Title	ビデオネットワークにおける相互接続を考慮した資源管理機構に関する研究
Author(s)	牧野，義樹
Citation	
Issue Date	2003-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/1686
Rights	
Description	Supervisor:丹 康雄，情報科学研究科，修士

修 士 論 文

ビデオネットワークにおける相互接続を考慮した資源管理機構に関する研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

牧野 義樹

2003 年 3 月

修 士 論 文

ビデオネットワークにおける相互接続を考慮した資源管理機構に関する研究

指導教官　丹康雄 助教授

審査委員主査　丹康雄 助教授
審査委員　篠田陽一 教授
審査委員　日比野靖 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

910104 牧野 義樹

提出年月: 2003 年 2 月

概 要

本稿では、異なった技術を利用したビデオネットワークや異なる組織により管理されるビデオネットワーク間を接続するための資源管理機構の設計と実装について提案する。

目 次

第 1 章　はじめに **1**

第 2 章　ビデオネットワーク **2**
2.1　ビデオネットワークに必要な機能 2
2.2　媒体占有型のビデオネットワーク 2
2.3　媒体共有型のビデオネットワーク 3
2.3.1　IP ネットワークにおけるビデオネットワーク技術 3
2.3.2　IEEE1394 におけるビデオネットワーク 6

第 3 章　ビデオネットワークの相互接続 **10**
3.1　アドレス体系 10
3.1.1　アドレスと識別子 10
3.1.2　アドレスとデバイスの存在場所 10
3.1.3　アドレスと名前 11
3.2　シグナリング 11
3.3　伝送に利用されるプロトコル 11
3.3.1　IEEE1394 プロトコルスタック 11
3.3.2　RTP プロトコルスタック 12
3.4　ビデオフォーマット 13
3.4.1　圧縮方式の相違 14
3.4.2　映像の表示サイズ 14
3.4.3　アスペクト比 14
3.4.4　フレーム数 15
3.5　まとめ 17

第 4 章　ビデオインターネットワーキングアーキテクチャ **18**
4.1　概要 18
4.2　アドレス体系 19
4.2.1　アドレス解決 20
4.3　コネクションとセッション 21
4.4　デバイスコントローラ 22

4.4.1 バーチャルビデオノード 22
4.4.2 デバイスコントローラの動作 27
4.5 ビデオゲートウェイ 27
4.5.1 構成 28
4.5.2 リソースマネージャへの登録 29
4.6 リソースマネージャ 29
4.6.1 デバイスマネージャ 29
4.6.2 コネクションマネージャ 30
4.6.3 セッションマネージャ 31
4.6.4 ルーティングエンジン 32
4.7 メッセージの概要 33
4.7.1 デバイスコントローラとリソースマネージャ間のメッセージ 33

第 5 章 JAIST VideoLAN と DVTS を対象とした実装 35
5.1 対象となるビデオネットワーク 35
5.1.1 JAIST VideoLAN 35
5.1.2 DVTS 36
5.2 ノードアドレスとネットワークアドレス 37
5.3 リソースマネージャとデバイスコントローラに共通するクラス 39
5.3.1 メッセージの送受信 39
5.3.2 プログラム内でのメッセージの配送 39
5.3.3 他オブジェクトへの情報通知のためのクラス 40
5.4 リソースマネージャの概要 41
5.4.1 リソースマネージャを構成する主なクラス 41
5.4.2 データベース 41
5.5 デバイスコントローラの概要 45
5.5.1 デバイスコントローラで利用されるクラス 45
5.6 JAIST VideoLAN におけるデバイスコントローラ 47
5.6.1 アドレス変換 47
5.6.2 デバイスの検出 47
5.6.3 ターミナルシステムとデバイスコントローラの通信 47
5.7 DVTS におけるデバイスコントローラ 49
5.7.1 アドレス変換 49
5.7.2 DVTS コマンドの実行 49
5.8 ビデオゲートウェイの構成 49
5.9 動作 49
5.9.1 デバイスコントローラの登録と抹消 50
5.9.2 バーチャルビデオノードの登録と抹消 50
5.9.3 ビデオゲートウェイの登録と抹消 50

5.9.4 コネクションの確立と切断 51
5.9.5 セッションの確立と切断 51
5.9.6 経路情報の交換 51
5.9.7 デバイスコントローラとリソースマネージャ間のメッセージ 52
5.9.8 リソースマネージャ間のメッセージ 55
5.10 接続実験 55

第 6 章 考察と今後の課題 60
6.1 資源管理機構の分散化 60
6.2 ビデオゲートウェイの必要数 61
6.3 経路の検索 62
6.4 障害とリソースマネージャの冗長性 62
6.5 マルチキャスト 63
6.6 プロダクションレベルでの利用 63

第 7 章 おわりに 65

付 録 A デバイスコントローラとリソースマネージャの両方の処理に関係するクラス 67
A.1 NetAddress 67
A.2 NetAddress 68
A.3 Peer 70
A.4 Message 71
A.5 Socket 73
A.6 Dispatcher 74
A.7 DispatchReceiver 75
A.8 NotificationReceiver 75

付 録 B デバイスコントローラに関する処理を行うクラス 77
B.1 DeviceController 77
B.2 DeviceDetectorInfo 81
B.3 DeviceDetector 82
B.4 VirtualVideoNode 82

付 録 C リソースマネージャに関する処理を行うクラス 85
C.1 DeviceControllerInfo 85
C.2 VirtualVideoNodeInfo 85
C.3 ConnectionInfo 86
C.4 SessionInfo 87
C.5 NeighborInfo 87
C.6 ResourceManager 88

C.7 DeviceManager 89
C.8 ConnectionManager 93
C.9 SessionManager 96
C.10 RoutingEngine 99

付 録 D メッセージ **102**

第1章　はじめに

近年、コンピュータネットワークや家庭内ネットワークが一般的になりつつある。また、これらのネットワークは高速化が目覚しい勢いですすんでおり、今まででは考えることもできなかったほどの転送レートが実現されている。

高速化が実現される一方で ATM や IEEE1394 といった QoS(Quarity Of Service) を保証したネットワークも一般的に利用されている。QoS はネットワーク品質を保証するのに非常に重要なものであり、IP ネットワークのような最善努力型のプロトコルであっても DiffServ や IntServ といった技術が開発されてきている。

このような技術の進歩によって、ネットワークを介して高品質なビデオデータの転送が可能となっている。RTP などを利用し、品質保証はないが十分な帯域の経路を利用することで高品質なビデオデータの転送を IP ネットワーク上で実現するものや IEEE1394 などの QoS 保証をサポートするネットワークを利用しビデオデータの転送を行うものなど多様である。

一般的に、これらのビデオネットワークは個々で閉じたシステムとなっており相互接続に関して考えられていない。そこで本稿ではビデオネットワークの相互接続に関して考察を行い、ビデオネットワーク間を接続するシステムとして各ビデオネットワークが資源管理機構により管理されるビデオインターネットワーキングアーキテクチャを提案する。

第2章　ビデオネットワーク

ビデオネットワークとはビデオデータを転送するために特化されたネットワークである。本章では本稿がターゲットとするビデオネットワークについて説明する。

2.1　ビデオネットワークに必要な機能

ビデオネットワーク内にはさまざまなビデオ機器が存在するが、それらはビデオデータの送信あるいは受信をするものでなければならない。送信側の機器としてはビデオカメラなどがあり、受信側の機器としてテレビなどが接続されることが考えられる。ビデオネットワーク内で転送されるビデオデータはどのようなものでも問題はなく動画を送ることができさえすればよい。ただし、コンピュータからの制御信号等によってビデオネットワーク内に存在する送信者となるビデオ機器と受信者となるビデオ機器の接続を簡単に変更することができなければならない。

2.2　媒体占有型のビデオネットワーク

媒体占有型のビデオネットワークにおけるビデオ機器間の接続は図 2.1 のように直接接続することが一般的である。しかし、マトリックススイッチャなどを利用して図 2.2 のようにビデオ機器を接続し、自由に接続状態を変更する事も可能である。マトリックススイッチャ内部の接続を外部からシリアルなどを利用して操作可能であればこれはビデオネットワークであると言える。このようなビデオネットワークではビデオ機器間の接続はマトリックススイッチャのみに依存し、末端となるビデオ機器は接続先を知ることがなくビデオデータを媒体に送信、あるいは受信するだけである。

図 2.1: 媒体占有型なビデオ機器の直接接続

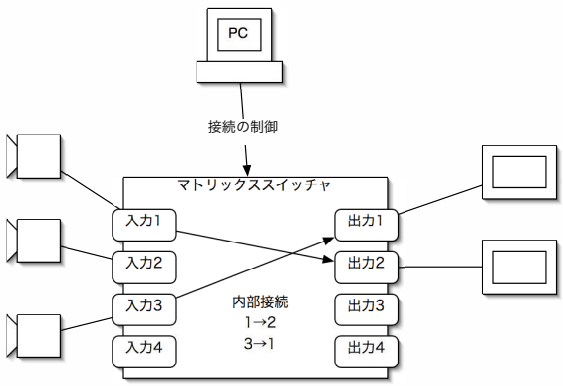

図 2.2: 媒体占有型のマトリックススイッチャを使った接続

2.3 媒体共有型のビデオネットワーク

媒体共有型のビデオネットワークでは一つの媒体であっても多重化することで複数のビデオデータを転送することができる。IP ネットワークを利用したビデオネットワークなどがこれにあたる。このようなビデオネットワークでは送信側は宛先のビデオ機器を指定してビデオデータを送信しなければならない。もし指定せずに送ることができてもビデオデータが多重化された時点ですべてのビデオデータが混じり合い意味のないものになってしまう。

ビデオデータが正しく転送されるために二つの作業が必要になる。送信側と受信側で送受信の設定を行うシグナリングと呼ばれる作業と実際にビデオデータを転送するという作業である。シグナリングは媒体占有型におけるマトリックススイッチャの設定に相当する。ここでは IP ネットワークにおけるビデオネットワークで利用される技術について説明する。また、ビデオ機器などにインターフェースが搭載されていることが多くビデオデータの転送に適した技術である IEEE1394 についても説明する。

2.3.1 IP ネットワークにおけるビデオネットワーク技術

IP ネットワークを利用したビデオネットワークでビデオデータを転送する際には Real-time transport protocol(RTP)[1] と RTP control protocol(RTCP) と呼ばれるプロトコルが利用されるのが一般的である。シグナリングプロトコルとしては Resource ReSerVation Protocol(RSVP)[2] や Session Initiation Protocol(SIP)[3] などが利用されている。

ビデオデータ転送のためのプロトコル

ビデオデータの転送のために一般的に利用されるプロトコルは RTP である。RTP はさまざまなビデオフォーマットに対応したプロトコルであるが、RTP がどのように利用され

るかはビデオフォーマットによって異なる。そこで、プロファイルと呼ばれる規格によって各ビデオデータ転送でどのように RTP が利用されるかを規定している。

RTP は UDP を利用するトランスポート層のプロトコルとみなすことができる。ビデオデータのようなリアルタイム性を持ったデータを転送するために必要である情報を RTP ヘッダに格納することで、リアルタイムデータ転送のサポートを行う。RTP ヘッダの構成は図 2.3 のようになる。

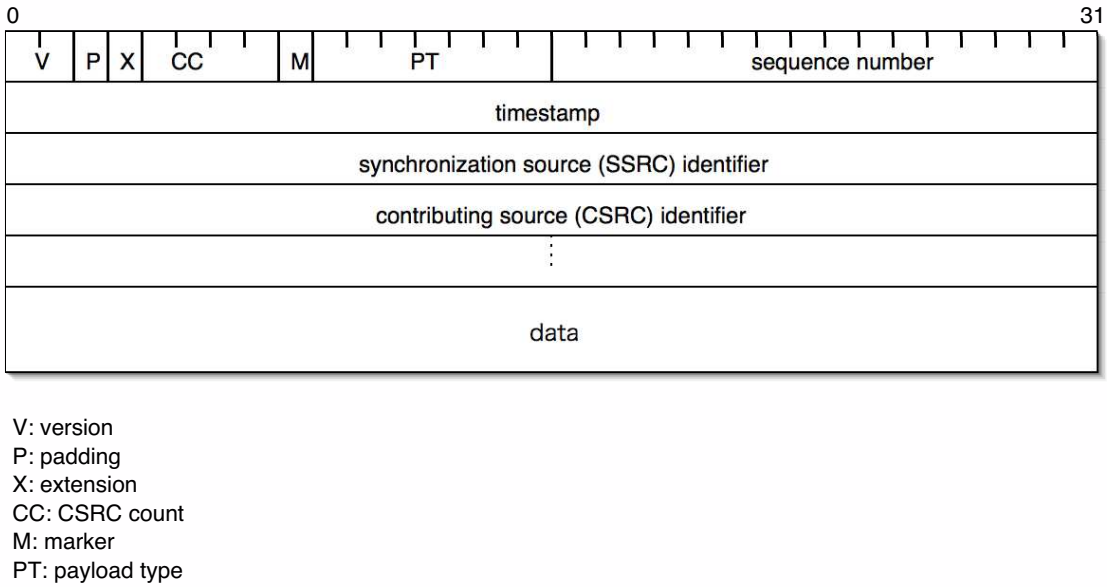

図 2.3: RTP ヘッダ

version 現在の RTP では 2 が入る

padding ペイロードが実際の長さより長く詰物がしてある場合は 1 になる

extension 拡張ヘッダが存在する場合は 1 になる

CSRC count CSRC identifier の数が入る

marker プロファイル依存の情報である

payload type ペイロードのタイプが入る

sequence number パケットの通し番号である

timestamp データが抽出された時刻が入る

SSRC ストリームのソースを識別する値が入る

CSRC 結合されたストリームを識別するための値が入る

RTP はリアルタイムデータを転送するだけであり、データ転送の遅延やロス率などの転送状況に関する情報の送受信は RTCP プロトコルを用いることで行われる。送信者は受信者から送信された RTCP を調べることで、ネットワークの輻輳を回避するために画質は下がるが送信データの圧縮率を上げる、といったことが可能となる。

シグナリングのためのプロトコル

ここでは RSVP と SIP について簡単に説明する。

RSVP は接続先の相手までリアルタイムデータなどを転送するために必要な資源をルータ上に予約するために利用されるプロトコルである。これによって、データを送信したい相手までの通進路が確保されるため一定の品質を保ったデータ通信が可能となる。RSVP は仮想的に媒体占有のケーブルを作り出す作業とみなすことができる。RSVP によってルータが設定されるまでの動作を図 2.4 に示す。RSVP で資源予約を行いたい送信装置は最初に最寄りの RSVP 対応ルータに PATH メッセージを転送する。PATH メッセージにはルータが保証しなければならない資源に関する情報や PATH メッセージが通過してきたルータの情報が格納されており、いくつかのルータを介して受信装置が PATH を受信する。次に受信装置は RESV メッセージを送信する。RESV メッセージは PATH メッセージが通過してきた順番と逆にルータを経由して送信装置まで転送される。RESV メッセージをルータが受信すると各ルータ内で資源予約が行われる。

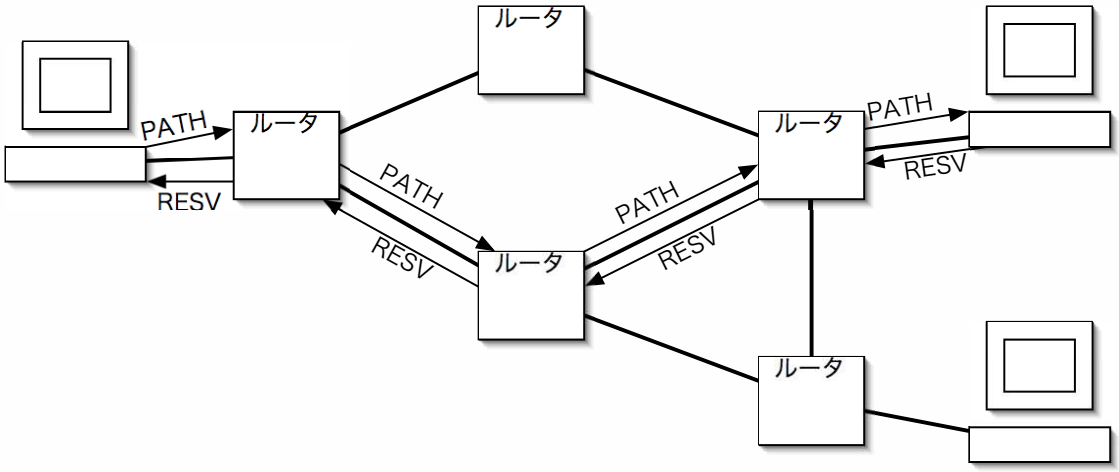

図 2.4: RSVP による資源予約

一方 SIP は送信側と受信側でセッションを開始させるためのプロトコルで、Voice over IP(VoIP) で利用されるようになってきた。SIP で転送されるデータはテキストデータで HTTP に似たものとなっている。SIP では通信相手を指定するのに URI や ENUM と呼ばれる電話番号に対応したアドレスを利用する。また、SIP はセッションを開始する前に転

送可能なメディアなどの情報交換を行うことで何のデータをどのように送信するか決定する。SIP サーバを利用する一般的な SIP の動作を図 2.5 に示す。

セッションを開始したい機器は INVITE メッセージを送信する。INVITE メッセージは一般的に最初に SIP プロキシサーバなどに送られてから宛先まで送信される。INVITE メッセージを受信した機器は音を鳴らしたり画面に何らかの表示をすることでユーザにセッション要求があることを知らせる。受信側の機器でユーザがセッションの開始を許可すると OK が INVITE の送信者に送信される。そののち送信者は ACK を送信しセッションが開始される。また、セッションを切断するために BYE メッセージが利用される。

2.3.2 IEEE1394 におけるビデオネットワーク

IEEE1394 として現在一般的に利用されているものは 400Mbps のものであり、アシンクロナス通信とアイソクロナス通信という二つの転送方式をサポートする。アイソクロナス通信では IEEE1394 に接続されたビデオ機器は 125 μ 秒に一度データを転送することが保証されており、ビデオデータなどのリアルタイム性が重視されるデータを送受信するのに適した方式となっている。このような利点から、IEEE1394 は映像を扱う家電機器間を接続するバスとして利用されている。

IEEE1394 のケーブル内ではアイソクロナスデータは多重化されており、それらの一つ一つはチャネルと呼ばれる番号で識別される。そのためビデオデータを送受信したい機器は前もってチャネルと帯域を確保しておかねばならない。チャネルや利用されている帯域の管理はアイソクロナスリソースマネージャと呼ばれる機器が行っている。

IEEE1394 で利用されるビデオネットワークとしては HAVi[4] が存在する。HAVi に接続するデバイスは以下の四つにわけられる。

- FAV (Full AV Devices)
- IAV (Intermediate AV devices)
- BAV (Base AV devices)
- LAV (Legacy AV devices)

LAV は HAVi に対応していないデバイスである。LAV と BAV は他のデバイスをコントロールすることはできず IAV や FAV からコントロールされるものとなる。IAV や FAV は他のデバイスをコントロールするための DCM や FCM と呼ばれるソフトウェアを既に持っているか新たにダウンロードしインストールすることで LAV や BAV、また IAV や FAV をコントロールすることができる。IAV と FAV の内部には複数の HAVi コンポーネントが存在し、これらはメッセージ通信を行うことで他のコンポーネントと情報のやりとりを行う。

ユーザインターフェースを画面に表示し、ユーザからの命令を受信するテレビを FAV や IAV デバイスとして構築可能である。HAVi に対応しているビデオデッキや DVD プレー

ヤ、あるいはLAVであってもテレビが対応しているデバイスであれば、IEEE1394を利用してテレビに接続することでテレビの画面からこれらのデバイスをコントロールする事が可能となる。このようなテレビのHAViの構成を図2.6に示す。

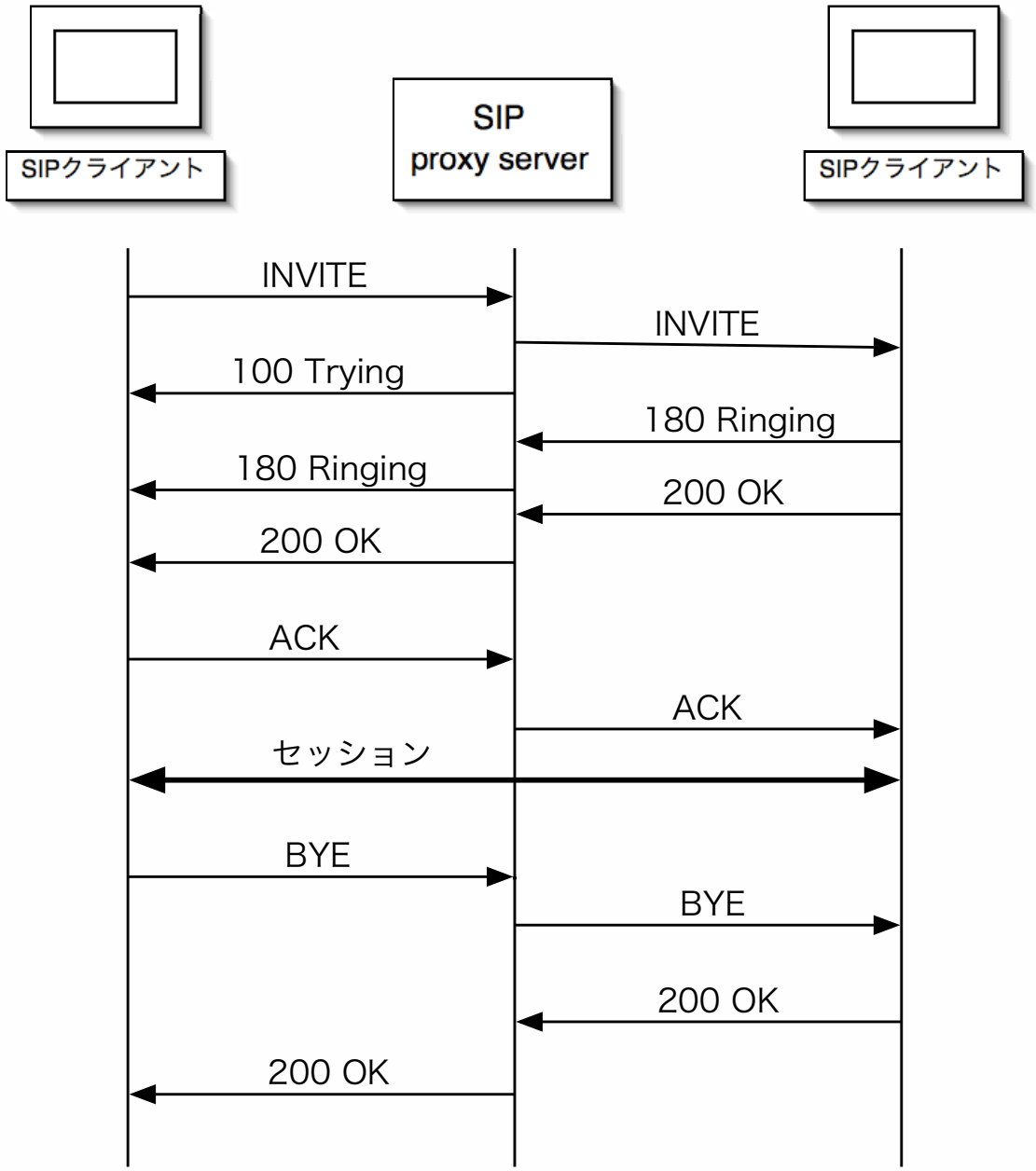
SIPクライアント
SIP
proxy server
SIPクライアント
INVITE
INVITE
100 Trying
180 Ringing
180 Ringing
200 OK
200 OK
ACK
ACK
セッション
BYE
BYE
200 OK
200 OK

図 2.5: SIP の動作

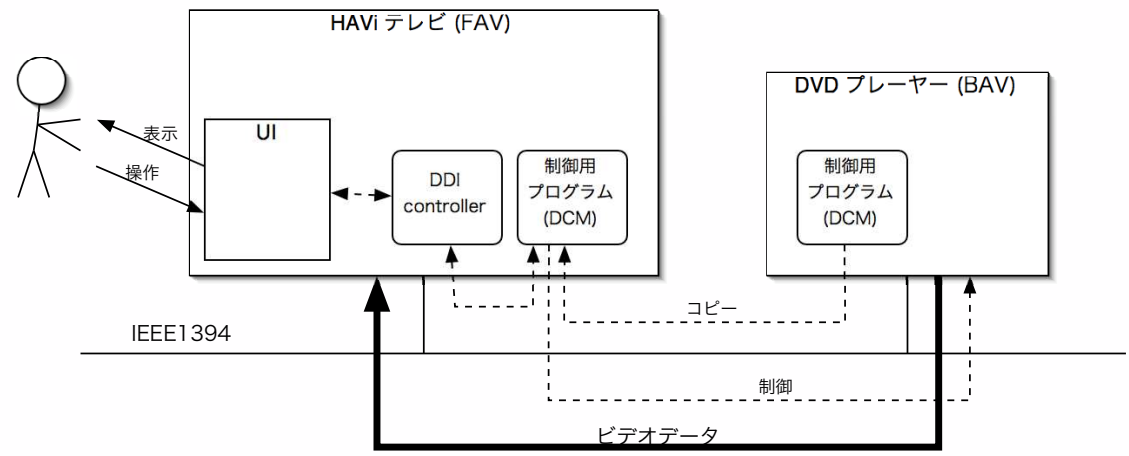

図 2.6: HAVi による DVD プレーヤーの操作

第3章　ビデオネットワークの相互接続

異なったビデオネットワークを接続するためには様々な問題が生じる。ビデオネットワークは想定された規模や利用目的などにより、それぞれのネットワークに特化されたものとなっている。ビデオネットワーク間を相互接続するためにはこれらの相異点を吸収しなければならない。本章ではビデオネットワーク間を接続する際に問題となる次の四つについて考察する。

