2008－OS－109 (12)
2008／8／7

WebVDIのためのVNC Proxy

高橋 一志 † 笹田 耕一 † 竹内 郁雄 †

Virtual Desktop Infrastracture (VDI) とは，仮想デスクトップ環境を提供するための基盤技術である．VDIを実現する従来のThin-Clientでは，HTTP上での利用ができなかった．一方，YouOSなどのHTTP上で実現されているVDIでは，既存のソフトウェア資産を利用することができなかった．そこで，広く利用されているThin-Clientを実現するためのシステムであるVNCに着目し，Webブラウザで容易にVDIを実現するためのシステムであるVoXY: VNC over HTTP Proxyを開発した．VoXYは，VNCで利用されているRFBプロトコルをHTTPへ変換するプロキシである．評価の結果，実用的な性能でWebVDIを実現できることがわかった．

Proposal and implementation of VNC proxy for WebVDI

KAZUSHI TAKAHASHI ,† KOICHI SASADA † and IKUO TAKEUCHI †

Virtual Desktop Infrastructure (VDI) is base technology to achieve Virtual Desktop Environment. Existing thin-clients can not be used over HTTP. On the other hand, existing software resources are not used on web-browser based VDI such as YouOS. Therefore we focus on VNC, which is widely used to implement thin-client systems and develop VoXY: VNC over HTTP Proxy. VoXY is a proxy from RFB protocol which used by VNC to HTTP protocol. Our results show that VoXY achieve practical performance to build WebVDI systems.

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，古いものでは古典的なThin-Clientシステムが挙げられる．有名なThin-Clientとしては，例えばSun MicrosystemsのSun Ray[11]，MicrosoftのWindows Terminal[7]，VNC[10]などがある．これらのシステムでは，計算機の画面データを何らかの方法で転送することにより，仮想デスクトップを実現している．

もともとのThin-Clientの目的は，高価だった計算機資源を複数の廉価な端末から同時に利用できるようにすることである．しかしながらThin-Clientの持つ中央集権的なアーキテクチャがセキュリティ対策にも有効であることがわかり，その価値が見直されている．現在ではThin-Clientを情報漏洩対策として使用する企業も数多く存在している．また，VPNとThin-Clientを使用して，外出先から社内のコンピュータにアクセスするという使い方も一般的になっている．さらに，Thin-Client端末は一般的な計算機に比べて消費電力が格段に低い．そのため，省エネルギーの観点からも注目されている．

一方で，Thin-Client以外のVDIも存在している．WebShakaが開発したYouOS[14]は，デスクトップマネージャのようなシステムを通して，Webブラウザ上で各種Webアプリケーションを統合的に扱うことができる．これは従来のVDIが提供する”仮想デスクトップ”とは，技術的観点からはやや異なる．しかしながら，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と考える．われわれはこのように，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とThin-Clientは，それぞれ大きな利点がある．WebVDIには誰でも簡単に使うことのできるWebブラウザがクライアントとして機能する点と，制約の多いネットワーク環境下においても通信可能なHTTPを利用する利点がある．Thin-Clientには従来のアプリケーションやデータをThin-Client上で引き続き使用できるという利点がある．

われわれは有名なThin-ClientであるVNCに注目し，HTTPとJavaScriptを使用した，Webブラウザベースの Thin-Clientを実現するためのVNC proxy，VoXYを提案する．VoXYによって，Thin-ClientであるVNCをHTTP上で利用することが可能となる．

† 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

VoXY は WebVDI と従来の Thin-Client の利点をそれぞれ継承することができる．

本論文の構成を以下に示す．2 章にて，WebVDI と Thin-Client の利点と欠点を分析し，VoXY が持つ機能の必要性について論じる．3 章にて，開発した VoXY の簡単な概要を説明した後，VoXY プロトコルの設計について論じる．4 章にて，VoXY に対していくつかの定量的評価を行う．5 章で関連研究の紹介と，6 章にてまとめを示す．

