修士論文　　2005 年度 (平成 17 年度)

環境資源に適応する
自律協調型メディア配信システムの設計と構築

慶應義塾大学 大学院 政策・メディア研究科

氏名：三島 和宏

指導教員

主査：村井 純　(慶應義塾大学 大学院 政策・メディア研究科)
副査：中村 修　(慶應義塾大学 大学院 政策・メディア研究科)
副査：徳田 英幸 (慶應義塾大学 大学院 政策・メディア研究科)

平成 18 年 2 月 12 日

修士論文要旨 - 2005 年度 (平成 17 年度)

環境資源に適応する自律協調型メディア配信システムの設計と構築

本研究では，プロセッサ，メモリなどの計算機資源，ならびにネットワーク資源といったインターネットを介したメディア配信を取り巻く環境資源に適応する自律協調型メディア配信システムを実現した．

インターネットを介したメディア配信では，1) ネットワーク環境，2) 計算機環境，3) メディアフォーマットに対する判別・考慮が必要不可欠である．ネットワークインフラストラクチャの発達によりネットワーク環境が発達し，プロセッサやメモリ性能の向上により計算機環境が発達した．これにより，ユーザ間で得られる環境資源に格差が生じるようになった．インターネットを介したメディア配信ではこれらの状態変化や格差への考慮が必要となる．

本研究では，インターネットを介したメディア配信において， 1) ネットワーク環境, 2) 計算機環境 の 2 つの状態を判別し，対応可能な処理を行う自律協調型メディア配信を実現した．また，本手法の実現のためにネットワーク資源ならびに計算機資源といった環境資源を常時把握し，指標化するシステムを開発した．

自律協調性を確保するために AMSS (Adaptive Media Streaming System) の設計・実装を行った．AMSS は，1) ノードのネットワーク的位置特定, 2) 環境資源の把握と指標化, 3) ノード間ネゴシエーション, 4) 自律協調型配信制御 の 4 つの機能を有する．

自律協調型メディア配信の有効性評価のために，既存の DVTS に対して AMSS を実装した．AMSS を実装した DVTS によって環境資源の変動の検知ならびに状況に応じた配信制御が実現できた．計算機とネットワークの各資源の変化に応じて行うべき制御を自律的に再設定することで，状況に応じた有効な制御をシステム全体で実現する．

本手法を用いることにより，ネットワークならびに計算機のリソースをユーザが意識することなく，自律的に状況に適したメディアフォーマット選択を可能とする映像・音声配信環境が実現する．

キーワード
1. 計算機資源，2. ネットワーク資源，3. 伝送特性，4. メディア配信，5. DVTS

慶應義塾大学 大学院 政策・メディア研究科
三島 和宏

Abstract of Master's Thesis - Academic Year 2005

Design and Implementation of Adaptable Media Streaming Architecture considering Environmental Characteristics

In this research, autonomous collaborative media streaming system using the Internet infrastructure is developed considering computational resources such as processor, memory, and network resource required for media transport.

Media transport system using the Internet requires to negotiate 1) Network Environment, 2) Computational Environment, and 3) Consideration of media format. Development of network infrastructure revolutionized the network application infrastructure and computational power such as processor speed and memory access speed. Wide range of availability in network and computational power to the users results in complex negotiations of system definition. Recognition scheme of environmental status or the system definition negotiation in media streaming is required to the next generation internet streaming application.

This research developed the recognition scheme based on 1) Network Environment and 2) Computational Environment adaptable to media streaming. To realize the recognition mechanism of environmental information of the application, network and computational environment condition is negotiated between applications for the seamless collaboration.

AMSS (Adaptive Media Streaming System) is designed and implemented to support autonomous and collaborative system. AMSS equips 4 mechanisms. 1) Network recognition mechanism within the node, 2) Environmental information recognition mechanism, 3) Negotiation mechanism within the node, and 4) Autonomous and collaborative streaming control. AMSS has been implemented to the traditional DVTS for evaluation of recognition and collaborative scheme.

AMSS embedded DVTS successfully negotiated the environmental information within the node resulting autonomous and collaborative streaming adjusting mechanism. By enhancing this mechanism, users does not require to negotiate between the end application for suitable environmental information.

Keywords :
1. Computer Resource, 2. Network Resource, 3. Transport Characteristic, 4. Media Streaming, 5. DVTS

Keio University, Graduate School of Media and Governance
Kazuhiro Mishima

目 次

第 1 章　序論 **1**
1.1　メディア配信を取り巻く環境の変化 1
1.1.1　インフラストラクチャの変化 1
1.1.2　メディアの変化 1
1.1.3　システムの変化 3
1.2　メディア配信の実現要項 4
1.3　本研究の目的 6
1.4　本論文の構成 6

第 2 章　インターネットにおけるメディア配信技術 **7**
2.1　ネットワーク環境に対する配慮 7
2.1.1　ストリーミング再生方式とジッタの揺らぎ 8
2.1.2　ジッタへの対応 8
2.1.3　インターネットにおける伝送特性 9
2.2　伝送フォーマットに対する考慮 10
2.2.1　RTP 10
2.2.2　RTCP 10
2.3　メディアタイプに対する考慮 10
2.3.1　SDP を用いたセッション情報記述 11
2.3.2　SAP を用いたセッション情報配信 11
2.3.3　SIP を用いた呼制御 12
2.4　システム環境に対する考慮 13

第 3 章　既存技術とその問題点 **15**
3.1　メディアセッション情報記述プロトコル 15
3.1.1　SDP 15
3.1.2　H.245 16
3.1.3　MPEG21+MPEG7 16
3.1.4　メディアセッション情報記述プロトコルに関するまとめ 17
3.2　メディア配信アプリケーション 17
3.2.1　DVTS 18
3.2.2　Windows Media Player 19
3.2.3　VLC (VideoLAN Client) 21
3.2.4　メディア配信アプリケーションに関するまとめ 23
3.3　まとめ 23

第 4 章　環境資源に適応する自律協調型メディア配信手法の提案　24
4.1 自律協調型メディア配信 24
4.2 環境資源の把握と制御 24
4.2.1 計算機資源量を用いた計算機状態把握と配信制御 24
4.2.2 伝送特性を用いた動的レート制御 28
4.3 本研究のアプローチ 31
4.3.1 複数フォーマットへの対応とフォーマットスイッチング 31
4.3.2 統一的な資源の指標化と状態変化の把握 32
4.3.3 ノード間での交換と共有 33
4.4 まとめ 34

第 5 章　自律協調型メディア配信機構の設計　35
5.1 設計概要 35
5.2 複数フォーマットへの対応 36
5.3 統一的な資源の指標化 36
5.3.1 プロセッサ資源の指標化 36
5.3.2 メモリ資源の指標化 37
5.3.3 ネットワーク資源の指標化 38
5.3.4 資源の記述手法 40
5.4 ノード間での交換と共有 41
5.4.1 リクエスト 41
5.4.2 レスポンス 42
5.4.3 各リクエストメソッド 43
5.5 メディア配信システムへの適用 45
5.5.1 全体構成 45
5.5.2 動作概要 46
5.5.3 フォーマットに依存した制御処理 49
5.6 まとめ 50

第 6 章　自律協調型メディア配信機構の実装　51
6.1 実装概要 51
6.1.1 実装環境 51
6.1.2 実装アプリケーション 52
6.2 アプリケーションのクラス一覧と処理の流れ 53
6.2.1 送信ノードでの処理の流れ 54
6.2.2 受信ノードでの処理の流れ 54
6.3 各モジュールの実装 54
6.3.1 RTP/IP パケットカプセル化・脱カプセル化モジュール 54
6.3.2 システム資源計測モジュール 56
6.3.3 ネットワーク資源計測モジュール 58
6.3.4 リクエスト・レスポンス制御モジュール 60
6.4 まとめ 70

第 7 章　評価 **71**
7.1 評価概要 71
7.2 本機構の実現した機能 71
7.3 実験 1：状況に適応したフォーマットの選択 72
7.3.1 実験手順と仮説 72
7.3.2 実験結果 72
7.3.3 考察 73
7.4 実験 2：CHGFMT リクエストの動作 74
7.4.1 実験手順と仮説 74
7.4.2 実験結果と考察 76
7.5 実験 3：CHGFMT リクエスト・CHGCFG リクエストの動作 78
7.5.1 実験手順と仮説 78
7.5.2 実験結果と考察 80
7.6 まとめ 83

第 8 章　結論 **84**
8.1 今後の課題 85
8.1.1 さらなる多フォーマット化への課題 85
8.1.2 他オペレーティングシステムへの考慮 85
8.2 将来的展望：インターネットを介した統合的なメディア配信環境を目指して . . 86

付 録 A RTP **91**
A.1 RTP(Real-time Transport Protocol) 91
A.2 RTCP(Real-time Transport Control Protocol) 93

付 録 B AMSS (Adaptive Media Streaming System) **98**
B.1 Windows オペレーティングシステムにおける AMSS の特徴 98
B.2 DirectShow テクノロジ 98
B.3 DirectShow と COM 98
B.4 AMSS と DirectShow/COM 99
B.5 DV/RTP フィルタ 100
B.5.1 DV/RTP パケットの配信に用いるフィルタ 100
B.5.2 DV/RTP フィルタ 100
B.6 HDV/RTP フィルタ 101
B.6.1 HDV/RTP パケットの配信に用いるフィルタ 101
B.6.2 HDV/RTP フィルタ 102
B.7 J2K/RTP フィルタ 102
B.7.1 J2K/RTP パケットの配信に用いるフィルタ 102
B.7.2 Motion-JPEG2000/RTP フィルタ 103
B.8 Flea/RTP フィルタ 103
B.8.1 非圧縮ビットマップ/RTP パケットの配信に用いるフィルタ 103
B.8.2 Flea/RTP フィルタ 104

図 目 次

1.1 ネットワークの帯域変化 2
1.2 計算機における bogomips 値の変化 3
1.3 メディア配信のモデル 5
1.4 最適でない映像伝送の例 6

2.1 ダウンロード再生方式 7
2.2 ストリーミング再生方式 7
2.3 正常な再生処理 8
2.4 ジッタの発生と再生処理 8
2.5 SDP 記述の例 11
2.6 SAP によるセッション情報のマルチキャスト配信 11
2.7 SIP による呼制御 12
2.8 メディア配信のモデル 13

3.1 H.245 による Capability Exchange 16
3.2 DVTS for Windows 18
3.3 Windows Media Player 10 20
3.4 VideoLAN Client 21

4.1 利用した配信環境 25
4.2 DV フォーマットでの資源使用率変化 26
4.3 MPEG2-TS フォーマットでの資源使用率変化 27
4.4 非圧縮ビットマップフォーマットでの資源使用率変化 27
4.5 テスト時の環境概要図 29
4.6 輻輳時のジッタとパケットロス - DV ストリーム と Netperf 30
4.7 輻輳時のジッタとパケットロス - DV ストリーム 2 本 30
4.8 レート制御の流れ 31

5.1 全体概要図 35
5.2 拡張 SDP メッセージ 40
5.3 リクエストメソッド 41
5.4 拡張 SDP メッセージ 42
5.5 レスポンスメッセージ 43
5.6 ノードリスト 44
5.7 各モジュール間関係概要 46
5.8 リクエスト・レスポンスの流れ 47

5.9 送信ノードでの処理のモデル 48
5.10 受信ノードでの処理のモデル 49

6.1 AMSS スクリーンショット 52
6.2 送信ノードにおける処理の流れ 54
6.3 受信ノードにおける処理の流れ 55
6.4 $C*RTPSendFilter::SetDestination$ 関数の実装 56
6.5 $C*RTPRecvFilter::SetPortNumber$ 関数の実装 56
6.6 $C*RTPSendFilter::SetFrameRate$ 関数の実装 57
6.7 $CAmssServerDlg::OnInitDialog$ 関数における初期化処理 58
6.8 $CAmssServerDlg:OnTimer$ 関数における資源値取得 59
6.9 パケットロス値の取得処理 60
6.10 伝搬遅延時間の取得処理 60
6.11 $AMSSResourceDesc$ 構造体 61
6.12 $AMSSNetCharDesc$ 構造体 61
6.13 $AMSSRequestData$ 構造体 61
6.14 $CAmssServerDlg::SubmitRequest$ 関数におけるリクエスト発行処理 62
6.15 $CAmssServerDlg::SubmitRequest$ 関数におけるリクエストヘッダ生成 . . . 63
6.16 $CAmssServerDlg::SubmitRequest$ 関数におけるリクエストヘッダ生成 . . . 64
6.17 $CAmssServerDlg::SubmitRequest$ 関数におけるリクエストヘッダ生成 . . . 64
6.18 $CAmssServerDlg::SubmitRequest$ 関数におけるレスポンス処理 65
6.19 $CAmssRRServer::ParseRequest$ 関数におけるレスポンス作成処理 66
6.20 $CAmssRRServer::ParseRequest$ 関数におけるリクエスト判別処理 67
6.21 $CAmssRRServer::ParseRequest$ 関数における SDP 読み込み処理 68
6.22 $CAmssRRServer::ParseRequest$ 関数におけるレスポンス作成処理 69
6.23 AMSS スクリーンショット - レポート 69
6.24 $CAmssServerDlg:OnTimer$ 関数におけるレポート処理 70

7.1 実験 1：実験環境 72
7.2 受信ノードでの動作ログ 73
7.3 送信ノードでの動作ログ 73
7.4 実験 2：実験環境 74
7.5 実験 2 のタイミング 75
7.6 受信ノードでの動作ログ 76
7.7 送信ノードでの動作ログ 76
7.8 実験 2 の結果 77
7.9 実験 3：実験環境 78
7.10 実験 3 のタイミング 79
7.11 受信ノードでの動作ログ 80
7.12 送信ノードでの動作ログ 81
7.13 実験 3 の結果 82

8.1 統合的なメディア配信環境 86

A.1 RTP パケットフォーマット 92
A.2 RTCP SR パケットフォーマット 94
A.3 RTCP RR パケットフォーマット 95

B.1 COM オブジェクトの例 99
B.2 送信ノードでの処理のモデル (DVTS) 99
B.3 受信ノードでの処理のモデル (DVTS) 100
B.4 送信フィルタの持つ ID 101
B.5 受信フィルタの持つ ID 101
B.6 送信フィルタの持つ ID 102
B.7 受信フィルタの持つ ID 102
B.8 送信フィルタの持つ ID 103
B.9 受信フィルタの持つ ID 103
B.10 送信フィルタの持つ ID 104
B.11 受信フィルタの持つ ID 104

表 目 次

1.1 インターネット放送局の種類と実例 2

3.1 メディア情報記述手法のまとめ 17
3.2 メディア配信システムのまとめ 23

4.1 利用した計算機 25

5.1 プロセッサ種別による資源定数一覧 (E_p) 37
5.2 メディアフォーマットによるフォーマット定数一覧 (F_p) 37
5.3 プロセッサ種別による資源定数一覧 (E_m) 38
5.4 メディアフォーマットによるフォーマット定数一覧 (F_m) 39
5.5 メディアフォーマットによるフォーマット定数一覧 (F_n) 39
5.6 リクエストメソッド一覧 41
5.7 応答 (レスポンス) コード一覧 43

6.1 実装ソフトウェア環境 52
6.2 実装ハードウェア環境 52
6.3 パフォーマンスオブジェクト 57
6.4 リクエストタイプ一覧 62

7.1 実験 1 に用いた計算機環境 72
7.2 実験 2 に用いた計算機環境 74
7.3 実験 3 に用いた計算機環境 78

第1章　序論

本章では，本研究の背景としてインターネットインフラストラクチャの広帯域化と，それに伴うメディア配信の現状，本研究の目的について述べる．

1.1　メディア配信を取り巻く環境の変化

インターネットを介したメディア配信を取り巻く環境は，以下の 3 点に挙げる変化をしている．これらの変化に伴い，映像・音声の配信が企業・個人レベルにかかわらず広く行われるようになった．

1. インフラストラクチャの変化
2. メディアの変化
3. システムの変化

1.1.1　インフラストラクチャの変化

インターネットインフラストラクチャの発達により，バックボーンネットワークでは，図 1.1に示すように 10Gigabit Ethernet(IEEE802.3ae[1]) や Gigabit Ethernet (IEEE802.3ab[2]) をはじめとする広帯域データリンクが普及しつつある．また，個人が利用可能なネットワーク環境も，FTTH(Fiber To The Home) などのサービスにより広帯域化が進んでいる．

エンドユーザが得られるネットワーク環境の多様化が進んでおり，100Mbps を超える光ファイバサービス，8Mbps 程度の電話回線を用いる ADSL サービスなど多種多様なサービスが選択可能である．ユーザによって得られるネットワーク帯域幅は，利用するサービスにより異なるため，ユーザ間で差が生じている．

1.1.2　メディアの変化

近年，様々な種類のメディア配信システムが開発され，利用するシステムを選択できるようになった．映像・音声のインターネットを介した配信が容易となり，企業レベルだけでなく，個人レベルでも様々な配信が行われている．

メディア配信の特徴は，利用するメディアフォーマットによって異なる．メディアフォーマットは，インターネットを介したメディア配信に用いたり，計算機上で映像や音声の圧縮・伸張に用いられる．以下にメディア配信に用いられるメディアフォーマットの一例を挙げる．

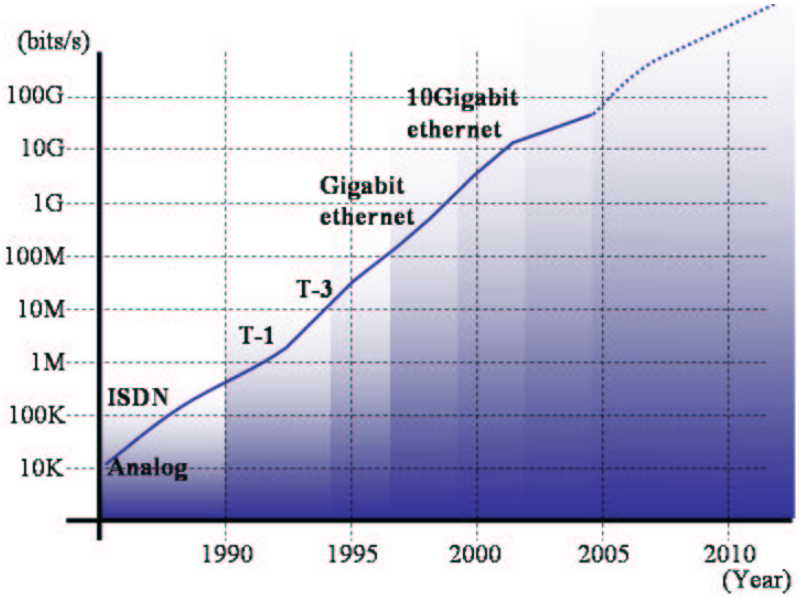

図 1.1: ネットワークの帯域変化

表 1.1: インターネット放送局の種類と実例

種類	実例
テレビ・ラジオ局によるニュース配信	TBS[3], 文化放送 [4], コミュニティ局 [5] など
企業によるインターネット放送局	Impress TV[6], 企業による広告 など
地域ベースのインターネット放送局	千葉県 [7], 茨城県 [8] など
個人ベースでのインターネット放送局	Triumphal Records, Boogie Woogie Cafe[9] など
教育機関での利用	SoI[10], SFC-GC[11] など

- DV (Digital Video)[12]
  DV フォーマットは民生用デジタル VCR のために規格化されたメディアフォーマットである．フレーム内圧縮を用い，フレーム単体は JPEG 圧縮される．解像度は，720×480 ピクセル固定であり，フレームレートも 29.97fps で固定であるため，送出されるデータ量は一定である．データは IEEE1394 バスを通じて扱う．

- HDV (Hi-definition Digital Video)[13]
  HDV フォーマットは，DV フォーマットの高品位版と言えるメディアフォーマットで，キヤノン，ソニー，シャープ，ビクターによって規格化された．フレーム間圧縮を用い，データは MPEG2[14] で圧縮される．解像度は 1400×720(720p) と 1920×1080(1080i) がある．HDV は MPEG2 圧縮を用いることにより，DV フォーマットと同様のデータ量で HDTV 品質の映像を扱うことができる．データは IEEE1394 バスを通じて MPEG2-TS 形式のパケットで扱う．

- H.264/AVC (MPEG4)[15]
  H.264/AVC(Advanced Video Coding) フォーマットは，ITU-T の Video Coding Ex-

perts Group によって策定されたメディアフォーマットであり，フレーム間圧縮を行う．H.264/AVC は，MPEG と共同で規格化されたため，H.264/MPEG-4 AVC とも呼ばれる．圧縮アルゴリズムに対して複数の改良を加え，従来の MPEG2 方式の 10 倍以上の圧縮率を実現する．HDTV などの高品位動画像の圧縮に用いられ，従来より高い計算機の処理能力を必要とする．

- Motion-JPEG2000
  Motion-JPEG2000 は JPEG2000[16] 形式の画像を動画像として扱うためのメディアフォーマットである．フレーム内圧縮を用い，JPEG 圧縮と比較してより高い圧縮率を実現する．JPEG 圧縮に存在するブロックノイズの発生がなく，可逆圧縮にも対応している．JPEG2000 形式はこれまでの JPEG 形式とは互換性がなく，従来より高い計算機の処理能力を必要とする．

1.1.3 システムの変化

高品質な動画像配信には，豊富な計算機資源を持つ計算機が必要である．個人レベルのパーソナルコンピュータにおいても機器の高速化により高度な処理が可能となった．その一方で，それぞれのユーザによって計算機の性能に大きな差が生じている．あるユーザでは利用可能なメディア配信システムでも，別のユーザでは計算機環境が異なるため利用できないことがある．

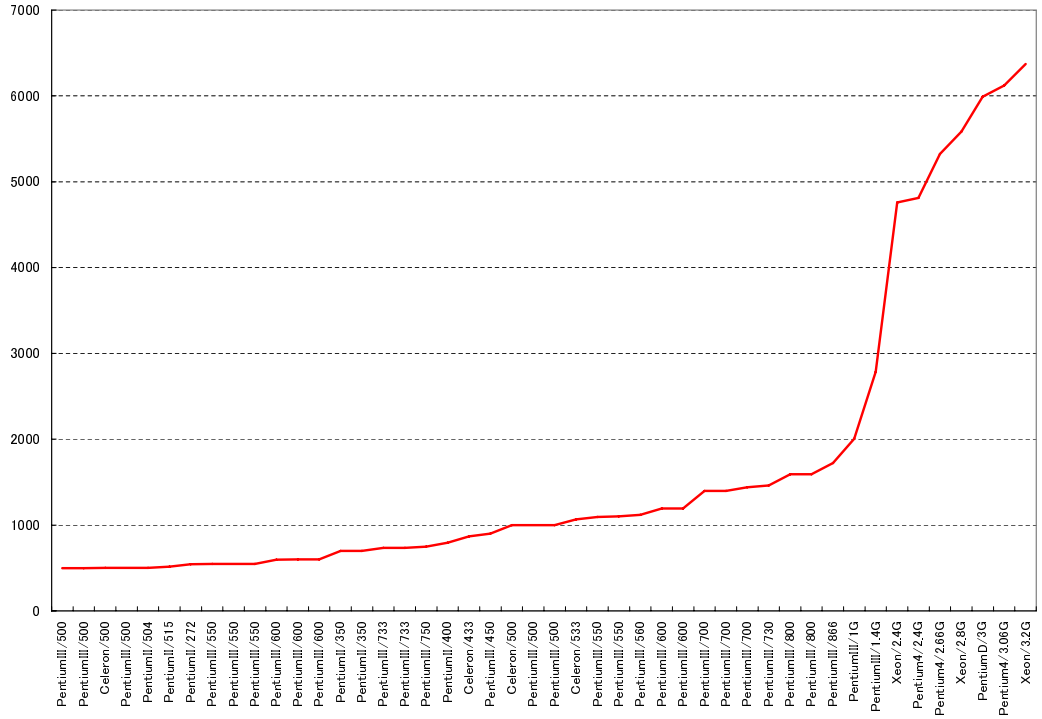

図 1.2: 計算機における bogomips 値の変化

メディア配信において使用する計算機資源は複数存在し，メディアフォーマットを利用中はフォーマットの必要とする量を確保する必要がある．

- プロセッサ資源
  プロセッサにおける処理資源をプロセッサ資源と定義する．メディア配信では特にメディアフォーマットのエンコーディング (送信側)・デコーディング (受信側) のために処理能力の高さが要求される．また，計算機上のすべての処理を司る機能としても必要となるため，他のプロセスとの関係により利用可能な資源量は変化する．

- 描画資源
  グラフィックスカードにおいて画面描画を行うための資源を描画資源と定義する．メディア配信では受信したデータを画面上で表示 (再生) するために必要となる．近年のメディア配信の高解像度化により，描画能力の高いシステムが要求される．

- メモリ資源
  一次記憶資源をメモリ資源と定義する．メディア配信ではアプリケーションの実行作業領域，送受信するメディアデータの一時保存領域 (バッファ) として必要となる．

- 記録資源
  ハードディスクなどの二次記憶資源を記録資源と定義する．VoD のようなディスクに書き出しを行い，二次的に再配信を行う場合に必要となる．近年のメディア配信の高解像度化により，メディアデータのサイズは飛躍的に増加しており，データを保存するための膨大な領域と読み出しのための高速なアクセス速度が必要となる．

1.2 メディア配信の実現要項

インターネットを介したメディア配信には，必ず送信ノードと受信ノードが存在する．送信ノードならびに受信ノードでは，以下の 4 つの条件が満たされない場合，映像・音声を送受信・再生できない．

- 送信ノード・受信ノード双方の存在するネットワーク的な位置が特定されていること (双方アドレスの把握)
- 送信ノード・受信ノードで利用するメディアフォーマットが同一であること (フォーマットの同一性)
- 送受信ノード間で利用可能なネットワーク環境が利用するフォーマットの要求を満たすこと (ネットワーク資源の確保)
- 送受信ノード双方で利用可能な計算機環境が利用するフォーマットを処理するための資源要求を満たすこと (計算機資源の確保)

インターネットを介したメディア配信は，データの適切な配信が行われることと，ネットワーク資源の有効に活用した配信状態を維持するシステム設計が必要である．ユーザが利用可能な資源は送信ノードの持つ入力デバイスから受信ノードの持つ出力デバイスまでの以下の資源を含む．図 1.3に各資源とメディア配信の関係を示す．

- 計算機資源
  - 入力デバイス
    ビデオ入力デバイス (フォーマット，解像度，フレームレートなど)，オーディオ入力デバイス (入力形式，ビットレート，チャネル数など)
  - 出力デバイス
    ビデオ出力デバイス (フォーマット，解像度，リフレッシュレートなど)，サウンド出力デバイス (出力形式，出力可能音質など)
  - 処理デバイス
    プロセッサ (周波数，アーキテクチャなど)，メモリ (容量，バス，転送速度など)，HDD (容量，バス，転送速度など)
- ネットワーク資源
  ネットワーク帯域，インフラストラクチャ種別，接続デバイス

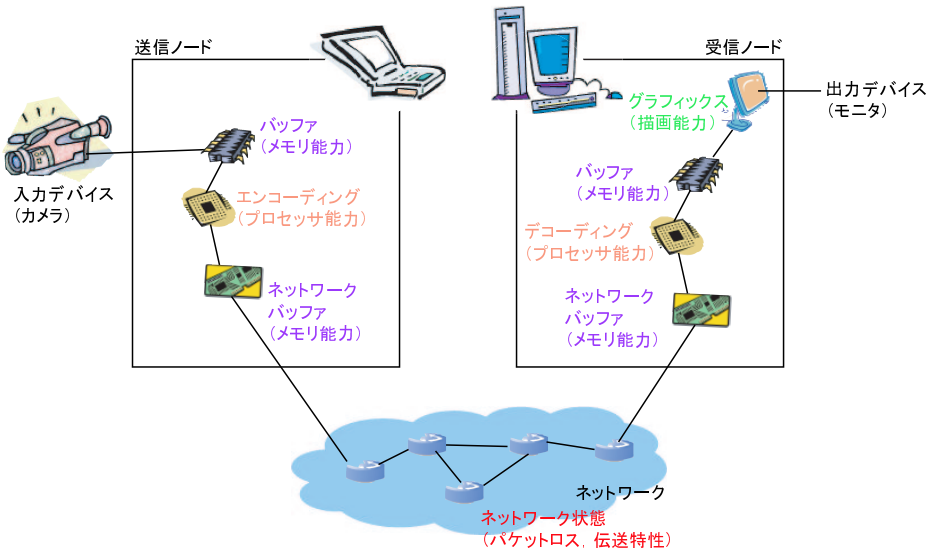

図 1.3: メディア配信のモデル

メディアフォーマット数の増加により，配信を行うユーザは自身の持つネットワーク環境や計算機環境に応じて利用可能なフォーマットの選択を行う必要がある．今後さらに利用可能なメディアフォーマット数は増加するため，ユーザ選択の重要度はさらに高まり，選択の難易度も高まる．

状況に適していないメディア配信の例を図 1.4に示す．この例では受信ノードに出力デバイス性能の低い PDA を利用しているため，送信ノードが伝送した高品位な映像は表示できない．性能の低い出力デバイスで表示を行うには，機器の性能にあわせた品質での映像伝送が必要である．

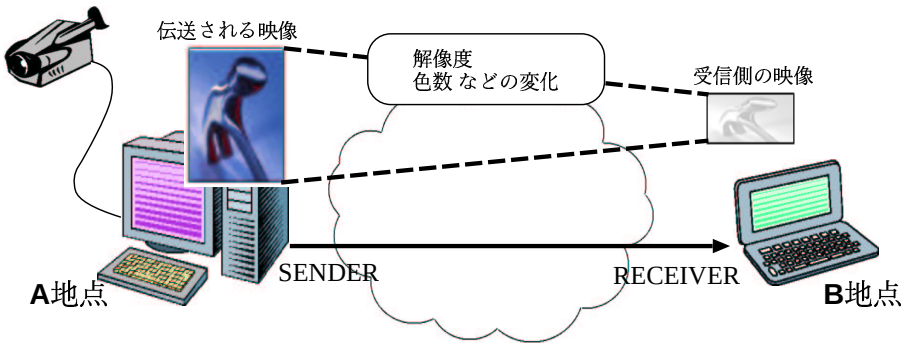

図 1.4: 最適でない映像伝送の例

また，メディア配信では多少のデータ損失よりもリアルタイム性がより重視されるため，再送制御や輻輳制御を考慮しない UDP(User Daragram Protocol)[17] を用いる．しかし，インターネットはベストエフォート型のネットワークであるため，ネットワーク状態の変化によりそれまで正常に利用できたフォーマットでも再生に支障が出る可能性がある．

1.3 本研究の目的

本研究では，インターネットを介したメディア配信において，エンドノード間で協調しユーザの利用する計算機やネットワーク状況を判別し，その状況に適したメディアフォーマットを自律的に選択・制御する自律協調型メディア配信手法を開発する．

エンドノード間での自律協調型メディア配信の実現のために，1) 送受信ノードの特定，2) 各ノードにおける計算機ならびにネットワーク資源の数値化とノード間での情報交換・共有，3) 取得した情報を基にした送信前・送信中の配信制御を行う手法を開発する．

1.4 本論文の構成

本論文は 8 章により構成される．第 2 章では本研究における要素技術として，現在のインターネットにおけるメディア配信技術の現状について述べる．第 3 章では本研究の対象とする関連技術とその問題点について述べる．第 4 章ではその問題点を解決するための本研究のアプローチとその有用性について述べる．第 5 章ではアプローチに基づいたシステムの設計を述べる．第 6 章では本システムの実装について述べる．第 7 章では本システムの評価を行う．最後に，第 8 章では本研究の結論を述べる．

第2章　インターネットにおけるメディア配信技術

本章では，本研究の要素技術として，インターネットにおけるメディア配信技術に関する概要を述べる．

インターネットを介したメディア配信に用いられる再生方式には，データ全体を受信した後に再生を行うダウンロード再生方式と，データの受信を再生と並行して逐次行うストリーミング再生方式の 2 つがある．図 2.1にダウンロード再生方式の概要，図 2.2にストリーミング再生方式の概要をそれぞれ示す．リアルタイムな映像・音声の配信には，ストリーミング再生方式が用いられる．本研究ではストリーミング再生方式を扱う．

