold <
   tmp_recv_time < tmp_max_threshold){
  /* 許容揺らぎパケット数カウントのリセット */
  multi_stream_param->threshold_shift_count = 0;
}else{
  /*** 許容揺らぎパケット数超過処理 ***/
  if(multi_stream_param->threshold_shift_count >
    multi_stream_param->avg_accept_packet){
    /* 平均遅延時間 (own_avg_time) の置き換え */
    multi_stream_param->own_avg_time =
      multi_stream_param->tmp_time;
    /* 許容揺らぎパケット数カウントのリセット */
    multi_stream_param->threshold_shift_count=0;
  }else{
    /** 許容揺らぎパケット数を超えた場合に備えての仮平均時間の算出 **/
    multi_stream_param->tmp_time =
      (multi_stream_param->own_avg_delay +
      (multi_stream_param->recv_time -
       multi_stream_param->recv_time_prev))/2;
    /*許容揺らぎパケット数のカウント*/
    multi_stream_param->threshold_shift_count++;
  (省略)
 else{
    if(multi_stream_param->init_loop_count ==
       multi_stream_param->avg_init_time){
        /* (Master: Negotiation Module の呼び出し) */
  (省略)
```

図 5.7: Average Counter Module: 揺らぎの吸収

Master におけるネゴシエーションパケット送信関数 (*negotiation_write*) をタイマ登録することによって定期的に実行する．タイマ登録概要を図 5.10 に示す．*sigaction()* を用いて *negotiation_write()* のシグナルハンドリングを行い，*setitimer()* によってタイマ登録を行った．実行の間隔は *negotiation_time_term* 変数に設定された値を用いる．

ネゴシエーションパケット送信関数の概要を図 5.11 に示す．始めにネゴシエーションパケットに挿入する未来のフレームシーケンス番号と，その受信予定時間の算出を行う．

```
struct multi_stream_param{
(省略)
  /******** Negotiation Module *********/
  int read_bsock; /*受信用ソケット*/
  int write_bsock; /*送信用ソケット*/
  int bcast_port; /*ポート番号*/
  char *broadcast_addr;/*ブロードキャストアドレス*/
  int broadcast_freq;         /* ネゴシエーション時間  */
  unsigned int additional_delay;/* 収録・再生環境用追加遅延時間 */
  struct sockaddr_storage in_addr;/* 受信用ソケットアドレス構造体 */
  struct sockaddr_storage out_addr;/* 送信用ソケットアドレス構造体 */
  unsigned int result_delay;              /* 発生遅延時間 */
  unsigned int rts;/* リセットリクエスト用追加要求時間 */
  int negotiation_flag;/* リセットリクエスト用追加要求時間 */

  /*** Masterから送信されたデータ  ***/
  int master_negotiation_flag;  /* ネゴシエーションフラグ */
  int negotiation_seq_term;/* ブロードキャスト間隔 */
  int master_time;       /* 再生指定時間 */
  int recv_seq;  /* 再生指定フレームシーケンス番号 */
  int recv_seq_prev; /* 再生指定フレームシーケンス番号 */

  /* time stamp and seq number */
  int recv_time;          /*コンテンツ受信時間*/
  int recv_time_prev;       /*コンテンツ前回受信時間*/
  int own_seq;       /*受信フレームシーケンス番号*/

  /* reserve factor */
  float reserve_factor;/*遅延追加要求係数: 1以上*/

  /* packet structure */
  negotiation_pkt_t *negotiation_pkt;
          /* 受信ネゴシエーションパケット */
(省略)
}
```

図 5.8: multi_stream_param 構造体 Negotiation Module 関係部分

```
/** Negotiation Header **/
typedef struct {
  unsigned int version:1; /* プロトコルバージョン */
  unsigned int r:1;       /* フラグ */
  unsigned int p:30;      /* 予備領域 */
  unsigned int ts:32;     /* 遅延時間 */
  unsigned int rts:32;    /* 処理不可時間 */
} negotiation_hdr_t;
```

図 5.9: Negotiation Module：パケット構造体

受信予定フレームシーケンス番号 (*request_seq*) は，単純増加される．受信予定時間 (*request_time*) は，Master 端末での最新のフレーム受信時間 (*recv_time*) と平均受信遅延時間 (*own_avg_delay*)，設定されたネゴシエーション間隔 (*negotiation_seq_term*) より，以下の式で表される．

$$request\_time = recv\_time + (own\_avg\_delay * negotiation\_seq\_term);$$

ネゴシエーションパケットに関して，Slave に対する動作指示を目的としたフラグ挿入を行う．フラグで表現されるブロードキャスト情報は以下の 4 つである．

- 定常
  Master から定常時に送信される．定常 flag を受信した場合，通常の遅延時間計算が継続して行われる．Master が指定した任意のフレームのシーケンス番号 (seq) と，それを受信する絶対時間 (*ts*) がある．

- リセット要求
  Slave において，Master から要求された時間に該当するデータを処理できないと判断した場合に送信される．絶対時間 (*ts*) に対し，受信が間に合わない受信端末は，追加遅延要求として，処理不可時間 (*rts*) フィールドに不足時間を挿入し，ブロードキャストを行う．Master はリセット要求通知を受信・処理するが，他の Slave はこれを受け付けない．

- リセット
  リセット要求を受けた Master が Slave に再計算を求めるために投げる．これを受けた Slave はそれまでの遅延時間計算結果を破棄し，再計算を行う．

- 本動作開始
  遅延時間計算結果を基に本動作を開始する．遅延時間計算は引き続き行われ，log として出力される．

```
int init_negotiation_write
  (struct multi_stream_param *multi_stream_param){
  struct sigaction sa_alarm;
  struct itimerval itimer;

  multi_stream_param->master_negotiation_flag=NEG_DEFAULT;

  memset (&sa_alarm, 0, sizeof(sa_alarm));
  sa_alarm.sa_handler=negotiation_write; /*アクションの設定*/
  sa_alarm.sa_flags=0;

  if(sigaction(SIGALRM, &sa_alarm, NULL) < 0){
    perror("sigaction");
  }

  /** ネゴシエーションパケット送信間隔設定 **/
  itimer.it_value.tv_sec =
    itimer.it_interval.tv_sec =
multi_stream_param->negotiation_time_term;
  (省略)
  /** negotiation_write の呼び出しタイマ登録 **/
  if(setitimer(ITIMER_REAL, &itimer, NULL) < 0){
    perror("setitimer");
    exit(1);
  }

  return 1;
}
```

図 5.10: Negotiation Module:Master: ネゴシエーションパケット送信タイマ登録

ネゴシエーションパケットに挿入するフラグ設定を図5.11に示す．通常時は*NEG_DEFAULT*が設定される．Slave よりリセット要求を受信した場合，受信予定時刻 (*request_time*) は，Slave より受信したネゴシエーションパケット内の処理不可時間 (*rts*) と，設定された遅延追加要求係数 (*reserve_factor*) を用いて以下の式によって算出される．

$$request\_time = request\_time + (rts * reserve\_factor)$$

受信予定時刻の再計算の後，フラグはリセット要求を示す*NEG_ALL_RESET*が挿入される．また，一連のネゴシエーションが終了した後，フラグは*NEG_START_STREAM_SYNC*

が設定され，受信機群内でのネゴシエーションは完了したとして扱われる．

```
void negotiation_write(){
  (省略)
  /* 再生遅延時間の初期値 */
  if(multi_stream_param.delay_time == 0){
    multi_stream_param.delay_time = multi_stream_param.own_avg_delay;
  }
  (省略)
  /************ フラグ設定 ***********/
  /* ネゴシエーション終了後設定 */
  if(multi_stream_param.negotiation_flag==3){
    multi_stream_param.master_negotiation_flag=NEG_START_STREAM_SYNC;
  }
  /* Slave よりリセット要求受信時 */
  else if(multi_stream_param.master_negotiation_flag==NEG_RESET_REQ){
    multi_stream_param.delay_time =
      multi_stream_param.negotiation_pkt->ts +
      (multi_stream_param.negotiation_pkt->rts *
      multi_stream_param.reserve_factor); /*再計算*/
    /* 全 Slave に対する再計算要求 */
    multi_stream_param.master_negotiation_flag=NEG_ALL_RESET;
  }
  else{
    /* 通常時 */
    multi_stream_param.master_negotiation_flag=NEG_DEFAULT;
  }
(省略)
}
```

図 5.11: Negotiation Module: Master: 計算処理とネゴシエーションフラグ設定

ネゴシエーションパケットへのデータ設定と，その送信の概要を図5.12に示す．前述したネゴシエーションフラグ (*negotiation_flag*)，指定フレームシーケンス番号 (*request_seq*) とその受信予定時間 (*request_time*) をネゴシエーションパケット (*negotiation_pkt*) に挿入する，送信する．送信したネゴシエーションパケットにリセットフラグ (*NEG_ALL_RESET*) が立っていた場合，デフォルト値 (*NEG_DEFAULT*) の再設定を行う．

