USBドライバスタックを拡張したリモートデバイス利用方式

広渕 崇宏[†]　河合 栄治[‡§]　中村 豊[¶]
藤川 和利[¶]　砂原 秀樹[¶]

概要: 広帯域のネットワークインフラの普及とともに、ネットワークに直接接続されたデバイスや、ネットワーク上の計算機に接続されたデバイスが増加しつつある。しかしながら、これらネットワークを介してアクセスするデバイスの利用方法と手元の計算機に直接接続されているデバイスの利用方法との間には大きな開きがある。そのため、ユーザは両者を統一的な方法でアクセスできない。ゆえに、計算機に直接接続されているデバイスのアクセスモデルを、そのセマンティクスを維持したまま、ネットワーク上のデバイスのために拡張することが考えられる。そこで、本論文では、現在広く使用されているオペレーティングシステムのひとつである Linux の USB ドライバスタックに注目し、これを拡張することでネットワーク透過のデバイスアクセス手法を提案する。また、提案システムのプロトタイプの実装についても述べる。

キーワード: Linux、デバイスドライバ、リモートデバイス、インターネット、USB

A Device Access Method over Network by Extending USB Driver Stack of Linux

HIROFUCHI Takahiro[†]　KAWAI Eiji[‡§]　NAKAMURA Yutaka[¶]
FUJIKAWA Kazutoshi[¶]　SUNAHARA Hideki[¶]

Abstract: The spread of high bandwidth network brings about the increasing number of network enabled devices, which are directly attached to network or which are attached to network available computers. However there is a big difference between access methods to devices beyond network and to devices directly attached to local computers. Therefore users cannot access both kinds of devices by an unified method. To solve this issue, we propose an extended access model for remote devices that preserves the device access semantics. In our system, we extend USB driver stack of Linux which is one of the most popular operating systems and design device access method over network via the existing device file semantics. In this paper, we describe the design of our system and a prototype implementation.

Keywords: Linux, device driver, remote device, Internet, USB

1　はじめに

インターネットに代表されるネットワークはますます広帯域化しつつあり、我々の生活のすみずみまでインフラとして行きとどきつつある。従来であれば、計算機の周辺機器として、計算機に直接接続されているデバイスのみを考えた。しかし、ネットワークがデバイスの制御データの送受信に十分耐えるほど高速化すると、ネットワークを介して利用可能なデバイスも計算機の周辺機器とみなすことができる。これら計算機からネットワークを介して利用可能なデバイスをリモートデバイスと呼ぶ。

このようなリモートデバイスとしては、ストレージをネットワーク化した iSCSI [1] のように、従来の周辺機器にイーサネットのネットワークインタフェースを搭載することでネットワークを介した制御を可能にしているものが存在する。また、リモートの計算機上の記憶装置や入出力装置などのように、ネットワーク接続された計算機に存在することで間接的にネットワークを介したアクセスを可能にしているデバイスも、リモートデバイスとしてみなせる。

[†]奈良先端科学技術大学院大学 情報科学研究科
Graduate School of Information Science, NAIST
[‡]奈良先端科学技術大学院大学 附属図書館
Digital Library, NAIST
[§]科学技術振興事業団 さきがけ研究 21
PREST, Japan Science and Technology Corporation
[¶]奈良先端科学技術大学院大学 情報科学センター
Information Technology Center, NAIST

既存の各種UNIXオペレーティングシステム(OS)は、計算機上に存在するデバイスへのアクセスのためにデバイスファイルという抽象化されたインタフェースを提供している。しかし、リモートデバイスへのアクセスについては何ら抽象化されたインタフェースを提供していない。リモートデバイスをアプリケーションから利用するためには、専用のライブラリやミドルウェアを使用するか、ネットワークプログラミングを行う必要がある。これは、デバイスファイルを用いてデバイスにアクセスしている、既存のアプリケーションをそのまま利用することが難しいことを意味し、ソースコードの改変やプログラムの再コンパイル/再リンクを要するため、開発コストが高い。