1. アドレス体系
2. シグナリング
3. 伝送に利用されるプロトコル
4. ビデオフォーマット

3.1　アドレス体系

個々のビデオ機器を指し示すためにビデオネットワークではアドレスを用いることになる。しかしながら、異なるビデオネットワークはそれぞれの専用のアドレス体系を用いるのが普通でありビデオネットワークを接続した際に問題となる。

3.1.1　アドレスと識別子

アドレスは割り当てられた機器が存在する位置を示す。また、アドレスと同様に扱われることがあるものとして識別子が存在する。識別子は個々のエンティティに割り当てられたものであり、個々のエンティティを識別することが可能となる。あるエンティティに割り当てられた識別子が変更されることはない。一方アドレスも個々のエンティティに割り当てられるものであるが、対応付けは変更される可能性がある。そのため識別子をアドレスとして利用することはできない。

3.1.2　アドレスとデバイスの存在場所

近年、移動しながら通信を行う携帯電話のような移動体通信、あるいはパーソナルコンピュータなどを自分の好きな場所に持っていき、インターネットを利用するといった通信形態が一般的になってきている。このような通信を行う場合インターネットでは持ち運ん

でいる機器のアドレスを変更しなければならないといったことが生じる。一方、携帯電話はどこにいても同じ電話番号で個体を識別できる。このような携帯電話システムのような仕組みをインターネットに応用した技術として Mobile IP があげられる。 Mobile IP を用いることでどこに接続しても、個々の機器に振られた IP アドレスを変更することなく同じ IP アドレスを使い続けることができる。Mobile IP を利用した場合には IP アドレスを調べただけでは、機器が実際にどこにいるのかを識別することはできない。

3.1.3 アドレスと名前

よく利用されるアドレスは数字の羅列であることが多い。電話番号や IP アドレスはただの数字の羅列である。このような数字の羅列は言葉として意味を持たないため、記憶するなど人間が利用するのには適さない。そこでもっと覚えやすく意味のある名前を利用してアドレスを指し示す方法がある。このようなシステムでは名前とアドレスの組を持ったデータベースが必要となる。インターネットではこのような名前を IP アドレスに変換する世界規模の分散データベースである DNS が利用されている。また、個人が所有する携帯電話の住所録なども同様に名前などの人間に分かりやすい情報から電話番号に変換するということを行う。

3.2 シグナリング

ビデオ機器間でビデオデータを送受信する際、前もって帯域を確保したり接続を確立する必要がある。これらはシグナリングと呼ばれる機能により実現されるが、個々のビデオネットワークにより異なっており互換性のないものである可能性が高い。

3.3 伝送に利用されるプロトコル

ビデオデータを伝送するためには情報を伝えるためのプロトコルが必要である。インターフェースのハードウェアを直接利用してビデオデータを送るという方法や、ハードウェアより上位に位置するソフトウェアの層を利用するといった方法がある。これらのプロトコルが異なるビデオネットワーク間ではビデオデータのフォーマットが同一であっても送信者が送信したデータを受信側が受信することをできない。ここでは IEEE1394 と RTP におけるプロトコルスタックを説明する。

3.3.1 IEEE1394 プロトコルスタック

HAVi などのビデオネットワークは IEEE1394 などのハードウェアに依存してビデオデータを送受信する。このようなビデオネットワークでは異なる伝送媒体を利用したビデオ

ネットワークとは直接ビデオデータの送信を行う事はできない。IEEE1394 のプロトコルスタックは図 3.1 のようになっている。

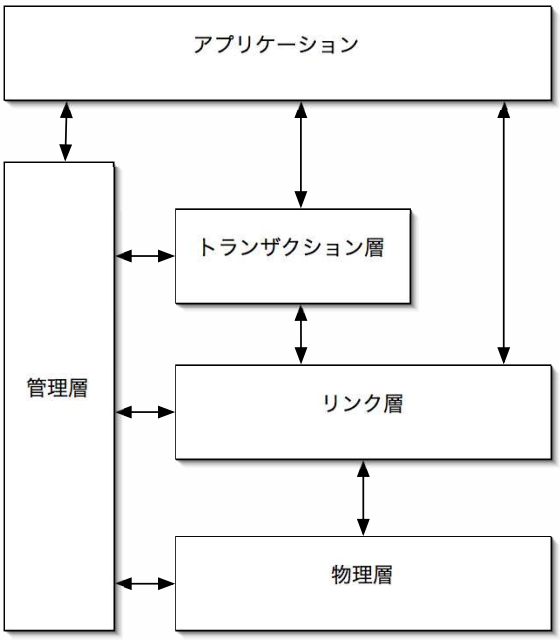

図 3.1: IEEE1394 のプロトコルスタック

リンク層はパケットを処理する層である。また、アイソクロナス通信のサポートをしておりハードウェアとして実装される。トランザクション層はアシンクロナス通信を行う際に利用される。IEEE1394 でサポートされているアイソクロナス通信は時間の制約などが厳しくソフトウェアで処理するのには適していない。従って、IEEE1394 と同等の帯域予約をできないネットワークでリンク層を実現するのは困難である。

3.3.2 RTP プロトコルスタック

インターネットストリーミングなどで利用される RTP を用いて IP ネットワークを利用して構築されたビデオネットワークは、ハードウェアに依存せずソフトウェアでビデオデータの転送が処理される。このようなビデオネットワークでは IP プロトコルが実装されていればどのようなハードウェアでも利用可能であるが、IP パケットの送受信が可能でなければならない。RTP のプロトコルスタックは図 3.2 のような構成になっている。

RTP の下位層は UDP と IP である。IP はさまざまなネットワークプロトコルの上に載ることができ、最善努力配送を行うパケット転送を実現する。これにより、ほとんどどのようなコンピュータネットワーク上であっても IP を実装できるが、ハードウェアがサポートする様々なサービスを RTP から利用することは基本的に不可能となる。

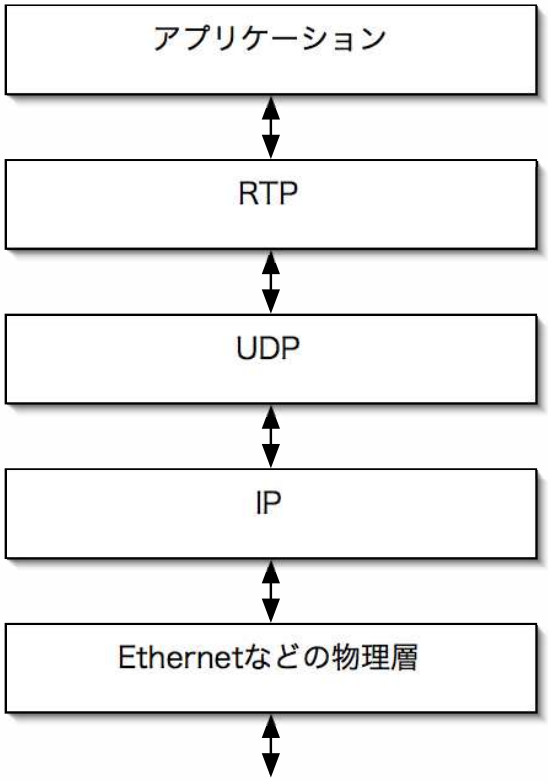

図 3.2: RTP のプロトコルスタック

3.4 ビデオフォーマット

フォーマットの事なっている相互に接続しているビデオネットワーク間でビデオデータを転送するためにはフォーマットを適した形に変換する必要がある。広帯域のビデオネットワークではビデオデータを転送するための比較的広い帯域を利用可能であるため、高画質のビデオデータを送信可能である。一方、狭帯域のネットワークでは低画質のビデオデータしか転送できない。異なるビデオネットワークでビデオフォーマットが異なっている場合には、送信者が送信したビデオデータを受信者が受信できるかもしれないが受信者が理解できる形式ではないためデータを処理できない。

また、ビデオフォーマットによっては縦と横の比率であるアスペクト比や映像自体の大きさが異なる可能性がある。このようなビデオフォーマット間では映像を伸長したり縮尺したりして別のフォーマットに変換しなければならない。

3.4.1 圧縮方式の相違

一般的にビデオデータは圧縮されて転送される。機器の処理能力の向上や技術の進歩に伴い、圧縮率が高いが画像の劣化を人間の目にはそれほど感じさせない圧縮方法が登場している。圧縮率が高い方式を用いると画質の劣化が顕著になるのが一般的であり、比較的圧縮率の高くないもののほうが画質はよいものとなる。またある程度以上の画質であれば画質の劣化は人間の目ではほとんど感じられないものとなるため、非圧縮なビデオデータではなく圧縮されたものを利用することで、それほど帯域を必要とせず劣化をほとんど感じることのないビデオデータの転送を行うことができる。

狭帯域であるビデオネットワークは高い圧縮率を実現した圧縮アルゴリズムが利用され、また広帯域なビデオネットワークでは画質がほとんど変化しないような圧縮アルゴリズムが利用されることが想定される。これらのビデオネットワークを接続するためにはある圧縮方式から異なった圧縮方式にビデオデータを変換しなければならない。

3.4.2 映像の表示サイズ

また、狭帯域なビデオネットワークでは映像の表示サイズを小さくすることで必要とする帯域を減らすことが可能である。そのため、圧縮方式の変更に加えて映像の表示サイズを拡大したり縮小するなどして別のビデオネットワークで利用される大きさに変更しなければならない。

3.4.3 アスペクト比

日本で標準的に利用されているテレビ放送の形式である NTSC は横と縦の比率であるアスペクト比が 4:3 である。しかしハイビジョンなどの映像の比率は 16:9 となり NTSC と比較して横長になっている。このようにアスペクト比が異なるビデオネットワーク間でビデオデータの転送を行う際にどのようにアスペクト比の相違を扱うかが問題となる。この問題を解決する方法として以下の三つの方法がある。

1. そのままの映像を新たなアスペクト比の中におさめる (図 3.3)
2. オリジナルの映像をそのまま拡大しはみだした部分は切り取る (図 3.4)
3. オリジナルの映像を縮小してすべての映像をおさめる (図 3.5)

ここで示した図はアスペクト比が 4:3 である映像をアスペクト比が 16:9 である映像に変換した場合である。

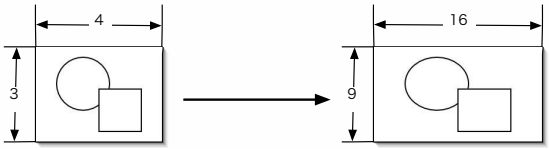

図 3.3: アスペクト比の変更

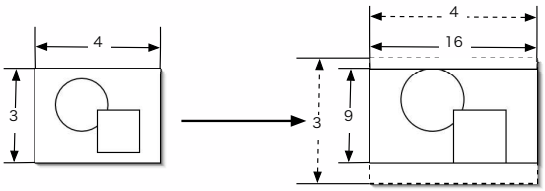

図 3.4: 拡大を行ったアスペクト比の変更

3.4.4 フレーム数

動画は実際に動作している画像を表示しているわけではなく、図 3.6 のようにある静止した一連の映像を順番に表示させることで動画として認識させる。これらの一枚一枚の静止画をフレームと言う。テレビなどに静止画が映し出されると、人間の目には残像が残り、その静止画が表示され続けていると錯覚する。その残像が目の中に残っている間にテレビは次の静止画を表示することで人間は動画を見ていると錯覚する。日本でテレビ放送として利用されている NTSC では毎秒 30 フレーム程度の静止画を表示することで動画として放送している。フレーム数が少なくなると人間の目には滑らかな動画でないと感じるようになる。

このように動画は連続したフレームとして送信されることになるがフォーマットによって毎秒のフレーム数が異なる場合が考えられる。毎秒 24 フレームのビデオデータを毎秒 30 フレームのビデオデータに変換するためには、24 フレームを 30 フレームに増やさなければならない。二つの連続したフレームを得られても、その間に位置するはずのフレームを得る事はできないため、あるビデオデータをフレームレートの異なるビデオデータに変換することで映像の品質は劣化してしまう。

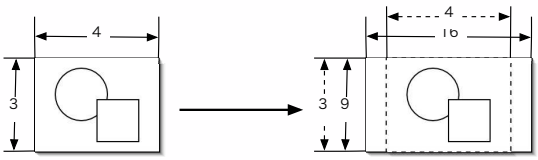

図 3.5: 縮小を行ったアスペクト比の変更

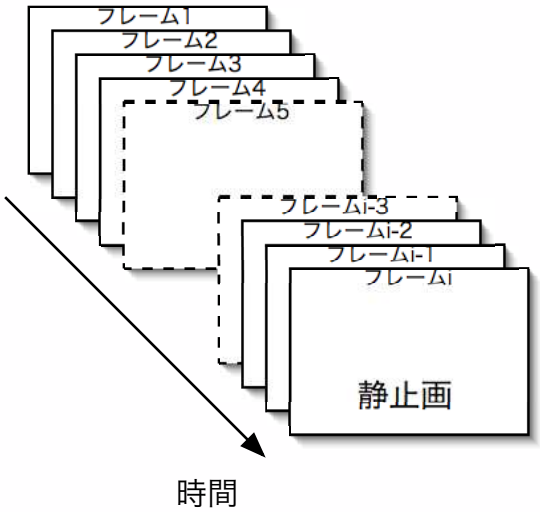

図 3.6: 動画のフレーム

映画などをテレビで出力する場合について考える。一般的に映画等で利用されているフレームレートは 24 フレーム毎秒で、日本で利用されている一般的なテレビでは 30 フレーム毎秒である。このことからテレビで映画の動画を出力するためにはフレームレートを 24 フレーム毎秒から 30 フレーム毎秒に変更しなければならない。テレビは 1 フレームを偶数番目の走査線と奇数番目の走査線の二つにわけて実際には一秒間に 60 回表示を書き換えるインターレース方式が用いられている。この映像をフィールドと呼び、実際には 24 フレームを 60 フィールドに変換しなければならない。従って、テレビのフィールドと映画のフレーム数の割合は 5 対 2 である。そこで、テレビの 5 フィールドには映画の 2 フレームが割り当てられることが分かる。そこで図 3.7 のように 24 フレームを 30 フレームに変更することができる。

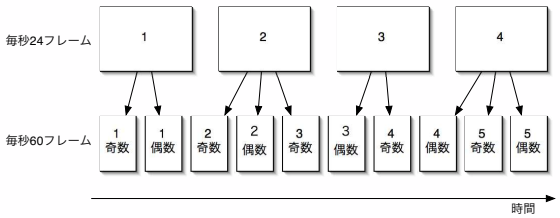

図 3.7: 2-3 プルダウン方式によるフレームレートの変換

3.5 まとめ

本章ではビデオネットワークを相互接続する際に問題となる点について明確にした。ビデオ機器間で行われるシグナリングの相違、ビデオデータの転送されるプロトコルの相違、またビデオデータのフォーマットが異なる際の問題点などの議論を行った。これらの問題を解消し相互接続を実現するためにはこれらの問題点を解決しなければならない。これらを解決し相互接続を可能とするシステムについて次章で説明する。

第4章　ビデオインターネットワーキングアーキテクチャ

ここでは本研究で提案するビデオインターネットワーキングアーキテクチャについて説明する。このアーキテクチャにより異なった技術を用いているため直接は接続できないビデオネットワーク間や、管理組織が同一でないビデオネットワーク間におけるビデオ機器の相互接続を可能にする。

4.1　概要

本システムはリソースマネージャを中心とした以下の四つのコンポーネントから成り図4.1のようになる。

- リソースマネージャ
- デバイスコントローラ
- ビデオ機器
- ビデオゲートウェイ

各ビデオネットワークはコネクションや機器情報などの資源管理を行うためのリソースマネージャを一つ持つ。またビデオネットワークにはビデオ機器の他にデバイスコントローラが一台以上接続される。デバイスコントローラはビデオ機器の接続を監視し、またビデオ機器に対してシグナリング要求などを行う。

図4.2のようにデバイスコントローラは実際のビデオネットワークへのビデオ機器のコントロールを行い、またビデオ機器をバーチャルノードとして抽象化を行う。これによって資源管理を行うリソースマネージャは抽象化されたビデオ機器であるバーチャルビデオノードを扱うことになり、各ビデオネットワークの詳細について考慮する必要がなくなる。ビデオ機器はテレビやVCRなどのビデオネットワークに接続されるビデオデータの送受信を行う通常の機器のことである。

ビデオゲートウェイは図4.3のように二組のデバイスコントローラとビデオ機器からなり、ビデオ機器の一つはビデオデータをビデオネットワークから受信しもう一つのビデオ機器に渡す。ビデオデータを渡されたビデオ機器は自分自身が接続されたビデオネット

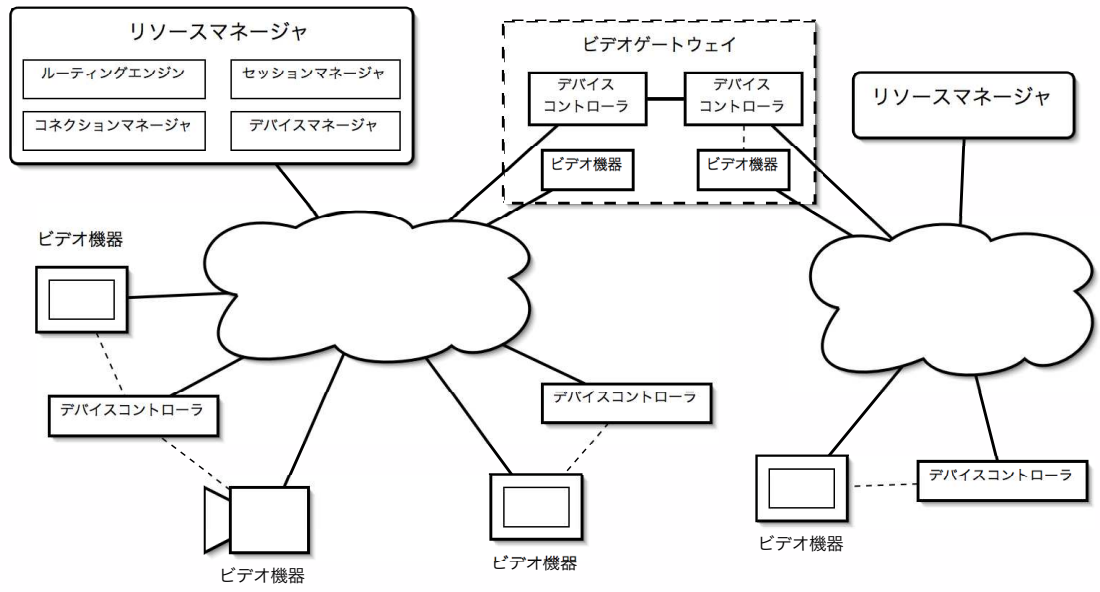

図 4.1: ビデオインターネットワーキングアーキテクチャ

ワークにビデオデータを転送する。ビデオゲートウェイにより異なったビデオネットワーク間でのビデオデータの送受信が可能となる。

4.2 アドレス体系

ビデオインターネットワーキングアーキテクチャの内部ではアドレスは次の三つに分けられる。

1. ネットワークアドレス
2. ノードアドレス
3. 機器固有アドレス

これらのアドレスは図 4.4 のように利用される。

ネットワークアドレスは各ビデオネットワークを識別するためのアドレスである。これによりビデオインターネットワーキングアーキテクチャを構成する個々のビデオネットワークを識別する。ビデオインターネットワーキングアーキテクチャの中で同一のネットワークアドレスを持つビデオネットワークが複数あってはならない。

一方、ノードアドレスはビデオネットワーク内のビデオ機器を識別するために利用されるアドレスとなる。一つのビデオネットワーク内で同一のノードアドレスを持つビデオ機器が複数存在してはならない。しかし、異なるビデオネットワークであれば同一のノードアドレスを持つビデオ機器が存在しても問題はない。

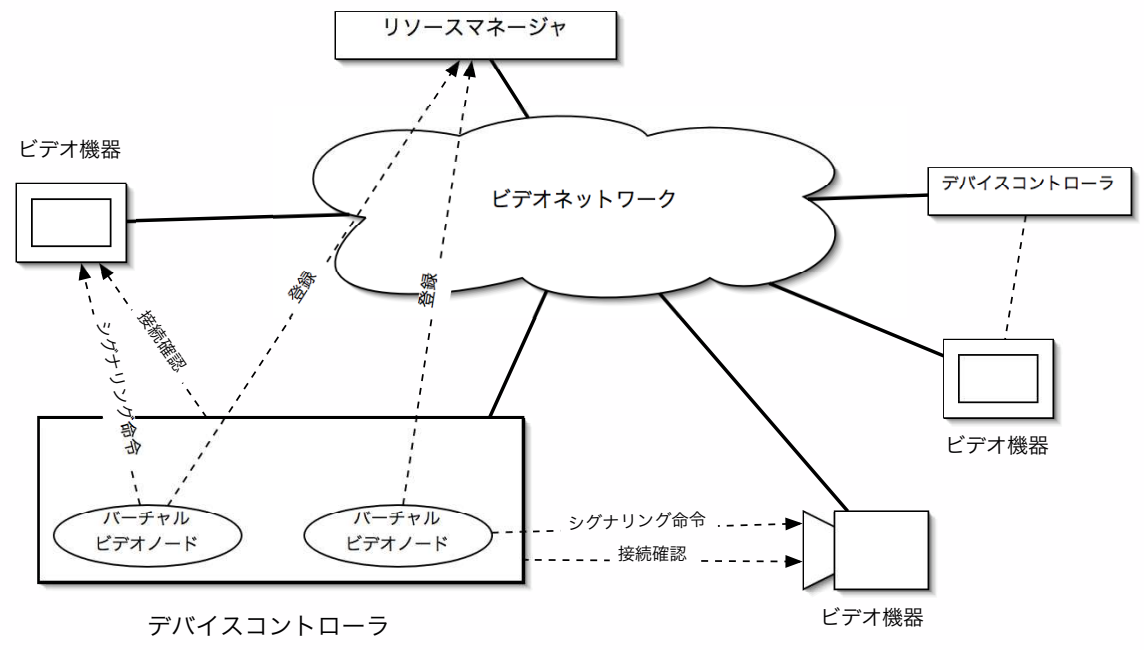

図 4.2: ビデオ機器の抽象化

機器固有アドレスはベースとなるビデオネットワーク内でのビデオ機器のアドレスであり、デバイスコントローラの節で説明するバーチャルビデオノードによって隠蔽される。また、機器固有アドレスはビデオ機器間でシグナリングを行う際に必要となる。

4.2.1 アドレス解決

ノードアドレスは個々のビデオ機器を識別するために利用されるが、実際のシグナリングを行う際には機器固有アドレスを利用しなければならない。そこで、接続先であるビデオ機器に割り当てられたノードアドレスから機器固有アドレスに変換する機構が必要である。

このようなアドレス変換の機構のうち、イーサネットで利用されている ARP と呼ばれるプロトコルについて簡単に説明する。扱いが容易で安価で高速であるためローカルエリアネットワークではイーサネットが利用されていることが多い。イーサネットはデータリンク層と物理層の技術である。各イーサネットインターフェースは MAC アドレスと呼ばれる 48bit のアドレスを持っており、実際にフレームを転送する際にはこのアドレスが送信者と受信者を識別するために利用される。しかし、一般的なアプリケーションは通信を行う際には MAC アドレスではなくネットワーク層のアドレスである IP アドレスを用いられることが多い。そこで、実際にイーサネットフレームを宛先に転送するためには IP アドレスを MAC アドレスに変換しなければならない。これを行うのが ARP である。

