情報処理学会論文誌　コンピューティングシステム　Vol. 2　No. 1　53–63 (Mar. 2009)

VMMでWebVDIを実現するシステムの実装と評価

高橋一志[†1]　笹田耕一[†1]　竹内郁雄[†1]

Virtual Desktop Infrastructure（VDI）とは，仮想デスクトップを実現するための基盤技術である．thin client は VDI を実現するシステムの 1 つであるが，インターネットを通して不特定多数の人々に対して仮想デスクトップを提供するシステムとしては不向きである．なぜなら，thin client は専用クライアントを必要とし，それをユーザに導入させるのには困難がともなうからである．そこで我々は，コモディティ化した Web ブラウザをクライアントとして使用できる新たな VDI（WebVDI）を提案する．さらに，ユーザへは独立した 1 つの計算機環境を提供できる一方，サービスの提供基盤たるサーバの安全性を最大限に確保できる WebVDI も目指している．本論文では，仮想マシンモニタ（VMM）上で動作するゲスト OS の画面を HTTP に変換することで，Web ブラウザをクライアントとして使用できるシステムである VoXY を提案する．VoXY は多くの VMM がサポートする VNC プロトコルを HTTP に変換するプロキシ機能と，VMM を操作するインタフェースを持つ．本システムは，ユーザが Web ブラウザを動作させることのできる端末を用意するだけで利用することができる．また，VMM を利用することで，ユーザは仮想的な計算機環境を独占して自由に利用することができる．評価の結果，実用的な性能で WebVDI を実現できることが分かった．

Implementation and Evaluation of WebVDI System with VMM

Kazushi Takahashi,[†1] Koichi Sasada[†1]
and Ikuo Takeuchi[†1]

Virtual Desktop Infrastructure (VDI) is a base technology to achieve a Virtual Desktop Environment. Thin client is one of the technologies that implement a VDI. However, it is not good at providing virtual desktops to the general public in the Internet. Because existing thin clients require dedicated clients and it is difficult for average users to install. Thus, we propose a new VD, a WebVDI, which can use commodity web-browsers as its clients. In addition, we can offer one individual computer environment to every user and aim at a WebVDI which assures the safety of the server, the base of the service. In this paper, we propose the VoXY system which uses web-browsers as clients by converting a screen of the guest OS working on a virtual machine monitor (VMM) into HTTP. The VoXY system consists of proxy function which converts the VNC protocol supported by many VMMs into HTTP and interface to operate VMMs. This system requires only clients on which commodity web browser is installed. Moreover, this system offers individual virtual computer environment to the users by utilizing VMMs. Results of our evaluation show that the VoXY system has a practical performance to build WebVDIs.

1. はじめに

VDI: Virtual Desktop Infrastructure とは，ネットワークを通じて，遠隔地にある計算機上のデスクトップ環境を，手元の計算機上に再現するためのインフラストラクチャのことである．VDI により再現されたデスクトップを “仮想デスクトップ” と呼ぶ．

VDI の概念は目新しいものではなく，代表例としては古典的な thin client システムがあげられる．thin client は既存の OS やウィンドウマネージャをそのまま仮想デスクトップにすることができるため，今まで使用してたソフトウェアやデータをそのまま引き継いで利用できるという利点がある．

現状，thin client は，プライベートなネットワーク内で組織に所属する従業員が使用するといった，クローズドな運用形態が多い．これは，専用クライアントやアプライアンスのインストール，特殊なプロトコルを使用するにあたって通信インフラの整備が必要であることが原因である．つまり，既存の thin client は新たなユーザに対して，専用のアプライアンスの割当てや，専用のソフトウェアをインストールし適切な設定を行う必要があり，これにはある程度の人的サポートを必要とする．このことから，従来の thin client は，インターネットを利用している人すべてが利用者となりうるような “広く開放されたサービス” の基盤として利用するには不向きである．

一方で，thin client 以外の VDI も存在している．WebShaka が開発した YouOS[1)] は JavaScript で書かれた Web アプリケーションである．YouOS はデスクトップマネージャのような Web アプリケーションを通して，Web ブラウザ上でさまざまな Web アプリケーションを統合的に扱うことができる．これは，thin client が提供する “仮想デスクトップ”

†1 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo

とは，技術的観点からはやや異なる．しかし，ユーザビリティの観点からすれば，VDI と同一視してもよい．このように，Web ブラウザ上で仮想デスクトップを動作させることのできるシステムを，新たに WebVDI と定義する．

WebVDI は，ユーザが利用する端末にコモディティ化された Web ブラウザと，その上で動作する JavaScript のみで実装された Web アプリケーションを使用する．そのため，インストールレスであり，“特定の組織に所属しない人に対する”，広く開放されたサービス基盤としての利用を考えると，ユーザにとって WebVDI は thin client に比べて導入が簡単であるという利点がある[⋆1]．しかし，WebVDI は Web アプリケーションであるため，既存のソフトウェア資産を引き継いで使用するのが難しい．

そこで我々は，既存のアプリケーションを再利用できる新たな WebVDI を提案する．また，最終的な目標としては，ユーザへは独立した 1 つの計算機環境を提供できる一方，サービスの提供基盤たるサーバの安全性は最大限に確保できる WebVDI を目指している．

本論文では，多くの VMM が KVM[⋆2] データを VNC のプロトコルで外部に出力できる点に注目し，VMM 上で動作するゲスト OS を Web ブラウザ経由で操作することを可能とするシステムである VoXY を提案する．これにより，WebVDI と thin client の利点をそれぞれ引き継いだ新たな VDI が構築可能である．VoXY は端末としてコモディティ化された Web ブラウザが使用可能であるため，新たに専用の機器やソフトウェアを導入しなくてもよい．また，ユーザは VMM が提供する完全にアイソレートされた仮想計算機資源を利用するため，ユーザに高い自由度を与えることができる．本システムはいわば，Web ブラウザを通じて計算機資源の貸し出しを行うシステムである．

