IP デバイス空間実現にむけた USB over IP によるデバイス制御機構の提案

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ネットワークインタフェースの低価格化によって、IP ネットワークを介して制御可能な計算機の周辺機器が登場しつつある。我々は、IP ネットワーク上に遍在する計算機の周辺機器をオンデマンドかつシームレスに使用するアーキテクチャとして IP デバイス空間を開発中である。IP デバイス空間実現のためには、IP ネットワークを介した周辺機器制御技術が不可欠である。しかし、既存の IP ネットワークを介した周辺機器制御技術は、いずれも IP デバイス空間に適さない。そこで、IP デバイス空間上でのデバイス制御手法として、USB over IP によるデバイス制御を提案する。本論文では、IP デバイス空間について説明した後、USB over IP によるデバイス制御手法を提案し、その設計について述べる。

The Proposal of a Device Control Framework with USB over IP for Realizing IP Device Space

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

Abstract: The spread of high bandwidth networks and low-priced network interfaces brings about the appearance of devices controlled over IP network. We try to realize *IP Device Space*, where users can access devices on-demand and seamlessly which are ubiquitous over IP network. To realize IP Device Space, a device control technology over IP network is required. However, the existing device control methods over IP network cannot be applied to IP Device Space. Thus, we propose a device control framework with USB over IP. In this paper, we describe IP Device Space and then discuss the design of USB over IP framework.

1 はじめに

IP 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。また、ネットワークインタフェースの低価格化はめざましい。従来であれば、計算機の周辺機器は、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、IP 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、IP パケットを媒体とする周辺機器制御が可能になると考える。今後、従来の計算機の周辺機器の中にはネットワークインタフェースを備えるものが登場すると期待できる。

我々は、計算機のすべての周辺機器を IP ネットワーク上に位置付けることで、計算機からデバイスへのオンデマンドかつシームレスなアクセスを提供し、来るべきユビキタス環境にふさわしいコンピューティング環境の構築を目指している。すべての周辺機器を IP ネットワーク上に位置付ける形態を IP デバイス空間と呼ぶ。IP デバイス空間では、(1) すべてのデバイス識別子の IP アドレス化、(2) IP ネットワーク上のデバイスの仮想接続、(3) ネットワーク QoS と連携したデバイスサービス によって、計算機の周辺機器を IP ネットワーク上で動的に再構成する。これらによって、必要とされるデバイスが必要とする計算機で利用される環境を構築する。

IP デバイス空間実現のためには、IP パケットを媒体とする周辺機器制御機構が必要になる。しかしながら、現状では IP デバイス空間に適した周辺機器制御機構は存在しない。iSCSI [1] は IP ネットワークを介したデバイス制御を可能とするものの、対象とするデバイスがストレージに限られる。また、情報家電等で用いられる UPnP [2] は、単体でも機能するデバイスを対象とするため、IP ネットワークを介した計算機の周辺機器制御には適さない。なぜなら、計算機からの制御を前提として単純化されている周辺機器を UPnP に対応化させるためには、デバイス自身の複雑化を招き、その普及を妨げると考える。

そこで、IP デバイス空間実現に不可欠な、IP パケットを媒体とするデバイス制御手法を提案する。提案するデバイス制御手法は USB [3] コマンドを IP パケットによってカプセル化 (USB over IP) する。これによって、

† 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
¶ 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