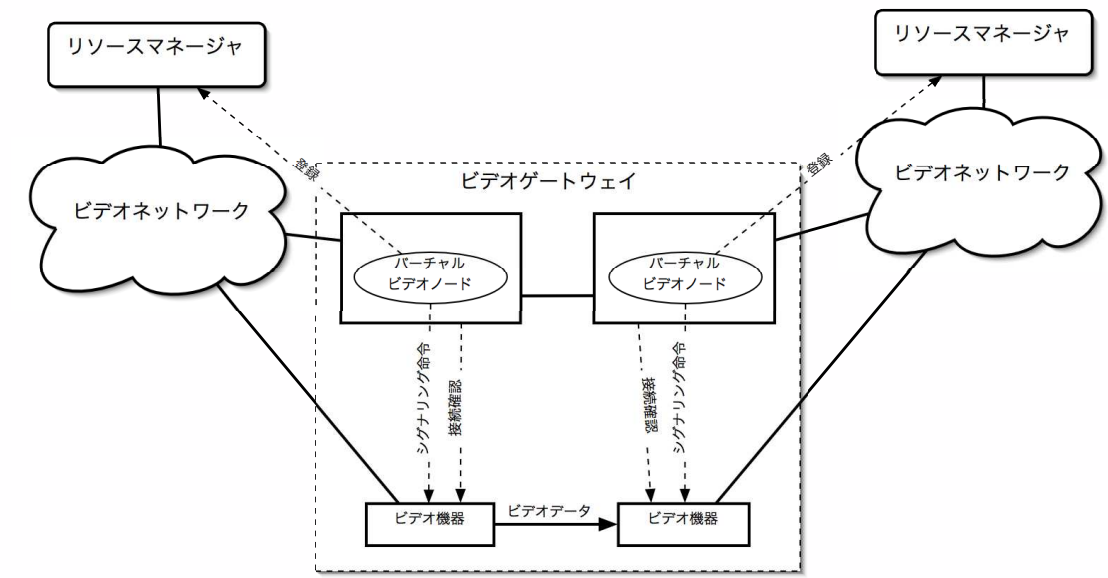

図 4.3: ビデオゲートウェイ

各機器は変換を行いたい IP アドレスを指定したフレームをブロードキャストすることで MAC アドレスへの変換を開始する。指定された IP アドレスを持っている機器はその IP アドレスに対応したインターフェースの MAC アドレスを指定して返答する。こうして IP アドレスから MAC アドレスに変換することが可能であり、IP パケットをイーサネットを介して転送することが可能となる。

ビデオネットワークでも同様に図 4.5 のようにブロードキャストやマルチキャストを用いてノードアドレスを機器固有アドレスに変換する方法が考えられる。他の方法として図 4.6 のようなノードアドレスと機器固有アドレスのマッピングを管理するシステムを利用する方法がある。ノードアドレスから機器固有アドレスを求める時にはこのシステムにアドレスの変換を要求する。また逆に機器固有アドレスからノードアドレスに変換することも可能である。一方、図 4.7 のようにノードアドレスが機器固有アドレスを内包可能であれば、ノードアドレスの一部を取り出すだけで機器固有アドレスとなるようにノードアドレスを割り当てることも可能である。

4.3 コネクションとセッション

本システムではビデオ機器間の接続の種類が二つある。これらはコネクションとセッションと呼ばれる。コネクションはローカルなビデオネットワーク内だけのビデオ機器間の接続でる。一方セッションは複数のビデオネットワークにまたがって張られたビデオ機器間の接続であり、図 4.8 のように複数のコネクションからなる。

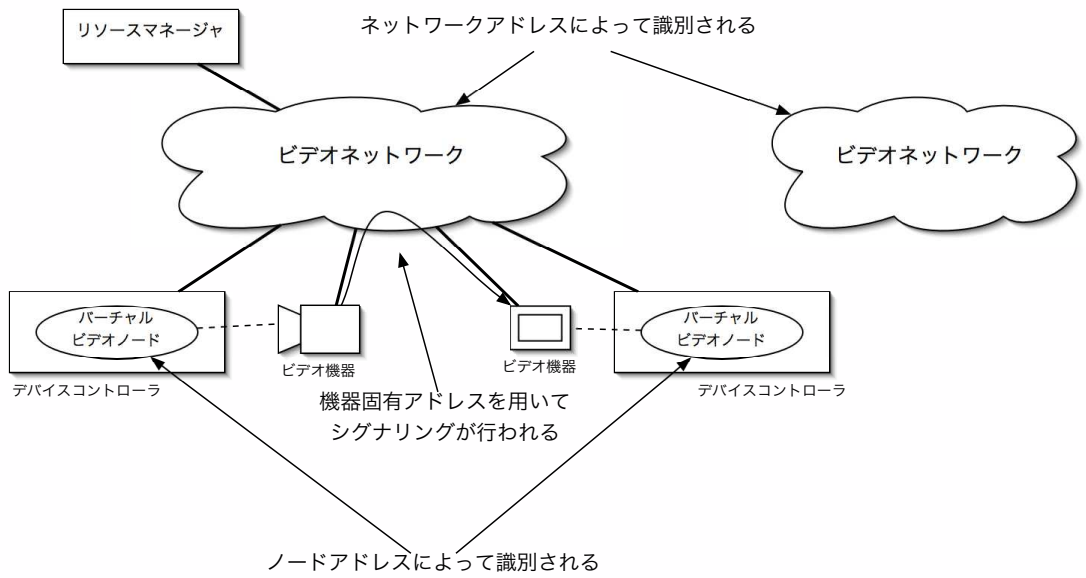

図 4.4: ビデオインターネットワーキングアーキテクチャのアドレス

4.4 デバイスコントローラ

デバイスコントローラの役割は以下の四つである。

1. ビデオ機器の接続の監視
2. バーチャルビデオノードの生成
3. バーチャルビデオノードの登録と抹消
4. バーチャルビデオノードへのシグナリング要求

図 4.9 のようにデバイスコントローラはビデオ機器がビデオネットワークに接続されたか監視をしており、ビデオ機器が接続された場合にはバーチャルビデオノードの生成と登録を行う。これによりデバイスコントローラやリソースマネージャが扱うものはバーチャルビデオノードとなり、ビデオ機器に関する細かい情報について考慮する必要がなくなる。また、コネクションの確立は図 4.10 のように行われる。この際、シグナリングをビデオ機器に行わせるために必要な操作の詳細はバーチャルビデオノードの内部に隠蔽することで、デバイスコントローラのメインプログラムがシグナリングの詳細を知る必要がなくなる。

4.4.1 バーチャルビデオノード

実際のビデオ機器を抽象化したバーチャルビデオノードについて説明する。バーチャルビデオノードはビデオネットワークにビデオ機器が接続された際にデバイスコントローラ

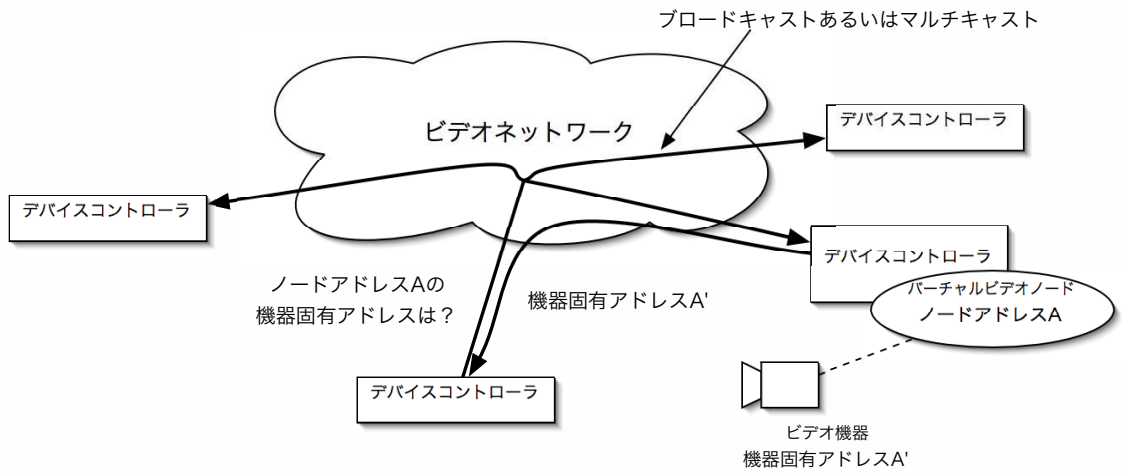

図 4.5: ブロードキャストあるいはマルチキャストを利用したアドレス変換

内部で生成される抽象機器である。

ビデオ機器の抽象化

全てのビデオ機器は一つのノードアドレスを持ち、送信あるいは受信のどちらかのビデオデータの向きを持つバーチャルビデオノードとして抽象化される。扱うことが可能であるビデオフォーマットや映像の品質、遅延などの情報についてデバイスコントローラは現在のところ取り扱わない。そのためこれらの情報が必要である場合には、システム内にコンポーネントを追加するか本システムの外部に必要な情報を管理する仕組みが必要となる。

生成と解放

図 4.9 のようにバーチャルビデオノードはビデオ機器がビデオネットワークに接続されたときに生成される。このとき対応するビデオアドレスが決定される。このビデオアドレスは固定的なものであっても自動的に生成されたものであってもよいが、一つのビデオネットワーク内で同一のビデオアドレスを持つバーチャルビデオノードが複数存在することは許されない。既に別のバーチャルビデオノードが利用しているビデオアドレスを持つバーチャルビデオノードを新たにリソースマネージャに登録しようとすると失敗する。

バーチャルビデオノードの生成には、デバイスコントローラがビデオ機器の接続を監視し自動的にバーチャルビデオノードを生成する方法とビデオ機器を接続したのち手動でバーチャルビデオノードを生成させる方法がある。ビデオ機器が接続したか常に監視することでバーチャルビデオノードを生成する場合には、デバイスコントローラはビデオ機器の接続状態を調べ続けなければならない。一方、ビデオ機器を接続したのち手動でデバイ

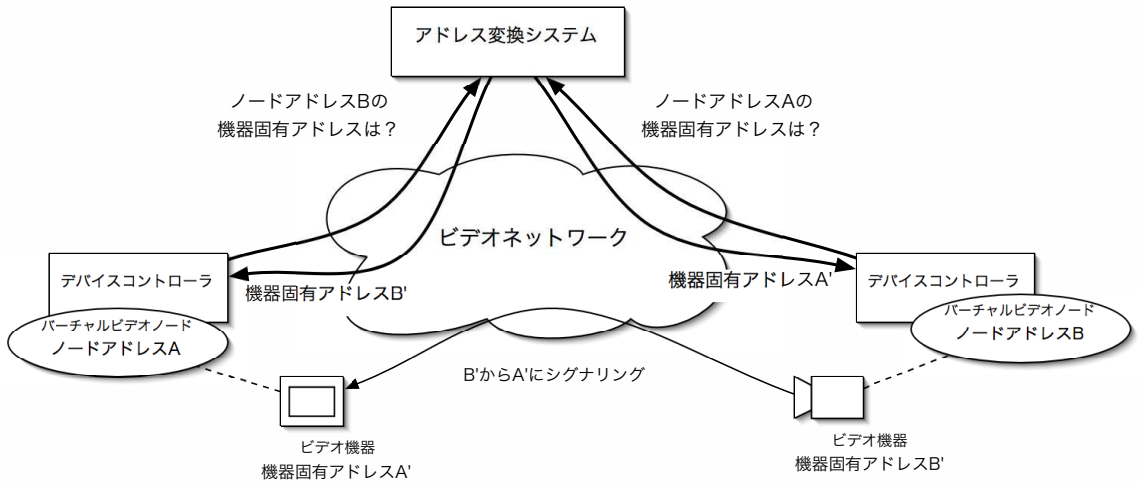

図 4.6: アドレス変換システムを利用したアドレス変換

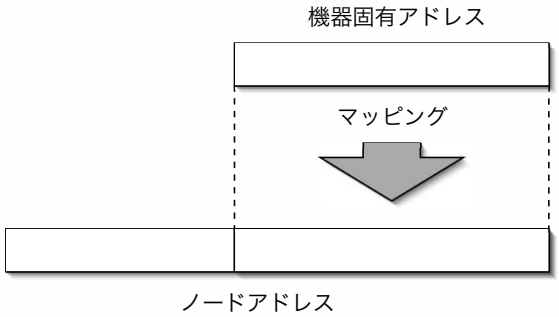

図 4.7: 機器固有アドレスを含んだノードアドレス

スコントローラに新たにバーチャルビデオノードを生成するという方法を用いた場合はデバイスコントローラがビデオ機器の監視をする必要はないが、手作業で生成を要求する必要があり、手間が増えてしまう。

また、ビデオ機器がビデオネットワークから取り外された場合には対応するバーチャルビデオノードは解放されなければならない。同時にリソースマネージャに登録されているバーチャルビデオノードに関する情報も抹消しなければならない。

登録と抹消

デバイスコントローラはリソースマネージャと通信を行い、デバイスマネージャが管理を行うデータベースにバーチャルビデオノードを登録する。登録するタイミングは次の二つが考えられる。

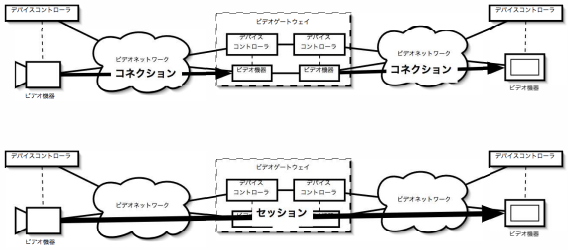

図 4.8: コネクションとセッション

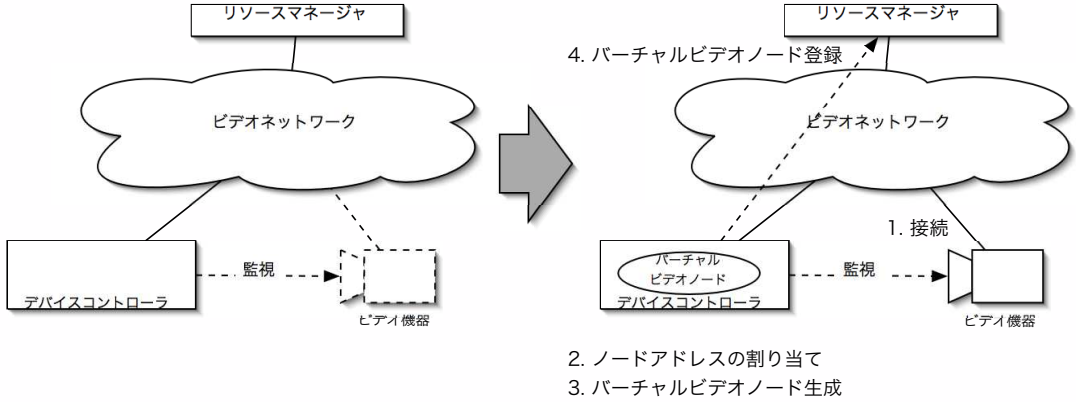

図 4.9: ビデオ機器の監視

1. バーチャルビデオノード生成後
2. ユーザからの要求後

ビデオインターネットワーキングアーキテクチャに影響されることなくビデオ機器が動作を行うためには、その機器を抽象化して生成されたバーチャルビデオノードをリソースマネージャ内のデバイスマネージャに登録してはならない。登録をしてしまうと予期しないコネクションの要求をリソースマネージャから受けてしまう可能性がある。このようにリソースマネージャからコントロールされてはならないビデオ機器は普段は登録を行わず、ビデオインターネットワーキングの端末になる必要がある時のみリソースマネージャに登録すればよい。

ノードアドレスの生成

バーチャルビデオノードの生成の際にノードアドレスが決められる。ここではどのようにノードアドレスを決定するかについて議論する。

1. ノードアドレスにビデオネットワークにおける機器固有アドレスを埋め込む

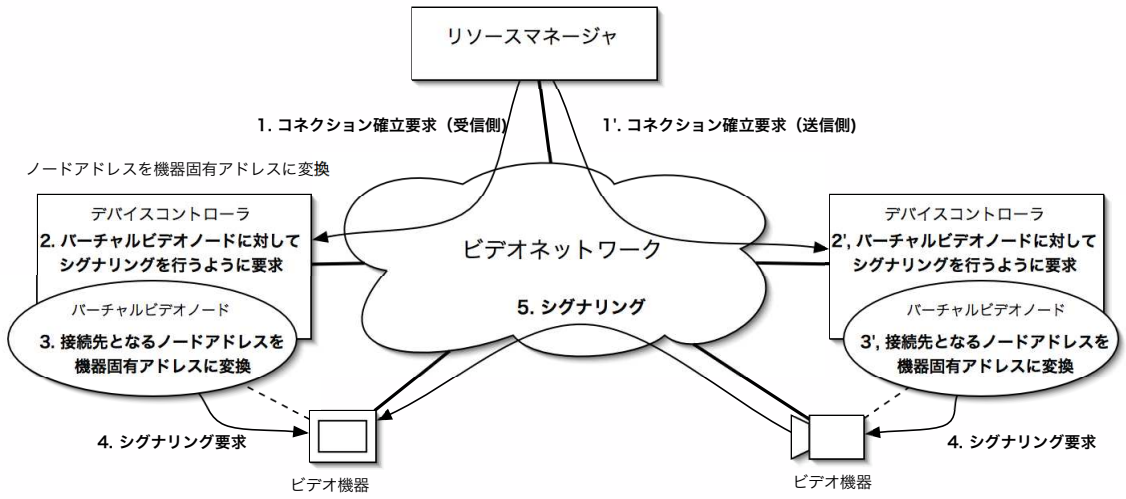

図 4.10: コネクション確立

2. ランダムに生成する
3. ノードアドレス管理を行うシステムを利用する

通常、ビデオネットワークは機器固有アドレスを用いてビデオ機器を識別する。そこで、機器固有アドレスを内包したノードアドレスを利用する方法がある。先に図 4.7 で説明した通り、この方法ではノードアドレスを得られると、簡単な計算で機器固有アドレスを得ることが可能である。ただし、ビデオアドレスが機器固有アドレスよりも短い場合には機器固有アドレスの一部が欠落したものがノードアドレスとなるため一つのビデオネットワークに同じノードアドレスを持つバーチャルビデオノードが作られてしまう可能性がある。

次にランダムにノードアドレスを生成する方法がある。この方法では同じノードアドレスを持ったバーチャルビデオノードが発生する可能性があるが、その場合にはデバイスマネージャに登録できない。そこで登録に失敗した場合には別のノードアドレスで登録を再度行うということを成功するまで繰り返す。この方法ではビデオ機器を繋げるたびにノードアドレスが変更されることになり、ノードアドレスを見ただけでは、どこにつながったどのビデオ機器であるか、という情報がまったく得られなくなってしまう。そのため、このようなアドレスとビデオ機器に関する情報を管理するシステムが外部に必要となる。

最後にノードアドレス管理を行うシステムを外部に持つ方法について説明する。ビデオ機器が接続されたとき、デバイスコントローラはノードアドレスを何にすればよいかという質問をこの外部システムに送信する。この際、ビデオ機器に関する情報も一緒に送信し、これらの情報を用いて外部のシステムはノードアドレスを決定する。これにより同一ビデオネットワーク内のノードアドレスの利用状況は外部に配置された一つのシステムにより管理されることとなり同一のビデオアドレスを持つバーチャルビデオノードが存在し

ないことが保証される。

機器固有アドレスをノードアドレスとして用いる方法は単純であるが、常に適用可能な方式ではない。ランダムに生成する方法ではノードアドレスがどのように生成されるかはデバイスコントローラに依存する。外部システムにノードアドレスを生成してもらう方法ではどのようなノードアドレスを割り当てるかは外部システムに依存することとなり管理を行うといった観点からすると優れたものとなり得る。本システムではノードアドレスの生成は各ビデオネットワークで自由に行う。ただし、重複したノードアドレスが生成されないことと、ノードアドレスから機器固有アドレスに変換できることが必要である。

アドレス変換とコネクション要求

コネクションを確立するためにビデオ機器はシグナリングを行う必要がある。デバイスコントローラはビデオ機器に対してシグナリングを行うように命令を出さなければならない。この際、デバイスコントローラが分かっているのは接続先のバーチャルビデオノードのノードアドレスのみであり、シグナリングに必要な機器固有アドレスについての知識を持っていない。そこで、接続先のビデオ機器に割り当てられた機器固有アドレスについての情報をシグナリング要求をする前に得る必要がある。

ノードアドレスの中に機器固有アドレスを含むようにアドレスが付けられていることが保証されているのであれば、ノードアドレスから必要な機器固有アドレスを求められる。しかしながら、ランダムに生成されたアドレスであったり、他の管理機構の助けを借りて付けられたアドレスである場合には、図 4.5 や図 4.6 のように、アドレスから機器固有アドレスに変換するためのシステムが存在しなければならない。

このようにして得られた機器固有アドレスは、ビデオ機器にシグナリング要求を出すときに利用される。

4.4.2 デバイスコントローラの動作

デバイスコントローラの状態変化は図 4.11 のようになる。デバイスコントローラは最初に自分自身をリソースマネージャに登録する。そののちビデオ機器の監視とリソースマネージャからの命令の受信を行う。新たなビデオ機器が接続された場合にはノードアドレスの生成を行いバーチャルビデオノードを構築しリソースマネージャに登録する。また、ビデオ機器が取り外された場合には登録されているバーチャルビデオノードの抹消を行う。

4.5 ビデオゲートウェイ

ここではデバイスコントローラの特殊な形態であるビデオゲートウェイについて説明する。ビデオゲートウェイはビデオネットワーク間でシグナリングを終端し必要であればビ

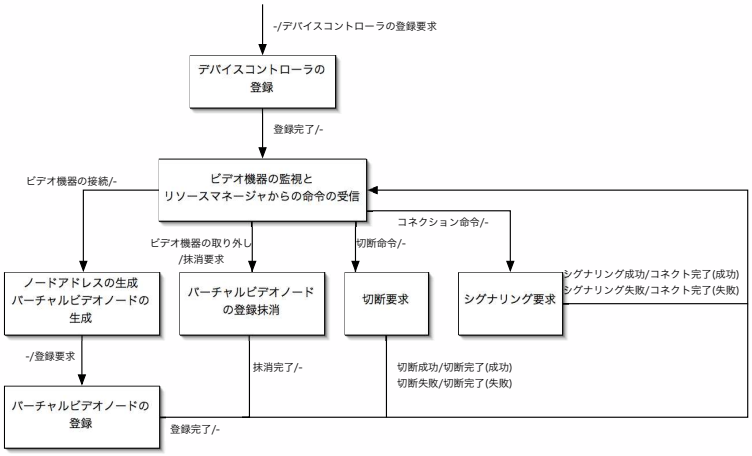

図 4.11: デバイスコントローラの状態

デオデータのフォーマットを変更する。ビデオネットワークの相互接続には必要不可欠なコンポーネントである。

4.5.1 構成

ビデオゲートウェイは図 4.12 のように二つの異なるビデオネットワークに存在するデバイスコントローラからなる。また双方のデバイスコントローラには相互に異なったビデオネットワーク間を接続するためのバーチャルビデオノードが存在し、それぞれ実際のビデオ機器と対応している。これらのビデオ機器は異なったビデオネットワーク間でビデオデータを転送できるように相互に接続されていなければならない。

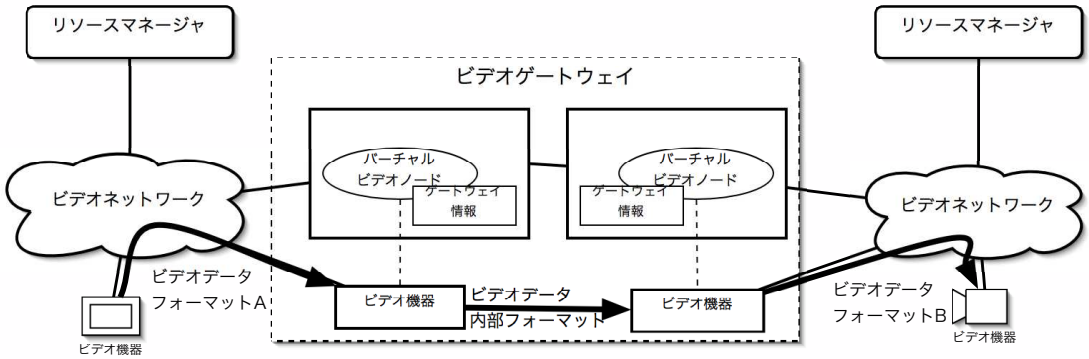

図 4.12: ビデオゲートウェイの構成

ビデオゲートウェイは各ビデオネットワークのシグナリングの終端となることで異なるシグナリングを用いたビデオネットワーク間の接続を可能とする。また各ビデオネット

ワークに属するビデオ機器間でビデオフォーマットの変換を行い、異なる転送媒体へのビデオデータの転送を可能とする。

4.5.2 リソースマネージャへの登録

ビデオゲートウェイの構成は基本的にデバイスコントローラの組であるのでリソースマネージャへの登録方法は通常のデバイスコントローラと変わらない。また、異なるビデオネットワークにビデオデータを転送するために利用されるビデオ機器についても、通常どおりバーチャルビデオノードとして登録される。しかし、この時にこれらのバーチャルノードがビデオゲートウェイの一部であるということも登録しなければならない。この際、ビデオゲートウェイがどのビデオネットワークのどのバーチャルビデオノードの対になっているかという情報が必要となるため、図 4.13 で示すように相手のビデオネットワークに割り当てられたネットワークアドレスとビデオゲートウェイにおいて対となるバーチャルビデオノードのノードアドレスについての情報もリソースマネージャに送信される。

4.6 リソースマネージャ

リソースマネージャは各ビデオネットワークに必ず一つだけ存在する。各リソースマネージャはネットワークアドレスを保持している。内部は以下のような四つのコンポーネントからなる。

1. デバイスマネージャ
2. コネクションマネージャ
3. セッションマネージャ
4. ルーティングエンジン

4.6.1 デバイスマネージャ

デバイスの登録や抹消を管理するサブコンポーネントである。

管理されるデータ

デバイスマネージャで管理されるデータについて説明する。デバイスマネージャによって管理されるデータは次の三種類である。

1. デバイスコントローラ
2. バーチャルビデオノード

3. ゲートウェイ

実際のビデオ機器の接続や切断の監視やシグナリング命令を送信するデバイスコントローラは、デバイスマネージャに自分自身の登録を行うことでリソースマネージャが管理するビデオネットワークの一部となる。デバイスマネージャではデバイスコントローラに関して以下の二つのデータが管理される。

