卒業論文　　2008年度

All-IP Computerにおける入出力デバイスの研究

慶應義塾大学 環境情報学部

氏名：六田 佳祐

指導教員

慶應義塾大学 環境情報学部

村井 純

徳田 英幸

中村 修

楠本 博之

高汐 一紀

三次 仁

植原 啓介

重近 範行

中澤 仁

Rodney D. Van Meter III

平成21年2月10日

卒業論文要旨 2008 年度

All-IP Computer における入出力デバイスの研究

本研究では，All-IP Computer における入出力デバイスを実現する手法について述べる．本研究における All-IP Computer とは，コンピュータの持つ内部バスで扱うデバイス情報を個別の機器接続規格ではなく，IP パケットとして送受信するコンピュータである．All-IP Computer はデバイス情報を IP パケットとして送受信するため，IP ネットワークへの接続性が保たれればデバイス個々の物理的制約に捕らわれることなくコンピュータ環境を構成することが可能である．

コンピュータを構成するデバイスは，各々の接続規格に基づいて設計・開発されている．接続規格には PCI などの内部バスや，USB や IEEE1394 などの外部バスが存在する．それら接続規格を用いて，入出力デバイスや記憶デバイスなどはコンピュータに接続されている．しかし，それら接続規格には物理的距離，1 台のコンピュータ上におけるバス数およびデバイス数，構成の動的変更，異なるコンピュータ間におけるデバイス共有の制限など様々な制約がある．

本研究ではディスプレイ出力デバイスであるフレームバッファに着目し，フレームバッファの描画データ管理を IP ネットワークを経由して行うことで，IP ネットワークに接続したディスプレイデバイスを実現する．本研究では IP ディスプレイデバイスの実装を，All-IP Computer 環境における” Display over IP” として Linux Kernel 2.6 上で行った．フレームバッファメモリの内容を IP ネットワークを経由して転送することで，遠隔コンピュータのディスプレイ出力を利用することを可能にした．

IP ネットワークの利点として，多様なデータを世界中と送受信できることがある．しかし，IP ネットワークの欠点として，デバイスの接続を目的としたバスよりも低速であることや，遅延やパケットロスなどが生じることがある．Display over IP は，あるホストの持つフレームバッファメモリの内容を別のホストに転送し，その内容を表示することを利用上支障のない速度で実現した．評価の上で，解像度 XGA 16 ビットカラーの環境において毎秒 20 フレーム程度の動作を実現した．実装は，X Window System とフレームバッファコンソールの双方で利用可能である．また，本実装は以前に送信したデータを記録し送信時に送信データを過去のデータと比較することで，不要なデータの再送信を防止しネットワーク帯域消費量を抑制する機構を持つ．

キーワード
1．All-IP Computer，2．ディスプレイ，3．フレームバッファ，4. デバイス機能要件

慶應義塾大学 環境情報学部

六田 佳祐

Abstract of Bachelor's Thesis

Academic Year 2008

Input and Output Devices in an All-IP Computer Environment

This research discusses input and output devices on an All-IP Computer environment. The All-IP Computer is a computer that communicates with I/O devices using Internet Protocol (IP), rather than a specific physical bus, such as USB or SCSI. Since device function information is transported as IP packets, there are no physical placement limitations for devices as long as there is connectivity to the IP network.

The devices that constitute a computer are designed and developed in accordance with a variety of standards. There are internal buses such as PCI, and external buses such as USB and IEEE1394. These standards are used for connecting various devices, such as input/output devices and storage devices. However, they have limitations such as: the physical distance of connections, the number of buses and devices on a computer, the difficulty of computer environment configuration, and the difficulty of sharing devices among different computers.

This research targets a framebuffer, a display output device, giving it the ability to transport its data over an IP network. The prototype implementation is done on the Linux Kernel 2.6. By transporting the contents of framebuffer memory, the implementation realizes an IP-connected display output from a distant computer.

An advantage of using an IP network is the fact that the data can be transmitted to the entire world, as long as there is connectivity. However, there are disadvantages to using IP networks, such as their low speed compared to device buses, and other characteristics of IP networks such as delays and packet losses. The implementation is able to transfer the framebuffer memory data from a host to another with usable speed, realizing the "Display over IP" in an All-IP Computer environment, achieving frame rates of approximately 20 frames per second for XGA resolution with 16-bit colors. The implementation functions for both X Window System and framebuffer console use, decreasing unnecessary network traffic by comparing cached data to the actual send data and preventing from re-sending unchanged regions of the framebuffer.

Keywords :
1. All-IP Computer, 2. Display, 3. Framebuffer, 4. Device Requirements

Keio University , Faculty of Environment and Information Studies

Keisuke Muda

目 次

図 目 次

表 目 次

第1章　序章

1.1　はじめに

近年，企業・大学・家庭など様々な場面で，コンピュータネットワークを用いた多様な情報処理が行われている．ユーザの持つコンピュータ環境は様々であり，また，そのコンピュータで扱われる情報も多様である．このようにコンピュータは多種多様な用途に用いられているが，今日ではコンピュータの中でも，ノート型コンピュータや携帯電話など，携帯可能なコンピュータの普及がめざましい．この背景には，移動中や外出先などデスクトップ型コンピュータのない場所でも，ユーザがコンピュータとその中のデータを利用することが一般化している点が挙げられる．

しかし，携帯可能なコンピュータを用いても，ユーザがコンピュータ環境やその中に保存されたデータを自由に持ち運べる状態になっているとは言い難い．現在のコンピュータアーキテクチャでは，周辺機器とコンピュータ本体は物理的に接続されている．そして，ユーザの持つデータやそれを記憶するコンピュータの構成部品も，同様にコンピュータに直接接続されている．つまり，現在のコンピュータアーキテクチャにおいてコンピュータ・構成部品・データはすべて 1 つの” 箱” として構成されている．

現在のコンピュータは，大別して 2 つのネットワークを持っている．1 つは，コンピュータという箱同士の情報交換に用いられる IP ネットワークである．そしてもう 1 つは，コンピュータという箱内外のデバイス同士を接続する機構である．コンピュータは個々の機能を持つデバイスの集合体であり，それは一種のネットワークである．

近年，これらネットワークの高速化がめざましい．通信網である IP ネットワークにおいては，10G Ethernet やそれを越える 40G Ethernet などの超高速ネットワークの研究開発が進んでいる．また，コンピュータ内におけるデバイス間ネットワークでは，PCI Express や USB 3.0 といった，従来よりも高速な規格が開発されつつある．

コンピュータ内外のネットワークが高速化している今日，両ネットワークを，1 つの統合されたネットワークとして扱うことが可能となりつつある．例えば，iSCSI という規格では，従来から存在する SCSI デバイスの処理命令を，IP ネットワークを介して行っている．IP ネットワークを利用してデバイスを論理的に接続することにより，コンピュータの箱内外のネットワークを分断する物理的な壁が取り除かれる．

ユーザのコンピュータ利用の目的は様々であり，目的に合わせたコンピュータの構成部品の接続・切断を動的に実現するコンピュータが実現したとき，ユーザのコンピュータ利用の自由度を向上させることが可能となる．このような機能を実現するコンピュータを本研究では”All-IP Computer” と呼ぶ．All-IP Computer における構成要素にはリソース管理や構成要素のネットワークへの接続といったものがあり，各構成要素を IP

ネットワークを介して論理的に接続・切断する．論理的接続・切断による構成要素の利用は，コンピュータとデバイスの間の物理的制約を解消する．All-IP Computer は，ユーザの求めるコンピュータ利用環境を，IP ネットワーク上で自由に構成・提供するコンピュータである．

All-IP Computer の実現にあたり，コンピュータを構成する構成要素の IP ネットワークへの接続が必要である．本研究ではそのようなネットワークに接続された構成要素を”Device over IP” と呼称する．Device over IP の中でも，ユーザのコンピュータ利用に多大な影響を及ぼすものが，入出力機器である．入力・出力の際に発生する遅延などはユーザのコンピュータ利用に弊害をもたらすが，IP ネットワークは構成要素専用のバスと比較して低速で，遅延やパケットロスが発生しやすいものである．遅延を少なくし，且つ既存のコンピュータ環境と同等の性能を発揮することが，All-IP Computer の実現には必要不可欠である．

1.2 目的

本研究は，All-IP Computer の実現に必要となる出力デバイスであるディスプレイを，IP ネットワークを経由して利用することを実現する．また，All-IP Computer を利用するにあたり必要な入力デバイスの実現についても触れる．両者を併せて All-IP Computer を想定した入出力デバイスを実現し，OS における Device over IP を実現する．

デバイスをネットワークで利用することは，単にデバイスを IP ネットワークに接続するだけでは実現しない．IP ネットワークにはデバイス専用に設計されたバスにはない性質があり，その特性を乗り越えなくてはデバイスを IP ネットワーク経由で利用することはできないからである．各デバイスには優先されるべき機能と，妥協点を見出すことが可能な機能がある．さらに，Device over IP の実現にはデバイスの IP ネットワークへの接続だけでなく，OS におけるデバイスの抽象化など他の課題も解決する必要がある．本研究ではディスプレイ出力デバイスを IP ネットワークに接続し，ディスプレイ出力デバイスの実現に必要な機能の決定と，ディスプレイ出力デバイスに特化した IP ネットワーク化の手法について検討する．

1.3 論文の構成

本論文は 9 章から構成される．第 2 章では，本研究におけるコンピュータの概念である All-IP Computer について述べる．第 3 章では，ネットワークを経由したデバイスの利用である Device over IP について述べる．第 4 章では，Device over IP の OS におけるハードウェア利用と抽象化について述べる．第 5 章ではディスプレイデバイスとして要求される事柄について述べる．第 6 章ではシステムを設計し，第 7 章で実装について述べる．そして，第 8 章で評価を行う．

第2章　All-IP Computer

本研究は，現在のコンピュータアーキテクチャとは異なる，All-IP Computer におけるディスプレイ環境の実現を目標とする．本章ではまず，All-IP Computer の概要を述べる．

2.1　背景

今日，ユーザは多様な情報処理を様々な環境で行っている．1 人のユーザが利用できるコンピュータ環境には，デスクトップ型コンピュータ，ノートブック型コンピュータ，小型端末などがあり，それらの設置場所も家庭や職場など多様である．それらコンピュータは，管理者の運用方針や利用しているコンピュータ毎に異なる構成を持っている．各ユーザは利用可能な異なる環境の上で，各ユーザが必要とするアプリケーションを利用した作業を行っているのが現状である．

コンピュータ環境の差異は，ユーザがコンピュータを利用する上で時として妨げとなる．全てのコンピュータ上で同一の環境，同一のアプリケーションが利用できるとは限らない．また，ファイルや個人の連絡先，コンピュータの設定など，ユーザが作成したデータは各コンピュータに個別に保存されることが一般的である．各コンピュータに個別に保存されたデータを複数環境で利用するためには，ユーザによってそのデータの所存や版数を確認し，状況に適した同期を取る必要がある．この作業は誤りや予想外の障害，利用者のミスなどによって失敗したり忘れたりすることもあり，複数環境でデータを扱うことが煩雑であることが分かる．

そこで近年の流れとして，Web 上にユーザ環境やデータを全て保存し，各コンピュータからは Web ブラウザを用いてアクセスするという方法が多く開発されている．例えば Google 社の Google Apps [1] や ThinkFree 社の ThinkFree Online [2] では，アプリケーションは全て Web ブラウザ上で稼働する．また，ユーザの持つデータやアプリケーションの設定は，全てサービス提供者のサーバに保管される．アプリケーションだけではなく，StartForce [3] など Web ブラウザでデスクトップ環境を操作し，そのデスクトップ環境上で提供されるアプリケーションを利用するサービスもある．これらの Web サービスは，ユーザに共通のコンピュータ利用環境の提供と，データの共有の両方を実現している．

一方で，ソフトウェアだけではなく，ハードウェアをネットワーク経由で利用する手法については，多くの技術開発が行われてきた．ハードウェアのネットワーク共有の例として，記憶装置やプリンタを対象としたものがある．Samba [4] などのネットワーク

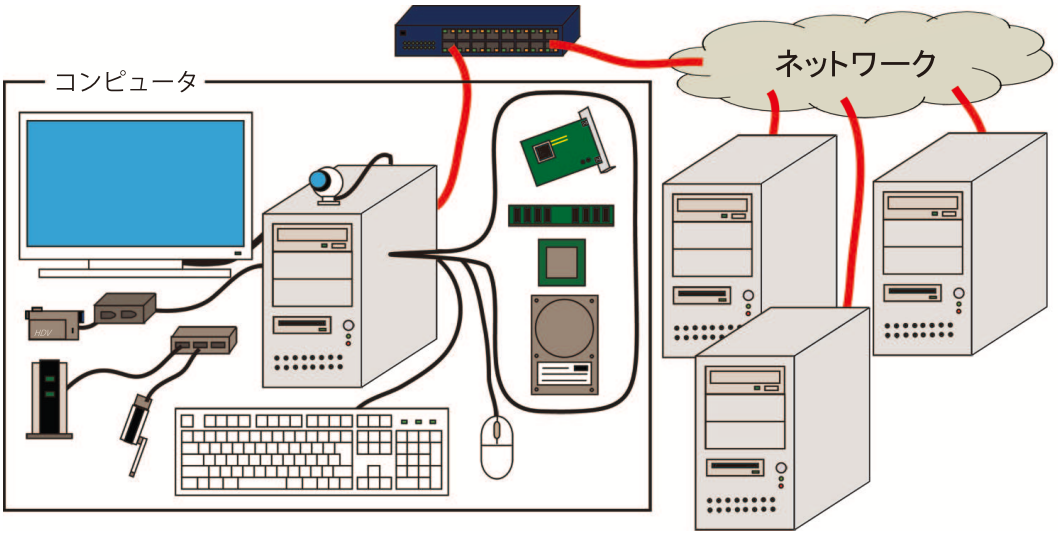

図 2.1: 現在のコンピュータアーキテクチャ

サービスを実行することで，コンピュータの持つハードウェアを他のコンピュータから利用できるようにする手法が一般的である．また，プリンタや USB デバイスなどをネットワーク経由で利用できるようにする” デバイスサーバ” 等の製品も，近年では一般的になってきた．プリンタ機能のみを対象としたデバイスサーバには，前述の Samba と同等の機構を持っているものがある．また，プリンタだけでなく，スキャナ等 USB デバイスを一般的に扱えるデバイスサーバには，USB の命令自体をネットワーク経由で送受信するものもある．両者の手法は異なるが，ハードウェアをネットワーク経由で利用することは現在でも試みられていることであり，今後も利用されると考えられる．

このように，ソフトウェア，データ，ハードウェアはすべてネットワークを介して利用することが一般的となりつつある．本研究における All-IP Computer はこのような背景を前提に，コンピュータを構成する要素を IP ネットワークを利用して論理的に相互接続するものである．本章では，現在のコンピュータアーキテクチャと All-IP Computer アーキテクチャを説明し，その利用シナリオについて述べる．

2.2 現在のコンピュータアーキテクチャ

現在のコンピュータアーキテクチャを図 2.1 に示す．コンピュータ 1 台は様々な” デバイス” で構成されている．本研究におけるデバイスとは，中央演算装置・記憶装置・入出力装置など，コンピュータを構成する部品のことである．デバイスはそれぞれの役割を持って，ユーザがコンピュータに求める機能を提供している．

現在のコンピュータアーキテクチャにおいて，コンピュータは 2 つの” ネットワーク” を持っている．1 つ目は，通信網としての IP ネットワークである．ユーザの扱う情報は，この IP ネットワーク上で送受信されている．2 つ目は，コンピュータの持つデバイス同士を接続している専用の伝送路である．

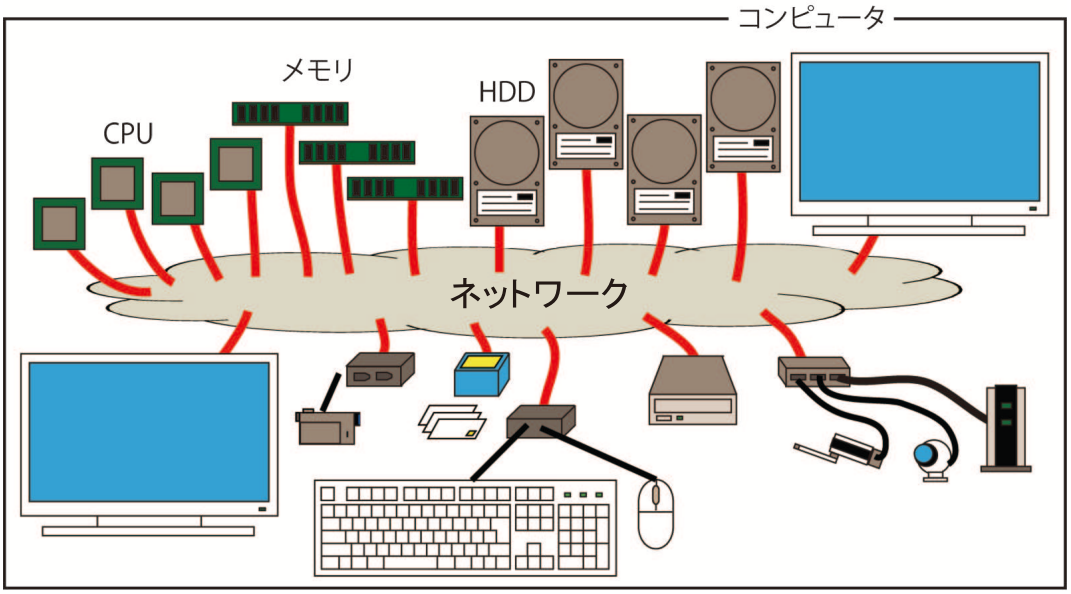

図 2.2: All-IP Computer アーキテクチャ

現在のコンピュータアーキテクチャにおける IP ネットワークとは，コンピュータの相互接続基盤である．一方で，デバイス間の処理はそれぞれのコンピュータ内で完結しており，ユーザのサービス利用に用いられる IP ネットワークには影響しない．2 つのネットワークは独立しており，互いに影響を与えない．

2.3 All-IP Computer の概念

All-IP Computer で実現する新しいコンピュータアーキテクチャを図 2.2 に示す．現在のコンピュータアーキテクチャにおけるコンピュータとは，コンピュータの構成要素であるデバイスを物理的に集約したものである．また，IP ネットワークとはコンピュータという箱の相互接続基盤である．All-IP Computer においては，デバイスのネットワークであるバス上で交換する情報も，ユーザの取り扱う情報も，すべて同一の IP ネットワーク上で送受信される．

現在のコンピュータアーキテクチャに利用されているデバイスは専用のバスに接続され，相互にデータの交換を行うことを前提として設計されている．しかし，コンピュータバスを前提とする機器接続規格は IP ネットワークで発生するような遅延や情報損失を想定して設計されておらず，デバイスが IP ネットワークに接続される All-IP Computer では，低遅延，広帯域な超高速ネットワークと，そのような IP ネットワークの影響を吸収する新しいデバイス間通信の機構が必要である．

現在利用されている超高速ネットワークの例として，10G Ethernet が挙げられる．近年では，この 10G Ethernet を越える高速なネットワークも登場しつつある．All-IP Computer は，このような超高速ネットワークと本研究にて実現するデバイス共有機構

があってこそ実現できる，新しいコンピュータである．

2.4 利用シナリオ

All-IP Computer の実現により，ユーザは自由かつ動的に利用用途に応じたコンピュータ環境を構築することが可能となる．また，デバイスが IP ネットワークに接続されることで，デバイスやコンピュータ資源を複数ユーザで共有することも可能である．

例えば，複数ユーザが共同作業を行っていると仮定する．All-IP Computer 上で作成されている資料を，各ユーザが利用している画面に出力する．各ユーザの持つ入力機器によって同一の All-IP Computer を操作することで，共同作業が効率化される．

次に，計算シミュレーションなどユーザが豊富なコンピュータ資源を必要とする状況を想定する．その際，All-IP Computer はネットワークを介してスーパーコンピュータのコンピュータ資源を利用し，自身の計算能力を一時的に増強することができる．

さらに，映画鑑賞時にも All-IP Computer を利用することで鑑賞に適したコンピュータ環境を利用できる．映画鑑賞に適した環境としてサラウンド音響システム，単純な入力デバイスとしてリモコンなどがある．これらのコンピュータ環境を All-IP Computer は動的に選定しユーザに提供することができる．

それぞれの処理は，コンピュータの利用シーンとしては異なるものである．しかし，ここで利用されているコンピュータは常に同一の All-IP Computer であり，動的な拡張や構成によって，異なるコンピュータ環境を構築している．自由かつ動的なコンピュータ環境の構築を，All-IP Computer は可能とする．

2.5 関連研究

本節では，All-IP Computer アーキテクチャの関連研究について述べる．

2.5.1 Desk Area Network

Desk Area Network (DAN) [5] は，コンピュータの持つすべてのデバイス接続を Asynchronous Transfer Mode (ATM) 網を介して送受信する．コンピュータのデバイスをネットワーク経由で利用するという概念は，All-IP Computer と同様である．DAN の実装例として，マサチューセッツ工科大学コンピュータ科学研究所による ViewStation [6] がある．