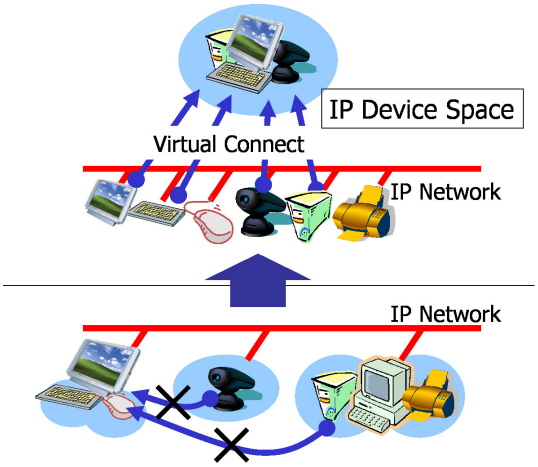

図 1　IP デバイス空間概念モデル

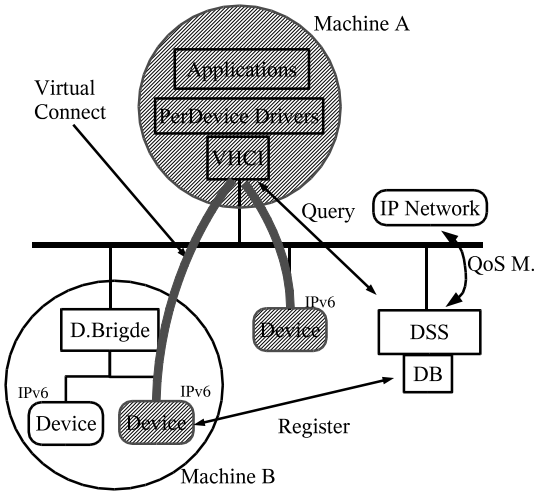

図 2　IP デバイス空間アーキテクチャ

IP デバイス空間上で、デバイス制御をデバイス種ごとにクラス化し、IP ネットワークを介した制御として適度に抽象化された制御が可能になる。USB が対象とするデバイスの多種性を IP ネットワーク上にも拡張できる。

以下に、本論文の構成を示す。2 節で IP デバイス空間を説明し、3 節で IP ネットワークを介したデバイス制御手法への要求事項を整理する。4 節で USB over IP を提案し、5 節で USB over IP の設計について述べる。6 節でプロトタイプの実装と動作状況を記す。そして 7 節でまとめる。

2　IP デバイス空間

IP デバイス空間は、計算機の周辺機器を必要とする計算機にシームレスに提供することを目指して、我々が構築中のアーキテクチャである。本節では、USB over IP によるデバイス制御手法を適用するプラットフォームである IP デバイス空間について説明する。

2.1　概念モデル

低価格化した様々なネットワークインタフェースと IPv6 [4] の豊富な IP アドレスによって、今後 IP ネットワークを介して制御可能な計算機の周辺機器が登場してくると仮定する。これら IP パケットで制御される周辺機器と遍在化した IP ネットワークを用いて、計算機の物理的なデバイス構成に縛られることなく、必要とされる周辺機器を必要とする計算機にシームレスに提供できると考えている。

しかし、従来の計算機は物理的に接続した周辺機器を制御の対象としてきた。そのため、計算機が IP ネットワークに接続されても、現状の OS のデバイスアクセス機構は、IP パケットを媒体とする周辺機器を従来の周辺機器と違いなく扱えるとは言い難い。当然、計算機に接続された周辺機器を、ネットワーク上の他の計算機と共有することは、困難であった。このような状況下では、IP ネットワークを介した制御の利点を生かしたオンデマンドなデバイスアクセスを提供することは難しい (図 1 下)。

そこで、我々は計算機の周辺機器を必要とする計算機にシームレスに提供することを目指して、IP デバイス空間を提案している (図 1 上)。IP デバイス空間上では、ネットワークインタフェースを持つ周辺機器も、計算機に接続された周辺機器も、すべて IP ネットワーク上に位置付け、IP ネットワークを介した制御を可能にする。計算機は、IP ネットワーク上に存在する周辺機器を必要に応じて仮想的に接続していく。これによって、計算機は自身の物理的なデバイス構成とは全く別に、ネットワーク上に自身が使用するデバイス群を構成する。言わば、密結合である従来の計算機と周辺機器の関係を、疎結合の関係として捉え直し柔軟に再構築することで、必要とされるデバイスが必要とする計算機に使用される環境を作る。

例えば、複数の計算機を同時に使用する際、手元のマウスやキーボード、ディスプレイを含めるように、次々と使用中の計算機のデバイス群を再構成することで、あたかも KVM Switch のように端末を切りかえられる。また、知人の計算機が手元の DVD ドライブを含めてデバイス群を再構成することで、距離に関係なく自分の DVD ソフトを知人に貸すことができる。さらに、外出先に存在する CD-R ドライブやスキャナ等でノート PC のデバイス群を再構成することで、面倒な接続や設定をすることなく、ゲストアカウントで使用させてもらうことなどができる。