```
void negotiation_write(){
  (省略)
  /**** negotiation_pkt へのデータ設定 ****/
  negotiation_pkt->f = multi_stream_param.negotiation_flag; /* flag */
  negotiation_pkt->seq = request_seq;
  negotiation_pkt->ts = request_time;
  negotiation_pkt->rts = 0;
  /**** broadcast *****/
  if((ret = sendto(multi_stream_param.write_bsock,
    (char *)&negotiation_pkt, sizeof(negotiation_pkt), 0,
    (struct sockaddr *)&multi_stream_param.out_addr,
    length)) < 1){
    perror("negotiation_write sendto");
  }

  /* after broadcast */
  if(multi_stream_param.master_negotiation_flag==NEG_ALL_RESET){
    /* reset flag */
    multi_stream_param.master_negotiation_flag=NEG_DEFAULT;
  }
  (省略)
```

図 5.12: Negotiation Module: Master: ネゴシエーションパケットの送信

### Slave 処理

Slave は第 4.4.4 項で述べた端末間協調同期を実現するために，以下の 4 つの処理を行う．

- Master から送信されたネゴシエーションパケットの受信
- ネゴシエーションパケットに挿入された再生遅延時間との差分計算
- ネゴシエーションパケットに挿入，指定された時間に対し，自端末の受信遅延時間が上回っていた場合のリセット要求処理
- Buffering Module への接続と本動作の開始

これら 4 つの処理遷移をフラグ (*negotiation_flag*) を用いて管理をする．Slave 内の処理は以下の 3 つに区分される．Buffering Module への接続は Negotiation Module の実行中に行われるため，フラグ表現としては Negotiation Module 動作開始に含まれる．

- Average Counter Module 開始
  個々の受信端末で絶対時間とフレームシーケンス番号より遅延時間計算が行われる期間を指す.

- Negotiation Module 初期設定
  計算に用いる値の初期化の他，Buffering Module を呼び出すまでのタイマ設定処理を行う.

- Negotiation Module 動作開始
  Negotiation Module の動作開始期間を示す.

次にそれぞれの処理についての詳細を述べる.

Master から送信されたネゴシエーションパケットの受信概要を図 5.13 に示す．ネゴシエーションパケット (*negotiation_pkt*) 受信後，ホストバイトオーダーへの変換を行う．次に，受信したネゴシエーションパケット内のフラグによって以下の 3 つの動作に分類される.

- Master から送られてきた再生遅延時間との比較・計算処理 (*NEG_DEFAULT*)
- 自端末が Master であった場合，Slave から送信された再計算要求に対する処理,
- 再計算処理

再生遅延時間差の計算処理 (*cal_diff_time*) の概要を図 5.14 に示す．Master から送信された再生遅延時間 (*ts*) に対し，自端末で他端末との同期に必要な待機時間 (*result_delay*) を求める必要がある．Average Counter Module で算出された平均受信遅延時間 (*own_avg_delay*)，Master から送信された再生遅延時間 (*ts*)，最新のフレーム受信時刻より，再生遅延時間の差分 (*delay_diff*) を求める．算出には以下の式を用いる.

$$delay\_diff = ts - own\_avg\_delay$$

求められた時間が正の値になった場合，解を必要バッファ時間 (result_delay) として扱う．解が負の値になった場合，再生時間が Master が指定したものよりも大きいため，絶対値を処理不可時間 (rts) として扱い，リセットリクエストの送信 (send_reset_request) の呼び出しを行う.

リセット要求 (*send_reset_request*) の概要を図 5.15 に示す．ネゴシエーションパケットのフラグに *NEG_RESET_REQ* を挿入する．フレームシーケンス番号 (*seq*) とタイムスタンプ (*ts*) の領域には，マスタから受信したものを継続して代入する．処理不可時間 (*rts*) の挿入を行い，設定したブロードキャストアドレスに対して送信を行う.

また，Buffering Module の呼び出しは Negotiation Module が実行後，設定された時間 (*broadcast_freq*) 経過した後に発行される．Negotiation Module 内における Buffering Module 呼び出しタイマ登録を図 5.16 に示す．Buffering Module 呼び出し関数 (*connect_buffering_module()*) に対し，*sigaction()* を用いたシグナルハンドリングを行った後，*setitimer()* によってタイマ登録される.

```
/***** Cal Negotiation Time from recv time stamp *****/
memset(buf, 0, sizeof(buf));
negotiation_pkt = (negotiation_pkt_t *)buf;

if(( n = recvfrom(multi_stream_param->read_bsock,
  buf, sizeof(buf),0,
  (struct sockaddr *)&from, &length)) < 0){
(省略)
  /* ネットワークバイトオーダーからホストバイトオーダーへの変換 */
(省略)
  switch(negotiation_pkt->f){
    case NEG_DEFAULT: /*通常時*/
      /*Master との時間差分計算*/
      cal_diff_time(multi_stream_param);
      break;
    case NEG_RESET_REQ: /*Slave の投げたリセット要求*/
      if(multi_stream_param->master == 1){
        /** Master の場合，再計算処理の開始 **/
        multi_stream_param->master_negotiation_flag=NEG_RESET_REQ;
      }
/** Slave の場合は処理は処理なし **/
      break;
    case NEG_ALL_RESET: /*再計算要求*/
      /*それまでの値を破棄*/
      multi_stream_param->result_delay = 0;
      break;
(省略)
```

図 5.13: Negotiation Module: ネゴシエーションパケットの受信

### 5.4.5 Buffering Module への接続

Buffering Module への接続用関数である *connect_buffering_module()* の概要を図 5.17 に示す．Buffering Module は Negotiation Module で求められた待機時間 (*result_delay*) を基に動作を行う．また，後述する Buffering Module として採用した dummynet の動作に必要な引数として，dummynet 動作端末の IP アドレス (*dummynet_addr*) とパイプ番号 (*dummynet_pipe_id*) が必要である．この ipfw コマンドを rsh，または ssh を用いて実行する．コマンドの呼び出しは popen を用いて行う．

```
void cal_diff_time(struct multi_stream_param *multi_stream_param){
  (省略)
  delay_diff =
    multi_stream_param->negotiation_pkt->ts -
    multi_stream_param->own_avg_delay;

  if( delay_diff > 0){
    /** Masterから送信された再生遅延時間より早い場合 **/
    multi_stream_param->result_delay = delay_diff;
  }
  else{
    /** Masterから送信された再生遅延時間より遅い場合 **/
    multi_stream_param->rts = delay_diff * (-1);
  (省略)
    /* リセットリクエストの送信 */
    if( send_reset_request( multi_stream_param) < 0 ){
      printf("send reset request Failed!\n");
    }
  (省略)
```

図 5.14: Negotiation Module: Master との時間軸差異計算

## 5.5 Buffering Module

本研究では帯域制御システムとして dummynet[29] を用いる．dummynet は本来はプロトコルテスト用にデザインされたネットワークエミュレータである．以下に dummynet を用いるメリットを述べる．

- ipfw と併用する事で中継ノードとして動作ができる
- ms 単位の遅延を生成する事ができる
- パイプ単位での制御が可能
- Unix 用ソフトウェアとして，OpenSource で提供されている

これらの特徴より，本研究は中継ノードとして動作，パイプ単位で ms レベルでの制御が可能な dummynet を帯域制御システムとして使用，実装を行う．

```
int send_reset_request
  (struct multi_stream_param *multi_stream_param){
  (省略)
  /**** negotiation_pkt へのデータ設定 ****/
  negotiation_pkt.f = NEG_RESET_REQ; /* リセットリクエスト */
  /*マスタから受信した recv_seq と recv_time を送り返す*/
  negotiation_pkt.seq = multi_stream_param->recv_seq;
  negotiation_pkt.ts = multi_stream_param->recv_time;
  /* 遅延追加時間 */
  negotiation_pkt.rts = multi_stream_param->rts;

  /**** broadcast *****/
  if((ret = sendto(multi_stream_param->write_bsock,
    (char *)&negotiation_pkt, sizeof(negotiation_pkt), 0,
    (struct sockaddr *)&multi_stream_param->out_addr,
    length)) < 1){
    perror("negotiation_write sendto");
  }
(省略)
```

図 5.15: Negotiation Module: リセット要求の送信

## 5.6 まとめ

本章では，第 4 章で述べた設計に基づいた実装について述べた．次章では実装された本機構を計測，評価を行う．