1. デバイスコントローラ ID
2. デバイスコントローラアドレス

ここで登録されるデバイスコントローラアドレスはリソースマネージャがデバイスコントローラと通信を行う際に必要となるものであり、IP を利用して通信を行うのであればデバイスコントローラの IP アドレスやポート番号などになる。

バーチャルビデオノードはデバイスコントローラにより登録が行われる。ここで必要となる情報は以下の三つである。

1. デバイスコントローラ ID
2. ノードアドレス
3. ビデオデータの向き

バーチャルビデオノードのノードアドレスとビデオデータの向き、そしてそのバーチャルビデオノードの管理を行っているデバイスコントローラ ID を管理することになる。

バーチャルビデオノードでビデオゲートウェイの一部になるものはゲートウェイとしての情報も登録される。ここで登録される情報は以下の三つである。

1. ローカルノードアドレス
2. リモートネットワークアドレス
3. リモートノードアドレス

ローカルのネットワークに存在するバーチャルビデオノードのノードアドレスであるローカルノードアドレスと、接続先のビデオネットワークのネットワークアドレスであるリモートネットワークアドレス及びバーチャルビデオノードのノードアドレスであるリモートノードアドレスである。図 4.13 にアドレスの対応を示す。

4.6.2 コネクションマネージャ

単一ビデオネットワーク内でのコネクションの確立を行うためのサブコンポーネントである。コネクションを確立するためには、送信元ノードアドレス、受信先ノードアドレスを指定する必要がある。コネクションマネージャは次のようなコネクション情報を保存する。

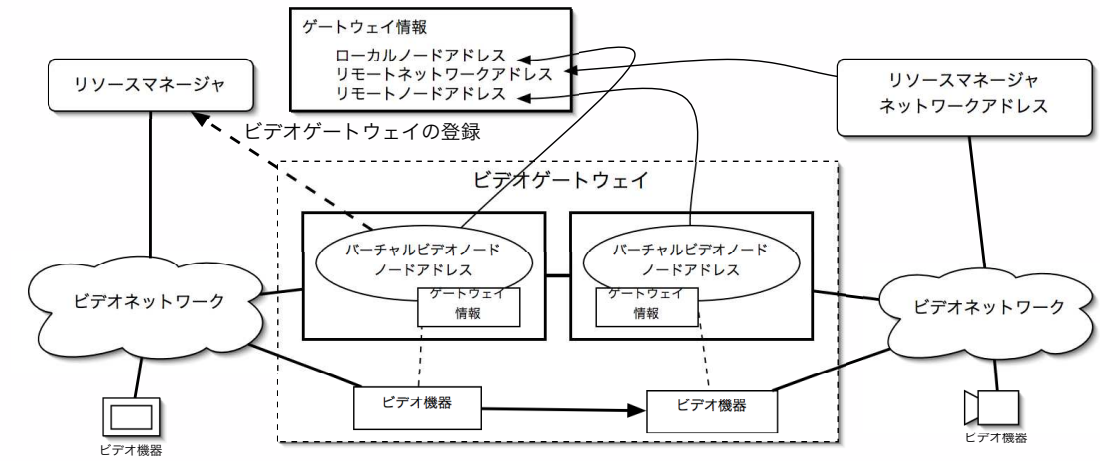

図 4.13: ビデオゲートウェイに関係するアドレス

1. コネクション ID
2. 送信元ノードアドレス
3. 受信先ノードアドレス

コネクション ID はコネクション一つ一つを識別するために付けられるユニークな番号である。送信元ビデオアドレスはビデオデータの送信元であるビデオ機器に対応したバーチャルビデオノードのビデオアドレスであり、同様に受信先ビデオアドレスは受信先のビデオ機器に対応したバーチャルビデオノードのビデオアドレスとなる。

4.6.3 セッションマネージャ

ビデオインターネットワーキング全体でビデオデータの接続を行うために必要なサブコンポーネントである。送信元と受信先のビデオ機器の存在するビデオネットワーク以外のビデオデータがただ経由するだけのビデオネットワークにおいてセッションの確立を行うためにローカルのビデオネットワークに存在する適切なビデオゲートウェイ間のコネクションを確立する必要がある。また送信元と受信先のビデオ機器が存在するビデオネットワークでは、送信元バーチャルビデオノードと送信元のビデオネットワークに接続されたビデオゲートウェイ、受信先バーチャルビデオノードと受信先のビデオネットワークに接続されたビデオゲートウェイのコネクションを確立する必要がある。セッションマネージャでは以下のセッション情報を保持する。

1. セッション ID
2. コネクション ID
3. 送信元ネットワークアドレス

4. 送信元ノードアドレス
5. 受信先ネットワークアドレス
6. 受信先ノードアドレス

セッション ID はビデオインターネットワーキングアーキテクチャ内でセッションを一意に識別する番号である。また、コネクション ID はローカルのビデオネットワークのコネクションのうち、このセッションの一部となっているものの番号である。この番号をコネクション ID として持つデータをコネクション情報から得ることで、どのバーチャルビデオノードからどのバーチャルビデオノードに張られたコネクションであるか調べることができる。

4.6.4 ルーティングエンジン

ビデオネットワーク間の接続状況を管理し、ビデオデータの転送をどのビデオネットワークを介して行うか決定するためのサブコンポーネントである。ルーティングエンジンの経路選択に従い複数のビデオネットワークを介したセッションが張られる。ルーティングエンジンでは近隣のリソースマネージャの情報として以下の四つを保持する。ネットワークアドレスによりビデオネットワークを識別する。さらにリソースマネージャと通信を行うために送信先となる IP アドレスとポート番号についての情報を持っている。

1. ネットワークアドレス
2. IP アドレス
3. ポート番号
4. 生存しているか

また、ルーティングエンジンは他のビデオネットワークへの最短経路を見付けるために必要な情報を他のリソースマネージャと交換する。ここで得られる情報は経路を選択するためにセッションマネージャが利用する。最短経路を発見するプロトコルとして、ここでは距離ベクトル型とリンクステート型についてここでは説明する。

距離ベクトル型

目的とするネットワークまでの距離を保持することで最短経路を見つけ出すプロトコルである。代表的なものとしては IP ネットワークで利用されるルーティングプロトコルである RIP[5] や BGP[6] があげられる。

RIP は比較的単純なプロトコルで、比較的複雑ではないローカルエリアネットワークで動的経路選択を行う際に利用される。各ルータは各ネットワークまでの距離を保持し、その情報を近隣のルータにブロードキャストすることで自分が持っている経路を伝達する。各ルータは他のルータから得られる距離の情報と自分が持っている情報を比較し、ある

ネットワークに対する経路についてより距離の近いルータが見付かった場合には新しい情報に更新する。

BGP は組織間の経路を制御する EGP と呼ばれるプロトコルの一種であり、ネットワーク管理者は経路選択に関する様々なポリシーを制御できる。BGP では RIP と違い、各ネットワークまでの距離だけではなく、どのような経路で目的とするネットワークに到達するかという情報も伝達する。これにより、経路がループをしていないかというチェックが可能となる。

リンクステート型

ネットワークのトポロジ情報を全てのノードが管理する方法である。代表的なプロトコルとしては IP ネットワークで利用される OSPF[7] や OSI 標準の経路選択プロトコルである IS-IS[8] がある。

OSPF は各ルータが自分自身のネットワークへの接続状況を把握しており、その情報を他のルータに伝達する。このようにしてすべてのルータ間で接続状況の交換が行われる。すべてのルータの接続状況を得ることができるため、ネットワークの完全なトポロジを各ルータは共有することになる。各ルータは隣接ルータとの接続に重みをつけており、この重みを元に各ルータが同一のアルゴリズムを利用することで経路を選択することになる。

4.7 メッセージの概要

ここではデバイスコントローラとリソースマネージャ、またリソースマネージャ間で送受信されるメッセージについて説明する。

4.7.1 デバイスコントローラとリソースマネージャ間のメッセージ

デバイスコントローラはリソースマネージャにデバイスコントローラ、バーチャルビデオノード、そしてビデオゲートウェイに関する情報を登録する。そこで、デバイスコントローラからリソースマネージャへ送られるメッセージは以下の六つである。また、いずれのメッセージに対してもリソースマネージャは登録、あるいは登録抹消が成功したかを伝えるためのメッセージを返信する。これらのメッセージはいずれもリソースマネージャのサブコンポーネントであるデバイスマネージャが受信する。

1. デバイスコントローラの登録
2. デバイスコントローラの登録抹消
3. バーチャルビデオノードの登録
4. バーチャルビデオノードの登録抹消
5. ビデオゲートウェイの登録

6. ビデオゲートウェイの登録抹消

リソースマネージャはデバイスコントローラにシグナリングを行うようにコネクション命令を送信する。そこで、リソースマネージャからデバイスコントローラへ送信されるメッセージは以下の二つである。また、デバイスコントローラはコネクションの確立、切断が成功したか失敗したかを伝えるメッセージを返信する。これらのメッセージはいずれもリソースマネージャのサブコンポーネントであるコネクションマネージャが送信する。

1. コネクションの確立
2. コネクションの切断

リソースマネージャは近隣のリソースマネージャに生存を知らせるメッセージを送信する。もし他のリソースマネージャの生存を知らせるメッセージを受信できなければ、そのリソースマネージャが管理をしているビデオネットワークは動作していないものとみなす。また、ビデオネットワーク間の接続状況を得るために経路情報の交換をリソースマネージャは常に行う必要がある。リソースマネージャ間で転送されるメッセージの種類は以下の四つである。このうちセッションの確立と切断はリソースマネージャのサブコンポーネントのセッションマネージャが送受信を行う。また、生存確認と経路情報の交換はルーティングエンジンが送受信を行う。

1. 生存確認
2. 経路情報の交換
3. セッションの確立
4. セッションの切断

第5章　JAIST VideoLANとDVTSを対象とした実装

本章では JAIST VideoLAN[9][10][11] と DVTS[13] を対象としたビデオインターネットワーキングアーキテクチャの実装の概要について説明する。ここで説明する実装は JAIST VideoLAN や DVTS でのみ有効なものではなく他のビデオネットワークへの適用も可能であると考えられる。本実装の API とメッセージの詳細については付録を参照していただきたい。また、本実装ではプログラミング言語として C++を利用した。

5.1　対象となるビデオネットワーク

5.1.1　JAIST VideoLAN

JAIST VIdeoLAN の概要を図 5.1 に示す。JAIST VideoLAN は IEEE1394 機器間で送受信される高品質ビデオデータである DV を ATM を介して送受信するシステムである。IEEE1394 と ATM という QoS 保証を行うネットワークを利用することで品質を損なうことなくビデオデータの転送が可能である。JAIST VideoLAN は IEEE1394 と ATM の間のブリッジを行うターミナルシステムと接続されている DV 機器やコネクション情報の管理を行う資源管理エージェント [12] からなる。

JAIST VideoLAN における実装

JAIST VideoLAN にビデオインターネットワーキングアーキテクチャを適用する方法は以下の二つの方法が考えられる。

- JAIST VideoLAN の資源管理エージェントをデバイスコントローラが管理を行う
- JAIST VideoLAN のシステム自体をビデオインターネットワーキングに変更

前者の方式は図 5.2 のようになる。リソースマネージャ内のビデオ機器に関する情報をデバイスコントローラが監視を行い、変更があった場合にデバイスマネージャに通知を行う。またコネクションの要求はデバイスコントローラを介することで資源管理エージェントに送信される。後者の方式は図 5.3 のようになる。資源管理エージェントをリソースマ

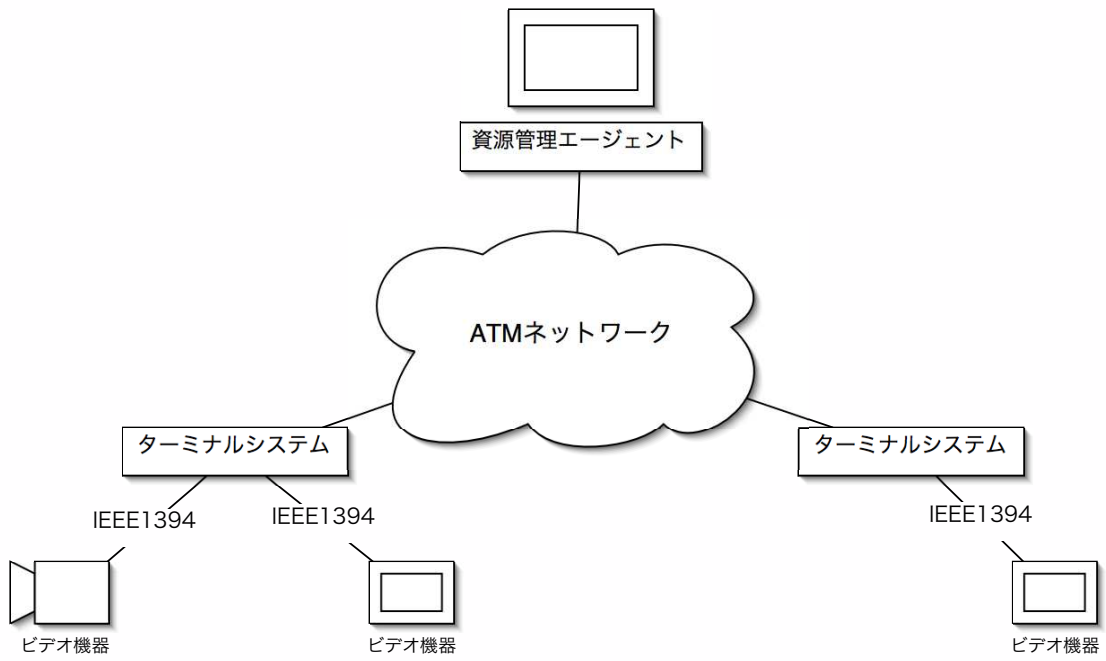

図 5.1: JAIST VideoLAN

ネージャに置き換え、またターミナルシステムにデバイスコントローラとしての機能を加えることで実装が行われる。

本実装では後者の JAIST VideoLAN のシステム自体に変更を加えるという方法を用いた。この方法では、先に述べたようにターミナルシステムをデバイスコントローラとして動作するように変更する必要があるが、本実装ではターミナルシステムとリソースマネージャの間にデバイスコントローラとして動作する PC を配置する事でターミナルシステム自体には手を加えないという方法を用いた。

5.1.2 DVTS

DVTS は図 5.4 のような構成になる。DVTS は JAIST VideoLAN と同様に IEEE1394 機器の DV データを利用するが ATM ではなく IP ネットワークを利用してデータの転送を行う。DVTS ではビデオネットワークの管理システムと呼べるようなものはなく IEEE1394 インターフェースを持ちビデオ機器が接続された PC 間でデータの送受信を行うだけである。ビデオデータ送信側の PC では受信側の PC の IP アドレスを指定して dvsend コマンドを実行しビデオデータ受信側の PC では dvrecv コマンドを実行する。これにより送信側の PC の IEEE1394 インターフェースに接続されたビデオカメラ等の機器から DV データがビデオネットワーク介して送信され受信側の PC が受信する。送信側の PC は DV データを RTP パケットに格納し受信側の PC に向けて送信を行う。受信側の PC が RTP パケッ

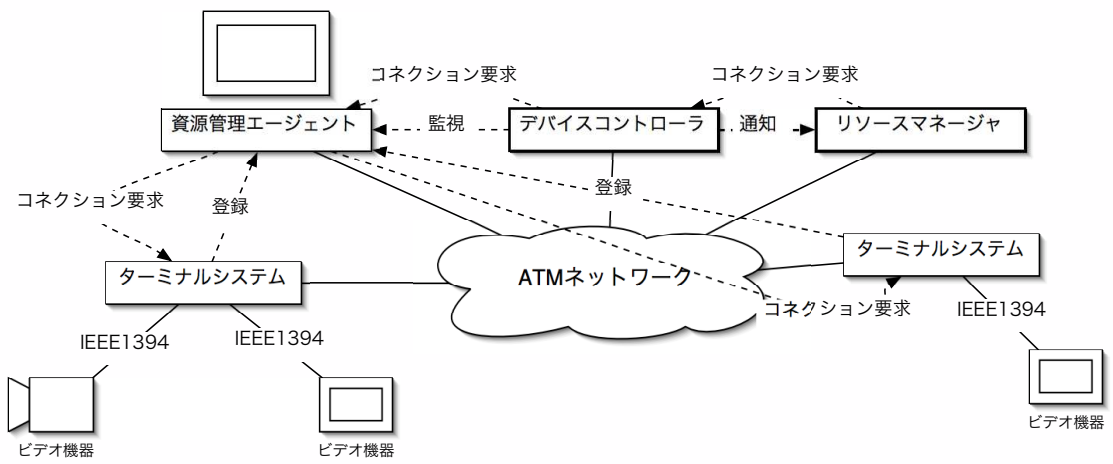

図 5.2: JAIST VideoLAN における実装 1

トを受信すると再び DV データに戻され受信側の PC の IEEE1394 インターフェースに接続しているビデオ機器に対してデータが送信される。

DVTS における実装

DVTS ではビデオ機器間のコネクションやビデオ機器の接続状況管理しているシステムは存在しない。接続の両端で dvsend と dvrecv が実行されていればビデオデータが転送されるだけである。DVTS への本システムの実装は、デバイスコントローラを DVTS を実行させる PC 上に構築するこで行った。デバイスコントローラはビデオ機器の接続を監視しており接続が行われた際にリソースマネージャに登録を行う。また取り外された場合には登録の抹消を行う。また、リソースマネージャからの接続要求を受け取るとビデオデータの方向にあわせて dvsend か dvrecv を実行することでビデオデータの実際の送受信を行うことになる。

5.2 ノードアドレスとネットワークアドレス

本実装ではノードアドレスは 64 の符号無し整数を用いた。また、ネットワークアドレスは 16bit の符号無し整数を用いた。これらのアドレスは文字列表記では十六進数を用いてノードアドレスでは 16bit ごとに”:”を挿入する。またすべて 0 のアドレスと 1 のアドレスは利用できない。以下のようなアドレスが有効なアドレスである。

- **ネットワークアドレス**　0001 - fffe

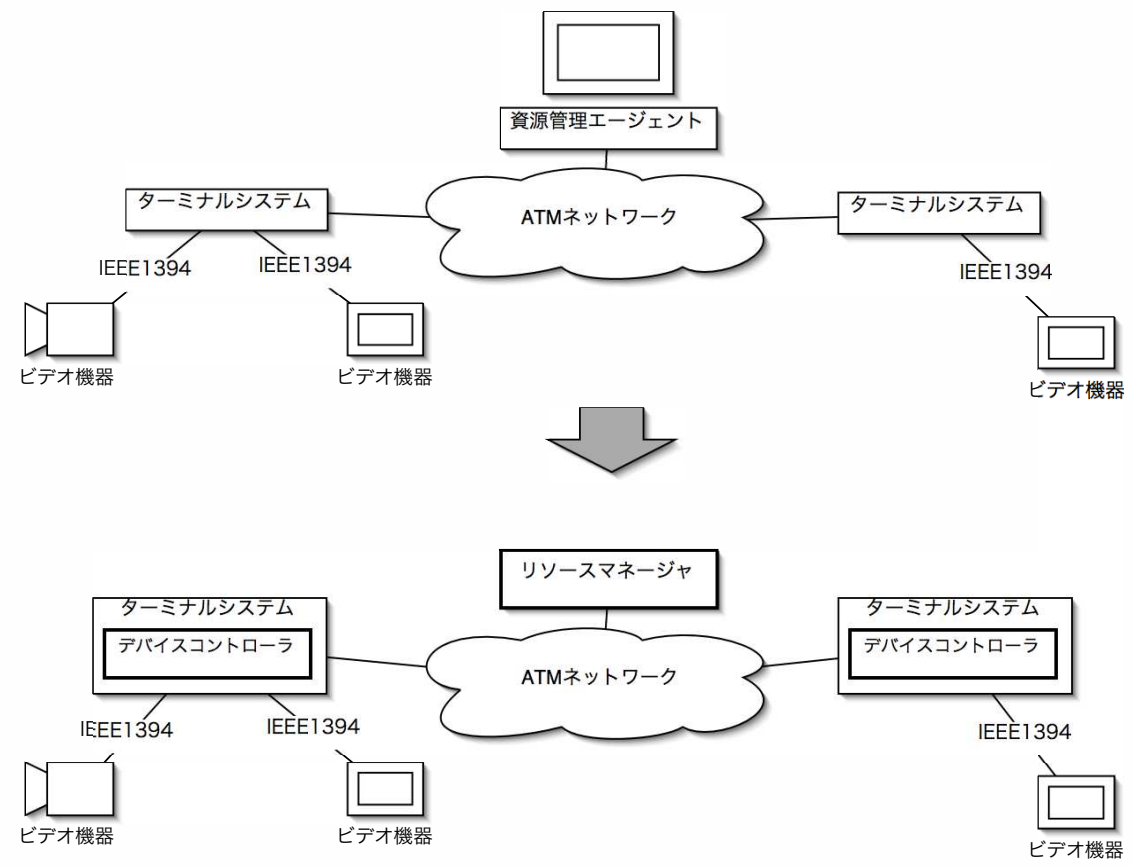

図 5.3: JAIST VideoLAN における実装 2

- **ノードアドレス**　0000:0000:0000:0001 - ffff:ffff:ffff:fffe

ノードアドレスはローカルなビデオネットワーク内でのバーチャルビデオノードを識別するのに利用される。またネットワークアドレスはビデオネットワークを識別するのに利用される。以下のクラスによりノードアドレスとネットワークアドレスは管理される。

- class NodeAddress
- class NetAddress

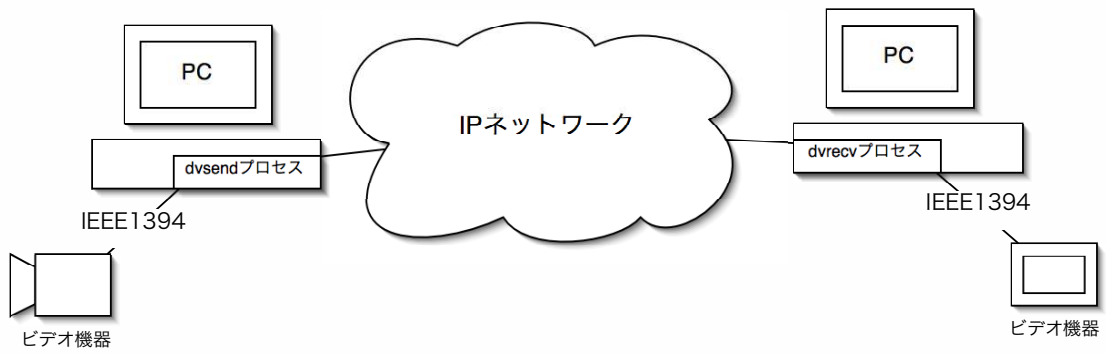

図 5.4: DVTS の構成

5.3 リソースマネージャとデバイスコントローラに共通するクラス

5.3.1 メッセージの送受信

メッセージとして Message、ネットワークに Message を転送するためのクラスとして Socket を利用する。デバイスコントローラとリソースマネージャ、またリソースマネージャ間は IP ネットワークを利用して通信を行う。このためデバイスコントローラとリソースマネージャは相互に接続可能な IP ネットワークに接続していなければならない。Message は Peer 構造体を内包することで送信先あるいは送信元の IP アドレスとポート番号を指定する。 Socket の send メンバ関数により Message は Peer 構造体が指し示す宛先に転送される。また、Socket の recv メンバ関数を実行することで、 Message オブジェクトとして受信したメッセージを受け取ることができる。

- class Socket
- class Message
- struct Peer

5.3.2 プログラム内でのメッセージの配送

受信したメッセージは図 5.5 のように配送される。配送を行うためのクラスは以下の二つである。

- class Dispatcher
- class DispatchReceiver

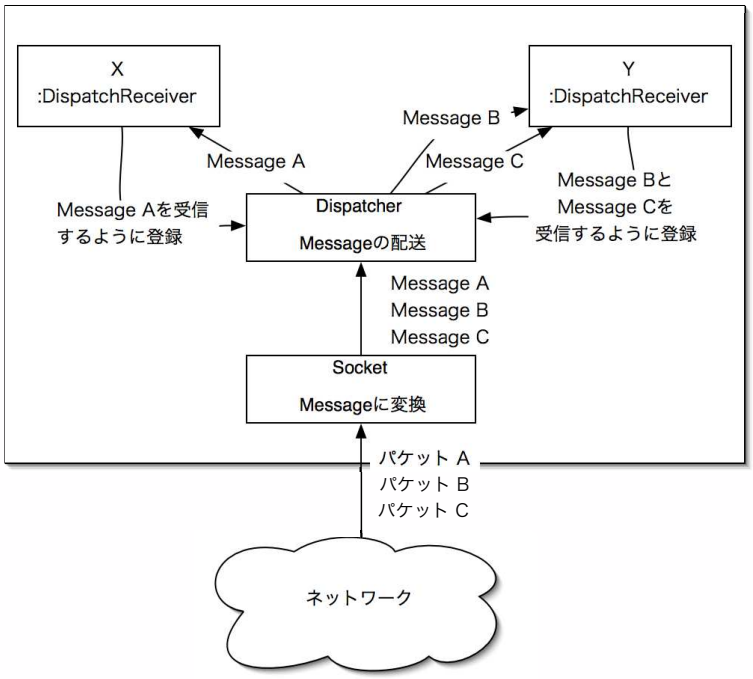

図 5.5: メッセージの配送

Dispatcher がメッセージの振り分けを行い、登録された DispatchReceiver の派生クラスのメンバ関数を呼び出すことでメッセージの配送が行われる。DispatchReceiver は仮想クラスである。