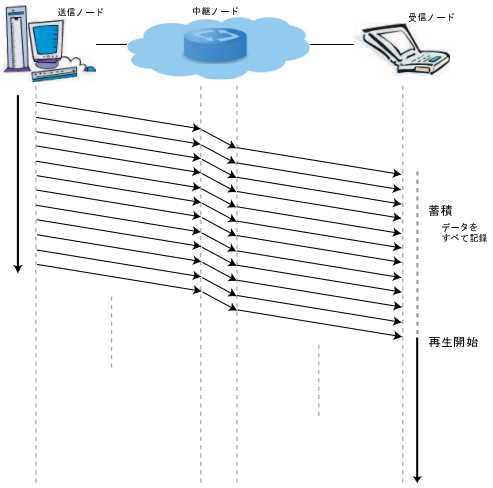

図 2.1: ダウンロード再生方式

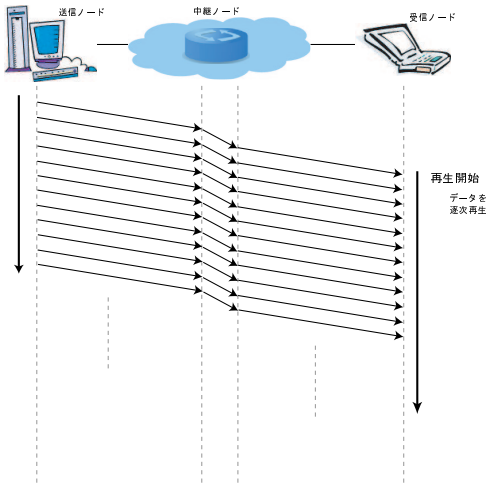

図 2.2: ストリーミング再生方式

2.1　ネットワーク環境に対する配慮

映像や音声をインターネットを介して転送する場合，途中経路においてパケットロスや転送遅延が発生する可能性があるため，ネットワーク転送において以下の 3 点のデータ信頼性確保に対する考慮が必要である．

- なるべくデータ欠損を出さないようにする考慮
- データ欠損に対応するために欠損を判別可能にするための考慮
- ネットワークにおける伝送特性に対する考慮

インターネットを介したメディア配信では，再生に対して影響を与える大きな要因としてネットワーク環境の変化が挙げられる．本節では，ネットワーク環境の変化によるメディア配信システム対する影響，配信システムの変化に対する考慮について述べる．

2.1.1 ストリーミング再生方式とジッタの揺らぎ

インターネットを介したデータ転送では，すべてのパケットの到達時間が保証されないため，パケット到着時間に揺らぎが発生する．この揺らぎをジッタと呼ぶ．ストリーミング再生方式におけるジッタの発生と再生の関係を図 2.3，図 2.4に示す．

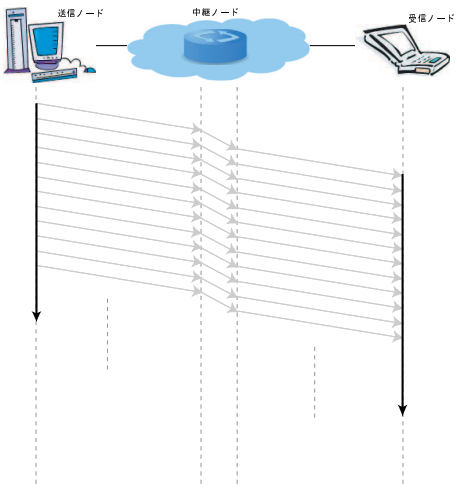

図 2.3: 正常な再生処理

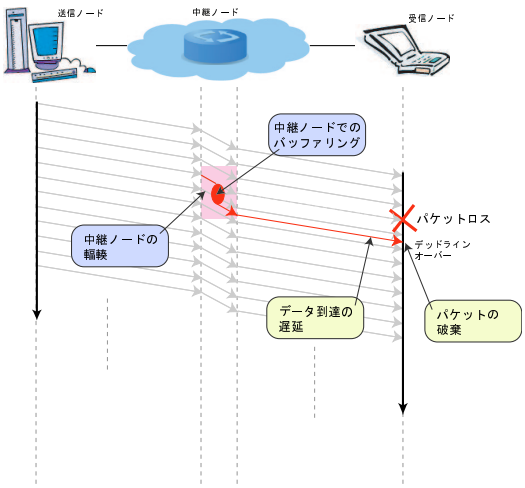

図 2.4: ジッタの発生と再生処理

図 2.3に示したデータの流れでは全てのパケットが正しく受信ノードに到達している．この場合，受信ノードでは映像・音声の再生が可能となる．しかし，図 2.4に示したデータの流れでは 1 つのパケットが中継ネットワークでの遅延の発生により到達が遅れている．この場合，再生のためのデッドラインまでにパケットが受信ノードに到達できず，再生途中にパケットロスが発生していると判断している．また，デッドラインから遅れたパケットは，受信ノードに遅れて到達したとしても，再生には利用されず破棄される．

2.1.2 ジッタへの対応

データ到達間隔の揺らぎによる再生への影響を軽減するため，再生プログラムではバッファを用意し，受信時に一定量のデータを蓄積し，再生はその蓄積されたデータから消費する方法が用いられる．これをバッファリングと呼ぶ．バッファリングは蓄積のためにデータ消費を一時的に遅らせるため，バッファリングを行う量とデータの到達遅延時間には比例関係がある．

また，あらかじめ保持するバッファ長によって異なる種類の配信手法が存在する．

- 再送型メディア配信
  Windows Media Player[18] や Real Player[19] などに用いられるメディア配信技術は，パケットロスが発生し再生が困難になった場合，一度再生を停止し十分な量のデータをバッファリングした後に再生を再開する．また，再生開始時にも十分な量のデータをバッファリングする必要がある．バッファ長が長いため，一時的なデータ欠損には強くなるが，エラー時に再度バッファリングするための時間分のエラー復帰時間が長くなる傾向がある．

- 非再送型メディア配信
  ビデオ会議システムや DVTS などに用いられるメディア配信技術は，バッファ長をなるべく短くすることで再生のリアルタイム性を重視した構造となっている．しかし，バッファ長が短いため，一時的なデータ欠損に対しても映像・音声の表示が乱れる問題があるが，エラー時の復帰時間が短いという利点もある．

2.1.3 インターネットにおける伝送特性

インターネットにおける通信路の状態はさまざまな要因によって変化する．通信路の状態変化はパケットの伝送特性の変化として表れる．この伝送特性の変化を観測することでネットワークの状態予測や検知が可能である．特にエンドノード間で情報取得可能な伝送特性として実効帯域幅，往復伝搬遅延時間，遅延と伝搬時間の揺らぎ，パケット喪失率の 4 つが挙げられる．

- 実効帯域幅 (実効スループット)
  送信ノードから受信ノードまで実際に利用できる帯域幅を実効帯域幅と呼ぶ．実効帯域幅はパケットが送信ホストから受信ホストまで到達する時間とパケットサイズを基に推測できる．パケットペアスキーム [20] を用いた測定手法が存在するが，測定に多大な時間と通信量を要する．

- 往復伝搬遅延時間 (RTT)
  送信ノードと受信ノード間でパケットの往復に要する時間を RTT(Round Trip Time) と呼ぶ．RTT によりネットワークの大まかな遅延を知ることができる．RTT は ping などのツールを用いて測定できる．

- 片方向遅延時間と伝搬時間の揺らぎ (ジッタ)
  送信ノードから受信ノードまでのネットワーク状態に応じて伝搬にかかる時間が変化する．さらに，2.1.1項にて述べる要因によりパケットの到達時間は前後する．これを片方向遅延時間と呼ぶ．各パケットは送信された時からある程度遅れて受信ノードに到達する．この伝搬時間の揺らぎは到達時間に関して保証のないインターネットでは一定ではない．

- パケット喪失率
  送信パケット数と受信パケット数を比較することで中継経路におけるパケット喪失数を知ることができる．喪失したパケットの割合をパケット喪失率と呼び，喪失したパケット数を送信パケット数で割ることで求める．パケット喪失率によりネットワークのトラフィック状況を推測できる．

2.2 伝送フォーマットに対する考慮

インターネットを介したメディア配信では，通常のインターネット上のパケット伝送と異なり，リアルタイムかつデータ欠損のないパケット伝送が要求される．また，パケット欠損が発生した場合でも即座に把握するための考慮が必要となる．本節ではパケット伝送におけるメディア配信システムの考慮について述べる．

2.2.1 RTP

映像や音声の配信を行うアプリケーションの多くは，パケットの配送に，TCP (Transmission Control Protocol)[21] ではなく，UDP を用いる．その理由は TCP による再送制御による問題が発生するためである．

TCP では受信者がパケットを受け取る度に確認応答 (acknowledgement) を送信者に対して送信することにより信頼性の確保をしている．送信者はパケットを送信する際にタイマを設定し，そのタイマが切れるまでに確認応答を受け取れない場合，パケットロスが発生したと判断し，該当するパケットの再送 (retransmit) を行う．しかし，映像や音声の再生ではロスしたパケットが再送されたとしても，再生データの到着が再生に間に合わない場合，再送されたパケットは意味を失う．

UDP を用いた場合，TCP を用いた場合と比較してトランスポート層におけるパケット到達順序の保証がなくなるため，パケットの到達順序が正しいかどうかの判断をアプリケーション層にて行う必要がある．

到達順序を知る手法として RTP (Realtime Transport Protocol)[22] の利用がある．RTP は，リアルタイム性が重視される映像・音声データの転送に利用されるプロトコルであり，下位層のトランスポートプロトコルとは独立した設計となっている．RTP はパケットの順序番号やタイムスタンプなどの情報を付加する．この情報を元に，受信側は正しいパケットの順番を知ることができるが，RTP は配送や配送の到達順序自体を保証しない．

2.2.2 RTCP

映像や音声の配信に用いられる RTP では，パケット順序やパケット喪失の情報を計測はできるが，伝搬遅延・ジッタなどのネットワーク状態の把握に必要な情報を取得できない．これらの情報を取得するために RTCP(Realtime Transport Control Protocol)[22] が用いられる．RTP セッションに対する遅延・ジッタ・帯域幅・輻輳状態などのフィードバック情報を RTCP を用いて交換することにより，ネットワークの状態を把握し，配信の制御を行うことができる．

しかし，RTCP は，RTP のための通知プロトコルであり，データ配送や TCP のようなエンドノード間でのデータ到達性を保証しない．

2.3 メディアタイプに対する考慮

インターネットを介したメディア配信では，1.2節にて述べた“ 双方アドレスの把握 ”と“ フォーマットの同一性 ”が要求される．これらの実現にはアドレス情報とフォーマット情報の把握が必要である．本節では既存のメディア配信システムにおける実現手法について述べる．

2.3.1 SDP を用いたセッション情報記述

インターネットを介したメディア配信では，セッション内容について記述する手法に SDP(Session Description Protocol)[23] がある．メディアの開始時刻や送信者の情報，接続先のアドレス情報などが SDP によって記述される．SDP によるセッション記述の例を図 2.5に示す．この例では，pluto.kddi.co.jp から映像を H.261[24]，音声を PCM μ-raw で伝送することを表している．

```
v=0
o=bob 2890844526 2890844526 IN IP4 pluto.kddi.co.jp
c=IN IP4 pluto.kddi.co.jp
m=audio 49170 RTP/AVP 0 1
a=rtpmap : 47920 PCMU/8000
m=video 51372 RTP/AVP 31
a=rtpmap : 31 H261/90000
```

図 2.5: SDP 記述の例

SDP では field=attribute という形式で必要な情報を保持する．o(owner) フィールド，c(connection) フィールドはセッションに関する記述 (セッションの識別と接続先アドレスなどを保持)，m(media) フィールドはメディアに関する記述 (使用しているフォーマットなどを保持) をそれぞれ行い，これらのフィールドは必須フィールドである．また，a(attribute) フィールドはメディア記述のオプションとして利用される．

2.3.2 SAP を用いたセッション情報配信

SAP(Session Announcement Protocol)[25] は，SDP で記述されたコンテンツ情報をマルチキャストネットワークに対して定期的に広告し，そのデータを利用して受信ノードで番組表を生成するために用いられる．基本的な動作を図 2.6に示す．

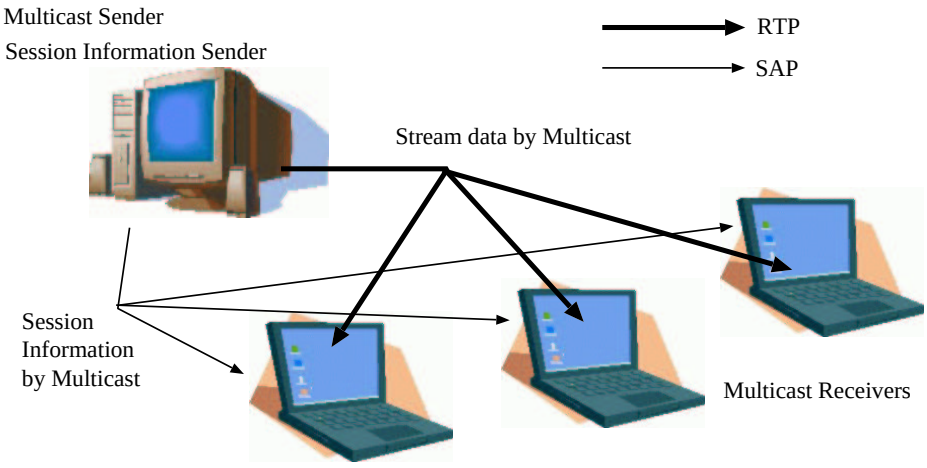

図 2.6: SAP によるセッション情報のマルチキャスト配信

SAP を用いたコンテンツ情報配信は，あらかじめ決められたマルチキャストアドレスに対して定期的に SDP 記述を含んだデータを送信することで行われるため，マルチキャストの利用が前提となる．受信ノードでは，このマルチキャストアドレスに対して参加 (Join) することで

定期的にメディアに関する情報を取得し，自身の持つ番組表データを更新する．メディアを受信するためにはこの番組表データを用いる．また，SAP はマルチキャストの配信エリア制御同様に TTL(Time To Live) をエリア制御に用いる．

2.3.3 SIP を用いた呼制御

SIP(Session Initiation Protocol)[26] は，インターネット電話システムに多く利用されているプロトコルで，IETF により標準化されている．SIP は，電話のような 1 対 1 のメディア配信から，複数グループが参加するマルチキャスト会議のようなメディア配信まで幅広く応用可能である．SIP は以下のような機能を持つ．

- **セッションの設定：**双方アドレスの把握を行うとともに，あるノードからあるノードに対しての接続要求を行うことでセッションを確立する．
- **メディアネゴシエーション：**SDP を用いて，送受信ノード間で利用可能なメディアフォーマットを記述し，ノード間でネゴシエーションを行い，フォーマットを決定する．
- **セッション変更：**一度確定したメディアフォーマットの変更を要求する．
- **セッションの終了とキャンセル：**一度確立したメディアセッションを終了，または呼び出しを中止できる．
- **呼制御：**セッションの他ノードへの転送など，複雑な呼び出し制御ができる．

SIP を用いた呼制御は，送受信ノードであるクライアント，仲介サービスノードであるプロキシ (またはリダイレクトサーバ) を用いることで実現する．相手ノードへの接続の流れを図 2.7に示す．

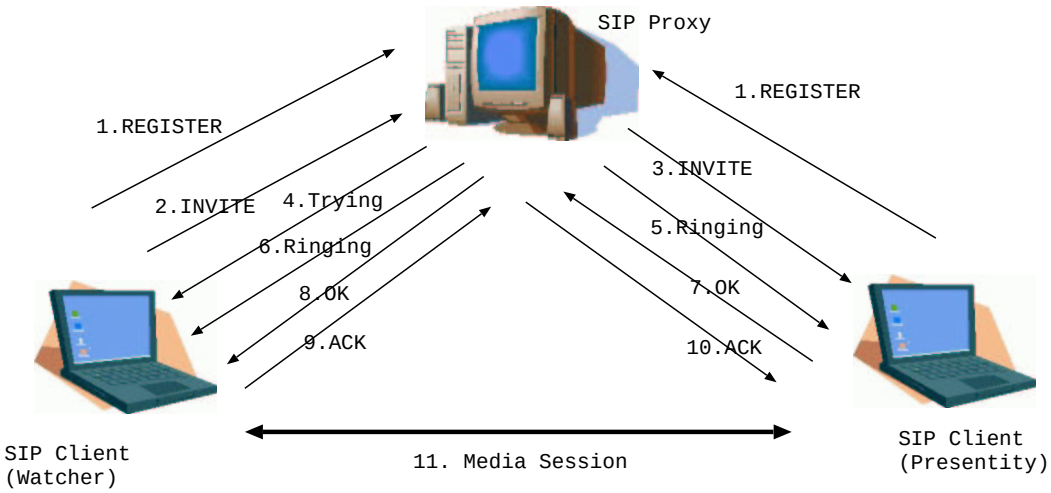

図 2.7: SIP による呼制御

あらかじめクライアントはリダイレクトサーバに対して REGISTER 要求を発行することで自身のネットワーク位置を登録する．相手ノードを呼び出したいクライアントは，プロキシに対して接続要求 (INVITE 要求) を発行する．要求を受けたプロキシは，登録されているクライ

アント情報から相手ノードの情報を検索し，適切な相手ノードに対して接続要求を代理で行う．要求を受けた相手ノードは，着呼可能な状態であれば OK を返し，送受信ノード間でメディアセッションを確立する．クライアントが接続先相手の情報を保持している場合は，プロキシを介さずにセッションを確立できる．

2.4 システム環境に対する考慮

メディア配信は，1) 映像・音声データを生成する入力デバイス，2) 配信を行う送信ノード，3) 配信を受ける受信ノード，4) 映像・音声データを表示する出力デバイス，5) データの配送を行う中継ノード (ネットワーク) の 5 つの要素で構成され，各要素が協調して動作することで実現している．送信ノードから受信ノードまでのメディア配信の構成要素を図 2.8に示す．

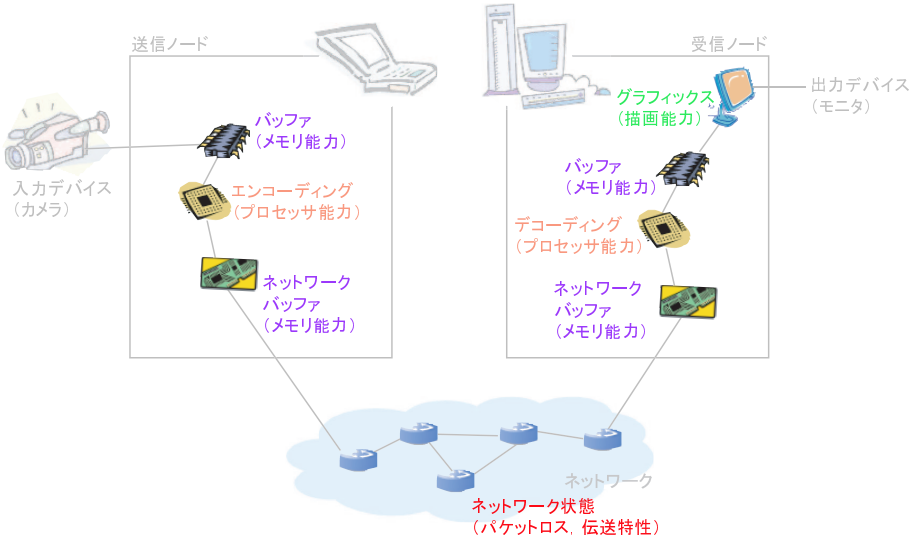

図 2.8: メディア配信のモデル

メディア配信において行われる処理は以下の 5 つの機能に分類される．

- **入力：**入力デバイスから映像・音声データが入力される
- **エンコーディング (符号化)：**情報を配信するために必要な形式に変換される (この処理は入力デバイスで事前に行われる場合がある)
- **伝送：**ネットワークを通じて相手ノードに対して配信される
- **デコーディング (復号化)：**情報を表示するために符号化されているデータを復号する (この処理は出力デバイスで同時に行われる場合がある)
- **出力：**出力デバイスに映像・音声データを出力する

また，これらの処理に必要な資源は以下の 4 つに分類される．

- **プロセッサ資源：**
  映像・音声データのエンコーディング処理やデコーディング処理はプロセッサ資源が大きく影響する．また，すべてのシステム処理を実行するためにプロセッサは必要となる．プロセッサはオペレーティングシステムを通じて必要なプロセスを処理する．プロセッサの処理資源を超えたシステム処理ができない．

- **描画資源：**
  映像・音声の表示処理では描画資源が大きく影響する．特に解像度の大きなデータの表示にはより多くの描画資源を必要とする．

- **メモリ資源：**
  メディア配信では各処理においてデータを一時的に格納するバッファリング処理を行う．このバッファリング処理ではメモリ資源が大きく影響する．データの格納に必要なメモリ領域を確保できない場合，データ欠損などの問題が発生する可能性がある．また，すべてのシステム処理を実行するためにメモリは必要となる．

- **ネットワーク資源：**
  メディア配信ではデータの配送にネットワーク資源を利用する．インターネットはベストエフォート型ネットワークであるため，突発的な要因により伝送状況が変化する可能性がある．また，ネットワーク資源を超えたデータ配送やバースト的なデータ配送は中継ノードにおける処理遅延やデータ欠損が発生する要因となる．

第3章　既存技術とその問題点

本章では，既存のメディア配信システムならびにメディア情報記述・配信システムの現状について述べ，メディア配信システムの持つ問題点について整理する．

3.1　メディアセッション情報記述プロトコル

本節では，既存のメディア情報記述手法の概要とインターネットを介したメディア配信に対する考慮の状況，問題点を述べる．

3.1.1　SDP

SDP は，2.3.1項で述べたメディアセッションに対する情報記述プロトコルである．SDP ではテレビなどの番組表のように，データが配信されるチャンネル (マルチキャストアドレス)，番組の内容，開始終了時刻の情報を記述できる．

SDP における考慮事項と問題点の整理

インターネットを介したメディア配信において考慮すべき項目について，SDP は以下の対応を行っている．

- **ネットワーク環境に対する考慮：**使用する帯域幅について記述するフィールドが存在するが，ネットワーク状態の変化に対する記述を行うフィールドは存在しない．
- **伝送フォーマットに対する考慮：**使用する伝送フォーマットについて記述するフィールドが存在する．
- **メディアタイプに対する考慮：**複数メディアフォーマットについて個別に記述できる．
- **システム環境の状態変化に対する考慮：**システム環境の状態変化に対して考慮するフィールドを持たない．

SDP を用いた記述は，その時点におけるメディアセッションの状態を記述するものであり，SIP などのシグナリングプロトコルを組み合わせて利用する必要がある．また，ネットワークや計算機資源について記述する項目が存在しないため環境資源の状況変化を考慮した記述ができない．

3.1.2 H.245

H.245[27] は H.323 プロトコルセット [28] が利用する呼び出し制御プロトコルである．H.245 はメディアセッションを送受信ノード間で確立するために必要な手法が定義されている．図 3.1にノード間での情報交換の流れを示す．

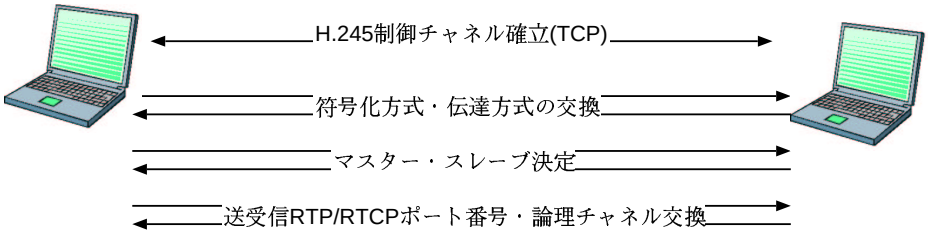

図 3.1: H.245 による Capability Exchange

H.245 では着呼時に Capability Exchange を用いてフォーマットの決定が行われる．Capability Exchange では実効可能な符号化方式と伝達方式に関して交換する．

H.245 における考慮事項と問題点の整理

インターネットを介したメディア配信において考慮すべき項目について，H.245 は以下の対応を行っている．

- **ネットワーク環境に対する考慮：**伝送に利用するポート番号などの情報を記述するフィールドが存在するが，ネットワーク状態について記述できない．
- **伝送フォーマットに対する考慮：**使用する伝送フォーマットについて記述するフィールドが存在する．ただし，記述可能フォーマットは，ITU 系フォーマット (H.263[29], G.721[30] など) に限られている．
- **メディアタイプに対する考慮：**複数メディアフォーマットについて個別に記述できる．
- **システム環境の状態変化に対する考慮：**システム環境の状態変化に対して考慮するフィールドを持たない．

H.245 を用いた記述はノードの持つフォーマット情報の交換を主目的としたものであり，状況変化に対して適応するための記述ができない．また，ネットワークや計算機資源について記述する項目が存在しないため，環境資源の状態変化を考慮した記述ができない．

3.1.3 MPEG21+MPEG7

MPEG21[31] はメディアデータのハンドリングを行うための情報データ形式の標準，MPEG7[32] はメディアに関するメタデータの記述を行うための標準である．

MPEG21+MPEG7 における考慮事項と問題点の整理

インターネットを介したメディア配信において考慮すべき項目について，MPEG21+MPEG7 は以下の対応を行っている．

- **ネットワーク環境に対する考慮：**伝送に利用するポート番号などの情報を記述するフィールドが存在するが，ネットワーク状態について記述できない．
- **伝送フォーマットに対する考慮：**使用する伝送フォーマットについて記述するフィールドが存在する．ただし，記述可能フォーマットは，MPEG 系フォーマット (現段階では，静止画の配信には JPEG2000，動画の配信には MPEG4 のみ) に限られている．
- **メディアタイプに対する考慮：**複数メディアフォーマットについて個別に記述できる．
- **システム環境の状態変化に対する考慮：**システム環境の状態変化に対して考慮するフィールドが存在するが，状態変化に対して記述できない．

MPEG21+MPEG7 を用いた記述はノードの持つフォーマット情報とノードの持つ計算機資源情報の交換を主目的としたものである．この手法を応用することで，クライアントの状態にあわせて動的にフォーマットの変更を行う処理が可能であるが，セッション開始時のネゴシエーションに限られるため，環境資源の変化に応じた記述ができない．

3.1.4 メディアセッション情報記述プロトコルに関するまとめ

本節では，既存のメディア情報記述手法の特徴を述べ，インターネットを介したメディア配信において考慮すべき項目への対応状況について整理した．表 3.1に対応状況の結果をまとめる．前項までに挙げたメディア情報記述手法について，ネットワーク環境に対する記述が行えるか (インフラ)，伝送フォーマットに対する記述が行えるか (伝送フォーマット)，メディアタイプに対する記述が行えるか (メディア)，システム環境の状態変化に対する記述が行えるか (システム) という 4 つの観点から，4 段階 (◎，○，△，×) で評価した．

表 3.1: メディア情報記述手法のまとめ

既存手法	インフラ	伝送フォーマット	メディア	システム
SDP	△	○	○	×
H.245	△	△	○	×
MPEG21+MPEG7	△	△	○	○

表 3.1より，いずれのプロトコルにおいても伝送フォーマットについて記述可能である．しかし，ネットワーク状態や計算機状態の変化に対して動的に対応できる記述ができないため，既存のセッション記述手法では，動的な変化というものに着目した記述ができない．

3.2 メディア配信アプリケーション

本節では，現在インターネットで利用されている 3 つのメディア配信アプリケーションについて，その特徴と個別の状況を述べ，メディア配信アプリケーションにおける問題点を整理する．

3.2.1 DVTS

民生用デジタル AV 機器を用いた映像・音声配信を行うシステムとして，DVTS (Digital Video Transfer System)[33] がある．DVTS は非再送型メディア配信システムの 1 つである．

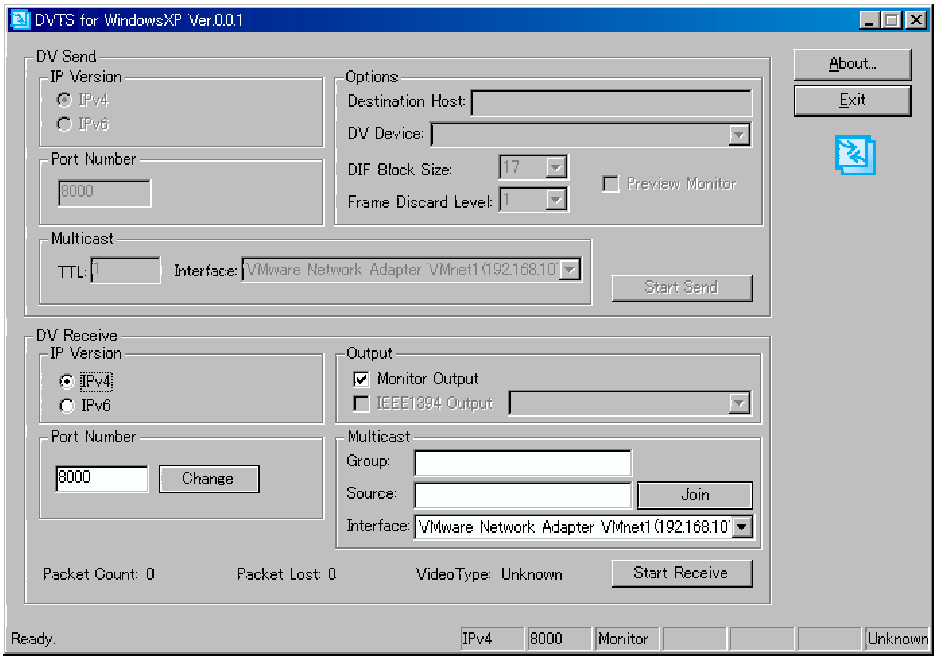

図 3.2: DVTS for Windows

DVTS は，IEEE1394 インタフェース [34][35][36] から DV フォーマット [37] のストリームを取得し，そのストリームに対し IP データグラム化を行い，インターネットを介して映像・音声の配信を行う．このシステムを利用することにより，リアルタイム，かつ高品質な映像の配信が可能である．DVTS は以下の 5 つの機能を有する．

- IPv4/IPv6 への対応
- RTP, UDP を利用した通信
- フレーム間引き機能を有し，送出データ量の調整が可能
- 複数地点へのストリーム送信が可能なマルチキャスト (ASM,SSM)[38] への対応
- UNIX のほか，Windows や MacOSX など幅広いオペレーティングシステムに対応

DVTS は DV フォーマットの持つ以下の 5 つの特徴を持つ．

- フレーム内圧縮
- 解像度は 720pixel × 480pixel
- フルフレーム時のフレームレートは 29.97fps
- 圧縮率は一定
- フルフレーム時の送出データ量は約 32Mbps

DVTS における考慮事項と問題点の整理

インターネットを介したメディア配信において考慮すべき項目について，DVTS は以下の対応を行っている．

- **ネットワーク環境に対する考慮：**フレーム間引き機能によりデータ量の調整が可能．また，1)TCP-Friendly Congestion Control for DVTS，2)Packet Lossless TCP-Friendly DVTS using ECN，3) パケット到達間ジッタによる配信制御の 3 つのデータ送出量制御機能がある．
- **伝送フォーマットに対する考慮：**RTP/RTCP に対応し，IP ネットワーク上を RTP パケットとして伝送される．
- **メディアタイプに対する考慮：**SDP/SAP を利用したセッション情報配信機構を持つが，DVTS とは別モジュールで動作する．
- **システム環境の状態変化に対する考慮：**システム環境の状態変化に対して考慮する機構を持たない．

DVTS では一部機能についてメディア配信において考慮すべき項目を満たしている．しかし，これらの機能は独立して実装されており，統合した制御を行う実装は存在しない．また，メディアタイプやシステム環境の状態変化を処理する機構を持たないため，フォーマットを意識したメディアネゴシエーションやシステム環境の状態変化を意識した配信制御ができない．