All-IP Computer と異なる点として，DAN ではシステムにおける境界が定義されていることがある．“同一ネットワーク” として定義された範囲であれば，ネットワークを介してデバイスの接続を行う．しかし，同一ネットワーク外のデバイスとは接続しないという点が，DAN の特徴である．デバイスの発見と接続の処理や認証などを同一ネットワーク内に限定することは，管理やセキュリティの面で有利である．また，各デバイスは同一ネットワークに存在することから，アドレス等の管理は容易であり，そ

のアドレス割り当てなどのポリシーもネットワーク管理者が任意に決定できる．更に，インターネットを経由しないため外部者による盗聴などの危険が少ない．そして，同一ネットワーク上に存在するデバイスをシステムが把握することが比較的容易であることから，登録されたデバイス全ての存在を把握・監視することが可能であるとされている．

しかし，デバイスの利用を同一ネットワーク内の接続に制限してしまうため，IP ネットワークの規模性を有効活用することは出来ない．DAN と All-IP Computer の違いは，All-IP Computer では物理的な場所の制約に囚われず，IP ネットワーク上で到達性のあるすべてのデバイスを利用することができる点である．

2.5.2 Netstation

Netstation [7] は，南カリフォルニア大学情報科学研究所で行われていた研究である．Netstation は All-IP Computer と同様に，デバイスを IP ネットワークを介して接続する．

Netstation は DAN とは異なり，IP を用いてデバイスの接続を行い，デバイス利用が可能となるネットワーク範囲の制限等も無い．研究においてデバイスの発見の難しさやセキュリティについて述べ，問題提起がされているが，その解決法は定義されていない．

2.5.3 Plan 9

Plan 9 [8] は，ベル研究所で開発されたオペレーティングシステムの一種である．元来の UNIX の開発に携わったエンジニアが多数参加しており，UNIX の設計上の問題を修正する形で設計されている．その中でデバイスのネットワーク利用について触れており，All-IP Computer に通ずる部分がある．

Plan 9 は，多数のコンピュータ資源や利用者を集中管理することに主眼をおいている．まず，コンピュータを構成するコンピュータ資源は，具体的な規格に関わらず種類や目的に応じて共通の規則で命名されている．例えば，Plan 9 では種類に関係なく `/dev/cons` というファイルとしてコンソールは抽象化されており，日付を表示するコマンドはその種類に関わらず`/bin/date` を実行することで結果を取得できる．特定の空間での名前を，グローバルインターネット上へ対応するようにマッピングすることが，Plan 9 における名前管理の特徴である．

次に，Plan 9 におけるファイルは，一箇所のファイルサーバで集中管理されている．独立したファイルサーバにファイルを集中して保存することで，管理コストを軽減することが可能である．ファイルにはユーザが作成するファイルと OS を構成するファイルの 2 種類があるが，Plan 9 では OS を構成するシステムファイルも共通して一箇所で集中管理する。このことにより，OS ソフトウェアの更新なども集中管理することが可能になり，設定変更などもその OS を参照するすべての Plan 9 システムに適用することができる．

更に，ファイルだけでなく，ユーザも一箇所で集中管理できることも，Plan 9 の特徴である．ユーザを管理するデータベースも一台で統一管理されており，Plan 9 システムを利用できるユーザであれば，どのシステムであっても利用できる．

集中管理をすることは，利用者・管理者の双方にとって管理が容易になるという利点がある．一方で，特定の一台のサーバにすべての情報を管理することは単一障害点を作成することとなり，規模性や耐故障性の面では不利な点もある．このような Plan 9 の概念は本研究における All-IP Computer にも関連する要素であり，今後の検討が必要な点でもある．

2.6 本研究の位置付け

All-IP Computer を構成する要素は，大きく分けて 3 つある．まず，All-IP Computer を構成するデバイス群がある．コンピュータを構成するためには，そのコンピュータの CPU，メモリ，二次記憶装置，入出力機器などのデバイスが必要である．デバイスを IP 化するためには，物理的に隔離されたバスプロトコルで動作するデバイスを，汎用的・論理的な IP ネットワーク上で動作させる必要がある．そのために解決しなくてはならない点は多く，例えばデバイスの IP ネットワークへの接続，遅延などの考慮，信頼性の提供などがある．

次に，デバイスやユーザアプリケーションを動作させるための OS がある．デバイスを構成するだけでは，コンピュータはその役割を果たさない．ユーザからの要求に合わせた動作をするアプリケーションや，そのアプリケーションの動作を管理する OS が必要である．また，デバイスを IP ネットワークへ接続するためには，そのデバイスや接続先で IP ネットワークへの到達性を管理する仕組みも必要である．ユーザからの要求やコンピュータ自身の構成を管理するために，OS は必要である．

また，All-IP Computer を構成する要素を管理・発見・認証するためのサービス発見機構も必要である．コンピュータの構成要素が IP ネットワークへ接続しても，その構成要素をどのように接続するかといった点や，どの構成要素がどのような状況下にあるのかを管理しなくては，実際にその構成要素を利用することはできない．All-IP Computer アーキテクチャでは，構成要素の状態を把握する Rendezvous Manager (RM) が IP ネットワーク上に設置され，構成要素が起動すると自身のデバイス名や IP アドレスといった情報を RM に登録する．RM は All-IP Computer からの要求に応じ，必要なデバイスの存在を All-IP Computer に通知する．そして，All-IP Computer とデバイスが接続されれば，RM はそのデバイスが利用中であることを記憶し，別のコンピュータからの利用要求があっても利用中である旨の応答をする．そして，デバイスの利用終了と共に，再度デバイスが利用可能な状態とする．この動作を繰り返すことで，All-IP Computer を構成する要素の管理・発見・認証を行う．

上記の構成要素を，ネットワークという統一されたバスを利用して相互接続し実現するものが，All-IP Computer である．本研究は All-IP Computer を構成する要素の内，デバイスのネットワーク化を対象とする．IP ネットワーク上でデバイスと OS を接続するためにはコンピュータ専用に開発されたバスとは異なり，様々な解決すべき課題

がある．例えば，OS がデバイスを利用するために必要な抽象化を行うこと，デバイスの持つデータを IP ネットワークへ送信するためのデータ形式や方式の定義，IP ネットワーク上のデバイスを識別・利用すること，IP ネットワークにおける特性への対応等が挙げられる．All-IP Computer におけるデバイスの IP 化において，必要となる要件の整理・検証を行うことが本研究の位置付けである．

第3章　Device over IP

All-IP Computer は，ネットワーク上に存在するデバイスによって構成された 1 台のコンピュータである．All-IP Computer を構成する IP ネットワーク上のデバイスを，本研究では”Device over IP”と呼ぶ．

3.1　Device over IP とは

Device over IP とは，通常はコンピュータ本体に何らかのケーブルを接続することで利用するデバイスを，IP ネットワークを介して利用するものである．例えば，コンピュータにおけるキーボードは，コンピュータ本体に PS/2 規格や USB 規格によって定義されたケーブルで物理的に接続されている (図 3.1)．しかし，Device over IP により，PS/2 規格や USB 規格によるケーブルではなく，別のコンピュータに接続されたデバイスを IP ネットワークから利用することを可能とする (図 3.2)．

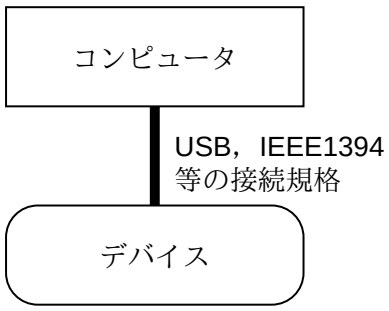

図 3.1: 従来のデバイス接続形態

3.1.1　Device over IP の利点

Device over IP には，IP ネットワークを用いることによる利点がある．まず，IP ネットワークを利用してデバイスとコンピュータを接続するため，コンピュータバスへデバイスを直結する必要がない．デバイスをコンピュータへ接続するためには，内蔵型デバイスはコンピュータのバススロットにデバイスを挿入し，外付型デバイスはコンピュータのポートへデバイスを接続する必要がある．しかし，IP ネットワークを利用することで，このようなデバイスのコンピュータへの物理的な接続がなくともデバイ

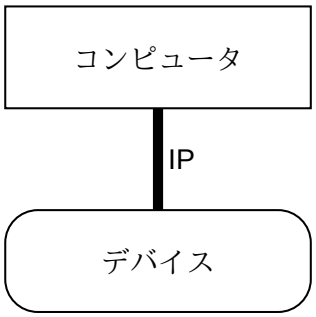

図 3.2: Device over IP の接続形態

スの利用が可能となる．その際必要となるのは，デバイス・コンピュータ間の IP ネットワークへの論理的な接続性のみである．

次に Device over IP では，コンピュータへのデバイスの接続に必要となる，物理バス数の制限を受けないことがある．現状，コンピュータの持つ拡張カードスロットやデバイスの外部接続ポートは有限であり，その数を超えてデバイスを接続することができない．しかし，IP ネットワークを用いてデバイスを接続するためには，コンピュータに必要なインタフェースはネットワークインタフェースのみである．各デバイス情報の送受信を 1 つのネットワークインタフェースのみで実現できることは，コンピュータの規模性における Device over IP の利点である．

また，IP ネットワークの処理を用いたデバイスの接続・切断も，Device over IP における利点となる．ネットワークを介してデバイスを接続することは，ネットワークの論理的接続・切断の処理のみでデバイスの接続と切断を実現する．Device over IP では，コンピュータへの物理的なデバイスの接続・切断を行うことなく，コンピュータの構成を切り替えることが可能である．利用者がコンピュータに求める機能は，その目的によって異なる．デバイスの追加・削除をネットワークの処理のみで行えることは利用者のコンピュータ環境の再構成を簡略化することとなり，利用者にとって自由なコンピューティング環境の提供が容易となる．

このように，デバイスとコンピュータの物理的な接続が必要なくなり，コンピュータの再構成を簡略化できることが Device over IP の利点である．

3.1.2 Device over IP の実現に向けた課題

一方で，Device over IP には IP ネットワークを利用することにより生じる問題点もある．まず，一般的な IP ネットワークはコンピュータのバス速度と比べ，通信速度が遅いことが挙げられる．デバイスバスの接続帯域速度には，近年では数百 Mbps から数 Gbps を要求するものがある．ネットワーク回線の帯域は速いものでも 1Gbps 程度のものが一般的であり，近年のデバイスバスの接続帯域速度よりも遅い．

次に，IP ネットワークはデバイス専用のバスではないため，安定性の面で問題がある点が挙げられる．IP ネットワークではデバイスの管理情報だけではなく，ユーザが

利用するデータや IP ネットワークのコントロールに関連する情報など，多様な情報が送受信されている．帯域の混雑やパケットの損失などは，デバイス専用に設計されたバスでは生じ難いものである．IP ネットワークの持つ特性がデバイスに影響を与えると想定されるため，これらの特性を克服することが Device over IP の実現には必要である．

また，IP ネットワークには盗聴などの危険がある．IP ネットワークのパケットは，その IP ネットワークを構成する中継ノードによって転送される．中継ノードなどでデータの盗聴があった場合，利用者が利用しているデバイスの全ての情報が攻撃者によって閲覧できてしまう．キーボードやディスプレイなどのデバイス情報が盗聴された場合，パスワードの盗難やプライバシーの侵害などの問題が生じると考えられる．

IP ネットワークという様々なデータが送受信される”バス”を利用することは，IP ネットワークの持つ問題点を抱えることとなる．従って，Device over IP の設計には，これら IP ネットワークを用いることにより生じる問題点を考慮する必要がある．

3.2 関連研究

本研究で述べる Device over IP の考え方は，その概念や実装手法によっていくつか関連研究が存在する．本節では Device over IP の関連研究を述べる．

3.2.1 USB/IP

USB/IP [9] [10] は，奈良先端技術大学院大学で開発された，USB デバイスをネットワーク経由で利用する Linux デバイスドライバである．USB/IP を利用すれば，多種の USB デバイスをネットワーク経由で扱うことができる．OS からも，USB 機器をそのコンピュータに接続しているかのように扱うことが可能である．

USB のデバイスドライバモデルは，図 3.3 のように階層化された構造となっている．USB キーボードや USB マウスなどの USB デバイスは，いずれも USB コントローラによって制御されており，デバイスに対応した Per-Device ドライバを入れ換えることでそれ以下の USB ホストコントローラの機能に変更を加えることなくデバイスを利用することが可能である．

USB/IP のデバイスドライバのモデルを，図 3.4 に示す．USB/IP ではホストコントローラドライバの代わりに，仮想ホストコントローラと呼ばれるドライバを利用する．USB/IP ではデバイスが接続されたホストを” サーバ”，デバイスを利用するホストを” クライアント” と呼ぶ．両者の通信は，デバイスが持つべき信頼性という観点から，TCP を用いて行われている．

クライアントの持つ仮想ホストコントローラは，アプリケーションやデバイスドライバから発行された USB Request Block (URB) を，サーバの持つ” スタブドライバ” に送信する．サーバは他の USB デバイスと同様に，URB をデバイスに対して発行する．USB デバイスからの要求がある場合，この逆の操作が行われる．このようにして URB を IP カプセル化し，転送するものが USB/IP のデバイスドライバモデルである．

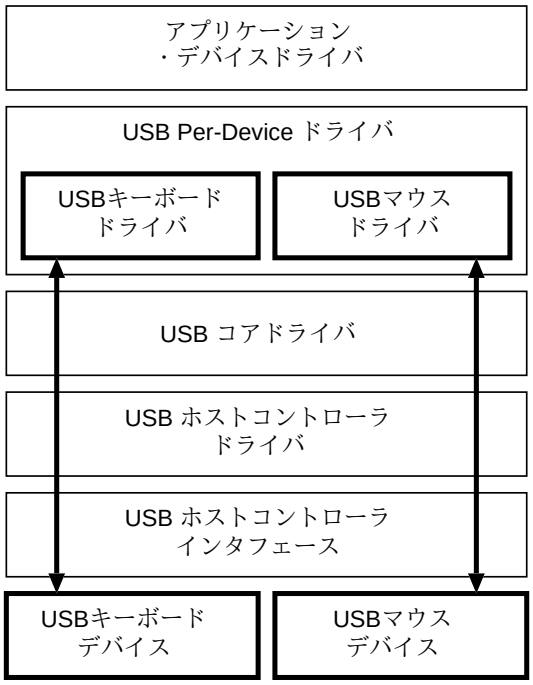

図 3.3: USB デバイスと USB コントローラの位置づけ

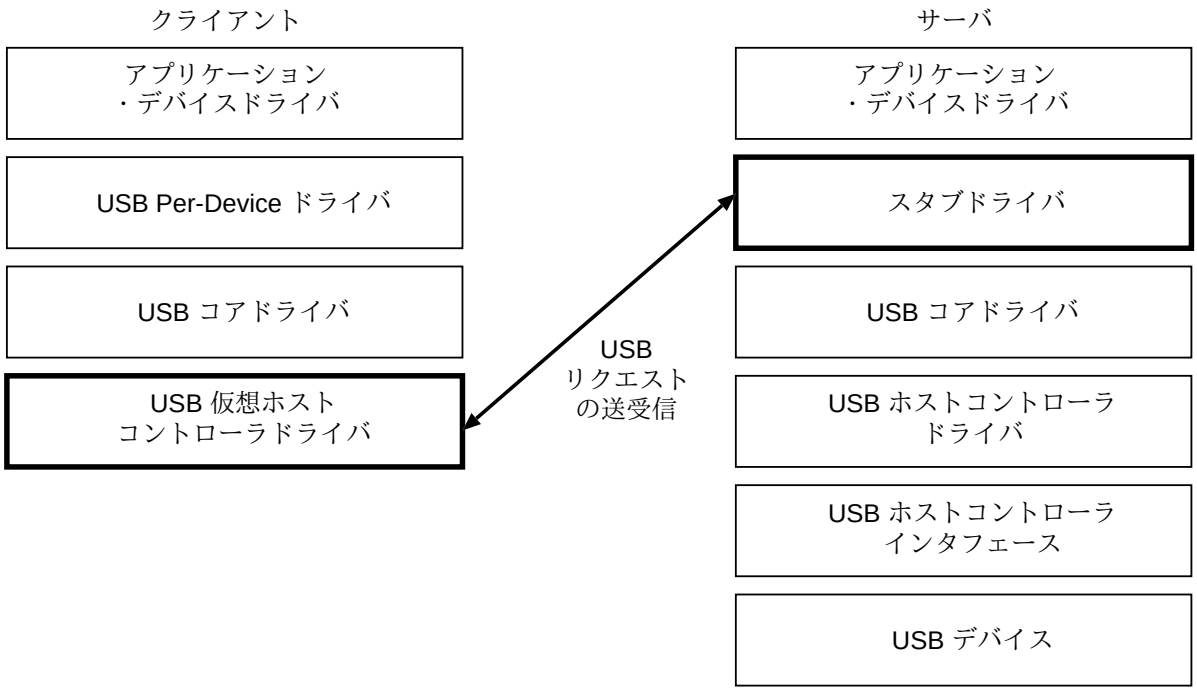

図 3.4: USB/IP デバイスドライバモデル

3.2.2 iSCSI

Internet Small Computer System Interface (iSCSI) [11] は，SCSI デバイスをネットワーク経由で接続することを可能とする規格である．iSCSI は，インターネット技術の標準規格を策定している Internet Engineering Task Force (IETF) によって規格化されている．

iSCSI は RFC 3720 で定義されており，例えば Microsoft 社の Windows Vista や一部の Linux ディストリビューションなど，近年公開・発売されている OS には iSCSI を標準で利用できるものも増えている．

3.2.3 VISA

第 2.5.2 項で挙げた NetStation の研究として，Virtual Internet SCSI Adapter (VISA) [12] がある．VISA も iSCSI と同様に，SCSI 命令を IP パケットにカプセル化し，ネットワーク経由で利用することを実現するものである．このようにハードウェアの命令を IP パケットにカプセル化することは，iSCSI 以前からも行われている研究である．

3.2.4 ATA over Ethernet

ATA over Ethernet (AoE) [13] は，イーサネットを介して ATA 接続のストレージデバイスを利用するプロトコルである．AoE はイーサネット上で利用する規格であり，IP や TCP などの上位のプロトコルは利用しない．従って，インターネットを介して利用することはできず，Storage Area Network (SAN) の用途に限られる．

3.2.5 関連研究のまとめ

USB/IP や iSCSI などの関連研究に共通する点として，デバイスの接続手法は IP ネットワークを利用する点と，通常のデバイスの接続方法とは違うものの OS からの認識は従来のデバイスと変わらない点がある．専用ユーティリティが OS 上で実行されている必要があるなど若干異なるが，従来のデバイスの形を極力変更しない設計である．

従来のデバイスを IP でカプセル化することで OS を大きく改変することは，開発に負担がかかる．OS に対する改変を少なくすることで IP カプセル化にかかるコストを減らすことが，Device over IP を実現する上で必要である．

3.3 Device over IP への要求

本章の内容を踏まえた上での Device over IP への要求には，以下のようなものが考えられる．

一点目は，IP ネットワーク上のデバイスへの接続・切断を実現することである．一般的なデバイスは，ケーブルを接続することによってデバイスの利用を開始する．また，ケーブルの物理的な取り外しによって物理的にデバイスの利用を終了する．Device over IP では，デバイスは常に IP ネットワーク上に存在し，IP ネットワーク上のパケットを用いて論理的な接続・切断を行う．従って，IP ネットワーク上のデバイスの存在確認や，そのデバイスの利用開始・終了が行える必要がある．

二点目に，OS に対する IP ネットワークの隠蔽が必要な点がある．IP ネットワークを介して接続されているデバイスも，利用者の視点に立った場合，一般的なデバイスと同等に扱えるべきである．また，IP ネットワークを介したデバイスの特性上特殊な処理が必要となる場合，OS に対する改変も必要となる可能性がある．出来る限り必要最小限の変更で Device over IP を実現するためには，OS に対しては利用しているデバイスが，IP ネットワークを介して接続されていることを隠蔽すべきである．

三点目に，IP ネットワークの特性やその帯域消費を抑制する必要がある．第 3.1.2 項で述べた通り，IP ネットワークはデバイス専用のバスではない．従って，IP ネットワーク上で発生し得る遅延やパケットロスへの対応が必要である．また，IP ネットワークで利用できる実効帯域は，デバイス専用に設計されたバスよりも遅いことが多い．デバイスの持つ全ての命令や規格を IP ネットワーク経由で送受信するのではなく，必要な命令のみを送受信するように最適化を行う必要がある．

第4章　OSによるデバイスの利用

オペレーティングシステム (OS) の役割には，デバイスリソースの管理や利用環境の提供がある．OS はデバイスを管理するためにデバイスの種類に応じて固有の識別番号を割り当てるなどして，複数の同一種類のデバイスを管理する．また，ユーザアプリケーションからの利用を容易にするために，デバイスをアプリケーションから極力隠蔽し，その環境をユーザに提供する．Device over IP はバスとして IP ネットワークを用いるが，提供する機能や概念は既存デバイスから極力変更しないことが望ましい．本章では OS におけるデバイスの利用について述べ，その枠組みの中における Device over IP の実現手法を述べる．