```
int negotiation_read(struct multi_stream_param *multi_stream_param){
  if(multi_stream_param->negotiation_flag == 1){
    struct sigaction sa_alarm;
    struct itimerval itimer;
  (省略)
    /** Buffering Module 接続関数の呼び出し登録 **/
    memset (&sa_alarm, 0, sizeof(sa_alarm));
    sa_alarm.sa_handler=connect_buffering_module;
    sa_alarm.sa_flags=0;
    if(sigaction(SIGALRM, &sa_alarm, NULL) < 0){
      perror("sigaction");
    }
    /** 呼び出しタイマ計算 **/
    itimer.it_value.tv_sec =
      itimer.it_interval.tv_sec =
      multi_stream_param->broadcast_freq;
    itimer.it_value.tv_usec =
      itimer.it_interval.tv_usec = 0;
    /** 呼び出しタイマ設定 **/
    if(setitimer(ITIMER_REAL, &itimer, NULL) < 0){
      perror("setitimer");
      exit(1);
    }
  (省略)
```

図 5.16: Buffering Module の呼び出しタイマ登録

```
 void connect_buffering_module(){
(省略)
  /*Buffering Module(dummynet)実行のコマンド設定*/
  snprintf(cmd, sizeof(cmd),
    "rsh root@%s ipfw pipe %d config delay %d",
    multi_stream_param.dummynet_addr,
    multi_stream_param.dummynet_pipe_id,
    multi_stream_param.result_delay);