Message を受信したいオブジェクトは DispatchReceiver を継承しメンバ関数 void operator()(Message) をオーバーライドしていなければならない。Dispatcher のオブジェクトに対してキーとなる文字列と DispatchReceiver の派生クラスのオブジェクトへのポインタを引数として attach メンバ関数を呼び出すと、メッセージを受信するたびにメンバ関数 void operator()(Message) が呼び出されるようになる。

5.3.3 他オブジェクトへの情報通知のためのクラス

内部情報に変更があった場合に他のオブジェクトに変更の通知を行うためのクラスについて説明する。NotificationReceiver は仮想クラスであり利用するためにはメンバ関数を実装した派生クラスが必要である。通知を行いクラスは NotificationReceiver の派生クラスのオブジェクトを登録するためのメンバ関数を持ち、また内部情報に変更があった時には登録された NotificationReceiver のメンバ関数 void operator() を呼び出す。これらの動作は図 5.6 のようになる。

- class NotificationReceiver

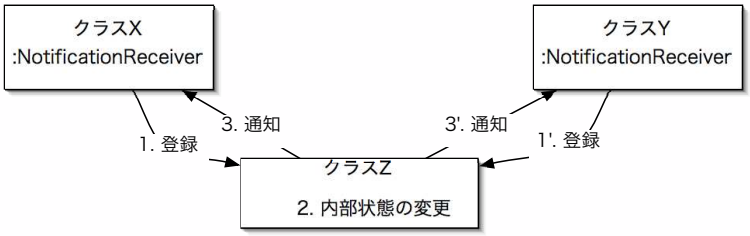

図 5.6: 内部情報変更の通知

5.4 リソースマネージャの概要

ここではリソースマネージャの実装の概要について説明する。リソースマネージャは対象となるビデオネットワークに依存しないコンポーネントである。

5.4.1 リソースマネージャを構成する主なクラス

ResourceManager がメインのクラスとなり、デバイスコントローラなどの他の機器からのメッセージの受信を行う。受信したメッセージは Message クラスのオブジェクトに変換されたのち Dispatcher に渡され DeviceManager などの他のサブコンポーネントに配送される。リソースマネージャを構成する主なクラスは以下の六つである。

- class ResourceManager
- class DeviceManager
- class ConnectionManager
- class SessionManager
- class RoutingEngine

リソースマネージャ内の Socket で受信した Message の流れは図 5.7 のようになる。

5.4.2 データベース

本実装ではデータベースシステムを用いることで保持する必要のあるデータを管理する。以下のようなテーブルをデータベースシステムに構築した。

DeviceController

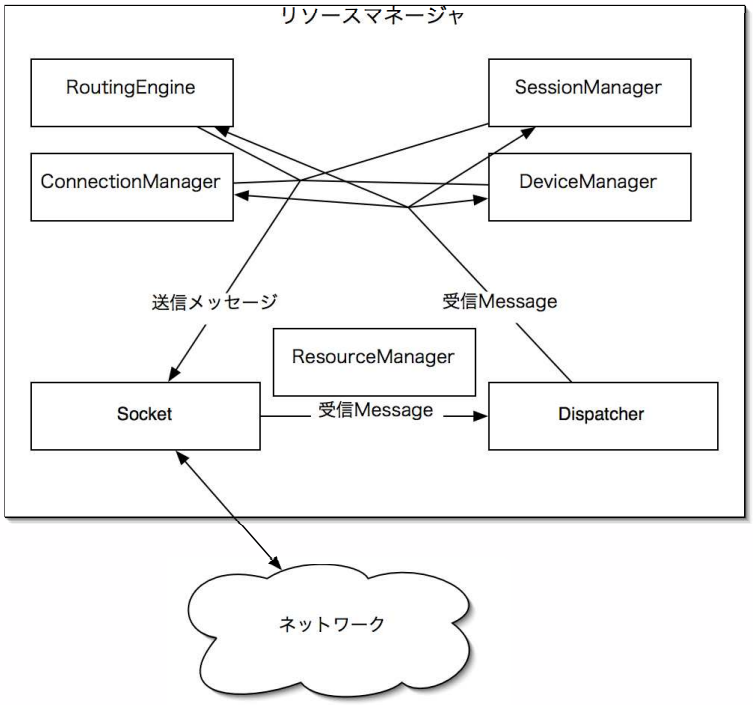

図 5.7: リソースマネージャで利用されるクラス

ID	IPAddress	Port
デバイスコントローラ ID	IP アドレス	ポート番号

ID

各デバイスコントローラを識別する番号である。全てのデバイスコントローラはローカルなビデオネットワークにおいて一意に定まる番号を持っている。

IPAddress

デバイスコントローラの IP アドレスである。

Port

デバイスコントローラが利用しているポート番号である。IP アドレスとこのポート番号によってデバイスコントローラとの通信が可能となる。

VirtualVideoNode

NodeAddress	Direction	DeviceControllerID
ノードアドレス	ビデオデータの方向	デバイスコントローラ ID

NodeAddress

バーチャルビデオノードに割り当てられたノードアドレスである。同一のビデオネットワーク内では同じノードアドレスをもったバーチャルビデオノードが複数存在することは許されない。

Direction

バーチャルビデオノードが対応するビデオ機器のビデオデータの入出力方向である。ビデオ機器からビデオデータが出力される OUT かビデオデータが入力されモニタなどに出力する IN のどちらかになる。

DeviceControllerID

バーチャルビデオノードが存在するデバイスコントローラの番号である。

VideoGateway

LocalNodeAddress	RemoteNetAddress	RemoveNodeAddress
ローカルのノードアドレス	リモートのノードアドレス	リモートのネットワークアドレス

LocalNodeAddress

ビデオゲートウェイでローカルのビデオネットワークに存在するバーチャルビデオノードのノードアドレスである。また、このノードアドレスをキーとして VirtualVideoNode テーブルの検索を行うことで対応するバーチャルビデオノードの情報を得られる。

RemoteNetAddress

ビデオゲートウェイが接続している先のネットワークアドレスである。

RemoteNodeAddress

ビデオゲートウェイが接続している先のバーチャルビデオアドレスに割り当てられたノードアドレスである。LocalNodeAddress をノードアドレスとするバーチャルビデオノードの入出力方向が IN である場合には、LocalNodeAddress を持ったバーチャルビデオノードと対になり RemoteNetAddress をネットワークアドレスとするビデオネットワークに接続している RemoteNodeAddress の入出力方向は OUT になっていなければならない。この場合には、ローカルのビデオネットワークから RemoteNetAddress を持ったビデオネットワークへとビデオデータを送信するために利用されるビデオゲートウェイであることが分かる。

Connection

ID	SrcNodeAddress	DstNodeAddress
コネクション ID	送信者のノードアドレス	受信者のノードアドレス

ID

コネクションの番号である。同一のビデオネットワーク内では異なるコネクションが同じコネクション ID を持つことはない。

SrcNodeAddress

このコネクションの送信元となるバーチャルビデオノードのノードアドレスである。このバーチャルビデオノードの Direction は OUT になっていなければならない。

DstNodeAddress

このコネクションの受信先となるバーチャルビデオノードのノードアドレスである。このバーチャルビデオノードの Direction は IN になっていなければならない。

Session

ID	SrcNetAddress	SrcNodeAddress	DstNetAddress	DstNodeAddress	ConnectionID
セッション ID	送信側ネットワークアドレス	送信側ノードアドレス	受信側ネットワークアドレス	受信側ノードアドレス	コネクション ID

ID

セッションを識別する番号である。このセッション番号はビデオインターネットワーキングアーキテクチャ内でユニークでなければならない。そこで、ID は 32bit とし、上位 16bit に送信元のビデオ機器の存在するビデオネットワークに割り当てられたネットワークアドレスを割り当てることで異なったビデオネットワークで ID が同じにならないことを保証する。

SrcNetAddress

送信元のビデオ機器が接続しているビデオネットワークのネットワークアドレスである。

SrcNodeAddress

送信元のビデオ機器に対応するバーチャルビデオノードに割り当てられたノードアドレスである。

DstNetAddress

受信先のビデオ機器が接続しているビデオネットワークのネットワークアドレスである。

DstNodeAddress
受信先のビデオ機器に対応するバーチャルビデオノードに割り当てられたノードアドレスである。

ConnectionID
ローカルのビデオネットワークに張られたコネクションで、このセッションの一部となっているもののコネクション番号である。

Neighbor

NetAddress	IPAddress	Port	Alive
ネットワークアドレス	IP アドレス	ポート番号	生存チェック

NetAddress
近隣に存在するビデオネットワークのネットワークアドレスである。

IPAddress
ビデオネットワークを管理しているリソースマネージャの IP アドレスである。

Port
ビデオネットワークを管理しているリソースマネージャと通信を行う際に利用されるポート番号である。IPAddress と Port を用いることで近隣のビデオネットワークのリソースマネージャと通信が可能となる。

Alive
近隣のビデオネットワークのリソースマネージャが動作しているかを格納する。動作が確認されている場合は YES になり、動作をしていない場合は NO になる。

5.5 デバイスコントローラの概要

ここではデバイスコントローラの実装の概要について説明する。

5.5.1 デバイスコントローラで利用されるクラス

DeviceController クラスがメインのクラスとなる。DeviceController はリソースマネージャからのメッセージを受信し ResourceManager が行うのと同様に Message に変換をし、Dispatcher を利用して他のオブジェクトに配送される。また、DeviceDetector はビデオ機器の接続を監視するクラスであり新たにビデオ機器が接続された場合には VirtualVideoNode を生成する。また、DeviceDetector は仮想クラスでありターゲットとなるビデオネットワークにあわせて派生クラスを定義する必要がある。

- class DeviceController
- class VirtualVideoNode
- class DeviceDetector

これらのクラスの構成は図 5.8 のようになる。デバイスコントローラが受信した Message は DeviceController が Socket から読みだし Dispatcher に渡す。Dispatcher に渡された Message は設定されたオブジェクトに配送されるが、現在のところ DeviceController 以外が受信することはない。DeviceController は受信した Message の中身を解釈し VirtualVideoNode にコネクションの命令などを送る。

また、DeviceController は定期的に DeviceDetector にデバイス検出を要求する。DeviceDetector は新たに検出されたビデオ機器のために VirtualVideoNode を生成すると同時に取り外されたビデオ機器の VirtualVideoNode についての情報も DeviceController に渡す。これにより DeviceController は新たな VirtualVideoNode を得、また不必要な VirtualVideoNode を解放し、バーチャルビデオノードの登録、あるいは登録抹消の Message を Socket を利用してリソースマネージャに送信する。

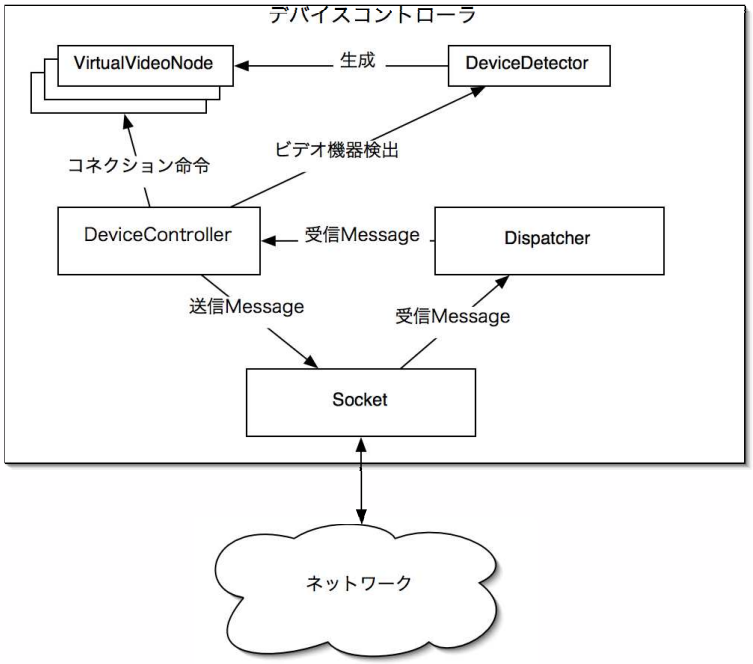

図 5.8: デバイスコントローラで利用されるクラス

5.6 JAIST VideoLANにおけるデバイスコントローラ

JAIST VideoLANにおけるデバイスコントローラの実装について説明する。

5.6.1 アドレス変換

JAIST VideoLANにおけるビデオ機器のアドレスはNUIDと呼ばれるIEEE1394のアドレスであるが、シグナリングを行う際には160ビットの長さを持つATMアドレスが利用される。NUIDは64ビットでありノードアドレスに格納できるが、ATMアドレスはノードアドレスの64ビットよりも大きいためノードアドレスとして用いることができない。しかしながら、シグナリングを行う場合には160ビットのATMアドレスが必要となるためアドレス変換のためのシステムが必要となる。このシステムでは160ビットのATMアドレスと64ビットのNUIDの組を持ち、NUIDをキーとしてATMアドレスを引き出すことができなければならない。このシステムは以下のようなテーブルをリソースマネージャに新たに追加することで実現した。デバイスコントローラはバーチャルビデオノードを登録する時に同時にNUIDとATMの登録も行わなければならない。

NUIDATM

NUID	ATM
ノードユニークID	ATMアドレス

5.6.2 デバイスの検出

デバイスコントローラとターミナルシステム間はIPで通信を行う。デバイスコントローラはリソースマネージャからの命令をターミナルシステムにあわせて変換をすることになる。またデバイスコントローラは定期的にターミナルシステムにビデオ機器の存在を問い合わせることでビデオ機器の接続や取り外しが行われたことを感知する。これを行うターミナルシステムの命令はGETNODESである。

5.6.3 ターミナルシステムとデバイスコントローラの通信

ターミナルシステムが受信するメッセージで、本システムが利用するメッセージは以下の四つである。

- GETNODES
  現在接続しているビデオ機器の情報を得る。
- CONNECTSEND
  ATMのシグナリングを行いコネクションを確立する。

- DISCONNECTSEND
  確立しているコネクションを切断する。

- GETCONNECTRESULT
  コネクションが張られたかチェックをする。

デバイスコントローラはターミナルシステムに対して定期的に GETNODES 命令を発行することで新たなビデオ機器の接続や、切断についての情報を得る。デバイスコントローラはリソースマネージャからメッセージを受信すると適当な命令に置き換えターミナルシステムに発行する。最初にリソースマネージャに転送するメッセージについて述べる。デバイスコントローラはリソースマネージャにバーチャルビデオノードの登録を行わなければならない。そのためにはデバイスコントローラは接続しているビデオ機器があるかどうか調べなければならないのだが、これはターミナルシステムに対して GETNODES メッセージを送信することで行う。GETNODES メッセージを受信したターミナルシステムは現在接続しているビデオ機器の情報として NUID を返信する。こうして得た NUID をノードアドレスとしてバーチャルビデオノードを構築しリソースマネージャに登録する。

ターミナルシステムへのメッセージ	リソースマネージャへのメッセージ
GETNODES	REQUEST_ATTACHVVN
GETNODES	REQUEST_DETACHVVN

次に接続する際に利用されるメッセージについて述べる。接続するためには送信元のビデオ機器の接続されたターミナルシステムへは CONNECTSEND メッセージを送信し、受信先のビデオ機器の接続されたターミナルシステムへは GETCONNECTRESULT を送信する。CONNECTSEND は実際にコネクションを行うように命令をし、GETCONNECTRESULT はコネクションが正しく張られているか調べるものである。本システムでは REQUEST_CONNECTTO と REQUEST_CONNECTFROM のどちらが先にデバイスコントローラに到達するか規定していないため、REQUEST_CONNECTFROM を受信したデバイスコントローラがすぐに GETCONNECTRESULT をターミナルシステムに送信しても失敗をする可能性がある。そこで、受信側のデバイスコントローラはしばらく待ってから GETCONNECTRESULT を行う必要がある。

リソースマネージャからのメッセージ	ターミナルシステムへのメッセージ
REQUEST_CONNECTTO	CONNECTSEND
REQUEST_CONNECTFROM	GETCONNECTRESULT
REQUEST_DISCONNECTTO	DISCONNECTSEND
REQUEST_DISCONNECTFROM	GETCONNECTRESULT

5.7 DVTSにおけるデバイスコントローラ

DVTSを利用したデバイスコントローラの実装について説明する。

5.7.1 アドレス変換

ノードアドレスとしてDVTSが動作するPCのIPアドレスを利用した。64bitの長さを持つノードアドレスよりも短いためノードアドレスに完全に収まる。IPアドレスに対応しないノードアドレスの一部は0に設定される。

5.7.2 DVTSコマンドの実行

リソースマネージャからコネクション命令を受信すると、バーチャルビデオノードに設定されているビデオデータの方向によってdvsendコマンドかdvrecvコマンドが実行される。dvsendではアドレス変換でノードアドレスから変換されたIPアドレスが宛先として利用される。

5.8 ビデオゲートウェイの構成

ビデオゲートウェイは上記で説明したJAIST VideoLAN用のデバイスコントローラとDVTS用のデバイスコントローラを組合せ相互に情報のやりとりを行うようにしたものである。双方のデバイスコントローラ間でゲートウェイとなるバーチャルビデオノードのノードアドレスとネットワークアドレスの情報を交換する。JAIST VideoLANとDVTSの間のビデオゲートウェイは図5.9のような構成になる。ここで示したものはJAIST VideoLANからDVTSへビデオデータが転送されるビデオゲートウェイであるが、逆向きのビデオゲートウェイも同等のものになる。本構成ではビデオゲートウェイで利用されるビデオ機器としてNTSC-DVコンバーターを利用しビデオゲートウェイ内の内部ビデオデータとしてNTSCを利用した。

5.9 動作

ここでは実装したビデオインターネットワーキングの動作とそれに伴うメッセージの交換について述べる。

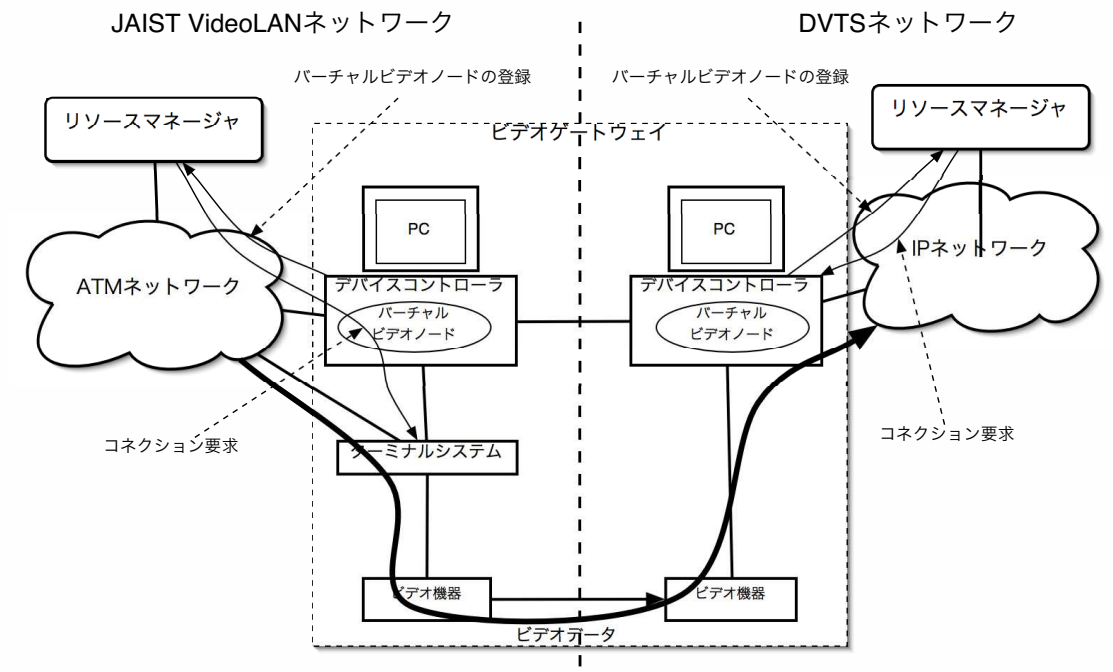

図 5.9: JAIST VideoLAN と DVTS 間のビデオゲートウェイ

5.9.1 デバイスコントローラの登録と抹消

デバイスコントローラは起動時にリソースマネージャ内にあるデバイスマネージャに自分自身の登録を行う。この際、デバイスコントローラにはユニークな番号が割り当てられ以後この番号を用いることで自分自身の識別を行いリソースマネージャと通信を行う。

5.9.2 バーチャルビデオノードの登録と抹消

デバイスコントローラは接続されたビデオ機器の情報を常に監視しており、新たに接続された場合にはバーチャルビデオノードをデバイスマネージャに登録し、また切断された場合にはデバイスマネージャから登録を抹消する。

5.9.3 ビデオゲートウェイの登録と抹消

ビデオゲートウェイの登録は対応するバーチャルビデオノードの登録後に行う。ビデオゲートウェイを登録するにはビデオゲートウェイで対となるビデオ機器に対応したバーチャルビデオノードのノードアドレスとそのバーチャルビデオノードが存在するビデオネットワークのネットワークアドレスが必要である。

5.9.4 コネクションの確立と切断

コネクションを確立するために、送信側にはREQUEST_CONNECTTOを送信し、受信側にはREQUEST_CONNECTFROMを送信する。これらのメッセージは送信側と受信側のノードアドレスも一緒に渡される。これらによってどのノードアドレスからシグナリングが行われ、またどノードアドレスにシグナリングが行われなければならないか分かる。同様にコネクションを切断する際には送信側にはREQUEST_DISCONNECTOが送信され、受信側にはREQUEST_DISCONNECTFROMが送信される。

5.9.5 セッションの確立と切断

セッションを確立するためにREQUEST_MAKESESSIONメッセージがリソースマネージャ間で転送される。REQUEST_MAKESESSIONを受信したリソースマネージャはDispatcherを通してセッションマネージャにREQUEST_MAKESESSIONが渡される。セッションマネージャはREQUEST_MAKESESSIONと一緒に送信される経由するビデオゲートウェイのノードアドレスを調べる。このノードアドレスの割り当てられたバーチャルビデオノードに対応したビデオ機器からビデオデータがローカルのビデオネットワークに送信される。セッションマネージャはルーティングエンジンを利用し次のビデオネットワークを決める。次にそのビデオネットワークに接続しているビデオゲートウェイのうち現在コネクションを張られていないものを調べ、そのビデオゲートウェイに割り当てられたノードアドレスを得る。また、同時にそのビデオゲートウェイで対となるバーチャルビデオノードのノードアドレスも調べ、次のビデオネットワークに送信するREQUEST_MAKESESSIONで利用する。

5.9.6 経路情報の交換

経路情報の交換にはROUTEINFORMATIONメッセージを利用する。本実装では単純な距離ベクトルプロトコルをもとに実装を行った。あるビデオネットワークが別のビデオネットワークに接続しているためには次の三つを満たしていなければならない。

- Neighborテーブルに登録されている
- NeighborテーブルのAliveの項目がYESである
- そのネットワークに接続しているビデオゲートウェイが存在する

このときに、ローカルのビデオネットワークは近隣のビデオネットワークに接続しておりその距離は1とする。このように集められた情報と近隣から受信したROUTEINFORMATIONメッセージの情報を集めてROUTEINFORMATIONメッセージとして近隣で生存しているリソースマネージャに送信する。また、このときにすべてのネットワークへの

距離は 1 を加えておく。ROUTEINFORMATION メッセージはある決まった時間間隔で送信され、しばらく更新されない項目についてはある一定時間後に自動的に消去される。

5.9.7 デバイスコントローラとリソースマネージャ間のメッセージ

デバイスコントローラがリソースマネージャに送信するメッセージは以下の六つである。また、これらのメッセージをリソースマネージャが受信をすると登録が成功したか失敗したかを伝えるメッセージが返信される。成功した場合には ACCEPT で始まるメッセージが返信され失敗した場合には REJECT で始まるメッセージが返信される。これらのメッセージは全てデバイスマネージャが受信をしそれぞれのデータベースに情報を格納、あるいはデータベースから情報を消去する。バーチャルビデオノードが登録されるまでのメッセージの流れを図 5.10 に示す。

- **要求: REQUEST_ATTACHDC**
  新たにデバイスコントローラを登録する。登録が成功した場合にはリソースマネージャはデバイスコントローラの ID を返す。
  - **成功: ACCEPT_ATTACHDC**
  - **失敗: REJECT_ATTACHDC**
- **要求: REQUEST_DETACHDC**
  デバイスコントローラの登録を抹消する。抹消するデバイスコントローラの ID を指定する。
  - **成功: ACCEPT_DETACHDC**
  - **失敗: REJECT_DETACHDC**
- **要求: REQUEST_ATTACHVVN**
  新たにバーチャルビデオノードを登録する。
  - **成功: ACCEPT_ATTACHVVN**
  - **失敗: REJECT_ATTACHVVN**
- **要求: REQUEST_DETACHVVN**
  バーチャルビデオノードの登録を抹消する。
  - **成功: ACCEPT_DETACHVVN**
  - **失敗: REJECT_DETACHVVN**