既存研究の中には、デバイスファイルのセマンティクスでリモートデバイスを使用可能にしているものが存在する。しかし、その多くは、送受信するプロコトルが、OSのデバイス・アクセス・モデルに強く依存している。それゆえ、OSやアーキテクチャが異なるシステム間での利用が困難であり、ネットワークに直接接続するデバイスへの対応も難しい。また、既存研究の多くは対応デバイスの種類が少ない。さらに、その構造上、多種類のデバイスへの対応が困難である。

そこで、本研究では現在広く使用されているオペレーティングシステムのひとつであるLinuxのUSBドライバスタックに注目し、これを拡張することで、リモートデバイスへのアクセスを既存のデバイスファイルを用いるセマンティクスで可能にすることを提案する。

LinuxのUSBドライバスタックに注目した理由は、すでにさまざまなデバイスのためのドライバが存在する点や、その階層構造により機能拡張が容易である点、また、活線挿抜の機構がリモートデバイスにも応用可能である点、などである。

この方式により、リモートデバイスに対してもデバイスファイルのインタフェースを提供できる。デバイスファイルを用いてデバイスを制御する既存アプリケーションを、変更なしにリモートデバイスにも適用できる。また、既存研究でみられた問題点を解決でき、他の計算機上に存在するデバイスだけでなく、ネットワークに直接つながるデバイスへも応用可能になる。さらに、さまざまなデバイスへの対応が可能である。

本論文では、2節で関連する既存研究にふれた後、3節で提案方式を説明し、4節でそのプロトタイプの実装とその動作状況について述べる。そして、5節で今後の課題を述べる。

2 既存研究

本節では、デバイスファイルのセマンティクスによって、リモートデバイスへのアクセスを提供している研究事例を概観する。

2.1 RDC

RDC (Remote Device Control) [2] は、デバイスファイルに対するシステムコールをカーネル内部で他の計算機に転送することで、リモートデバイスに対するアクセスを提供する。

この方式の問題は、システムコールの引数がOSごとに少しずつ異なる点やioctl()がデバイスごとに違うという点に対して、個別に変換ドライバを作成しなければならないことである。ゆえに、多種類のデバイスファイルへの対応や、異なるOS間での利用は困難である。また、ネットワーク上に直接接続されているデバイスの使用はできない。

2.2 スタブドライバ

スタブドライバ [3] は、デバイスドライバ内の各関数呼び出しを、別の計算機の同じデバイスドライバの関数呼び出しとして転送することで、別の計算機につながったデバイスへのアクセスを提供する。

この手法でさまざまなデバイスを扱うためには、各デバイスドライバごとに対応する必要がある。異なるOS間での利用は難しく、ネットワークに直接接続されているデバイスへの応用も困難である。また、デバイスドライバの各関数は今後変更される可能性が高く、呼び出しを転送する関数として不適切である。

2.3 RFS

RFS (Remote File Sharing) [4] はファイルシステムとして、リモートデバイスへのアクセスを提供する。他のネットワークファイルシステムと違い、一般のファイルだけでなく、デバイスファイルや名前付きパイプなども、共有できる。

計算機上にデバイスファイルとして存在するデバイスは共有が可能である。しかし、RFS は、ファイルシステムのセマンティクスを維持するために、その実装は複雑になる。また、異なる OS 間での共有やネットワークに直接接続するデバイスへの対応は、困難である。

2.4 SSI-Linux

SSI-Linux (Single System Image Linux) [5] は、複数のノードからなるシステムをあたかも一つの計算機のようにみせ、ノード間でプロセスのマイグレーションを可能にしている。その際、デバイスファイルも各ノード間で共有しているため、リモートデバイスへのアクセスが可能である。

しかし、これは、クラスタリングを目的とするシステムであり、リモートデバイスの使用を目的とはしていない。ゆえに、リモートデバイスの利用のために、SSI-Linux をインストールすることは現実的でない。RFS と同様、計算機上にデバイスファイルとして存在するデバイスの共有であるため、ネットワークに直接接続されたデバイスは利用できない。また、異なる OS 間での共有もできない。