本論文の構成は以下のとおりである．まず 2 章で既存の VDI の分析を行い，VoXY が持つ機能の必要性について述べる．3 章で提案手法の概略を示した後，4 章で開発した VoXY に関する詳細な解説と，VoXY が使用するプロトコル設計について論じる．5 章で，VoXY に対していくつかの定量的評価を行う．6 章で関連研究の紹介と，7 章でまとめを示す．

2. VDI の現状

本章では既存の VDI として，thin client と WebVDI を取り上げ，問題分析を行う．

2.1 thin client

thin client を実現するためのプロトコルとして代表的なものには X[2]，ICA[3]，RDP[4]，VNC[5]，SLIM[6] などがあげられる．これらはすべて “画面転送方式” のプロトコルである．この方式では仮想デスクトップの情報はすべてディスプレイに表示される画面データとして扱われる．そのため，今まで使用していたアプリケーションをそのまま引き継げるという利点がある．

さらに，画面転送方式のプロトコルは，2 つに分類することができる．X や RDP は画面データとして画面描画命令そのものを転送する[⋆3]．この手法は大きなデータを転送することなく，OS やウィンドウマネージャの画面を転送することができるため，高速であるという利点がある．しかし，この手法は使用する OS やウィンドウマネージャに強く依存し，移植が困難であるという問題点があるため，幅広いプラットフォームで使用できないという欠点がある．一方，ICA，SLIM では画面を単純な画像データとして扱う．そのため，特定システムに依存することなく，多くのプラットフォームに対応できるという利点がある．

VNC も “画面転送方式” のプロトコルである．しかしながら，VNC が利用するプロトコルは RFB protocol[7] として広く公開されており，多くの実装や亜種が存在している．そのため，同じ “画面転送方式” である ICA や SLIM と比べると，より一般的であると考える．

これらの thin client は専用のソフトウェアや専用ハードウェアとして提供されている．RDP は Windows 上でリモートデスクトップとして使用できる．VNC や ICA はソフトウェアとして実装されている．また，SLIM はハードウェアとして実装されており，Sun Microsystems から Sun Ray[8] として販売されている．

2.2 thin client 以外の VDI

VDI は thin client 以外にも存在する．WebShaka が開発した YouOS は，JavaScript で書かれた Web アプリケーションである．YouOS はウィンドウマネージャのようなシステムを通して，多くのアプリケーションを統合的に扱うことができる．thin client と同様に，アプリケーションや，アプリケーションで作成したデータは，YouOS が提供するサーバ側に保存され，クライアント側には保存されない．このような Web 上で動作する VDI（WebVDI）は，Web ブラウザ上で動作する “仮想デスクトップ” であるといえる．類似の WebVDI は YouOS 以外にも多数存在している．WebVDI はクライアントにコモディティ化された Web

⋆1 以後，“ユーザへの導入が簡単” という言葉はすべて，広く開放されたサービス基盤としての利用を考えると，ユーザにとっては，WebVDI のほうが従来の thin client よりも敷居が低いという意味で使用する．
⋆2 Keyboard，Video，Mouse のこと．
⋆3 フォント A を使用して，(x，y) の位置に背景色は白で “Hello world” と描画せよ，といった API レベルの命令．

ブラウザのみを使用している．そのため，すでに計算機を所有し Web ブラウザを使用している人であれば，即利用可能である．

また別に，Web ブラウザ上で thin client を扱うシステムとしては Java Applet 版の VNC Viewer や Flash として実装された VNC Viewer などがある．このソフトウェアは Web ブラウザ上で動作する Plug-in として動作しており，専用のソフトウェアが Web ブラウザ側に埋め込まれる形になっている．

2.3　既存システムの問題点

まず，thin client を実現するための既存のプロトコルは，すべて専用のクライアント（ハードウェア，ソフトウェア）を使用する．そのため，ユーザが新たに導入するにコストが高く，広く開放されたサービス基盤としての利用には不向きである．たとえば，Web ページ上で仮想計算機環境（仮想デスクトップ）を購入できるようなサービスを構築することを考える．このとき，ユーザに Web ブラウザ上でクレジットカード番号を入力させ，購入後に専用クライアントを新たにインストールさせるよりも，購入した仮想計算機を使用できる環境をそのまま Web ブラウザ上で与えるほうが，ユーザは受け入れやすい．

また，既存の thin client は，公共無線 LAN のように，HTTP，HTTPS のみしか通信を許されていないような，制限の強いネットワーク環境下では使用できない．このような制限下においては，Web ブラウザ経由で ActiveX や Java アプレットとして実装されたソフトウェアをインストールさせることで，SSL-VPN 経由で利用することができるが，結果的にはユーザに Web ブラウザ以外のソフトウェアも意識させる必要があるため，好ましくない．

YouOS などといった既存の WebVDI を使用すれば，Web ブラウザの機能と HTTP のみで，仮想デスクトップが利用できる．通信経路の暗号化としては，Web ブラウザにデフォルトで実装されている SSL（https）が使用できるため，他の Plug-in はいっさい必要ない．しかし，WebVDI は既存のソフトウェア資産を再利用することは難しいという問題点がある．

3. 提 案 手 法

本章では 2.3 節でまとめた既存 VDI の問題点をもとに，我々の提案手法を述べる．

我々は既存のソフトウェアやデータを引き継げる thin client の特性と，WebVDI が持つ純粋な HTTP のみによる通信の利点をそれぞれ持った新たな VDI が必要であると考える．そこで，WebVDI と同様にユーザが使用する端末には Web ブラウザを使用する．なるべく多くの Web ブラウザに対応するため JavaScript で実装された単純なクライアントを使用する．そのため，Web ブラウザ上での Plug-in などは必要ない．

また，VDI のサーバ側は VMM を使用したアーキテクチャを採用する．これによりサーバは一元化したまま，各ユーザに対して独立した計算機を提供可能である．さらに，仮想マシンであるため，ユーザのニーズに応じて，設定済みの新たな計算機を導入したり，廃止したりすることも容易である．我々の最終的な目標は，Web ブラウザ上から計算機を借り貸しするシステムを提案することにある．

4. VoXY（Vnc Over http proXY）

我々は，HTTP を使用して仮想マシンを Web ブラウザ上から操ることができるシステム，VoXY を開発した．VoXY のシステム概念図を**図 1** に示す．