3.2.2 Windows Media Player

Windows Media Player は Microsoft 社によって開発されたメディア配信システムである．配信を行う Encoder，仲介する Server，受信を行う Player があり，小規模な配信から大規模な配信までさまざまな配信に利用可能である．Windows Media Player は再送型メディア配信システムの 1 つである．

Windows Media Player で利用されるフォーマットは，Windows Media Techlonogy(WMT) と総称され，圧縮機能を用いることで HDTV 品質の映像を数 Mbps 程度の帯域幅で送受信できる．また，送信側の入力ソースは IEEE1394 インタフェースまたは標準キャプチャデバイスを利用し，そのデータを WMT 方式でエンコードを行った上で IP データグラム化し，インターネットを介して映像・音声の転送を行う．再送型メディア配信システムであるため，受信側ではやや長めのバッファリングが必要であり，リアルタイム性は損なわれるが帯域使用が少なく映像品質が高いことから現在広く利用されている．Windows Media Player は，以下の 6 つの機能を有する．

- IPv4/IPv6 への対応
- RTP, HTTP, UDP, MMS を利用した通信
- 細かな送出データ量の調整が可能
- 映像圧縮はフレーム間圧縮

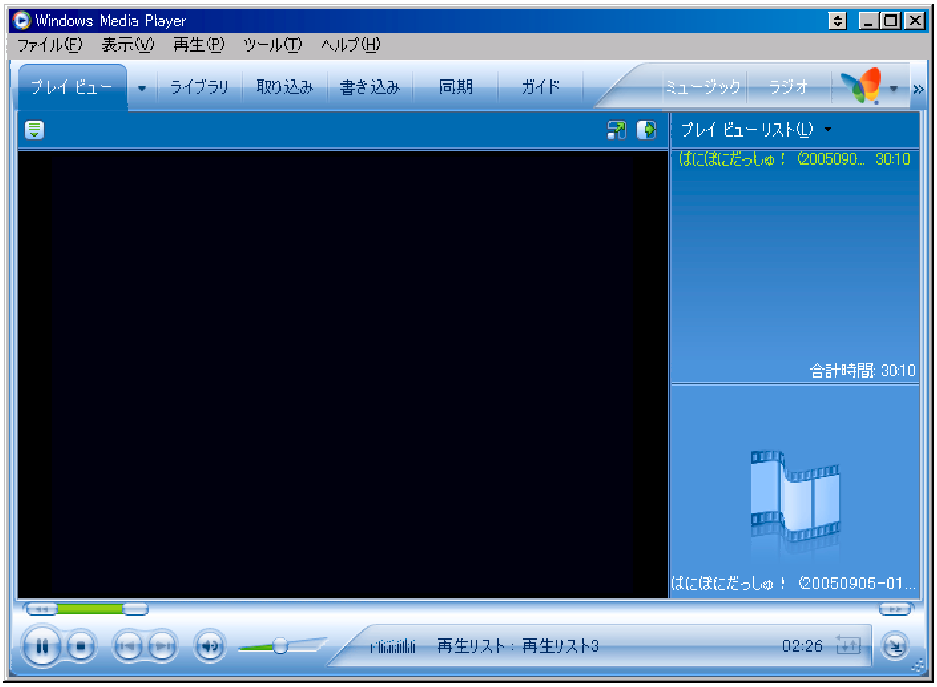

図 3.3: Windows Media Player 10

- 複数地点へのストリーム送信が可能なマルチキャストへの対応
- UNIX のほか，Windows や MacOSX など幅広いオペレーティングシステムに対応

WMT 方式のエンコードは MPEG4 をベースとしたフレーム間圧縮を用いるため，Windows Media Player ではフォーマットの持つ以下の 5 つの特徴を持つ．

- フレーム間圧縮
- 解像度は自由に変更が可能 (送出側の資源量による)
- フレームレートは自由に変更が可能
- 圧縮率は VBR
- HD-WMT の送出データ量は約 8Mbps

Windows Media Player における考慮事項と問題点の整理

インターネットを介したメディア配信において考慮すべき項目について，Windows Media Player は以下の対応を行っている．

- **ネットワーク環境に対する考慮：**データの圧縮を行う際に解像度の変更・フレームレートの変更・圧縮率の変更を行うことで細かな伝送レートの調整を行う．また，複数レートのデータを同時に有することで，帯域幅の変動に対して対応可能である．

- **伝送フォーマットに対する考慮：**配信を行う Encoder からの配信では基本的に HTTP を用いた配信を行う．仲介を行う Server を経由する際には MMS を用いることもできる．また，RTP に対応し，IP ネットワーク上を RTP パケットとして伝送される．
- **メディアタイプに対する考慮：**メディアタイプが WMT 方式固定であるため，メディアフォーマットに関する広告機能は有さない．しかし，セッション内容に関する記述はメディアファイル内で行う．
- **システム環境の状態変化に対する考慮：**システム環境の状態変化に対して考慮する機構を持たない．

Windows Media Player では，ネットワーク環境に対する考慮は行われているが，ネットワーク環境に対する考慮以外の機能を含めて，統合的な制御を行う機能は存在しない．また，配信を行う際に複数のメディアフォーマットを意識したシステムではないため，フォーマットを意識したメディアネゴシエーションやシステム環境の状態変化を意識した配信制御ができない．

3.2.3 VLC (VideoLAN Client)

VLC は，VideoLAN Project[39] によって開発されたクロスプラットフォームメディアプレイヤーであり，複数のメディアフォーマット，複数のプラットフォームに対応している．また，単体でメディア配信を行える非再送型メディア配信システムの 1 つでもある．

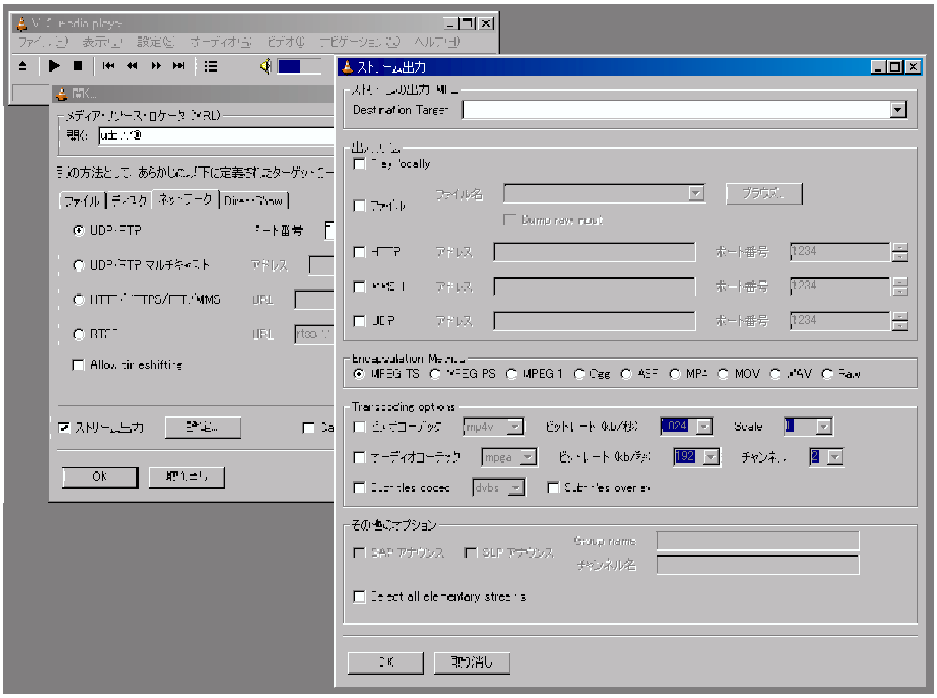

図 3.4: VideoLAN Client

VLC は，MPEG2 デコーダを独自で持ち，単体で DVD 再生が可能なメディアプレイヤーとして広く利用されている．また，メディアフォーマットとして，MPEG2，MPEG1，WMV，AVI，DivX，MP3，OGG，WMA など多種多様なフォーマットに対応しており，インターネッ

トを介したこれらのフォーマットの受信にも適用可能である．VLC は以下の 6 つの機能を有する．

- IPv4/IPv6 への対応
- 単体で複数のメディアフォーマットに対応
- RTP, UDP, HTTP, MMSH を利用した通信
- トランスコーディングによるフレームレートの調整が可能
- 複数地点へのストリーム送信が可能なマルチキャスト (ASM,SSM) への対応
- UNIX のほか，Windows や MacOSX など幅広いオペレーティングシステムに対応

VLC では，トランスコーディング機能を用いることで，様々なメディアフォーマットを利用したメディア配信が可能である．VLC が配信に対応するメディアフォーマットは，MPEG-TS, MPEG-PS, MPEG1, Ogg, ASF, MP4, MOV, WAV, Raw がある．また，トランスコーディングに対応するフォーマットには，MPEG1, MPEG2, MPEG4, DivX, H.263, H.264, WMT, Motion-JPEG などがある．これらのフォーマットを利用する VLC ではフォーマットの持つ以下の 4 つの特徴を持つ．

- 圧縮方式はメディアフォーマットによって異なる (主にフレーム間圧縮)
- フレームレートは自由に変更が可能 (トランスコーディング機能による)
- 圧縮率はメディアフォーマットによって異なる (主に VBR)
- 送出データ量はメディアフォーマットによって異なる

VLC における考慮事項と問題点の整理

インターネットを介したメディア配信において考慮すべき項目について，VLC は以下の対応を行っている．

- **ネットワーク環境に対する考慮：**トランスコーディングを行う際にフレームレートの変更が可能である．しかし，ネットワークの状況に応じたレート調整は基本的に不可能である．
- **伝送フォーマットに対する考慮：**RTP に対応し，IP ネットワーク上を RTP パケットとして伝送される．また，HTTP や MMSH を用いた配信にも対応している．
- **メディアタイプに対する考慮:**複数メディアに対応し，セッションに関する情報を SAP/SDP で広告する機能がある．また，受信データの解析により，自動的に利用するデコーダを切り替えることでセッション広告手法を用いない形で動的なメディアタイプ判別を行う．
- **システム環境の状態変化に対する考慮：**システム環境の状態変化に対して考慮する機構を持たない．

VLC では，複数メディアへの対応を行うための機能として，受信データからのメディアタイプ判別ならびに SAP/SDP での広告機能を有する．しかし，ネットワークの状況や計算機の状況に応じてフォーマットを自動決定したり，動的に変更する機能を持たない．

3.2.4 メディア配信アプリケーションに関するまとめ

本節では既存のメディア配信システムの特徴を述べ，インターネットを介したメディア配信において考慮すべき項目への対応状況について整理した．表 3.2に対応状況の結果をまとめる．前項までに挙げたメディア配信システムについて，ネットワーク環境に対する考慮がされているか (インフラ)，伝送フォーマットに対する考慮がされているか (伝送フォーマット)，メディアタイプに対する考慮がされているか (メディア)，システム環境の状態変化に対する考慮がされているか (システム) という 4 つの観点から，4 段階 (◎，○，△，×) で評価した．

表 3.2: メディア配信システムのまとめ

既存手法	インフラ	伝送フォーマット	メディア	システム
DVTS(DV フォーマット)	△	◎	△	×
WMP(WMT フォーマット)	○	○	△	×
VLC	△	◎	○	×

表 3.2より，いずれのメディア配信アプリケーションにおいても，ネットワークを介した配信を行うための伝送フォーマットに対する考慮が行われ，ネットワーク状態に応じてデータの送出調整することでレート制御を行う機能がある．しかし，計算機状態に応じた配信制御やアプリケーションにおけるメディアタイプに対する考慮が不十分であるため，いずれのメディア配信システムにおいてもすべての項目が満たされていない．

3.3 まとめ

本章では既存の手法のシステム的特徴とインターネットを介したメディア配信における考慮点への対応状況について述べた．

既存の手法では，一部に対して有効な手法が存在しているものの，インフラストラクチャ・メディア・システムについて統一的に扱い，かつ自律協調処理が可能なシステムが存在しない．これらの問題を解決するべく次章にて新たな自律協調型メディア配信手法について提案する．

第4章　環境資源に適応する自律協調型メディア配信手法の提案

本章では，環境資源に適応する自律協調型メディア配信手法の提案を行う．4.2節にて本研究に求められる要件を整理し，それぞれの手法の有効性について実証する．さらに，本手法を実現するためにメディア配信システムが必要とする事項についてまとめる．

4.1　自律協調型メディア配信

インターネットを介したメディア配信は，さまざまな状態変化に応じた映像・音声の配信を行う必要がある．

インターネットを介したメディア配信では，1) ネットワーク環境，2) 計算機環境，3) メディアフォーマットに対する判別・考慮が必要不可欠である．本研究では，状態に応じたメディア配信を自律協調型メディア配信と定義する．

使用する計算機の資源やネットワーク資源は，状況に応じて変化する．自律協調型メディア配信を実現するためには，計算機やネットワークを構成する要素の動作状況を把握し，状況に応じた制御が必要である．また，計算機やネットワークの状況を記述し，その情報を基に送信ノード，受信ノード，中継ノードが協調動作する統一的な手法が必要である．

4.2　環境資源の把握と制御

本研究ではメディア配信に必要な環境資源を以下の 2 つと定義する．その上で，環境資源に応じた配信制御について述べ，その手法の有効性について検証する．

1. 計算機資源：計算機の資源量を用いた計算機状態把握
2. ネットワーク資源：伝送特性を用いた動的レート制御

4.2.1　計算機資源量を用いた計算機状態把握と配信制御

メディア配信システムでは，送受信ノードにおける計算機の状態によって映像・音声の再生に問題が発生する可能性があるため，再生に必要な計算機資源の確保と管理が必要である．本研究では 1) 再生開始時，2) 再生中の 2 つの時点における計算機資源の使用状況の把握と使用状況に応じた配信制御の手法を提案する．本手法では，メディア配信に必要な計算機資源のうち，以下の 2 点について着目する．

- プロセッサ資源：処理可能な資源が残存しているかどうか
- メモリ資源：処理可能な資源が残存しているかどうか

計算機状態に応じたメディア配信は，計算機上における必要な情報の取得，評価値を基にしたレート制御とフォーマット制御を行うことで実現する．メディア配信は，より多くの計算機資源を必要とするため，これらの指標を用いた配信制御は多くのメディア配信アプリケーションにおける配信制御手法として適している．

本手法の有効性検証

メディア配信に利用される計算機資源は 2.4節に述べた通りである．本項では，実際の動作における資源量の変化を実験により確認し，資源量の変化を用いた制御の有効性について検証する．

メディアフォーマットと資源量の関係を確認するために，メディアフォーマットとして DV，MPEG2-TS，非圧縮ビットマップのそれぞれを配信するシステムを構築し，送受信ノード間で実際にデータ配信を行う．本実験は，図 4.1に示す環境で行われ，利用した計算機を表 4.1に示す．

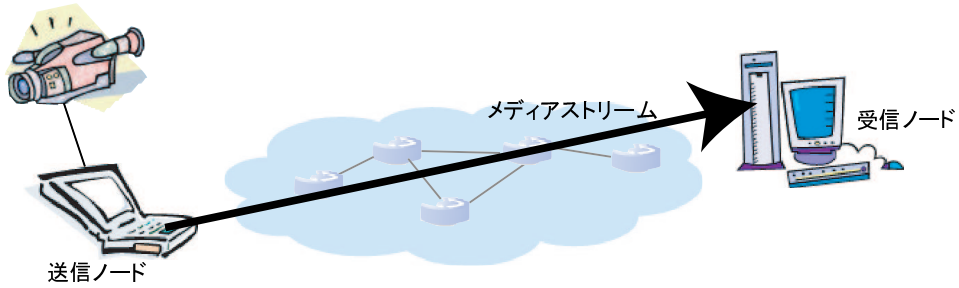

図 4.1: 利用した配信環境

表 4.1: 利用した計算機

CPU	Intel Xeon Processor 3.2GHz
メモリ	2048MB
ハードディスク	73GB
NIC	100Base-TX, 1000Base-T
映像・音声機器 (1)	Victor HDV カメラ GR-HD1
映像・音声機器 (2)	Logicool USB カメラ QuickCam Pro for Notebook

各メディア配信システムは，Microsoft WindowsXP Professional を動作対象オペレーティングシステムとする．各データの取得は Windows のパフォーマンスカウンタを用いる．データは 1 秒ごとに取得され，1) 定常状態，2) データ送受信，3) 定常状態の計 5 分間で計測を行った．

- DV
  DV を用いた場合の各資源量の使用率変化のグラフを図 4.2 に示す.

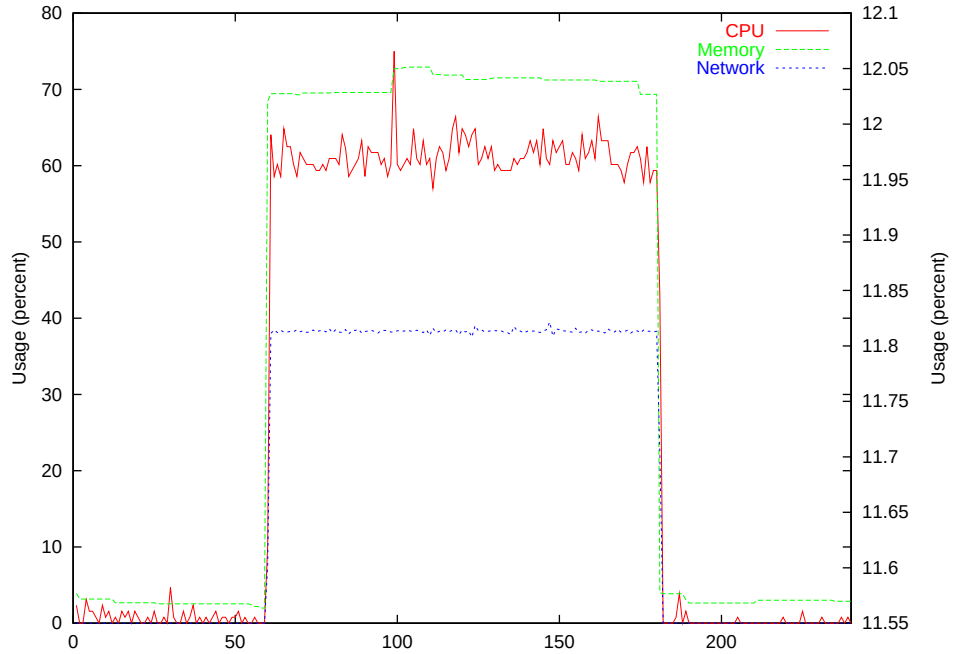

図 4.2: DV フォーマットでの資源使用率変化

本実験では 60 秒から 180 秒にかけて送信ノードから受信ノードに対して映像・音声データの配信を行った．その直後にアプリケーションを終了させた．メモリ使用率については変動幅が小さいため Y_2 軸を利用している．グラフより映像・音声の配信中にはすべての資源が利用されている．

実験を行った計算機では，DV フォーマットでの配信にはプロセッサ資源を 60%から 70%，メモリ資源を 2%，ネットワーク資源を 35%要した．DV フォーマットは，映像・音声のデコーディングのためプロセッサ資源を，データ転送のためネットワーク資源を多く必要とする．

- MPEG2-TS
  MPEG2-TS を用いた場合の各資源量の使用率変化のグラフを図 4.3に示す.

本実験では 60 秒から 180 秒にかけて送信ノードから受信ノードに対して映像・音声データの配信を行った．その直後にアプリケーションを終了させた．メモリ使用率については変動幅が小さいため Y_2 軸を利用している．グラフより映像・音声の配信中にはすべての資源が利用されている．

実験を行った計算機では，MPEG2-TS フォーマットでの配信にはプロセッサ資源を 70%から 85%，メモリ資源を 2%，ネットワーク資源を 30%要した．MPEG2-TS(HDV) フォーマットでは，高解像度な映像のデコーディングに DV フォーマット以上のプロセッサ資源を必要とする．

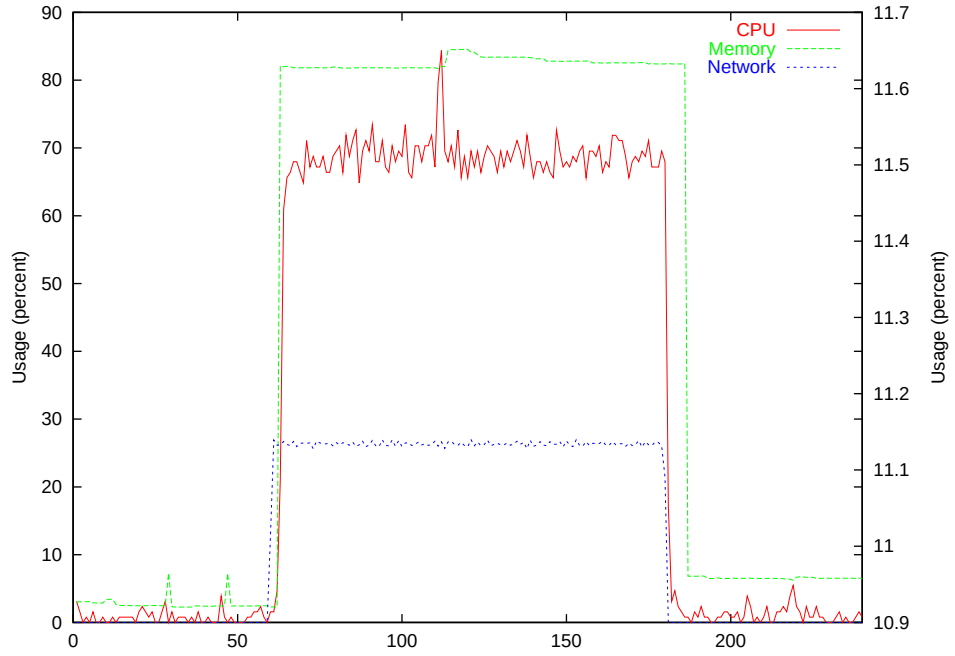

図 4.3: MPEG2-TS フォーマットでの資源使用率変化

- 非圧縮ビットマップ
  非圧縮ビットマップを用いた場合の各資源量の使用率変化のグラフを図 4.4に示す.

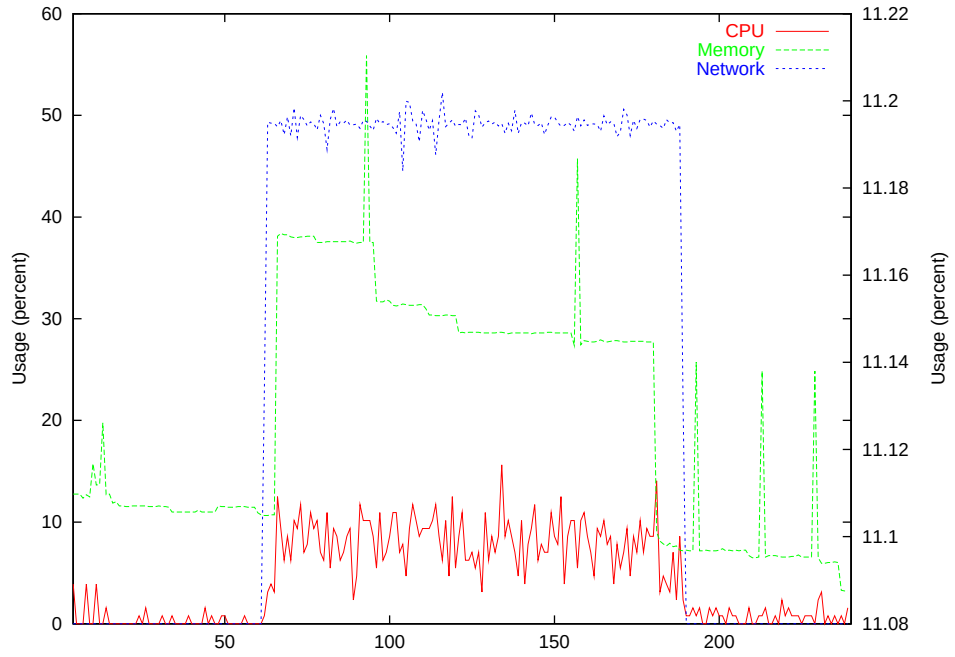

図 4.4: 非圧縮ビットマップフォーマットでの資源使用率変化

本実験では 60 秒から 180 秒にかけて送信ノードから受信ノードに対して映像・音声データの配信を行った．その直後にアプリケーションを終了させた．メモリ使用率については変動幅が小さいため，Y_2 軸を利用している．グラフより映像・音声の配信中にはすべての資源が利用されている．

実験を行った計算機では，非圧縮ビットマップフォーマットでの配信にはプロセッサ資源を 10%から 20%，メモリ資源を 1%，ネットワーク資源を 50%要した．非圧縮ビットマップフォーマットでは，映像・音声のデコーディングが不要であるためプロセッサ資源に対する要求は低いが，ネットワーク資源に対する要求が高い．

このように，映像・音声の配信においては 1) プロセッサ，2) メモリ，3) ネットワークのそれぞれの資源量が定常状態と比較して変化する．また，使用するメディアフォーマットによっても必要とする資源量が異なる．計算機がその時点で利用可能な資源量によって，使用できるフォーマットやパラメータの推測が可能である．

メディアフォーマットに依存する制御

メディア配信システムは，利用するメディアフォーマットにより制御可能なパラメータが異なる．例えば，メディアフォーマットに DV を用いる場合，フレーム内圧縮，データ量不変という特徴を持つため，圧縮率の変更ができないが，フレーム間引きによるレート制御を行うことは可能である．また，MPEG2-TS を用いる場合，圧縮率の変更やフレームレートの変更による制御はできるが，フレーム間引きによるレート制御は MPEG フォーマットの特性上行えない．

このように，フォーマットによって可能な制御と不可能な制御があるため，それぞれのメディアフォーマット可能な制御処理に優先度を設定した制御リストを用いて対応する．

4.2.2 伝送特性を用いた動的レート制御

メディア配信中のネットワークの変化を伝送特性によって検知し，メディア配信の動的なレート制御を行う．この処理では以下に挙げる 3 つの要件が求められる．

- エンドノード間でのみ取得可能な情報を基にした動的レート制御手法
- パケットロスが発生しない輻輳検知可能な動的レート制御手法
- 特定の RTP パケット伝送のために最適化された動的レート制御手法

2.1.1項よりリアルタイムアプリケーションにおける遅延やジッタは再生と密接な関係があることから，伝送特性として用いる“ 伝搬時間の揺らぎ (ジッタ) ”は計測の容易さ，有用性の観点から適切である．

本手法の有効性検証

2.1.1項に挙げた要因によりパケットの伝搬時間は変動する．ジッタの揺らぎの多くは，計算機やネットワーク状態が高負荷になり，負荷軽減処理としてバッファリングが発生することにより起こる．本項では，通信状態の変化が実際にどのような形でジッタの揺らぎとして現れるかを確認することで伝送特性としてジッタを用いた制御が可能かどうか検討する．

ジッタの発生とネットワーク状態の変化の関連性を確認するために，メディア配信システムとして DVTS を用いた実験を行う．ジッタの値を測定するために，図 4.5に示す実験ネットワークを構築した．また，擬似的に輻輳状態を発生させるため，中継ルータにおいて 35Mbps の帯域制限をかけている．

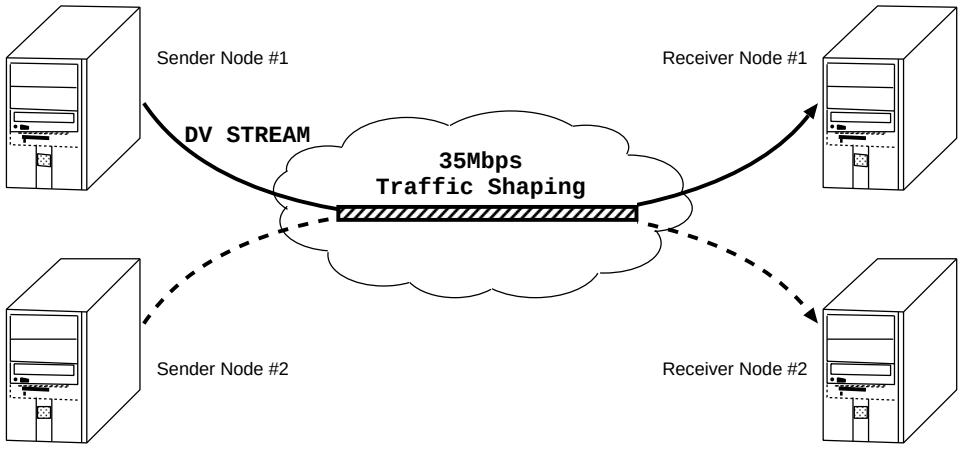

図 4.5: テスト時の環境概要図

図 4.6は，図 4.5に示したネットワークに DV ストリーム 1 本が流れている状態で，バーストトラフィックにより輻輳が発生した場合の $SenderNode\#1$ でのジッタの変動とパケットロスの値を示したグラフである．バーストトラフィックの発生には，Netperf[40] を用いた．Netperf は，パケットペアスキームを用いてネットワークの帯域幅を測定するアプリケーションであり，測定時にバーストトラフィックを発生させる．

このグラフからバーストトラフィックの発生とともにジッタは大きく正負に変動していることが確認できる．このようにジッタは，ネットワークの変化，特に帯域幅の不足や伝搬の遅延時間変化の際に大きな振れ幅となって現れる．この値を利用することで通信状態を把握できる．

先程と同様のネットワークにおいて，DV ストリームが 2 本流れたときの $SenderNode\#1$ でのジッタの変動とパケットロスの値を示したグラフを図 4.7に示す．

この場合もバースト的なトラフィックの発生ならびに収束とともにジッタが大きく正負に変動していることが確認できる．しかし，一定時間が経過するとジッタは大きな変化から小さな変化へと移行し，トラフィックが収束するまで一定以上の増減を繰り返していることも確認できる．

ジッタの値は，輻輳状態になる直前やバーストトラフィックの収束直後などネットワークの状態が急激に変化する際に大きく変動するが，継続的な輻輳状態の間やネットワーク状態が緩やかに変化する際には緩やかな変動を続ける．