2.5 iSCSI

iSCSI [1] は周辺機器制御インタフェースのひとつである SCSI のバスコマンドを TCP/IP を用いて送受信することで、ネットワーク透過のデバイスアクセスを可能にしている。現在すでに実用化されており、ネットワーク・ストレージに用いられている。

iSCSI の利点として、そのモデルが、OS や計算機のアーキテクチャに依存しない点が挙げられる。異なる OS 間での接続が可能であり、また、ネットワークに直接接続するデバイスとして製品も存在する。しかし、SCSI コマンドの特徴上、iSCSI でさまざまな種類のデバイスをリモートデバイスとすることは難しい。

2.6 既存研究の問題点

RDC、RFS、SSI-Linux はデバイスファイルを計算機間で共有できる。しかし、いずれも、そのモデルが、OS のデバイス・アクセス・モデルに強く依存しており、異なる OS 間での共有や、ネットワークに直接接続するデバイスへの応用が困難である。また、スタブドライバも、OS の実装に強く依存するため、同様の問題点がある。iSCSI は、OS や計算機のアーキテクチャに依存しない、SCSI のバスコマンドを採用することで、異なる OS 間での接続や、ネットワークに直接接続するデバイスを実現している。しかし、対応デバイスがストレージ系デバイスに限られる。

以上のように、デバイスファイルのセマンティクスでリモートデバイスへのアクセスを提供している研究事例は数多く存在する。しかし、異機種間での接続、ネットワークに直接接続するデバイスへの応用、さまざまな種類のデバイスへの対応を、すべて満すものは存在しない。

そこで、本研究では、USB のバスコマンドをプロトコルに取り入れることで、これらの問題点をすべて解決する。本研究の提案方式は USB のバスコマンドを IP ネットワーク上で送受信する点で iSCSI と似ている。しかし、本研究の提案手法は、iSCSI よりも数多くの種類のデバイスを利用できる点において、iSCSI よりも優位性がある。

3 提案方式

この節では、USB のドライバスタックに注目した理由、および、本方式の設計について述べる。

3.1 USB

USB (Universal Serial Bus) [6] は、それまでパーソナルコンピュータで使われてきた旧式のインターフェスを置き換える目的で開発された。1996 年に USB 1.0 仕様が公開され、その後 USB 1.1 を経て、

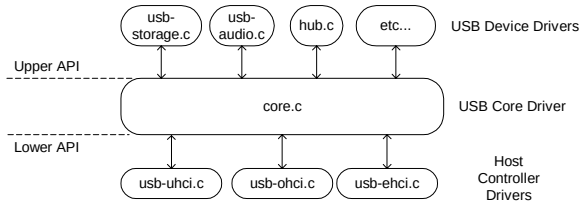

図 1: Linux の USB ドライバの構成

2000 年に USB 2.0 仕様が公開されている。

キーボードやマウスなどの入力装置や、ハードディスクや CD-ROM などのストレージ系デバイス、USB カメラや USB オーディオ機器などのマルチメディア系デバイス、その他プリンタなど非常に多くの種類のデバイスが USB で提供されている。これらさまざまなデバイスをサポートするために、4 つのデータ転送方式 (コントロール転送、バルク転送、インタラプト転送、アイソクロナス転送) と 3 つのデータ転送速度 (1.5Mbps、12Mbps、480Mbps) を用意しており、今後登場するデバイスにも対応可能になっている。

また、使いやすさを設計目標として掲げているため、デバイスの自動認識や活線挿抜が可能であり、ユーザの設定が不要である。

3.2 Linux における USB ドライバの実装

Linux における USB ドライバは大きく分けて 3 層の構成 (図 1) になっている。ドライバの改変や追加が容易になるよう、ホストコントローラ[1]の違いやデバイスの違いをなるべく抽象化している。

まず、各種ホストコントローラごとのドライバ (HC ドライバ) が一番下位の層に存在する。この HC ドライバでは、計算機のホストコントローラの状態を監視し、また、上位の層から託された USB コマンドのデバイスへの送信をホストコントローラに対して命令することなどを担う。この HC ドライバは、USB core ドライバが定める API の仕様に沿って作成されており、より上位のドライバに対してホストコントローラごとの違いを意識させないようになっている。