仮想マシンの画面データ転送には，VNC で利用されている RFB protocol を使用する．仮想マシンの画面データを RFB protocol として出力できる VMM は多い．代表的なものとして．Xen[9]，QEMU[10]，KVM[11]，VMware[12] などがあげられる．また，これ以外にも RFB protocol 3.8[7] に準拠している VMM であれば，どれでも使用可能である．

さらに，VoXY はマルチユーザに対応している．複数人のユーザそれぞれに 1 台ずつ仮想マシンを割り当てたり，複数人のユーザに対して 1 台の仮想マシンを共有させたりすることも可能である．httpd は cgi スクリプトの実行に対応しているものであればどれでも使用することができるため，ロードバランサなども既存のものが使用できる．VMM と管理サー

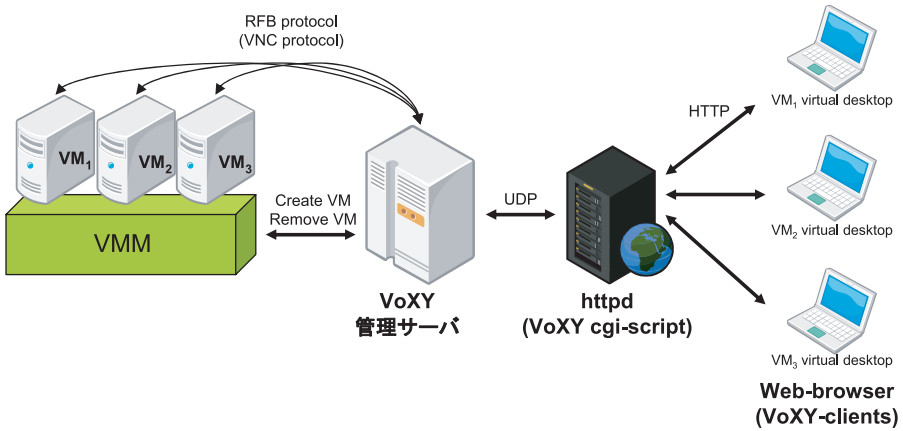


図 1　VoXY アーキテクチャの概念図
Fig. 1　Overview of VoXY architecture.

バは分離した設計になっているため，VMM の数を増やすことも容易である．このように，各コンポーネントを分離したアーキテクチャを採用したことで，VoXY 管理者は柔軟でスケーラブルなシステム構成を行うことが可能である．本章では，VoXY を構成するコンポーネントの解説と，VoXY が使用するプロトコルや，処理の流れについての解説を行う．

4.1 VoXY のコンポーネント

VoXY は大きく分けて，VMM の管理やユーザ管理などを行う VoXY 管理サーバと，Web ブラウザ上で動作する，JavaScript で実装された VoXY クライアント，VoXY クライアントと VoXY 管理サーバを接続するための，VoXY cgi スクリプトの 3 つのコンポーネントに分けられる．

4.1.1 VoXY 管理サーバ

VoXY 管理サーバは，VMM を管理する VoXY VMM Manager と，VoXY Window Manager の 2 つのコンポーネントに分けられる．それぞれの詳細を以下に示す．なお，VoXY 管理サーバの実装は C++で行われており，4,000 行ほどである．

VoXY VMM Manager　VoXY VMM Manager は VMM に対して，仮想マシンの起動，新規作成，削除などの命令を発行する．仮想マシンの画面データを RFB protocol で出力できる VMM は多数存在しているものの，仮想マシン自体の制御[⋆1]に関して，VMM は統一されたインタフェースを持っているわけではない．そのためこの部分は VMM ごとに異なる実装を行う必要がある．本論文では QEMU 用の VMM Manager の実装を行った．

VoXY Window Manager　VoXY Window Manager は主に，ユーザの管理と画面データの管理を行う部分である．VoXY VMM Manager が管理する仮想マシンに対してユーザを割り当てたり，画面データを取得したりする．

4.1.2 VoXY CGI スクリプト

VoXY cgi スクリプトは，httpd 上で動作する VoXY クライアントと VoXY 管理サーバ間の中継を行うための CGI スクリプトである．通信は UDP ソケットを使って独自のプロトコルで行う．UDP ソケットで通信を行う理由は，httpd からの要求に対して CGI の実行プロセスが立ち上がり，CGI スクリプト実行後は CGI の実行プロセスは終了するという CGI の性質を考えると，TCP ソケットによる通信では，ソケットの接続/切断が連続して発生するため，TCP のコネクションのコストが高いと考えたからである．

⋆1 パワーオン，リセット，仮想マシンの作成や削除といった操作．

今回の実装では，Common Lisp で書いた CGI スクリプトを利用することで，VoXY 管理サーバと httpd の通信を行っている．しかし，この部分は，httpd と VoXY 管理サーバ間の通信を行える機構を備えたものであれば，どのようなものでもよい．また今回は，VoXY cgi スクリプトと VoXY 管理サーバを同一計算機内においているため，パケットロスの可能性は低い．そのため，UDP による通信を選択した．しかし，2 つが分離し信頼性を持たせなければならなくなったとき，TCP に変更することは容易である．

4.1.3 VoXY クライアント

VoXY クライアントは Web ブラウザで動作するクライアントである．これは，仮想マシンの作成，削除といった VMM のインタフェースと，仮想マシンの画面データの表示や，仮想マシンに対して送信する Keyboard や Mouse のイベント取得などを行う．通信は Ajax を用いて，非同期に httpd 上にある VoXY cgi スクリプトへデータを送る．このクライアントは JavaScript を使用して実装されており，400 行ほどで実現されている．コードは多くの Web ブラウザが共通して持つ機能のみを使用しているため，多くのブラウザ間での互換性は高い．

4.2 VoXY Window Manager による Web ブラウザへの描画手法

仮想マシンの画面を HTTP 経由で転送するにあたっての課題は，いかにして，JavaScript のみで仮想マシンから送られてきたピクセルデータを Web ブラウザ上に描画するかである．JavaScript を使用して，任意のピクセルデータを Web ブラウザ上に描画する手法はいくつか存在している．どの手法をとるかによって，VoXY プロトコルの設計が大きく変わってくる．