- **要求: REQUEST_ATTACHGW**
  新たにビデオゲートウェイを登録する。ビデオゲートウェイとして設定されるバーチャルビデオノードは予め登録されていなければならない。
  - **成功: ACCEPT_ATTACHGW**
  - **失敗: REJECT_ATTACHGW**
- **要求: REQUEST_DETACHGW**
  ビデオゲートウェイの登録を抹消する。
  - **成功: ACCEPT_DETACHGW**
  - **失敗: REJECT_DETACHGW**

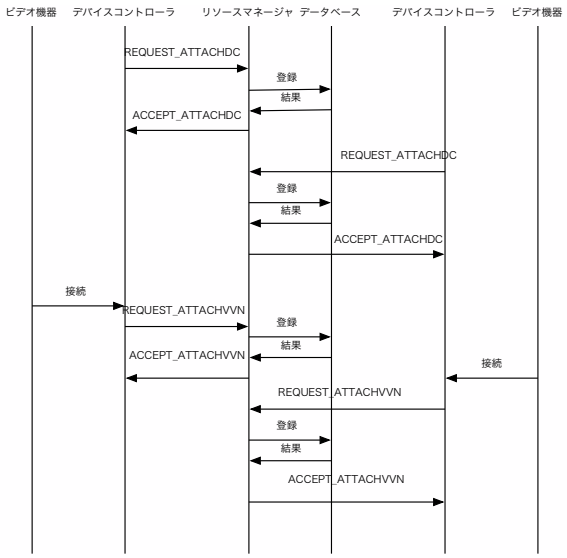

図 5.10: バーチャルビデオノードの登録

リソースマネージャからデバイスコントローラに送信されるメッセージは以下の四つである。これらのメッセージはコネクションマネージャからデバイスコントローラに送信される。これらのメッセージを用いてコネクションが張られるまでを図 5.11 に示した。

- **REQUEST_CONNECTTO**
  送信元としてバーチャルビデオノードのコネクションを要求する。
  - **成功: ACCEPT_CONNECTTO**
  - **失敗: REJECT_CONNECTTO**

- **REQUEST_CONNECTFROM**
  受信先としてバーチャルビデオノードのコネクションを要求する。
  - **成功: ACCEPT_CONNECTFROM**
  - **失敗: REJECT_CONNECTFROM**
- **REQUEST_DISCONNECTTO**
  送信元としてバーチャルビデオノードの切断を要求する。
  - **成功: ACCEPT_DISCONNECTTO**
  - **失敗: REJECT_DISCONNECTTO**
- **REQUEST_DISCONNECTFROM**
  受信先としてバーチャルビデオノードの切断を要求する。
  - **成功: ACCEPT_DISCONNECTFROM**
  - **失敗: REJECT_DISCONNECTFROM**

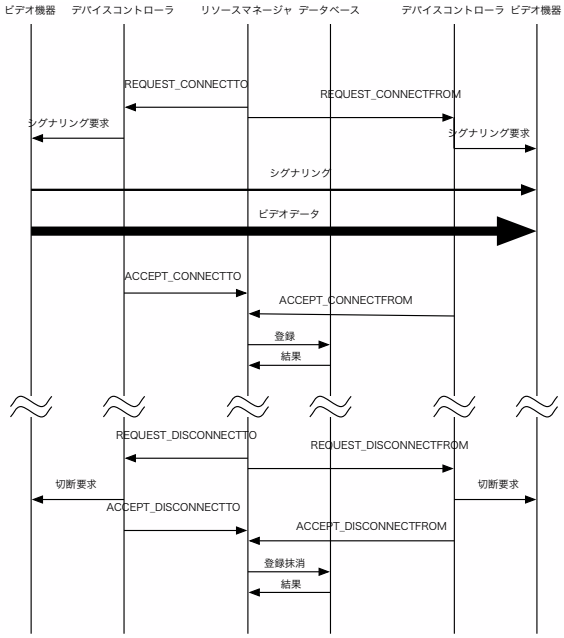

図 5.11: バーチャルビデオノードの登録

5.9.8 リソースマネージャ間のメッセージ

リソースマネージャ間で転送されるメッセージは以下の四つである。ALIVE と ROUTE-INFORMATION はルーティングエンジンが送受信するメッセージであり、REQUEST_MAKESESSION と REQUEST_CLEANSESSION はセッションマネージャが送受信を行うメッセージである。セッションを張る際に最初のリソースマネージャの動作と送受信されるメッセージは図 5.12 のようになる。また、途中のリソースマネージャの動作はメッセージは図 5.13 のようになる。

- **ALIVE**
  リソースマネージャ間での生存確認を行う。このメッセージに対する返信は行われない。近隣のリソースマネージャであっても、このメッセージを一定時間送信してこないリソースマネージャは動作していないとみなし、Neighbor テーブルの Alive 項目を NO にする。また、このメッセージを送信する近隣のリソースマネージャについては Alive 項目を YES にする。

- **ROUTEINFORMATION**
  経路情報を交換する。このメッセージに経路の情報を格納し近隣のリソースマネージャに転送することで経路の交換が行われる。このメッセージに対する返信は行われない。

- **REQUEST_MAKESESSION**
  新たにセッションを確立する。送信元のビデオネットワークに存在するリソースマネージャが次のビデオネットワークに最初に送信を行い、受信先のビデオ機器が存在するビデオネットワークに到達するまで順々に送られる。

  - **成功: ACCEPT_MAKESESSION**
  - **失敗: REJECT_MAKESESSION**

- **REQUEST_CLEANSESSION**
  セッションを抹消する。

  - **成功: ACCEPT_CLEANSESSION**
  - **失敗: REJECT_CLEANSESSION**

5.10 接続実験

本実装の実験を行うために図 5.14 のような構成でビデオネットワークを構築した。GUI からセッションの設定を行うことで、左のビデオカメラから右のモニタまでセッションが張られることを確認した。また DVTS 側のリソースマネージャのデータベースの状態は以下のようになった。

DeviceController

id	ip_address	port
1	192.168.0.1	5555
2	192.168.0.2	5555

VirtualVideoNode

node_address	direction	device_controller_id
0000:0000:c0a8:0001	OUT	1
0000:0000:c0a8:0002	IN	2

VideoGateway

local_node_address	remote_net_address	remote_node_address
0000:0000:c0a8:0002	0002	0800:4603:004d:efdc

Connection

id	src_node_address	dst_node_address
1	0000:0000:c0a8:0001	0000:0000:c0a8:0002

Session

id	src_net_address	src_node_address	
65537	0001	0000:0000:c0a8:0001	

dst_net_address	dst_node_address	connection_id
0002	0800:4603:004d:f18e	1

Neighbor

net_address	ip_address	port	alive
0002	192.168.0.3	5554	YES

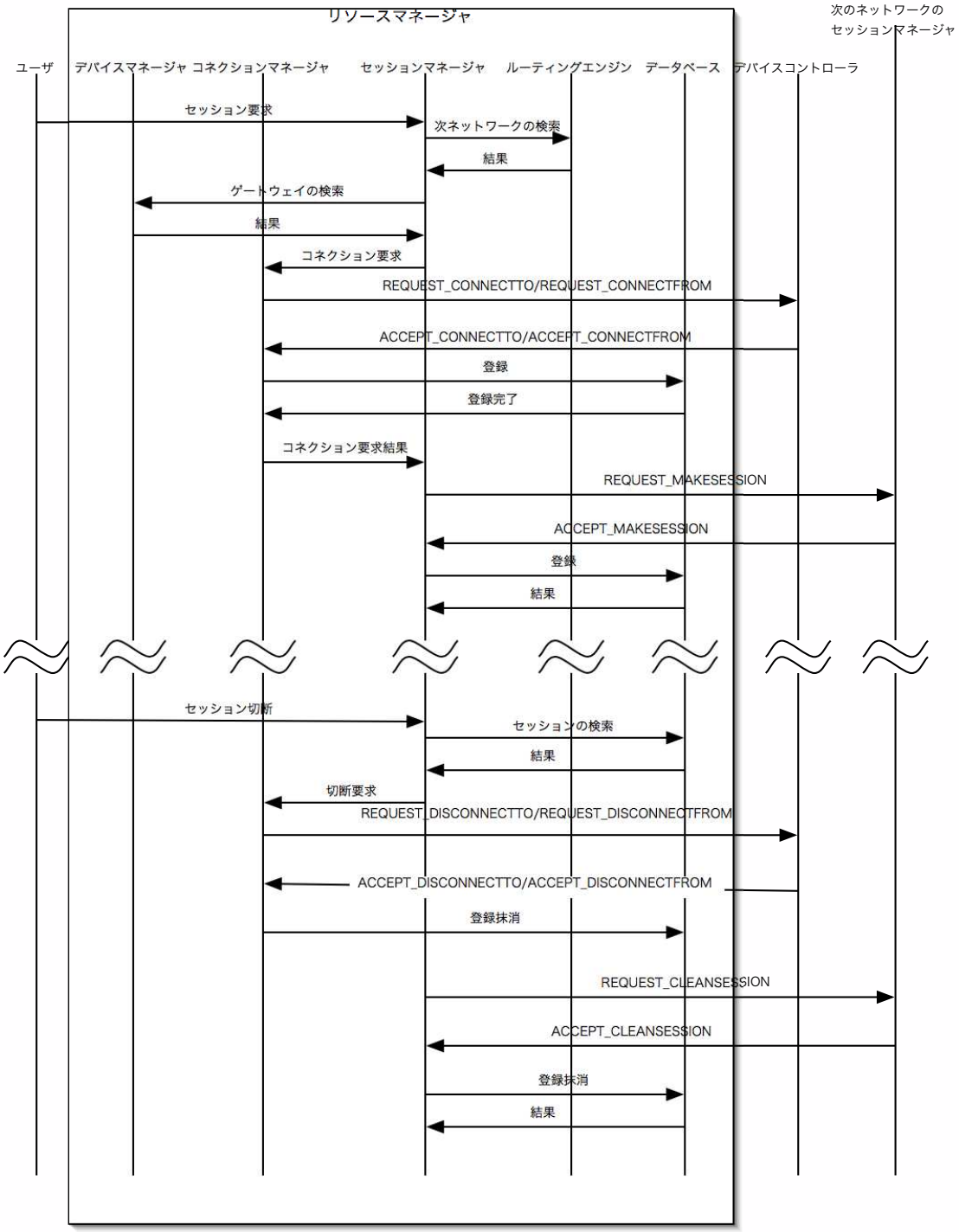

図 5.12: 最初のリソースマネージャがセッションを張る際の動作

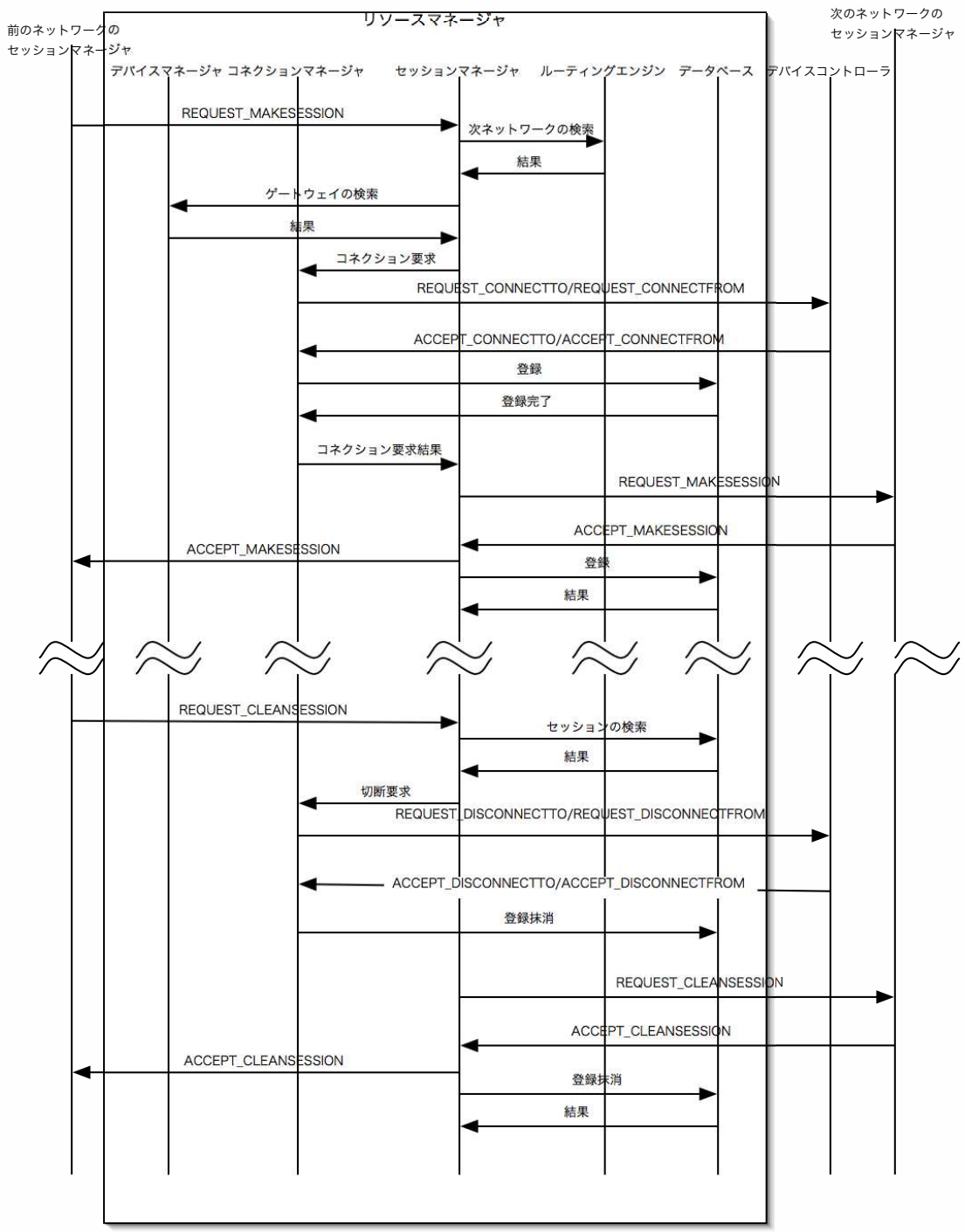

図 5.13: 途中のリソースマネージャがセッションを張る際の動作

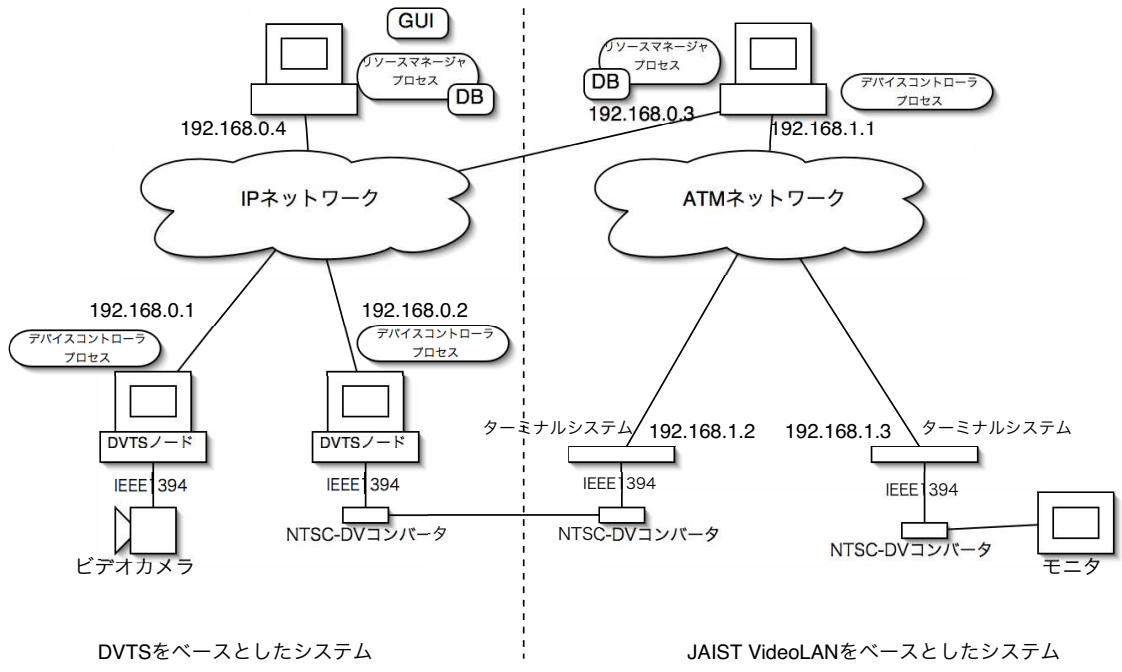
GUI
リソースマネージャ
プロセス
DB
192.168.0.4
IPネットワーク
192.168.0.1
192.168.0.2
デバイスコントローラ
プロセス
DVTSノード
IEEE1394
ビデオカメラ
NTSC-DVコンバータ
DVTSをベースとしたシステム
リソースマネージャ
プロセス
DB
192.168.0.3
192.168.1.1
デバイスコントローラ
プロセス
ATMネットワーク
ターミナルシステム
192.168.1.2
192.168.1.3
IEEE1394
NTSC-DVコンバータ
モニタ
JAIST VideoLANをベースとしたシステム

図 5.14: 機器の構成

第6章　考察と今後の課題

本章では本研究で提案したビデオインターネットワーキングアーキテクチャについて考察する。また、本システムをより実用的にし、適用範囲を広げるために必要となる項目についていくつか説明する。

6.1　資源管理機構の分散化

本研究で提案を行ったビデオインターネットワーキングアーキテクチャはリソースマネージャと呼ばれる資源管理機構を個々のビデオネットワークが一つずつ持っている。他には、すべての情報を一つの資源管理機構で管理するという方法や、資源管理はすべて末端の機器が行うという方法が考えられる。

すべての情報の管理を一つの資源管理機構で行う方法では、取り扱わなければならない情報の量は多くなるが他の資源管理機構との間での情報のやりとりが必要ないためシステムとしては単純になると考えられる。しかしこの方式では以下の欠点がある。

- 一つの資源管理機構の故障によりシステム全体が動作しなくなる
- 複数の組織で個々に資源管理機構を構築できない

また、末端の機器が資源管理を行う場合には、うまく接続されるかは個々の機器に依存することとなりシステム全体を管理するものはいなくなる。他のシステムに依存するということがなくなりシステムの一部が故障しても他の機器は問題なく動作する。しかしこの方式では以下の欠点がある。

- 組織としてビデオネットワークの管理をできない
- 末端の機器が複雑化する

本システムでは集中型の資源管理機構を利用するが、資源管理機構自体は分散化している。そのため双方の方式の利点を持ちつつ、欠点も補うシステムとなっている。

- 一つの資源管理機構の故障の影響はローカルに抑えられる
- 複数の組織で別々の資源管理機構を構築できる
- コネクションなどの情報は資源管理機構で管理される

6.2 ビデオゲートウェイの必要数

本システムでは一つのビデオストリームを他のビデオネットワークに送信するためにビデオゲートウェイを一つ利用する。ビデオゲートウェイは排他的に利用されるためビデオネットワーク間を転送するビデオデータの数だけビデオゲートウェイが必要となる。また、ビデオゲートウェイのビデオデータの向きは固定であり双方向通信を行うためには向きの異なるビデオゲートウェイが二つ必要となる。従って、接続されたビデオネットワークで相互にビデオデータの送信を行う場合には多数のビデオゲートウェイが必要となる。

ここでは、図 6.1 のようにビデオネットワーク A においてビデオゲートウェイではないビデオ機器が n 台接続され、このビデオネットワークは他のビデオ機器が m 台接続されたビデオネットワーク B に接続している場合を考える。また、すべてのビデオ機器が送受信を行っていると考える。両ビデオネットワークのビデオ機器のうち $\frac{n+m}{2}$ 台は送信を行い、残りの $\frac{n+m}{2}$ 台はビデオデータの受信を行うことになる。ビデオネットワーク A の送信ビデオ機器は $\frac{n}{2}$ 台であり受信ビデオ機器は $\frac{n}{2}$ 台である。これから、(1) ビデオネットワーク A 内のみで送受信するビデオ機器と (2) ビデオネットワーク B に送信するビデオ機器、また (3) ビデオネットワーク B からビデオデータを受信するビデオ機器の数は以下のようになる。

1. $\frac{n^2}{n+m}$
2. $\frac{nm}{2(n+m)}$
3. $\frac{nm}{2(n+m)}$

従ってビデオネットワーク A からビデオネットワーク B にビデオデータを転送するビデオゲートウェイは $\frac{nm}{2(n+m)}$ 台必要である。同様にビデオネットワーク B からビデオネットワーク A にビデオデータを転送するビデオゲートウェイは $\frac{nm}{2(n+m)}$ 台必要であることが分かる。

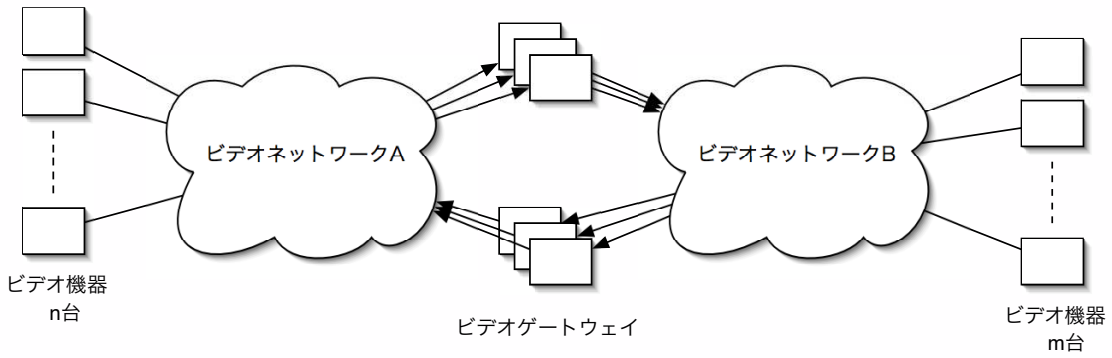

図 6.1: 仮定するビデオネットワーク

6.3 経路の検索

本稿で述べた実装ではセッションを張るときに選ばれるビデオネットワークの経路は目的となるビデオネットワークへの距離のみで選択した。このようなアルゴリズムを用いるとあまり良い品質を得られないビデオネットワークを介することで途中でビデオデータの品質が劣化してしまうことが考えられる。また、ビデオゲートウェイの個数も有限であるため同じ経路を使おうとするセッションの量にも限界がある。そこで以下のような項目に着目をすることでより良い経路選択アルゴリズムが得られると考えられる。

- 品質の劣化
- 遅延
- ビデオゲートウェイの個数

6.4 障害とリソースマネージャの冗長性

リソースマネージャはデータベースを保持している。このデータベースは外部に存在してもよいためリソースマネージャが故障を起こした場合であっても、異なったリソースマネージャが同一のデータベースを利用しサービスを開始することでサービスの停止時間を短くすることが可能である。また同様にデータベースのミラーを構成しておくことで、データベースの故障が生じてもリソースマネージャは実行を続けることが可能となる。

また、ビデオネットワークの網に障害が発生ししばらくデバイスコントローラとリソースマネージャが通信を行えないといった問題が発生することも考えられる。デバイスコントローラがビデオネットワークから取り外される前にリソースマネージャに自分自身の登録の抹消を行うが、障害時には抹消のための通信が不可能となる。このため、デバイスコントローラが取り外されたにも関わらずリソースマネージャにはデバイスコントローラが登録されたままになっているなどといった問題が生じる。

機器の故障やネットワークの障害の直後ではデータベースの内容と実際のビデオネットワークの状態が異なる場合が考えられるためこれらの同期をとらなければならない。本システムはリソースマネージャのデータベースにすべての情報が存在し、その情報の通りにコネクションや機器が構成されているとみなす。矛盾が生じている場合は、データベースの情報とリソースマネージャ外の構成が異なっている場合のことである。これらの状態を正常な状態に戻すためには以下のような作業が必要であると考えられる。

1. データベースの情報を元にコネクションなどを張り直す
2. 存在しないバーチャルビデオノードの登録などは抹消する

また、セッションについては近隣のシステムと矛盾がないようにしていなければならない。各リソースマネージャはセッションが存在する間は、そのセッションで下流に位置す

るビデオネットワークに存在するリソースマネージャに、一定の時間間隔で常に維持メッセージを送信することでセッションの維持を行うことが可能であると考えられる。

6.5 マルチキャスト

本システムではビデオデータをユニキャストすることのみを目的としてきたが、一つのビデオ機器からのデータを複数のビデオ機器が受信する場合にはマルチキャストを利用することが有効である。

マルチキャストでは一つの機器から送信されたデータをネットワーク内で必要に応じてコピーして配送していくことでネットワーク内に流されるデータが必要最低限になる。ユニキャストではすべての宛先に向けて送信者がデータを送信するため帯域が無駄に利用されてしまう。