  if((bm = popen(cmd, "r")) == NULL){
    perror("connect buffering module popen:");
    exit(-1);
  }
(省略)
```

図 5.17: Buffering Module: Buffering Module の実行

# 第6章　複数ストリームにおける同期再生機構の評価

本章では，第 5 章において実装された本システムに対する評価について述べる．

## 6.1　本機構の実現した機能

本研究では複数ストリームを同期伝送するための処理遅延を級数する為の機構を構築した．実現した機能は以下の 4 つである．

- 収録・再生環境に対する変更を最小限にするため，DDL と DIL に分類したモジュールを設計・実装した．
- バッファリング機構として，収録・再生機器と独立した帯域制御システムを用いることにより，収録・再生環境に左右されない同期機構を実現した
- ネゴシエーションパケットを用いた端末間時間協調を実現した
- 遅延時間の揺らぎを偶発的なものと慢性的なものに分類するための平均遅延時間の算出機構を設計・実装した

評価として受信端末間における協調同期を用いた同期精度の計測を行った．次に送信端末間における自律分散同期との併用による同期精度の計測を行った．その上で第 2.4 節で示した同期許容時間について述べる．比較実験として同一の実験トポロジにおいて，DVTS を用いた計測を行った．次節で評価環境について述べた後，それぞれの計測結果と考察を述べる．

## 6.2　評価環境

評価に用いた計算機を表 6.1 に示す．本機構を構成する送受信機器，及び Buffering Module 用機器の他に，遅延時間を発生させるための帯域制御用機器を用いた．

本機構の動作する送受信端末では，全てに NTP の設定を行った．本実験で参照した NTP サーバリストを表 6.2 に示す．同期精度向上のために，実験環境から高い同期精度を確保可能であり，且つ Stratum1 に分類される NTP サーバのうち，3 台を参照した [26].

表 6.1: 評価に用いた計算機

| | 送信機器 | 受信機器 | Buffering Module 用帯域制御機器 | 帯域制御用機器 |
|---|---|---|---|---|
| 型番 | 富士通 LOOX T70GN | - | DELL PE1850 | DELL PE1850 |
| CPU | Intel PentiumM 1GHz | Intel Pentium4 2.80GHz | Intel Xeon 3.60GHz | Intel Xeon 3.60GHz |
| メモリ | 512MB | 512MB | 3GB | 3GB |
| OS | debian GNU Linux Kernel 2.6.8-3-686 | debian GNU Linux Kernel 2.6.8-3-686 | FreeBSD 5.4 Release | FreeBSD 5.4 Release |

表 6.2: 参照した NTP サーバ

| NTP Server | Stratum | refid | Synch Distance) |
|---|---|---|---|
| ntp.nc.u-tokyo.ac.jp | 1 | GPS | 0.00240 |
| ntp.mita.keio.ac.jp | 1 | GPS | 0.00043 |
| ntp1.tohoku.ac.jp | 1 | GPS | 0.00018 |

処理遅延時間の計測手法を図 6.1 に示す．NTSC Test Signal Generator[30] のフレームメモリを用いて，白画面及び黒画面を 1 秒ずつ交互に表示したもの (White Raster) を，DV/NTSC コンバータに入力し，DV 伝送を行う．受信端末に接続された DV/NTSC コンバータの出力を 2ch オシロスコープに接続し，再生遅延時間の差を計測した．

次節以降では各実験トポロジについて述べた後，それぞれの計測結果について述べる．

## 6.3 受信端末間における協調同期

複数ストリーム転送時における，同期精度に関する評価を行う．評価環境を図 6.2 に示す．DVTS，及び本機構の持つ，複数アドレスに向けてのユニキャスト機能を用い，単一の送信機から送信された DV データの同期精度について計測を行った．また，途中経路に dummynet を設置し，送信先アドレスによって遅延時間を掛けることで遅延時間を発生させた．以下に計測結果とその考察を述べる．

始めに通常の DVTS を用いた計測結果を述べる．図 6.2 のトポロジにおいて，dummynet による伝送遅延時間を発生させなかった場合を図 6.3 に示す．その結果，2 つのストリームの間に約 5ms ほどの伝送遅延時間差があることが分かった．これは DVTS の実装として，送信先 IP アドレスをリンクドリストとして持ち，ループさせることによってソケットを切り替え，送信を行うために発生すると考えられる．

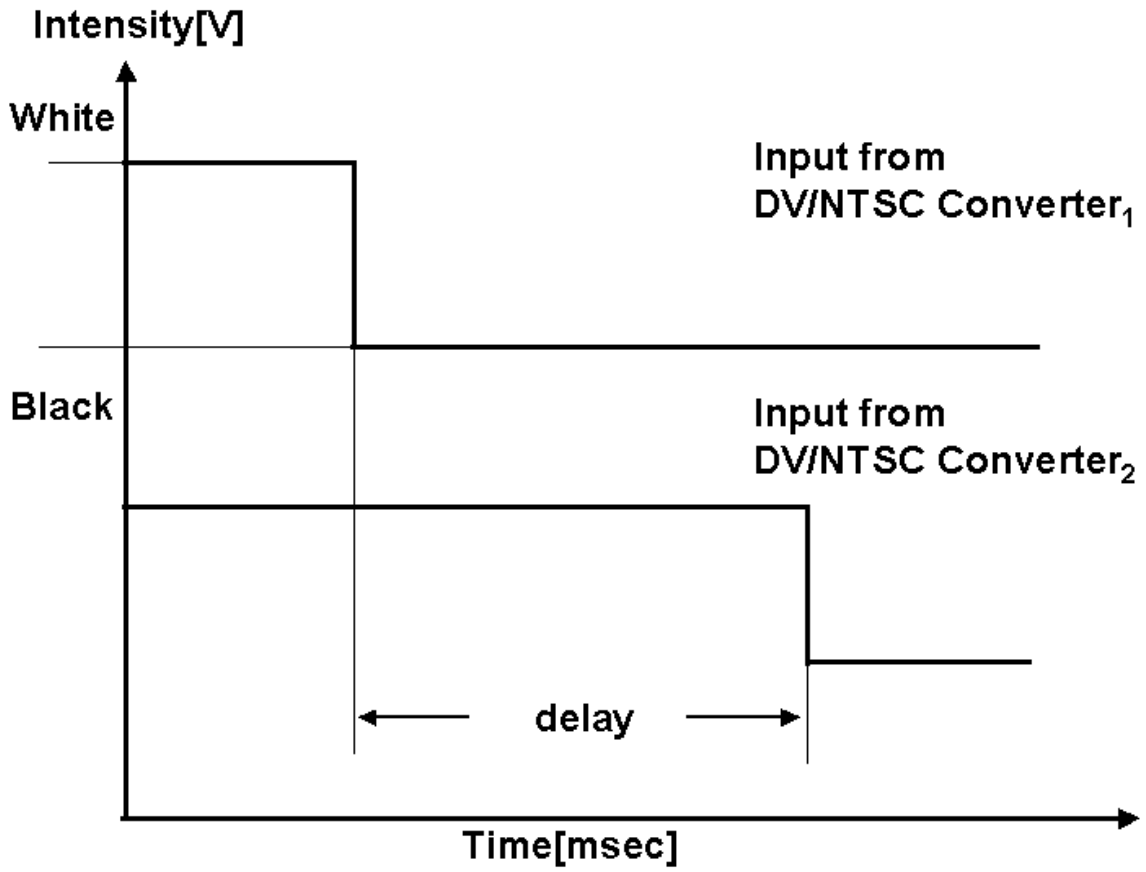


図 6.1: 処理遅延時間の計測

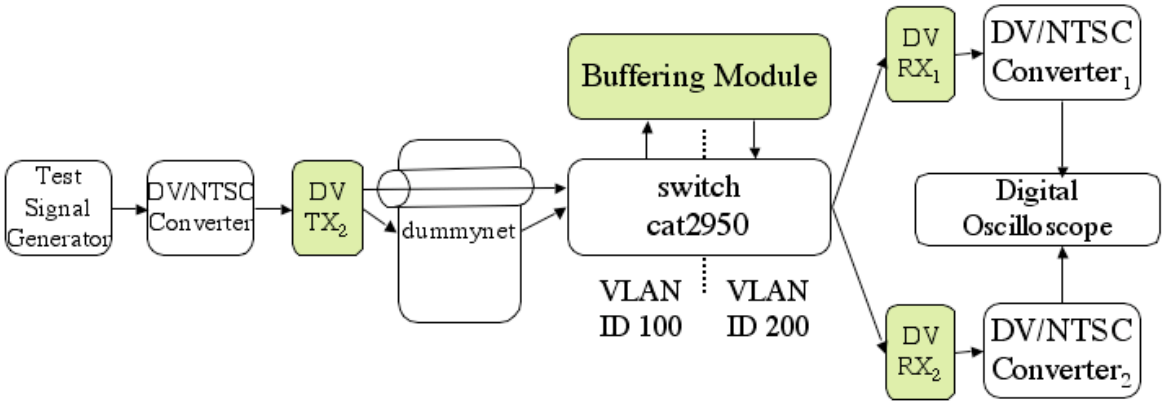


図 6.2: 評価環境：受信側端末間協調同期

通常の DVTS に対し，一方の送信先アドレスに対するストリームにのみ，100ms と 200ms の伝送遅延時間を発生させた場合を図 6.4 に示す．発生した遅延時間だけ，伝送遅延時間