まず最初に仮想マシンモニタがサポートする VNC のプロトコルである RFB Protocol についての簡単な解説を行う．次に，JavaScript における画面描画手法を検討した後，HTTP 上で動作する VoXY のプロトコルについて解説する．

4.2.1 （VNC）RFB Protocol の概要

VNC の内部では RFB Protocol（Remote Frame Buffer Protocol）[7] と呼ばれるプロトコルが使用されている．現時点で最新の RFB Protocol は version 3.8 である．VoXY は RFB Protocol 3.8 に準拠するプロトコルを使用する VMM であれば，どれでも使用可能である．

RFB protocol はきわめてシンプルなプロトコルであり，クライアントをステートレスに実現可能である．画面データのフォーマットは，“与えられた x，y 座標に横，縦分だけピクセルデータを描画する” という単一の描画規則に基づいている．

表 1　主要ブラウザと拡張タグ
Table 1　The main browsers and extended tags.

Browser	Tags
FireFox	svg, canvas
Safari	svg (since Safari 3), canvas
Internet Explorer	vml

また，RFB protocol は Client-pull 型のプロトコルである．VNC クライアントが VNC サーバに向かって FramebufferUpdateRequest メッセージを送ると，VNC サーバから FramebufferUpdate メッセージが返る．VNC クライアントは FramebufferUpdateRequest メッセージの送信間隔を調整することができる．これにより，使用する回線状況に応じて通信トラフィックを調整することができる．

4.2.2　JavaScript での描画手法の検討

RFB protocol の仕様をふまえたうえで，Web ブラウザへの画面データの描画手法についての議論を行う．JavaScript を使用して，Web ブラウザに任意のピクセルデータを描画するいくつかの手法を以下に示す．

(a) 1 × 1の DIV タグによる描画　1 画素に 1 つの DIV タグを割り当て，それを縦横のピクセル数分だけ敷き詰める手法である．画素の色は DIV の color エレメントで指定する．JavaScript による単純な Draw ツールの実装などは，この手法が使われていることが多い．この手法は単純ではあるが強力であり，ピクセル数が低ければ十分使用に耐える．しかしながら画面データのように，大きなピクセル数を描画する場合にはブラウザへの負荷が高い．簡単な実験を行ったところ，描画速度が極端に落ち，ブラウザが応答しなくなることが分かった．そのため本手法は採用しない．

(b) 拡張タグを使用する　主要な Web ブラウザでは，JavaScript からピクセルデータを描画するための，拡張タグと API が用意されている．**表 1** に主要ブラウザと対応する拡張タグの一覧を示す．いくつかの拡張タグがあるが，機能自体はどれも同様のものが用意されている．また，HTML5 からは Canvas タグが標準[13] になる予定である．Canvas タグには，点を打つ，特定の四角形領域を特定の色で塗りつぶす，特定座標の四角形領域を，別の座標へコピーする，といったプリミティブな描画 API に加えて，画像データを，指定した座標へ表示するといったことも可能である．特定の画像データを扱う場合，画像データの指定の方法は，サーバにあるファイルのパスを指定する方法や，Base64 エンコードによるバイナリの表現などがある．実験のために，VNC サーバから送られてくる FramebufferUpdate メッセージごとに PNG ファイルを生成し，生成されたファイルごとに Canvas タグに描いてゆくという簡単なプログラムを作成した．その結果，描画速度が遅く[⋆1]．thin client として，十分なパフォーマンスを得ることができなかった．

(c) DOM による IMG タグの動的加工　画面全体をある程度の大きさを持つ正方形ブロックに分割する．Web ブラウザは IMG タグを使用して，分割された正方形ブロックを画面に敷き詰める．変更箇所があれば，該当する箇所を含むブロックを更新する．この手法は，単純な IMG タグの挿入と変更で実現されているため，少ない実装コストでほとんどすべての Web ブラウザに対応できる．また，試験的実装の結果では，thin client としてのパフォーマンスも良好であった．

以上，Web ブラウザで使用可能ないくつかの描画手法を検討してきた．その結果，最終的に，パフォーマンスや現時点での互換性といった総合的な観点から (c) の手法を採用した．

(b) に関しても，(c) と同様にいくつかの正方形ブロックに区切る手法で thin client としてのパフォーマンスを上げることは可能であると考える．また，将来的には Canvas タグが HTML5 で標準化され，各ブラウザ間での互換性も整うものと思われる．そのため Canvas タグを使用した描画手法に関しては，今後検討したい．

4.2.3　ブロックのサイズ

手法 (c) においては，各ブロックの大きさをどう決定するかという問題がある．適切なブロックサイズを決定するために予備実験を行った．

実験を行うにあたって，実験用の CGI スクリプトと JavaScript コードを書いた．実験用の CGI スクリプトは，任意の縦横の長さを指定すると，その大きさを持つ PNG イメージと XML データを生成する．XML には生成した PNG ファイルを，ブラウザのどの座標（x，y）に，どの大きさで（width，height）描画するかが書かれている．JavaScript のコードは，CGI スクリプトが生成した XML のデータをもとに DOM を操作し，IMG タグを挿入する．Web ブラウザは，IMG タグが挿入されると PNG データを httpd からダウンロードして画面に表示する．

いくつかの異なる縦横の幅を持つブロックで実験を行った．評価環境は，ブラウザに Internet Explorer 7.0 を使用し，httpd は Apache/2.2.8 を使用した．Internet Explorer と Apache 間のネットワークの遅延時間は考えない．画面解像度は 800 × 600 を使用した．

⋆1 ここでの “描画速度の遅さ” は “Canvas タグそれ自体の描画速度の遅さ” を意味するものではない．この手法による実装の全体を見たときに，thin client として大幅な遅延を感じるという意味である．

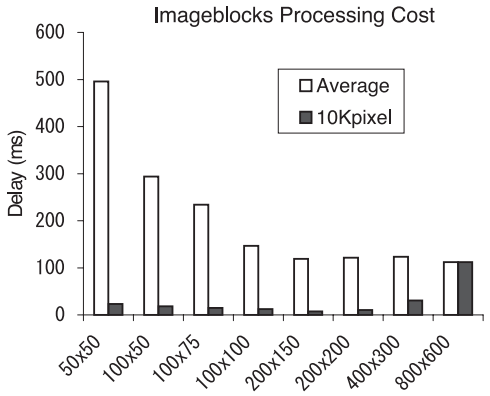


図 2　ベンチマーク結果
Fig. 2　Benchmark results.