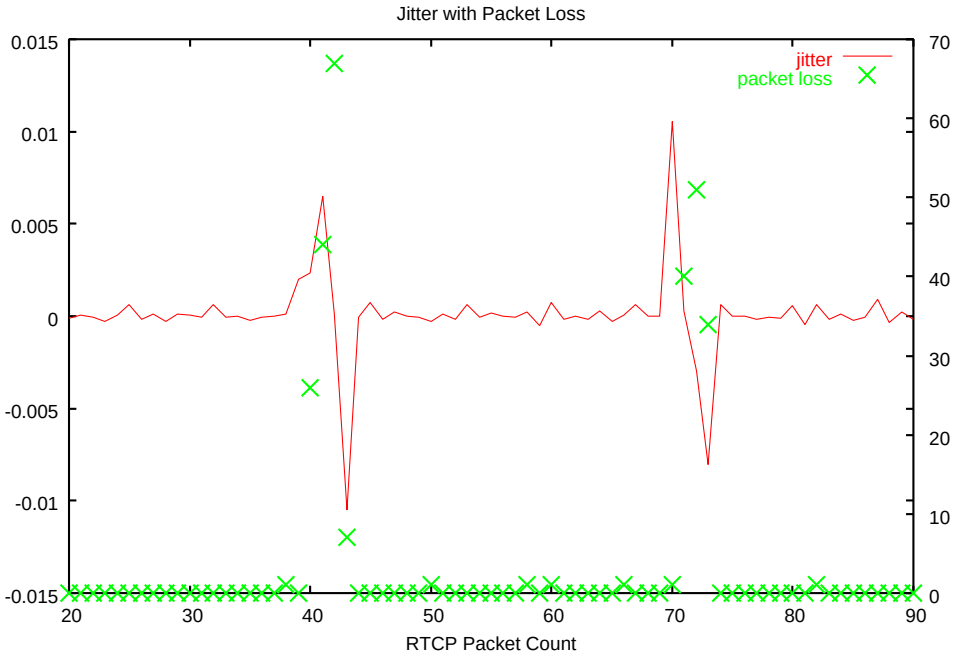

図 4.6: 輻輳時のジッタとパケットロス - DV ストリーム と Netperf

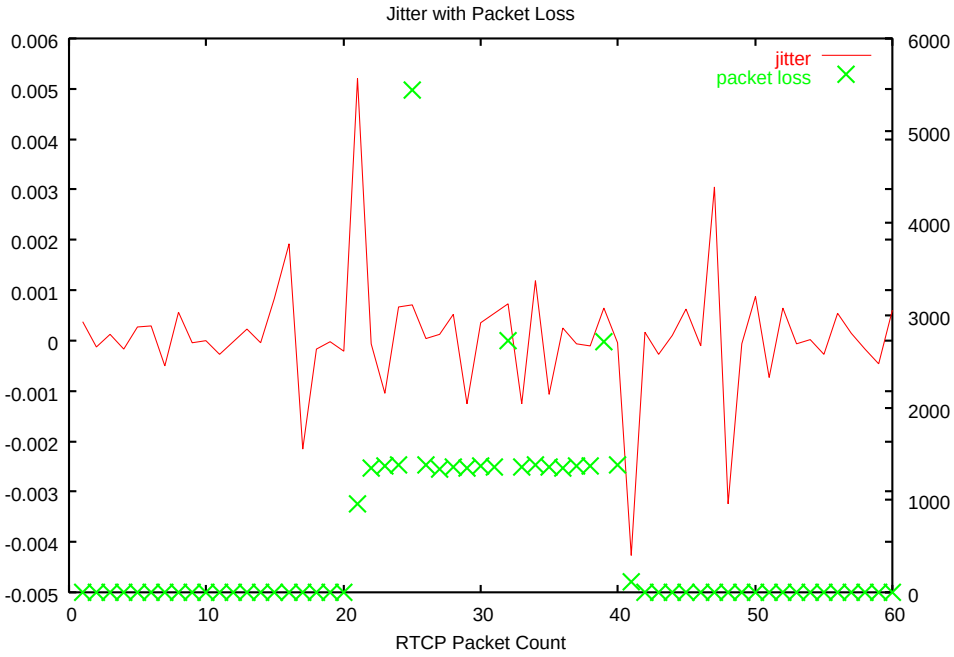

図 4.7: 輻輳時のジッタとパケットロス - DV ストリーム 2 本

ネットワークの変化を検知するためにジッタを用いることは有効であるが，継続的な輻輳状態の検知にはジッタによる判断は弱いという特性が確認された．そこで，ネットワークの変化の検知とメディアストリームのトラフィック減少制御にジッタを，その後の増加制御にはパケッ

トロスの値を用いる．図 4.8に本研究が用いるレート制御の流れを示す．レートの減少処理はジッタが許容値を超えた場合かパケットロスが許容値を超えた場合に行う．レートの増加処理はパケットロスの値が一定の間 0 であった場合に行う．

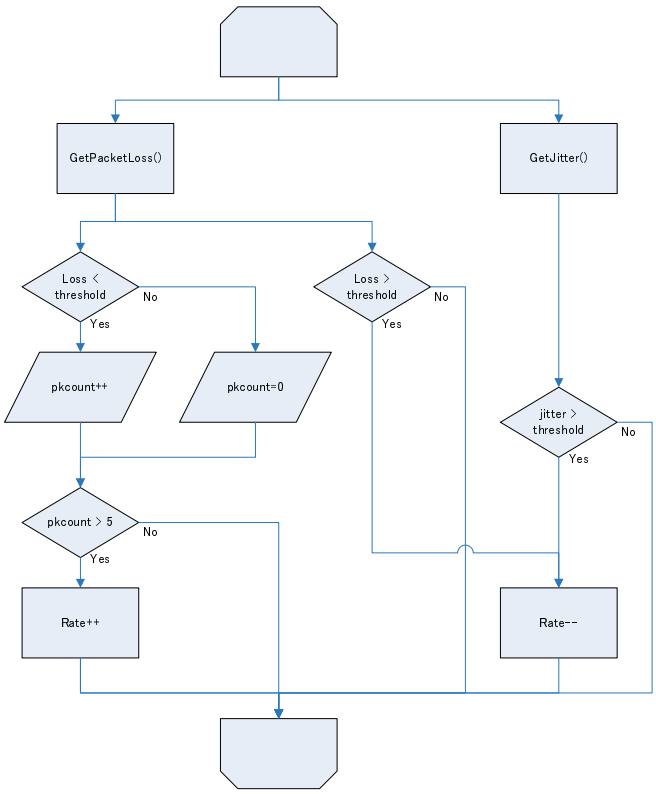

図 4.8: レート制御の流れ

4.3 本研究のアプローチ

本研究では，環境資源を用いた自律協調型メディア配信の実現のために，ネットワーク資源ならびに計算機資源の変化を基にしたメディア配信制御手法を提案する．本手法の実現のために以下に述べる 4 つの手法を確立する．

4.3.1 複数フォーマットへの対応とフォーマットスイッチング

メディア配信において送受信ノード間では利用するメディアフォーマットを統一する必要がある．多種多様なメディア配信に対応するためには，以下の 2 つのうち，いずれかに対応する必要がある．

- **複数メディアフォーマット：複数メディア配信システム**
  単一のメディアフォーマットに対応した，単一のメディア配信システムが，複数個で集合体を形成し，相互に協調動作することで複数メディアフォーマットに対応したもの．システム全体を統一的に制御できない場合，それぞれのシステムが独立して処理を行うため自律協調的なメディア配信を行うことが困難となる．

- **複数メディアフォーマット：単一メディア配信システム**
  単一のメディア配信システムが，複数のメディアフォーマットに対応したもの．システム全体を統一的に扱えるが，利用するメディアフォーマットを識別し，動的な対応ができない場合，映像・音声データを復号化できなくなる．

既存のメディア配信システムでは，複数のメディアフォーマットに対応したシステムは少ない．本研究では，複数のメディアフォーマットの送受信に対応し，必要に応じてそのフォーマットを動的に切り替えられる手法を提案する．

4.3.2 統一的な資源の指標化と状態変化の把握

ユーザの持つ環境資源は使用する計算機やオペレーティングシステムによって取得に用いる手法や取得可能なデータ形式が異なる．これらの差異を吸収するため，本研究では資源に関して統一的な指標化を行う．

本手法では，計算機の持つ環境資源値を計測時点での各資源の使用率として表現することにより，各オペレーティングシステムによって異なる資源値を持っている場合であっても，その差異を吸収可能である．本研究では，以下の 3 つの資源量に対する指標化を行う手法を提供する．

- プロセッサ使用率 (P_p)
- メモリ使用率 (P_m)
- ネットワーク使用率 (P_n)

4.1式に資源使用率の計算式を示す．この計算式によって，計測時点での他のプロセスが使用している資源量を使用率として求める．各資源の使用率は判断値の計算に用いる．

各資源ごとの資源使用率の計算

$$P = \frac{U}{A} \tag{4.1}$$

P： 資源使用率

U： 計測時点での資源の使用量

A： 使用可能な資源の総量

4.2式に本手法が用いる判断値を求める計算式を示す．メディア配信システムが使用する資源量は，利用するメディアフォーマットやプロセッサ性能などによって異なる．例えば，同一プロセッサの場合，プロセッサのクロック性能に比例してメディア配信に使用されるプロセッサ

使用率は低くなる．このため，メディア配信システムが使用する資源量の算出には，計算機の持つ各資源の性能に応じたフォーマット定数，資源計数を用いる．

判断値は，他の全プロセスが使用している資源量と本機構が必要とする資源量により求められた値とあらかじめ設定された許容値を比較し判断する．

各資源ごとの判断値の計算

$$D = P + \frac{F}{R} > T \tag{4.2}$$

D： 判断値

P： 計測時点での資源の使用率

F： フォーマット定数

R： 資源計数

T： 各資源の使用許容量

また，ネットワークの状況変化は伝送特性に表れることを考慮し，“ パケット到達間隔の揺らぎ ”ならびに“ パケット喪失数 ”についても判断値として用いる．

4.3.3 ノード間での交換と共有

各ノードで取得した資源値は，各ノードにおける状況判断に用いられる他，相手ノードに対しての要求を行う際にも利用される．取得した資源値を相手ノードに対して通知し，相互ノード間で情報の共有が行われる機構が必要である．

ノード間での情報交換と共有には HTTP などで用いられるリクエスト・レスポンスモデルを利用する．リクエスト・レスポンスモデルでは，何らかの変化が発生した場合に，必要な要求を発行するモデルである．要求発行時に要求の理由として資源値を添付し，相手ノードに処理を判断させる．

メディア配信においては RTCP を用いたネットワーク情報交換プロトコルが存在する．本手法では RTCP を用いず，独自の通知プロトコルを用いる．これにより，相手ノードにおいて用いない情報がネットワークに対して配信されないシンプルなモデルを構築可能である．

4.4 まとめ

本章では，計算機資源に適応する自律協調型メディア配信手法の提案を行った．また，メディア配信システムにおける自律協調型配信手法の適用に必要な要素条件について，ネットワーク環境ならびに計算機環境の視点から検討し，以下の項目の必要性について述べた．

- 計算機資源によるメディアフォーマットの決定・変更
- ネットワーク特性によるメディアフォーマットのパラメータ決定・変更

また，本研究の視点として，自律協調型メディア配信実現のために必要な機能を洗い出し，各機能について詳しく述べた．本研究では，配信開始時や配信中のメディア配信制御に計算機資源を利用した処理を行う．また，配信中のメディアフォーマットのパラメータ制御はネットワークの伝送特性と計算機資源量を用いた処理を行う．次章では本手法の実証例として DVTS を用いる場合の設計について述べる．

第5章　自律協調型メディア配信機構の設計

本章では環境資源に適応する自律協調型メディア配信手法の実証例として DVTS に対して適用する場合の設計について述べる．また，必要な資源の指標化ならびにノード間での共有手法の設計についても述べる．

5.1　設計概要

本研究では，提案する手法の有効性について検証するため，既存のメディア配信システムとして DVTS を用い，自律協調型メディア配信の実現に必要な各機能を実装する．

4章にて本研究のアプローチについて検討した．2章にて述べたインターネット上でのメディア配信技術，3章にて述べた既存技術の概要，4.4節にて述べた要件を満たす機能について考慮し，以下の 4 つの機能を本機構の満たすべき機能とする．

- 複数のメディアフォーマットに対応し，必要に応じた動的な切り替え機構の提供
- 1 対 1 の通信ノード間で自律協調的に環境資源の計測・情報交換機構の提供
- 環境資源の統一的な指標化を行い，ノード環境に依存しない情報定義の提供
- ノード間で協調し，必要に応じた配信制御機構の提供

図 5.1にシステム概要を示す．

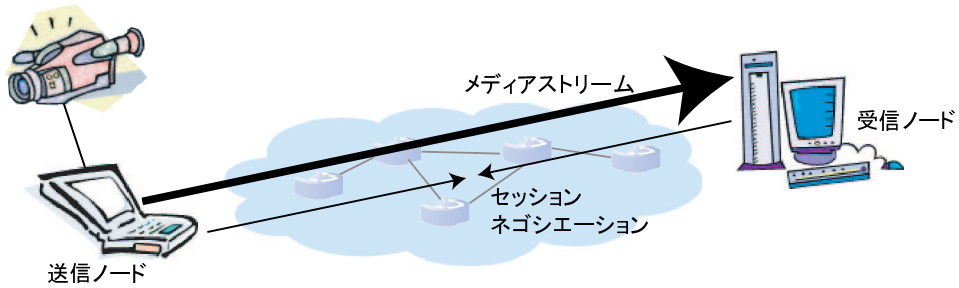

図 5.1: 全体概要図

本機構は送信ノードならびに受信ノードの 2 つの計算機により構成される．送信ノードでは複数のメディアフォーマットの送信に対応できる入力デバイスと伝送アプリケーションが実装されている．また，受信ノードでは複数のメディアフォーマットの受信に対応できる CODEC が実装されており，受信した映像・音声データは自身の持つモニタ上に表示される．計算機間は IP ネットワークを通じて，RTP パケット化した映像・音声データが配信される．

5.2 複数フォーマットへの対応

本機構では，単一のメディア配信システムにおいて複数のメディアフォーマットに対応したマルチフォーマット型のメディア配信システムを実装する．以下に挙げるメディアフォーマットに対応し，IP ネットワーク上での配信に必要な機能を実装する．

- DV フォーマット
- MPEG2 フォーマット (MPEG2-TS)
- JPEG2000 フォーマット (Motion-JPEG2000)
- RGB フォーマット

また，本機構では，送受信ノード間で利用するフォーマットの識別を，ノード間で環境資源情報を交換・共有する機構を応用することで実現する．各フォーマットの持つ解像度ならびに画質の特徴から，本機構では，フォーマットの選択に関して以下の順序で優先付けを行う．

1. MPEG2 フォーマット (MPEG2-TS)
2. DV フォーマット
3. RGB フォーマット
4. JPEG2000 フォーマット (MotionJPEG2000)

5.3 統一的な資源の指標化

本機構では，計算機資源の判別に資源計数とフォーマット計数を基にした判断値を用いる．各係数は 4.2.1項で行った実験を複数の計算機環境で実施し，各資源の使用率を求め，得られた結果を基に計算により各資源の判断値が算出できる値を割り出した．

5.3.1 プロセッサ資源の指標化

本機構ではプロセッサ資源の判断に 5.2式に示す判断値を用いる．

メディア配信に利用されるプロセッサ使用率は，プロセッサ種別と動作クロックに比例して低くなる．このため，プロセッサに応じた資源計数を 5.1式に挙げる式により求める．

プロセッサ資源計数の計算

$$R_p = E_p \times C \tag{5.1}$$

R_p： プロセッサ資源計数

E_p： プロセッサ種別による資源定数

C： プロセッサの動作クロック (GHz)

表 5.1に Intel 製プロセッサにおける資源定数を示す.

表 5.1: プロセッサ種別による資源定数一覧 (E_p)

CPU 種別	定数
Pentium3	4.0
Pentium4	1.8
PentiumM	3.9
Xeon(Pentium4)	1.8

プロセッサの判断に用いる値は 5.2式を用いて求める．プロセッサ使用率は計測時点での他プロセスによるプロセッサ使用状況，フォーマット定数は表 5.2 に挙げる値，プロセッサ資源計数は 5.1式にて求めた値を用いる.

プロセッサ判断値の計算

$$D_p = U_p + (\frac{F_p}{R_p}) > T_p \tag{5.2}$$

D_p： 資源判断値

U_p： プロセッサ使用率

F_p： フォーマット定数

R_p： プロセッサ資源計数

T_p： プロセッサ資源の使用許容量

表 5.2: メディアフォーマットによるフォーマット定数一覧 (F_p)

フォーマット	定数
DV	350
MPEG2	438
JPEG2000	500
RGB	88

5.3.2 メモリ資源の指標化

本機構では，メモリ資源の判断時に 5.4式に示す資源判断値を用いる.

メモリ使用率は，プロセッサの持つクロックに比例して低くなる．このため，判断値の計算に用いる資源計数は 5.3式に挙げる式により求める.

メモリ資源計数の計算

$$R_m = E_m \times C \tag{5.3}$$

R_m： メモリ資源計数

E_m： プロセッサ種別による資源定数

C： プロセッサの動作クロック (GHz)

表 5.3に Intel 製プロセッサにおける資源定数を示す.

表 5.3: プロセッサ種別による資源定数一覧 (E_m)

CPU 種別	定数
Pentium3	0.4
Pentium4	0.18
PentiumM	0.39
Xeon(Pentium4)	0.18

メモリの判断に用いる値は 5.4式を用いて求める．メモリ使用率は計測時点での他プロセスによるメモリ使用状況，フォーマット計数は表 5.4に挙げる値，メモリ資源計数は 5.3式にて求めた値を用いる.

メモリ判断値の計算

$$D_m = U_m + (\frac{F_m}{R_m}) > T_m \tag{5.4}$$

D_m： 資源判断値

U_m： メモリ使用率

F_m： フォーマット定数

R_m： メモリ資源計数

T_m： メモリ資源の使用許容量

5.3.3 ネットワーク資源の指標化

ネットワーク資源の判断時に 5.5式に示す資源判断値を用いる.

表 5.4: メディアフォーマットによるフォーマット定数一覧 (F_m)

フォーマット	定数
DV	0.9
MPEG2	1.2
JPEG2000	1.5
RGB	0.3

$$D_n = U_n + F_n > T_n \tag{5.5}$$

D_n： ネットワーク資源判断値

U_n： ネットワーク使用率

F_n： フォーマット定数

T_n： ネットワーク資源の使用許容量

表 5.5: メディアフォーマットによるフォーマット定数一覧 (F_n)

フォーマット	定数
DV	35
MPEG2	28
JPEG2000	10
RGB	50

また，ネットワーク状態変化を把握する指標として伝送特性を用いる．伝送特性はパケット到達間隔の揺らぎならびにパケット喪失数を用い，それぞれ以下のように計測を行う．

パケット喪失数計測処理

RTP ヘッダには RTP パケットの統計情報を求めるために必要な情報が含まれている．本処理では 5.6式のように RTP ヘッダに含まれる RTP シーケンス番号の差分を求めることでパケット喪失数を計測する．

$$L = S_j - S_i \tag{5.6}$$

L： パケット喪失数

S_n： 時刻 n 時点でのシーケンス番号 ($j > i$)

パケット到達間隔計測処理

本処理では 5.7式のようにパケット到達時間の差分を求めることでジッタの値を計測する．また，時間情報に正負の情報を付加することで純粋な伝送時間の変化も取得する．

$$J = (R_j - S_j) - (R_i - Si) \tag{5.7}$$

J： パケット到達間ジッタ

R_n： n 時点でのパケット到達時刻 $(j > i)$

S_n： n 時点でのパケット送出時刻 $(j > i)$

5.3.4 資源の記述手法

各環境資源の記述には拡張 SDP 記述を用いる．本機構における拡張 SDP 記述を以下に示す．本記述を資源 SDP 記述と定義する．

```
v=0
o=three 2890844526 2890842807 IN IP4 203.178.143.40
s=0
c=IN IP4 203.178.143.40
t=0 0
a=processor:43
a=memory:34
a=network:23
m=audio 49170 RTP/AVP 0
m=video 51372 RTP/AVP 31
```

図 5.2: 拡張 SDP メッセージ

各環境資源の情報は SDP のセッションレベル属性として定義され，a(attribute) フィールドを用いて記述する．値は求められた資源判断値を利用する．この例では，プロセッサ資源として 43，メモリ資源として 34，ネットワーク資源として 23 がそれぞれ記述されている．また，ノードが利用可能なメディアフォーマットに関する情報を m フィールドとして記述する．SDP 記述のうち，[v,o,s,t] フィールドは利用しない．しかし，SDP 記述においてこれらのフィールドは必須パラメータであるため暫定的な値を入れ記述する．

5.4 ノード間での交換と共有

本機構では，環境資源情報やその他接続に必要な情報を送受信ノード間で交換する．交換には，HTTP のリクエスト・レスポンスを用いた通信を応用する．

情報の交換は表 5.6に挙げるリクエストメソッドを用いて行われる．本機構では相手ノードに対する要求処理を“ リクエスト ”，相手ノードからの要求処理に対する返答処理を“ レスポンス ”とそれぞれ定義する．リクエストには必ず何らかの要求を示す“ メソッド ”と行うべき処理に必要な情報を記述した“ メッセージ ”が記述される．レスポンスには相手ノードにて要求した処理を実行した結果が記述されるとともに，相手ノードにおいて必要な情報が“ メッセージ ”として追加的に記述される．

表 5.6: リクエストメソッド一覧

要求	処理内容	利用可能ノード
REGISTER	初期ノードへの参加要求	送，受
NODELIST	初期ノードへのノードリストの配信要求	送，受
START	メディア配信の開始要求	送，受
CAPEX	ノード能力チェック要求	受
RESTART	メディア配信の再開要求	送，受
CHGFMT	フォーマットスイッチング要求	受
CHGCFG	フォーマット設定変更要求	受
STOP	送信停止要求	受
LEAVE	ノード離脱要求	送，受

5.4.1 リクエスト

図 5.3に記述されるリクエストメソッドの例を示す．リクエストは 1 行ごとに情報が記述され，改行コード (CR+LF) で区切られる．

```
REQUEST AMSS/1.0 [CR+LF]
To: 218.45.27.9:8002 [CR+LF]
From: UserA <userA@domain.tld> [CR+LF]
Call-ID: 8487294571@domain.tld [CR+LF]
Contact: 211.10.16.44:8002 [CR+LF]
Content-Type: application/sdp [CR+LF]
Content-Length: 203 [CR+LF]
[CR+LF]
SDP メッセージ または メッセージ [CR+LF]
[CR+LF]
```

図 5.3: リクエストメソッド

リクエストは 1 行目に必ず相手に対して希望する要求を記述する．この行を“ リクエストライン ”と呼ぶ．2 行目以降，空白行が来るまでを“ リクエストヘッダ ”と呼ぶ．リクエストヘッダではリクエストを行ったノードの情報やリクエストに必要な情報が記述される．また，相手ノードが行うべき処理に必要な情報が存在する場合，空白行の次の行から記述する．これを“ メッセージボディ ”と呼び，SDP メッセージないしは特定のメッセージを記述する．リクエストには，リクエストライン，リクエストヘッダ，メッセージボディのすべてが必要である．

メッセージボディに記述される SDP メッセージは SDP に定義されている記述を拡張したものを用いる．リクエストに利用される拡張 SDP 記述を以下に示す．本定義をリクエスト SDP 記述と呼ぶ．

```
v=0
o=three 2890844526 2890842807 IN IP4 203.178.143.40
s=0
c=IN IP4 203.178.143.40
t=0 0
m=audio 49170 RTP/AVP 0
m=video 51372 RTP/AVP 31
a=dvdif:13
a=jitter:-0.231
```

図 5.4: 拡張 SDP メッセージ

各環境資源の情報は，SDP のメディアレベル属性として定義され，a(attribute) フィールドを用いて記述する．この例では，ジッタの変動に対応してビデオフォーマットの設定を 13 に変更をを行う要求の記述である．資源 SDP 記述と同様に，[v,o,s,t] フィールドは利用しない．しかし，SDP 記述においてこれらのフィールドは必須パラメータであるため暫定的な値を入れ記述する．

5.4.2 レスポンス

図 5.5に記述されるメッセージの例を示す．リクエスト同様，レスポンスも 1 行ごとに情報が記述され，改行コード (CR+LF) で区切られる．

レスポンスは 1 行目に必ずリクエストに対する処理の結果を記述する．この行を“ ステータスライン ”と呼ぶ．ステータスラインでは，表 5.7に示す“ レスポンスコード ”が記述され，相手ノードでは，この値を基に処理結果を判断する．2 行目以降，空白行が来るまでを“ ヘッダフィールド ”と呼ぶ．ヘッダフィールドではリクエストを行ったノードの情報やレスポンスを行うノードの情報が記述される．また，レスポンスに加えて何らかの情報をリクエスト元に送信する場合，空白行の次の行から記述する．これを“ メッセージボディ ”と呼び，本機構では特定のメッセージを記述する．レスポンスは，ステータスラインならびにヘッダフィールドが必須項目，メッセージボディがオプション項目である．

```
AMSS/1.0 200 OK [CR+LF]
To: 218.45.27.9:8002 [CR+LF]
From: UserA <userA@domain.tld> [CR+LF]
Call-ID: 8487294571@domain.tld [CR+LF]
Contact: 218.45.27.9:8002 [CR+LF]
Content-Type: application/sdp [CR+LF]
Content-Length: 120 [CR+LF]
[CR+LF]
メッセージ [CR+LF]
[CR+LF]
```

図 5.5: レスポンスメッセージ

表 5.7: 応答 (レスポンス) コード一覧

応答コード	内容	詳細
1xx	暫定レスポンスまたは情報	リクエスト処理中または未完了
2xx	成功	リクエスト成功
4xx	クライアントエラー	リクエストにエラーがあり失敗，再試行は可能
6xx	グローバルエラー	リクエストの処理に失敗，再試行も不能

5.4.3 各リクエスト メソッド

本項では，本機構が定義するリクエストメソッドについて，それぞれの詳細について述べる．各リクエストメソッドならびにそれに対するレスポンスにおいて記述の必要なメッセージがある場合はあわせて述べる．

REGISTER リクエスト /NODELIST リクエスト

新たにメディア配信を開始する場合，新たに参加するノードは相手先ノードのネットワーク的な位置を特定する必要がある．本機構では，ノードの特定のために，REGISTER リクエストと NODELIST リクエストを定義する．新規参加ノードは，特定のノード (初期ノード) に対して REGISTER リクエストを発行し，NODELIST リクエストを発行することで，接続可能な全ノードリストを取得できる．

初期ノードは，新規参加ノードからの NODELIST リクエストを受け，レスポンスメッセージとして，自身の保持するノードリストを返す．レスポンスメッセージではそのノードが受信専用ノードか送受信可能ノードかという情報も返す．その際のレスポンスメッセージの例を以下に示す．

```
ノードの IP アドレス:リクエスト待ち受けポート番号:ノード種別 (0-受信専用/1-送受信)
211.10.16.44:8002:0
218.45.27.11:9002:1
203.178.143.40:8002:1
```

図 5.6: ノードリスト

START リクエスト /RESTART リクエスト /STOP リクエスト

要求元ノードが要求先ノードに対してメディア配信の開始・再送・停止を要求するためのリクエストメソッドとして，START，RESTART，STOP リクエストをそれぞれ定義する．STOP リクエストは受信ノードから送信ノードに対してのみ発行できる．本要求を受けた要求先ノードは，そのリクエストの内容に応じて，必要な処理を実行する．

本要求には SDP メッセージならびにリクエストメッセージは付与されない．また，レスポンスメッセージも付与されない．

CAPEX リクエスト

ノードの持つ資源値を交換するためのリクエストメソッドとして，CAPEX リクエストを定義する．CAPEX リクエストは受信ノードから送信ノードに対してのみ発行できる．CAPEX リクエストには，受信ノードの持つ資源値ならびに利用可能なメディアフォーマット情報を資源 SDP 記述として，リクエストメッセージ内に記述する．

この処理により，送信ノードが受信ノードに対して送信を行うべきメディアフォーマットの決定，受信ノードの待ち受け状態 (IP アドレス，ポート番号) の把握が可能である．

CHGFMT リクエスト /CHGCFG リクエスト

計算機やネットワークの状況に応じて，送信ノードに対して設定の変更やフォーマットスイッチングを要求するためのリクエストメソッドとして，CHGFMT，CHGCFG リクエストを定義する．これらの要求は，受信ノードから送信ノードに対してのみ発行できる．

CHGFMT リクエストには，受信ノードの持つ資源値ならびに変更希望のメディアフォーマット情報を資源 SDP 記述として，リクエストメッセージ内に記述する．CHGFMT リクエストを受けた送信ノードでは，フォーマットスイッチング処理を行い，その結果についてレスポンスメッセージを返す．ただし，レスポンスコード以外にメッセージを記述しない．

CHGCFG リクエストには，受信ノードの持つ資源値ならびに変更希望のメディアフォーマットに対するパラメータをリクエスト SDP 記述として，リクエストメッセージ内に記述する．CHGCFG リクエストを受けた送信ノードでは，パラメータの変更処理を行い，その結果についてレスポンスメッセージを返す．ただし，レスポンスコード以外にメッセージを記述しない．

5.5 メディア配信システムへの適用

本節では，環境資源に応じた自律協調型メディア配信手法の実証例として DVTS に対して適用する場合の設計について述べる．5.5.1項では本機構の全体構成，5.5.2項では本機構の動作概要について述べる．

5.5.1 全体構成

本機構を構成するモジュールを以下にまとめる．

- **送信ノード**
  - **RTP カプセル化/IP カプセル化モジュール**
    受信した映像・音声データをネットワークを介して送信可能にするため，RTP カプセル化ならびに IP カプセル化を行う．本モジュールは DirectShow フォーマット名/RTP 送信フィルタと呼ぶ場合がある．
  - **システム資源計測モジュール**
    送信ノードにおける計算機資源の増減を監視し，必要に応じてリクエスト・レスポンス制御モジュールにレポートする．
  - **リクエスト・レスポンス制御モジュール**
    受信ノードからのリクエスト受付ならびに送信ノードから受信ノードへのリクエスト発行を行い，必要に応じてカプセル化モジュールの制御を行う．
- **受信ノード**
  - **RTP 脱カプセル化/IP 脱カプセル化モジュール**
    受信した IP パケットから映像・音声データを取得し，出力デバイスに送信する．本モジュールは DirectShow フォーマット名/RTP 受信フィルタと呼ぶ場合がある．
  - **システム資源計測モジュール**
    受信ノードにおける計算機資源の増減を監視し，必要に応じてリクエスト・レスポンス制御モジュールにレポートする．
  - **ネットワーク資源計測モジュール**
    受信ノードにおけるネットワーク資源の増減を監視し，必要に応じてリクエスト・レスポンス制御モジュールにレポートする．
  - **リクエスト・レスポンス制御モジュール**
    送信ノードからのリクエスト受付ならびに受信ノードから送信ノードへのリクエスト発行を行い，必要に応じてカプセル化モジュールの制御を行う．

5.5.2 動作概要

本機構における各モジュールの関係及び動作概要を図 5.7に示す.

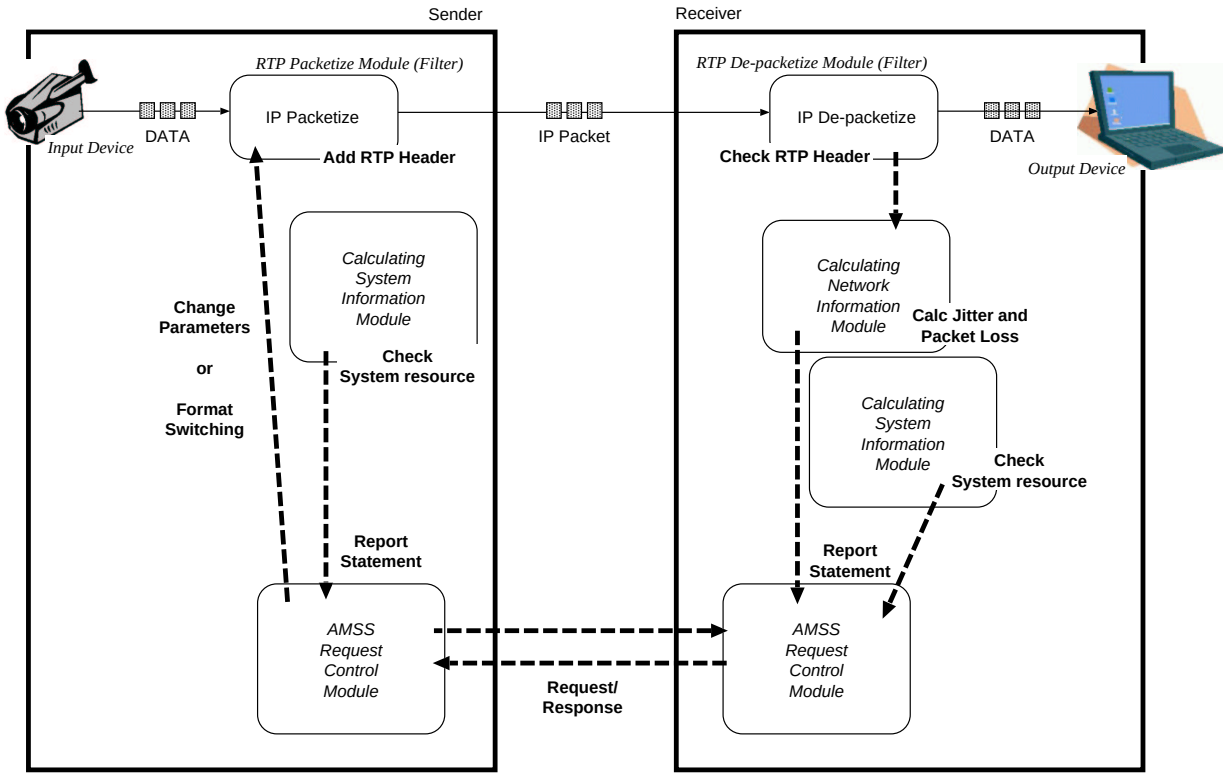

図 5.7: 各モジュール間関係概要

本手法は Microsoft Windows オペレーティングシステム上で実装するものとし，図に示したモジュールで構成される．システム資源計測モジュールならびにネットワーク資源計測モジュールは，ウィンドウメッセージを用いて，リクエスト・レスポンス制御モジュールに情報を通知し，相手ノードに対して必要な処理を要求する．送信ノードではリクエスト・レスポンス制御モジュールからカプセル化フィルタに対してメッセージ通知を行うことで，動的なフォーマットスイッチングならびにパラメータ変更を実現する．また，リクエスト・レスポンス制御モジュールならびにカプセル化・脱カプセル化モジュールは相互に IP ネットワークを通じてデータの送受信を行う.