その上に USB core ドライバが位置する。この USB core ドライバにおいては、上位の各種 USB デバイスごとのドライバや下位の各種ホストコントローラごとのドライバに対して、共通の API を提供する。また、計算機上に存在する USB デバイスに固有の識別子を割り当てて管理したり、デバイスが USB ポートに接続された時に対応するドライバを呼び出す役割を持つ。

そして USB core ドライバの上位にそれぞれの USB デバイスのためのドライバがある。各種 USB ドライバは、USB core ドライバが提供する API を用いて、目的のデバイスに対して USB コマンドを発行する。

これら 3 層のドライバ内では、個々の USB デバイスに対する USB コマンドを表現するために、URB (USB Request Block) という構造体を用意している。この構造体中には、どのデバイスのどのエンドポイントに対して、どのような転送方式で、どのバッファのデータを送信するのか、またどのバッファに受信するのかという情報を含んでいる。さらに、発行した URB の完了通知のために実行される関数へのポインタを含んでいる。

3.3 提案方式の設計方針

本研究の目的は、ネットワーク透過のデバイスアクセスを既存のデバイスファイルを用いるセマンティクスで行えるようにすることである。そこで、以上に述べた USB の特徴と Linux の USB ドライバの構造に注目し、リモートデバイスを扱うための仮想ホストコントローラドライバ (VHC) を提案する。本方式の利点には、以下のものが挙げられる。

第一に、USB の特徴としてさまざまなデバイスをサポートしている点が挙げられる。すでにカーネル内には多種多様な USB デバイスのためのドライバが存在しており、これらをほぼそのまま利用できれば、即座にさまざまなリモートデバイスに対応できる。

第二に、Linux の USB デバイスドライバの構造が 3 層構成になっている。ホストコントローラ部分のドライバとしてリモートデバイスのための仮想ホストコントローラドライバを追加するだけで、さまざまな種類の USB デバイスのドライバをそのままリモートデバイスに対しても利用できる。

第三に、USB は活線挿抜の機構を供えている。この機構に対応するために、USB ドライバでは USB デ

[1] USB デバイスを制御する計算機上のハードウェア (UHCI、OHCI、EHCI などが存在)

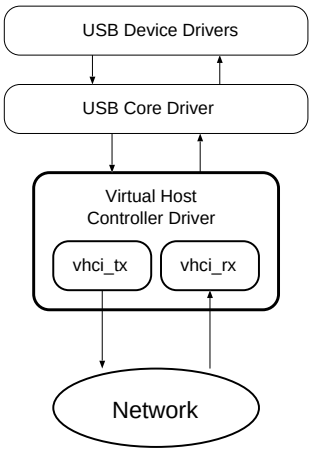

図 2: 仮想ホストコントローラドライバの構成

バイスの動的追加や削除をサポートしている。Linux では、計算機に新たに USB デバイスが追加されると動的に対応ドライバに操作がわたるようになっている。リモートデバイスにおいても動的な追加や削除への対応が必要になるので、USB ドライバの持つこれらの機能はリモートデバイスに対してもそのまま利用できる。

3.4 システム構成

提案方式のシステム構成について説明する。ここで、リモートデバイスを提供する側をサーバ、仮想ホストコントローラを追加しリモートデバイスを利用する計算機をクライアントと定義する。

まず、クライアント側では USB ドライバ内に仮想ホストコントローラドライバ (VHC ドライバ) (図 2) を登録する。これによって、vhci_rx と vhci_tx という 2 つのカーネルスレッドが生成される。

USB ドライバが URB を発行すると、VHC ドライバは USB core ドライバから渡された URB の情報をもとに、ネットワーク上に送信するデータを生成し、送信キューにためる。vhci_tx スレッドは送信データを送信キューから取り出し、これを TCP/IP パケットとしてネットワーク上のサーバに対して送信する。

vhci_rx スレッドはサーバから受信した TCP/IP データ内の情報をもとに、このパケットが対応する URB を見つけ出し、その内容を再構成する。そして、その URB を完了通知待ちキューに登録する。完了通知待ちキューに URB が存在すると、割り込みコンテキストにおいて URB 中の完了通知関数が実行され、URB を発行した USB ドライバに対して URB の要求が実行されたことを通知する。