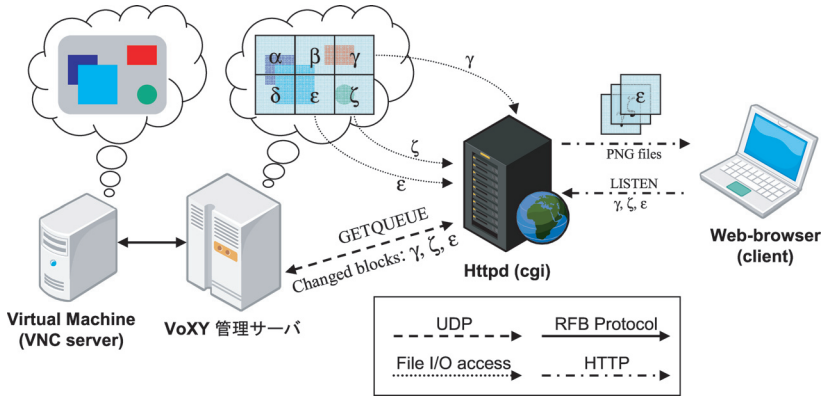


図 3　VoXY プロトコルの概略図
Fig. 3　VoXY protocol overview.

評価は 2 つの観点から行った．1 つは，800 × 600 の全画面を更新したときにかかる時間に着目した．2 つは，全画面更新ではなく，800 × 600 の一部分である 10 Kpixel 分のみを更新したときにかかる時間に着目した．この 10 Kpixel という値は，いくつかの実用的なアプリケーションでは，ユーザからのインプットの 50%近くが 10 Kpixel よりも小さい値の更新しか引き起こさないという Schmidt らの研究[6] をもとにした値である．1 番目の評価はブロックごとにそれぞれ 10 回測定して平均値を記録した値である．2 番目の評価は 1 番目の更新時間をもとに算出した値である．結果を**図 2** に示す．

この結果によると，画面全体の更新時間は，200 × 150 以上の大きさでは大差がない．これに加えて 10 Kpixel 分の更新時間を考慮すると，最適値は 200 × 150，200 × 200 のいずれかとなる．そこで，今回は正方形ブロックである 200 × 200 を選択した．

4.3　VoXY プロトコルの設計

4.2 節での画面議論をもとに VoXY プロトコルを設計した．プロトコルの概略図を**図 3** に示す．

4.3.1　VoXY プロトコルの詳細

管理サーバは VNC サーバへ TCP コネクションを確立する．VNC サーバ 1 台につき 1 つのスレッドを生成して，VNC サーバに向かって FramebufferUpdateReqest メッセージを送信する．その後は，VNC サーバからの返答を待つ．

VNC サーバから送られてきた FramebufferUpdate メッセージの画面データは，いったん管理サーバ内のメモリバッファに蓄えられる．ここで，画面データ全体は，複数の正方形タイルに分割されて管理される（図 3：α，β，γ，δ，ϵ，ζ）．それぞれの正方形タイルには，固有の ID と，ファイルディスクリプタが与えられる．つまり，各ブロックはそれぞれ 1 つの PNG ファイルに対応付けられる．また，それぞれのタイルの辺の長さは，4.2.3 項で議論したとおりである．管理サーバが VNC サーバから FramebufferUpdate メッセージを受け取ると，そのデータは，更新箇所を含むブロックへ送られる．管理サーバは変更があったブロックの内容を PNG ファイルに圧縮して書き出す．最後に，変更箇所があったブロックの ID を，QUEUE に書き出す．

4.3.2　LISTEN イベントによるポーリング

図 4 に VoXY プロトコルのシークエンス図を示す．クライアントとなる Web ブラウザ側は，Ajax による非同期通信を使用して，定期的に httpd へとポーリングを行う（図 4：(a1) の LISTEN end から，(c1) の LISTEN start までがポーリングの間隔である）．これは，サーバ側からの擬似的なプッシュを実現するためである[⋆1]．なお，ポーリングの間隔は 100 ms である．ポーリングの際に送信するコマンドは，LISTEN コマンドである．クライアントが発行した LISTEN コマンドを受け取ると，httpd 上で実行されている cgi スクリプトは UDP ソケットを使用して，管理サーバへ問合せを行う．このときに使用されるコマン

⋆1 ディスプレイの画面更新は，必ずしもユーザのアクションに対して行われるわけではない．

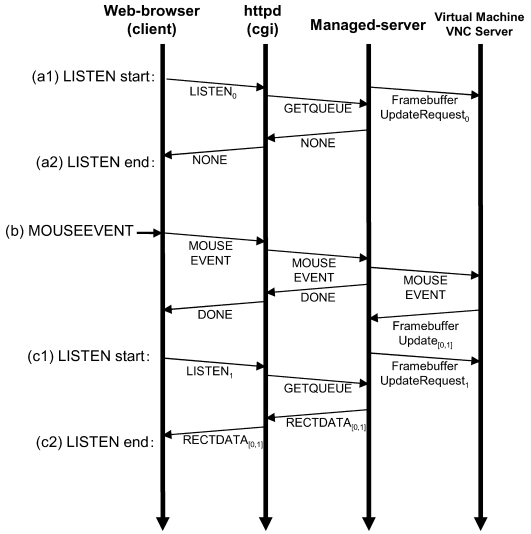


図 4　VoXY プロトコルシークエンス図
Fig. 4　VoXY protocol sequence diagram.