ビデオネットワークがマルチキャストをサポートしている場合は少しの変更で問題なくマルチキャストが利用可能になると考えられるが、ユニキャストしかできないビデオネットワークではマルチキャストを直接利用することはできない。このようなネットワークでは、一つのビデオデータを受信し複数の宛先に送信するといった機器を用いることでマルチキャストのエミュレーションが可能となる。このようなシステムをビデオネットワーク内に組込み、このシステムを利用するようにリソースマネージャを対応させることでマルチキャストと同等の機能を提供できると考えられる。

6.6 プロダクションレベルでの利用

本システムは放送局のような非常にシビアに映像を扱う場合には適用することはできない。放送局などではすべての機器は同期して動作しており、また時刻に非常に敏感であることが想定される。そこで本システムをこのようなシビアな世界に適用するためには以下のような項目について考慮をしたものにしなければならない。

- ビデオ機器の同期
- ビデオデータの転送遅延の固定

ビデオ機器の同期を行うことで機器間の周波数のずれがなくなる。周波数がわずかにずれていると送信者と受信者の間でフレームレートが少しずつずれることとなり、どこかでつじつまを合わせなければいけなくなる。またこの場合には双方のシステムで時間が異なってしまう。この問題を解決するために、ISDN などの同期信号を持ったネットワークを利用してすべての機器の同期を取るという方法がある。

また、ビデオデータを送信しはじめてから受信するまでの遅延時間が常に一定であるという保証が必要である。これが分からなければ、受信側で映像をスタートしたい時刻を決めてあったとしても、その映像の送信側ではいつビデオデータの送信を始めればいいのか

分からない。遅延の時間が t と分かれば、開始したい時刻の t 前に送信を開始すればいいのである。

これを実現するためには、すべてのビデオネットワークにおいてビデオデータが転送に用いられる時間とビデオゲートウェイの内部で費される時間が計算できなければならない。

他の方法として、時刻の同期した機器間であれば時刻情報をビデオデータとともに送信するといった方法がある。ビデオデータに時刻情報を挿入する事が可能であれば、受信側で送信側が挿入した時刻情報を調べることでビデオデータが転送されるのにどの程度の時間がかかったか遅延を調べることが可能である。

第7章　おわりに

本稿では異なったビデオネットワークを相互接続する際に問題となる点とそれを解決するシステムについて述べた。

異なるビデオネットワークが利用しているアドレス体系はそれぞれ異なっており、新たに包括的なアドレス体系が必要となる。また、ビデオデータを転送するプロトコル、フォーマット、そしてコネクションの構築を行うために必要なシグナリングの方式が異なっているため異なったビデオネットワークに存在するビデオ機器が直接接続することは不可能である。

これらの問題を解決したビデオインターネットワーキングアーキテクチャを提案した。本システムではデバイスコントローラと呼ばれるコンポーネントを導入し、ビデオネットワークで利用されるアドレス体系やシグナリング方式の抽象化を行った。またビデオゲートウェイと呼ばれるビデオデータのフォーマット変換を行うシステムを導入した。また、これらの機器情報やコネクション情報などを一元管理するリソースマネージャと呼ばれる資源管理機構について述べた。

また、JAIST VideoLAN と DVTS におけるビデオインターネットワーキングアーキテクチャの実装について説明した。

最後に本システムをより実用的にするための方法について検討した。

謝辞

本研究を進めるにあたり、新たな着目点などを指摘頂くなど有益な指導を常に賜わりました丹康雄助教授に深く感謝致します。また、貴重な意見を下さった丹研究室の皆様に感謝致します。

最後にどんな時でも暖かく励まし続けて下さった友人と家族に感謝致します。

付 録A　デバイスコントローラとリソースマネージャの両方の処理に関係するクラス

ここで説明するクラスはすべて名前空間 vin に属する。

A.1　NetAddress

ヘッダーファイル Address.h

概要

ノードアドレスを格納する。

コンストラクタ

```
NodeAddress::NodeAddress(std::string &addr);
NodeAddress::NodeAddress(NodeAddress &naddr);
```

説明

文字列表記が addr であるノードアドレスを生成する。また、naddr と同じアドレスを示すノードアドレスを生成する。生成に失敗すると”0000:0000:0000:0000”を渡した場合と同じになる。

メンバ関数

NodeAddress::operator== 等しいか調べる

インターフェース

```
boot NodeAddress::operator==(const NodeAddress &addr) const;
```

説明

ノードアドレスが等しいか調べる等しい場合には true が返り、等しくない場合には false が返る。

NodeAddress::operator= 代入する

インターフェース

```
boot NodeAddress::operator=(const NodeAddress &addr);
```

説明

ノードアドレスの内容を addr と同じものに変換する。

NodeAddress::length アドレスのバイト数を調べる

インターフェース

```
int NodeAddress::length() const;
```

説明

ノードアドレスのバイト数を調べる。バイト数が返る。

NodeAddress::byteAt バイトを得る

インターフェース

```
char byteAt(int i) const;
```

説明

ノードアドレスの i 番目のバイトを返す。i は 0 以上で length() 未満でなければならない。

NodeAddress::operator std::string 文字列に変換する

インターフェース

```
NodeAddress::operator std::string() const;
```

説明

ノードアドレスを文字列に変換する。

A.2 NetAddress

ヘッダーファイル Address.h

概要

ノードアドレスを格納する。

コンストラクタ

```
NetAddress::NetAddress(std::string &addr);
NetAddress::NetAddress(NetAddress &naddr);
```

説明

文字列表記が addr であるネットワークアドレスを生成する。また、naddr と同じアドレスを示すネットワークアドレスを生成する。生成に失敗すると"0000"を渡した場合と同じになる。

メンバ関数

NetAddress::operator== 等しいか調べる

インターフェース

```
boot NetAddress::operator==(const NetAddress &addr) const;
```

説明

ネットワークアドレスが等しいか調べる等しい場合には true が返り、等しくない場合には false が返る。

NetAddress::operator= 代入する

インターフェース

```
boot NetAddress::operator=(const NetAddress &addr);
```

説明

ネットワークアドレスの内容を addr と同じものに変換する。

NetAddress::length アドレスのバイト数を調べる

インターフェース

```
int NetAddress::length() const;
```

説明

ノードアドレスのバイト数を調べる。バイト数が返る。

NetAddress::byteAt バイトを得る

インターフェース

```
char NetAddress::byteAt() const;
```

説明

ネットワークアドレスを文字列に変換する。

NetAddress::operator std::string 文字列に変換する

インターフェース

```
NetAddress::operator std::string() const;
```

説明

ネットワークアドレスを文字列に変換する。

A.3 Peer

ヘッダーファイル Message.h

概要

Message の送受信先の IP アドレスとポート番号を格納する。

コンストラクタ

```
Peer::Peer(uint32_t addr=0, int16_t port=0);
```

説明

IP アドレスが addr でポート番号が port の Peer を構築する。デフォルトでは両方とも 0 となる。

メンバ変数

```
Peer::addr
```

説明

IP アドレスを格納する

```
Peer::port
```

説明
ポート番号を格納する

A.4 Message

ヘッダーファイル Message.h

概要

ネットワークを介して転送されるメッセージである。メッセージは複数の文字列からなり、空白文字一つにより文字列の区切りが示される。

コンストラクタ

```
Message::Message(const std::string &msg);
Message::Message(const std::string &msg, Peer &peer);
```

説明
msg を初期値とした Message を作る。msg は複数の文字列からなりそれぞれの文字列は空白文字一つによって区切られる。また、メッセージの送受信の相手として peer を設定する。peer を指定しない場合はデフォルト値の Peer が利用される。

メンバ関数

Message::length メッセージに含まれる文字列の数

インターフェース

```
int Message::length() const;
```

説明

Message が格納している文字列の個数を返す。

Message::at 文字列を一つ返す

インターフェース

```
std::string Message::at(int n) const;
```

説明

Message が格納している文字列の n 番目のものを返す。最初の文字列は 0 番目になっている。負数あるいは Message::length() 以上の値を渡してはならない。

Message::setAt 文字列を設定する

インターフェース

```
bool Message::setAt(int n, std::string &str);
```

説明

n 番目の文字列として str をセットする。Message::length() 以上の要素にはセットしてはならない。

Message::peer メッセージの送受信先を返す

インターフェース

```
Peer Message::peer() const;
```

説明

このメッセージの送受信先である Peer を返す。メッセージを受信すると、その送り主が設定される。ここで返される値は、その設定された値である。メッセージに返信を行うためには、このメソッドの戻り値宛てにメッセージを送信すればよい。

Message::setPeer メッセージの送受信先を設定する

インターフェース

```
bool Message::setPeer(Peer &peer);
```

説明

このメッセージの送受信先として peer を設定する。このメッセージを Socket::send に渡すと、ここで設定された宛先に送信される。

Message::stringRep 文字列で表現する

インターフェース

```
std::string Message::stringRep() const;
```

説明

このメッセージの文字列での表現を返す。この中には Peer の内容は含まれない。

A.5 Socket

ヘッダーファイル Socket.h

概要

Message を送受信する。

コンストラクタ

```
Socket::Socket(int16_t port);
```

説明

port をポート番号とする Socket を構築する。

メンバ関数

Socket::send メッセージを送信する

インターフェース

```
bool Socket::send(Message &msg);
```

説明

msg を送信する。送信先は msg が保持する Peer によって確定される。Peer が有効でなく送信ができなかった場合には false を返す。

Socket::send メッセージを受信する

インターフェース

```
Message Socket::recv();
```

説明

Message を受信する。少なくとも一つの Message を受信するまで Socket::recv は戻らない。また、Socket が有効でない場合はすぐに戻り、Message の内容は空になる。

Socket::port ポート番号を得る

インターフェース

int16_t Socket::port() const;

説明ポート番号を返す。ポート番号が有効でない場合は-1 を返す。

Socket::isAvailable 有効であるか調べる

インターフェース

```
bool Socket::isValid() const;
```

説明

ソケットが有効な場合は true を返し、無効な場合には false を返す。これが false を返すときは Socket::send は失敗し、Socket::recv は即座に返る。

A.6 Dispatcher

ヘッダーファイル Dispatcher.h

概要

受信したメッセージを他のコンポーネントに配送する。

コンストラクタ

デフォルトコンストラクタを利用する。

メンバ関数

Dispatcher::attach DispatchReceiver を登録する。

インターフェース

```
void Dispatcher::attach(DispatchReceiver *receiver, std::string &msg);
```

説明

receiver を登録する。receiver は Message の最初の文字列が msg であるメッセージを受信する。

Dispatcher::detach DispatchReceiver の登録を解除する。

インターフェース

```
void Dispatcher::detach(DispatchReceiver *receiver, std::string &msg);
void Dispatcher::detach(DispatchReceiver *receiver);
```

説明

Message の最初の文字列が msg であるメッセージを受信するように設定された receiver の登録を解除する。msg が指定されていない場合には receiver のすべての登録が抹消される。

Dispatcher::dispatch インターフェース

```
void Dispatcher::dispatch(Message &msg);
```

説明

msg を登録された DispatchReceiver に配送する。

A.7 DispatchReceiver

ヘッダーファイル Dispatcher.h

概要

Dispatcher からの配送を受信するクラス。仮想クラスであり実際に利用するためにはメンバ関数 `void operator()(Message &msg)` を定義した派生クラスが必要となる。

コンストラクタ

デフォルトコンストラクタを利用する。

メンバ関数

DispatchReceiver::operator() メッセージの配送先

インターフェース

```
virtual void Dispatcher::operator()(Message &msg)
```

説明

msg を受信する。Dispatcher から呼び出されるメンバ関数である。

A.8 NotificationReceiver

ヘッダーファイル Notification.h

概要

オブジェクト内部の情報が変化した場合に呼び出される。このクラスは仮想クラスであり利用するためには `void operator()(std::vector<std::string> &msg)` を定義した派生クラスが必要となる。DeviceManager、ConnectionManager、SessionManager、RoutingEngine に登録することができる。また、どのような引数で呼び出されるからそれぞれのクラスの説明の**外部への報告**項目に説明されている。

コンストラクタ

デフォルトコンストラクタを利用する。

メンバ関数

NotificationReceiver::operator() メッセージの配送先

インターフェース

```
virtual void Dispatcher::operator()(std::vector<std::string> &msg)
```

説明

msg を受信する。msg の中身は変更があった項目についての情報であり、文字列の配列となっている。

付 録B　デバイスコントローラに関する処理を行うクラス

ここで説明するクラスはすべて名前空間 vin に属する。

B.1　DeviceController

ヘッダーファイル DeviceController.h

概要

デバイスコントローラのメインクラスであり、DeviceController::mainLoop を呼び出すことで実行を開始する。

コンストラクタ

```
DeviceController::DeviceController(Socket *sock, Peer, peer,
    DeviceDetector *det);
```

説明

外部にあるリソースマネージャなどと通信を行うための Socket とデバイスの接続状態の監視を行う DeviceDetector を指定して DeviceController を構築する。

メンバ関数

DeviceController::registerDC デバイスコントローラの登録

インターフェース

```
bool DeviceController::registerDC();
```

説明

デバイスコントローラをリソースマネージャに登録する。成功した場合に true を返し、失敗した場合には false を返す。

DeviceController::deleteDC デバイスコントローラの登録抹消

インターフェース

```
bool DeviceController::deleteDC();
```

説明

デバイスコントローラをリソースマネージャから抹消する。成功した場合に true を返し、失敗した場合には false を返す。

DeviceController::registerVVN バーチャルビデオノードを登録

インターフェース

```
void DeviceController::registerVVN(VirtualVideoNode *vvn);
```

説明

vvn をリソースマネージャに登録する。

DeviceController::deleteVVN バーチャルビデオノードの登録を抹消

インターフェース

```
void DeviceController::deleteVVN(VirtualVideoNode *vvn);
```

説明

vvn をリソースマネージャから抹消する。

DeviceController::attachVVN バーチャルビデオノードの接続

インターフェース

```
void DeviceController::attachVVN(VirtualVideoNode *vvn);
```

説明

vvn をデバイスコントローラに登録する。また DeviceController::registerVVN を呼び出す事でバーチャルビデオノードの登録要求をリソースマネージャに送信する。

DeviceController::detachVVN バーチャルビデオノードの取り外し

インターフェース

```
void DeviceController::detachVVN(VirtualVideoNode *vvn);
```

説明

vvn をデバイスコントローラから抹消する。また DeviceController::deleteVVN を呼び出す事でバーチャルビデオノードの登録要求をリソースマネージャに送信する。

DeviceController::registerGW ゲートウェイの登録

インターフェース

```
void DeviceController::registerGW(VirtualVideoNode *vvn,
    const NetAddress &netAddr, const NodeAddress &nodeAddr);
```

説明

vvn をゲートウェイとして登録する。接続先のネットワークは netAddr でバーチャルビデオノードは nodeAddr である。

DeviceController::deleteGW ゲートウェイの登録を抹消

インターフェース

```
void DeviceController::deleteGW(VirtualVideoNode *vvn);
```

説明

ゲートウェイとして登録された vvn を登録抹消する。

DeviceController::VVNs 接続しているバーチャルビデオノードの取得

インターフェース

```
std::list<VirtualVideoNode *> DeviceController::VVNs();
```

説明

デバイスコントローラに登録された VirtualVideoNode のリストを返す。

DeviceController::VVNof バーチャルビデオノードの取得

インターフェース

```
VirtualVideoNode * DeviceController::VVNof(NodeAddress &addr);
```

説明

デバイスコントローラに登録されたもので、addr というノードアドレスをもった VirtualVideoNode を返す。存在しない場合には 0 が返る。

DeviceController::connectTo コネクションを張る

インターフェース

```
bool DeviceController::connectTo(NodeAddress &src NodeAddress &dst);
```

説明

src というアドレスを持った VirtualVideoNode から dst に対してシグナリングを行いビデオデータの転送が可能になるようにする。src はこのオブジェクトで登録された VirtualVideoNode のアドレスでなければならない。

DeviceController::connectFrom コネクションを張る

インターフェース

```
bool DeviceController::connectFrom(NodeAddress &src NodeAddress &dst);
```

説明

src から dst というアドレスを持った VirtualVideoNode に対してシグナリングを行いビデオデータの転送が可能になるようにする。dst はこのオブジェクトで登録された VirtualVideoNode のアドレスでなければならない。

DeviceController::disconnectTo コネクションの切断

インターフェース

```
bool DeviceController::disconnectTo(NodeAddress &src NodeAddress &dst);
```

説明

src から dst というアドレスを持った VirtualVideoNode に対して張られているコネクションを切断する。src はこのオブジェクトで登録された VirtualVideoNode のアドレスでなければならない。

DeviceController::disconnectFrom コネクションの切断

インターフェース

```
bool DeviceController::disconnectFrom(NodeAddress &src NodeAddress &dst);
```

説明

src から dst というアドレスを持った VirtualVideoNode に対して張られているコネクションを切断する。dst はこのオブジェクトで登録された VirtualVideoNode のアドレスでなければならない。

DeviceController::detectDevice デバイスの検出

インターフェース

```
void DeviceController::detectDevice
```

説明

新たなデバイスが接続されたり取り外されたかチェックを行う。新たに接続されたデバイスはリソースマネージャに登録され取り外されたデバイスはリソースマネジャーに登録抹消要求を出す。

DeviceController::mainLoop メインループ

インターフェース

```
void DeviceController::mainLoop();
```

説明

デバイスコントローラとして動作を始めるために無限ループに入る。

B.2 DeviceDetectorInfo

ヘッダーファイル DeviceDetectorInfo.h

概要

新たに接続されたデバイスと取り外されたデバイスのリストである。

メンバ変数

std::list⟨VirtualVideoNode *⟩ DeviceDetectorInfo::added デバイスのリスト

説明

新たに接続されたデバイスのバーチャルビデオノードのリストである。

std::list⟨ VirtualVideoNode *⟩ DeviceDetectorInfo::removed デバイスのリスト

説明

取り外されたデバイスのバーチャルビデオノードのリストである。

B.3 DeviceDetector

ヘッダーファイル DeviceDetector.h

概要

デバイスの接続を監視するために利用される仮想クラスである。各ビデオネットワークに適合するようにサブクラスの実装を行い、DeviceController のコンストラクタを呼び出すときに引数として渡さなければならない。

コンストラクタ

デフォルトコンストラクタを利用する。

メンバ関数

DeviceDetector::checkVVNs デバイスのチェック

インターフェース

```
virtual DeviceDetectorInfo DeviceDetector::checkVVNs() = 0
```

説明

新たに接続されたデバイスと取り外されたデバイスを DeviceDetectorInfo 構造体に入れて返す。このメソッドは派生クラスで定義しなければならない。

B.4 VirtualVideoNode

ヘッダーファイル VirtualVideoNode.h

概要

ビデオ機器の抽象化を行う仮想クラスである。ビデオネットワークに適合するようにサブクラスで実装を行う必要がある。

コンストラクタ

```
virtual VirtualVideoNode(NodeAddress addr, Direction d = in);
```

説明 NodeAddress が addr でありビデオデータの向きが d である VirtualVideoNode を構築する。

メンバ関数

VirtualVideoNode::address アドレスを返す

インターフェース

```
virtual NodeAddress VirtualVideoNode::address() const;
```

説明

割り当てられた NodeAddress を返す。

VirtualVideoNode::setAddress アドレスを設定する

インターフェース

```
virtual void VirtualVideoNode::setAddress(NodeAddress &addr);
```

説明

addr にアドレスを変更する。

VirtualVideoNode::direction ビデオデータの方向を返す

インターフェース

```
virtual direction VirtualVideoNode::direction() const;
```

説明

ビデオデータの方向を返す。入力であれば VirtualVideoNode::in を返し出力であれば VirtualVideoNode::out を返す。

VirtualVideoNode::setDirection ビデオデータの方向を設定する

インターフェース

```
virtual void VirtualVideoNode::setDirection(Direction d);
```

説明

ビデオデータの方向を d に変更する。

VirtualVideoNode::connectTo コネクションを張る

インターフェース

```
virtual bool VirtualVideoNode::connectTo(NodeAddress &addr) = 0
```

説明

このバーチャルビデオノードに対応したビデオ機器から addr にビデオデータを転送するようにシグナリングを行う。

VirtualVideoNode::connectFrom コネクションを張る

インターフェース

```
virtual bool VirtualVideoNode::connectFrom(NodeAddress &addr);
```

説明 addr からビデオデータを転送するようにシグナリングを行う。

VirtualVideoNode::disconnectTo コネクションを切断する

インターフェース

```
virtual bool VirtualVideoNode::disconnectTo(NodeAddress &addr);
```

説明

addr からビデオデータを転送するようになっているコネクションの切断を行う。。

VirtualVideoNode::disconnectFrom コネクションを切断する

インターフェース

```
virtual bool VirtualVideoNode::disconnectFrom(NodeAddress &addr);
```

説明

addr にビデオデータを転送するようになっているコネクションの切断を行う。。

付 録C　リソースマネージャに関する処理を行うクラス

ここで説明するクラスはすべて名前空間 vin に属する。

C.1　DeviceControllerInfo

ヘッダーファイル VINInfo.h

概要

デバイスコントローラに関する情報を格納する構造体

メンバ変数

int ID; デバイスコントローラの ID

uint32_t addr; IP アドレス

int16_t port; ポート番号

説明

変数 ID にはデバイスコントローラの ID を格納する。ID は 0 以上の整数であり、-1 のときはこの構造体のデータが有効でない事を示す。また、複数のデバイスコントローラが同じ ID を持つことはない。addr と port にはデバイスコントローラに割り当てられた IP アドレスとポート番号が入る。

C.2　VirtualVideoNodeInfo

ヘッダーファイル VINInfo.h

概要

バーチャルビデオノードに関する情報を格納する構造体

メンバ変数

int dcID; 属しているデバイスコントローラの ID

NodeAddress addr; ノードアドレス

Direction direction; ビデオデータの方向

説明

dcID はバーチャルビデオノードの管理を行っているデバイスコントローラの ID である。addr はバーチャルビデオノードのノードアドレスであり、direction はバーチャルビデオノードが送受信できるビデオデータの向きである。addr が 0000:0000:0000:0000 である VirtualVideoNodeInfo は有効ではない。

C.3 ConnectionInfo

ヘッダーファイル VINInfo.h

概要

コネクションに関する情報を格納する構造体

メンバ変数

int ID; コネクション ID

NodeAddress src; 送信元ノードアドレス

NodeAddress dst; 受信先ノードアドレス

説明

ID はコネクションの番号であり、一つのリソースマネージャ内ではユニークになっている。src はこのコネクションによるビデオデータの転送の送信元であり dst は受信先となっている。ID は 0 以上の整数であり、ID が-1 の場合には無効な情報であることを表す。

C.4 SessionInfo

ヘッダーファイル VINInfo.h

概要

セッションに関する情報を格納する構造体

メンバ変数

int ID; セッション ID

NetAddress netSrc; 送信元ネットワークアドレス

NodeAddress src; 送信元ノードアドレス

NetAddress netDst; 受信先ネットワークアドレス

NodeAddress dst; 受信先ノードアドレス

int conID; ローカルのコネクション ID

> **説明** ID はセッションの番号である。この番号はユニークでありセッションを一意に示すことができる。netSrc と src はビデオデータの送信元のネットワークとバーチャルビデオノードのアドレスである。同様に netDst と dst はビデオデータの受信先のネットワークとバーチャルビデオノードのアドレスである。conID はこのセッションを構成するコネクションのうち、ローカルのものであるコネクション ID である。ID は 0 以上の整数であり、ID が-1 の場合には無効な情報であることを表す。

C.5 NeighborInfo

ヘッダーファイル VINInfo.h

概要

近隣のリソースマネージャについての情報を格納する構造体

メンバ変数

NetAddress netAddr; ネットワークアドレス

uint32_t ipAddr; IP アドレス

int16_t port; ポート番号

説明

netAddr は近隣のリソースマネージャに割り当てられたネットアドレスである。ipAddr と port は IP アドレスとポート番号であり、近隣のリソースマネージャと通信を行うのに利用される。netAddr が 0000 である NeighborInfo は有効ではない。

C.6 ResourceManager

ヘッダーファイル ResourceManager.h

概要

Socket からメッセージを読みだし Dispatcher に渡すという無限ループを実行する。

メンバ変数

DeviceManager *deviceManager; デバイスマネージャ

ConnectionManager *connectionManager; コネクションマネージャ

SessionManager *sessionManager; セッションマネージャ

RoutingEngine *routingEngine; ルーティングエンジン

説明

これらはそれぞれのサブコンポーネントのオブジェクトが生成されるときに設定される。

コンストラクタ

```
ResourceManager::ResourceManager(NetAddress &addr, Socket *sock);
```

説明

ネットワークアドレスが addr でネットワークとの間のインターフェースとなる Socket を sock として ResourceManager を構築する。

メンバ関数

ResourceManager::mainLoop リソースマネージャを開始する

インターフェース

```
void ResourceManager::mainLoop();
```

説明

リソースマネージャのメインループを開始する。

ResourceManager::socket 設定された Socket を得る

インターフェース

```
Socket *ResourceManager::socket();
```

説明

このリソースマネージャに登録された Socket を返す。

ResourceManager::address 設定された Socket を得る

インターフェース

```
NetAddress ResourceManager::address();
```

説明

このリソースマネージャに登録されたネットワークアドレスを返す。

ResourceManager::dispatcher() **インターフェース**

```
Dispatcher *ResourceManager::dispatcher();
```