4.1　既存 OS の構造

既存 OS は，一般的に階層化された設計となっている．まず，OS には大きく分けてカーネルスペースとユーザスペースがある．ユーザが利用するアプリケーションなどが実行される部分がユーザスペースであり，OS 自身の実行に関わる部分がカーネルスペースである．

既存 OS の構造は，大きく次の層に分けることができる．

- **アプリケーション**: 一般ユーザが実行するアプリケーションが実行される層
- **カーネル**: OS の中核となる部分
- **デバイスドライバ**: ハードウェアをコントロールするためのソフトウェア
- **デバイス**: 実際に人間が触れる機器と，それをコントロールするためのホストコントローラ

また，アプリケーションと OS カーネルの通信を仲介するものとして**システムコールインタフェース**がある．システムコールインタフェースは，アプリケーションなどの上位層からデバイスに対して，共通の処理を提供する役割を持つ．例えば UNIX システムコールである `read`，`write`，`mmap` などは，デバイスが異なっても共通して利用できるものである．

一方で，OS カーネルとデバイスドライバの通信を仲介するものとして**デバイスドライバインタフェース**がある．デバイスドライバインタフェースは，OS カーネルがデバイスドライバに対して指示するために使われるものである．この指示にはアプリケーション層からの指示に加え，デバイスを操作するための特殊な命令などもある．

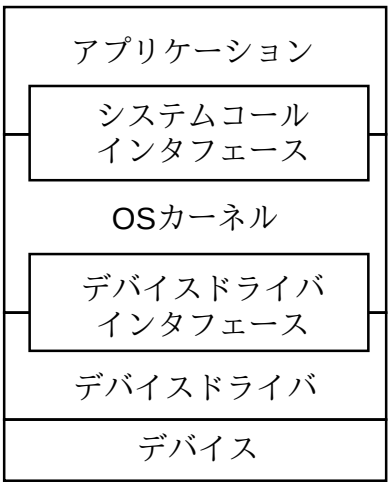

図 4.1: OS の階層構造

4.2　デバイスの利用

前節で述べた通り，OS は階層化された構造を持っている．OS がデバイスを扱うためには，何らかの形で上位層からデバイスを操作する機構が必要である．このような機構を，一般的に” 抽象化” と呼ぶ．本節では，デバイス利用における抽象化の役割について述べる．

4.2.1　OS のデバイス利用における抽象化

OS やアプリケーションがデバイスを操作するためには，そのデバイスの操作を管理するための機構が必要である．アプリケーションからの要求を受け入れる際，異なるデバイス間でも可能な限り操作を共通化することが望ましい．OS カーネルはアプリケーションからの要求を受け，デバイスドライバに対してその要求を一元的に渡す役割を担っている．

OS 上ではデバイスは，大きく分けてキャラクタデバイスとブロックデバイスの 2 種類に分類される．キャラクタデバイスとは，1 文字 (8 ビット，1 バイト) ごとにデータの入出力を行うデバイスのことである．ブロックデバイスとは，そのデバイスが定義した一定量のデータを単位として，データの入出力を行うデバイスのことである．キーボードやマウスなど多くのデバイスはキャラクタデバイスであり，磁気ディスクや光学ディスクドライブなどのストレージデバイスはブロックデバイスである．

デバイスは，UNIX OS の多くではファイルシステム上の`/dev`ディレクトリに，ファイルの形で抽象化されている．このようにデバイスが抽象化されたファイルを”デバイスファイル”と呼ぶ．このデバイスファイルに対して，通常のファイルと同様に`open`，`close`，`read`，`write`といった処理を行うことでデバイスに対する要求を発行するものが，デバイスファイルを用いたデバイスの抽象化モデルである．デバイスファイルは

多くのデバイスに対して共通の手順の操作を実現する，Application Program Interface (API) の役割を担っている．

4.2.2 デバイスへの命令

デバイスファイルは，アプリケーションなどに対して共通して利用できるインタフェースを提供する抽象化である．アプリケーションなどからデバイスファイルに対する命令が発行されれば，次にデバイスドライバに対してデバイス固有の命令を発行する．

例えばLinux フレームバッファのデバイスドライバには，`fb_open`，`fb_release`，`fb_read`，`fb_write` といった全てのデバイスに共通のシステムコールとそれに対応する関数が含まれている．デバイスファイルに対して `read` システムコールが発行されれば，それはデバイスドライバインタフェース・デバイスドライバを介して `fb_read` と解釈され，デバイスに命令が発行される (図 4.2)．

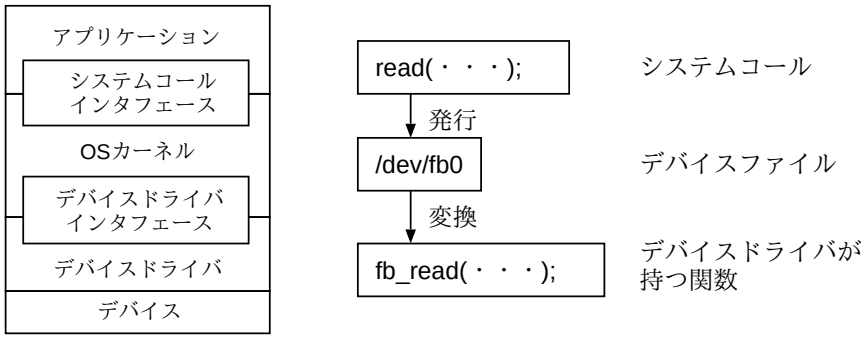

図 4.2: システムコールをデバイスドライバの持つ関数に変換

また，デバイスドライバにはデバイスの存在を OS に登録する `register` 関数や登録を解除する `unregister` 関数，その他デバイスを利用するために必要な関数が含まれる．これらはデバイスドライバが対応するデバイスに対して，固有の関数である．デバイス固有の関数も，予めそのようにプログラムしておけばアプリケーションなどから実行することも可能である．その場合にはシステムコールからデバイス固有の命令への変換は必要なく，そのまま命令が渡される．

このように共通して利用できる API と，デバイス固有の命令を発行できるようにする関数を利用できるようにすることが，OS におけるデバイスの抽象化である．

4.3 デバイスのネットワーク接続手法

Device over IP の実現には，デバイスをネットワークに接続する必要がある．既存の OS の構造やその抽象化の仕組みを踏まえた上でのデバイスのネットワーク接続手法として，図 4.3，図 4.4 の手法が考えられる．両手法では，ネットワーク転送を行う単位・処理を担当するシステムが異なる．両手法の比較を，表 4.1 に示す．

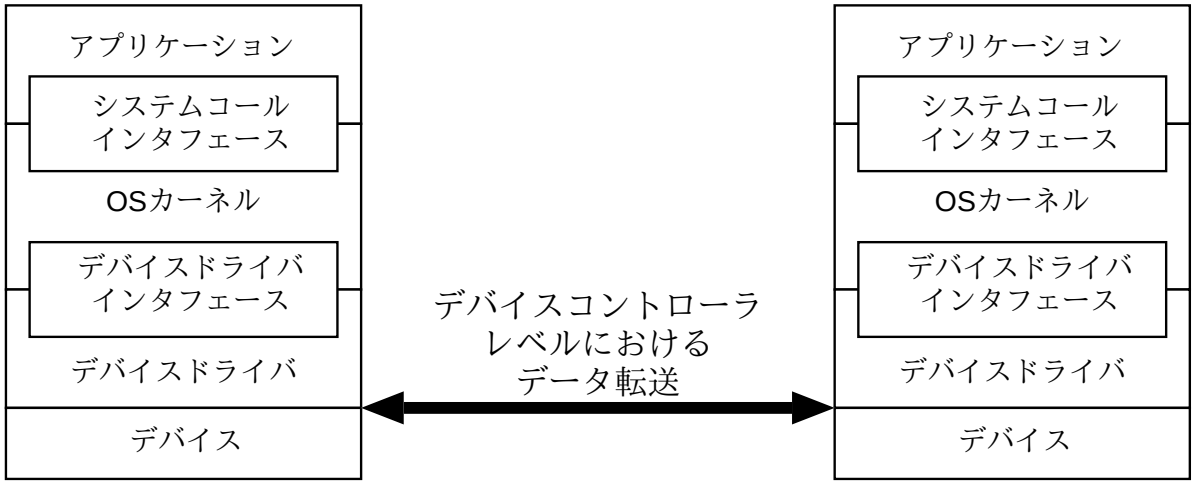

図 4.3: デバイスのネットワーク接続手法 1

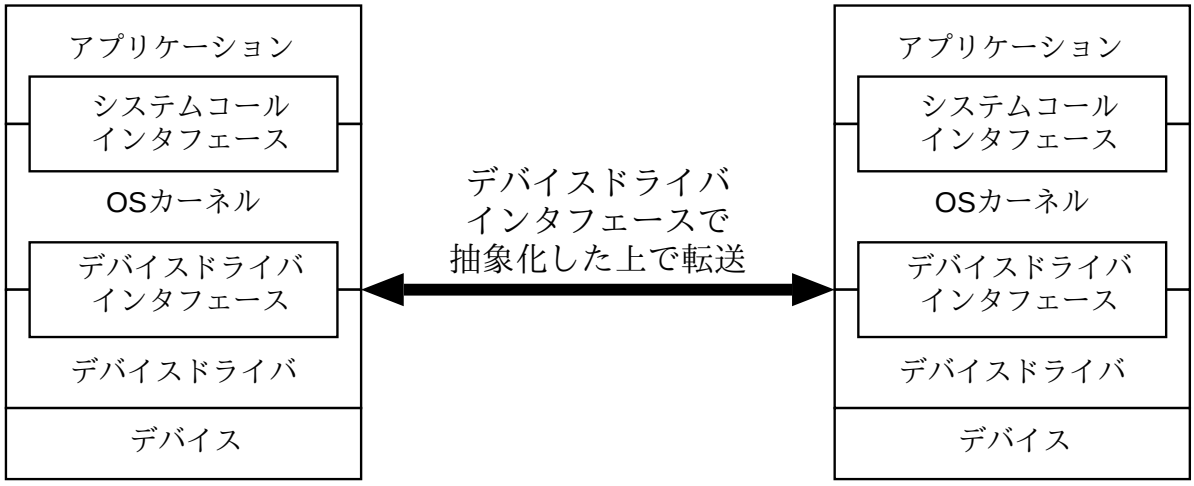

図 4.4: デバイスのネットワーク接続手法 2

両手法について，コンピュータシステムが行う処理の流れ自体は同じである．しかし，ネットワーク転送を行うタイミングに違いがある．手法 1 は，デバイスに近い層でネットワーク転送を行っている．コントローラの持つ情報をネットワーク転送し，受信したコントローラ情報を用いてデバイスの処理を実行するものである．この手法であれば，同一コントローラによって制御出来るデバイスであれば，どのようなデバイスであっても利用することができる．例えば，関連研究として挙げた USB/IP はこの手法で実現されている．

手法 2 は，OS の持つデバイスの種類に基づいた情報をネットワーク経由で転送するモデルである．この手法であれば，コントローラの種類に依らず，デバイス情報の送受信をデバイスの抽象化によって扱うことが可能である．例えば，PCI 接続のディスプレイアダプタも USB 接続のディスプレイアダプタも，OS にとっては”ディスプレイ

表 4.1: デバイス情報をネットワークに送信する手法

手法	図	利点	欠点	例
1	図 4.3	バス規格に特化: 同一規格ならば汎用	バス規格に依存: 新規規格に対応できない	USB/IP, iSCSI
2	図 4.4	事前にデバイスを抽象化: 幅広いバス規格に対応	デバイス毎に抽象化: デバイス毎に機構が必要	本研究

アダプタ”として抽象化できる．コントローラに依存しないデバイスの抽象化によって，異なるバスを用いた同一種のデバイスを幅広く IP 化することが可能である．

4.4 関連研究の問題点

第 3.2 節に挙げた Device over IP の技術は，既存のプロトコルに対して極力変更を必要としないように実現されている．この手法は，互換性を達成する上では有効な手法である．例えば USB/IP(第 3.2.1 項) は USB コントローラの処理を IP ネットワーク経由で発行・処理しており，この手法を用いればほぼ全ての USB デバイスを IP ネットワーク経由で利用することが可能である．PCI 規格および USB 規格のビデオ表示装置とサウンド装置をバスの規格で IP ネットワーク化したものは，図 4.5 の枠組みで表すことができる．

	ビデオ デバイス	サウンド デバイス
PCI規格 による接続	PCI ビデオカード	PCI サウンドカード
USB規格 による接続	USB ビデオアダプタ	USBサウンド デバイス

図 4.5: バス命令のネットワーク化

同一規格のバスを IP ネットワーク化することで，幅広い種類のデバイスの IP ネットワーク化を実現できる．しかし，この手法はデバイスの特徴や機能に最適化されたデータ転送の手法ではない．デバイスには，デバイス毎に要求される機能や性能があり，IP ネットワーク経由では利用できない方が好ましい機能や，ネットワーク特性を越える上ではより優先されるべき機能がある．また，デバイスの動作には不要であっても，バスの仕様で定義されているデータ転送などによる，余分なデータ転送効率やシステム全体の処理効率の低下が生じる可能性がある．

全てのデバイスにおいて，共通した IP ネットワークを経由した利用を実現することは難しい．例えば USB デバイスには，表 4.2 に示す 3 種類の転送速度がある．USB 2.0 High Speed の利用には規格上，480Mbps の帯域が必要である．しかし，今日の IP ネットワークの転送速度は，LAN では 1000Mbps の帯域が利用できることもあるが，WAN での利用は 100Mbps を下回ることが一般的である．これらの数値は規格値であり，実際に利用できる実効帯域は更に少なく，帯域が不足することが分かる．

表 4.2: USB1.1/2.0 各規格の比較

転送モード	転送規格速度
USB 1.1/2.0 Low Speed	1.5Mbps
USB 1.1/2.0 Full Speed	12Mbps
USB 2.0 High Speed	480Mbps

また，USB のデータ転送には” コントロール”・” インタラプト”・” バルク”・” アイソクロナス” といった種類がある．コントロールはデバイスを管理・制御するための命令である．インタラプトは，一定の時間ごとにデバイスの持つデータを取得するデータ転送モデルである．バルクは，一定量のデータを一括で処理するデータ転送モデルである．そして，アイソクロナスはデータを連続して処理するデータ転送モデルである．デバイスの用途や種類に応じてデータ転送方式は変化し，キーボードやマウスといったデータ量の少ない入力機器，ストレージデバイスのようにある程度のデータのバッファリングを行える機器，ディスプレイデバイスなどの表示にかかる時間がユーザの体感性能に影響する機器など様々な機器の種類が存在する．データ量は少ないが，IP ネットワークの特徴である遅延や再送が生じることで，入出力内容が適切に転送されないといった状況も考えられる．

従って，単純にデバイスの規格命令をそのまま IP ネットワーク上で転送するだけでは，全てのデバイスにおいて有用であるとは限らない．但し，一部のソフトウェアにはデバイスの挙動に依存したものもあり，互換性・汎用性を主眼におくと変更を加えない方が良い場合もある．互換性と効率性のどちらに主眼を置くか，両者の特徴を踏まえることが必要である．

4.5 アプローチ

OS がコンピュータ上のデバイスを利用するためには，何らかの形でそのデバイスへアクセスする手法として抽象化が必要である．その抽象化手法は OS やデバイスの種類によって若干異なるが，デバイスの種類とその接続規格に基づいて行われることが一般的である．この手法でデバイスを抽象化することは，既存 OS との互換性において有効である．実際に第 3.2 節で挙げた関連研究も，デバイスの IP 化を既存デバイスのバスプロトコルの枠組みで実現している．

しかし，この手法ではデバイスのバスが持つ全ての通信を IP ネットワーク上で送受信する必要が生じる．各デバイスには要求される機能や性能があり，全ての情報を IP ネットワーク経由で利用する必要はない．更に，デバイスに対して直接命令を送信するものなど，IP ネットワークを介して利用できない方が望ましい機能もある．デバイスに対して必要な要件やその機能，特徴に応じたデバイスの IP 化が必要である．

そこで，本研究ではデバイスの種類に着目した IP ネットワーク化を行う．先述の例と同じ PCI 規格および USB 規格のビデオデバイスとサウンドデバイスを，デバイスの種類に着目して IP ネットワーク化すると図 4.6 のように分類できる．

	ビデオ デバイス	サウンド デバイス
PCI規格 による接続	PCI ビデオカード	PCI サウンドカード
USB規格 による接続	USB ビデオアダプタ	USBサウンド デバイス

図 4.6: デバイスの種類に着目したネットワーク化

デバイスを IP ネットワークへ接続するためには，IP アドレスの管理やそのデバイスが所属するネットワークの管理，デバイス自身の状態を把握するなど様々な処理が必要である．複雑な処理をデバイス単独で実行するには，デバイスを管理するための専用の OS が必要であると考えられる．本研究では，この” デバイスを管理する” ことを専門とする OS を，Tiny OS と呼称する．Tiny OS はデバイスの基本的な接続状況の管理のみを担当し，それ以外のユーザが要求する動作は All-IP Computer 全体に掛かる OS が担当する．Tiny OS の役割とは，デバイスを All-IP Computer へ接続するために必要な処理を行うことである．

更に，デバイスを利用するためには OS が利用できる形での抽象化が必要である．All-IP Computer における抽象化を，ここまでの内容を踏まえて表したものが図 4.7 である．例えば現在の Linux において，IDE 接続のハードディスクは `/dev/hda` などといった名前のデバイスファイルで抽象化され，Serial ATA 接続や USB 接続のハードディスクは `/dev/sda` などといった名前のデバイスファイルで抽象化される．これらはデバイスの種類と接続規格の両方に着目した抽象化であるが，All-IP Computer では `/dev/disk` のように，デバイスの種類にのみ着目した抽象化を行う．これには All-IP Computer において，IP ネットワークを介したデバイス利用を容易とする目的がある．デバイスの規格による差異をシステム内で吸収し，デバイスの種類によって行われた抽象化を All-IP Computer における抽象化として定義する．

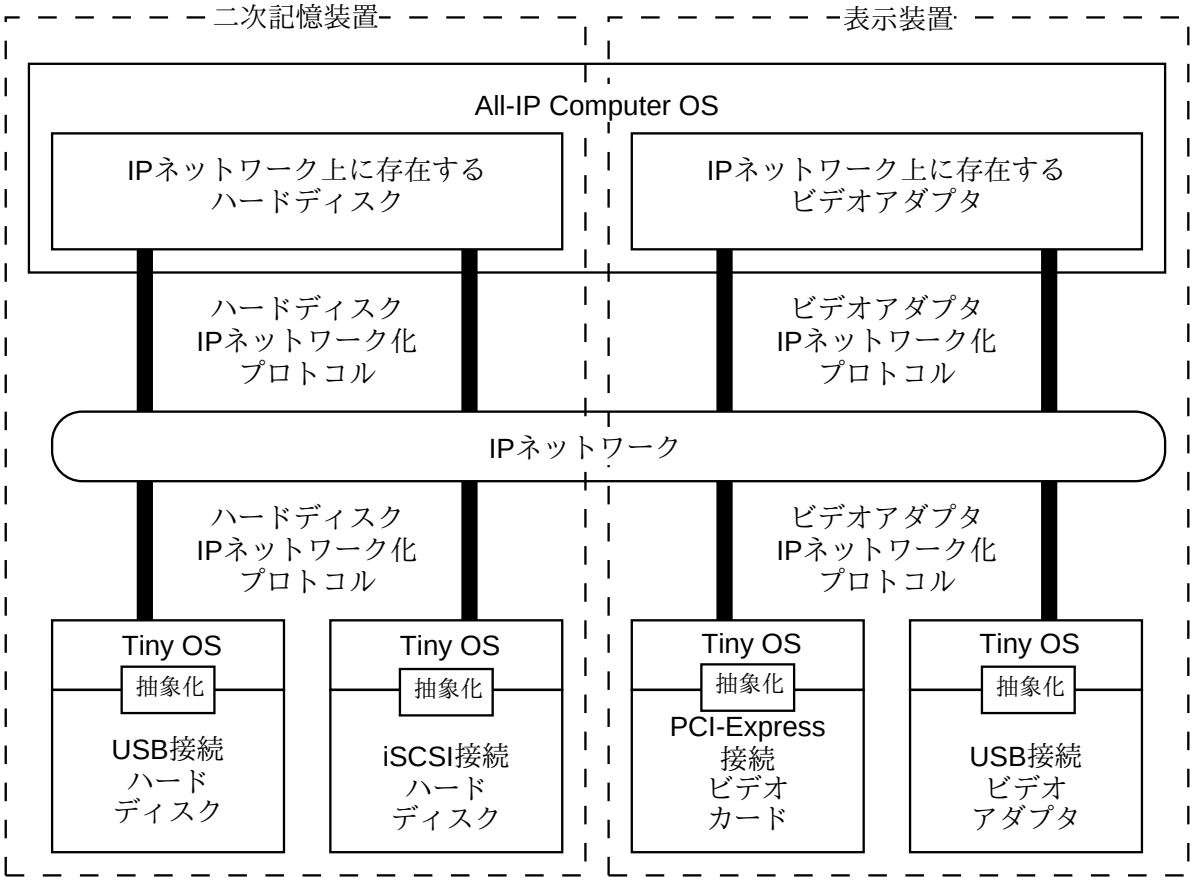

図 4.7: All-IP Computer における抽象化

第5章　ディスプレイの機能

本章では，既存のコンピュータシステムにおけるディスプレイの機能と本研究が提案するシステムの前提項目を整理し，Display over IP の機能要件を議論する．