次に、サーバ側について述べる。サーバに期待される動作は、クライアントから送られてきたリクエストをもとにリモートデバイスとして振る舞い、その結果をクライアントに返すことである。ゆえに、例えば、ライブカメラとして振る舞うリモードデバイスを実現するために、リクエストを解釈するエンジンを載せたイーサネットインタフェースつきのカメラを設計することも可能である。また、計算機でリアルタイムに取り込んだ動画データをリクエストに応じて送信するユーザランドのアプリケーションでも可能である。しかし、ここでは設計の容易さから、Linux が動作している別の計算機に接続した USB デバイスをリモートデバイスとして考える。

サーバとして振る舞う計算機において、リモートデバイスとして提供する USB デバイスのドライバとして、stub ドライバ (図 3) が割り当てられるようにする。USB core ドライバは接続されたデバイスの VenderID、DeviceID などを参考にして、目的のドライバに操作を依頼する。ここでは、リモートデバイスとして提供する USB デバイスのドライバとして、あらかじめ stub ドライバを登録しておく。stub ドライバがロードされると、stub_rx、stub_tx という 2 つのカーネルスレッドが生成される。

stub_rx は、クライアントからのリクエストを受けとると、それを目的のデバイス宛の URB に変換して、その URB を発行する。USB core ドライバと実際のホストコントローラドライバを経て、ハードウェアにリクエストが到達する。この URB が完了すると、割り込みコンテキスト内で URB 中に設定した完了通知関数が呼ばれるので、この時完了キューに URB を登録する。その後、stub_tx スレッドは完了キューから URB を取り出し、送信データを形成してクライアントに向けて送信する。

3.5 プロトコル

IP ネットワーク上で送受信するプロトコルについて説明する。

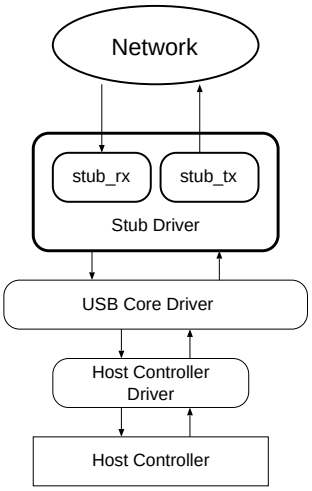

図 3: stub ドライバの構成

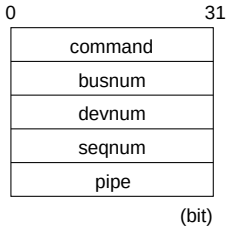

図 4: 基本ヘッダの構成

プロトコルとして定義するコマンドは以下のとおりである。すべてのコマンドは基本ヘッダ (図 4) と submit ヘッダ (図 5) および return ヘッダ (図 6) の組み合わせからなる。

submit カプセル化した URB を送受信する。基本ヘッダ中の command に SUBMIT_COMMAND を設定し、submit ヘッダが続く。

unlink 過去に送受信した URB を取り消す。基本ヘッダ中の command に UNLINK_COMMAND を設定する。

return URB の実行結果を送受信する。基本ヘッダ中の command に RETURN_COMMAND を設定し、return ヘッダが続く。

ack 到着確認通知を送受信する。基本ヘッダ中の command に ACK_COMMAND を設定する。

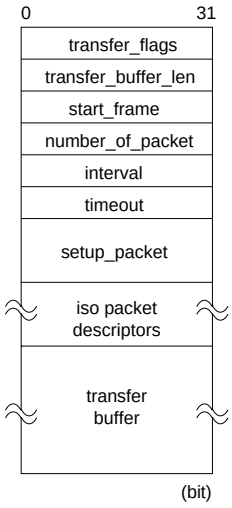

図 5: submit ヘッダの構成

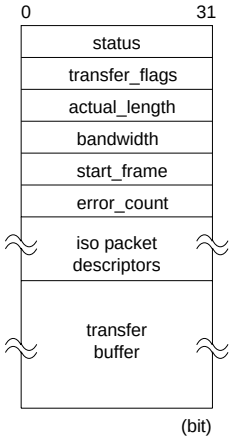

図 6: return ヘッダの構成

一つの URB を処理する時にクライアントとサーバ間でやりとりされる一連のプロトコルを、コントロール転送とバルク転送およびアイソクロナス転送の場合を図 7 に、インタラプト転送の場合を図 8 に示す。