ドは GETQUEUE である．GETQUEUE コマンドに対し，VoXY 管理サーバは，キューに入っている変更のあったブロックのファイル名を返す．（図 3：γ，ζ，ϵ）このとき管理サーバが返す内容は，変更点があったブロックに対応する PNG ファイル名である．PNG ファイルのバイナリデータは，管理サーバがファイルに書き出す．最終的に，クライアントとなる Web ブラウザ側は，LISTEN コマンドの送信の結果として，変更のあったブロックの PNG ファイル名のリストを得る．クライアントは Web ページの DOM を操作して，変更のあったブロックを表示する IMG タグの src element を変更する．src element が変更されたことで，Web ブラウザは PNG のファイルのダウンロードを httpd に要求する．PNG ファイルのダウンロードが終わると，画像データがブラウザ上に表示される．

4.3.3　ユーザイベントの扱い

ユーザイベント（Keyboard，Mouse など）は，LISTEN イベントとは独立して送信される．マウスイベントが発生すると（図 4：(b) の MOUSEEVENT）マウスイベント監視用のループがその値を送信する．マウスイベント監視のループは，JavaScript の SetTimeout 機能を使用して，擬似的なスレッドを実現している．そのため，ループ間隔が短すぎるとブラウザへの負荷が高くなり応答しなくなる．ここでは，ループ間隔は 100 ms としている．これは Schmidt らの先行研究が示した，アプリケーションに入力されるユーザイベントの 70%が 10 Hz 以下である[6] という結果をもとにした値である．

図 4 が示すとおり，現時点での実装では，ユーザイベントのコマンドにはただちに DONE が返され，それ以外の値を返さない．試験的に LISTEN を取りやめ，ユーザイベントの結果として RECTDATA を返すプログラムを作成したところ，画面更新の遅延が目立つようになり，thin client の操作性が著しく悪化した．結果的に，LISTEN コマンドの間隔である程度バッファリングを行ったうえで結果を得るほうが，thin client としての性能がよくなることが分かった．

4.4　VoXY VMM Manager

4.2 節で，仮想マシン上での KVM データをどのように取り扱うかに関する議論を行った．本節では，仮想マシンを管理する VMM へのコントロールを行う VoXY VMM Manager について解説する．

VoXY VMM Manager は VMM に対して，仮想マシンの起動，作成，破棄，コピー，一時停止，などのコマンドを発行する．ここは使用する VMM ごとに異なる実装が必要である．今回作成した VoXY VMM Manager は QEMU のみをサポートしており，system() 命令を使用して仮想マシンの起動を行っている．

また，多数の異なる VMM を統一したインタフェースで操作するための機構として libvirt が提案されている．これは，VMM 上で動作する仮想マシンの制御を抽象化したライブラリであり，サポートする VMM の数も多い．このライブラリが提供する API を使用すれば，統一したインタフェースで多くの VMM をコントロールすることができる．VoXY VMM Manager の libvirt 対応に関しては，今後検討したい．

5. 評　　価

本章では，VoXY の評価を行った結果を記す．評価環境は以下のとおりである．

管理サーバと httpd，VMM はともに Intel (R) Core (TM) 2 Quad Q6600 @ 2.40 GHz, 2 Gbyte RAM，Intel Gigabit NIC からなるマシン上で動作させた．OS は Ubuntu Linux 8.04 である．なお，VMM は QEMU 0.9.0 を使用しており，kqemu アクセラレータを使用した．その上でゲスト OS として Fedora Core 6 を動作させたものを使用している．httpd は Apache/2.2.8 を使用した．クライアント側もサーバと同じマシンを使用して，OS は Windows XP SP2 である．クライアントとなる Web ブラウザは，Internet Explorer 6

表 2　ベンチマークアプリケーション
Table 2　Benchmark applications.

Category	Application
Image Processing	GIMP
Web Browsing	Mozilla FireFox
Word Processing	OpenOffice Writer
PIM	KAddressBook

表 3　ユーザへのタスク
Table 3　Task for application user.

Application	Task
GIMP	人間の顔を描け
Mozilla FireFox	特定の Web サイトを閲覧せよ
OpenOffice Writer	SICP の「はじめに」をタイプせよ
KAddressBook	自分のアドレス帳を作れ

を使用した．クライアントとサーバをつなぐハブは Gigabit HUB を使用している．サーバ側からクライアント側へ ping コマンドで測定したネットワークの遅延平均は 0 ms であった．そのためネットワークの遅延はないものとして扱う．

5.1　評 価 手 法

システムを評価するにあたって，我々は，まずはじめにいくつかの実用的なアプリケーションを選び出した．選定したアプリケーションを**表 2** に示す．これらのアプリケーションを，それぞれ 1 回ずつユーザに 5 分間使ってもらい，そのときの LISTEN コマンドが発行され，その結果として RECTDATA が返ってきてから，ブラウザにその RECTDATA が実際に描画されるまでの遅延時間を測定した．図 4 で示すと，(a1) から (a2) 間の時間に，ブラウザが PNG ファイルをダウンロードしてきて表示する時間を足した値である．なお，VMM による遅延影響がどの程度あるかを測定するため，これらのアプリケーションを QEMU 上で動作させたときと，QEMU なしで，アプリケーションを直接サーバ上で動作させ，X の VNC サーバを使用したときとを比較した．つまり，表 2 のアプリケーションにつき，VMM ありとなしの 2 回ずつ評価を行った．また，ユーザにはアプリケーションごとに課題を提示した．課題の詳細を**表 3** に示す．なお，いずれのアプリケーションにおいても，画面解像度は 800 × 600 である．

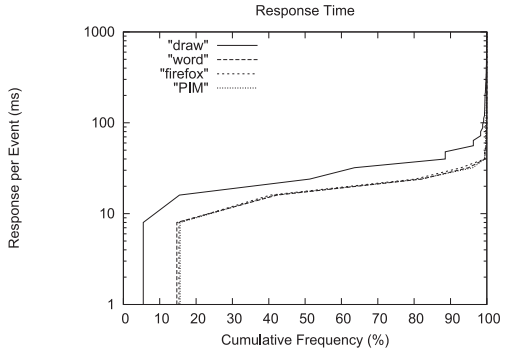


図 5　LISTEN イベントから，Web ブラウザ上の画面がアップデートされるまでの時間をまとめた，累積度数分布図．ヒストグラムの階級は 8 ms（VNC サーバの場合）
Fig. 5　Cumulative distributions of Web-browser display update service time per LISTEN event. Histogram bucket size is 8 ms (with VNC server).