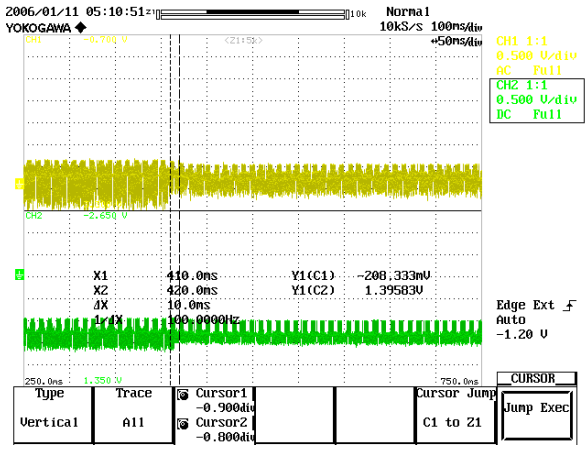


図 6.3: 実験 1:DVTS 送受信 (伝送遅延時間無し)

の差として現れていることが分かる．

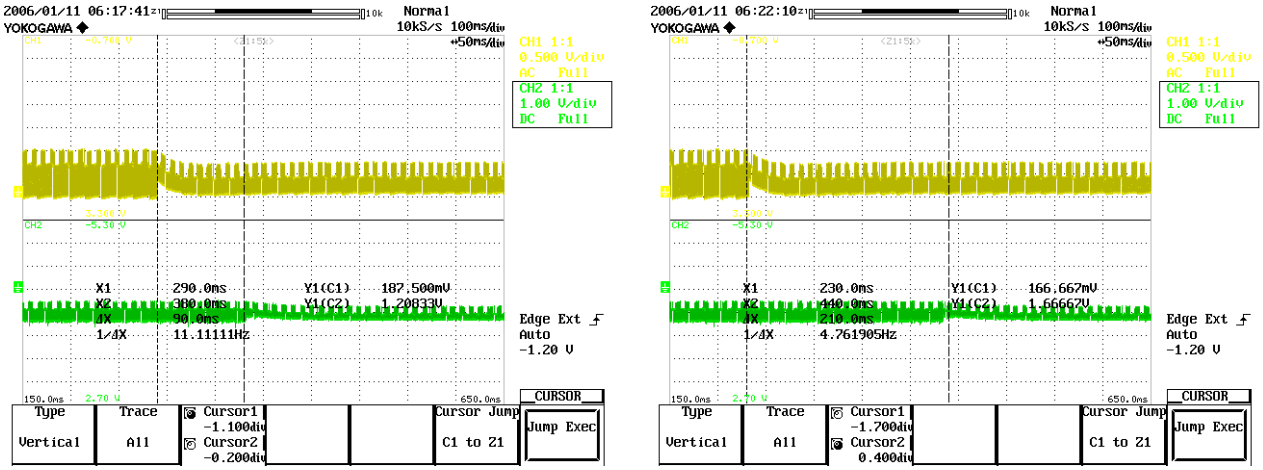


図 6.4: 実験 2,3:DVTS 送受信 伝送遅延時間 100ms(左)，伝送遅延時間 200ms(右)

次に本機構を用いた計測結果について述べる．伝送遅延時間を発生させなかった場合を図 6.5 に示し，Average Counter Module と Negotiation Module の処理が与える影響について述べる．その結果，伝送遅延時間の差は約 5ms であり，通常の DVTS と同等であることが分かる．

100ms の伝送遅延時間が発生した場合の本機構の動作について計測を行う．Buffering Module 適用前の伝送遅延時間の差を図 6.6 左に示す．オシロスコープで計測した遅延時間の差は約 90ms であった．これに対し，Negotiation Module によって算出された遅延時間の差は 104ms であった．この値を基に Buffering Module が動作した際の伝送遅延時間の差を図 6.6 右に示す．Buffering Module 動作後の伝送遅延時間の差は-10ms であり，同

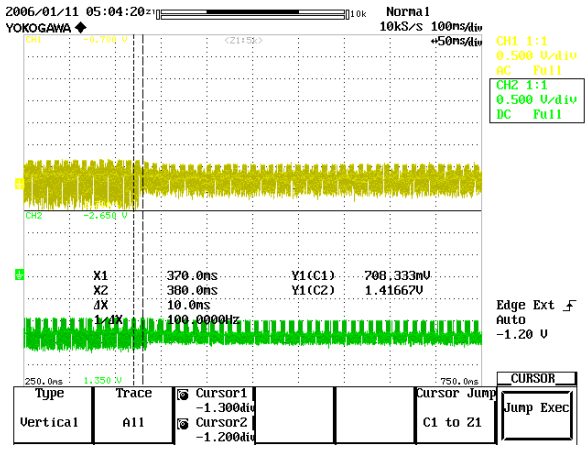


図 6.5: 実験 4:DVTS 送受信 (伝送遅延時間無し)

期許容時間内の同期を達成した.

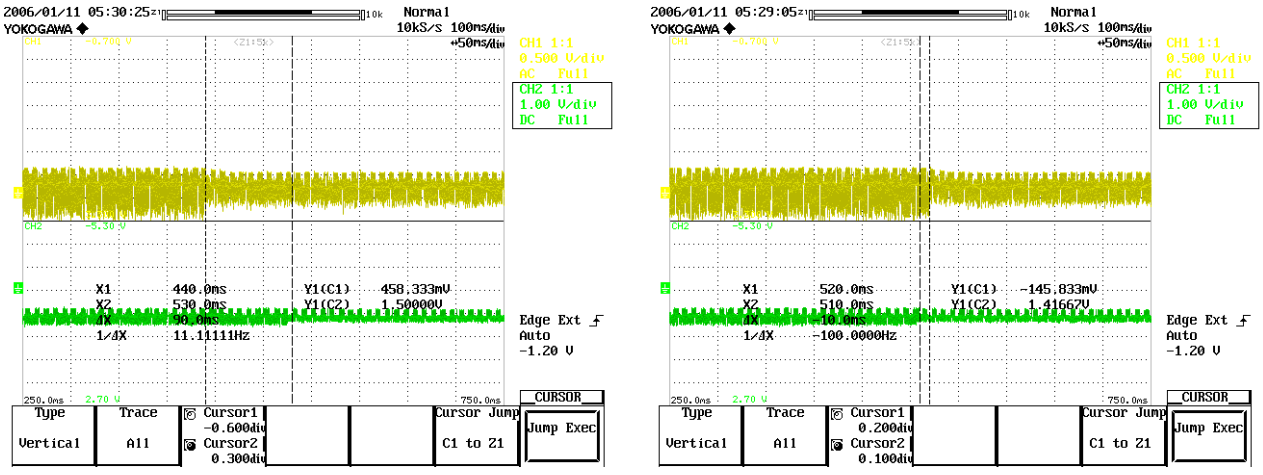


図 6.6: 実験 5:Buffering Module 適用前 (左)，適用後 (右)(遅延時間 100ms)

本機構において 200ms の伝送遅延時間が発生した場合を図 6.7 に示す．オシロスコープで計測した遅延時間の差は約 210ms であった (図 6.7 左)．これに対し，Negotiation Module によって算出された遅延時間の差は 199ms であった．この値を基に Buffering Module が動作した際の伝送遅延時間の差は-10ms であり (図 6.7 右)，同期許容時間を達成した.

## 6.4 送信端末間における自律分散同期

第 6.3 節では，1 対多通信における受信端末間における協調同期の精度について述べた．本節では，第 6.3 節を基に，更に送信機器を分けた多対多通信における同期精度について

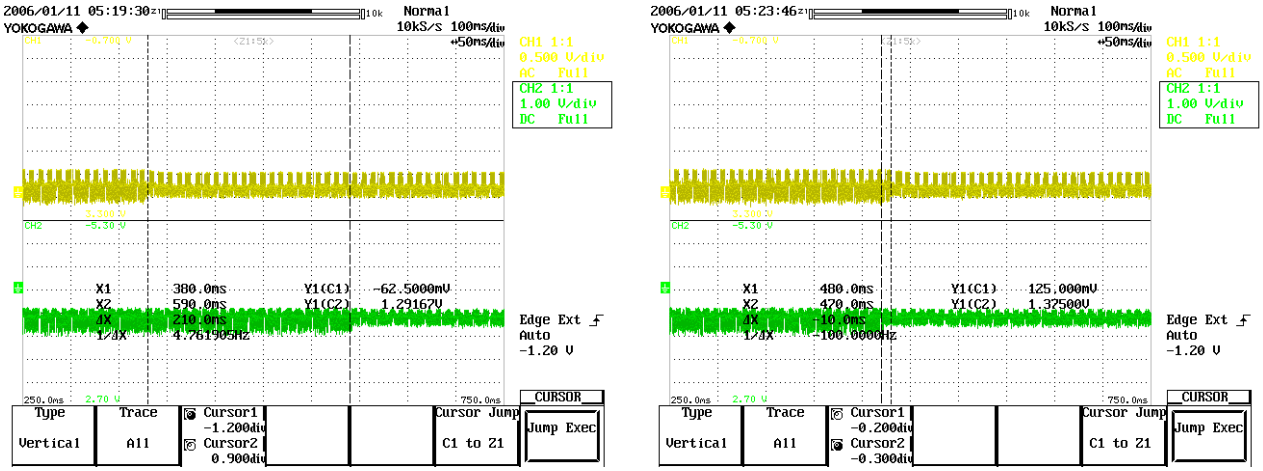


図 6.7: 実験 6:Buffering Module 適用前 (左)，適用後 (右)(遅延時間 200ms)

述べる．

送信端末間における自律分散同期の評価環境を図 6.8 に示す．送信端末とスイッチ間において，1 台の途中経路に伝送遅延時間を発生させるための帯域制御用ブリッジノードを設置した．以下に計測結果とその考察について述べる．

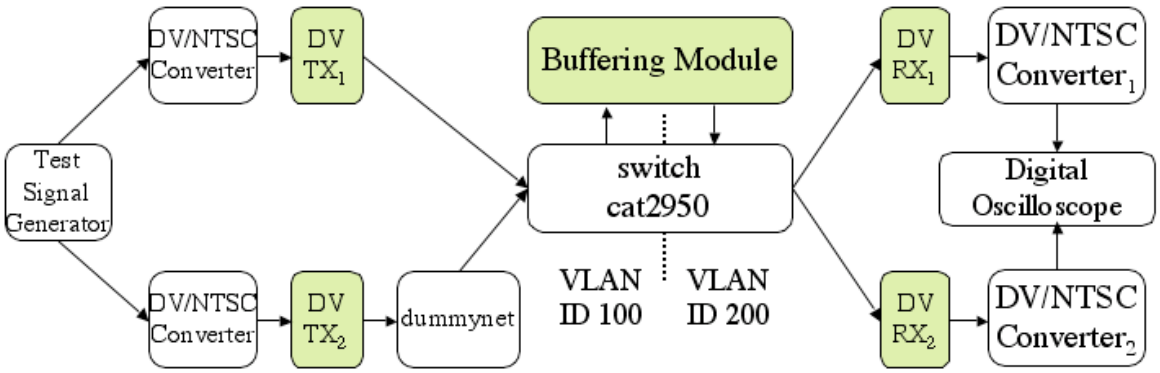


図 6.8: 評価環境：送信側自立分散同期

比較実験として，2 台の DVTS 送信端末のうち，1 台の途中経路にブリッジノードを挟むことによる同期遅延時間への影響について述べる．ブリッジノードを介在させなかった場合の伝送遅延時間の差を図 6.9 に示す．これより，伝送遅延時間の差はほぼ 0 であることが分かる．

次に片方の DVTS 送信端末の中継経路にブリッジノードを設置した場合の伝送遅延時間の差を図 6.10 に示す．これより，ブリッジノードを設置することに寄る伝送遅延時間

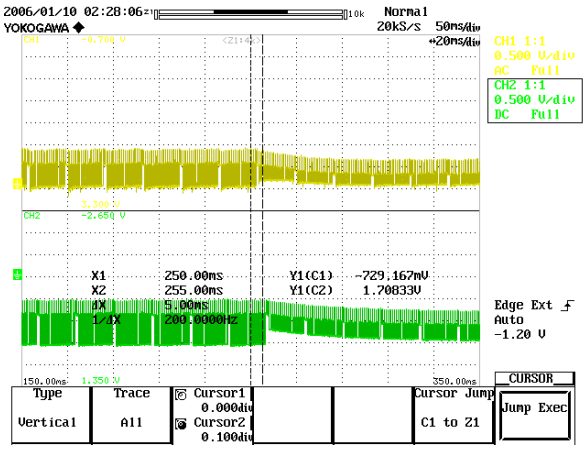


図 6.9: 実験 7:複数送信機による DVTS 送受信

は，約 10ms 追加されることが分かる．

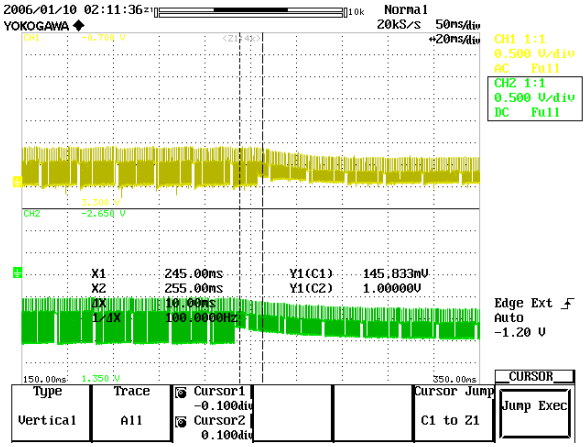


図 6.10: 実験 8:複数送信機による DVTS 送受信 (伝送遅延時間 0ms)

100ms と 200ms の伝送遅延時間を発生させた場合の DVTS の伝送遅延時間の差を図 6.6 に示す．100ms の伝送遅延時間追加時は約 95ms の伝送遅延時間の差が発生した．同様に 200ms の伝送遅延時間追加時は，約 200ms の伝送遅延時間の差が発生した．以上より，追加した伝送遅延時間に等しい伝送遅延時間が追加されていることが分かる．

次に本機構を用いた計測結果について述べる．伝送遅延時間を発生させなかった場合を図 6.12 に示す．Buffering Module 適用前の伝送遅延時間の差は 10ms であった (図 6.12 左)．これに対し，Negotiation Module によって算出された遅延時間の差は約 40ms であった．この値を基に Buffering Module が動作した際の伝送遅延時間の差は-30ms であった (図 6.12 右)．伝送遅延時間の差は本機構の実験 8 に比べ拡大したが，同期許容時間を達成している．

100ms の伝送遅延時間が発生した場合を図 6.13 に示す．Buffering Module 適用前の伝

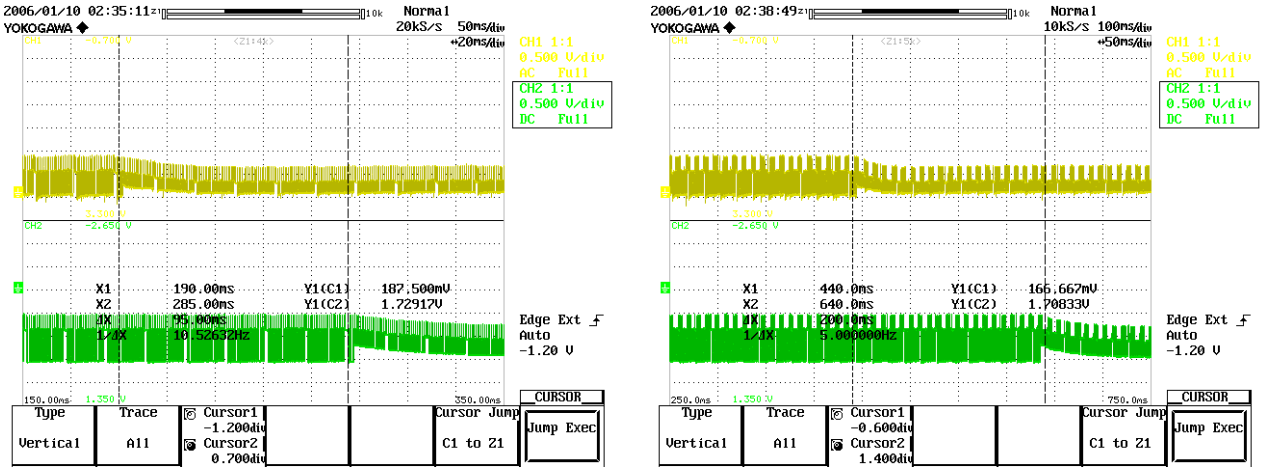