このリソースマネージャが利用している Dispatcher を返す。メッセージを受信したい場合には、この Dispatcher に DispatchReceiver を登録すればよい。

C.7 DeviceManager

ヘッダーファイル DeviceManager.h

概要

デバイスマネージャのメインクラスである。ビデオネットワークに所属しているデバイスコントローラとバーチャルビデオノードに関する情報を管理する。

コンストラクタ

```
DeviceManager::DeviceManager(ResourceManager *rm);
```

説明
リソースマネージャのメインクラスである ResourceManager のインスタンスを引数として呼び出す。この DeviceManager は rm に所属する DeviceManager になる。

メンバ関数

DeviceManager::registerDC デバイスコントローラの登録

インターフェース

```
bool DeviceManager::registerDC(const DeviceControllerInfo &dci);
```

説明

デバイスコントローラを登録する。成功した場合に true を返し、失敗した場合には false を返す。

DeviceManager::deleteDC デバイスコントローラの登録の抹消

インターフェース

```
bool DeviceManager::deleteDC(const DeviceControllerInfo &dci);
```

説明

デバイスコントローラの登録を抹消する。このデバイスコントローラに所属するバーチャルビデオノードの登録も同時に抹消されるため、個々のバーチャルビデオノードの登録を抹消する必要はない。成功した場合に true を返し、失敗した場合には false を返す。

DeviceManager::registerVVN バーチャルビデオノードの登録

インターフェース

```
bool DeviceManager::registerVVN(VirtualVideoNodeInfo *vvni);
```

説明 vvni をバーチャルビデオノードとして登録する。このバーチャルビデオノードが所属するデバイスコントローラは予め登録されていなければならない。成功した場合に true を返し、失敗した場合には false を返す。

DeviceManager::deleteVVN バーチャルビデオノードの登録の抹消

インターフェース

```
bool DeviceManager::deleteVVN(VirtualVideoNode *vvn);
```

説明

vvni の登録を抹消する。成功した場合に true を返し、失敗した場合には false を返す。

DeviceManager::registerGW ビデオゲートウェイの登録

インターフェース

```
bool DeviceManager::registerGW(GatewayInfo *gwi);
```

説明

ゲートウェイを登録する。ゲートウェイとなるバーチャルビデオノードは予めデバイスマネージャに登録されていなければならない。成功した場合に true を返し、失敗した場合には false を返す。

DeviceManager::deleteGW ゲートウェイの登録の抹消

インターフェース

```
void DeviceManager::deleteGW(GatewayInfo *gwi);
```

説明

ゲートウェイの登録を抹消する。成功した場合に true を返し、失敗した場合には false を返す。

DeviceManager::searchDC デバイスコントローラの検索

インターフェース

```
DeviceControllerInfo DeviceManager::searchDC(int id) const;
DeviceControllerInfo DeviceManager::searchDC(
    uint32_t addr, int16_t port) const;
```

説明

デバイスコントローラ ID が id であるデバイスコントローラを検索しその結果を DeviceControllerInfo として返す。二つ目のメンバ関数を利用した場合には、IP アドレスが addr でポート番号が port であるデバイスコントローラの検索を行う。デバイスコントローラが見付からない場合には ID が-1 である DeviceControllerInfo を返す。

DeviceManager::searchVVN バーチャルビデオノードの検索

インターフェース

```
VirtualVideoNodeInfo DeviceManager::searchVVN(
    const NodeAddress &addr) const;
```

説明

ノードアドレスが addr であるバーチャルビデオノードを検索し、その結果を VirtualVideoNodeInfo として返す。指定したバーチャルビデオノードが存在しない場合には NodeAddress のデフォルト値のままである addr を持った VirtualVideoNodeInfo を返す。

DeviceManager::searchVVNs バーチャルビデオノードの検索

インターフェース

```
std::list<VirtualVideoNodeInfo> DeviceManager::searchVVNs(
    const DeviceControllerInfo &dci) const;
```

説明

dci で指定されているデバイスコントローラに接続されているバーチャルビデオノードについての情報を VirtualVideoNodeInfo のリストとして返す。

DeviceManager::searchGW ゲートウェイの検索

インターフェース

```
std::list<GatewayInfo> DeviceManager::searchGW(
    const NetAddress &addr, direction direct) const;
```

説明

addr で指定されたビデオネットワークに direct の方向で接続しているビデオゲートウェイに関する情報を GatewayInfo のリストとして返す。

DeviceManager::attachReceiver 通知オブジェクトの追加

インターフェース

```
void DeviceManager::attachReceiver(NotificationReceiver *receiver);
```

説明

receiver を内部状態変化時の呼び出しオブジェクトとして登録する。

DeviceManager::detachReceiver 通知オブジェクトの削除

インターフェース

```
void DeviceManager::detachReceiver(NotificationReceiver *receiver);
```

説明

receiver を内部状態変化時の呼び出しオブジェクトの登録から削除する。

DeviceManager の外部への報告

ATTACHEDDC (ID)
新たに ID が (ID) であるデバイスコントローラが登録された
DETACHEDDC (ID)
ID が (ID) であるデバイスコントローラの登録が抹消された
ATTACHEDVVN (NODEADDRESS)
ノードアドレスが (NODEADDRESS) であるバーチャルビデオノードが登録された
DETACHEDVVN (NODEADDRESS)
ノードアドレスが (NODEADDRESS) であるバーチャルビデオノードの登録が抹消された
ATTACHEDGW (NODEADDRESS)
ノードアドレスが (NODEADDRESS) であるビデオゲートウェイが登録された
DETACHEDGW (NODEADDRESS)
ノードアドレスが (NODEADDRESS) であるビデオゲートウェイの登録が抹消された

C.8 ConnectionManager

ヘッダーファイル ConnectionManager.h

概要

コネクションマネージャのメインクラスである。ビデオネットワーク内のコネクションに関する情報を管理する。

コンストラクタ

```
ConnectionManager::ConnectionManager(ResourceManager *rm);
```

説明

リソースマネージャのメインクラスである ResourceManager のインスタンスを引数として呼び出す。この ConnectionManager は rm に所属する ConnectionManager になる。

メンバ関数

ConnectionManager::connect コネクションを張る

インターフェース

```
bool ConnectionManager::connect(const NodeAddress &src,
    const NodeAddress &dst);
```

説明

src から dst にコネクションを張る。この関数が成功した場合には、コネクションの情報は既に登録されているため registeterConnection を呼び出す必要はない。成功した場合に true を返し、失敗した場合には false を返す。

ConnectionManager::disconnect コネクションを切断する

インターフェース

```
bool ConnectionManager::disconnect(int ID);
```

説明

コネクション ID が ID であるコネクションを切断する。この関数が成功した場合には、コネクションの情報は抹消済になるため deleteConnection を呼び出す必要はない。成功した場合に true を返し、失敗した場合には false を返す。

ConnectionManager::registerConnection コネクションを登録する

インターフェース

```
bool ConnectionManager::registerConnection(ConnectionInfo ci);
```

説明

ci で示されたコネクションを登録する。成功した場合に true を返し、失敗した場合には false を返す。

ConnectionManager::deleteConnection コネクションの登録を抹消する

インターフェース

```
bool ConnectionManager::deleteConnection(ConnectionInfo ci);
```

説明

ci で示されたコネクションの登録を抹消する。成功した場合に true を返し、失敗した場合には false を返す。

ConnectionManager::searchConnection コネクションの検索をする

インターフェース

```
ConnectionInfo ConnectionManager::searchConnection(
        const NodeAddress &src, const NodeAddress &dst) const ;
ConnectionInfo ConnectionManager::searchConnection(int id) const;
```

説明

src が送信者で dst が受信者であるコネクションを検索する。またはコネクション ID が id であるコネクションを検索する。

ConnectionManager::searchConnections コネクションの検索をする

インターフェース

```
std::list<ConnectionInfo> ConnectionManager::searchConnections(
    const NodeAddress &addr) const;
```

説明

addr が送信者か受信者であるコネクションを検索する。

ConnectionManager::attachReceiver 通知オブジェクトの追加

インターフェース

```
void ConnetionManager::attachReceiver(NotificationReceiver *receiver);
```

説明

receiver を内部状態変化時の呼び出しオブジェクトとして登録する。

ConnetionManager::detachReceiver 通知オブジェクトの削除

インターフェース

```
void ConnetionManager::detachReceiver(NotificationReceiver *receiver);
```

説明

receiver を内部状態変化時の呼び出しオブジェクトの登録から削除する。

ConnectionManager の外部への報告

CONNECTED (ID) (SRC) (DST)
ID が (ID) で送信者のノードアドレスが (SRC)、受信者のノードアドレスが (DST) のコネクションが登録された
DISCONNECTED (ID) (SRC) (DST)
ID が (ID) で送信者のノードアドレスが (SRC)、受信者のノードアドレスが (DST) のコネクションの登録が抹消された

C.9 SessionManager

ヘッダーファイル SessionManager.h

概要

セッションマネージャのメインクラスである。ビデオネットワーク内のセッションに関する情報を管理する。

コンストラクタ

```
SessionManager::SessionManager(ResourceManager *rm);
```

説明
リソースマネージャのメインクラスである ResourceManager のインスタンスを引数として呼び出す。この SessionManager は rm に所属する SessionManager になる。

メンバ関数

SessionManager::makeSession 新たなセッションの確立をする

インターフェース

```
bool SessionManager::makeSession(const NetAddress &netSrc,
    const NodeAddress &src, const NetAddress &netDst,
    const NodeAddress &dst);
```

説明

ネットアドレスが netSrc であるビデオネットワークに存在しノードアドレスが src であるバーチャルビデオノードから、ネットアドレスが netDst であるビデオネット

ワークに存在するノードアドレス dst のバーチャルビデオノードにセッションを張る。成功した場合に true を返し、失敗した場合には false を返す。

SessionManager::transitSession セッションの中継をする

インターフェース

```
bool SessionManager::transitSession(int ID, NodeAddress trans,
    const NetAddress &netSrc, const NodeAddress &src,
    const NetAddress &netDst, const NodeAddress &dst);
```

説明

ネットアドレスが netSrc であるビデオネットワークに存在しノードアドレスが src であるバーチャルビデオノードから、ネットアドレスが netDst であるビデオネットワークに存在するノードアドレス dst のバーチャルビデオノードにセッションを張る。この際にセッション番号は ID であり、直前のビデオネットワークからのビデオデータはノードアドレスが trans であるゲートウェイを介して転送される。成功した場合に true を返し、失敗した場合には false を返す。

SessionManager::cleanSession セッションの切断を行う

インターフェース

```
SessionManager::cleanSession(int ID);
```

説明セッション ID が ID であるセッションを切断する。成功した場合に true を返し、失敗した場合には false を返す。

SessionManager::registerSession セッションの登録を行う

インターフェース

```
bool SessionManager::registerSession(int ID, int connectionID,
    const NetAddress &netSrc, const NodeAddress &src
    const NetAddress &netDst, const NodeAddress &dst);
```

説明

セッション番号が ID であるセッションを登録する。このセッションを構成するローカルなコネクションの番号は connectionID で指定される。また、このセッションの送信元のネットワークアドレスとノードアドレスは netSrc と src であり、受信先のネットワークアドレスとノードアドレスは netDst と dst である。

SessionManager::deleteSession セッションの登録を抹消する

インターフェース

```
bool SessionManager::deleteSession(int ID);
```

説明

セッション番号が ID であるセッションを検索する。検索結果が返される。

SessionManager::searchSession セッションを検索する

インターフェース

```
std::list<SessionInfo> SessionManager::searchSessions(
    const NetAddress src, const NetAddress dst) const;
std::list<SessionInfo> SessionManager::searchSessions(int conID) const;
```

説明

送信側のネットワークアドレスが src、受信側が dst であるセッションを検索する。
またコネクション ID が conID であるコネクションが一部となっているセッション
を検索する。検索した結果を SessinoInfo のリストとして返す。

SessionManager::searchSession セッションを検索する

インターフェース

```
bool SessionManager::searchSession(int ID);
```

説明

セッション番号が ID であるセッションの検索を行う。

SessionManager::attachReceiver 通知オブジェクトの追加

インターフェース

```
void SessinManager::attachReceiver(NotificationReceiver *receiver);
```

説明

receiver を内部状態変化時の呼び出しオブジェクトとして登録する。

SessinManager::detachReceiver 通知オブジェクトの削除

インターフェース

```
void SessinManager::detachReceiver(NotificationReceiver *receiver);
```

説明

receiver を内部状態変化時の呼び出しオブジェクトの登録から削除する。

SessionManager の外部への報告

MADESESSION ID CONID
セッション ID が (ID) で、ローカルのコネクション ID が (CONID) であるセッションが確立された

CLEANEDSESSION ID CONID
セッション ID が (ID) で、ローカルのコネクション ID が (CONID) であるセッションが切断された

C.10 RoutingEngine

コンストラクタ

```
RoutingEngine::RoutingEngine(ResourceManager *rm);
```

メンバ関数

RoutingEngine::addNeighbor 近隣のネットワークの追加をする

インターフェース

```
void RoutingEngine::addNeighbor(const NeghborInfo &neighbor);
```

説明

neighbor を近隣のビデオネットワークのリソースマネージャとして登録する。

RoutingEngine::deleteNeighbor 近隣のネットワークの抹消をする

インターフェース

```
void RoutingEngine::deleteNeighbor(const NeghborInfo &neighbor);
```

説明

近隣のビデオネットワークのリソースマネージャとして登録されている neighbor を抹消する。

RoutingEngine::searchNeighbor 近隣のネットワークの検索をする

インターフェース

```
NeighborInfo RoutingEngine::searchNeighbor(const NetAddress &addr) const;
```

説明

ネットワークアドレスが addr である近隣のビデオネットワークを検索する。

RoutingEngine::isNeighbor 近隣のネットワークか調べる

インターフェース

```
bool RoutingEngine::isNeighbor(const NetAddress &addr);
```

説明

ネットワークアドレスが addr であるビデオネットワークが近隣であるか調べる。近隣であれば true が戻り、近隣でなければ false が戻る。

RoutingEngine::isAlive 動作中か調べる

インターフェース

```
bool RoutingEngine::isAlive(const NetAddress &addr);
```

説明

ネットワークアドレスが addr であるビデオネットワークが動作しているか調べる。動作していれば true が戻り、動作していないか近隣のビデオネットワークでなければ false が戻る。

RoutingEngine::searchNextNet 次ホップのネットワークを検索する

インターフェース

```
NetAddress RoutingEngine::searchNextNet(const NetAddress &addr) const;
```

説明

ネットワークアドレスが addr であるビデオネットワークにたどり着く経路で、ローカルのビデオネットワークの次に到達するビデオネットワークのネットワークアドレスを検索する。

RoutingEngine::attachReceiver 通知オブジェクトの追加

インターフェース

```
void RoutingEngine::attachReceiver(NotificationReceiver *receiver);
```

説明

receiver を内部状態変化時の呼び出しオブジェクトとして登録する。

RoutingEngine::detachReceiver 通知オブジェクトの削除

インターフェース

```
void RoutingEngine::detachReceiver(NotificationReceiver *receiver);
```

説明

receiver を内部状態変化時の呼び出しオブジェクトの登録から削除する。

RoutingEngine の外部への報告

ALIVE (NETADDRESS)
ネットワークアドレスが (NETADDRESS) であるビデオネットワークに存在するリソースマネージャの動作が新たに確認された

DEAD (NETADDRESS)
ネットワークアドレスが (NETADDRESS) であるビデオネットワークに存在するリソースマネージャの動作が確認できなくなった。

ATTACHEDNEIGHBOR (NETADDRESS)
ネットワークアドレスが (NETADDRESS) であるビデオネットワークが近隣のものとして設定された。

DETACHEDNEWNEIGHBOR (NETADDRESS)
ネットワークアドレスが (NETADDRESS) であるビデオネットワークが近隣のものではなくなった。

付 録D　メッセージ

ここではリソースマネージャとデバイスコントローラ、あるいはリソースマネージャ間で送受信されるメッセージをすべて示す。REQUEST で始まるメッセージへは ACCEPT か REJECT ではじまるメッセージが返信される。

デバイスコントローラの登録と抹消

REQUEST_ATTACHDC (IPADDRESS) (PORT)
(IPADDRESS): デバイスコントローラの IP アドレス
(PORT): デバイスコントローラのポート番号

ACCEPT_ATTACHDC (ID) (IPADDRESS) (PORT)
(ID): デバイスコントローラ ID
(IPADDRESS): デバイスコントローラの IP アドレス
(PORT): デバイスコントローラのポート番号

REJECT_ATTACHDC (IPADDRESS) (PORT)
(IPADDRESS): デバイスコントローラの IP アドレス
(PORT): デバイスコントローラのポート番号

REQUEST_DETACHDC (ID)
(ID): デバイスコントローラの ID

ACCEPT_DETACHDC (ID)
(ID): デバイスコントローラの ID

REJECT_DETACHDC (ID)
(ID): デバイスコントローラの ID

REQUEST_ATTACHVVN (ID) (NODEADDRESS) (DIRECTION)
(ID): デバイスコントローラ ID
(NODEADDRESS): バーチャルビデオノードのノードアドレス
(DIRECTION): ビデオデータの方向

ACCEPT_ATTACHVVN (ID) (NODEADDRESS) (DIRECTION)
(ID): デバイスコントローラ ID
(NODEADDRESS): バーチャルビデオノードのノードアドレス
(DIRECTION): ビデオデータの方向

REJECT_ATTACHVVN (ID) (NODEADDRESS) (DIRECTION)
(ID): デバイスコントローラ ID
(NODEADDRESS): バーチャルビデオノードのノードアドレス
(DIRECTION): ビデオデータの方向

REQUEST_DETACHVVN (ID) (NODEADDRESS)
(ID): デバイスコントローラ ID
(NODEADDRESS): バーチャルビデオノードのノードアドレス

ACCEPT_DETACHVVN (ID) (NODEADDRESS)
(ID): デバイスコントローラ ID
(NODEADDRESS): バーチャルビデオノードのノードアドレス

REJECT_DETACHVVN (ID) (NODEADDRESS)
(ID): デバイスコントローラ ID
(NODEADDRESS): バーチャルビデオノードのノードアドレス

REQUEST_ATTACHGW (ID) (NODEADDRESS) (NETPEER) (NODEPEER)
(ID): デバイスコントローラ ID
(NODEADDRESS): バーチャルビデオノードのノードアドレス
(NETPEER): 接続先のネットワークアドレス
(NODEPEER): 接続先のノードアドレス

ACCEPT_ATTACHGW (ID) (NODEADDRESS) (NETPEER) (NODEPEER)
(ID): デバイスコントローラ ID

(NODEADDRESS): バーチャルビデオノードのノードアドレス
(NETPEER): 接続先のネットワークアドレス
(NODEPEER): 接続先のノードアドレス

REJECT_ATTACHGW (ID) (NODEADDRESS) (NETPEER) (NODEPEER)
(ID): デバイスコントローラ ID
(NODEADDRESS): バーチャルビデオノードのノードアドレス
(NETPEER): 接続先のネットワークアドレス
(NODEPEER): 接続先のノードアドレス

REQUEST_DETACHGW (ID) (NODEADDRESS)
(ID): デバイスコントローラ ID
(NODEADDRESS): バーチャルビデオノードのノードアドレス
(NETPEER): 接続先のネットワークアドレス
(NODEPEER): 接続先のノードアドレス

ACCEPT_DETACHGW (ID) (NODEADDRESS)
(ID): デバイスコントローラ ID
(NODEADDRESS): バーチャルビデオノードのノードアドレス
(NETPEER): 接続先のネットワークアドレス
(NODEPEER): 接続先のノードアドレス

REJECT_DETACHGW (ID) (NODEADDRESS)
(ID): デバイスコントローラ ID
(NODEADDRESS): バーチャルビデオノードのノードアドレス
(NETPEER): 接続先のネットワークアドレス
(NODEPEER): 接続先のノードアドレス

コネクション要求と切断要求

REQUEST_CONNECTTO (NODESRC) (NODEDST)
(NODESRC):
(NODEDST): デバイスコントローラのポート番号

ACCEPT_CONNECTTO (NODESRC) (NODEDST)
(ID): デバイスコントローラ ID
(IPADDRESS): デバイスコントローラの IP アドレス

(PORT): デバイスコントローラのポート番号

REJECT_CONNECTTO (NODESRC) (NODEDST)
(IPADDRESS): デバイスコントローラの IP アドレス
(PORT): デバイスコントローラのポート番号

REQUEST_CONNECTFROM (NODESRC) (NODEDST)
(NODESRC):
(NODEDST): デバイスコントローラのポート番号

ACCEPT_CONNECTFROM (NODESRC) (NODEDST)
(ID): デバイスコントローラ ID
(IPADDRESS): デバイスコントローラの IP アドレス
(PORT): デバイスコントローラのポート番号

REJECT_CONNECTFROM (NODESRC) (NODEDST)
(IPADDRESS): デバイスコントローラの IP アドレス
(PORT): デバイスコントローラのポート番号

REQUEST_DISCONNECTTO (NODESRC) (NODEDST)
(NODESRC): 送信側のノードアドレス
(NODEDST): 受信側のノードアドレス

ACCEPT_DISCONNECTTO (NODESRC) (NODEDST)
(NODESRC): 送信側のノードアドレス
(NODEDST): 受信側のノードアドレス

REJECT_DISCONNECTTO (NODESRC) (NODEDST)
(NODESRC): 送信側のノードアドレス
(NODEDST): 受信側のノードアドレス

REQUEST_DISCONNECTFROM (NODESRC) (NODEDST)
(NODESRC): 送信側のノードアドレス
(NODEDST): 受信側のノードアドレス

ACCEPT_DISONNECTFROM (NODESRC) (NODEDST)
(NODESRC): 送信側のノードアドレス
(NODEDST): 受信側のノードアドレス

REJECT_DISONNECTFROM (NODESRC) (NODEDST)
(NODESRC): 送信側のノードアドレス
(NODEDST): 受信側のノードアドレス

セッション要求と切断要求

REQUEST_MAKESESSION (ID) (NETSRC) (NODESRC) (NETDST) (NODEDST) (LOCALSRC)
(ID): セッション ID
(NETSRC): 送信側のネットワークアドレス
(NODESRC): 送信側のノードアドレス
(NETDST): 送信側のネットワークアドレス
(NODEDST): 受信側のノードアドレス　(LOCALSRC): ローカルなビデオネットワークにおけるビデオデータの送信元

ACCEPT_MAKESESSION (ID)
(ID): セッション ID

REJECT_MAKESESSION (ID)
(ID): セッション ID

リソースマネージャの生存確認

ALIVE (NETADDRESS)
(NETADDRESS): リソースマネージャのネットワークアドレス

経路情報の交換

ROUTEINFORMATION (NETADDRESS) {(NETDST) (METRIC)}*
(NETADDRESS): ネットワークアドレス
(NETDST): 宛先のネットワークアドレス
(METRIC): 宛先のネットワークまでの距離

リソースマネージャ情報の要求

REQUEST_NETADDRESS (ID)
(ID): デバイスコントローラ ID

ACCEPT_NETADDRESS (ID) (NETADDRESS)
(ID): デバイスコントローラ ID
(NETADDRESS): リソースマネージャのネットワークアドレス

REJECT_NETADDRESS (ID)
(ID): デバイスコントローラ ID

参考文献

[1] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications”, RFC1889, January 1996.

[2] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, and S. Jamin, “Resource ReSerVation Protocol(RSVP) – Version 1 Functional Specification”, RFC2205, September 1997.

[3] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, “”, RFC3261, June 2002.

[4] HAVi, Inc. “HAVi SPECIFICATION Version 1.1”, May 15,2001.

[5] C. Hedrick, “Routing Information Protocol”, RFC1058, June 1988.

[6] Y. Rekhter, and T. Li “A Border Gateway Protocol 4 (BGP-4)”, RFC1654, March 1995.

[7] J. Moy, “OSPF Version 2”, RFC2328, April 1998.

[8] D. Oran, Editor, “OSI IS-IS Intra-domain Routing Protocol”, RFC1142, February 1990.

[9] Yasuo Tan, “Scaling up IEEE1394 DV network to an enterprise video LAN with ATM technology”, In Digest of technical papers of IEEE International Conference on Consumer Electronics 1998, 1998.

[10] Yasuo Tan, “Scalable digital video network with IEEE1394 and ATM”, In International Distributed Conference 1999, 1999.

[11] Yasuo Tan, Takashi Nomura, Hirofumi Tamori and Kouji Koshiba, “Plug and Play Campus Digital Video Network with IEEE1394 and ATM”, In Proceedings of the International Conference on Computer Communication 1999, 1999.

[12] 倉岡貴志, 丹康雄, “AV 系ネットワークシステムにおける資源管理法に関する一手法”, 第 59 回情報処理学会全国大会, 情報処理学会, 9, 1999.

[13] WIDE project. “DV Stream on IEEE1394 Encapsulated into IP”, http://www.sfc.wide.ad.jp/DVTS/

[14] Yoshiki Makino, and Yasuo Tan, “A design of Video Internetworking System using resource managers”, 情報処理学会 第5回高品質インターネット研究発表会, pp. 5-20, 2002.

[15] 牧野 義樹, 丹康雄, “ビデオインターネットワーキングアーキテクチャの提案”, 情報処理学会 情報家電コンピューティング研究グループ第4回研究会