5.2　VNC サーバにおける遅延時間

はじめに，VNC サーバを使用したときの測定結果を**図 5** に示す．グラフを見たとき，全体的に時間がかかっているアプリケーションは GIMP であった．これは，絵を描くときに，更新される部分が多かったからと考えられる．また，FireFox も，画面スクロールや他のサイトへのジャンプなどといった操作で，画面の大部分が更新される割合が多いため，2 番目に遅延時間が大きい．一方，最も時間が少ないアプリケーションは word である．これは，キータイプによって更新される画面が，局所的なものに過ぎないためである．

全体的に見て最も時間がかかっている GIMP でも，全体の 94%が 50 ms 以内の遅延時間で収まっている．また，最も更新時間の少ない word は 99%が 50 ms 以内の遅延時間で画面更新を行えていることが分かる．

5.3　QEMU 使用時における遅延時間

次に，QEMU を使用したときの遅延時間を測定した．結果を**図 6** に示す．当初予想では，VMM より VNC を使用したほうが全体的に遅延時間が短くなると考えていた．しかし，結果は反対であった．この原因は，使用した RFB protocol の実装が異なる点にあると考える．

同様に最も時間がかかっているのは GIMP であり，全体の 99%が 50 ms 以内の遅延時間で収まっている．また，最も更新時間の少ない word はほぼ 100%が 50 ms 以内の遅延時間で画面更新を行えていることが分かる．

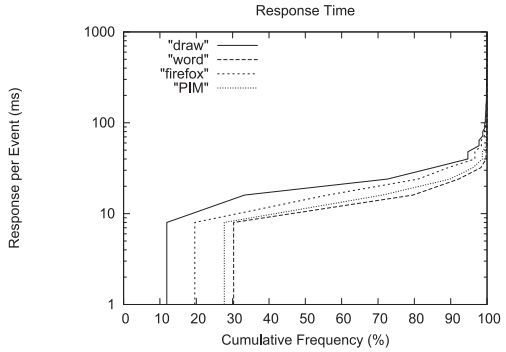


図 6　LISTEN イベントから，Web ブラウザ上の画面がアップデートされるまでの時間をまとめた累積度数分布図．ヒストグラムの階級は 8 ms（QEMU の場合）

Fig. 6　Cumulative distributions of Web-browser display update service time per LISTEN event. Histogram bucket size is 8 ms (with QEMU).

5.4　遅延時間のまとめ

5.3 節と 5.2 節のまとめを示す．結果から VNC サーバを使用したときと，VMM を使用したときとで，VMM の部分がボトルネックなるような現象は観測されなかった．

thin client の性能評価では対話応答性を評価することが重要である．ユーザインタフェースの研究によると，対話応答性の評価にあたっては，ユーザのイベントに対する更新の応答時間がよい検証値になり，人間が遅延を感じ始める時間は，50–150 ms ぐらいから始まるという研究結果[14] がある．5.3 節と 5.2 節で行った評価値（これを Lt と表記する）では，LISTEN コマンドが発行された時点で，ユーザのイベント発生時から見て，すでに遅延が発生している．一方，対話応答性の評価に必要な値は，図 4 で示すと，(b) から，(c2) が終わってその後 Web ブラウザの画面更新が終わるまでの時間（これを Rt と表記する）である．そのため，対話応答性となりうる遅延時間は，LISTEN のポーリング間隔を考慮して，最悪でも $Lt + 100$ ms 以下である．

評価において Lt の 94%が 50 ms であることを考えると，最悪で $Rt = 150$ ms の遅延が発生していることが分かる．$Rt = 150$ ms という値を，先ほどの 50–150 ms と比べると，VoXY は（概算値を見る限りでは）十分実用的な対話応答性を持つことが分かる．

また，この LISTEN ポーリング間隔は，実装の洗練でさらに縮めることが可能である．定期的なポーリング以外で，サーバからの（擬似的な）プッシュを実現することができる実装技術を使用することで，大幅に遅延時間を縮めることができる．

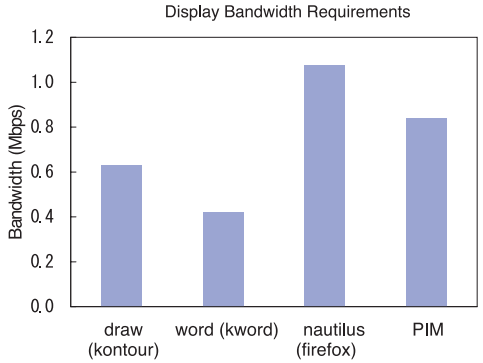


図 7　ベンチマークアプリケーションごとの平均消費帯域

Fig. 7　Average bandwidth consumed by the benchmark appications under the VoXY.

5.5　帯域測定

次に，各アプリケーションを使用したときの使用帯域を測定した．この評価では，解像度は 640 × 480 を使用しており，アプリケーションも 5.1 節で使用したものとは少々異なっている．**図 7** に結果を示す．最も画面更新量が多い nautilus（firefox）でも，1.076 Mbps 程度の帯域しか使用していない．また，kword（word）にいたっては 0.420 Mbps 程度である．

参考までに，一般的な家庭用ブロードバンド回線における下り帯域値を測定してみると，約 19.027 Mbps であった．このことから，VoXY は今日普及する家庭用ブロードバンド回線上であれば十分に使用可能であることが分かる．そのため，VoXY はインターネット利用者がすべて利用者となりうる，広く開放されたサービス基盤として使用できることが分かった．

6. 関連研究

FLOZ: Free Live OS Zoo[15] は，QEMU と Java Applet 版 VNC クライアント（VNC Viewer）を使用し，Web ブラウザ上でゲスト OS をブラウザ上で動作させることができるシステムである．利用者はサービス提供者があらかじめ用意した種々の OS イメージを自由に選ぶことができ，選択した OS の画面データが Web ブラウザ上の VNC Viewer に転送される．FLOZ は Web ブラウザ上で，プラグインとして動作する VNC クライアントを，QEMU が操る VNC サーバの TCP ポート（5900 版以降）に直結するという形をとっている．そのため，HTTP プロトコルの上ですべてを運用することができる本システムとは性