また，本機構におけるリクエスト・レスポンスの流れを図 5.8に示す．

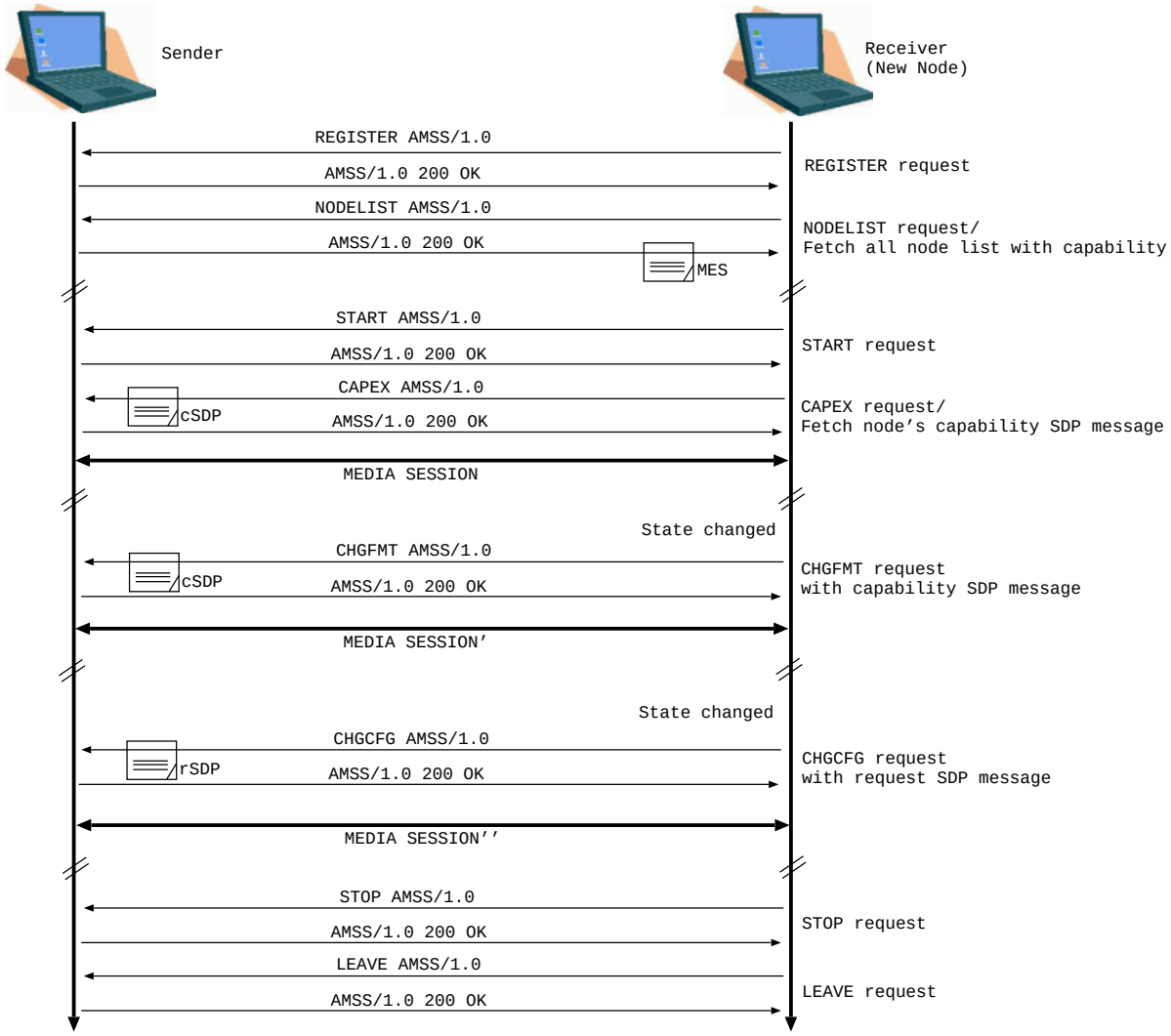

図 5.8: リクエスト・レスポンスの流れ

送受信ノードは，以下に示す流れに従いメディア配信を開始する．送信ノードは，図中左側の計算機であり，メディア配信における初期ノードを兼ねる．受信ノードは，図中右側の計算機であり，メディア配信に対して新たに参加するノードとなる．

1. 新たに送受信に参加するノードは初期ノードに対して登録要求 ($REGISTER$ 要求) を発行する
2. さらに初期ノードに対して保持するノードリスト要求 ($NODELIST$ 要求) を発行する
3. 初期ノードからノードリストが返答される
4. 受信ノードから送信ノードに対して送信開始要求 ($START$ 要求) を発行する
5. 受信ノードから送信ノードに対して資源チェック要求 ($CAPEX$ 要求) を発行する
6. 送受信ノード間で利用するメディアフォーマットを確定する
7. メディア配信を開始する

メディア配信を行っている間に計算機やネットワーク状態が変化した場合は，$CHGCFG$ 要求ないしは $CHGFMT$ 要求を発行し，状態変化にあわせたメディア配信を継続させる．

また，メディア配信の停止を要求する場合は受信ノードから送信ノードに対して $STOP$ 要求を発行する．送受信ノードから離脱する場合は初期ノードに対して $LEAVE$ 要求を発行する．送信ノードにおいて送受信時に行われる処理の状態遷移図を図 5.9に示す．

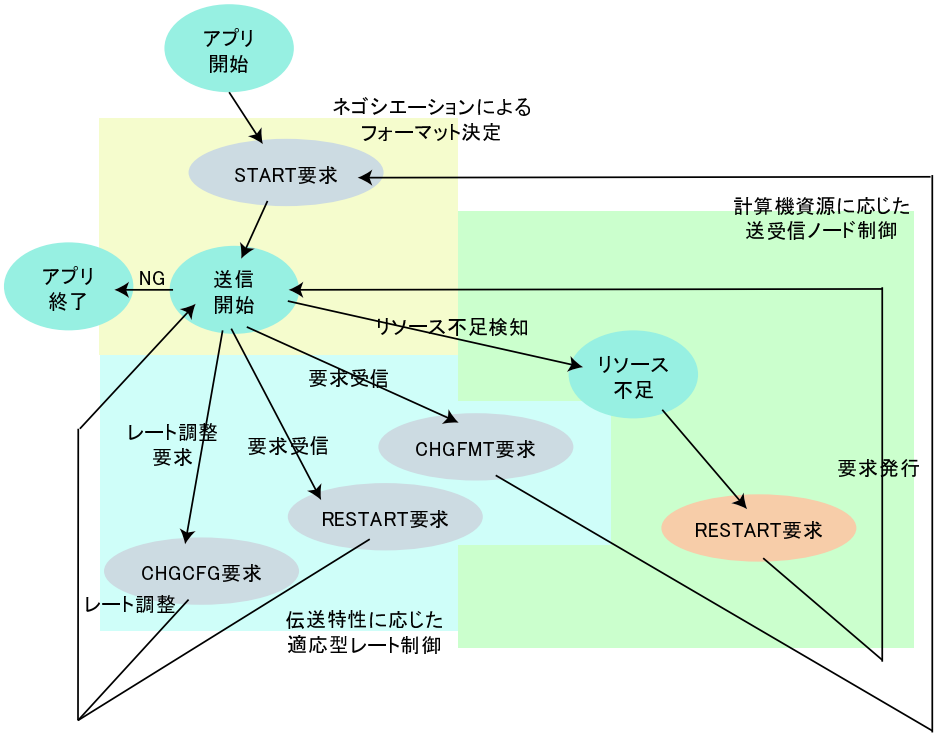

図 5.9: 送信ノードでの処理のモデル

送信ノードでは基本的に受信ノードからのリクエストを受けて処理が行われる．送信開始後，送信ノードにおける計算機資源が枯渇してきた場合にのみ，受信ノードに対して再送信要求を発行できる．

受信ノードにおいて送受信時に行われる処理の状態遷移図を図 5.10に示す.

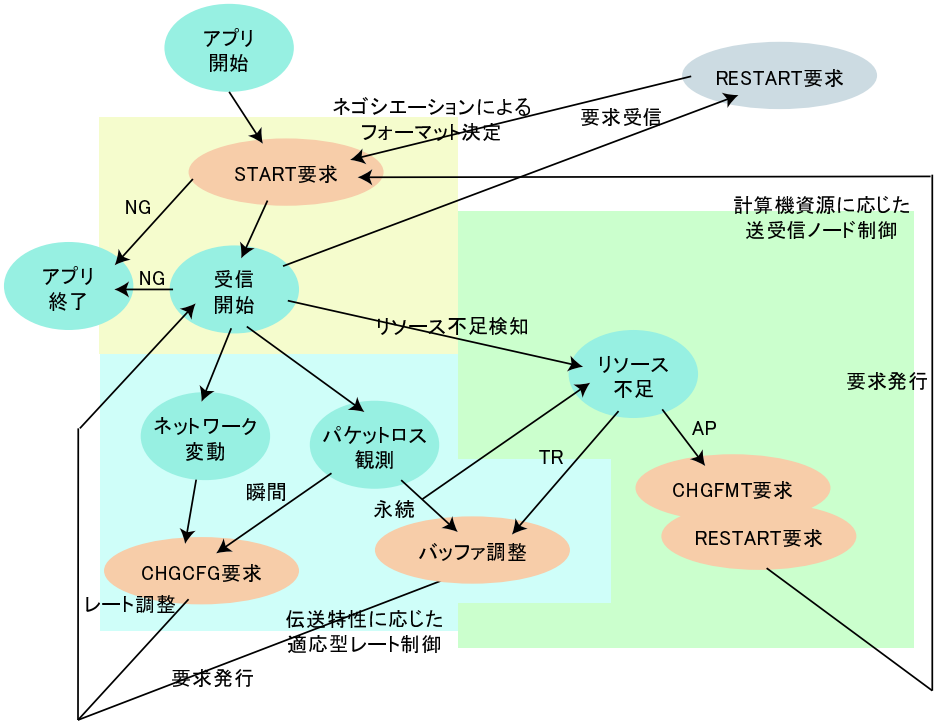

図 5.10: 受信ノードでの処理のモデル

受信ノードでは，ネットワークならびに計算機の状態に応じて，送信ノードに対してリクエストを発行することで必要な処理を実行する．ネットワークの変動は，ネットワーク使用率，パケット喪失率，ジッタの変動を基に判断し，変動が瞬間的に終了するか，しばらくの間永続するかで必要な処理を変更する．また，計算機状態の変動に応じて，フォーマットの変更，パラメータ値の変更を行う.

5.5.3 フォーマットに依存した制御処理

メディアフォーマットの特性により行うことのできる制御処理は異なる．それぞれのフォーマットの制御に関して，フォーマットに依存した処理について述べ，本手法で行うべき制御処理についてまとめる.

DV

- **セッション開始時の制御**
  環境資源により DV フォーマットを利用可能かどうかの判断を行い，可能な場合，送信を開始する.
- **セッション中の制御**
  伝送特性に応じたフレームレート制御 (間引き率の変更) を動的に行う．環境資源に応じたフォーマットの切り替えを動的に行う.

MPEG2-TS(HDV)

- **セッション開始時の制御**
  環境資源により MPEG2-TS フォーマットを利用可能かどうかの判断を行い，可能な場合，送信を開始する
- **セッション中の制御**
  環境資源に応じたフォーマットの切り替えを動的に行う．

Motion-JPEG2000

- **セッション開始時の制御**
  環境資源により Motion-JPEG2000 フォーマットを利用可能かどうかの判断を行い，可能な場合，送信を開始する
- **セッション中の制御**
  環境資源に応じたフォーマットの切り替えを動的に行う．

非圧縮ビットマップ

- **セッション開始時の制御**
  環境資源により非圧縮ビットマップフォーマットを利用可能かどうかの判断を行い，可能な場合，送信を開始する
- **セッション中の制御**
  伝送特性に応じたフレームレート制御 (間引き率の変更) を動的に行う．環境資源に応じたフォーマットの切り替えを動的に行う．

5.6 まとめ

本章では，環境資源に適応する自律協調型メディア配信手法を実現するために必要な設計要件を挙げ，実装に必要な設計を行うとともに，本手法を DVTS に対して適用するための設計を行った．

第6章　自律協調型メディア配信機構の実装

本章では，設計に基づいて実装を行った自律協調型メディア配信機構 AMSS (Adaptive Media Streaming System) について述べる．6.1節では実装概要と実装を行った環境について述べる．6.2節以降では本機構の各部分における詳細な実装について述べる．

6.1　実装概要

本研究では，インターネット上でメディア配信を行うシステムである DVTS に対して，複数メディアフォーマット処理機能を付加した上で，自律協調型メディア配信を行う AMSS(Adaptive Media Streaming System) を新たに実装した．

AMSS は以下の機能を有し，受信部ではメディア配信の状況監視と送信部とのネゴシエーション機能，送信部では受信部とのネゴシエーション機能をそれぞれ有する．

- 複数メディアフォーマットの動的切り替え機能
- 配信状況の監視機能
- 配信状況に応じたネゴシエーション機能
  - インフラストラクチャ(ネットワーク)
  - システム

また，5.5.1項より本機構は以下のモジュールを持つものとする．

- RTP/IP パケットカプセル化・脱カプセル化モジュール
- システム資源計測モジュール
- ネットワーク資源計測モジュール
- リクエスト・レスポンス制御モジュール

6.1.1　実装環境

本機構の実装を行ったソフトウェア環境を表 6.1に示す．
本機構の実装を行った機器環境を表 6.2に示す．

表 6.1: 実装ソフトウェア環境

OS	WindowsXP Professional + SP2
プログラミング言語	C++ (Microsoft VisualC++ .NET 2003)
ライブラリ	MFC (Microsoft Foundation Class) DirectX9 SDK, DirectShow9 WindowsSDK (Windows 2003 Server)

表 6.2: 実装ハードウェア環境

CPU(1)	Intel Pentium M 1.1GHz
メモリ (1)	1024MB
ハードディスク (1)	80GB
NIC(1)	100Base-TX
CPU(2)	Intel Xeon Processor 3.2GHz
メモリ (2)	2048MB
ハードディスク (2)	73GB
NIC(2)	100Base-TX, 1000Base-T
映像・音声機器 (1)	Victor HDV カメラ GR-HD1
映像・音声機器 (2)	Sony HDV カメラ HVR-Z1
映像・音声機器 (3)	Logicool USB カメラ QuickCam Pro for Notebook

6.1.2 実装アプリケーション

本研究のプロトタイプ実装を行ったアプリケーションのスクリーンショットを図 6.1に示す．アプリケーションウィンドウは，3 つのコントロールグループと 2 つのステータスグループで構成される．それぞれのグループは以下の機能を有している．

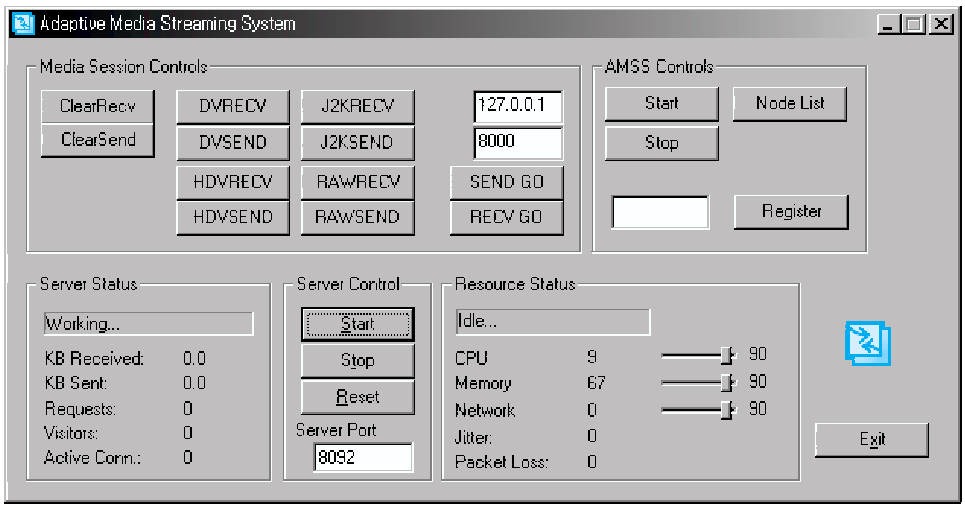

図 6.1: AMSS スクリーンショット

- **Media Session Controls グループ**
  メディア配信をコントロールする．各ボタンでは，受信・送信に機能を分け，それぞれの RTP/IP パケットカプセル化・脱カプセル化モジュールのコントロールが可能である．このコントロールグループは，AMSS が自律的に処理する際に用いるほか，必要に応じて手動操作を行う場合に利用される．

- **Server Controls グループ**
  AMSS のリクエスト・レスポンス制御モジュールのうち，リクエストの受付を行うサーバ機能のコントロールを行う．各ボタンではサーバ機能の開始・停止・再始動などを行える．

- **AMSS Controls グループ**
  AMSS の全機能のコントロールを行う．AMSS の持つ機能はここから利用開始する．初期ノードへの REGISTER 要求や LEAVE 要求などを行える．

- **Server Status グループ**
  AMSS のリクエスト・レスポンス制御モジュールのうち，リクエストの受付を行うサーバ機能のステータスを表示する．リクエストを受け付けると関連するステータス情報が更新される．

- **Resource Status グループ**
  各資源計測モジュールから計測された情報を表示する．また，各資源量に基づく処理に利用される Threshold 値の設定もここから行う．

6.2 アプリケーションのクラス一覧と処理の流れ

プロトタイプ実装を行うアプリケーションは以下のクラスを持つ．それぞれのクラスの機能について述べる．

- **CAmssServerApp クラス**
  アプリケーション本体を構成する．他クラスの構築・初期化を行い，アプリケーションウィンドウを表示させる．CWinApp クラスから派生する．

- **CAmssServerDlg クラス**
  アプリケーションウィンドウを構成する．CAmssServerApp クラスから構築される．本クラス内に AMSS を構成するすべてのモジュールの呼び出し関数が含まれる．また，メディア配信システムに利用する DirectShow モジュールもこのクラスから構築される．CDialog クラスから派生する．

- **CGenericServer クラス**
  AMSS のリクエスト・レスポンス制御モジュールのうち，リクエストの受付を行うサーバ機能を構成する．リクエスト・レスポンス処理を行うために必要なネットワークシステム部が含まれる．

- **CAmssRRServer クラス**
  CGenericServer クラスから派生する．リクエスト・レスポンス処理を行う機能のうち，AMSS 独自の処理を行うために必要な関数が含まれる．

6.2.1 送信ノードでの処理の流れ

送信ノードにおける AMSS に関係する関数とその処理の流れを図 6.2示す．送信ノードでは，受信ノードに対するメディア配信を行う機能の他，リクエスト・レスポンス処理を行う機能が含まれる．

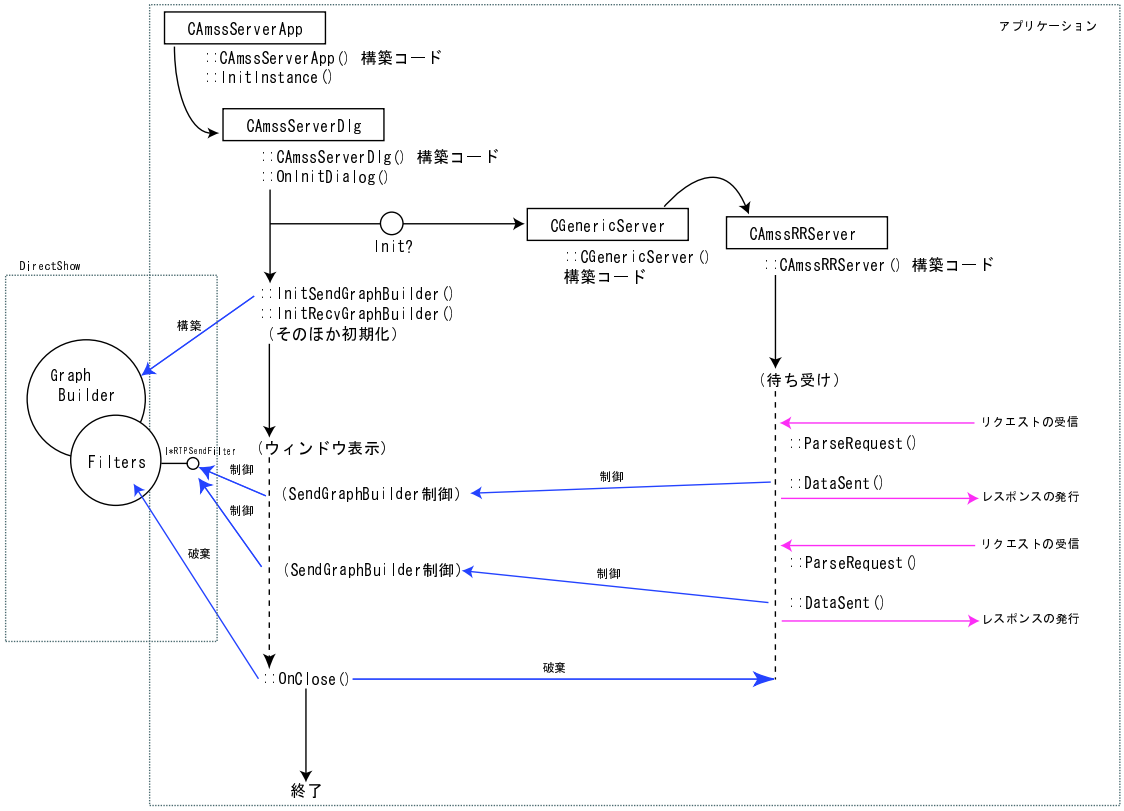

図 6.2: 送信ノードにおける処理の流れ

6.2.2 受信ノードでの処理の流れ

受信ノードにおける AMSS に関係する関数とその処理の流れを図 6.3に示す．受信ノードでは，送信ノードから受信したメディアデータの処理を行い，システムとネットワークに関する資源状態を計測，リクエスト・レスポンス制御モジュールに通知する機能が含まれる．また，送信ノード間とのネゴシエーション機能も含まれる．

6.3 各モジュールの実装

6.3.1 RTP/IP パケットカプセル化・脱カプセル化モジュール

本モジュールでは映像・音声データをインターネットを介して配信するために必要な RTP/IP パケットへのカプセル化ならびに脱カプセル化を行う．本モジュールは，1 フォーマットに対して，2 モジュール (送信用モジュール，受信用モジュール) から構成される．フォーマット依存部

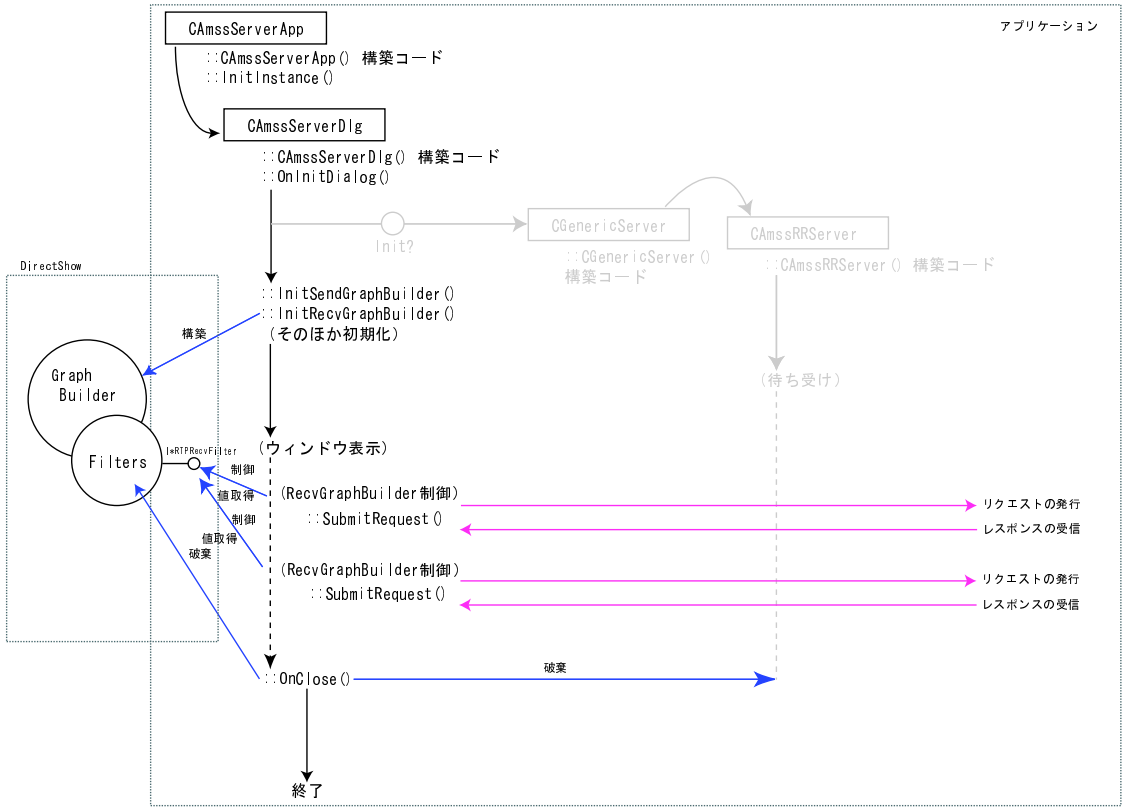

図 6.3: 受信ノードにおける処理の流れ

分を配信アプリケーションから隠蔽するため，アプリケーションの操作に必要なインタフェースのみを共通な呼び出し関数として定義し，抽象化を行った．各モジュールは，次のインタフェースを定義し，アプリケーション側に提供する．

- **送信開始に必要なパラメータ設定**
  送信先の IP アドレス，送信先のポート番号
- **受信開始に必要なパラメータ設定**
  受信を行うポート番号
- **送信パラメータ設定**
  フォーマットごとに定義される送信中に行うことのできるパラメータ設定値

送信開始に必要なパラメータ設定

送信開始に必要なパラメータを設定するため，各モジュールは以下のインタフェース関数 ($C*RTPSendFilter::SetDestination$) を持つ．この関数を用いることにより，アプリケーションから送信先情報を取得し，受信ノードに対して特定のデータを送信できる．また，本関数を呼び出さない場合，送信先 IP アドレスは 127.0.0.1，送信先ポート番号は 8000 となる．

```
STDMETHODIMP C*RTPSendFilter::SetDestination(char *pDestAddr, int DestPort)
{
    if (pDestAddr == NULL || DestPort == NULL) return E_UNEXPECTED;

    m_DestAddr.sin_addr.S_un.S_addr = inet_addr(pDestAddr);
    m_DestAddr.sin_port = htons( DestPort );
    return S_OK;
}
```

図 6.4: $C*RTPSendFilter :: SetDestination$ 関数の実装

受信開始に必要なパラメータ設定

受信開始に必要なパラメータを設定するため，各モジュールは以下のインタフェース関数 ($C*RTPRecvFilter :: SetPortNumber$) を持つ．この関数により，アプリケーションから待ち受けポート番号を取得し，データの受信を開始できる．また，本関数を呼び出さない場合，受信待ち受けポート番号は 8000 となる．

```
STDMETHODIMP C*RTPRecvFilter::SetPortNumber(int RecvPort)
{
    if(RecvPort == NULL) return E_UNEXPECTED;

    this->Stop();
    C*RTPOutputStream *lpOutStream = (C*RTPOutputStream*)m_paStreams[0];
    if(lpOutStream == NULL) {
        return E_UNEXPECTED;
    }
    lpOutStream->SetPortNumber(RecvPort);
    this->Run(NULL);

    return S_OK;
}
```

図 6.5: $C*RTPRecvFilter :: SetPortNumber$ 関数の実装

送信パラメータ設定

送信中にモジュールに対してパラメータ値の変更を行うため，各モジュールは必要なインタフェース関数を実装する．以下の例 ($CDVRTPSendFilter :: SetFrameRate$ 関数) は，メディアフォーマットに DV を用いる場合に，送出フレームレートを設定するために必要なインタフェース関数の定義を示す．

6.3.2 システム資源計測モジュール

本モジュールでは，計算機におけるプロセッサ・メモリ・ネットワークの各資源値を計測する．本モジュールは配信アプリケーション内部に実装されている．

```
STDMETHODIMP CDvRTPSendFilter::SetFrameRate(int frameRate)
{
    m_nFrameRate = frameRate;
    return NOERROR;
}
```

図 6.6: $C * RTPSendFilter :: SetFrameRate$ 関数の実装

パフォーマンスカウンタ

Microsoft Windows オペレーティングシステムでは，オペレーティングシステムが各種計算機資源に関するパフォーマンスデータを収集する手段として，パフォーマンスカウンタが実装されている．オペレーティングシステムには，アプリケーションで取得可能な定義済みカウンタセットが含まれており，カウンタセットは機能別に分類されている．カウンタからは生データないしは時間経過により変化する値が取得できる．

パフォーマンスカウンタは特定のパフォーマンスオブジェクトを指定することで取得する．主なパフォーマンスオブジェクトの例を表に示す．

表 6.3: パフォーマンスオブジェクト

カテゴリ	パフォーマンスオブジェクト	概要
プロセッサ	＼ Processor ＼% Processor Time	CPU の利用率
	＼ System ＼ Processor Queue Length	プロセッサ・キューのスレッド数
	＼ Processor ＼% Interrupts/sec	プロセッサが受け取って処理するハードウェア割込みの秒平均値
	＼ System ＼ Context switches/sec	任意のスレッドから他のスレッドに切り替える全てのプロセス率
メモリ	＼ Memory ＼ Available Bytes	実行中のプロセスに利用可能な物理メモリサイズ
	＼ Memory ＼ Cache Bytes	ファイル・システム・キャッシュが使用しているバイト数
	＼ Memory ＼ Page/sec	ハード・ページ・フォルト解決のためのディスク読み込み・書き込みページ数
	＼ Memory ＼ Page Faults/sec	ハードページ・フォルトの秒平均率
	＼ Memory ＼ Page Input/sec	ハード・ページ・フォルト解決のためにディスクから読み取られたページ数
	＼ Memory ＼ Page Reads/sec	ハード・ページ・フォルト解決のためにディスクが読み取られた回数
ディスク	＼ LogicalDisk ＼% Free Space	空き領域（%）
	＼ LogicalDisk ＼% Disk Time	アクセス時間（%）
	＼ PhysicalDisk ＼% Read/sec 1	1 秒にあたりの読み込み動作回数
	＼ PhysicalDisk ＼% Writes/sec 1	1 秒にあたりの書き込み動作回数
ネットワーク	＼ Network Segment ＼% Net Utilization	ネットワーク・ゼグメントの利用率
	＼ Network Interface ＼ Bytes Total/sec	1 秒にあたりの送受信バイト数
	＼ Network Interface ＼ Packets/sec	1 秒にあたりの送受信バケット数

パフォーマンスカウンタを用いたシステム資源計測

本モジュールではパフォーマンスカウンタを用いた計算機資源量の取得を行う．取得に必要なパフォーマンスオブジェクトは以下の通りである．

- プロセッサ：
  $\diagdown Processor \diagdown \%\ ProcessorTime$：プロセッサの使用率
- メモリ：
  $\diagdown Memory \diagdown CommittedBytes$：使用中のメモリの総バイト数
  $\diagdown Memory \diagdown CommitLimit$：利用可能なメモリの総バイト数
- ネットワーク：
  $\diagdown NetworkInterface \diagdown BytesTotal/sec$：使用中のネットワーク帯域
  $\diagdown NetworkInterface \diagdown CurrentBandwidth$：使用可能なネットワーク帯域