2.2　アーキテクチャ

図 2 に IP デバイス空間のアーキテクチャを示す。我々は、OS の既存デバイスアクセスセマンティクスに十分配慮した、IP デバイス空間実現のためのフレームワーク構築に取り組んでいる。

2.2.1　デバイス識別子の IP アドレス化

計算機が対象とする周辺機器を区別するため、IP ネットワーク上でユニークなデバイス識別子を導入する必要がある。そこで、本研究では IPv6 アドレスをデバイス識別子として割り当てる。この際、ネットワークインタフェースを持つデバイスだけではなく、特定の計算機につながっているデバイスに対しても固有の IPv6 アドレスを割り当てる。そして、ネットワーク上の計算機は、これらデバイスを、接続方式を意識することなく、一貫し

て IP を識別子として制御する。

IPv6 がもつ 128bit の豊富なアドレス空間と自動アドレッシング機能によって、集中的な識別子管理機構を設けることなしに、インターネット上で一意なデバイス識別子を決定できる。また、デバイス識別子が IP ネットワーク上の宛先を兼ねるので、デバイス識別子とメッセージ送受信先とのマッピング機構が不要になる。さらに、デバイス利用ごとに、ネットワーク QoS や、IPSec [5]、MobileIP [6] 等の既存技術を適用できる。デバイスへのネーミング機構と名前解決機構に DNS 等の既存技術を応用できる。

計算機内のデバイスにも IP アドレスを割り当てるために、計算機上に Device Bridge を構築している。計算機内のデバイスがあたかも IP ネットワーク上で固有の IP で存在するかのように見せる。Device Bridge では、IP パケットを媒体とするデバイス制御を計算機内部のデバイス制御方式 (例えば USB や IEEE1394 [7] など) に変換する。

2.2.2 デバイスの仮想接続

計算機は、IP ネットワーク上に存在するデバイスを制御できる必要がある。IP デバイス空間上のデバイスが計算機によって制御可能になった状態を仮想的に接続 (Virtual Connect) されたと呼ぶ。従来の OS のデバイスアクセス機構は、IP ネットワーク上のデバイスについて何ら考慮されていない。そこで、既存 OS の枠内でネットワーク上に存在する各種デバイスを仮想的に接続する機構が必要になる。そして、この機構は OS のデバイスアクセス部分以外への変更を最小限に留めることが求められる。

これにより、既存のソフトウェア資源を IP ネットワーク上のデバイスにも従来通り最大限利用できる。ユーザランドのアプリケーションには、既存のデバイスファイルによるデバイスアクセスを提供できる。また、カーネル内では、ファイルシステム等のコンポーネントを変更することなしに、IP ネットワーク上のデバイスに対しても適用できる。

デバイスの仮想接続のために、仮想ホストコントローラ (Virtual Host Controller Interface, VHCI) を作成した。デバイスを使用する計算機には、VHCI が存在し IP パケットを媒体とするデバイス制御を担う。デバイス制御のうち IP パケットを媒体とする上での処理の大半を VHCI で行う。そして、各デバイスごとの固有の処理はそれぞれ専用のデバイスドライバ (PerDevice Driver, PDD) が対応する。VHCI および PDD 部分以外では、従来の OS 構造を最大限維持する。

2.2.3 ネットワーク QoS と連携したデバイスサービス

計算機が目的とするデバイスを発見できる機構が必要になる。この時、デバイスは様々な条件でデバイスを指定できなければならない。例えば、デバイスの位置情報や所有者など、その属性は多岐に渡る。さらに、デバイスの使用に際しては、IP ネットワークの状態をデバイス制御に対して最適な状態にする必要がある。