質が異なる．

The JPC Project [16] はオックスフォード大学によって開発されたシステムであり，Java Applet 上で x86 エミュレータを動作させ，その上で各種ソフトウェアを動作させようという試みである．本プロジェクトはテスト段階であるが，すでに Web 上で FreeDOS が動作するデモを見ることができる．The JPC Project で実装された x86 エミューレータは，bochs と同じようにソフトウェア上で ISA レベルでのエミュレートを行うものである．そのため VMM を使用する本システムとは性質が異なる．

VirtualDesktop，XenDesktop は，それぞれ VMware，Citrix によって開発された VMM 型の thin client である．これらのシステムは thin client を実現するにあたって 2.1 節で述べたプロトコルを使用している．そのため，thin client のプロトコルとして HTTP を使用する本システムとは大きく異なる．

7. おわりに

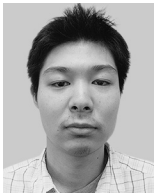
本論文では多くの VMM が Keyboard，Video，Mouse のデータを VNC のプロトコルで外部に出力できる点に注目し，VMM 上で動作するゲスト OS を Web ブラウザ経由で操作することを可能とするシステム，VoXY を提案した．これにより，WebVDI と thin client の利点をそれぞれ引き継いだ新たな VDI が構築できる．また仮想デスクトップには，VMM が提供する完全にアイソレートされた仮想計算機資源を提供する．そのため，ユーザに高い自由度を与えたまま，サーバ側を一元化することを可能とした．また，評価の結果，仮想マシンの画面データを現実的な数値で Web ブラウザへと転送できることが分かった．

次に今後の課題について述べる．4.4 節で述べたとおり，現在の VMM Manager は VMM として QEMU しかサポートしていない．そのため，QEMU 以外の VMM もサポートできるようにしたい．また，VMM として Xen を使用して，システムが 1 度にどの程度の数のユーザと仮想マシンをサポートできるかといった，スケーラビリティの観点からも評価を行いたい．

参考文献

1) WebShaka, I.: YouOS (2006). https://www.youos.com/
2) Nye, A.: *X Protocol Reference Manual*, O'Reilly, MA (1995).
3) Boca Research, Inc: *Citrix ICA Technology Brief*, Boca Raton, FL, Technical White Paper (1999).
4) Microsoft Corporation: *Comparing MS Windows NT Server 4.0, Terminal Server Edition and UNIX Application Deployment Solutions*, Redmond, WA, Technical White Paper (1999).
5) Richardson, T., Stafford-Fraser, Q., Wood, K.R. and Hopper, A.: Virtual Network Computing, *IEEE Internet Computing*, Vol.2, No.1, pp.33–38 (1998).
6) Schmidt, B.K., Lam, M.S. and Northcutt, J.D.: The interactive performance of SLIM: A stateless, thin-client architecture, *SOSP '99: Proc. 17th ACM symposium on Operating systems principles*, New York, NY, USA, pp.32–47, ACM (1999).
7) Richardson, T.: The RFB Protocol (2007). http://www.realvnc.com/docs/rfbproto.pdf
8) Tougaw, D. and Sanders, J.: SunRay: A Cost-Effective Desktop Computer Solution, *Computing in Science and Engg.*, Vol.4, No.1, pp.15–17 (2002).
9) Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the art of virtualization, *SIGOPS Oper. Syst. Rev.*, Vol.37, No.5, pp.164–177 (2003).
10) Fabrice, B.: QEMU Homepage (2008). http://fabrice.bellard.free.fr/qemu/
11) Kivity, A., Kamay, Y., Laor, D., Lublin, U. and Liguori, A.: kvm: the Linux Virtual Machine Monitor, *Proc. Linux Symposium*, pp.225–230 (2007).
12) VMware, Inc.: VMware. http://www.vmware.com/
13) Smith, M.: W3C: HTML 5 Publication Notes (2008). http://www.w3.org/TR/2008/NOTE-html5-pubnotes-20080610/
14) Shneiderman, B.: *Designing the User Interface: Strategies for Effective Human-Computer Interaction, 3rd ed.*, Addison-Wesley, MA (1998).
15) FreeOsZoo-Project: Free Live OS Zoo (2006). http://www.oszoo.org/wiki/index.php/Free_Live_OS_Zoo
16) JPCTeam: The JPC Project (2007). http://www-jpc.physics.ox.ac.uk/index.html

(平成 20 年 7 月 23 日受付)
(平成 20 年 11 月 15 日採録)

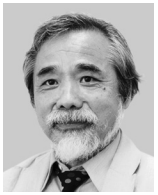


高橋　一志（学生会員）

1986 年生まれ．2008 年金沢工業大学工学部情報工学科卒業．現在，東京大学大学院情報理工学系研究科博士前期課程創造情報学専攻在学中．

笹田　耕一（正会員）

2004 年東京農工大学大学院工学研究科博士前期課程情報コミュニケーション工学専攻修了．2006 年同大学院工学教育部博士後期課程電子情報工学専攻退学．博士（情報理工学）（東京大学情報理工学系研究科 2007 年）．2006 年東京大学情報理工学系研究科助手，2008 年同講師（現職）．システムソフトウェア，とくに並列処理システム，言語処理系に関する研究に興味を持つ．



竹内　郁雄（正会員）

1969 年東京大学理学部数学科卒業．1971 年同大学院理学系研究科数学専攻修士．同年日本電信電話公社武蔵野電気通信研究所．記号処理システムの開発等に従事．1997 年電気通信大学情報工学科教授．2005 年東京大学情報理工学系研究科教授．博士（工学）．2000 年より未踏ソフトウェア創造事業プロジェクトマネージャ．2002 年から 5 年間（独）防災科学技術研究所に兼務して IT 防災の研究開発に携わり，現在も IT 防災研究プロジェクトに参加している．