パフォーマンスカウンタを使用するためにはカウンタの初期化が必要となる．そこで，アプリケーションの初期化を行う $CAmssServerDlg :: OnInitDialog$ 関数においてパフォーマンスカウンタの初期化を行う．

```
/* 新規クエリーを作成 */
PdhOpenQuery(NULL, 0, &hQuery);
    :
/* インタフェース名の設定 */
CString target,qname1,qname2;
target.Format("%s",m_TargetIfRow.bDescr);
target.Replace("#","_");
target.Replace("/","_");
qname1.Format("\\Network Interface(%s)\\Bytes Total/sec", target);
qname2.Format("\\Network Interface(%s)\\Current Bandwidth", target);

/* クエリに追加 */
PdhAddCounter(hQuery,"\\Processor(_Total)\\%Processor Time",0,&hCounter);
PdhAddCounter(hQuery2, "\\Memory\\Committed Bytes", 0, &hCounter2);
PdhAddCounter(hQuery3, "\\Memory\\Commit Limit", 0, &hCounter3);
PdhAddCounter(hQuery4, qname1, 0, &hCounter4);
PdhAddCounter(hQuery5, qname2, 0, &hCounter5);

/* 初回計測 */
PdhGetRawCounterValue(hCounter, NULL, &RawValue0);
    :
```

図 6.7: $CAmssServerDlg :: OnInitDialog$ 関数における初期化処理

また，資源値の取得は $CAmssServerDlg : OnTimer$ 関数において行う．

6.3.3 ネットワーク資源計測モジュール

本モジュールでは，ネットワークの伝送特性に基づいたパラメータ変更処理を行うために用いるネットワークの伝送特性値の取得を行う．本モジュールは，RTP/IP パケットカプセル化・

```
if(!isFirst){
    CopyMemory(&RawValue0, &RawValue1, sizeof(RawValue1));
        :
} else {
    isFirst=0;
}

PdhCollectQueryData(hQuery);
PdhGetRawCounterValue(hCounter, NULL, &RawValue1);
    :

PdhCalculateCounterFromRawValue(
    hCounter, PDH_FMT_LONG, &RawValue1, &RawValue0, &FmtValue);
PdhCalculateCounterFromRawValue(
    hCounter2, PDH_FMT_DOUBLE|PDH_FMT_NOSCALE,
    &RawValue21, &RawValue20, &FmtValue2);
    :
```

図 6.8: $CAmssServerDlg : OnTimer$ 関数における資源値取得

脱カプセル化モジュールの一部として実装されており，アプリケーションの操作に必要なインタフェースのみを共通な呼び出し関数として定義し，抽象化を行っている．アプリケーションが利用可能なインタフェースは以下の通りである．

パケットロス値の取得

パケットロス値の計測は受信した RTP パケットに含まれる RTP シーケンス番号から求める．RTP ヘッダの取り出し処理後に，前回取得した RTP シーケンス番号との差分を求めることで未到達のパケット数を計測する．計測した値は，グローバル変数 (メンバ変数) である m_PacketLosts に代入され，インタフェース関数である $GetPacketLoss$ 関数を通じてアプリケーションに渡される．

伝搬遅延時間の取得

伝搬遅延時間の計測は，RTP パケットの受信処理後に時刻を計測し，前回時刻との差分を求めることで計測する．計測した値は，グローバル変数 (メンバ変数) である m_Jitter に代入され，インタフェース関数である $GetJitter$ 関数を通じてアプリケーションに渡される．

また，時刻の計測は，Windows オペレーティングシステムの持つ高精度パフォーマンスカウンタを用いて，マイクロ秒精度で行う．高精度パフォーマンスカウンタは CPU クロックを基にしたカウンタである．
$QueryPerformanceCounter$ 関数によりカウンタ値が，$QueryPerformanceFrequency$ 関数により動作周波数が取得できる．

```
HRESULT C*RTPOutputStream::FillBuffer(IMediaSample *pSamp)
{
:
    while (m_DoProcess) {
:
        m_PacketLosts += ntohs(rtphdr->seq) - m_PrevRTPSequence - 1;
        m_PacketCounts++;
        m_PrevRTPSequence = ntohs(rtphdr->seq);
:
    }
}

void C*RTPOutputStream::GetPacketLoss(int *pkts_lost, int *pkts)
{
    *pkts_lost = m_PacketLosts;
    *pkts = m_PacketCounts;
}
```

図 6.9: パケットロス値の取得処理

```
HRESULT C*RTPOutputStream::FillBuffer(IMediaSample *pSamp)
{
:
    while (m_DoProcess) {
:
        n = recvfrom(m_Socket, (char *)m_SocketBuffer, ...
:
        r = QueryPerformanceFrequency( (LARGE_INTEGER *)&freq );
        if( r == 0 ) {
            MessageBox(NULL, "error: High-res counter", NULL, MB_OK);
            return E_UNEXPECTED;
        }

        QueryPerformanceCounter( (LARGE_INTEGER *)&m_CurTime );
        m_Jitter = (double)(m_CurTime-m_PrevTime) / freq * 1000000.0;
        m_PrevTime = m_CurTime;
:
    }

}

void C*RTPOutputStream::GetJitter(double *jitter)
{
    *jitter = m_Jitter;
}
```

図 6.10: 伝搬遅延時間の取得処理

6.3.4 リクエスト・レスポンス制御モジュール

本モジュールでは，取得した環境資源値を基に実際に行う配信制御の判別，ならびにノード間での情報交換・共有を行う．本モジュールはアプリケーション内部に実装されている．

他ノードへのリクエスト発行

他ノードに対してリクエストを発行する際に用いる構造体を 3 つ定義した．これらはリクエスト時に利用される SDP 記述を構成するために利用される．図 6.11，図 6.12，図 6.13にそれぞれの構造体定義を示す．

AMSSResourceDesc 構造体は資源 SDP 記述に含まれる各資源値の情報が格納される．それぞれの変数に代入された値は，リクエスト発行時，SDP セッションレベル属性を記述する際に利用される．

```
typedef struct AMSSResourceDesc {
    int ProcessorResource;
    int MemoryResource;
    int NetworkResource;
} AMSSResourceDesc;
```

図 6.11: *AMSSResourceDesc* 構造体

AMSSNetCharDesc 構造体はリクエスト SDP 記述に含まれる各資源値の情報が格納される．それぞれの変数に代入された値は，リクエスト発行時，SDP メディアレベル属性を記述する際に利用される．

```
typedef struct AMSSNetCharDesc {
    int PacketLoss;
    double Jitter;
    char ParameterName[128];
    int Parameter;
} AMSSNetCharDesc;
```

図 6.12: *AMSSNetCharDesc* 構造体

AMSSRequestData 構造体はリクエスト発行を行う関数で必要な変数が格納される．リクエスト発行前に，本構造体を生成した上で，リクエスト発行関数に対してポインタとして渡す．

```
typedef struct AMSSRequestData {
    char szTargetHost[128];
    int nTargetPort;
    char szRequestMethod[16];
    char szUser[128];
    AMSSResourceDesc Resource;
    int nVideoFormat;
    int nVideoPort;
    int nAudioFormat;
    int nAudioPort;
    AMSSNetCharDesc Netchar;
} AMSSRequestData;
```

図 6.13: *AMSSRequestData* 構造体

相手ノードに対してのリクエスト発行は $CAmssServerDlg :: SubmitRequest$ 関数を用いて行う．図6.14にリクエスト発行処理を行う関数定義を示す．$CAmssServerDlg :: SubmitRequest$ 関数は引数にリクエストタイプならびに $AMSSRequestData$ 構造体へのポインタを要求する．リクエストタイプは表 6.4に示すいずれかの値を代入する．

```
HRESULT CAmssServerDlg::SubmitRequest
(int nReqType, struct AMSSRequestData* Req)
{
:
    /* リクエスト用 Socket の Open */
    Socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
:
    /* ターゲットホストの設定 */
    SOCKADDR_IN saServer;
    saServer.sin_port = htons(Req->nTargetPort);
    saServer.sin_family = AF_INET;
    saServer.sin_addr = *((LPIN_ADDR)*lpHostEntry->h_addr_list);
:
    /* ターゲットホストへの接続 */
    nRet = connect(Socket, (LPSOCKADDR)&saServer, sizeof(SOCKADDR_IN));
:
/* リクエスト発行処理（後述）*/
    closesocket(Socket);
    return S_OK;
}
```

図 6.14: $CAmssServerDlg :: SubmitRequest$ 関数におけるリクエスト発行処理

表 6.4: リクエストタイプ一覧

nReqType (ID)	種別
0	メッセージを含まないリクエスト
1	資源 SDP 記述を含むリクエスト
2	リクエスト SDP 記述を含むリクエスト

$CAmssServerDlg :: SubmitRequest$ 関数ではリクエストに必要なリクエストヘッダならびにメッセージボディを生成する．図 6.15に全リクエストに共通なリクエストヘッダ生成処理，図 6.16に資源 SDP 記述に必要なリクエストヘッダ生成処理，図 6.17にリクエスト SDP 記述に必要なリクエストヘッダ生成処理をぞれぞれ示す．

```
HRESULT CAmssServerDlg::SubmitRequest
(int nReqType, struct AMSSRequestData* Req)
{
:
    sprintf(szBuffer, "%s AMSS/1.0\r\n", Req->szRequestMethod);
    nRet = send(Socket, szBuffer, strlen(szBuffer), 0);
/* リクエストヘッダ生成処理（一部略）*/
    sprintf(szBuffer, "\r\n");
    nRet = send(Socket, szBuffer, strlen(szBuffer), 0);
    sprintf(szBuffer, "v=0\r\n");
    nRet = send(Socket, szBuffer, strlen(szBuffer), 0);
/* SDP セッションレベル属性生成処理（一部略）*/
/* 資源 SDP 記述処理（後述）*/
    sprintf(szBuffer, "m=video %d RTP/AVP %d\r\n", Req->nVideoPort, Req->nVideoFormat);
    nRet = send(Socket, szBuffer, strlen(szBuffer), 0);
    sprintf(szBuffer, "m=audio %d RTP/AVP %d\r\n", Req->nAudioPort, Req->nAudioFormat);
    nRet = send(Socket, szBuffer, strlen(szBuffer), 0);
/* リクエスト SDP 記述処理（後述）*/
    sprintf(szBuffer, "\r\n");
    nRet = send(Socket, szBuffer, strlen(szBuffer), 0);
:
    return S_OK;
}
```

図 6.15: $CAmssServerDlg :: SubmitRequest$ 関数におけるリクエストヘッダ生成

```
HRESULT CAmssServerDlg::SubmitRequest
(int nReqType, struct AMSSRequestData* Req)
{
:
    if(nReqType == 1) {
        if( !Req->Resource.ProcessorResource ||
            !Req->Resource.MemoryResource ||
            !Req->Resource.NetworkResource )
        {
            MessageBox("Resource Information not found.");
            return E_FAIL;
        }

        sprintf(szBuffer, "a=processor:%d\r\n", Req->Resource.ProcessorResource);
        nRet = send(Socket, szBuffer, strlen(szBuffer), 0);
        sprintf(szBuffer, "a=memory:%d\r\n", Req->Resource.MemoryResource);
        nRet = send(Socket, szBuffer, strlen(szBuffer), 0);
        sprintf(szBuffer, "a=network:%d\r\n", Req->Resource.NetworkResource);
        nRet = send(Socket, szBuffer, strlen(szBuffer), 0);
    }
:
    return S_OK;
}
```

図 6.16: $CAmssServerDlg :: SubmitRequest$ 関数におけるリクエストヘッダ生成

```
HRESULT CAmssServerDlg::SubmitRequest
(int nReqType, struct AMSSRequestData* Req)
{
:
    if(nReqType == 2) {
        if( !Req->Netchar.Jitter ||
            !Req->Netchar.PacketLoss ||
            !Req->Netchar.ParameterName ||
            !Req->Netchar.PacketLoss )
        {
            MessageBox("Network Characteristics Information not found.");
            return E_FAIL;
        }

        sprintf(szBuffer,"a=%s:%d\r\n",Req->Netchar.ParameterName,Req->Netchar.Parameter);
        nRet = send(Socket, szBuffer, strlen(szBuffer), 0);
        sprintf(szBuffer, "a=packetloss:%d\r\n", Req->Netchar.PacketLoss);
        nRet = send(Socket, szBuffer, strlen(szBuffer), 0);
        sprintf(szBuffer, "a=jitter:%f\r\n", Req->Netchar.Jitter);
        nRet = send(Socket, szBuffer, strlen(szBuffer), 0);
    }
:
    return S_OK;
}
```

図 6.17: $CAmssServerDlg :: SubmitRequest$ 関数におけるリクエストヘッダ生成

$CAmssServerDlg :: SubmitRequest$ 関数ではレスポンスに対応した処理も行う．図 6.18にレスポンスの受信とその処理を行う関数定義を示す．

```
HRESULT CAmssServerDlg::SubmitRequest
(int nReqType, struct AMSSRequestData* Req)
{
:
    while(1)
    {
        ZeroMemory(szBuffer, sizeof(szBuffer));
        /* レスポンスの受信 */
        nRet = recv(Socket, szBuffer, sizeof(szBuffer), 0);
        if (nRet == SOCKET_ERROR)
        {
            MessageBox("recv()");
            break;
        } else if (nRet == 0) {
            break;
        }
        /* レスポンスに対応した処理 */
    }
:
    return S_OK;
}
```

図 6.18: $CAmssServerDlg :: SubmitRequest$ 関数におけるレスポンス処理

他ノードへのレスポンス発行

リクエストを受信したノードでは相手ノードに対してレスポンスを必ず発行する必要がある．$CAmssRRServer :: ParseRequest$ 関数ではレスポンスの生成ならびに発行を行う．図 6.19にレスポンス発行処理を行う関数定義を示す．

```
BOOL CAmssRRServer::ParseRequest
(string szRequest, string &szResponse, BOOL &bKeepAlive)
{
:
/* リクエスト判別・メッセージ生成処理（後述）*/
:
    /* コネクションタイプ (Keep-Alive) 判別 */
    n = szRequest.find("\nConnection: Keep-Alive", 0);
    if(n != string::npos)
        bKeepAlive = TRUE;
:
/* レスポンスヘッダ生成処理（後述）*/
:
    /* レスポンスメッセージ生成処理 */
    szResponse = string(pResponseHeader);
    if(pBuf)
        szResponse += string(pBuf, strlen(pBuf));
    delete pBuf;
    pBuf = NULL;

    return TRUE;
}
```

図 6.19: $CAmssRRServer :: ParseRequest$ 関数におけるレスポンス作成処理

レスポンスを発行するためには受信したリクエストの種別を判別する必要がある．図 6.20にリクエストの判別処理を行う関数定義を示す．

```
BOOL CAmssRRServer::ParseRequest
(string szRequest, string &szResponse, BOOL &bKeepAlive)
{
:
    n = szRequest.find(" ", 0);
    if(n != string::npos)
    {
        szMethod = szRequest.substr(0, n);
        if(szMethod == "REGISTER") {
            /* 必要な処理 */
        } else if(szMethod == "NODELIST") {
            /* 必要な処理 */
            /* メッセージ生成処理 */
                :
        } else {
            szStatusCode = "501 Not Implemented";
            szFileName = ERROR501;
        }
    }
:
    return TRUE;
}
```

図 6.20: $CAmssRRServer :: ParseRequest$ 関数におけるリクエスト判別処理

CAmssRRServer :: *ParseRequest* 関数では，受信したリクエストからメッセージボディを読み込み，必要な処理を行う．図 6.21に SDP の読み込み・判別処理を行う関数定義を示す．ここでは CHGFMT リクエストに含まれるビデオフォーマットに関する情報の読み込みを示している．

```
BOOL CAmssRRServer::ParseRequest
(string szRequest, string &szResponse, BOOL &bKeepAlive)
{
:
    n = szRequest.find(" ", 0);
    if(n != string::npos)
    {
        szMethod = szRequest.substr(0, n);
        if(szMethod == "REGISTER") {
            /* 処理 */
        } else if(szMethod == "CHGFMT") {
:
            /* ビデオフォーマットラインの読み込み */
            n = szRequest.find("\nm=video", 0);
            if(n != string::npos) {
                szVFormat = szRequest.substr(n+1, szRequest.length()-n-1);
                /* メッセージ読み込み */
                n1 = szVFormat.find(" ", 0);
                n2 = szVFormat.find(" ", n1 + 1);
                n3 = szVFormat.find(" ", n2 + 1);
                n4 = szVFormat.find("\r\n", n3 + 1);
                if( n1 != string::npos && n2 != string::npos
                     && n3 != string::npos && n4 != string::npos ) {
                    szVPort = szVFormat.substr(n1+1,n2-n1-1);
                    szVID = szVFormat.substr(n3+1, n4-n3-1);
                }
            }
:
    return TRUE;
}
```

図 6.21: *CAmssRRServer* :: *ParseRequest* 関数における SDP 読み込み処理

すべての処理を終了した後，その結果について相手ノードに対してレスポンスとしてメッセージを送信する必要がある．図 6.22にレスポンスの発行を行う関数定義を示す．

```
BOOL CAmssRRServer::ParseRequest
(string szRequest, string &szResponse, BOOL &bKeepAlive)
{
:
    /* レスポンス時刻取得 */
    char szDT[128];
    struct tm *newtime;
    long ltime;
    time(&ltime);
    newtime = gmtime(&ltime);
    strftime(szDT, 128, "%a, %d %b %Y %H:%M:%S GMT", newtime);
    /* レスポンス作成 */
    sprintf(pResponseHeader,
      "AMSS/1.0 %s\r\n
       Date: %s\r\n
       Server: %s\r\n
/* レスポンスヘッダ生成処理（一部略）*/
       Connection: %s\r\n
       Content-Type: %s\r\n
       \r\n",
      szStatusCode.c_str(), szDT, SERVERNAME,
      bKeepAlive ? "Keep-Alive" : "close", szContentType.c_str());
:
    return TRUE;
}
```

図 6.22: *CAmssRRServer* :: *ParseRequest* 関数におけるレスポンス作成処理

ユーザへのレポート

各資源値が許容値を超えた場合やユーザへのレポートが必要になった場合，図 6.23に示すウィンドウを通じてユーザにレポートする．

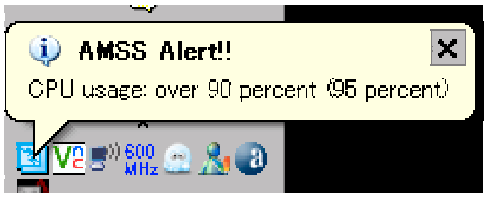

図 6.23: AMSS スクリーンショット - レポート

本機能は Windows の持つトレイアイコン機能とバルーンチップ機能を利用し実装されている．通常のメッセージボックスでは，メッセージ表示中は同一関数内の処理を並行して実行できない．バルーンチップを用いることで，複数の並行処理を行いながらユーザに対してレポートすることが可能である．図 6.24にレポートの発行を行う定義を示す．

```
if(CpuPercent>m_nCpuTH){
    szBaloonTip.Format(
        "CPU usage: over %d percent (%d percent)",
        m_nCpuTH, CpuPercent
    );
    m_TrayIcon.PopupBalloon(
        szBaloonTip,
        _T("AMSS Alert!!"),
        NIIF_INFO, 10000
    );
}
```

図 6.24: $CAmssServerDlg : OnTimer$ 関数におけるレポート処理

6.4 まとめ

本章では設計に基づき実装を行った自律協調型メディア配信システム AMSS の概要ならびに以下に示す各部の実装の詳細について述べた.

- RTP/IP パケットカプセル化・脱カプセル化モジュール
- システム資源計測モジュール
- ネットワーク資源計測モジュール
- リクエスト・レスポンス制御モジュール

次章では，本機構の動作について検証し，本論文にて提案した手法の有効性について評価する.

第7章　評価

本章では設計を基に実装を行った AMSS の評価について述べる．7.2節で本機構が実現した機能，7.3節以降で本機構の動作に関する評価について述べる．また，7.6節では本評価のまとめを述べる．

7.1　評価概要

本評価は本研究が提案する自律協調型メディア配信手法の有効性の検証を目的とする．定性評価ならびに定量評価は以下の項目について行う．

- **定性評価**：設計に基づき実装した AMSS の実現した機能
- **定量評価**：AMSS による配信制御の実現

7.2　本機構の実現した機能

本研究では計算機資源やネットワーク資源の変動に基づいた配信制御を行うメディア配信システムの構築を行った．本機構では以下の機能を実現した．

- 複数のメディアフォーマットの送受信
- 必要に応じたメディアフォーマットの動的に切り替え
- 1 対 1 の通信ノード間での自律協調的な環境資源の計測
- 1 対 1 の通信ノード間での情報交換 (ネゴシエーション)
- 環境資源の統一的な指標化・ノード環境に依存しない情報定義
- ノード間で協調した配信制御

7.3 実験 1：状況に適応したフォーマットの選択

本実験では送信開始前のネゴシエーションの動作について検証する．本実験では AMSS の定性評価を行う．

7.3.1 実験手順と仮説

実験 1 は図 7.1に示した送信ノードならびに受信ノードの 2 台の PC をネットワークスイッチを介して接続し行った．実験には表 7.1に示す計算機を用いる．

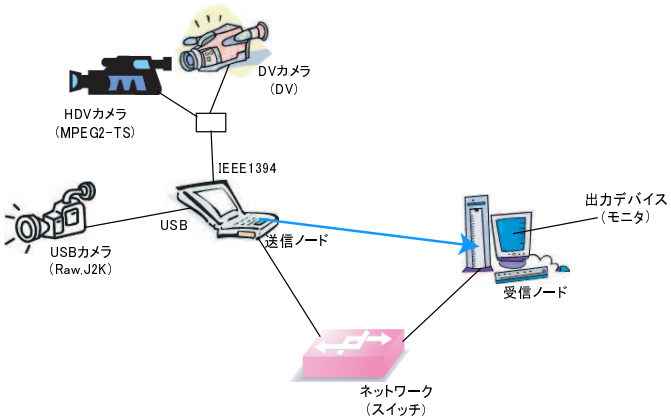

図 7.1: 実験 1：実験環境

表 7.1: 実験 1 に用いた計算機環境

CPU(受信)	Intel Pentium M 1.1GHz
メモリ (受信)	1024MB
ハードディスク (受信)	80GB
NIC(受信)	100Base-TX
CPU(送信)	Intel Xeon Processor 3.2GHz
メモリ (送信)	2048MB
ハードディスク (送信)	73GB
NIC(送信)	100Base-TX, 1000Base-T
映像・音声機器 (送信 1)	Victor HDV カメラ GR-HD1 (HDV モード)
映像・音声機器 (送信 2)	Victor HDV カメラ GR-HD1 (DV モード)

本実験では，受信ノードに優先度の高い MPEG2-TS フォーマットの受信に対して十分な資源を確保できない計算機を用いるため，実験では送信開始前の CAPEX リクエスト・レスポンスにより，優先度 2 の DV フォーマットが選択される．

7.3.2 実験結果

図 7.2に受信ノードでの AMSS の動作ログ，図 7.3に送信ノードでの AMSS の動作ログを示す．

```
1136636062 : Go Experiment Scenario #1. : OnBnClickedAmssstart()
1136636062 : REGISTER submitted. : OnBnClickedAmssstart()
1136636062 : NODELIST submitted. : OnBnClickedAmssstart()
1136636063 : START submitted. : OnBnClickedAmssstart()
1136636063 : FormatID is DV. : OnBnClickedAmssstart()
1136636065 : CAPEX submitted. : OnBnClickedAmssstart()
1136636065 : RecvGraphDestructed. : FreeRecvGraphBuilder()
1136636065 : RecvGraphInitialized. : InitRecvGraphBuilder()
1136636065 : DVRecvFilterInitialized. : InitDVRecvFilter()
1136636065 : RecvStart. : OnBnClickedRecv()
※ ACK に関するログは省略
```

図 7.2: 受信ノードでの動作ログ

```
1136636067 : REGISTER req received. : OnCommand()
1136636067 : NODELIST req received. : OnCommand()
1136636067 : START req received. : OnCommand()
1136636067 : CAPEX req received. : OnCommand()
1136636070 : SendGraphDestructed. : FreeSendGraphBuilder()
1136636070 : DVSendFilterInitialized. : InitDVSendFilter()
1136636070 : SendGraphInitialized. : InitSendGraphBuilder()
1136636070 : SendStart. : OnBnClickedSend()
```

図 7.3: 送信ノードでの動作ログ

7.3.3 考察

動作ログより，受信ノードにおける環境資源値の取得ならびに値による状況評価の結果，DV フォーマットが選択 (時刻:1136636063) され，送信ノードに対して CAPEX リクエストを発行 (時刻:1136636065) している．受信ノードでは，CAPEX リクエスト発行後，DV フォーマットでの受信を行う設定が行われた．送信ノードでは，リクエストを受け，DV フォーマットでの送信が開始 (時刻:1136636070) されている．

7.4 実験 2：CHGFMT リクエストの動作

本実験では資源状態に基づくフォーマット変更要求である CHGFMT リクエストの動作について検証する．本実験では AMSS の定性評価ならびに定量評価を行う．

7.4.1 実験手順と仮説

実験 2 は図 7.4に示した送信ノードならびに受信ノードの 3 台の PC をネットワークスイッチを介して接続し行った．実験には表 7.2に示す計算機を用いる．

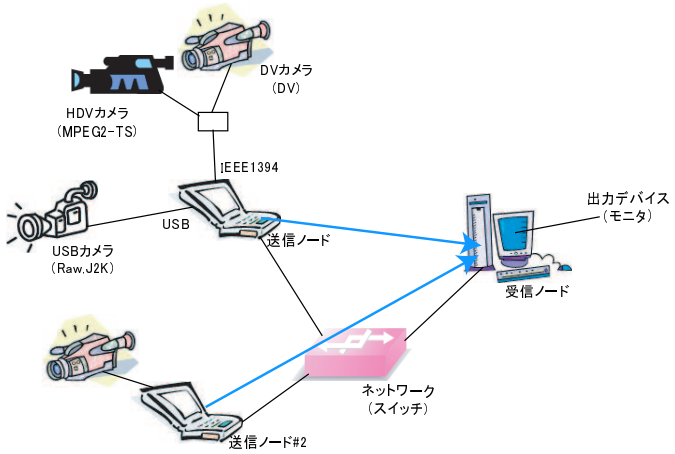

図 7.4: 実験 2：実験環境

表 7.2: 実験 2 に用いた計算機環境

CPU(受信)	Intel Pentium M 1.1GHz
メモリ (受信)	1024MB
ハードディスク (受信)	80GB
NIC(受信)	100Base-TX
CPU(送信)	Intel Xeon Processor 3.2GHz
メモリ (送信)	2048MB
ハードディスク (送信)	73GB
NIC(送信)	100Base-TX, 1000Base-T
映像・音声機器 (送信 1)	Victor HDV カメラ GR-HD1 (HDV モード)
映像・音声機器 (送信 2)	Victor HDV カメラ GR-HD1 (DV モード)
映像・音声機器 (送信 3)	Logicool USB カメラ QuickCam Pro for Notebook

図 7.5に実験のタイミングを示す.

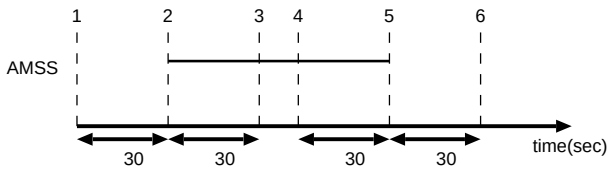

図 7.5: 実験 2 のタイミング

1. パフォーマンスログ記録を開始
2. AMSS を起動
3. シナリオ 2 を実行
4. CAPEX リクエストの発行
5. AMSS を終了
6. パフォーマンスログ記録を終了

本実験では，すでに別プロセスとして DV フォーマットの受信・再生を行っている環境にて，AMSS を実装したアプリケーションを動作させることで行う．別プロセスとして動作させるアプリケーションは，AMSS 非実装であり資源状態に対して考慮しないため，計算機では新たなメディア配信において MPEG2-TS フォーマットならびに DV フォーマットでの受信に対して十分な資源が確保できない．

本来であれば，送信開始前の CAPEX リクエスト・レスポンスで優先度 3 の非圧縮ビットマップフォーマットが選択されることが正しい動作であるが，これを無視し優先度 1 の MPEG2-TS フォーマットでの送受信を開始するようシステムの設定を行うため，実験では CHGFMT リクエスト・レスポンスが発行され，本来選択される優先度 3 の非圧縮ビットマップフォーマットにフォーマットが変更される．

7.4.2 実験結果と考察

図7.6に受信ノードでの AMSS の動作ログ，図7.7に送信ノードでの AMSS の動作ログを示す．

```
1136708149 : Go Experiment Scenario #2. : OnBnClickedAmssstart()
1136708149 : Register method loaded. : OnBnClickedAmssregist()
1136708149 : REGISTER submitted. : OnBnClickedAmssregist()
1136708150 : NODELIST submitted. : OnBnClickedAmssregist()
1136708150 : START submitted. : OnBnClickedAmssstart()
1136708150 : FormatID is MPEG2-TS. : OnBnClickedAmssstart()
1136708150 : CAPEX submitted. : OnBnClickedAmssstart()
1136708150 : RecvGraphDestructed. : FreeRecvGraphBuilder()
1136708150 : HDVRecvFilterInitialized. : InitHDVRecvFilter()
1136708150 : RecvGraphInitialized. : InitRecvGraphBuilder()
1136708150 : RecvStart. : OnBnClickedRecv()
1136708154 : Over-threshold-count. (21187836/diff: 21187836) : OnTimer()
1136708154 : RecvStop. : OnBnClickedRecv()
1136708154 : RecvGraphDestructed. : FreeRecvGraphBuilder()
1136708154 : FormatID is RAW. : OnTimer()
1136708154 : CHGFMT submitted. : OnTimer()
1136708154 : RecvGraphInitialized. : InitRecvGraphBuilder()
1136708154 : RAWRecvFilterInitialized. : InitRAWRecvFilter()
1136708154 : RecvStart. (21188101/diff: 265) : OnBnClickedRecv()
※ ACK に関するログは省略
```

図 7.6: 受信ノードでの動作ログ

```
1136708155 : REGISTER req received. : OnCommand()
1136708155 : NODELIST req received. : OnCommand()
1136708155 : START req received. : OnCommand()
1136708155 : CAPEX req received. : OnCommand()
1136708155 : SendGraphDestructed. : FreeSendGraphBuilder()
1136708156 : HDVSendFilterInitialized. : InitHDVSendFilter()
1136708156 : SendGraphInitialized. : InitSendGraphBuilder()
1136708156 : SendStart. : OnBnClickedSend()
1136708159 : CHGFMT req received. : OnCommand()
1136708159 : SendGraphDestructed. : FreeSendGraphBuilder()
1136708160 : RAWSendFilterInitialized. : InitRAWSendFilter()
1136708160 : SendGraphInitialized. : InitSendGraphBuilder()
1136708160 : SendStart. : OnBnClickedSend()
```

図 7.7: 送信ノードでの動作ログ

動作ログより，強制的に MPEG2-TS フォーマットを選択し，送信ノードに対して CAPEX リクエストを発行 (時刻:1136708150) している．受信ノードにおいて環境資源値の取得ならびに値による状況評価の結果，許容値の超過が認識 (時刻:1136708154) され，再度フォーマットの選択を行った結果，非圧縮ビットマップフォーマットを選択し，送信ノードに対して CHGFMT リクエストを発行 (時刻:1136708154) している．受信ノードでは，CAPEX リクエストならびに CHGFMT リクエスト発行後，各フォーマットでの受信を行う設定が行われた．送信ノードでは，CAPEX リクエストを受け MPEG2-TS フォーマットでの送信 (時刻:1136708156) が，CHGFMT リクエストを受け非圧縮ビットマップフォーマットでの送信 (時刻:1136708160) が