デバイス発見機能は DSS (Device Service System) で提供される。IP デバイス空間上に存在するデバイスは、DSS 上のデータベースにその属性情報 (位置情報、所有者など) とともに登録される。アプリケーションあるいはユーザが新たなデバイスを必要とすると、VHCI は DSS にデバイス検索を要求する。DSS はデバイス情報が登録されているデータベースを検索して、条件に合致するデバイスが存在すれば、デバイスを VHCI に対して通知する。既存の属性管理システムと柔軟に連携することで、様々な属性情報に基づくデバイス発見を用意できる。また、DSS は VHCI の依頼に応じて、デバイスの特性に合せたネットワーク QoS を制御する。

3 IP ネットワークを介したデバイス制御への要求事項

本稿では、IP デバイス空間上で求められる、IP ネットワークを介したデバイス制御手法を提案する。本節で IP デバイス空間上でのデバイス制御機構に対する要求事項をまとめる。

3.1 デバイスのクラス化

典型的なデバイスに対しては、あらかじめその制御プロコルを詳細に決める必要がある。これによって、デバイスと計算機間の相互接続性を高められる。また、OS 開発者のデバイスドライバ作成の負担を軽減する。さらに、デバイス制御モジュールの低価格化を実現し、その普及が期待できる。

3.2 デバイス制御の適度な抽象化

デバイス制御を適度に抽象化する必要がある。一般的に、計算機とデバイス間の通信量を減らそうとすると、計算機側のドライバが単純になる反面、デバイス側で高度な処理が必要になる。これは、デバイスのコストを高くし、その普及を妨げる点で望ましくない。逆に、計算機側での高度な処理を期待しデバイス側の機能を単純にすると、制御の粒度が細かくなるため両者間での通信量が増大する。これは、スループット低下の原因となるため、できる限り避けるべきである。

3.3 エラーリカバリ

IP ネットワーク上では、さまざまな原因で通信切断やパケット損失が生じる。そこで、信頼性のあるデバイス制御を提供するため、エラーリカバリの機構が必要になる。このエラーリカバリの機構は、デバイスに対して複雑な処理を要求してはならない。なぜなら、IP デバイス空間上のデバイス普及を促すためには、デバイス側を十分単純に保つ必要がある。

4 USB over IP の提案

以上に挙げた要求要件を満すデバイス制御手法として、USB over IP によるデバイス制御を提案する。本節では、前提として USB を説明し、その後提案手法について述べる。

4.1 USB

USB (Universal Serial Bus) [3] 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フェー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、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。

4.2 USB over IP

USB において用いられる USB コマンドを IP パケットにカプセル化する手法 (USB over IP) を、IP デバイス空間におけるデバイス制御手法として提案する。ここで上記要求事項を確認する。まず、USB over IP は IP ネットワークを介したデバイス制御として適度に抽象化できる。USB コマンドは、計算機側での高度な処理を前提として単純化されており、制御モジュールはハードウェアで容易に実現されている。また、一部のストレージ系のデバイス以外は 12Mbps あるいは 1.5Mbps の転送モードで制御される点が示すように、計算機とデバイス間の通信量は低く抑えられている。ゆえに、通信帯域の点で問題はない。ただし、IP ネットワークの RTT が比較的大きいゆえ、IP ネットワークへ最適化するために独自の拡張を施す必要がある。また、デバイスのクラス化という点で、USB においては、ストレージクラス、HID(Human Interface Device) クラス、オーディオクラス、ビデオクラス等があらかじめ存在する。IP デバイス空間においても計算機の周辺機器を対象とするので、基本的にこれらのクラス分けが有効である。エラーリカバリは、IP パケットに訂正符号を含めたり、トランザクション機構を実装することによって、達成できると考える。

IP パケットを媒体とするデバイス制御において用いるプロトコルは、USB および USB の各デバイスクラスで規定されているプロトコルを基にする。しかし、IP ネットワークを介して送受信される点を考慮して、独自の拡張を行う。ネットワーク遅延への対策として、タイムアウト値やコマンドデータ長を調整する。また、新たなコマンドを規定することで、複数のコマンド送受信からなる処理をより少ないコマンドで置換する。さらに、実装する際には、バッファ管理によってネットワークジッタを吸収したり、データの再送やエラー訂正によってパケットロスに対処する。