5.1　解像度調整

ディスプレイに要求される機能の一つに，解像度調整がある．解像度とは画面上に表示される画像の密度で，解像度が高いほどより多くの情報を一度に表示できる．そのディスプレイで利用できる最高の解像度を設定すれば最も多くの情報を表示できるが，解像度が増えるほど情報量も増えるため，IP ネットワークに対して送信するデータ量も増加する．従って，利用者によって目的とネットワーク帯域に合わせた解像度が設定できる必要がある．

コンピュータ用ディスプレイの解像度として一般的なものを，表 5.1 に示す．

表 5.1: 一般的なディスプレイ解像度

名称	横ピクセル数	縦ピクセル数
VGA	640	480
SVGA	800	600
XGA	1024	768
SXGA	1280	1024
XGA+	1152	870
SXGA+	1400	1050
UXGA	1600	1200
WVGA	800	480
WSVGA	1024	600
WXGA	1280	768
WXGA+	1440	900
WSXGA+	1680	1050
WUXGA	1920	1200

5.2 色数調整

もう一つのディスプレイの重要な機能に，色数調整がある．色数が多いほど，より詳細な図表等の表示が可能である．しかし，色数が多ければ情報量も多く，より多くの情報を転送する必要がある．本研究で行うディスプレイ情報のネットワーク転送には，情報量の抑制も必要である．

コンピュータ用ディスプレイの色数として一般的なものを，表 5.2 に示す．

表 5.2: 一般的なディスプレイ色数

ビット数	色数
1 ビット	2 色
2 ビット	4 色
4 ビット	16 色
8 ビット	256 色
16 ビット	65,536 色
24 ビット	16,777,216 色

5.3 表示位置調整

ディスプレイの持つ機能の一つに，映像の表示位置の調整がある．近年の OS の多くは，ディスプレイ表示位置を適切に検知・設定する．しかし，表示位置の検知で期待された通りの結果を得られない場合，利用者自身の手で表示位置を手動で調整できる必要がある．

5.4 周波数調整

ディスプレイにおける表示の調整に，同期周波数の調整がある．同期周波数には，垂直走査周波数と水平走査周波数の 2 種類がある．水平走査周波数とは，ディスプレイが 1 秒間に描画できるラインの数である．ラインとは，ディスプレイの描画を横方向に行う際の単位である．一方で垂直走査周波数とは，1 秒間にディスプレイの表示を再描画する回数である．例えば 60Hz であれば，1 秒間に 60 回の再描画が発生している．

これら 2 種類の走査周波数は，ディスプレイ表示の解像度や色数に影響する．一般的には走査周波数が高いほど，解像度や色数を増加することが可能である．

5.5 Linuxにおけるディスプレイ表示

Linuxにおけるディスプレイ表示には，主に二種類の方法がある．ハードウェアを直接利用する方法と，間接的な手法としてフレームバッファメモリを利用する方法である．

5.5.1 ビデオハードウェア

ビデオハードウェアを直接利用する方法は，ほとんどのLinuxシステムにおいて標準で利用されている．この手法は描画処理をハードウェアで行うため高速であるが，デバイスの仕様に依存する動作も多く，一つのデバイスに対するカーネルモジュールを他のデバイスでは利用できないものが多い．

5.5.2 フレームバッファ

個々のビデオデバイスが異なるデバイスドライバを利用することは，汎用性や新規のデバイスへの対応等の面において不利である．その対策として，VESA Framebuffer (VESAFB) という統一された規格がある．VESAFBはVideo Electronics Standards Association (VESA) [14] によって提唱されたVESA BIOS Extension [15] を基としており，主としてIntel x86アーキテクチャのPC/AT互換機で利用できる．VESAFBはOS上では，” フレームバッファ” デバイスとして認識される．

フレームバッファとは，コンピュータ上のグラフィックデバイスを抽象化するメモリデバイスである．OSがビデオメモリ (VRAM) を扱う方法の一つとして，フレーム1つ分をフレームバッファメモリに保存する．画面の再描画を行う際にはOSがグラフィックデバイスを操作するのではなく，フレームバッファメモリから描画データを転送し，描画することで実現する．

5.6 Linuxにおけるフレームバッファシステム

Linuxにおけるフレームバッファシステムは，カーネルモジュールの形式で実装されている．カーネルモジュールの形式は，カーネル機能としてのカーネルモジュール形式とLoadable Kernel Module (LKM) 形式から選択できる．

5.6.1 モノリシックカーネルとマイクロカーネル

カーネルとは，OSの中核となるソフトウェアのことである．カーネルにはモノリシックカーネル (図5.1) とマイクロカーネル (図5.2) の2種類の実装手法がある．モノリシックカーネルとは，多くの機能をOSの中心であるカーネルと共に実装する手法で

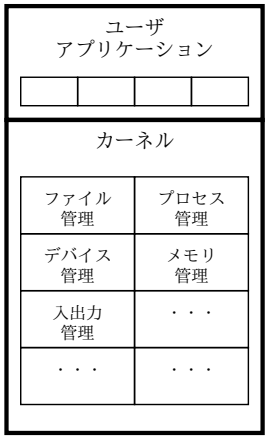

図 5.1: モノリシックカーネル

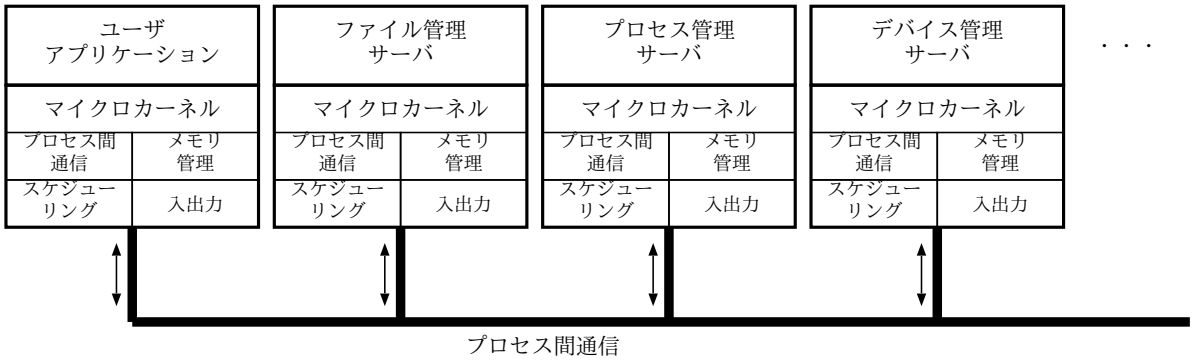

図 5.2: マイクロカーネル

ある．一方，マイクロカーネルとはOSのカーネルの機能を必要最小限にまとめ，その他デバイスのサポートなどをカーネル外で行う手法である．

マイクロカーネルはカーネルの機能を必要最小限に抑えるため，多くの機能がカーネル外に実装されている．その利点として，まずOSの持つ機能の入れ換えなどが容易であることが挙げられる．OSのある機能を交換するためには，その機能を担当する部分だけを交換すれば良い．この事はカーネルの安定性の向上に役立つとされている．例えばOSの機能をアップデートする場合などに，OSカーネル自体を停止しなくても，当該の機能のみを停止することで解決できる．また，ある機能が異常停止した場合でも，OSカーネル全体が停止することを避けることが可能である．

5.6.2 サブシステム

Linuxカーネルは，モノリシックカーネルとして実装されている．しかし，近年ではカーネル本体のサイズを必要最小限に留め，サブシステムやLKMを利用することが主流となりつつある．サブシステムとはOSの持つ特定の機能の集合体であり，Linuxに

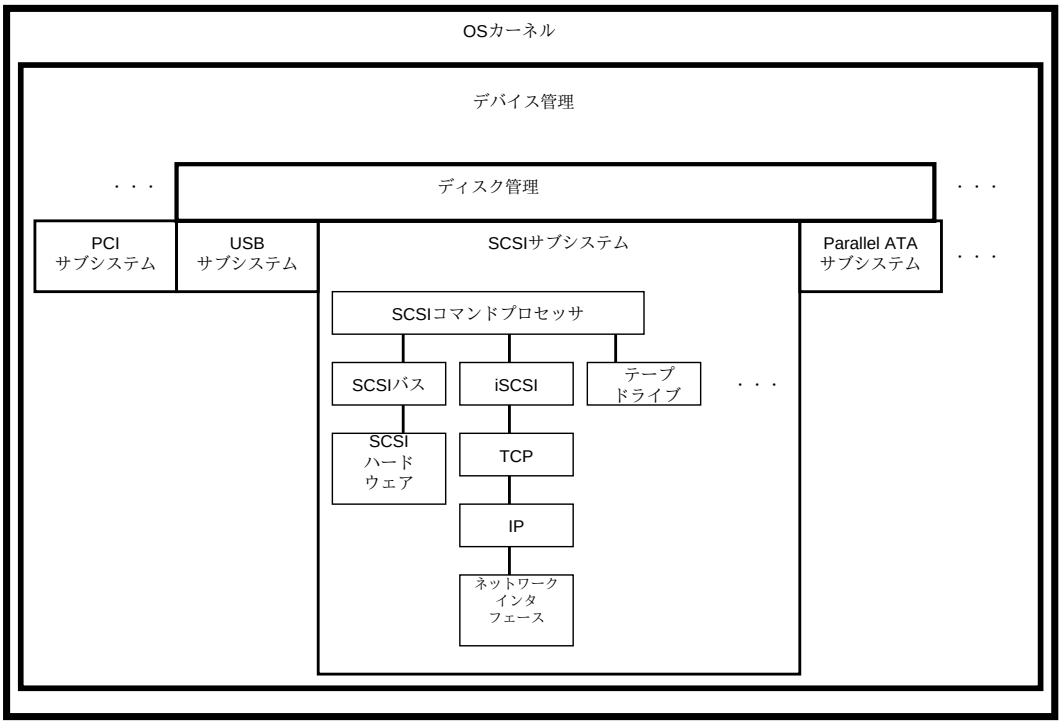

図 5.3: サブシステムの概念図

```
% ls -l /dev/input/
crw------- 1 root root 13, 64 event0
crw------- 1 root root 13, 65 event1
crw------- 1 root root 13, 63 mice
crw------- 1 root root 13, 32 mouse0
```

図 5.4: Input Subsystem デバイスファイル

おける例としてはUSB，SCSI，IDE などの接続規格を管理するサブシステムや，Disk，Input などデバイスの種類を管理するサブシステムもある．また，SELinux など，デバイスではない機能がサブシステムとして実装されているものもある．サブシステムの概念図を図 5.3 に示す．

Linux のサブシステムの 1 つに，Input Subsystem がある．Input Subsystem はキーボード・マウス・ジョイパッドなどの入力機器を統括して扱えるようにするものである．Input Subsystem で抽象化されたデバイスは，ファイルシステムの`/dev/input` 以下にデバイスファイルとして抽象化される (図 5.4)．

Input Subsystem で抽象化されたイベントは，メジャー番号 13 のデバイスとして認識される．各 event デバイスファイルの関連付けは，`/proc/bus/input/devices` ファイルを閲覧することで確認できる (図 5.5)．

```
I: Bus=0011 Vendor=0001 Product=0001 Version=ab83
N: Name="AT Translated Set 2 keyboard"
P: Phys=isa0060/serio0/input0
S: Sysfs=/class/input/input0
H: Handlers=kbd event0
B: EV=120013
B: KEY=4 2000078 3802078 f840d001 f2ffffdf ffefffff ffffffff
fffffffe
B: MSC=10
B: LED=7

I: Bus=0011 Vendor=0002 Product=0006 Version=0000
N: Name="ImExPS/2 Generic Explorer Mouse"
P: Phys=isa0060/serio1/input0
S: Sysfs=/class/input/input1
H: Handlers=mouse0 event1
B: EV=7
B: KEY=1f0000 0 0 0 0 0 0 0 0
B: REL=103
```

図 5.5: event デバイスファイル関連付けの例

```
% insmod module.ko
% rmmod module
```

図 5.6: LKM の利用開始・終了

5.6.3 Loadable Kernel Module

カーネルモジュールとは，新たな機能を個別のモジュールとしてカーネル本体に追加することで，カーネルの機能を拡張するものである．カーネルモジュールにはデバイスドライバの機能を持つものやネットワークの追加機能を提供するもの，データの暗号化アルゴリズムを提供するものなど様々なものがある．先に説明したサブシステムにも LKM として実装され，必要に応じて OS 上でロードして利用するものがある．

LKM とは，Linux が起動してから動的に利用の開始･終了が可能なカーネルモジュールである．LKM でないカーネルモジュールは，OS の起動と共にしか利用を開始できず，OS の終了と共にしか利用の終了ができない．LKM としてコンパイルされたカーネルモジュールは，ユーザの要求に応じて動的に利用を開始・終了できる (図 5.6)．

```
% ls -l /dev/fb0
crw------- 1 muda root 29, 0 /dev/fb0
```

図 5.7: フレームバッファデバイスファイル

5.6.4 フレームバッファメモリ

フレームバッファデバイスは，メジャー番号29番のキャラクタデバイスとしてファイルシステム上に抽象化される．1つのコンピュータシステムにおいて，最大32個までのフレームバッファデバイスを処理することができる．フレームバッファデバイスのデバイスファイルはファイルシステム上の/devディレクトリに，通常ではfb?(?はマイナー番号) の形式で抽象化される (図5.7)．

Linuxにおいて，フレームバッファデバイスはメモリデバイスと同等として扱われる．従って，基本的なシステムコールであるreadやwriteなど以外にも，seekやmmapなどのシステムコールも利用できる．それらシステムコールを抽象化されたデバイスファイル/dev/fb0などに対して実行すると，デバイスドライバインタフェースにおいて，フレームバッファカーネルモジュールで定義されているfb_readやfb_writeなどの関数に変換される．変換された命令はデバイスに発行され，その処理が行われる．この流れを図5.8に示す．

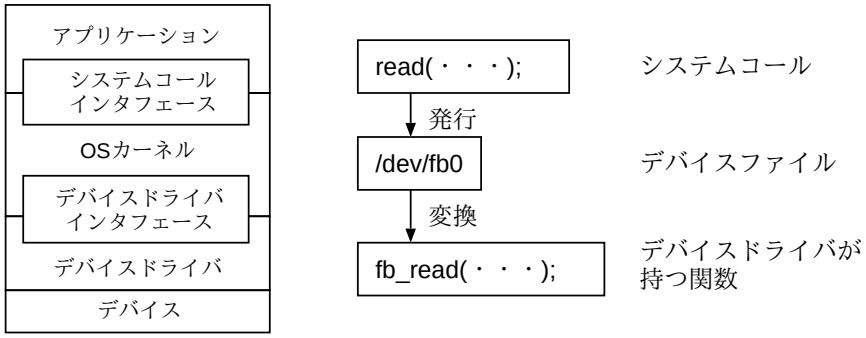

図 5.8: システムコールをデバイスドライバの持つ関数に変換 (再掲)

フレームバッファメモリは，一次元の一般的なメモリと同等の捉え方で扱うことができる．例えば，フレームバッファメモリから指定した長さのデータを読み取るfb_readは，static ssize_t fb_read(struct file *file, char __user *buf, size_t count, loff_t *ppos)として定義されており，その引数にファイルディスクリプタやバッファ以外に読み込む長さやメモリ上のオフセット値を取る．

一般的なビデオカードは，フレームバッファメモリで占有するメモリ領域よりも大きい容量のビデオメモリを持っている．フレームバッファメモリ分をビデオメモリ上に

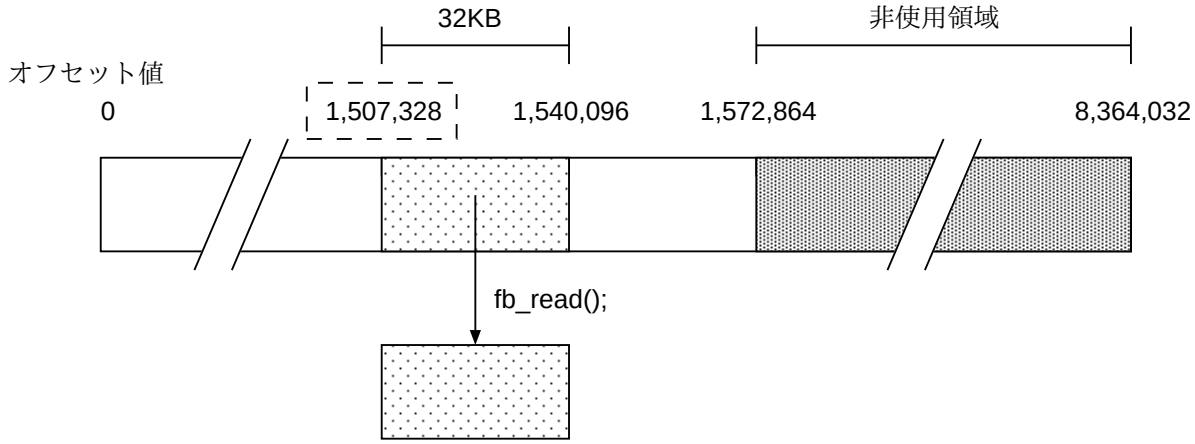

図 5.9: 指定オフセットからフレームバッファメモリのデータを取得

マッピングし，その領域に対してキャラクタデバイスと同様に処理できることが，Linuxにおけるフレームバッファデバイスの特徴である．

例えば，フレームバッファメモリのオフセット1,507,328番地から32,768バイト(32KB)のデータを取得する様子を図5.9に示す．ハードウェアは8,364,032バイト(8MB)のビデオメモリを持っており，ディスプレイ機能は16ビットカラーの解像度XGAで利用しているものとする．この時に必要なフレームバッファメモリの領域は，1,572,864バイト(1.5MB)である．なお，必要なフレームバッファメモリの容量は，下記の式で求めることができる．

$$(\text{X方向の解像度} \times \text{Y方向の解像度} \times \text{色数}) \div 8$$

上記の例では，$1024 \times 768 \times 16 \div 8 = 1{,}572{,}864$バイトとなる．

フレームバッファメモリの基本的な関数は，Linuxで利用できる殆どのフレームバッファデバイスで共通して利用される．従って，新しいデバイスの追加などがあっても基本的な部分への変更はない．

5.6.5 フレームバッファカーネルモジュール

Linuxフレームバッファシステムは，複数のフレームバッファデバイスの基幹となるカーネルモジュールと，デバイス毎の特性に応じたデバイスドライバの組み合わせとして実装されている．フレームバッファデバイスへのカーネルモジュールは，図5.10に示すファイルで構成されている．フレームバッファカーネルモジュールはLKMとして利用でき，Linux OS起動時に自動で，または起動後に動的に利用を開始することが可能である．

図5.10において，`fbcmap.c`，`fbcvt.c`，`fbmem.c`，`fbmon.c`，`fbsysfs.c`の各ファイルはフレームバッファデバイスドライバのほとんどで共通して使用される，汎用命令を受け付けるためのインタフェースの役割を担う．これらファイルはコンパイルされる

図 5.10: フレームバッファカーネルモジュールの構成

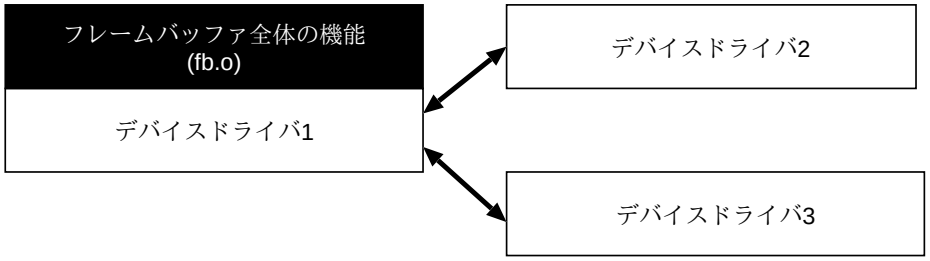

図 5.11: フレームバッファシステムの構成

と，fb.o という，フレームバッファ機能に共通して利用される仮想デバイスドライバとなる．この仮想デバイスドライバは，第 5.6.6 項で後述する各フレームバッファデバイス固有のデバイスドライバと組み合わせて利用される．図 5.11 に示すように，Linux におけるフレームバッファシステムは，異なる種類のビデオカードに共通した描画処理を行う機構を OS やアプリケーションに提供する．

フレームバッファカーネルモジュールには，いくつかの主要な関数がある．以下の関数は，フレームバッファメモリを処理する fbmem.c ファイル上に存在する関数である．

- `static int fb_open(struct inode *inode, struct file *file)`
  フレームバッファデバイスをカーネルモジュールが処理できるよう，デバイスを開く関数