4 プロトタイプの実装

提案方式の妥当性を確認するため、Linux kernel 2.4.20 上にプロトタイプを実装した。クライアント用に VHC ドライバを、またサーバ用に stub ドライバを、それぞれカーネルモジュールとして作成した。

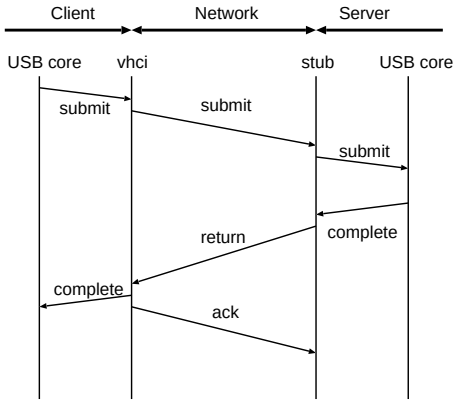

図 7: コントロール、ブロック、アイソクロナス転送のシーケンス

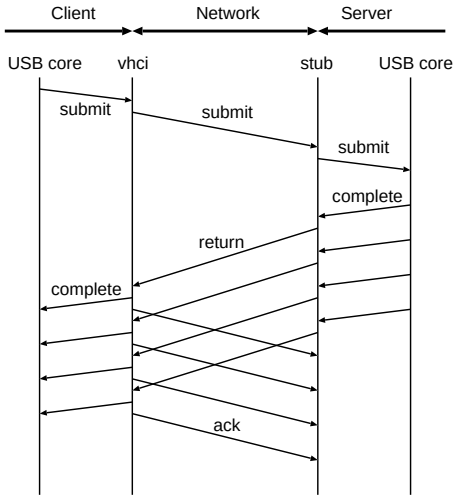

図 8: インタラプト転送のシーケンス

本来、同一計算機内にあるホストコントローラを制御するための HC ドライバで、ネットワーク上のデバイスを制御しようとしているため、HC ドライバよりも上位のドライバで設定されているタイムアウト値をより大きな値に変更する必要があった。プロトタイプにおいて変更したものを表 1 に示す。これ以外で、カーネルの他の個所を変更していない。

VHC ドライバをカネールにロードすると、proc ファイルシステム上に、`/proc/vhci/up` というファイルが生成され、

```
echo ポート IP バス番号 デバイス番号 > /proc/vhci/up
```

変数	場所
SCSI_TIMEOUT	`drivers/scsi/scsi.h`
timeout	usb_start_wait_urb() (`drivers/usb/usb.c`)

表 1: 変更したタイムアウト値

デバイス名	動作状態
USB マウス (Apple)	○
USB キーボード (PlatHome)	○
USB ZIP-100 (Iomega)	○
USB CDROM (IBM)	○
USB カメラ (LifeView)	enumeration のみ可

表 2: プロトタイプで動作確認したデバイス

のようにすることでリモートデバイスを仮想的に接続する。

プロトタイプにおいて動作確認したデバイスは表 2 のとおりである。コントロール転送、バルク転送、インタラプト転送の実装が済んでいる。USB デバイスは初期化時にコントロール転送に答えて基本的な情報を送信するので、すべての USB デバイスの初期化はできる。インタラプト転送を用いる USB キーボード、USB マウスなどは問題なく動作する。また、バルク転送を用いるストレージ系の USB デバイスも問題なく動作し、ディスク上にファイルシステムを作成、マウントし、読み書きが可能である。

5 今後の課題

現在、USB カメラや USB オーディオのようなアイソクロナス転送を用いる USB デバイスは動作できていない。しかし、これらデバイスは、実装の追加により動作可能と考えている。