このような対策をしても、IP ネットワークを介するために解決できない問題がある。一つはリアルタイム性を要求するアプリケーションに対して、ネットワーク遅延分の遅れが最低限発生する。また、通信接続性が不安定な環境においては、十分な性能でデバイス制御を提供することは難しい。しかし、ネットワーク QoS やデバイス制御データの冗長化等を組み合わせることによって、これら問題点をある程度は緩和できると考える。さらに、USB の特徴であるデバイスの多種性や自動認識機構等の利点を考慮すると、これら欠点は補われて余りあると考える。

5 USB over IP の設計

5.1 USB ドライバスタック

各 OS における USB の実装はおおよそ 3 層の構成になっている。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の違いやデバイスの違いをなるべく抽象化している。ホストコントローラは USB デバイスを制御する計算機上のハードウェアであり、UHCI (Universal Host Controller Interface) [8]、OHCI (Open Host Controller Interface) [9]、EHCI (Enhanced Host Controller Interface) [10] などが存在する。

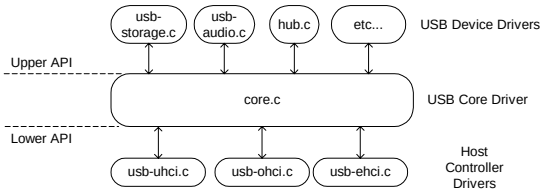

図 3 Linux の USB ドライバの構成

表 1 USB コマンドの種類と対応プロトコル

種類	Protocol	主なデバイス
Control	TCP	すべて
Bulk	TCP	ストレージ系
Interrupt	TCP	キーボード、マウス等
Isochronous	RTP	カメラ、スピーカ等

Linux の実装 (図 3) を例にとると、まず、各種ホストコントローラごとのドライバ (Host Controller Driver, HCD) が一番下位の層に存在する。この HCD 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 HCD 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 (USB PDD) や下位の各種 HCD 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 USB PDD がある。USB PDD 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。

5.2 VHCI の設計

以上のような構成になっているため、各 OS において USB コマンドを取り出すことは比較的容易である。そこで、HCD のひとつとして VHCI を作成する (図 4)。USB core ドライバから USB コマンドを得て、IP にカプセル化し、ネットワーク上のデバイスに対して送信する。VHCI では、トランザクション管理層によって、信頼性のあるデータ送受信を提供する。USB コマンドの種類に応じて、TCP/IP あるいは RTP/IP [11] によってカプセル化を行う (表 1)。

また、USB の PDD を基に、USB over IP 用の PDD を作成する。このドライバでは、4.2 節で述べたようにオリジナルのドライバを改変する。また、デバイスから

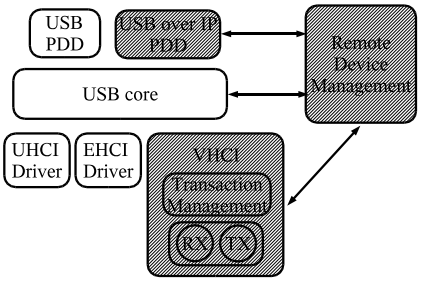

図 4　USB over IP ドライバの構成 (VHCI)

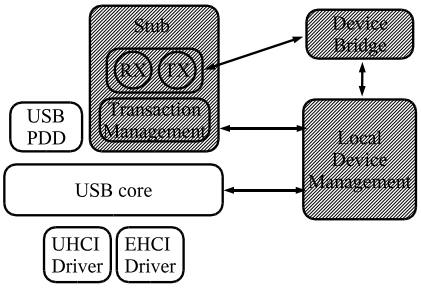

図 5　USB over IP ドライバの構成 (Stub)

の情報によって、IP パケットを媒体とするデバイスとしての動作を変更できるように拡張する。例えば、マウスに特定の動作をさせることで、デバイスの仮想的な接続先を変更できる。ほとんどのデバイスがクラスドライバ (ストレージクラス、HID クラス等のドライバ) で動作するので、PDD のうちクラスドライバを中心に改変するだけで、多数のデバイスに対応できる。