- `static int fb_release(struct inode *inode, struct file *file)`
  フレームバッファデバイスの利用を終了する関数

- `static ssize_t fb_read(struct file *file, char __user *buf, size_t count, loff_t *ppos)`
  フレームバッファデバイスから表示データを読み込む関数

- `static ssize_t fb_write(struct file *file, const char __user *buf, size_t count, loff_t *ppos)`
  フレームバッファデバイスに指定したメモリデータを書き込む関数

- static int fb_ioctl(struct inode *inode, struct file *file, unsigned int cmd, unsigned long arg)
  フレームバッファデバイスに対するI/Oコントロールを行う関数

- static int fb_mmap(struct file *file, struct vm_area_struct *vma)
  フレームバッファデバイスの処理用のメモリ領域を確保する関数

これら関数の利用は，fb_fops構造体にて定義されている (図5.12)．fb_fops構造体はフレームバッファLKMで定義されている，フレームバッファデバイスに対する操作の関数を指定するものである．

```
static struct file_operations fb_fops = {
    .owner = THIS_MODULE,
    .read = fb_read,
    .write = fb_write,
    .ioctl = fb_ioctl,
#ifdef CONFIG_COMPAT
    .compat_ioctl = fb_compat_ioctl,
#endif
    .mmap = fb_mmap,
    .open = fb_open,
    .release = fb_release,
#ifdef HAVE_ARCH_FB_UNMAPPED_AREA
    .get_unmapped_area = get_fb_unmapped_area,
#endif
};
```

図 5.12: fb_fops構造体

フレームバッファデバイスで利用できる関数は，fb_readやfb_write，fb_mmapなどメモリデバイスと同等の機能を提供する．第5.6.4項で述べた通り，フレームバッファデバイスはメモリデバイスと同様に扱うことができる．これら関数を利用することで，フレームバッファデバイスの内容を読み取ることや，フレームバッファデバイスにディスプレイ出力を書き出すことが可能である．

5.6.6 デバイス固有のデバイスドライバ

Linuxフレームバッファは，フレームバッファメモリのカーネルモジュールfb.ko単独では動作しない．fb.koはフレームバッファデバイスを共通して扱う機構を提供す

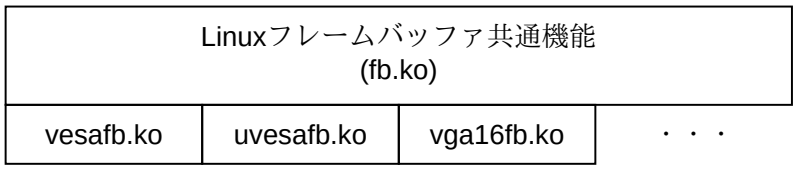

図 5.13: フレームバッファデバイスドライバ

るのみであり，フレームバッファを利用できる各デバイスに依存する機能は記述されていない．

フレームバッファデバイス固有のデバイスドライバは，Linux カーネルの `drivers/video/` ディレクトリ以下に用意されている他，利用者によって別途ファイルをダウンロード・コンパイルすることも可能である．Linux の基本的な構造は第 5.6.2 項で述べたサブシステムや，第 5.6.3 項で述べた LKM を用いるものである．フレームバッファに関しても同様で，フレームバッファの基本的な機能を持つカーネルモジュールと，デバイス固有の機能を持つデバイスドライバ (図 5.13) によって構成されている．このような構成を持つことで，デバイスドライバの開発者はフレームバッファに共通する機能については実装を行う必要がなく，開発者の手間を減らす設計となっている．

5.7 機能要求

本章を踏まえた上での Display over IP が必要とする機能を以下に整理する．

- 解像度調整
- 色数調整
- 表示位置調整
- ネットワークを介したデータの送受信

ネットワークを介したデータの送受信には，以下のことが必要である．

- 接続の確立
- 解像度等の設定情報の同期
- 表示内容の転送
- 接続の切断

本論文では，実装を Linux 2.6 上のフレームバッファシステムを利用して実現する．フレームバッファシステムは多数のデバイスに汎用的に利用でき，また動作中の OS で動的な利用の開始ができることから，本論文の目指す実装の要求を満たしている．

第6章　設計

本研究の実現にあたり，プロトタイプを Linux 2.6 上で設計・実装する．本章ではプロトタイプの設計について述べる．

6.1　基本設計

本研究における Display over IP は，第 5.6.5 項で述べた，`fb_fops` 構造体が参照する関数を改変することで実現する．`fb_fops` 構造体で参照されている `fb_read` 関数と `fb_write` 関数は，共にフレームバッファデバイスに対する読み込みと書き込みを行う関数である．これら関数を変更することで，フレームバッファ情報のネットワーク転送を実現する．`fb_read` 関数と `fb_write` 関数は，共に `fbmem.c` ファイルで定義されている．

実装形式は Linux Kernel 2.6 用 LKM とする．IP ネットワークを介してディスプレイ情報を転送するには，その宛先 IP アドレスとポート番号が必要である．Linux 起動と同時に読み込まれるカーネルモジュール形式では，宛先 IP アドレスとポート番号を指定することが難しい．また，Linux 起動時にはネットワークに接続されておらず，セッションの確立を行うこともできない．セッションの確立は，OS によるネットワークインタフェースの設定が完了してから行われる必要がある．

更に，All-IP Computer における目的として，動的にコンピュータのハードウェア環境を変更することがある．Linux 起動時に宛先 IP アドレスとポート番号を指定する設計では，動的なハードウェア環境の変更は難しい．起動と同時に読み込まれるカーネルモジュールに異なるパラメータを渡すためには Linux の再起動が必要であり，動的なハードウェア環境の構成に長時間を要してしまう．このように Linux 起動手順の問題，実装目的の問題の両観点から，Display over IP は LKM として実装することが望ましい．

デバイスを OS から認識させるためには，抽象化などの実装が併せて必要である．本研究における実装はディスプレイデバイスのネットワーク化を主眼に置き，第 4.5 節で述べたような抽象化の実現までは行わない．抽象化については今後の研究で実現を目指す．

6.2 用語の定義

本研究において実装するフレームバッファデバイスは，その動作の仕組みから構成要素の命名が必要となる．本研究では，次の用語を定義する．

- **表示ノード**:本研究における表示ノードは，接続要求を待ち受け，要求に対して処理ノードと接続を確立し，処理ノードのディスプレイ出力を表示する機器として定義する．
- **処理ノード**:本研究における処理ノードは，ユーザが実際にアプリケーション環境を利用するコンピュータを指す．

処理ノードはフレームバッファの持つデータを読み込み，表示ノードに送信する．表示ノードは受信したデータを自身のフレームバッファに書き込む．表示ノードの持つフレームバッファに書き込まれたデータは，そのディスプレイに表示される．両ノードの動作概要を，図 6.1 に示す．

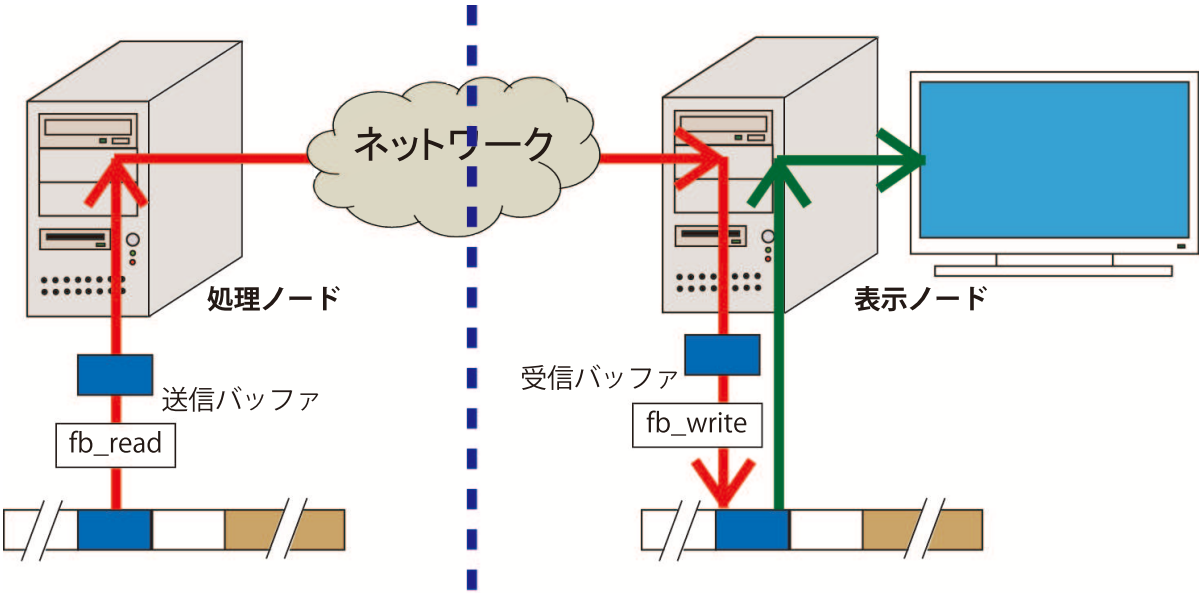

図 6.1: 表示ノード・処理ノードの動作概要

6.3 動作概要

この動作を実現するために必要であると考えられる動作シーケンスを図 6.2 に示す．まず処理ノードと表示ノードで接続を確立し，解像度や送信データのブロックサイズなどを決定する．次に両ノードで情報を共有したことを確認し，メモリデータの送信を開始する．

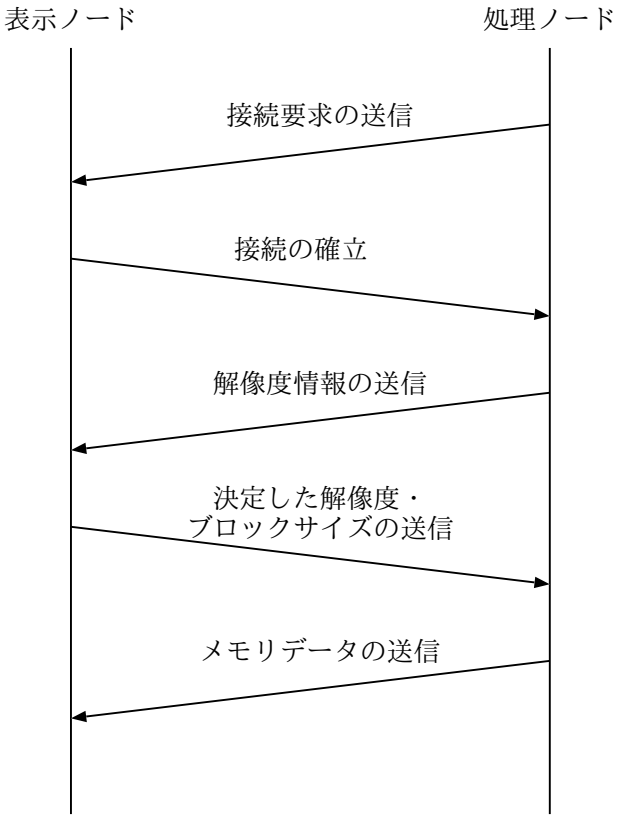

図 6.2: 表示ノード・処理ノード間における動作フロー

初期設定ではまず，処理ノードが持つ解像度情報を表示ノードに送信する．表示ノードは自身の解像度情報と受信した処理ノードの解像度情報を比較し，低い解像度に合わせて解像度を決定する．解像度を決定したら，表示ノードが利用を希望するデータブロックサイズと共に，解像度情報を返送する．両ノードは設定した解像度になるようデバイスの設定を変更し，フレームバッファメモリのデータの送受信を開始する．

解像度は，処理ノードまたは表示ノードが持つもので，低いものに設定される．これは，低い方の解像度に合わせることで，ディスプレイに何らかの情報を表示することを確実にするためである．一度設定された解像度を変更するには，`fbset` などのコマンドを用いる．解像度の変更は処理ノードで検知され，変更があった場合には表示ノードとの解像度調整が再び行われる．解像度の再設定の場合は，処理ノードで指定した解像度が利用される．この際，表示ノードで利用できない解像度を指定すると，表示が乱れるなどの不具合が生じる．

ディスプレイデバイスにおいて，解像度変更は一つの重要な機能である．処理ノードと表示ノードで利用可能な解像度が異なる場合，調整が必要である．本実装における解像度調整は，初期設定の場合は低い解像度に合わせて行い，その後は処理ノードで設定する解像度に従って行う実装となっている．

6.4 メッセージフォーマット

メッセージはデータの送受信だけではなく，接続の確立・切断や解像度の変更など，ネットワークを介したデバイスの操作に共通した形式が必要である．本実装におけるメッセージのフォーマットを図 6.3 に示す．

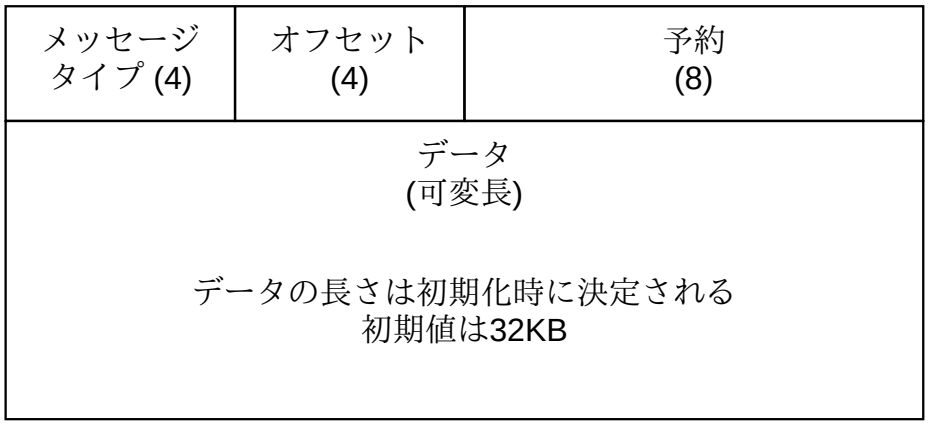

図 6.3: メッセージフォーマット

本実装におけるメッセージは，4 つの構成要素から成る．

- **メッセージタイプ**: メッセージの種類
- **オフセット**: フレームバッファメモリにおけるデータのオフセット番地
- **予約**: 現実装では利用しない，将来に向けた予約スペース
- **データ**: 該当するオフセット値からの，一度に読み込むサイズ分のデータ内容

データのサイズは，処理ノードと表示ノードで初期化を行う際に決定される．本実装における初期値は，32KB(32,768 オクテット) である．

メッセージタイプは，以下の通りとする．

- **1: FBIP_INIT**: 接続要求を実行し，送信ブロックサイズなどの通信パラメータを決定
- **2: FBIP_STOP_REQUEST**: 切断要求
- **3: FBIP_STOP_CONFIRM**: 切断確認
- **4: FBIP_NEGOTIATE_RESOLUTION**: 解像度情報を処理ノードから表示ノードへ送信
- **5: FBIP_CONFIRM_RESOLUTION**: 解像度情報を表示ノードから処理ノードへ送信

- **6: FBIP_DATA_SEND**: フレームバッファのデータ
- **7: FBIP_RESOLUTION_CHANGE_REQUEST**: 解像度変更を再度実行するよう要求
- **8: FBIP_RESOLUTION_CHANGE_CONFIRM**: 解像度変更の実行を確認

例えば，メモリオフセット 1,507,328 から 32KB のデータを送信する場合の動作を図 6.4 に示す．

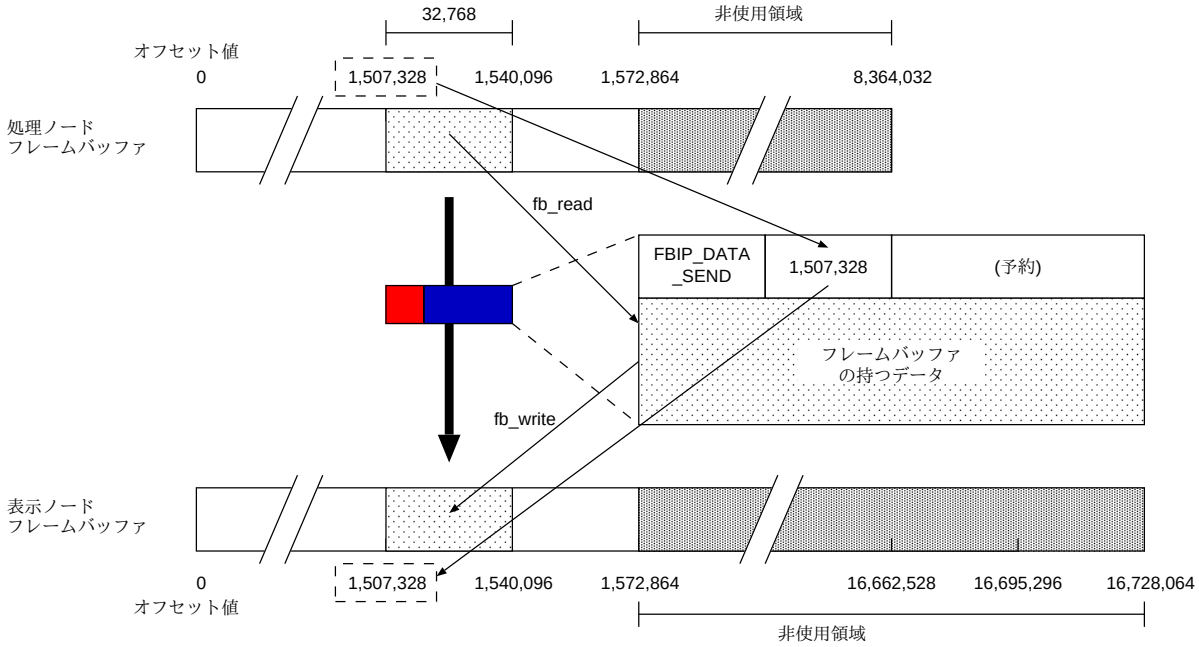

図 6.4: 送信メッセージの例

処理ノードは指定サイズ分のデータをフレームバッファメモリから読み取り，表示ノードに送信する．送信ブロックの大きさは利用者で設定できるようにしているが，今回は 32,768 (32KB) を数値として用いている．

本実装では，送信ブロックを 1 単位としてデータの比較を行っている．送信ブロック内に少しでも差分が見つかると，データは送信される．従って，送信ブロックサイズが大きすぎると差分が頻繁に発見されるため，送信ブロックサイズが大きすぎても効率は上がらない．一方で，フレームバッファデバイスに対する `read`，`write` などの処理には時間がかかる．ブロックサイズを少なく設定しすぎると処理に長い時間を要してしまう．行う処理や帯域幅にも応じて，ブロックサイズにも最適な設定が存在する．

6.5 各ノードの役割

処理ノードにおける動作フローを，図 6.5 に示す．処理ノードはフレームバッファメモリからデータを取得し，その内容を表示ノードに送信する．

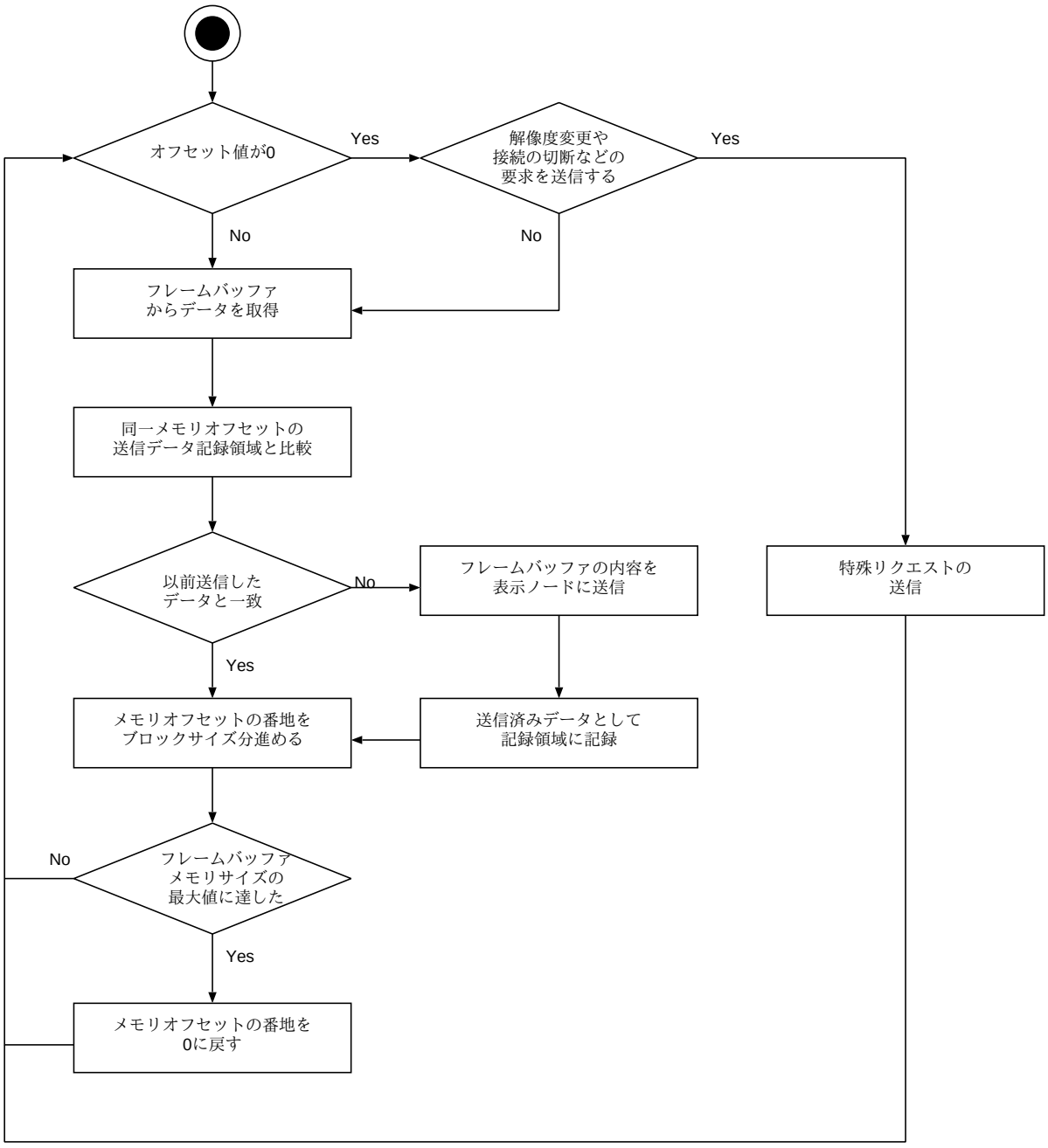

図 6.5: 処理ノードの動作フローチャート

また，表示ノードにおける動作フローを，図 6.6 に示す．表示ノードは処理ノードから受信したデータに基づき，その動作が変化する．メッセージのタイプフィールドにフレームバッファのデータが含まれていることを示す FBIP_DATA_SEND が含まれている場合，受信したデータの内容を自身の持つフレームバッファメモリの指定オフセット値に書き出す．一方，それ以外のメッセージタイプが含まれている場合，受信したメッセージタイプに応じた動作を行う．