図 6.11: 実験 9,10:複数送信機による DVTS 送受信 伝送遅延時間 100ms(左)，200ms(右)

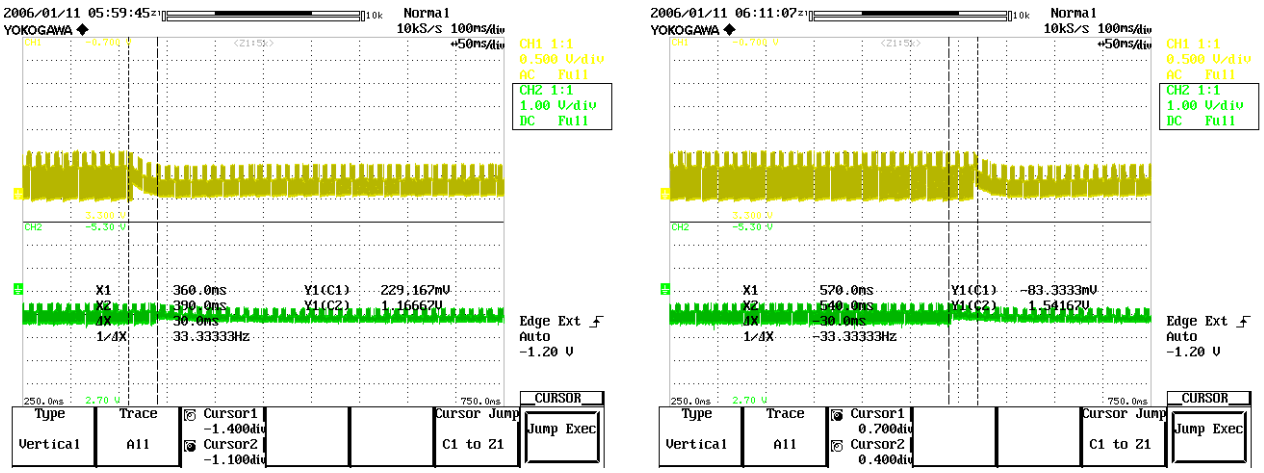


図 6.12: 実験 11:Buffering Module 適用前 (左)，適用後 (右)(遅延時間 0ms)

送遅延時間の差は 90ms であった (図 6.13 左)．これに対し，Negotiation Module の算出した伝送遅延時間の差は約 140ms であった．この値を基に Buffering Module が動作した際の伝送遅延時間の差は-40ms であり (図 6.13 右)，同期許容時間を達成した．

200ms の伝送遅延時間が発生した場合を図 6.14 に示す．Buffering Module 適用前の伝送遅延時間の差は 210ms であった (図 6.14 左)．これに対し，Negotiation Module の算出した伝送遅延時間の差は約 240ms であった．この値を基に Buffering Module が動作した際の伝送遅延時間の差は-20ms であり (図 6.14 右)，同期許容時間を達成した．

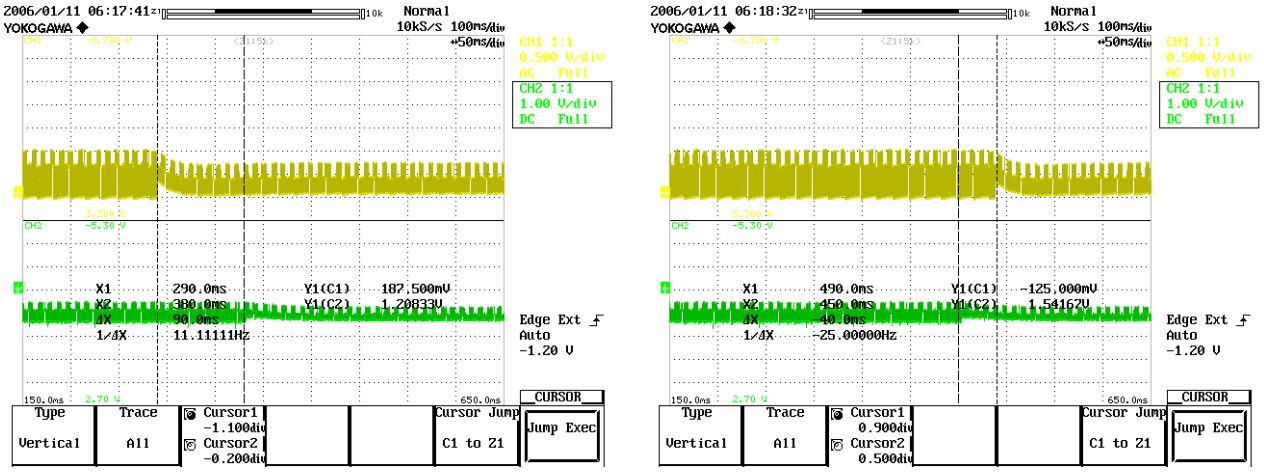


図 6.13: 実験 12:Buffering Module 適用前 (左)，適用後 (右)(遅延時間 100ms)

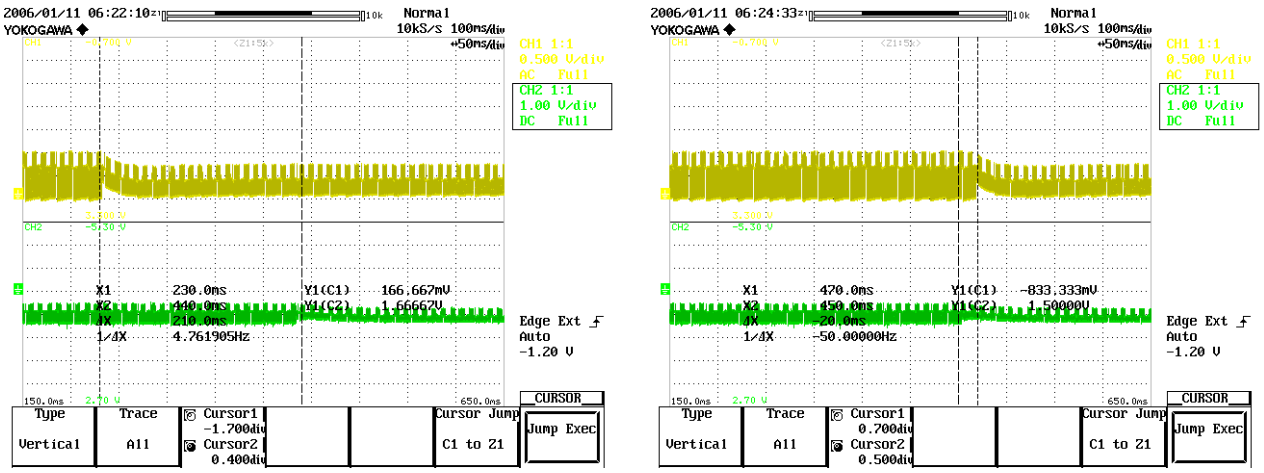


図 6.14: 実験 13:本機構適用前 (左)，適用後 (右)(遅延時間 200ms)

## 6.5 考察

受信端末間における協調同期により，異なる伝送遅延時間を持つストリームに対する許容時間内の同期を実現することができた．また，送信機端末間における自律分散同期と受信端末間における協調同期の併用によって異なる伝送遅延時間を持つ複数ストリームの同期を実現することができた．Average Counter Module に与える伝送遅延時間の許容揺らぎ設定により，更に精度を高めることができる可能性がある．

実験 11 の伝送遅延時間が発生しない環境下での本機構の動作について，実験 8 の通常の DVTS に比べ，伝送遅延時間差の拡大が発生した．本機構では全ての送受信機器が正

確に時刻合わせされていることが前提となる．加えて合わされた時刻を正確に保持し続ける必要がある．本実験では時刻合わせのために NTP を用いた．また，送信機として省電力型 CPU である Pentium M を搭載した機器を用いた．各端末においてより正確な時刻を保持する為に，GPS を用いた時刻合わせ機構や，送受信機端末の CPU アーキテクチャ，OS などの検討を行う必要がある．

## 6.6 まとめ

本章では第 5 章で述べた実装を計測，評価を行った．その結果，第 2.4 節で述べた同期許容時間を満たす複数ストリーム同期機構を実現した．

次章では本機構の発展について述べる．

# 第7章　結論

第5章で本研究の実装に関して述べた．本章では第5章から導きだされた内容より考察，まとめを行う．

## 7.1　まとめ

本研究では，インターネットを媒体とする映像配信環境において収録・再生機器，ネットワーク，映像フォーマットなどといった処理要因の違いによって発生する処理遅延に着目し，複数ストリームを同期転送するための処理遅延吸収機構の構築を行った．本研究を用いることにより，収録・再生環境に関わらず，時間軸を指標とした再生遅延時間の吸収を実現することができる．

複数ストリーム同期再生機構を実現する上で，処理要因によって発生する処理時間の分析を行った．遅延時間の発生部として，1）収録端末から送信端末を経て受信端末までに至る入力部，2）コンテンツフォーマットのデコードやトランスコーディングといった内部処理を変換部，3）受信端末出力から収録機器を経てユーザに対する出力までに至る出力部　に分別を行った．ネットワーク伝送遅延だけでなく，コンテンツフォーマットや収録・再生機器の組み合わせによって再生遅延時間は異なる．

収録・再生環境の多様性を隠蔽し，既存のシステムへの変更を最小限にするために，本機構を DDIL と DDL の 2 つに分類した．コンテンツフォーマットやアプリケーション，OS，収録・再生機器に依存性が高い部分を DDL とし，非依存な部分を DIL と定義した．

機構実現のために，全ての機器において絶対時間の統一を行った上で，以下の3つのモジュールの設計を行った．

- Negotiation Module
  受信側端末間協調を行う．受信したストリーミングに挿入されたコンテンツ時間軸の再生タイミングを共有する．

- Average Counter Module
  伝送遅延時間の計算を行う．

- Buffering Module
  収録・再生機器とは独立して動作をする帯域制御システムと協調をする．これにより収録・再生環境に非依存な再生遅延時間吸収を行う．

本研究の有用性を示すアプリケーションとして，DVTS を例にした実装を行い，評価を行った．

本研究により，複数コンテンツフォーマットによる複数ストリーム同期を時間軸を指標として実現，達成した．

## 7.2 今後の課題

## 7.3 本機構の応用例

本節では本機構の利用による応用例として，インターネット放送におけるザッピング技術とマルチアングルシステムについて述べる．