Remote Device Management (RDM) は、アプリケーションやユーザから新たなデバイスの要求を受けて、DSS に検索を依頼する。要求されたデバイスを発見できると、VHCI および PDD に対してデバイスの詳細を報告する。デバイスの仮想接続中にエラーが起きた場合、VHCI、USB core、PDD にエラーの詳細を報告し、各層での正確なエラー処理を支援する。また、PDD に対してデバイス接続を管理する API を提供し、先述の例のようにデバイス操作による仮想接続状態の変更を可能にする。

5.3　デバイスの設計

実装を容易にするため、IP デバイス空間上に提供するデバイスは、既存の USB デバイスを基にする。USB over IP による制御を可能にするために、USB ドライバスタック中の PDD として、Stub ドライバ (図 5) を作成する。IP デバイス空間に提供するデバイスは Stub ドライバが担当し、提供しないデバイスは従来の USB の PDD で対応する。Stub ドライバは、IP でカプセル化された USB コマンドを取り出し、下位のドライバに渡し、USB デバイスに命令する。そして、その結果を再び IP にカプセル化して、ネットワーク上の計算機に返す。Stub ドライバと VHCI は共存可能であるので、計算機は自分のデバイスを IP デバイス空間上に提供することもでき、かつ同時に、IP デバイス空間上のデバイスを使用することもできる。

表 2　実装環境

CPU/メモリ	PentiumII 400MHz, SDRAM 128MB
USB 1.1	Intel 82371AB USB Host Controller
USB 2.0	Buffalo IFC-USB2P5(μPD720100)
OS/コンパイラ	linux-kernel-2.4.20, gcc-2.95.4

Local Device Management (LDM) は、IP デバイス空間に提供可能なデバイスを DSS に登録する。この際、デバイスの属性情報 (位置情報など) も他のシステムと協調動作して登録する。デバイスが提供不能になると DSS に登録抹消依頼を出す。また、Device Bridge や Stub ドライバ等のエラー回復を支援する。

Device Bridge によって、IP デバイス空間に提供しているデバイスを IPv6 アドレスによって一意にアクセスできるようにする。提供するデバイスごとに擬似ネットワークインタフェースを作成し、それにネットワーク上で一意な IPv6 アドレスを割り当てる。Stub ドライバが送信する IP パケットは、この擬似ネットワークインタフェースを経て、実際にネットワークに送出される。

5.4　エラーリカバリ

VHCI および Stub 内の Transaction Management では信頼性のあるデバイス制御を提供する。状態遷移を把握し、必要ならばエラーリカバリを行う。

メッセージの紛失を検知した場合、紛失が許容されるメッセージでなければ再送する。また、計算機とデバイス間の IP 層での到達性が無くなった場合、VHCI 側では、デバイス使用の中止を RDM に報告する。そして、Stub 側では、デバイスの初期化を LDM に依頼する。

USB デバイスにおいて問題が発生した場合、VHCI 側の USB over IP 用 PDD ドライバが対応する。さらに、USB デバイスが外された場合には、Stub 側の LDM と Device Bridge でデバイス提供の中止処置を行い、VHCI はデバイス使用を中止する。

さらに、計算機側のソフトウェアおよびハードウェア的原因によって、使用中のデバイスに制御が及ばなくなった状態 (network orphan) への対処が必要になる。この場合、一定時間計算機からの反応が得られないと、Stub 側ではこのセッションは終了したものとみなし、自らをリセットする。そして、次の接続を受けつける。

6　プロトタイプの実装

我々は、USB を IP パケットにカプセル化するという基本的なアイディアの有効性を、既に文献 [12] において確かめている。USB over IP のプロトタイプを実装し、デバイスの動作状況を調べた。実装環境を表 2 に示す。プロトタイプ実装では、TCP/IP による USB コマンドの送受信機能を備えた VHCI および Stub ドライバを作成し、最低限の USB over IP によるデバイス制御が可能である。

デバイスの動作状況を表 3 に示す。性能上の問題はあるものの、確認した範囲では全ての USB デバイスを USB over IP で制御できた。RTP による送受信機能やトランザクション処理機構等を追加することで、問題点を改善できると考える。図 6 に USB over IP の使用例を示す。表 3 中の USB カメラを仮想的に接続した際に、デ