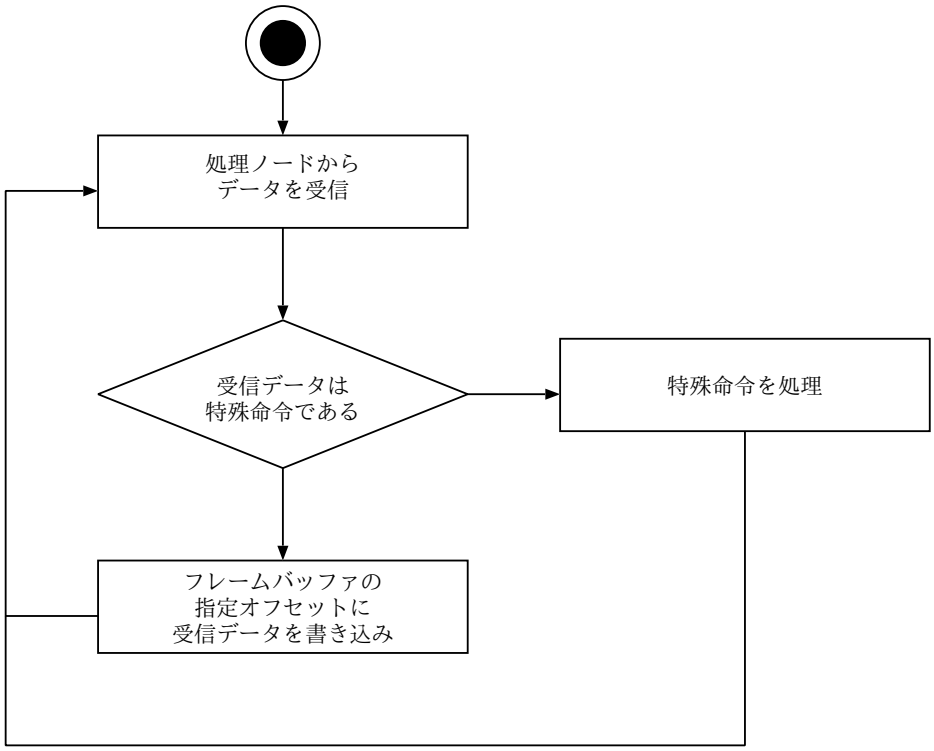

図 6.6: 表示ノードの動作フローチャート

6.6 最適化の必要性

デバイスをネットワーク経由で利用するためには，ネットワークの持つ特性に耐える設計とする必要がある．

6.6.1 送信データ量の削減

フレームバッファメモリの容量は各ハードウェアによって異なるが，その使用量は

$$(\text{X 方向の解像度} \times \text{Y 方向の解像度} \times \text{色数}) \div 8$$

という式で求めることができる．フレームバッファメモリの使用量は，その解像度や色数，描画補助機能の有無などの設定が同一であれば，異なるハードウェア上であっても等しい数字となる．

表 6.1 は，画面 1 フレームに必要と想定されるデータ量を表したものである．この数字は 1 フレームあたりのデータ量であるが，コンピュータディスプレイは 1 秒間に約 60 フレーム表示している機器が一般的である．また，仮に 60 フレーム表示しないとしても，映像などをコンピュータ上で再生するためには 30 フレーム程度の表示ができることが望ましい．例えば 1024x768 ピクセルの解像度を持ち，16 ビットの色数で表示するコンピュータの場合，

表 6.1: 代表的な解像度・色数におけるデータ量 (KB)

ピクセル数	色数 (ビット数)			
	4 ビット	8 ビット	16 ビット	24 ビット
640x480	153,600	307,200	614,400	1,228,800
800x600	240,000	480,000	960,000	1,440,000
1024x768	393,216	786,432	1,572,864	2,359,296
1280x1024	655,360	1,310,720	2,621,440	3,932,160

$$(1024 \times 768 \times 16) \div 8 \times 60 \text{ フレーム} = 94,371,840 \text{ バイト}$$

となり，毎秒約 94MB(754Mbps) のデータ転送が必要である計算になる．一般的なネットワークで利用できる帯域は最高で規格上 1000Mbps であるが，実際に利用可能な帯域は規格値よりも少ないことが一般的である．インターネットなど同一ネットワーク外のホストと通信する場合，家庭用の FTTH 接続などでは理論値 100Mbps，実効速度が 40Mbps 程度ということは珍しくない．従って，754Mbps の帯域を要求する実装では，最適化を行わなければネットワーク上でデータを送受信することは難しい．

この問題を解決するには，主として 2 種類の方法が考えられる．以前に送信したデータと比較し必要に応じて送信する送信方法の工夫と，送信データの符号化圧縮などデータサイズの縮小である．本来は両方できることが望ましいが，今回の実装では前者の，送信したデータと比較し必要に応じて送信する方法だけを実装した．

6.6.2 ネットワークを介した命令の利用

また，ネットワークを介したデバイスの利用には，既存のコンピュータの持つ機能と同等のものをすべて利用するとは限らない．ディスプレイの機能として画像の表示や解像度・色数等の調整がある．それ以外に，ネットワークを経由したディスプレイの利用として命令タイミングの調整や，ネットワーク特性への対応も必要である．

デバイスの持つ機能はローカルのコンピュータで利用することが前提となっており，ネットワーク上のデバイスでは，利用できない方が好ましい機能もある．例えば同期周波数を始めとする，ハードウェア依存の命令がある．Device over IP では，ネットワーク上のある地点に存在するデバイスを，ネットワークを介して利用する．ハードウェアには同種類のデバイスであってもそれぞれ特性があり，遠隔地にあるハードウェアの特性を正確に把握できるとは限らない．利用可能な機能を抽出し，必要な機能をネットワーク経由で利用できるようにすることが望ましい．

6.7 他技術との差異

ディスプレイのネットワーク転送そのものは，他の技術によっても実現されている．Display over IP にはこれらの技術と比較して，その抽象化手法や実現手法に共通点・

相違点の両方がある．

まず，アプリケーションレイヤでの実装がある (図 6.7)．X11 Forwarding や VNC，Remote Desktop Protocol など，ユーザによるアプリケーションの実行によって実現されるものがこれである．

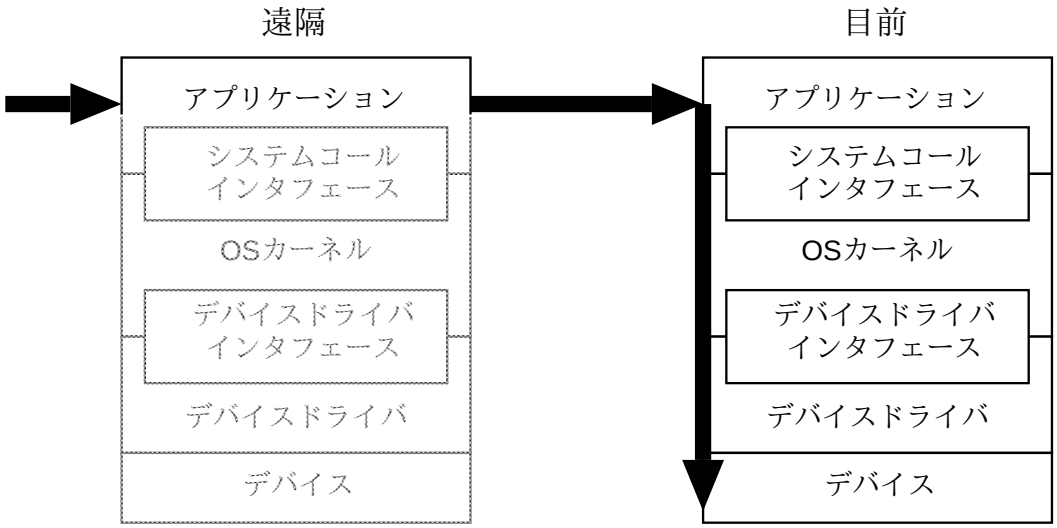

図 6.7: アプリケーションレイヤにおけるディスプレイ情報の伝送

6.7.1 X11

X11 は，UNIX システム上で多く利用されているグラフィカルユーザインタフェース (GUI) 表示システムである．実装には XFree86 [16] や X.Org [17]，その他商用製品など複数の種類が存在する．その動作は X Window System Protocol [18] で定義されている．

6.7.2 VNC

Virtual Network Computing (VNC) とは，遠隔のコンピュータシステムの画面出力をネットワーク経由で表示する技術である．その仕様は Remote Framebuffer Protocol (RFB) [19] として標準化されており，複数の実装が存在する．代表的なものには RealVNC [20] がある．

VNC は Linux システムに限らず，Windows や Mac OS X など他の OS 環境でも利用できる．ソフトウェアとして実装されており，ネットワーク環境に合わせたデータの圧縮や画面色の削減など，完成度の高い実装となっている．

6.7.3 Remote Desktop Protocol

Remote Desktop Protocol (RDP) [21] は，Microsoft Windows 向けに開発された遠隔コンピュータの画面を表示するプロトコルであり，開発も Microsoft が行っている．ITU-T.120 [22] と呼ばれる，ビデオ会議などを実装する際の標準があり，それを拡張する形で実装されている．

旧来は Windows Terminal Service と呼ばれており，主にサーバ製品を遠隔管理するためのアプリケーションであった．しかし，近年では外出先から家庭のコンピュータを操作するなどの利用方法が増え始めている．

クライアントソフトウェアは Windows XP や Windows Vista など，近年の Windows には標準で同梱されている他，Microsoft の Web ページからダウンロードすることも可能である．サーバソフトウェアは Windows Server 製品や一部のビジネス向けのクライアント OS に含まれている．また，rdesktop [23] などオープンソースのクライアントソフトウェアも少数ながら存在する．

6.7.4 USB/IP

第 3.2.1 節で挙げた USB/IP を用いても，ディスプレイをネットワーク転送することは可能である．図 6.8 のような，USB 接続規格を用いたディスプレイアダプタがいくつか製品化されている．このような USB ディスプレイアダプタを利用すれば，ハードウェアとしてディスプレイ情報を転送することは可能である．

図 6.8: USB-VGA アダプタ製品 「サインは VGA」

この手法は，図 6.9 に示したデバイスバスレイヤでのデバイス情報のネットワーク転送である．USB/IP は，デバイスをコントロールする USB ホストコントローラの持つ情報をネットワーク転送する．従って USB デバイスであればそのほとんどが，特殊な改変なしで動作することが特徴である．

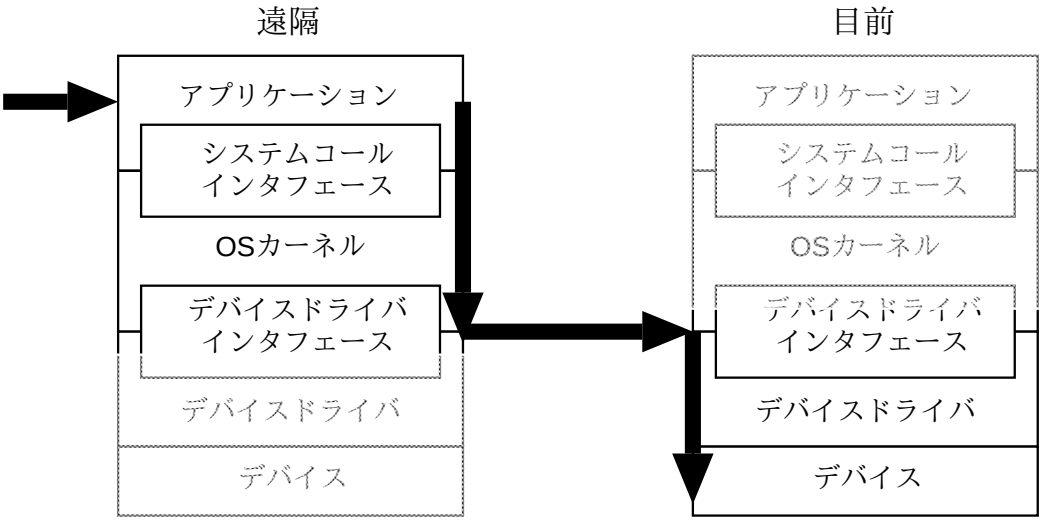

図 6.9: デバイスレイヤにおけるディスプレイ情報の伝送

6.8 実装手法

実装は，本章に挙げた要求を踏まえた上で，Linux において共通して利用されるフレームバッファカーネルモジュール (`fb.ko`) を改変することで実現する．具体的には Display over IP を実現するフレームバッファカーネルモジュールと，Linux カーネルに含まれるフレームバッファデバイスドライバを組み合わせることにより，フレームバッファデバイスドライバに共通して利用できる，ネットワークを介したディスプレイの利用機構を提供するものである．

実装では，Linux カーネルに含まれる uvesafb と呼ばれるデバイスドライバを利用する．uvesafb は VESA 規格に基づいたディスプレイデバイスであれば，異なるデバイスであっても共通して利用できるフレームバッファデバイスドライバである．今日市場で入手できるディスプレイデバイスのほとんどは，VESA 規格に準拠したものである．従って，本実装を uvesafb デバイスドライバを用いて実験することは，汎用性において有利である．

第7章　実装

第 6 章の設計を基に Display over IP の実装を行った．本章では，その実装について述べる．

7.1　実装概要

本研究の実現にあたり，Linux オペレーティングシステムにおける実装を行った．

7.1.1　実装環境

実装は以下の環境を用いて行った．

- **OS:** Debian GNU/Linux lenny (testing)
- **カーネル:** Linux Kernel 2.6.25.2
- **アーキテクチャ:** Intel x86
- **実装形態:** Linux LKM
- **実装言語・コンパイラ:** C 言語，gcc version 4.3.1 (Debian 4.3.1-2)
- **プロトコル:** TCP/IP

ネットワークプロトコルには，TCP/IP を使用した．映像配信などデータ量の多いアプリケーションでは，UDP/IP を用いることが多い．データ量が多く頻繁に表示内容が更新されることから，多少の表示の乱れが起きてもそれによる影響が比較的少ないからである．

本実装は” 画面を表示する” ことよりも，“コンピュータデバイスである” ということに着目して実装している．本研究を他のデバイスに応用することを想定した場合，UDP/IP ではパケットの損失がユーザの持つデータの損失に直結するなど，致命的な影響を及ぼす可能性がある．従って，ネットワークに対するオーバーヘッドは高くなるものの，信頼性の提供という利点を持つ TCP/IP を用いて実装した．

7.2　実装の説明

本節では，実装の構造やその中で特に考慮した点などについて述べる．

7.2.1 ノードの動作

処理ノードでは，フレームバッファカーネルモジュール上で動作するカーネルスレッドを実行し，そのカーネルスレッド中でフレームバッファメモリからデータを取得するfb_read関数を実行しつづけている．カーネルスレッドは，図7.1に示す方法で作成・実行している．

```
struct kthread_t {
    struct task_struct *thread;
    struct socket *sock;
    struct sockaddr_in addr;
    struct socket *sock_send;
    struct sockaddr_in addr_send;
    int running;
};

struct kthread_t *kthread = NULL;

kthread = kmalloc(sizeof(struct kthread_t), GFP_KERNEL);
memset(kthread, '\0', sizeof(struct kthread_t));
kthread->thread = kthread_create(fbip_proccessnode_thread_sendfb,
kthread->thread, MODABBR);
wake_up_process(kthread->thread);
```

図 7.1: カーネルスレッドの実行 (抜粋)

カーネルスレッドの作成で用いられるkthread_create関数に，カーネルスレッドとして実行する関数を引数として渡している．図7.1では，fbip_processnode_thread_sendfb関数である．

fbip_processnode_thread_sendfb関数は，最初に表示ノードとの接続を確立し，フレームバッファメモリからデータを読み出す．読み出したデータが以前に送信したデータと一致すればデータ送信は行わず，異なればデータ送信を行った上で送信したデータとして記録を行う．これを繰り返すことで，フレームバッファメモリの内容を表示ノードと共有する．

フレームバッファからのデータの読み出しと，そのデータの送信を行うソースコードを図7.2に示す．オフセット値の上限は，現在のフレームバッファメモリの上限値を計算した上で算出する．フレームバッファカーネルモジュールに定義されているfb_ioctl関数に必要な引数を渡すことで，関数が呼び出された時点での解像度や色数などの情報を取得することができる．この値を元に画面の描画に必要なビデオメモリ容量を算

出し，それに達するまでオフセット値を加算する．オフセット値が上限に達すれば0に戻し，この動作を繰り返す．

このカーネルスレッドは，表示ノードにおけるデータの受信と表示にも利用している．

7.2.2 メモリの確保

データの取得や送受信を行う上で，メモリを確保することは必要である．カーネル空間でメモリを確保する方法として，kmalloc関数がある．kmalloc関数の書式は，void *kmalloc(size_t size, gfp_t flags)である．sizeには確保するメモリのバイト数を指定し，flagsには確保するメモリの種類を指定する．本実装では，GFP_KERNELと呼ばれるメモリ確保を行っている．GFP_KERNELはLinuxカーネル内でメモリを確保する方法では，最も一般的な手法である．

kmalloc関数はカーネル空間でメモリを確保することができるため，コンテキストスイッチを伴わない動作であればユーザ空間でメモリを確保するよりも実効速度が出ると予測できる．しかし，kmalloc関数は，大きいメモリ空間を確保する上では利用すべきではないとされている．kmalloc関数で確保できる最大の連続メモリ空間はカーネルのバージョンやCPUアーキテクチャなどに応じて変化するが，移植性を求める場合は128KBを越えることは推奨されないとされている [24]．

従って，本実装では確保するメモリ空間の大きさに応じて利用する関数を切り替えている．kmalloc関数で連続したメモリ空間が確保できる場合はkmalloc，確保できない場合はvmalloc関数を用いている．vmalloc関数は効率が悪いため，できる限りkmalloc関数を用いることが望ましい．第6.3節で，データブロックのサイズは表示ノード・処理ノード間で適切なものを決定すると述べた．しかし，データブロックを格納するために確保するメモリ空間は，kmallocで確保できるサイズに納めることが望ましい．

図7.3では，sendbuf変数のメモリ空間を確保するにあたり，ブロックサイズでkmalloc関数とvmalloc関数のどちらを用いるかを判断している．

7.3 実装の実現

実装の結果，ネットワークに接続された2ホストで同一の内容を表示するディスプレイを実現した．実装が動作する様子を，図7.4に示す．

本実装と第6.7節に挙げた実装との最大の違いは，本実装はフレームバッファメモリに存在する内容であれば，どのような内容であってもネットワークを介して共有することが可能な点である．従って、VNCやX11 Forwardingでは実現できない，X11を用いないコンソールの共有も可能である．コンソールを共有している様子を，図7.5に示す．

```
int msgproto, msgsize;
loff_t cur_start;
long int screensize, csize;
unsigned long offset = 0;
char *sendbuf, reserved[8];
struct file *fb_fd;
mm_segment_t oldfs;
struct iovec iov[4];
struct msghdr msg;
struct fb_var_screeninfo var_current; mm_segment_t oldfs;
fb_ioctl(inode, fb_fd, FBIOGET_VSCREENINFO, (unsigned long)
&var_current);
screensize = var_current.xres * var_current.yres *
var_current.bits_per_pixel / 8;
while(!kthread_should_stop()) {
    cur_start = offset;
    fb_read(fb_fd, sendbuf, csize, &cur_start);
    msgproto = FBIP_DATA_SEND;
    iov[0].iov_base = &msgproto;
    iov[1].iov_base = &offset;
    iov[2].iov_base = &reserved;
    iov[3].iov_base = sendbuf;
    msg.msg_iov = iov;

    msgsize = sizeof(msgproto) + sizeof(offset) + sizeof(reserved) +
csize;
    oldfs = get_fs();
    set_fs(KERNEL_DS);
    sock_sendmsg(open_sock, &msg, msgsize);
    set_fs(oldfs);

    offset += csize;
    if(offset >= screensize) {
        offset = 0;
    }
}
```