### 7.3.1 ザッピング

異なるコンテンツフォーマットを汎用的に扱うことのできる本機構の利点として，限られた帯域幅に対するコンテンツ数が挙げられる．単一のフォーマットを用い，個々が独立した複数ストリーム転送を構築する方法と複数のフォーマットを用いて個々が協調した複数ストリーム転送を構築する方法の比較を図 7.1 に示す．有効帯域幅が n Mbps であり，任意のコンテンツフォーマットの消費帯域幅を m Mbps とする．単一のフォーマットを用いた場合に 4 本のストリーム転送が可能であった場合，m/2 Mbps や m/4 Mbps の消費帯域幅に収まるコンテンツフォーマットを持つストリームと併用することで，より多くのコンテンツを送信する事ができる．

収録・再生環境や用途によって柔軟にコンテンツフォーマットを選択することができることによって，より柔軟なストリーミング環境の構築が可能となる．特に有効な例として，インターネット放送におけるザッピング技術がある．

インターネット放送が一般化した際，複数のストリームを選択・変更するチャンネル切り替えに対する要望が高まると考えられる．従来のアナログ地上放送に見られるような，同一時間軸で進行するサムネイル表示や，ザッピング時の時間軸一致などの実現が求められると考える．

ザッピング応用例を図 7.2 に示す．本研究を用いる事により，消費帯域や品質の異なるコンテンツフォーマットを組み合わせる事ができるため，視聴者の嗜好に合わせたザッピング環境の構築ができる．

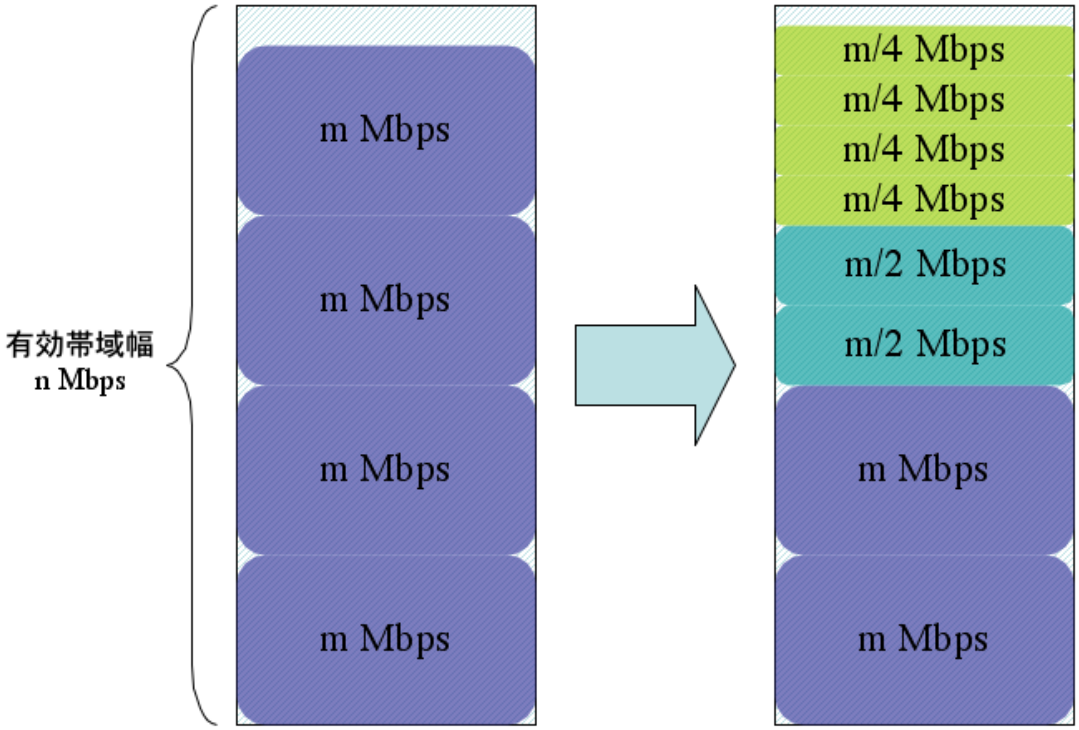


図 7.1: 単一のフォーマットを用いた複数ストリーム転送例 (右) と複数のフォーマットを用いた複数ストリーム転送例 (左)

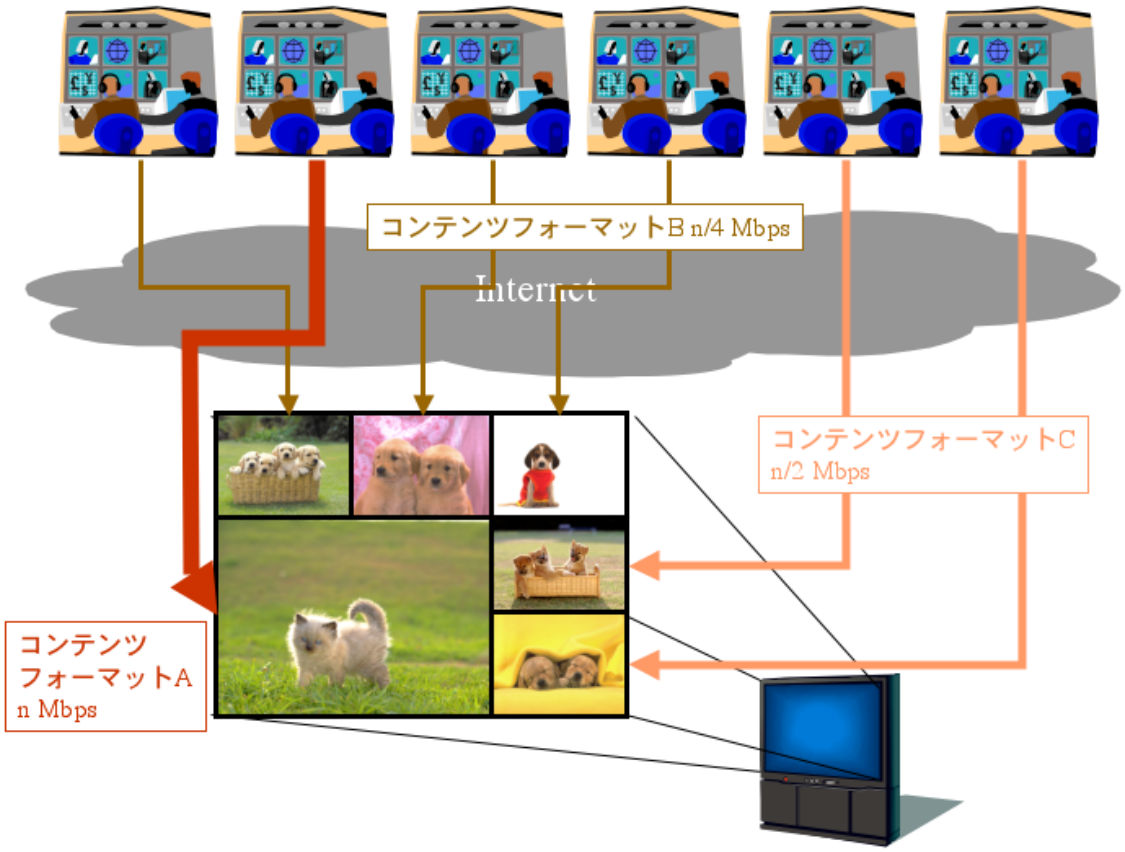


図 7.2: ザッピング応用例

# 謝辞

本論を進めるにあたり，主査である慶應義塾大学環境情報学部教授の村井純博士に感謝致します．また副査である慶應義塾大学環境情報学部助教授の中村修博士，慶應義塾大学環境情報学部の稲蔭正彦教授に感謝致します．

多大なる御指導を頂きました，慶應義塾大学政策・メディア研究科助手の杉浦一徳博士に感謝致します．評価にあたって快く機材貸与をしていただいた慶應義塾大学 Auto ID ラボ・ジャパン副所長三次仁氏，慶應義塾大学制作メディア研究科廣瀬峻氏に感謝致します．日頃より暖かい声援を掛けていただきました慶應義塾大学環境情報学部専任講師の重近範行博士，湧川隆次博士，同政策メディア研究科博士課程の海崎良氏に感謝致します．

多大なる研究生活のバックアップをして頂いた慶應義塾大学制作メディア研究科工藤紀篤氏に感謝致します．研究を進めるにあたって細やかな作業を手伝っていただきました政策メディア研究科修士課程の松園和久氏，環境情報学部遠峰隆史氏には来期以降の素敵な赤入れを感謝の意として差し上げたいと思います．そしてモービル広域ネットワーク (MAUI) プロジェクト及び徳田・村井・中村・楠本・南研究室の諸氏，特に STREAM 研究グループの皆様に感謝致します．

最後に修士論文に取り組んだ期間をはじめとして，私の学生生活を常にサポートして下さった両親，祖父母に感謝致します．

以上を持って謝辞といたします．

# 参考文献


[1] BB Cable Corporation. ブロードバンドケーブルTV BBTV. *http://www.bbtv.com/*, 2005.

[2] NICT. *IPv6* **マルチキャスト技術を放送へ応用** *JGN II* **を利用した広域実証実験を世界に先駆けて開始**, Feb 2005.

[3] digital cinema consortium. デジタルシネマ研究コンソーシアム WWW page. *http://dcc.imgl.sfc.keio.ac.jp*.

[4] *Specifications of Consumer-Use Digital VCRś using 6.3mm magnetic tape*, 1994. HD Digital VCR Conference.

[5] HD DIGITAL VCR CONFERENCE. Specifications of Consumer-Use Digital VCRs PART1 General Specifications of Consumer-Use Digital VCRs. *Specifications of Consumer-Use Digital VCRs using 6.3mm magnetic tape*, pages 1–61, Dec 1994.

[6] HDV Format Co promoters Office. *HDV Information WEB site*, 2004. http://www.hdv-info.org/.

[7] MPEG-2 Generic coding of moving pictures and associated audio information. *http://mpeg.telecomitalialab.com/standards/mpeg-2/mpeg-2.htm*, Oct 2000.

[8] D. LeGall. Mpeg: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, April 1991.

[9] Schwarz T. Schafer R., Wiegan T. *The emerging h.264/avc standard*, 2003.