表 3　デバイス動作状況

デバイス	動作状況	問題点とその対策
USB キーボード PFU FKB8579-USB	コンソール、X Window System で使用可能	正常動作
USB マウス Apple USB Mouse M848	X Window System で使用可能	正常動作
USB ハードディスク IO-DATA HDA-iU120	`mount /dev/sda0 /mnt` が可能 `fdisk, mkfs` およびファイルの読み書きも可能	スループットが小さい (直接接続時の約 10%) 実装上の改善およびコマンドの改良が必要
USB カメラ NOVAC NV-US120H	`xawtv -d /dev/video` などで使用可能	コマ落ちが多数発生 RTP 化による改善が必要
USB スピーカ ONKYO GX-R5U	`mpg123 -a /dev/dsp` などで使用	音飛びが多数発生 RTP 化とバッファ管理が必要

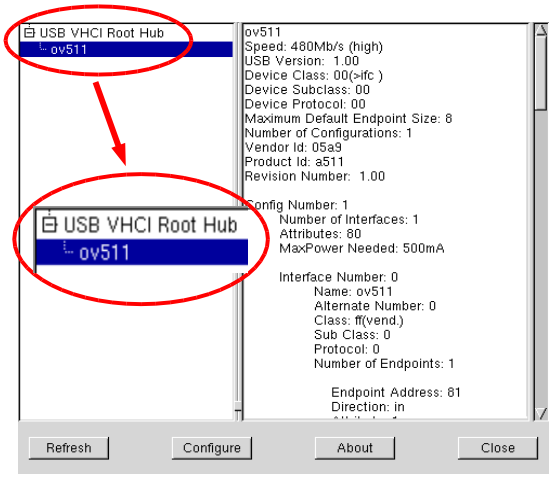

図 6　usbview 実行例

バイスを使用する計算機側で usbview [13] を実行した。VHCI というホストコントローラ上にリモートの USB カメラ (ov511) が存在する。

7　おわりに

本論文では、IP ネットワーク上で計算機の周辺機器を構成するアーキテクチャとして、現在我々が開発中の IP デバイス空間を説明した。そして、IP デバイス空間上で必要とされる、IP パケットを媒体とする周辺機器制御方式への要求事項を検討した。IP デバイス空間での周辺機器制御方式として USB over IP を提案し、その設計とプロトタイプの実装について述べた。プロトタイプの実装においてほとんどの USB デバイスの動作を確認できているので、USB を IP パケットでカプセル化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。

今後、USB over IP の本格的な実装を進め、IP デバイス空間実現にむけて取り組んでいく。その際、IP ネットワークの状態に応じた最適なデバイス制御手法についても考えたい。また、IP デバイス空間におけるアクセス制御手法の確立なども課題であろう。

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] Universal Plug and Play, `http://www.upnp.org`

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

[4] Deering, S. and Hinden, R.: Internet Protocol, Version 6 (IPv6) Specification, `http://www.ietf.org/rfc/rfc2732.txt` (1998)

[5] Kent, S., Corp, B. and Akinson, R.: Security Architecture for the Internet Protocol, `http://www.ietf.org/rfc/rfc2406.txt`

[6] Perkins, C.: IP Mobility Support, `http://www.ietf.org/rfc/rfc2002.txt`

[7] IEEE1394, `http://www.1394ta.org/`

[8] Universal Host Controller Interface (UHCI) Design Guide, `http://www.intel.com/design/USB/UHCI11D.htm`

[9] OpenHCI - Open Host Controller Interface Specification for USB, `http://h18000.www1.hp.com/productinfo/development/openhci.html`

[10] Universal Serial Bus - ECHI Specification, `http://www.intel.com/technology/usb/ehcispec.htm`

[11] RTP: A Transport Protocol for Real-Time Application, `http://www.ietf.org/rfc/rfc1889.txt`

[12] 広渕崇宏, 河合栄治, 中村豊, 藤川和利, 砂原秀樹：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, 情報処理学会研究報告, 2003-OS-93, pp. 41–48 (2003).

[13] usbview, `http://sourceforge.net/projects/usbview/`