図 7.2: 処理ノードのフレームバッファデータの送信 (抜粋)

```
#define MAX_SIZE_FOR_KMALLOC 131072
int csize;
char *sendbuf;

if(csize > (MAX_SIZE_FOR_KMALLOC) {
    sendbuf = vmalloc(csize);
} else {
    sendbuf = kmalloc(csize, GFP_KERNEL);
}
```

図 7.3: kmalloc 関数と vmalloc 関数の使い分け

図 7.4: 実装が動作する様子

図 7.5: コンソールの共有

第8章　評価

本章では，実装を用いた評価を行う．評価は他技術との比較，フレームレート，データ量と利用帯域について行う．評価では FreeBSD Dummynet Bridge を用い，遅延や帯域幅などのパラメータを変更しながら考察を行う．

8.1　評価環境

本研究における評価環境を，図 8.1 に示す．また，評価に利用した機器の構成は表 8.1，Dummynet Bridge に使用した PC サーバの構成は表 8.2 の通りである．

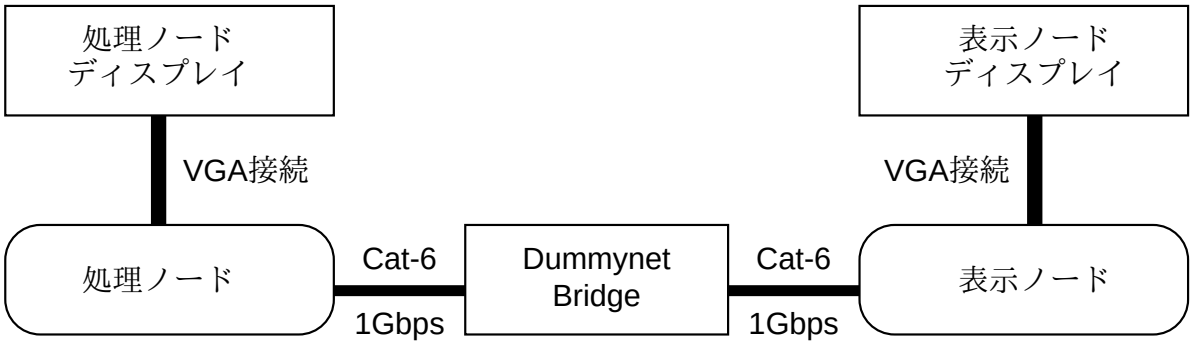

図 8.1: 評価環境

処理・表示ノードの両ノードには Debian GNU/Linux lenny をインストールし，カーネルを別途再構成・再コンパイルした．使用した Linux カーネルのバージョンは，2.6.25.2 である．

カーネルの再構成において，フレームバッファに関わるカーネルモジュールを LKM として設定する必要がある．現在の Linux の実装において，OS 起動時にカーネルモジュールを読み込む順番は，フレームバッファデバイスがネットワークデバイスより先である．また，表示ノードの IP アドレス及びポート番号は変更される場合があるため，必要に応じて利用時に指定できる方が望ましい．本実験で使用したカーネルのフレームバッファに関する設定を，図 8.2 に示す．

ディスプレイの解像度は，特に述べない限り解像度 1024 ピクセル× 768 ピクセル，16 ビットカラーとする．

表 8.1: 評価環境

	表示ノード	処理ノード
CPU	Intel Celeron D 331 2.66GHz	Intel Core 2 Duo E6750 2.6GHz, デュアルコア
メモリ	PC2-4200 DDR2 SDRAM 512MB	PC2-6400 DDR2 SDRAM 4GB
HDD	Western Digital WD800JD-08LSA0 Serial ATA 150, 7200rpm	Hitachi Deskstar HDS728080PLA380 Serial ATA 150, 7200rpm
ディスプレイ デバイス	Intel Graphics Media Accelerator 900 (Intel 910GL Express)	Intel Graphics Media Accelerator 3100 (Intel G33 Express)
ネットワーク アダプタ	Realtek RTL8169 PCI Gigabit Ethernet	Realtek RTL8168B/8111B PCI-E Gigabit Ethernet
OS	Debian GNU/Linux lenny (testing) Kernel 2.6.25.2	

表 8.2: Dummynet Bridge の環境

CPU	AMD Athlon 64 3500+ (2.2GHz, 512KB Cache)
メモリ	PC2-5300 DDR2 SDRAM 2GB
ネットワークアダプタ	Broadcom BCM5721
	Realtek 8169S PCI Gigabit Ethernet
OS	FreeBSD 7.0-RELEASE (amd64)

8.2 評価の手法

評価の手法として送信したフレーム数を計測し，それを経過した時間で割ることにより 1 秒あたりのフレーム数を算出する．図 8.1 に示した評価環境の FreeBSD Dummynet Bridge のパラメータを変更することにより，ネットワーク環境の帯域および遅延を変化させる．

ベンチマークテストでは処理ノードはフレームバッファメモリからデータを読み取り，以前送信した内容と比較する．比較した結果データに変更があった場合は，そのデータを送信する．この作業を画面 1 フレーム分行い，次に送信フレームの累計を加算する．この流れを繰り返すのが，本評価におけるベンチマークテストである．

ベンチマークテストには，図 8.3 のアニメーションを使用する．アニメーションは Adobe Flash 形式で，Web ブラウザである Mozilla Firefox 上に表示させた．

アニメーションは幅 500 ピクセル×高さ 500 ピクセルの範囲を，直径 100 ピクセルの円が毎秒 30 フレームの速さで移動するものである．横運動と縦運動を均等に行うよう，移動は斜めに行った．

本評価では，この試験を解像度 1024 ピクセル× 768 ピクセル，16 ビットカラーの環

```
Device Drivers  --->
      Graphics support  --->
            <M> Support for frame buffer devices  --->
                  --- Support for frame buffer devices
                  [*]   Enable firmware EDID
                  -*-   Enable Video Mode Handling Helpers
                  [*]   Enable Tile Blitting Support
                        *** Frame buffer hardware drivers ***
                  <M>   Userspace VESA VGA graphics support
               Console display driver support --->
                  -*- VGA text console
                  [*]   Video mode selection support
                  <M> Framebuffer Console support
                  [*]   Framebuffer Console Rotation
```

図 8.2: カーネル中のフレームバッファの設定

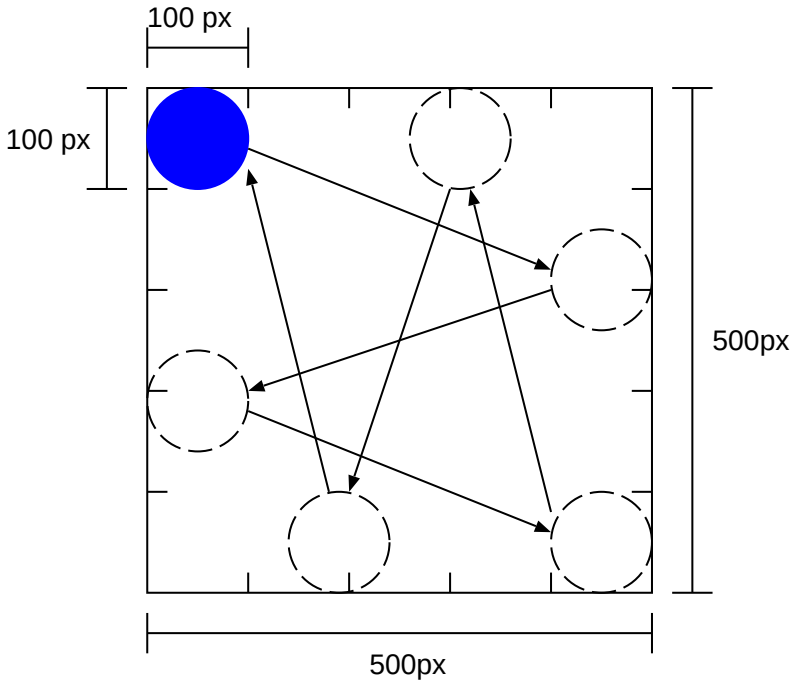

図 8.3: ベンチマーク用アニメーションの動作

境で 2 分間行い，5 回実行した．

8.3 遅延による影響

Dummynet Bridge の遅延時間を変更することで得られた結果を，図 8.4 に示す．

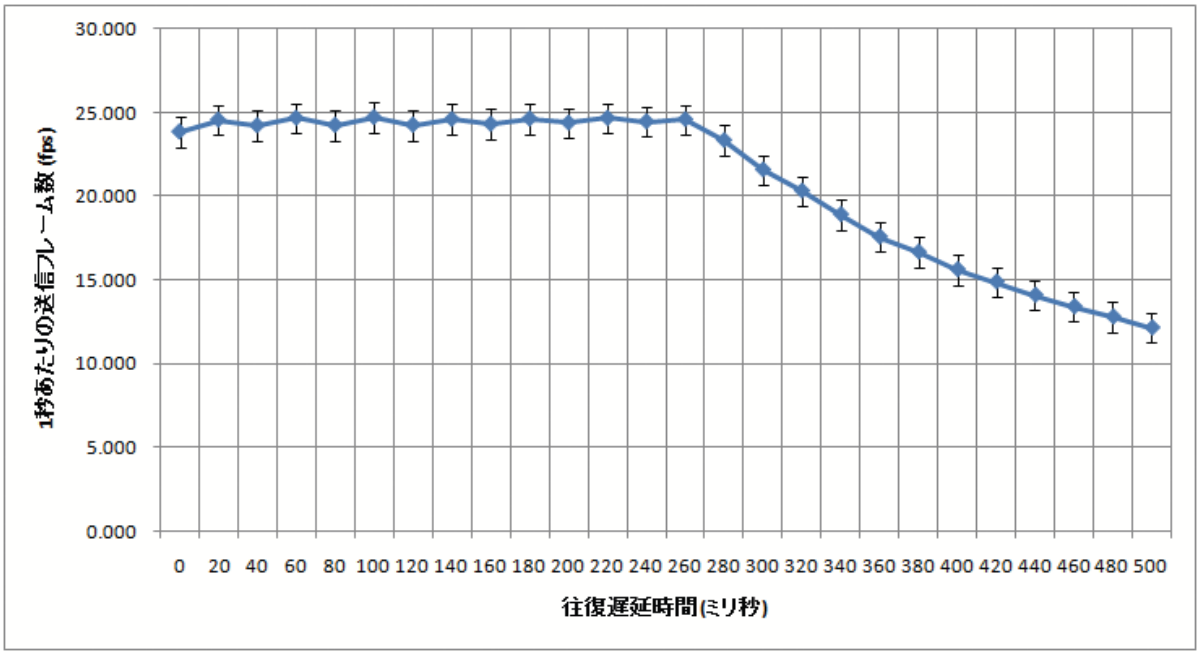

図 8.4: RTT の変化による本実装の送信フレーム数の変化

実験の結果，RTT 260 ミリ秒程度までは安定してフレーム数が送信できた．RTT 260 ミリ秒程度の環境とは，海外との通信にかかる遅延と同等か，それよりも長い時間である．従って，遅延だけに着目をするとヨーロッパ程度までであれば十分利用できることが分かる．論文執筆時点でのおおよその RTT を，表 8.3 に示す．

表 8.3: 論文執筆コンピュータから Debian サーバへの RTT (論文執筆時点)

国名	URL	RTT
日本	`www.jp.debian.org`	平均 4ms
米国	`www.us.debian.org`	平均 170ms
フランス	`www.fr.debian.org`	平均 270ms
ロシア	`www.ru.debian.org`	平均 330ms

本実装を使用した際の帯域使用量と往復遅延時間の関係を，図 8.5 に示す．帯域使用量は送信フレーム数と比例しており，遅延が大きくなるにつれて帯域使用量は減少する．

本実装は TCP をトランスポートプロトコルとしているが，遅延が大きくなるにつれて遅延 ACK の送信数が多くなるなどの傾向が見られる．実験環境において，処理ノー

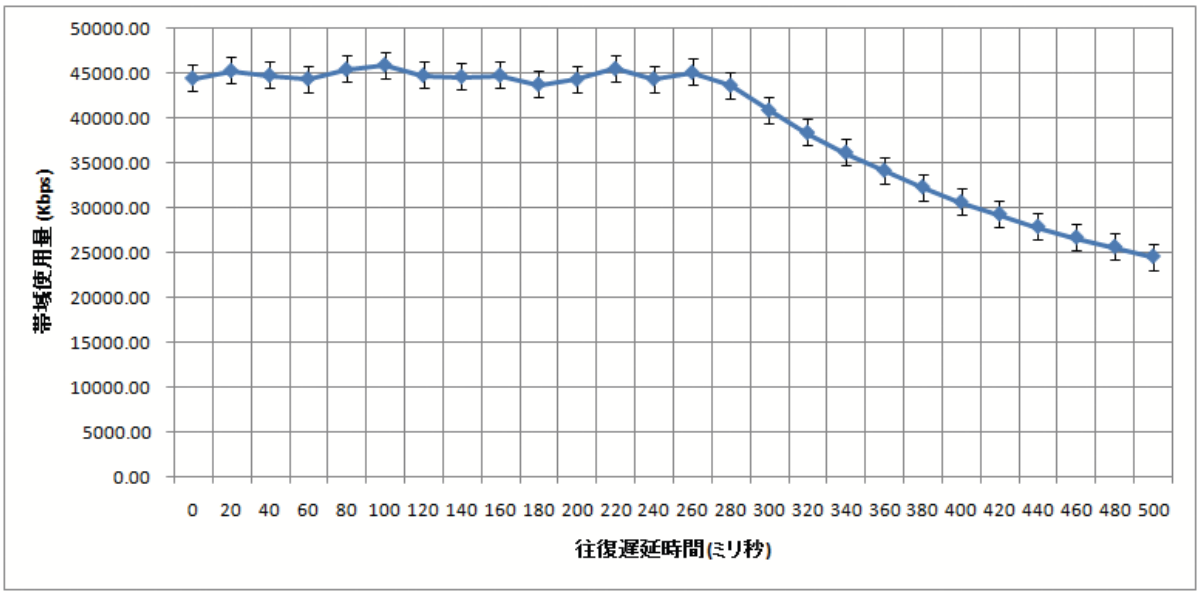

図 8.5: 本実装使用時の帯域使用量と往復遅延時間の関係

ドの TCP バッファサイズは 4,194,304 バイト，表示ノードのバッファサイズは 2,027,520 バイト確保されている．利用できるネットワーク資源の減少に伴い，ACK 送信に時間がかかるなどの現象が生じ，フレームは送信できているものの応答までに時間がかかるというものである．

本実装において TCP を用いる理由は，デバイスとしての信頼性を持つべきという点にある．転送されるべきデバイスの情報は，パケットロスなどで紛失するなどといったことが生じてはならない．デバイス情報は” 処理” されるものであり，デバイス情報の送受信は確実に行われる必要がある．

TCP では，例えばバッファサイズの変更など，パラメータチューニングを行うことが可能である．そのようなパラメータチューニングを行うことにより，より制約の高いネットワーク環境においての利用ができる可能性がある．本実験を行った際，表示ノードのソケットバッファサイズは 2,027,520 バイトであった．例えば帯域使用量 60Mbps で RTT が 260 ミリ秒あった場合，毎秒約 1.95MB のデータ転送が発生する．データの受信や ACK の返信が追い付かず，ソケットバッファにデータを格納しきれない場合があると想定できる．このような場合には，TCP のパラメータを，データの転送量やネットワークの状態に合わせて動的に調整することでパフォーマンスの向上を図ることが考えられる．

8.4 帯域幅による影響

Dummynet Bridge の帯域幅を変更することで得られた結果を，図 8.6 に示す．

結果より，帯域を 60Mbps 以上利用できる場合は安定してフレームを送信し，それ

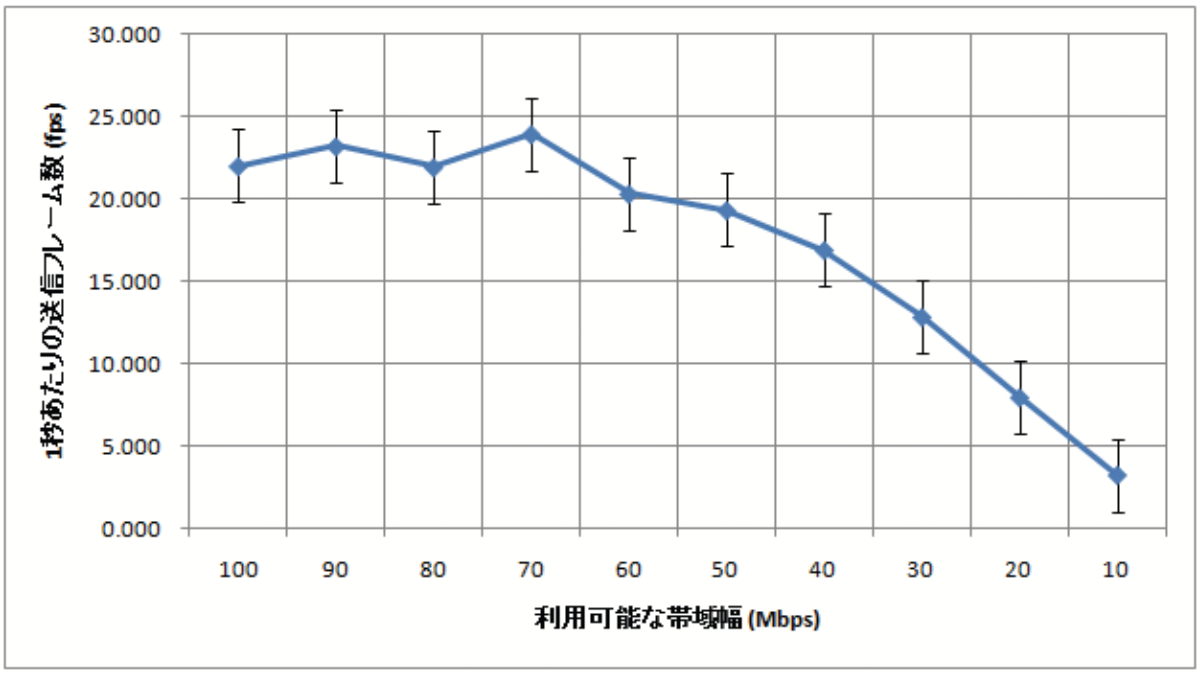

図 8.6: 帯域幅の変化による本実装の送信フレーム数の変化

よりも帯域幅が狭い場合は帯域幅と比例関係を持つことが分かる．その結果は図 8.6 からも得られる．通常の帯域使用量として，40-50Mbps 程度で推移している．従って利用可能な帯域幅がこの範囲を下回るまでは，安定したフレーム送信が可能である．

今回の評価における動作は，日本における一般家庭の FTTH 接続で多い実効スループットである 30-40Mbps 程度であっても毎秒 15 フレーム程度で動作し，利用するにあたっては問題のない速度で動作することになる．その後，20-30Mbps の範囲で毎秒 10 フレーム程度まで下落し，そのフレーム数でデスクトップ環境を利用することは利用者にとってストレスを感じさせる程度にまでなる．

8.5 他技術との比較

”ディスプレイを転送する”技術には，第 6.7 節に挙げた通り様々な実装が存在する．その中でも特に，“ハードウェア情報を転送する” ことから本実装の方向性と近い，USB/IP 上の USB-VGA アダプタに着目して比較する．なお，本実験で利用した USB-VGA アダプタは KAIREN 社の” サインは VGA”(緑箱，USB20SVGA-DG) である．

8.5.1 評価環境

比較対象として挙げた内，今回使用した USB/IP を用いた USB-VGA アダプタ製品による X11 の起動は，解像度 640 ピクセル× 480 ピクセルの環境でのみ可能であった．