それぞれ行われている．また，動作ログから，資源許容量の超過を認識してから新たなフォーマットへの切り替え・再生までに要した時間は，265 ミリ秒である．

受信ノードにおけるプロセッサ，メモリ，ネットワークの各資源使用率の推移を図 7.8に示す．

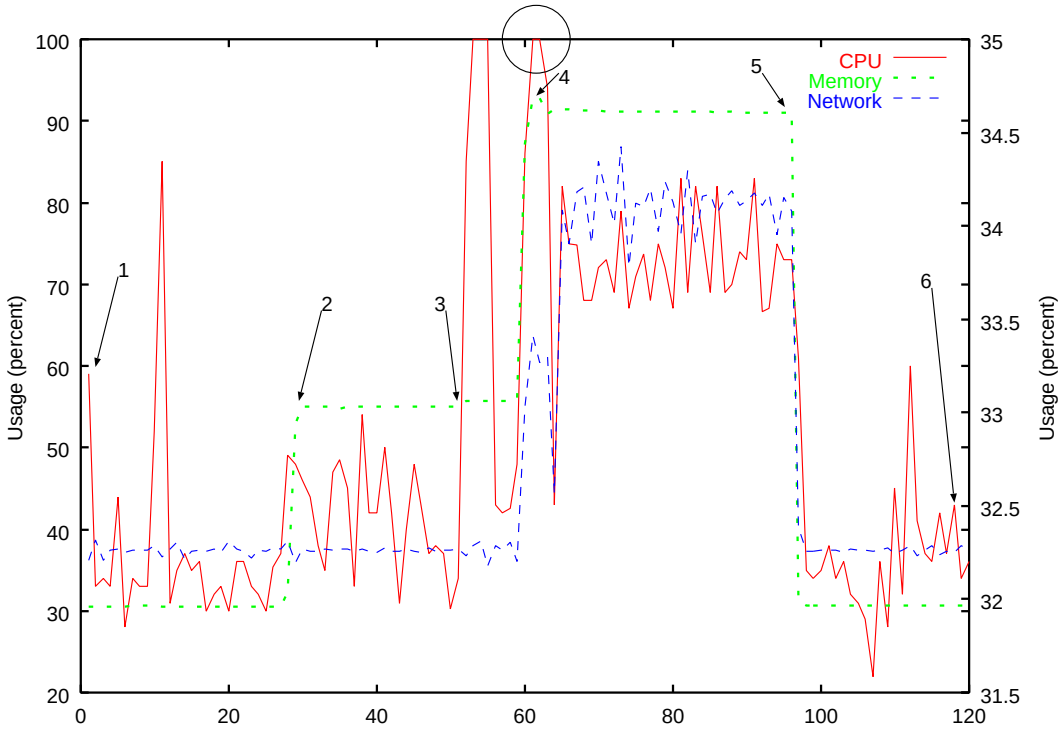

図 7.8: 実験 2 の結果

各資源量の推移のグラフより，AMSS の起動と同時にアプリケーションによるメモリ確保が，メディア配信の開始と同時にメディア配信のためのメモリ確保が行われている．また，MPEG2-TS フォーマットによる受信を開始した段階で，プロセッサ使用率が 100%に達しており，この時点で許容値を超えたためフォーマット変更処理が行われた．

7.5 実験 3：CHGFMT リクエスト・CHGCFG リクエストの動作

本実験では，資源状態に基づくフォーマット変更要求である CHGFMT リクエスト，ネットワーク伝送特性に基づくパラメータ変更要求である CHGCFG リクエストの動作について検証する．本実験では AMSS の定性評価ならびに定量評価を行う．

7.5.1 実験手順と仮説

実験 3 は図 7.9に示した送信ノードならびに受信ノードの 4 台の PC を用いて行った．実験には表 7.3に示す計算機を用いる．各 PC 間には，PC で作成したルータを設置し，片一方のインタフェースに dummynet を設定し，擬似的に 50Mbps に帯域幅を制限した．

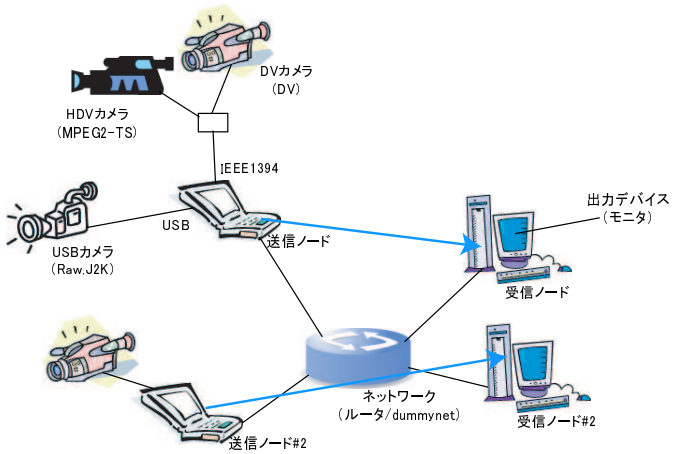

図 7.9: 実験 3：実験環境

表 7.3: 実験 3 に用いた計算機環境

CPU(受信)	Intel Pentium M 1.1GHz
メモリ (受信)	1024MB
ハードディスク (受信)	80GB
NIC(受信)	100Base-TX
CPU(送信)	Intel Pentium M 1.1GHz
メモリ (送信)	512MB
ハードディスク (送信)	80GB
NIC(送信)	100Base-TX
映像・音声機器 (送信 1)	Victor HDV カメラ GR-HD1 (HDV モード)
映像・音声機器 (送信 2)	Victor HDV カメラ GR-HD1 (DV モード)

実験 3 は 2 つの段階に分けて検証を行う．図 7.10にそのタイミングを示す．

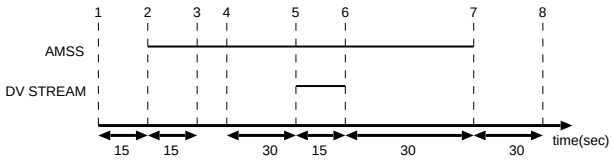

図 7.10: 実験 3 のタイミング

1. パフォーマンスログ記録を開始
2. AMSS を起動
3. シナリオ 3 を実行
4. CAPEX リクエストの発行
5. DV ストリームの開始
6. DV ストリームの終了
7. AMSS を終了
8. パフォーマンスログ記録を終了

第 1 段階では，実験 2 同様，CHGFMT リクエストの動作について検証する．受信ノードでは，優先度の高い MPEG2-TS フォーマットの受信に対して十分な資源を確保できない計算機を用いる．本来であれば，送信開始前の CAPEX リクエスト・レスポンスで優先度 2 の DV フォーマットが選択されることが正しい動作であるが，これを無視し優先度 1 の MPEG2-TS フォーマットでの送受信を開始するようシステムの設定を行うため，実験では CHGFMT リクエスト・レスポンスが発行され，本来選択される優先度 2 の DV フォーマットにフォーマットが変更される．

第 2 段階では，帯域の制限を行ったネットワークに対して別の DV ストリームの伝送を行うことで帯域不足が発生し，配信を行う DV フォーマットのパラメータ変更が必要となるため，実験では CHGCFG リクエスト・レスポンスが発行され，必要なパラメータ値が動的に変更される．また，別の DV ストリームの伝送停止後は，元のパラメータまで復帰する．

7.5.2 実験結果と考察

図 7.11に受信ノードでの AMSS の動作ログ，図 7.12に送信ノードでの AMSS の動作ログを示す.

```
1136746607 : Go Experiment Scenario #3. : OnBnClickedAmssstart()
1136746607 : Register method loaded. : OnBnClickedAmssregist()
1136746607 : REGISTER submitted. : OnBnClickedAmssregist()
1136746607 : NODELIST submitted. : OnBnClickedAmssregist()
1136746608 : START submitted. : OnBnClickedAmssstart()
1136746608 : FormatID is MPEG2-TS. : OnBnClickedAmssstart()
1136746608 : CAPEX submitted. : OnBnClickedAmssstart()
1136746608 : RecvGraphDestructed. : FreeRecvGraphBuilder()
1136746608 : HDVRecvFilterInitialized. : InitHDVRecvFilter()
1136746608 : RecvGraphInitialized. : InitRecvGraphBuilder()
1136746608 : RecvStart. : OnBnClickedRecv()
1136746612 : Over-threshold-count. (59645844/diff: 59645844) : OnTimer()
1136746612 : RecvGraphDestructed. : FreeRecvGraphBuilder()
1136746612 : FormatID is DV. : OnTimer()
1136746612 : CHGFMT submitted. : OnTimer()
1136746612 : DVRecvFilterInitialized. : InitDVRecvFilter()
1136746612 : RecvGraphInitialized. : InitRecvGraphBuilder()
1136746612 : RecvStart. (59646093/diff: 249) : OnBnClickedRecv()
1136746637 : CHGCFG/DV-ratedown submitted. : OnTimer()
1136746638 : CHGCFG/DV-ratedown submitted. : OnTimer()
1136746638 : CHGCFG/DV-ratedown submitted. : OnTimer()
1136746639 : CHGCFG/DV-ratedown submitted. : OnTimer()
1136746639 : CHGCFG/DV-ratedown submitted. : OnTimer()
1136746650 : CHGCFG/DV-ratedown submitted. : OnTimer()
1136746653 : CHGCFG/DV-ratedown submitted. : OnTimer()
1136746653 : CHGCFG/DV-ratedown submitted. : OnTimer()
1136746657 : CHGCFG/DV-rateup submitted. : OnTimer()
1136746659 : CHGCFG/DV-rateup submitted. : OnTimer()
1136746662 : CHGCFG/DV-rateup submitted. : OnTimer()
1136746665 : CHGCFG/DV-rateup submitted. : OnTimer()
1136746668 : CHGCFG/DV-rateup submitted. : OnTimer()
1136746675 : CHGCFG/DV-rateup submitted. : OnTimer()
1136746678 : CHGCFG/DV-rateup submitted. : OnTimer()
1136746681 : CHGCFG/DV-rateup submitted. : OnTimer()
※ ACK に関するログは省略
```

図 7.11: 受信ノードでの動作ログ

```
1136746786 : REGISTER req received. : OnCommand()
1136746786 : NODELIST req received. : OnCommand()
1136746786 : START req received. : OnCommand()
1136746786 : CAPEX req received. : OnCommand()
1136746786 : SendGraphDestructed. : FreeSendGraphBuilder()
1136746786 : HDVSendFilterInitialized. : InitHDVSendFilter()
1136746786 : SendGraphInitialized. : InitSendGraphBuilder()
1136746786 : SendStart. : OnBnClickedSend()
1136746791 : CHGFMT req received. : OnCommand()
1136746791 : SendGraphDestructed. : FreeSendGraphBuilder()
1136746791 : SendGraphInitialized. : InitSendGraphBuilder()
1136746791 : DVSendFilterInitialized. : InitDVSendFilter()
1136746791 : SendStart. : OnBnClickedSend()
1136746816 : CHGCFG/DV-ratedown req received. (2) : OnCommand()
1136746816 : CHGCFG/DV-ratedown req received. (3) : OnCommand()
1136746817 : CHGCFG/DV-ratedown req received. (4) : OnCommand()
1136746817 : CHGCFG/DV-ratedown req received. (5) : OnCommand()
1136746818 : CHGCFG/DV-ratedown req received. (6) : OnCommand()
1136746828 : CHGCFG/DV-ratedown req received. (7) : OnCommand()
1136746831 : CHGCFG/DV-ratedown req received. (8) : OnCommand()
1136746832 : CHGCFG/DV-ratedown req received. (9) : OnCommand()
1136746838 : CHGCFG/DV-rateup req received. (8) : OnCommand()
1136746841 : CHGCFG/DV-rateup req received. (7) : OnCommand()
1136746844 : CHGCFG/DV-rateup req received. (6) : OnCommand()
1136746847 : CHGCFG/DV-rateup req received. (5) : OnCommand()
1136746854 : CHGCFG/DV-rateup req received. (4) : OnCommand()
1136746857 : CHGCFG/DV-rateup req received. (3) : OnCommand()
1136746860 : CHGCFG/DV-rateup req received. (2) : OnCommand()
1136746863 : CHGCFG/DV-rateup req received. (1) : OnCommand()
```

図 7.12: 送信ノードでの動作ログ

動作ログにより，強制的に MPEG2-TS フォーマットを選択し，送信ノードに対して CAPEX リクエストを発行 (時刻:1136746608) している．受信ノードにおいて環境資源値の取得ならびに値による状況評価の結果，許容値の超過が認識 (時刻:1136746612) され，再度フォーマットの選択を行った結果，DV フォーマットを選択し，送信ノードに対して CHGFMT リクエストを発行 (時刻:1136746612) している．受信ノードでは，CAPEX リクエストならびに CHGFMT リクエスト発行後，各フォーマットでの受信を行う設定が行われた．送信ノードでは，CAPEX リクエストを受け MPEG2-TS フォーマットでの送信 (時刻:1136746786) が，CHGFMT リクエストを受け DV フォーマットでの送信 (時刻:1136746791) がそれぞれ行われている．

2 本目の DV ストリームの送信と同時に，受信ノードでパケットの伝搬遅延時間の揺らぎならびにパケットロスの増加を検知し，フレームレート変更のリクエストを CHGCFG リクエストにより発行している．(時刻：1136746637 から 1136746681) さらに，送信ノードでもリクエストに応じて，フレームレートの変更処理が行われている．(時刻:1136746816 から 1136746863)

また，動作ログから，資源許容量超過認識から新たなフォーマットへの切り替え・再生までに要した時間は，249 ミリ秒である．

受信ノードにおけるプロセッサ，メモリ，ネットワークの各資源使用率の推移を図 7.13に示す．

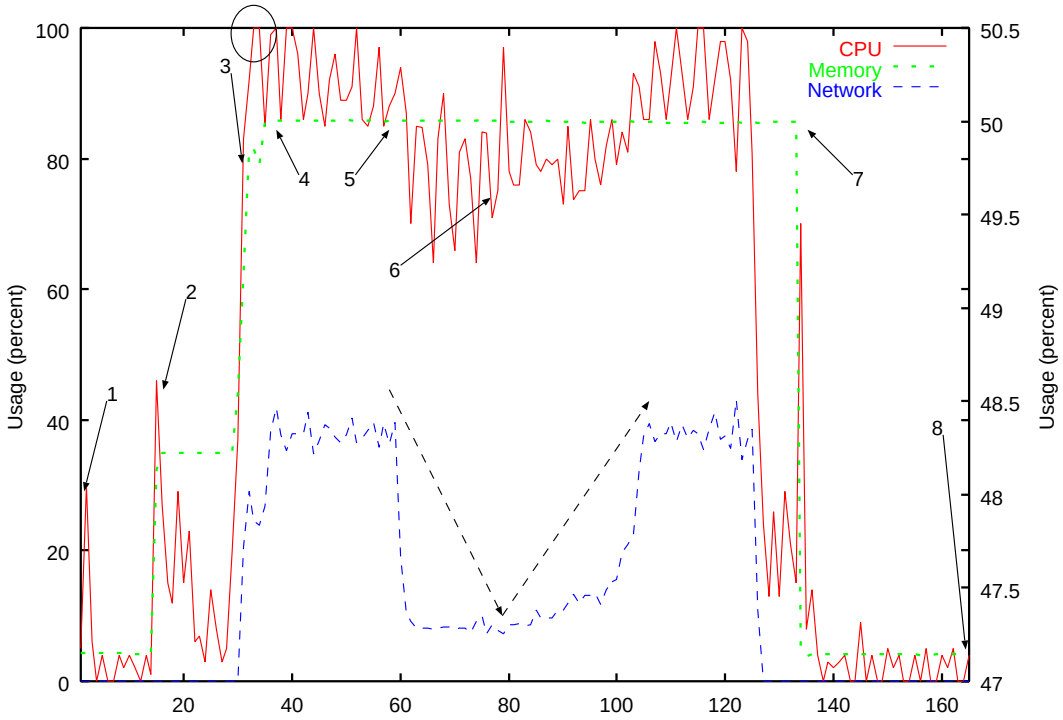

図 7.13: 実験 3 の結果

各資源量の推移のグラフでは，実験 2 同様に AMSS の起動と同時にアプリケーションによるメモリ確保が，メディア配信の開始と同時にメディア配信のためのメモリ確保が行われている．また，MPEG2-TS フォーマットによる受信を開始した段階で，プロセッサ使用率が 100%に達しており，この時点で許容値を超えたためフォーマット変更処理が行われた．

また，2 本目の DV ストリームが送信されると同時にネットワーク状態の変化を検知し，DV のフレームレートを下げる処理を行ったため，ネットワーク使用率が減少している．2 本目の DV ストリームの停止後は，レート増加処理に基づき，徐々にネットワーク使用率が増加している．

7.6 まとめ

以上の評価結果より，本機構が提供する環境資源に適応する自律協調型メディア配信手法は，以下に挙げる資源変化の検知と状態に適応したフォーマット変更処理やパラメータ変更処理を行いながらメディア配信が可能である．

- 計算機資源の増減によるメディアフォーマット変更
- 計算機資源量によるメディアフォーマット決定
- ネットワーク状態によるパラメータ変更

本機構を利用することにより，計算機資源ならびにネットワーク資源に適応し，必要に応じたフォーマット決定・変更，パラメータ変更を動的に行うことで自律協調的なメディア配信が実現された．

第8章　結論

本章では，本論文の結論として，本研究での成果を述べ，8.1節および 8.2節にて今後の課題と展望について述べる．

本研究では，計算機環境の持つプロセッサ，メモリなどの計算機資源，ならびにネットワーク資源と言ったメディア配信を取り巻く環境資源に適応する自律協調型メディア配信システムを実現した．

本研究では，インターネットを介したメディア配信の実現に必要な要件を整理し，以下に挙げる 3 点が配信に対して適した状態であることが必要であると述べた．

- インフラストラクチャ (ネットワーク)
- メディア (メディアフォーマット)
- システム (計算機環境)

メディア配信を取り巻く環境の変化に伴い，ユーザ間で得られるネットワーク環境や計算機環境に格差が生じるようになった．本研究では，これらの変化に対応するため計算機資源やネットワーク資源の指標化，ノード間での交換・共有，状況に応じたメディア配信制御を行う機構を開発した．また，ノード間で行う配信制御について，本研究では以下の 2 つの手法を提案した．

- ネットワーク伝送特性に基づく動的なレート制御手法
- 計算機資源情報に基づくフォーマット切り替え手法

本研究では，これら 2 つの配信制御手法を用いた自律協調型メディア配信手法の有効性を実証するために，既存のメディア配信システムである DVTS を基に，自律協調型メディア配信システム AMSS の設計をし，以下に挙げるモジュールの実装を行った．

- RTP/IP パケットカプセル化・脱カプセル化モジュール
- システム資源計測モジュール
- ネットワーク資源計測モジュール
- リクエスト・レスポンス制御モジュール

AMSS では，提案した資源指標化手法ならびにリクエスト・レスポンス型のメッセージング手法を用いて，ノード間で自律協調的な制御を行う．

AMSS を実装した DVTS の実際の挙動を確認し，評価を行った．評価の結果，環境資源の変動の検知ならびに状況に応じた配信制御が実現できた．

本研究により，インターネットを介したメディア配信において求められる 3 つの要件に対する自律的な判断と状況に応じたノード間の協調動作により，ユーザが資源状態を意識することなく，ネットワークならびに計算機状態に応じたメディア配信環境が実現できた．

8.1 今後の課題

8.1.1 さらなる多フォーマット化への課題

本研究では，対象とするメディアフォーマットを特定する形で設計・実装を行った．本機構は，RTP を用いて配信を行うメディアフォーマットについては設計を変更することなく適用可能である．しかし，制御の指標となる環境資源情報について，今回の設計では，制御に必要なものについて利用する．さらなる多フォーマット化を進めるためには，他の指標についても検討し，指標化を可能とする必要がある．また，各フォーマットに依存した制御についても，フォーマットごとではなく，ある程度決められた制御グループを定義し，特徴の似たメディアフォーマットについては抽象的に制御を表現できる必要がある．

8.1.2 他オペレーティングシステムへの考慮

本機構は，Windows オペレーティングシステムを対象とした設計・実装が行われている．しかし，本研究で指標化した資源値は，いずれも他オペレーティングシステムにおいても取得可能なものである．例えば，UNIX オペレーティングシステムでは，procfs などを用いることで取得できる．そこで，他オペレーティングシステムにおいて資源情報を取得するための本機構に対するライブラリを構成することで，本機構の他オペレーティングシステム環境に対する考慮が可能となる．前述した他の指標の検討とともに，オペレーティングシステムに応じた資源情報の取得ならびにオペレーティングシステムに依存しない AMSS への抽象化されたインタフェースの確保が必要である．

8.2 将来的展望：インターネットを介した統合的なメディア配信環境を目指して

本機構は個別のメディアセッションの制御を行う．今後は本論文で扱ったセッション制御手法の他，グローバルネットワークにおけるセッション情報の流通・管理についても考慮に入れる．図 8.1に AMSS ならびにセッション情報流通機構 (CNS:Contents Naming System) を用いたインターネットを介した統合的なメディア配信環境を示す．この例では，CNS を通じた一連のコンテンツ探索動作を実行した後，AMSS を用いたコンテンツサーバに対しての接続要求と接続環境のネゴシエーションを実行し，相互の設定を同期させた上でストリームデータの受信を行っている．

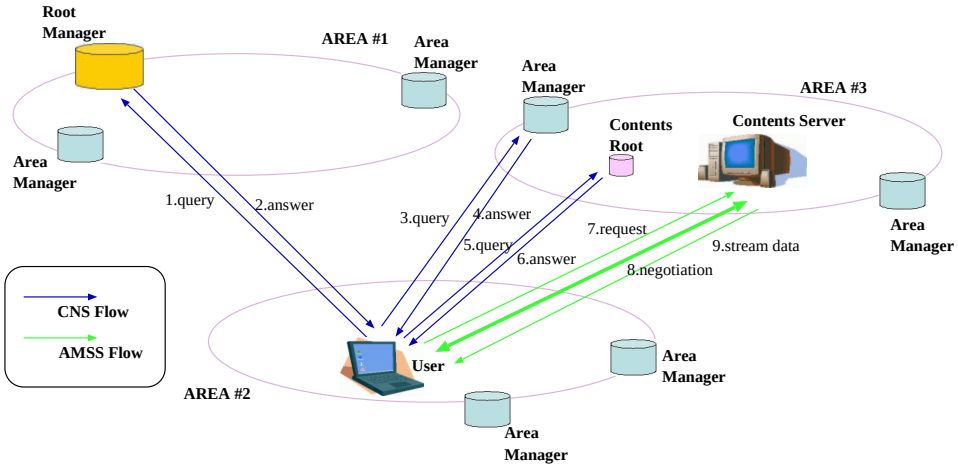

図 8.1: 統合的なメディア配信環境

メディア配信では，以下に挙げる要件が求められる．

- 送信ノード・受信ノード双方の存在するネットワーク的な位置が特定されていること (双方アドレスの把握)
- 送信ノード・受信ノードで利用するメディアフォーマットが同一であること (フォーマットの同一性)
- 送受信ノード間で利用可能なネットワーク環境が利用するフォーマットの要求を満たすこと (ネットワーク資源の確保)
- 送受信ノード双方で利用可能な計算機環境が利用するフォーマットの要求を満たすこと (計算機資源の確保)

AMSS はここに挙げる 4 つすべてを満たすが，前者 2 項目については考慮されていない部分も存在する．CNS ではセッション情報を配信エリアごとに分散配置し，情報をエリアごとの管理サーバが協調して保持，クライアントがエリア管理サーバを検索することで情報の配信を実現する．これらの実現により，必要とする資源情報とそれに適した配信，セッションの検索・管理が世界規模で実現可能なシステムの構築が可能となる．

謝辞

本研究を進めるにあたり，ご指導いただきました，慶應義塾常任理事 村井純博士，同大学環境情報学部教授 徳田秀幸博士，同学部助教授 中村修博士，楠本博之博士に感謝します．

また，絶えずご指導とご助言をいただきました，慶應義塾大学環境情報学部専任講師 重近範行博士，同大学院政策・メディア研究科助手 朝枝仁氏，同研究科博士課程 海崎良氏，SFC 研究所上席研究員 小川浩司氏に感謝します．

本研究を進めていく上でさまざまな励ましと助言，お手伝いをいただきました，松園和久氏，遠峰隆史氏をはじめとする慶應義塾大学 徳田・村井・楠本・中村・高汐・湧川合同研究会ならびにモービル広域ネットワークプロジェクトの諸氏に感謝します．特に，STREAM 研究グループならびに DVTS プロジェクトの皆様には多大なご協力をいただきました．また，色々と助けていただきました，慶應義塾大学大学院理工学研究科 総合デザイン工学専攻 石神“ ymo ”靖弘氏に感謝します．

様々な場面で多様に関わり合ってきて，ここまで一緒にいろいろとやって来た RG-00 の皆さん，本当にありがとう．これからも何卒よろしくお願いします．

そして，何より公私ともにおいて，様々なご指導・ご助言などをいただきました，慶應義塾大学大学院政策・メディア研究科講師 杉浦一徳博士に深い感謝の意を表します．ここまで一杯一杯で，ある意味楽しかった“ 冬 ”は今までにありませんでした．本当にありがとうございました．

以上を持って，謝辞といたします．

参考文献

[1] Institute of Electrical and Electronics Engineeers, Inc. Carrier Sense Multiple Access with Collision Detection(CSMA/CD) Access Method and Physical Layer Specifications - Media Access Control(MAC) Parameters, Physical Layer and Management Parameters for 10Gb/s Operation. *http://standards.ieee.org/reading/ieee/std/lanman/restricted/802.3ae-2002.pdf*, 2002.

[2] Institute of Electrical and Electronics Engineeers, Inc. IEEE Standard-Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications - Physical Layer Parameters and Specification for 1000 Mb/s Operation over 4-pair of Category 5 Balanced Copper Cabling, Type 1000 BASE-T. *http://standards.ieee.org/reading/ieee/std/lanman/restricted/8 02.3ab-1 999.pdf*, 2000.

[3] *TBS News i*. URL:http://news.tbs.co.jp/.

[4] **文化放送インターネット ラジオ** *BBQR*. URL:http://www.joqr.co.jp/bbqr/.

[5] *Shonan BeachFM 78.9*. URL:http://www.763.fm/fmlive.html.

[6] *impressTV*. URL:http://impress.tv/.

[7] **千葉県インターネット 放送局**. URL:http://www.pref.chiba.jp/stream/index.html.

[8] **いばらきインターネット 放送局**. URL:http://www.pref.ibaraki.jp/movie/.

[9] *Boogie Woogie Cafe*. URL:http://www.boogiewoogiecafe.com/radio/.

[10] *SOI(School of Internet)*. http://www.soi.wide.ad.jp/.

[11] *SFC-GC*. URL:http://gc.sfc.keio.ac.jp/.

[12] Specifications of consumer-use digital vcrś using 6.3mm magnetic tape. 1994. HD Digital VCR Conference.

[13] HDV INFORMATION WEB SITE. *http://www.hdv-info.org/*, Oct 2005.

[14] Moving Picture Experts Group . MPEG-2. *http://www.chiariglione.org/mpeg/standards/mpeg-2/mpeg-2.htm*, Oct 2000.

[15] The ITU Telecommunication Standardization Sector (ITU-T). H.264. *http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-H.264*, 2005.

[16] The Joint Photographic Experts Group. Jpeg2000. *http://www.jpeg.org/jpeg2000/index.html*, 2004.

[17] J. Postel. User Datagram Protocol. RFC 0768, IETF, August 1980.

[18] *Windows Media 9 Series*. URL:http://www.microsoft.com/japan/windowsmedia/.

[19] Real Networks Corporation. Real Networks. *http://www.real.com/*.

[20] S. Keshav. A Control-Theoretic Approach to Flow Control. *Proceedings of ACM SIGCOMM'91*, September 1991.

[21] J. Postel. Transmission Control Protocol. RFC 0793, IETF, September 1981.

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

[23] M. Handley and V. Jacobson. SDP: Session Description Protocol. RFC 2327, IETF, April 1998.

[24] The ITU Telecommunication Standardization Sector (ITU-T). H.261. *http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-H.261*, 2005.

[25] M. Handley, C. Perkins, and E. Whelan. Session Announcement Protocol. RFC 2974, IETF, October 2000.

[26] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, IETF, June 2002.

[27] The ITU Telecommunication Standardization Sector (ITU-T). H.245. *http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-H.245*, 2005.

[28] The ITU Telecommunication Standardization Sector (ITU-T). H.323. *http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-H.323*, 2005.

[29] The ITU Telecommunication Standardization Sector (ITU-T). H.263. *http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-H.263*, 2006.

[30] The ITU Telecommunication Standardization Sector (ITU-T). G.721. *http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-G.721*, 1988.

[31] Moving Picture Experts Group . Mpeg-21. *http://www.chiariglione.org/mpeg/standards/mpeg-21/mpeg-21.htm*, Oct 2002.

[32] Moving Picture Experts Group . Mpeg-7. *http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm*, 2001.

[33] DVTS Consortium. *DVTS (Digital Video Transport System) WWW page*, January 2005. URL:http://www.sfc.wide.ad.jp/DVTS/.

[34] Institute of Electrical and Electronics Engineeers, Inc. IEEE Std 1394-1995 High Performance Serial Bus. Aug 1996.

[35] Institute of Electrical and Electronics Engineeers, Inc. IEEE Std 1394a-2000, IEEE Standard for a High Performance Serial Bus,, Amendment 1. *http://standards.ieee.org/reading/ieee/std/busarch/1394a-2000.pdf*, Jun 2000.

[36] Institute of Electrical and Electronics Engineeers, Inc. IEEE Std 1394b-2002, Amendment to IEEE Std 1394-1995. *http://standards.ieee.org/reading/ieee/std/busarch/1394b-2002.pdf*, Dec 2002.

[37] HD DIGITAL VCR CONFERENCE. Specifications of Consumer-Use Digital VCRs PART2 SD Specifications of Consumer-Use Digital VCRs. *Specifications of Consumer-Use Digital VCRs using 6.3mm magnetic tape*, pages 1–380, Dec 1994.

[38] Ed. S. Bhattacharyya. An Overview of Source-Specific Multicast. RFC 3569, IETF, July 2003.

[39] VideoLAN Project. *http://www.videolan.org/*.

[40] *The Public Netperf Homepage*. URL:http://www.netperf.org/netperf/NetperfPage.html.

[41] Microsoft Corporation. Microsoft DirectX. *http://www.microsoft.com/windows/directx/*.

[42] Microsoft Corporation. COM: Component Object Model Technologies. *http://www.microsoft.com/com/*.

付 録A　RTP

本章では，インターネットを介した映像・音声配信に用いられる RTP について述べる．A.1.2 節では，本機構が受信者からジッタなどの情報を取得するために用いる RTCP についても併せて述べる．

A.1　RTP(Real-time Transport Protocol)

RTP は，インターネット上で映像や音声などのリアルタイム性が重視されるデータの転送に利用されるプロトコルである．RTP は，RFC1889 として IETF にて定義されている．RTP の持つサービスとしては，以下の 5 つの機能がある．

- ペイロードタイプ識別
- シーケンス番号付与
- タイムスタンプ付与
- ジッタ対策
- パケット喪失対策

RTP は，主に UDP 上で動作するプロトコルとして設計されているが，UDP 以外のプロトコル上でも動作可能である．RTP は，マルチキャストにも対応している．

RTP 自身は，到着時間の保証といった QoS 制御は行わない．また，RTP は配送や配送の到達順序自体を保証するプロトコルではない．RTP パケットの受信者は，RTP パケットに含まれるシーケンス番号を利用して，パケットの並び替えや喪失の検知を行うことができる．

RTP パケットフォーマット

RTP パケットヘッダのフォーマットを図 A.1に示す.