本方式で用いるプロトコルは、Linux の USB ドライバの構造に依存したものになっている。まず、このプロトコルをネットワーク上のデバイスを利用する上でより汎用的な形式に改善する必要がある。そして、異なる OS 間での接続性や、ネットワークに直接接続するデバイスへの対応を確認しなければならない。また、そのように改善した時に、仮想ホストコントローラによる枠組みが適切かを検証する必要がある。

また、IP ネットワーク上でデバイスを制御するた

めに、解決を要する課題が多数存在する。

まず、エラー時のリカバリである。データ転送中に起きるエラーとして、iSCSI では TCP/IP レイヤから受け取ったデータが壊れていた場合と TCP/IP のコネクションが切れた場合を挙げている。提案方式では、前者の場合、CRC で訂正を行ったりデータの再送を依頼することが考えらる。また、後者の場合、再びコネクションを張り直して転送を再開することが考えられる。しかし、両者の方式とも、相手側でどの段階まで処理が済んでいて、どの段階から再開するのか、ということを正確に把握しなければならない。さらに、転送を再開できなかった時に、一度発行した命令が完了しなかったことを、上位のドライバに対して通知する必要がある。以上のリカバリの機構を構築する時に、OS のファイルシステム等から再考を要する可能性がある。

次に、デバイス制御データの QoS (Quality of Service) を解決しなければならない。デバイスの種類やその使用形態によって、IP ネットワーク上でどのように制御データを送受信するのかが異なる。キーボードやマウスのようなデバイスは、使用感を損なわないよう一定の遅延内で制御データを送受信する必要がある。また、カメラやスピーカのようなデバイスでは、リアルタイム性が要求される反面、データ中の若干のエラーを許容する場合もあり得る。USB などの専用バス上で行われている通信制御を、IP ネットワーク上でも行うという観点からは、十分な研究がされていない。

認証やセキュリティについても、デバイスの種類やその使用形態ごとに要求は異なる。リモートデバイスの使用権限は状況に応じて変わる可能性があり、ネットワークを通じて、リモートデバイス利用の認証、権限、アカウンティングを管理できる機構が必要になる。さらに、制御データの暗号化を要する場合もある。

適切な名前空間をユーザに提供する必要もある。例えば、リモートデバイスとしてプロジェクタを使用する際にはその設置場所で指定したいと考えられる。また、ヘッドフォンを利用する場合には、その所有者名で指定したいと思われる。単に IP アドレスとデバイスの ID との組み合わせでは解決しない。

6 おわりに

本論文では、Linux の USB ドライバに仮想ホストコントローラを作成することで、ネットワーク上に存在するデバイスを既存のデバイスファイルのセマンティクスで制御する手法を提案し、そのプロトタイプを実装した。提案方式で動作するデバイスは実際の利用に耐える使用感を得ることができ、仮想ホストコントローラを作成するという基本的なアイディアの妥当性はあると考える。また、USB のバスコマンドが OS や計算機のアーキテクチャに依存しないので、異なる OS 間での接続やネットワークに直接接続するデバイスへの対応も可能であると考える。

今後、提案方式の定量的評価を行う。その後、リモートデバイスを OS の枠組みの中で扱っていくためのさまざまな課題に取り組んでいきたいと考えている。

参考文献

[1] IPS Working Group Internet Engineering Task Force, : iSCSI Internet Draft, `http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-20.txt` (2003).

[2] 佐藤純次, 河合栄治, 中村豊, 藤川和利, 砂原秀樹：リモート・デバイス利用に関する汎用的なフレームワークの設計と実装, 情報処理学会研究報告, 2003-OS-92, pp. 115–122 (2003).

[3] 佐藤友隆, 中山健, 小林良岳, 前川守：カーネルレベルで実現したネットワーク透過な周辺機器制御の枠組み, 電気情報通信学会技術研究報告, CPSY2001-26, pp. 9–15 (2001).

[4] Rfkin, A. P., Frobes, M. P., Hamilton, R. L., Sabrio, M., Shah, S. and Yueh, K.: RFS Architectural Overview, in *USENIX Conference Proceedings*, pp. 248–259 (1989).

[5] OpenSSI Cluster for Linux, `http://sourceforge.net/projects/ssic-linux/`.

[6] USB (Universal Serial Bus), `http://www.usb.org/`.