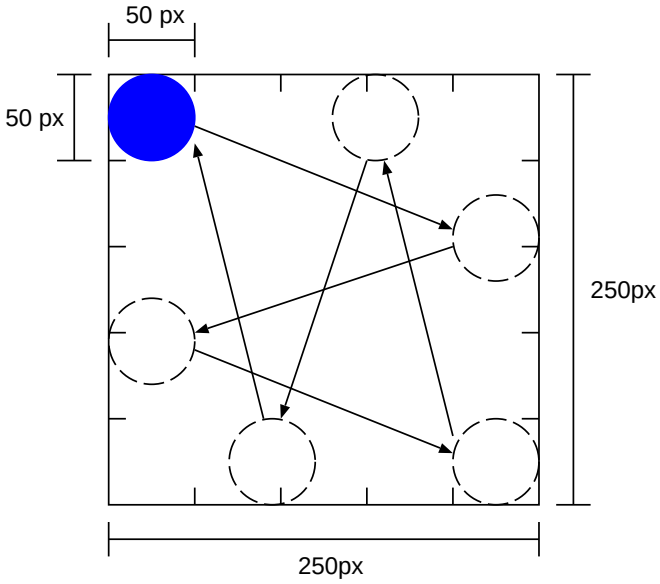

図 8.7: 縮小版ベンチマーク用アニメーションの動作

従って，本節における評価では，評価用アニメーションが画面からはみ出さないように縮小したものを用いている (図 8.7)．

8.5.2 比較データ

本実装と USB/IP 上の USB-VGA アダプタにおける，実験時の帯域使用量と遅延の関係を図 8.8 に示す．

結果より，USB/IP は遅延が生じると共に帯域使用量が減少していることが分かる．このことは本実装においても同様であるが，帯域使用量が減少し始める遅延時間が異なっている．

一方で，帯域幅と帯域使用量の関係を示したものが図 8.9 である．USB/IP 上の USB-VGA アダプタは，今回の実験で利用した環境では USB 1.1 互換の速度で動作している．従って，規格上の速度は約 12Mbps である．12Mbps の帯域幅で転送できるデータは，単純計算で

$$(12Mbit \times 1024 \times 1024) \div (1024px \times 768px \times 16bit) = 1$$

であり，毎秒 1 フレームという計算になる．実際には USB-VGA アダプタがフレームを表示する際には，ブロック単位で出力し，不要な再描画は行わないためこれ以上のフレーム数は確保できている．

遅延や帯域幅による帯域使用量への影響は，一定時間内に転送を完了したフレーム数が減少することを意味している．遅延の増加や帯域幅の低下により，1 フレームを送信するために必要な時間は長くなる．実際に転送可能な情報量が低下しているため，それに合わせて帯域使用量も減少していると推測できる．

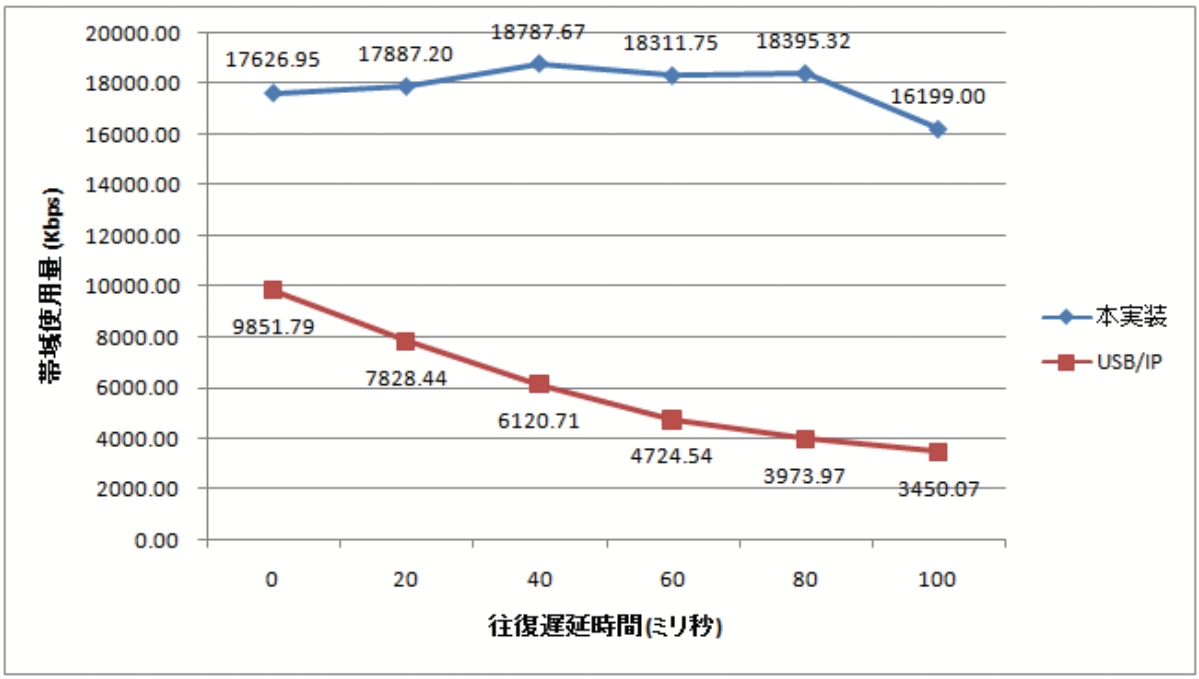

図 8.8: 本実装と USB/IP の帯域使用量と遅延の関係

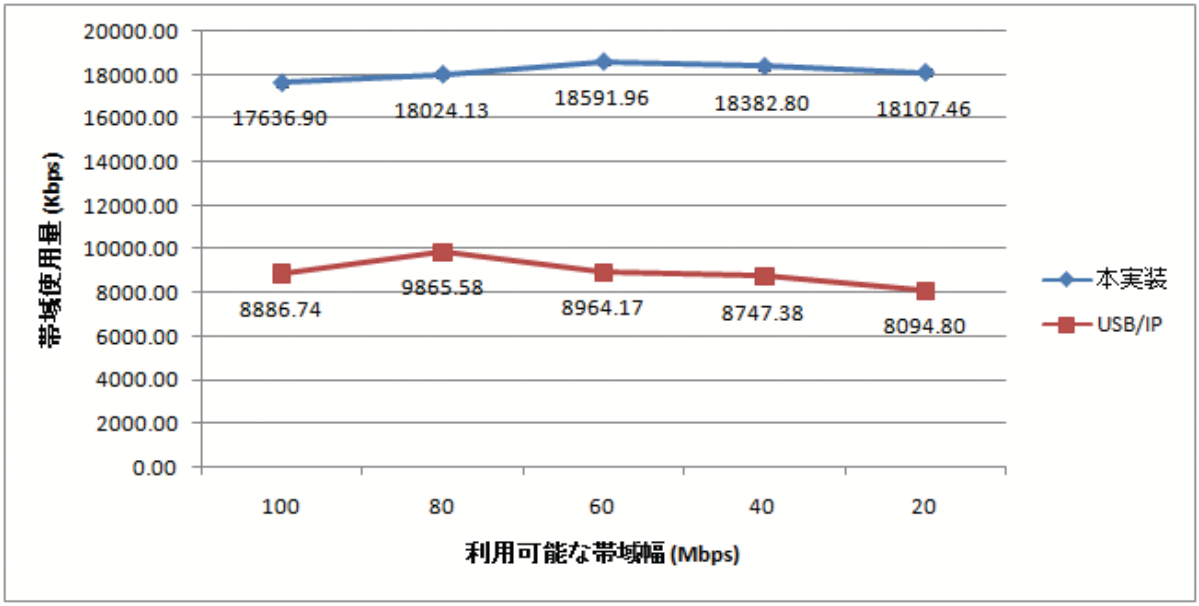

図 8.9: 本実装と USB/IP の帯域使用量と帯域幅の関係

8.5.3 動作の比較

本実装と USB/IP 上の USB-VGA アダプタでは，その画面描写や性能にいくつかの違いがある．まず，USB-VGA アダプタは主に X11 Window System を表示するために

使用できるが，tty1 などのコンソール画面を表示することは出来ない．本実装はフレームバッファモードで動作している限り，どのような画面でも遠隔から表示することが可能である．

次に、遅延が生じた際の動作が異なる．USB/IP は USB バスプロトコルの情報すべてを IP パケットにカプセル化し，送信する．従って，USB/IP を用いた USB-VGA アダプタでは，ディスプレイ出力デバイスとして全ての機能を不備なく利用できるとは限らない．例えば，遅延が大きくなったとき，USB/IP 上の USB-VGA アダプタでは図 8.10 のように，斜めに動くはずのマウスポインタが横・縦の運動を繰り返して描画される．これは図 8.8 及び図 8.9 に示した通り，遅延や利用可能な帯域幅によって送信できる情報が少なくなるためである．Linux におけるマウスポインタの移動は，X 軸方向・Y 軸方向の 2 つの情報によって行われている．ローカルコンピュータ上で実行する場合は両方の処理がほぼ同時に行われ斜めに動くように見えるが，遅延によって”X 軸方向の動作” と”Y 軸方向の動作” の間に時間差が生じてしまう．そのため，カーソルが斜めに動かず，階段状に動く．

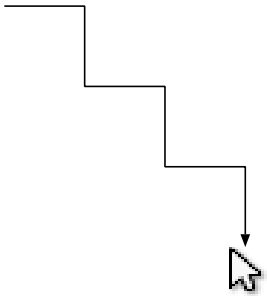

図 8.10: 遅延が大きくなったときの USB/IP USB-VGA アダプタのマウスポインタ

この遅延は，USB と USB/IP の仕様が原因であると考えられる．USB ではコントローラからデバイスへ命令が送信され，デバイス上でその命令に基づく処理が行われた後，確認応答がコントローラへ送信される．遅延が大きくなると遠隔地の USB デバイスへ送信した処理に対する，確認応答に要する時間が長くなる．すると，その後に続く処理が発行されるまでにかかる時間のずれが大きくなり，ディスプレイのように継続的に処理を続けるデバイスでは次々と発行待ちの命令が並ぶことになる．ディスプレイの表示が遅れてしまうため，利用者が意図したものとは異なった内容が表示されてしまう．本実装ではデバイスとしてデータを確実に配送するために信頼性を持つ TCP をトランスポートプロトコルとしているが，ディスプレイ表示の機能そのものはなるべく高速に表示することを主眼におき，ディスプレイ表示に特化した内容の確認はしていないためである．表示という機能に着目した動作をしているため，汎用的にバスを IP 化している USB/IP より，ディスプレイ表示という機能においては優位であると言える．

USB/IP に代表されるバスプロトコルの IP 化は，USB デバイスという幅広いデバイス群を IP 化できることから応用性は高い．しかし，そのデバイス群の幅広さから，必

ずしも全てのデバイス固有の要求に基づいた性能を発揮できるとは限らない．データ量の小さいデバイスや転送速度を要求しないデバイスであれば USB/IP でも不足なく動作する．しかし，ディスプレイデバイスというデータ量も大きく，即時的な転送を要求するデバイスでは，専用の機構を持つことが必要となる．

8.6 まとめ

遅延および帯域は，どちらに関しても利用できるネットワーク資源が減少すると，ユーザの操作に対する描画が遅くなるなどの現象が見られた．これは，表示処理に時間がかかっていることを意味している．

Device over IP で重要な点として，デバイスの特性を考慮した上でのデバイスの IP 化がある．本実装ではデバイス専用に設計されたバスプロトコルの，“広帯域・低エラー”という特徴に倣い，信頼性を提供する TCP をトランスポートプロトコルとして選定した．しかし，例えばデバイス情報の一部が紛失してもその機能を持続可能とするような設計にすることで，UDP をトランスポートプロトコルとして実装するモデルも考えられる．信頼性と速度のどちらを重視すべきかは，デバイスの特性や利用者の要求によって異なる．このような点を踏まえた実装を行うことが，All-IP Computer におけるデバイスの IP 化には必要である．

第9章　結論

9.1　まとめ

本研究では，All-IP Computer における入出力デバイス，特にディスプレイ出力装置としてのフレームバッファメモリのネットワーク転送を実現した．All-IP Computer におけるバスは，デバイス専用の規格ではなく IP ネットワークである．IP ネットワークを介してフレームバッファメモリの内容を送受信することで，遠隔コンピュータの持つフレームバッファメモリの内容をディスプレイ出力として表示することを実現した．

コンピュータを構成するデバイスは，各々の接続規格に基づいて設計・開発されている．デバイス接続規格には PCI などコンピュータ内部で利用する内部バスや，USB や IEEE1394 などの外付け周辺機器を接続するための外部バスがある．これらデバイスを接続するバスも一種の”ネットワーク”であると言える．そのネットワークに存在するデバイスがコンピュータという”箱”の枠組みから外れたとき，デバイスが IP ネットワークに接続する All-IP Computer という概念が成立する．

本研究はディスプレイ出力装置の研究であるが，ディスプレイ表示を IP ネットワークで転送する技術は既に存在する．VNC，X11 Forwarding，Remote Desktop Protocol などがそれである．これらはいずれもソフトウェアとして実装されており，広く使われている技術である．また，Device over IP を実現する技術に，USB/IP や iSCSI といった技術がある．それら技術は既存のデバイスを IP ネットワークに接続することは実現できるが，IP ネットワークには専用バスと比較した際の速度，遅延，パケットロスといった問題がある．既存デバイスの情報すべてをネットワークと送受信することは，非効率的な場合が多いと考えられる．各デバイスには，そのデバイスに定められた仕様があり，デバイスの要求する機能を満たした上でのネットワーク転送の方が，効率性は高まると考えられる．

本研究の実装は，フレームバッファメモリの内容を転送することで実現している．この実装を使用するためには，Linux カーネル上で利用できるフレームバッファデバイスが必要である．環境が明確である分、実現しなくてはならない要件を把握・実装することが可能であった．このようにデバイスの特性を把握し，それを利用したデバイスの IP 化が，All-IP Computer というデバイスバスに IP を用いるコンピュータを実現することに有効である．

9.2 今後の展望

本研究は，All-IP Computer を実現する一部分にすぎない．”すべてのバスを IP で接続する”All-IP Computer の実現には，より多くのデバイスの IP 化が必要である．本研究で用いたフレームバッファメモリは，Linux においてはキャラクタデバイスとして認識・利用される．本研究で利用したフレームバッファメモリの実現機構を，別のデバイス上で応用することは十分可能である．

例えば，フレームバッファデバイスの処理はメモリデバイスの処理と似ている部分がある．実際に Linux カーネルのドキュメンテーションには，フレームバッファメモリの操作は通常のコンピュータ上のメモリである`/dev/mem`と同様に行えるとの記述がある [25]．従って，例えばコンピュータ間でメモリ空間を共有するといった実装が，実現する可能性もある．今回行った実装が他のキャラクタデバイスに応用することが考えられ，より多くのデバイスの IP 化を今後の展望として捉えることができる．

コンピュータにおいて，ディスプレイ出力デバイスは実現が難しいデバイスの一つである．その膨大なデータ量から，他の実装では解像度や色数を減らしてネットワーク転送する情報量を減らしたり，データの送受信時に圧縮・伸長などの最適化をするものが多い．よりデータ量の少ない別のデバイスを実装することは，本研究よりデータ量を考えない分，また異なる実装となることが考えられる．

デバイスの IP 化には，デバイス毎に要求される要件がある．本研究ではディスプレイ出力デバイスに着目し，その要件について検討した．実装・評価を通して，信頼性と速度のどちらを重視するかは，Device over IP における課題の一つであることが確認できた．他のデバイス化の研究では，また異なる要求がある．それら要求に対応し，必要な機能の抽出やパラメータの設定を選定した上で，各デバイスを抽象化していくことが All-IP Computer の実現に必要である．

謝辞

本論文の作成にあたり，ご指導頂いた慶應義塾大学環境情報学部教授 村井純博士，同学部教授 徳田英幸博士，同学部教授 中村修博士，同学部准教授 楠本 博之博士，同学部准教授 高汐一紀博士, 同学部准教授 三次仁博士，同学部准教授植原啓介博士，同学部専任講師 重近範行博士，同学部専任講師 中澤仁博士，同学部専任講師 Rodney D. Van Meter III 博士に感謝致します．

Toyota ITC Senior Researcher，慶應義塾大学政策・メディア研究科講師 湧川隆次博士に感謝致します．湧川博士には All-IP Computer の研究を最初にご指導頂いた他，研究室内外での発表等においてもご指導頂きました．

慶應義塾大学政策・メディア研究科講師 吉藤英明博士に感謝致します．吉藤博士には実装面で数多くの助言を頂きました．吉藤博士の指導・助言なしでは実装の完成度を向上させることはできませんでした．

慶應義塾大学政策・メディア研究科後期博士課程 岡田耕司氏に感謝致します．氏には論文全体の指導から実装の指導まで，お忙しい中多くの時間を割いてご指導頂きました．氏の指導なしでは，本論文を完成させることは出来ませんでした．ありがとうございました．

慶應義塾大学総合政策学部 中里恵氏，同大学環境情報学部 黒宮佑介氏，峯木厳氏，立石幹人氏に感謝します．皆さんも卒業論文の執筆，お疲れさまでした．

慶應義塾大学政策・メディア研究科助教 佐藤雅明氏，同研究科博士課程 石原知洋氏，海崎良氏，堀場勝広氏，田崎創氏，工藤紀篤氏，久松剛氏，松園和久氏，水谷正慶氏，松谷健史氏，同研究科修士課程 金井瑛氏，空閑洋平氏，奥村祐介氏，遠峰隆史氏，本多倫夫氏，佐藤龍氏，同研究科卒業生 大藪勇輝氏，中村友一氏に感謝致します．同大学総合政策学部 上原雄貴氏，永山翔太氏，同大学環境情報学部 波多野敏明氏，永井ゆり氏，三部剛義氏，中村遼氏，同大学卒業生 尾崎隆亮氏に感謝します．皆様の助言・お話・雑談・外出・食事など，様々な場面で心の支えになって頂きました．

ARCH KG を始め，研究室での活動を支えてくださった合同研究会の皆様に感謝致します．皆様のご協力なくては，本論文は完成できませんでした．ありがとうございました．

平成 21 年 2 月 10 日
六田 佳祐

参考文献

[1] Google, Inc. Google Apps. http://www.google.com/a/.

[2] ThinkFree. ThinkFree Online. http://www.thinkfree.com/.

[3] StartForce, Inc. and フュージョン・ネットワークサービス株式会社. StartForce. http://www.startforce.com/.

[4] Samba Team. Samba. http://www.samba.org/.

[5] Mark Hayter and Derek McAuley. The Desk Area Network. *ACM SIGOPS Operating Systems Review*, Vol. 25(No. 4):pages 14–21, May 1991.

[6] Henry H. Houh, Joel F. Adam, Michael Ismert, Christopher J. Lindblad, and David L. Tennenhouse. The VuNet Desk Area Network: Architecture, Implementation, and Experience. *IEEE Journal of Selected Areas in Communications*, Vol. 13(No. 4):pages 710–721, May 1995.

[7] Gregory G. Finn. An Integration of Network Communication with Workstation Architecture. *ACM Computer Communication Review*, 21(5):18–29, October 1991.

[8] Rob Pike, Dave Presotto, Sean Dorward, Bob Flandrena, Ken Thompson, Howard Trickey, and Phil Winterbottom. Plan 9 from Bell Labs. *Computing Systems*, Vol. 8(No. 3):pages 221–254, Summer 1995.

[9] Takahiro Hirofuchi, Eiji Kawai, Kazutoshi Fujikawa, and Hideki Sunahara. USB/IP - a Peripheral Bus Extension for Device Sharing over IP Network. *Proceedings of the FREENIX Track: USENIX Annual Technical Conference*, pages 47–60, April 2005.

[10] Takahiro Hirofuchi, Eiji Kawai, Kazutoshi Fujikawa, and Hideki Sunahara. USB/IP: A Transparent Device Sharing Technology over IP Network. *IPSJ Transactions on Advanced Computing Systems*, Vol. 46(No. SIG11(ACS11)):pages 349–361, August 2005.

[11] J. Satran, K. Meth, C.Sapuntzakis, M.Chadalapaka, and E.Zeidner. Internet Small Computer Systems Interface (iSCSI), RFC 3720. http://www.ietf.org/rfc/rfc3720.txt, April 2004.

[12] Rodney Van Meter, Gregory G. Finn, and Steve Hotz. VISA: Netstation's Virtual Internet Adapter. *ASPLOS VIII*, pages 71–80, October 1998.

[13] S. Hopkins and B.Coile. AoE (ATA over Ethernet). http://www.coraid.com/documents/AoEr10.txt, May 2006.

[14] Video Electronics Standards Association. VESA. http://www.vesa.org/.

[15] Video Electronics Standards Association. VESA BIOS Extension (VBE) Code Functions Standard Version 3.0. Technical report, Video Electronics Standards Association, September 1998.

[16] The XFree86 Project. XFree86. http://www.xfree86.org/.

[17] X.Org Foundation. X.Org. http://www.x.org/.

[18] Rovert W. Scheifler. X Window System Protocol. ftp://ftp.x.org/pub/X11R7.0/doc/PDF/proto.pdf, 2004.

[19] Tristan Richardson. The RFB Protocol. Technical report, RealVNC Ltd., June 2007.

[20] RealVNC Project. RealVNC. http://www.realvnc.com/.

[21] Microsoft Corporation. Remote Desktop Protocol. http://msdn.microsoft.com/en-us/library/aa383015(VS.85).aspx.

[22] International Telecommunication Union. Data protocols for multimedia conferencing. January 2007.

[23] rdesktop Project. rdesktop: A Remote Desktop protocol Client. http://www.rdesktop.org/.

[24] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. *LINUX* デバイスドライバ 第 *3* 版. オライリー・ジャパン, 2005.

[25] Geert Uytterhoeven. The Frame Buffer Device. *Linux Kernel Documentation*, May 2001.