[10] MPEG-4 Industry Forum. *MPEG-4 The Media Standard*, July 2002.

[11] Real Networks Corporation. Real Networks WWW page. *http://www.real.com*.

[12] J. Postel. *Transmission Control Protocol*, September 1981. RFC 793.

[13] J. Postel. *User Datagram Protocol*, August 1980. RFC 768.

[14] Audio-Video Transport Working Group, H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*, January 1996. RFC 1889.

[15] Inc. Xilinx. Master control switcher -xilinx solutions for the broadcast chain app note. *http://www.xilinx.com*.

[16] Mansour J. Karam and Fouad A. Tobagi. Analysis of the delay and jitter of voice traffic over the internet. In *IEEE Infocom '01: The Conference on Computer Communications, Volume 2: 20th Annual Joint Conference of the IEEE Computer and Communications Societies (IC '01)*, pages 824–833, 2001.

[17] Jean-Chrysostome Bolot, Hugues Crepin, and Andreas Vega Garcia. Analysis of audio packet loss in the internet. In *Network and Operating System Support for Digital Audio and Video*, pages 154–165, 1995.

[18] A.Ogawa. *DVTS (Digital Video Transport System) WWW page*, November 2001. http://www.sfc.wide.ad.jp/DVTS/.

[19] 小川 晃通. 協調的輻輳制御を用いた映像配信機構の設計と実装. 1999.

[20] SONY Corporation. SONY SEU-TL100 WWW page. *http://www.sony.co.jp/sd/ProductsPark/Professional/LINKUNIT/seu_tl100/index.html*, 1999.

[21] 町澤朗彦, 杉浦一徳, 小峰隆弘, 植月修宏, 岡沢治夫, JinMin Chun, and 中川晋一. ATM による DV 伝送システムの遅延と品質について. *DICOMO2000* **シンポジウム**, pages p355–360, Jun 2000.

[22] P. Enge and P. Misra. Special issue on gps: The global positioning system. In *Proceedings of the IEEE*, pages 3–172, January 1999.

[23] US Naval Observatory. GPS Timing Data & Information. *http://tycho.usno.navy.mil/gps_datafiles.html*, 2005.

[24] D. Mills. *Network Time Protocol (Version 3) Specification, Implementation*, March 1992. RFC 1305.

[25] A. Pasztor and D. Veitch. Pc based precision timing without gps, 2002.

[26] 伊藤大輔, 山下聡, 三宅延久, and 外山勝保. インターネット上の高精度な時刻配信サーバの運用. **情報科学技術フォーラム**, Sep 2002.

[27] Riccardo Gusella and Stefano Zatti. The berkeley unix 4.3bsd time synchronization protocol. Technical Report UCB/CSD-85-250, EECS Department, University of California, Berkeley, 1985.

[28] NHK 放送技術研究所. NHK 技研だより 技術動向 長野オリンピックの新機材. Arg. 1998.

[29] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997.

[30] Leader Instruments Corporation. 411 ntsc test signal generator www page. *http://www.leaderusa.com/411.htm*, 2003.