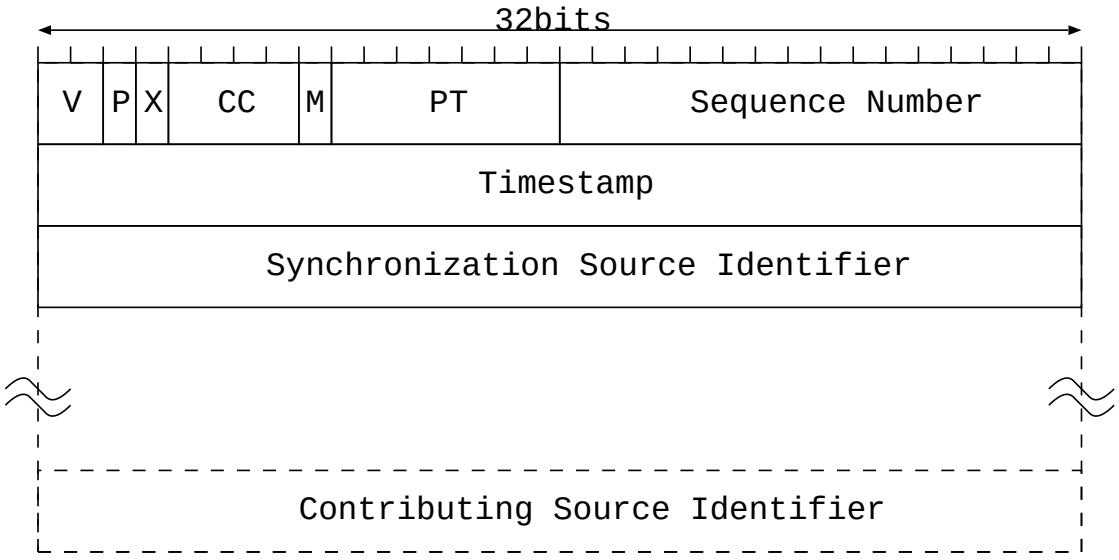

図 A.1: RTP パケットフォーマット

RTP パケットヘッダの各項目の内容を以下に示す.

- V(Version):2bits
  - RTP のバージョン番号を示す．値は 2
- P(Padding):1bit
  - パディングの有無を示す.
- X(eXtension):1bit
  - 拡張ヘッダの有無を示す．拡張ヘッダがある場合，RTP ヘッダの直後に拡張ヘッダが付加される.
- CC(CSRC Count):3bits
  - CSRC カウント．RTP ヘッダの直後に CSRC がいくつあるかを示す.
- M(Marker):1bit
  - マーカービット．ペイロード・タイプごとに使用法が異なる.
- PT(Payload Type):7bits
  - ペイロードタイプを示す．RTP ペイロードの種類を指定する.
- Sequence Number:16bits

  - シーケンス番号を示す．シーケンス番号の初期値は乱数によって指定する．

- Timestamp:32bits
  - タイムスタンプを示す．タイムスタンプのためのクロック基準周波数はペイロード・タイプごとに異なる．

- SSRC:32bits
  - ソース識別子を示す．異なる SSRC 値を用いることにより送信者のネットワークアドレスが同一でも区別できる．

- CSRC:32bits
  - 寄与ソース識別子を示す．RTP ミキサによって統合されたストリームの送信者リスト．

A.2 RTCP(Real-time Transport Control Protocol)

A.1節に挙げた RTP には，以下のような 4 つの問題点がある．

- フロー制御
- クロック同期
- メディア間同期
- 情報ソース識別

RFC1889 には，RTP と同時に RTCP も定義している．RTCP は，RTP セッションのモニタリングとその情報のフィードバックをセッション参加者に伝えるためのプロトコルである．RTCP では，送信者と受信者の間で RTCP パケットを交換することで，上記問題の解決を図る．しかし，RTCP は，RTP のための通知プロトコルであるため，RTCP 自体が何らかのデータを配送したり，TCP のようにエンドノード間での到達性を保証することはできない．

送受信者間で交換される RTCP パケットには以下の 5 つの種類がある．

- SR(Sender Report):
  - RTP セッションにおける送信者での状態通知に使用される RTCP メッセージ

- RR(Receiver Report):
  - RTP セッションにおける送信者以外の参加者における状態通知に使用される RTCP メッセージ

- BYE:
  - セッションからの離脱を通知する RTCP メッセージ

- SDES(Source DEScription):
  - RTP パケットの SSRC/CSRC の値とユーザ情報との関係を通知する RTCP メッセージ
- APP(APPlication):
  - RTCP 規定外アプリケーション固有の制御情報を通知する RTCP メッセージ

RTCP SR/RR パケットフォーマット

RTCP SR パケットのフォーマットを図 A.2に示す.

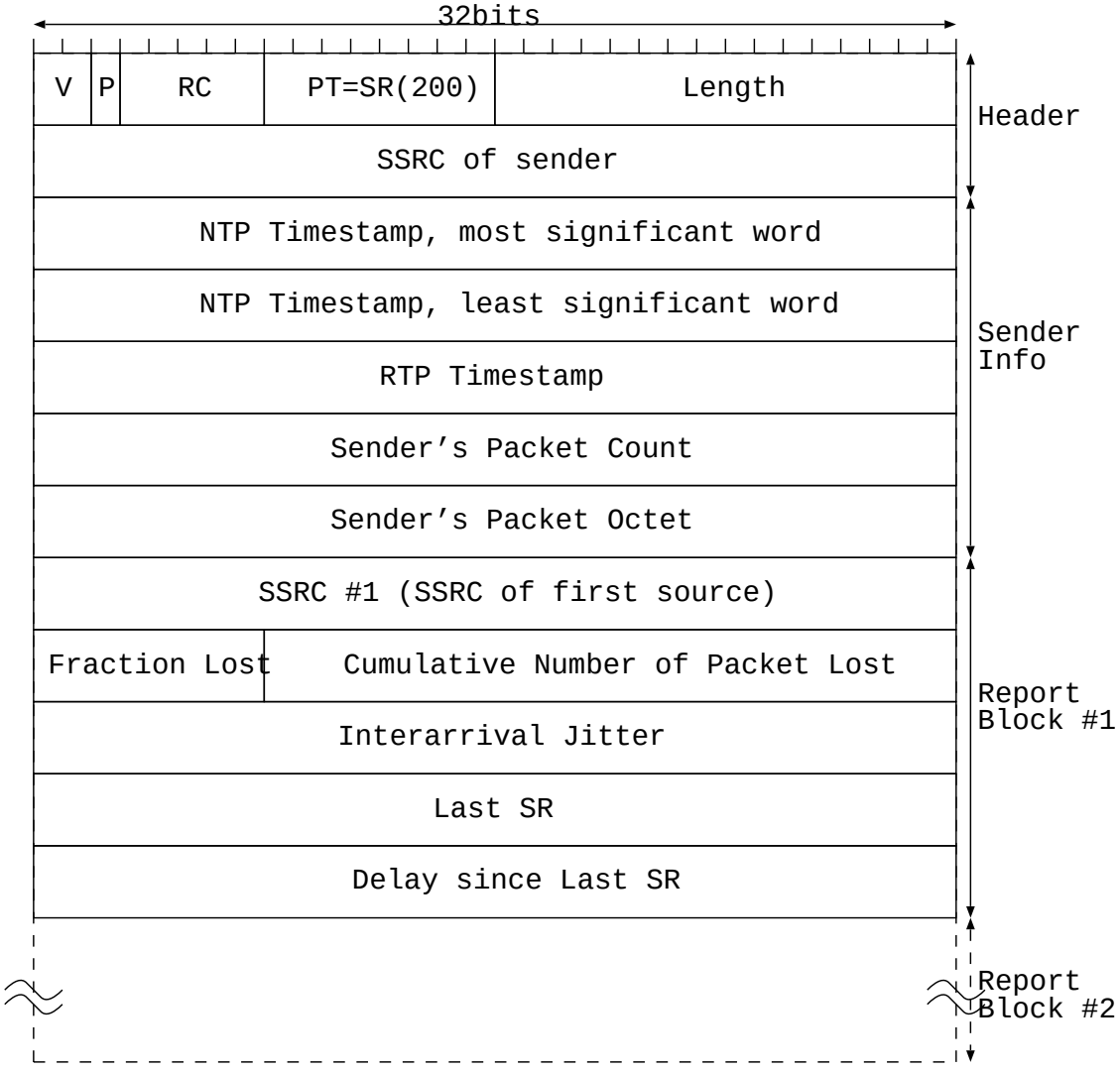

図 A.2: RTCP SR パケットフォーマット

RTCP RR パケットのフォーマットを図 A.3に示す.

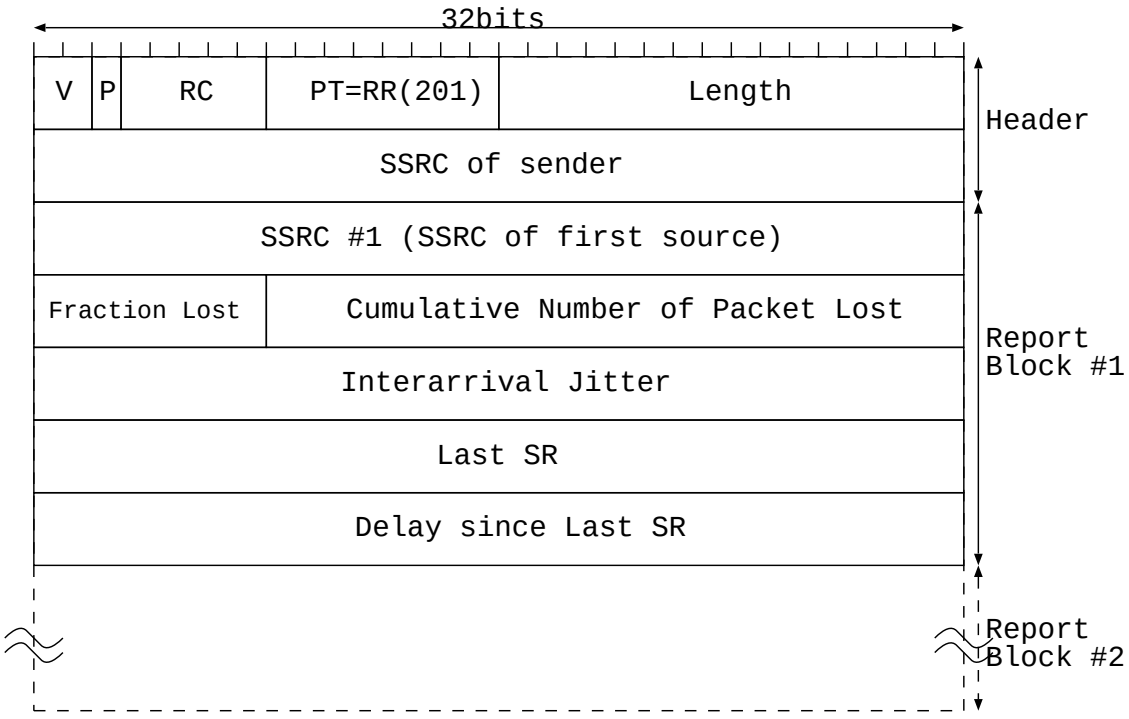

図 A.3: RTCP RR パケットフォーマット

RTCP SR/RR パケットは複数のセクションから構成される．1 つ目のセクションは，ヘッダ部であり，8bytes で構成される．ヘッダ部の構成を以下に示す．

- V(Version):2bits
  - RTP のバージョン番号を示す．値は 2
- P(Padding):1bit
  - パディングの有無を示す．
- RC(Reception Report Type):5bits
  - パケットに含まれる Reception Report 数を示す．
- PT(Packet Type):8bits
  - RTCP パケットの種類を示す．RTCP SR は”200”．RTCP RR は”201”．
- Length:16bits
  - RTCP パケット長を示す．
- SSRC:32bits
  - RTCP SR パケット送信者を特定するための番号を示す．

2つ目のセクションは，SR メッセージと RR メッセージにより異なる．SR メッセージには，送信者から送信されるストリームに関する情報 (Sender Info) が含まれ，20bytes で構成される．なお，RR メッセージの場合，この送信者情報は含まれない．Sender Info 部の構成を以下に示す．

- NTP Timestamp:64bits
  - この RTCP SR パケットが送信された時間を NTP による時間で表したもの．絶対時間も経過時間も取得できない場合は，0 を代入してよい．
- RTP Timestamp:32bits
  - この RTCP SR パケットが送信された時間を RTP タイムスタンプによって表現したもの．
- Sender's Packet Count:32bits
  - セッションが開始されて，この RTCP SR パケットが送信されるまでに送出された RTP パケット数．
- Sender's Octet Count:32bits
  - セッションが開始されて，この RTCP SR パケットが送信されるまでに送出された RTP パケットのペイロード部分の累積バイト数．

これ以降のセクションには，0 個もしくは RTCP ヘッダ部の RC が示す個数分のストリームを受信する受信者に関する情報 (Reception Report) が含まれる．Reception Report 部は，24bytes で構成される．Reception Report 部の構成を以下に示す．

- SSRC:32bits
  - この Reception Report を提供した RTP セッション参加者の SSRC を示す．
- Fraction Lost:8bits
  - 喪失した RTP パケット数を示す．喪失したパケット数を受け取るはずのパケット数で割った値が使用される．
- Cumulative Number of Packet Lost:24bits
  - 喪失した RTP パケット数の合計を示す．
- Extended Highest Sequence Number Received:32bits
  - 上位 16 ビットはシーケンス番号が初期値に戻った回数，下位 16 ビットは最後に受け取った RTP パケットのシーケンス番号をそれぞれ示す．
- Interarrival Jitter:32bits
  - 受け取った RTP パケットの到着時間の揺らぎ (ジッタ) を示す．

- LSR(Last SR timestamp):32bits
  - 最後に受信した RTCP SR パケットに含まれていた NTP タイムスタンプの値の 16 ビット目から 47 ビット目までの 32 ビットを示す.
- DLSR(Delay since Last SR):32bits
  - 最後に RTCP SR パケットを受信してからこのパケットを送信するまでの時間を, 1/65536 秒単位で表したもの.

付 録B　AMSS (Adaptive Media Streaming System)

本章では，プロトタイプ実装を行った AMSS のシステム的な特徴と利用する技術の詳細を述べる．また，実装を行った各メディアフォーマットの配信モジュールについても述べる．

B.1　Windows オペレーティングシステムにおける AMSS の特徴

AMSS は，Microsoft Windows オペレーティングシステム上で動作するメディア配信システム DVTS for Windows に対して必要な機能の拡張を行う形で実現した．Windows オペレーティングシステム上で実装された AMSS は以下の特徴を持つ．

- GUI をベースとしたユーザフレンド リなインタフェース設計
  - ステータスバーを介した状況通知
  - ツールバーを介した直感的操作
  - タスクバーを介したリアルタイムメッセージ通知
- DirectShow を用いた多メディア対応とシステムへのダイレクトアクセス
- ウィンドウメッセージを利用したプロセス間メッセージング

B.2　DirectShow テクノロジ

AMSS ならびに DVTS for Windows は，Microsoft Windows オペレーティングシステム上でマルチメディア処理を実現するために，Microsoft 社によるマルチメディア拡張 API である DirectShow テクノロジ [41] を利用する．DirectShow テクノロジを利用することにより，映像・音声データのストリーミング再生が可能となる．また，ハードウェアに対して，直接描画を行うための DirectDraw 機能へのインタフェースも備えるため，ビデオカードにおける描画資源を有効に活用することも可能である．DirectShow では，ハードウェアの抽象化を行うため，ハードウェアの差異を意識することなくアプリケーション開発が可能である．

B.3　DirectShow と COM

DirectShow は，Windows に実装されている Comonent Object Model (COM)[42] に基づく．DirectShow アプリケーションでは，この COM クライアントモデルに従って実装される．また，

各アプリケーションでは，DirectShow に用いられる COM に従った COM オブジェクトを実装することで，DirectShow に対する機能拡張を行うコンポーネントの開発が可能である．

COM オブジェクトは，自オブジェクトへのインタフェースとしてピンが実装される．COM オブジェクトの例を図 B.1に示す．この例では，1 つの入力ピンと 1 つの出力ピンで構成されるオブジェクトを示している．

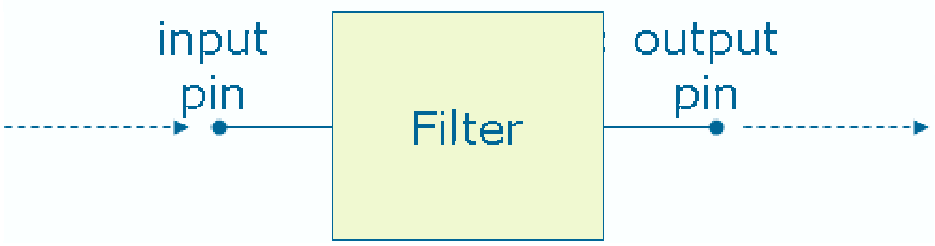

図 B.1: COM オブジェクトの例

COM オブジェクトが持つピンには，必ず CLSID(ClassID) ないしは IID(InterfaceID) を持ち，この値は一意に決められたものである．他の ID と混同しない値の定義を行う．アプリケーションは，この ID を元にピンのインタフェース検索を行う．

DirectShow では，この COM オブジェクトはフィルタと呼ばれる．アプリケーション開発者は，このフィルタを独自に開発することで様々なメディアフォーマットに対応した Windows アプリケーションを実装可能である．

B.4 AMSS と DirectShow/COM

AMSS の送信ノードにおけるフィルタの接続構成を図 B.2に示す．また，受信ノードにおけるフィルタの接続構成を図 B.3に示す．本図では，メディアフォーマットとして DV を用いる場合の例を示している．

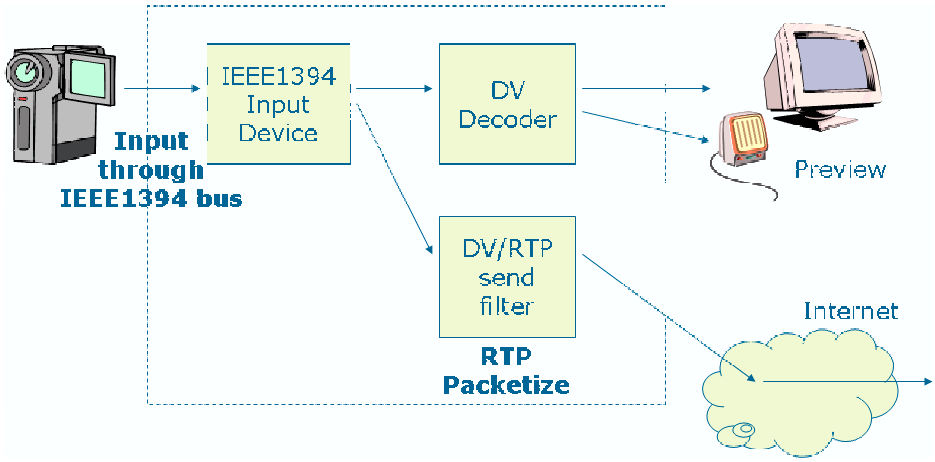

図 B.2: 送信ノードでの処理のモデル (DVTS)

IEEE1394 インタフェースから入力されたデータは，IEEE1394 入力デバイスフィルタを介して取得可能である．入力デバイスフィルタは，直接ハードウェアを制御する．入力デバイスか

ら入力されたデータは，DV/RTP 送信フィルタを介して IP パケット化され，ネットワークを通じて受信ノードに到達する．また，この例では，送信データのプレビューを行うために，受信データを複製し，DV デコーダフィルタ・ビデオ出力フィルタを通してモニタにデータを表示させている．

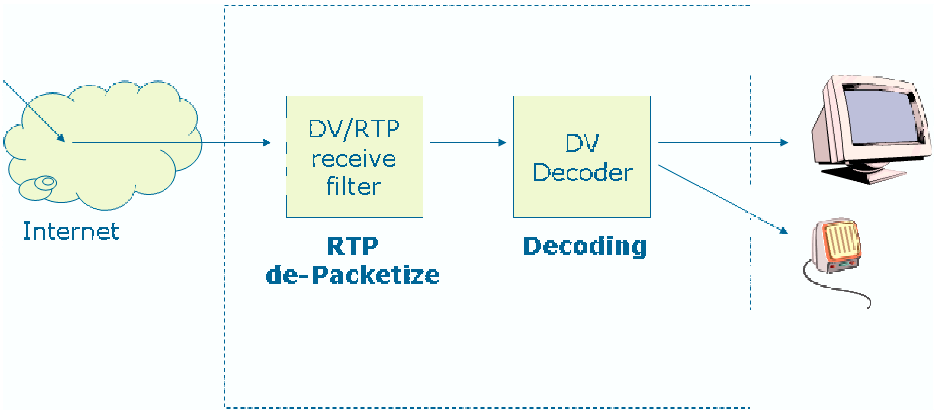

図 B.3: 受信ノードでの処理のモデル (DVTS)

受信ノードでは，ネットワークを通じて受信した IP パケットを，一度 DV/RTP 受信フィルタに入力させる．その上で，映像・音声データのデコードを行うフィルタにデータを送信し，最終的にモニタや音声デバイスといった出力デバイスに対してデータが送信される．これにより，受信データの表示が行われる．

本研究では，複数メディアフォーマットへの対応を行うため，複数の RTP 送受信フィルタを実装した．今回開発を行ったメディアフィルタは，以下の通りである．

B.5 DV/RTP フィルタ

B.5.1 DV/RTP パケットの配信に用いるフィルタ

DV フォーマットの配信を行う際に用いるフィルタは以下の通りである．

- DV/RTP Send Filter
- DV/RTP Receive Filter
- 周辺フィルタ
  - DV Capture Device Filter (MSDV)
  - DV Splitter Filter
  - DV Video Decoder Filter

B.5.2 DV/RTP フィルタ

DV/RTP フィルタが持つ CLSID ならびに IID は以下の通りである．

```
// CLSID_DvRtpSendFilter
// {B6D957EF-E1EF-4f1b-B37E-61C1C28235DA}
DEFINE_GUID(CLSID_DvRtpSendFilter,
0xb6d957ef, 0xe1ef, 0x4f1b, 0xb3, 0x7e, 0x61, 0xc1, 0xc2, 0x82, 0x35, 0xda);

// IID_IDvRtpSendFilter
// {A745A26C-ED9E-4d74-A3F5-2360736A7049}
DEFINE_GUID(IID_IDvRtpSendFilter,
0xa745a26c, 0xed9e, 0x4d74, 0xa3, 0xf5, 0x23, 0x60, 0x73, 0x6a, 0x70, 0x49);

// CLSID_DvRtpSendFilterProperty
// {C654929E-7BF0-4e26-B42D-373C27B07CC8}
DEFINE_GUID(CLSID_DvRtpSendFilterProperty,
0xc654929e, 0x7bf0, 0x4e26, 0xb4, 0x2d, 0x37, 0x3c, 0x27, 0xb0, 0x7c, 0xc8);
```

図 B.4: 送信フィルタの持つ ID

```
// CLSID_DVRTPRecvFilter
// {27A324FE-6FEF-4e9d-955F-E2C894C05B9A}
DEFINE_GUID(CLSID_DVRTPRecvFilter,
0x27a324fe, 0x6fef, 0x4e9d, 0x95, 0x5f, 0xe2, 0xc8, 0x94, 0xc0, 0x5b, 0x9a);

// IID_IDVRTPRecvFilter
// {8687A659-5ED8-43ca-8A6C-B5521E0AB77B}
DEFINE_GUID(IID_IDVRTPRecvFilter,
0x8687a659, 0x5ed8, 0x43ca, 0x8a, 0x6c, 0xb5, 0x52, 0x1e, 0xa, 0xb7, 0x7b);

// CLSID_DVRTPRecvFilterProperty
// {0CDD95E3-1ED9-4790-AE6B-FD031624C69A}
DEFINE_GUID(CLSID_DVRTPRecvFilterProperty,
0xcdd95e3, 0x1ed9, 0x4790, 0xae, 0x6b, 0xfd, 0x3, 0x16, 0x24, 0xc6, 0x9a);
```

図 B.5: 受信フィルタの持つ ID

B.6 HDV/RTP フィルタ

B.6.1 HDV/RTP パケットの配信に用いるフィルタ

HDV フォーマットの配信を行う際に用いるフィルタは以下の通りである.

- HDV/RTP Send Filter
- HDV/RTP Receive Filter
- 周辺フィルタ
  - Tape Device Capture Filter (MSTAPE)
  - MPEG2 Demultiplexor Filter
  - MPEG2 Video Decoder Filter
  - MPEG2 Audio Decoder Filter

B.6.2 HDV/RTP フィルタ

HDV/RTP フィルタが持つ CLSID ならびに IID は以下の通りである.

```
// CLSID_HDvRtpSendFilter
// {38E15BE8-11CF-4387-9C13-78BD7ABE7840}
DEFINE_GUID(CLSID_HDvRtpSendFilter,
0x38e15be8, 0x11cf, 0x4387, 0x9c, 0x13, 0x78, 0xbd, 0x7a, 0xbe, 0x78, 0x40);

// IID_IHDvRtpSendFilter
// {CB2A6256-4093-4f1e-8760-62E34EB88739}
DEFINE_GUID(IID_IHDvRtpSendFilter,
0xcb2a6256, 0x4093, 0x4f1e, 0x87, 0x60, 0x62, 0xe3, 0x4e, 0xb8, 0x87, 0x39);

// CLSID_HDvRtpSendFilterProperty
// {D1C9FC9F-7295-4a29-B998-13E28C4DA776}
DEFINE_GUID(CLSID_HDvRtpSendFilterProperty,
0xd1c9fc9f, 0x7295, 0x4a29, 0xb9, 0x98, 0x13, 0xe2, 0x8c, 0x4d, 0xa7, 0x76);
```

図 B.6: 送信フィルタの持つ ID

```
// CLSID_HDVRTPRecvFilter
// {D36B5D64-867F-4a13-8F50-54ABACD0AE53}
DEFINE_GUID(CLSID_HDVRTPRecvFilter,
0xd36b5d64, 0x867f, 0x4a13, 0x8f, 0x50, 0x54, 0xab, 0xac, 0xd0, 0xae, 0x53);

// IID_IHDVRTPRecvFilter
// {A02E6E4F-9A9E-4b41-AA84-40D47D8B8F50}
DEFINE_GUID(IID_IHDVRTPRecvFilter,
0xa02e6e4f, 0x9a9e, 0x4b41, 0xaa, 0x84, 0x40, 0xd4, 0x7d, 0x8b, 0x8f, 0x50);

// CLSID_HDVRTPRecvFilterProperty
// {6B0E29FA-ED77-400f-9C9A-94FAFD44AA2C}
DEFINE_GUID(CLSID_HDVRTPRecvFilterProperty,
0x6b0e29fa, 0xed77, 0x400f, 0x9c, 0x9a, 0x94, 0xfa, 0xfd, 0x44, 0xaa, 0x2c);
```

図 B.7: 受信フィルタの持つ ID

B.7 J2K/RTP フィルタ

B.7.1 J2K/RTP パケットの配信に用いるフィルタ

Motion-JPEG2000(J2K) フォーマットの配信を行う際に用いるフィルタは以下の通りである.

- J2K/RTP Send Filter
- J2K/RTP Receive Filter
- 周辺フィルタ

- USB Capture Device Filter
- Motion-JPEG2000 Video Decoder Filter (LEAD MJ2K Decoder)
- Motion-JPEG2000 Video Encoder Filter (LEAD MJ2K Encoder)

B.7.2 Motion-JPEG2000/RTP フィルタ

J2K/RTP フィルタが持つ CLSID ならびに IID は以下の通りである.

```
// CLSID_J2KRTPSendFilter
// {3974E5F4-397B-4683-8BE4-8042248FED5A}
static const GUID CLSID_J2KRTPSendFilter =
{ 0x3974e5f4, 0x397b, 0x4683, { 0x8b, 0xe4, 0x80, 0x42, 0x24, 0x8f, 0xed, 0x5a } };

// IID_IJ2KRTPSendFilter
// {94207E6B-568F-4f81-9D01-72030CE01C35}
DEFINE_GUID(IID_IJ2KRTPSendFilter,
0x94207e6b, 0x568f, 0x4f81, 0x9d, 0x1, 0x72, 0x3, 0xc, 0xe0, 0x1c, 0x35);

// CLSID_J2KRTPSendFilterProperty
// {ECEFC0DC-2001-427c-82F2-0E2981DC7EE5}
DEFINE_GUID(CLSID_J2KRTPSendFilterProperty,
0xecefc0dc, 0x2001, 0x427c, 0x82, 0xf2, 0xe, 0x29, 0x81, 0xdc, 0x7e, 0xe5);
```

図 B.8: 送信フィルタの持つ ID

```
// CLSID_J2KRTPRecvFilter
// {9B81C696-958E-4951-95ED-4426A99D02B5}
static const GUID CLSID_J2KRTPRecvFilter =
{ 0x9b81c696, 0x958e, 0x4951, { 0x95, 0xed, 0x44, 0x26, 0xa9, 0x9d, 0x2, 0xb5 } };

// IID_IJ2KRTPRecvFilter
// {50A91D22-DF1B-4630-9DEB-F5CF160F1EF3}
DEFINE_GUID(IID_IJ2KRTPRecvFilter,
0x50a91d22, 0xdf1b, 0x4630, 0x9d, 0xeb, 0xf5, 0xcf, 0x16, 0xf, 0x1e, 0xf3);

// CLSID_J2KRTPRecvFilterProperty
// {31646D92-37E4-4d9a-A07B-89792C231F27}
DEFINE_GUID(CLSID_J2KRTPRecvFilterProperty,
0x31646d92, 0x37e4, 0x4d9a, 0xa0, 0x7b, 0x89, 0x79, 0x2c, 0x23, 0x1f, 0x27);
```

図 B.9: 受信フィルタの持つ ID

B.8 Flea/RTP フィルタ

B.8.1 非圧縮ビットマップ/RTP パケットの配信に用いるフィルタ

非圧縮ビットマップフォーマットの配信を行う際に用いるフィルタは以下の通りである.

- Flea/RTP Send Filter
- Flea/RTP Receive Filter
- 周辺フィルタ
  - USB Capture Device Filter

B.8.2 Flea/RTP フィルタ

Flea/RTP フィルタが持つ CLSID ならびに IID は以下の通りである.

```
// CLSID_FleaRTPSendFilter
// {74D4EC5D-E83B-46a1-AD78-A9290DEC79EA}
DEFINE_GUID(CLSID_FleaRTPSendFilter,
0x74d4ec5d, 0xe83b, 0x46a1, 0xad, 0x78, 0xa9, 0x29, 0xd, 0xec, 0x79, 0xea);

// IID_IFleaRTPSendFilter
// {FBD5540F-376C-45a8-A458-0DB50B8A3DB1}
DEFINE_GUID(IID_IFleaRTPSendFilter,
0xfbd5540f, 0x376c, 0x45a8, 0xa4, 0x58, 0xd, 0xb5, 0xb, 0x8a, 0x3d, 0xb1);

// CLSID_FleaRTPSendFilterProperty
// {720C47A9-F84E-4ea1-8E89-9C97E62B0165}
DEFINE_GUID(CLSID_FleaRTPSendFilterProperty,
0x720c47a9, 0xf84e, 0x4ea1, 0x8e, 0x89, 0x9c, 0x97, 0xe6, 0x2b, 0x1, 0x65);
```

図 B.10: 送信フィルタの持つ ID

```
// CLSID_FleaRTPRecvFilter
// {85B97576-14D2-415b-A08E-8529B6FBE4BE}
DEFINE_GUID(CLSID_FleaRTPRecvFilter,
0x85b97576, 0x14d2, 0x415b, 0xa0, 0x8e, 0x85, 0x29, 0xb6, 0xfb, 0xe4, 0xbe);

// IID_IFleaRTPRecvFilter
// {E3AE5F72-B0D2-4dbc-ADAD-9FF4A48B6377}
DEFINE_GUID(IID_IFleaRTPRecvFilter,
0xe3ae5f72, 0xb0d2, 0x4dbc, 0xad, 0xad, 0x9f, 0xf4, 0xa4, 0x8b, 0x63, 0x77);

// CLSID_FleaRTPRecvFilterProperty
// {1204973A-D22C-4efd-8E31-C600FDC1BCEC}
DEFINE_GUID(CLSID_FleaRTPRecvFilterProperty,
0x1204973a, 0xd22c, 0x4efd, 0x8e, 0x31, 0xc6, 0x0, 0xfd, 0xc1, 0xbc, 0xec);
```

図 B.11: 受信フィルタの持つ ID