2. Thin-Client と WebVDI の現状

本章では，WebVDI と Thin-Client の現状を分析した上で，問題分析を行う．

WebVDI 上で仮想デスクトップを実現し，利用していこうという動きはすでにある．YouOS に代表されるいわゆる WebOS では，ユーザビリティの観点からはすでに既存デスクトップと比べて遜色のないものができている．しかしながら，YouOS は JavaScript で書かれた Web アプリケーションである．そのため，本格的に利用していくにあたっては，今まで利用していたソフトウェアや，データの引継ぎを断念しなければならないという問題点がある．

次に，計算機の画面データを転送する Thin-Client システムを考える．この方法であれば，使用している OS やシステムにもよるが，大抵はソフトウェアやデータを Thin-Client 上から引き続き使用できる．しかし，従来の Thin-Client のプロトコルでは．フリースポットのような制約の強いネットワーク環境において使用することが不可能である．こういったネットワークでは，SOCKS や VPN を併用することで利用することもできるが，ユーザに VPN や SOCKS を意識させる必要があり，簡単ではない．

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．これらの技術は Web ブラウザ上で Plug-in として動作する．しかし，これらは裏で TCP コネクションを VNC server と確立しており，前述の Thin-Client と同様の問題点がある．また，これらが必要とする Flash Player や JavaApplet といった Web ブラウザの Plug-in は，どの Web ブラウザ上であっても必ずしも使えるというわけではない．

われわれ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 による通信の利点を，それぞれ持った Thin-Client システムが必要であると考える．エンド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 等は必要ない．

Thin-Client を実現するためのプロトコルは数多く存在している．代表的なものには X[8]，ICA[3]，RDP[7]，VNC[10] などがあげられる．この中でも VNC は特に有名である．VNC が使用するプロトコル（後述する RFB Protocol[9]）は特定のシステムに依存しない Low-level での pixel データをやり取りする．そのため，移植が容易であり多くの実装や亜種が存在している．VNC は主要な OS 上ではたいてい利用することが可能である．

さらに近年では，仮想マシンモニタ（VMM : Virtual Machine Monitor）上にて，ゲスト OS の KVM☆データの通信のために，VNC が使用されている．現在出回っている多くの VMM☆☆では，VNC のプロトコルをサポートすることが事実上標準となってきている．

このように，VNC は古くから存在するものの，その利用形態は現在にいたるまで成長し続けており，ますますメジャーな存在となっている．

まとめると，われわれは，VNC で利用されているプロトコルを中継して，HTTP に変換する VNC proxy である VoXY を提案する．**図 1** にいくつかのプラットフォーム上での VoXY 利用例を示した．

3. VoXY（Vnc Over http proXY）のアーキテクチャと実装

われわれは，HTTP を使用した Web ブラウザ上での Thin-Client を実現するため，VNC で利用されているプロトコルを HTTP に変換する VNC proxy，VoXY を開発した．VoXY のシステム概念図を**図 2** に示す．

VoXY はマルチユーザに対応しており，1 つの VNC server の画面を，同時に複数の Web ブラウザ上に表示することが可能である．また，複数の VNC server を扱うことも可能であり，ユーザごとに異なる VNC server を対応付けることも可能である．

サポートする VNC server は，RFB protocol 3.8[9] に準拠しているものであればすべて使用可能である．また，httpd は cgi-script の実行に対応している httpd であれば，どれでも使用することができる．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理由は，利用者が柔軟なシステム構築を行えるためである．次に開発した内容に対して詳細な解説を行う．

3.1 VoXY のコンポーネント

VoXY は VoXY managed-server と，httpd 上で動作する VoXY cgi-script，Web-browser 上で実行される JavaScript で実装された VoXY client からなる．それぞれのコンポーネントの詳細を以下に示す．

☆ Keyboard Video Mouse

☆☆ VMware(version 6 から)[13],QEMU[2]，Xen[1] などは，VNC プロトコル対応である．

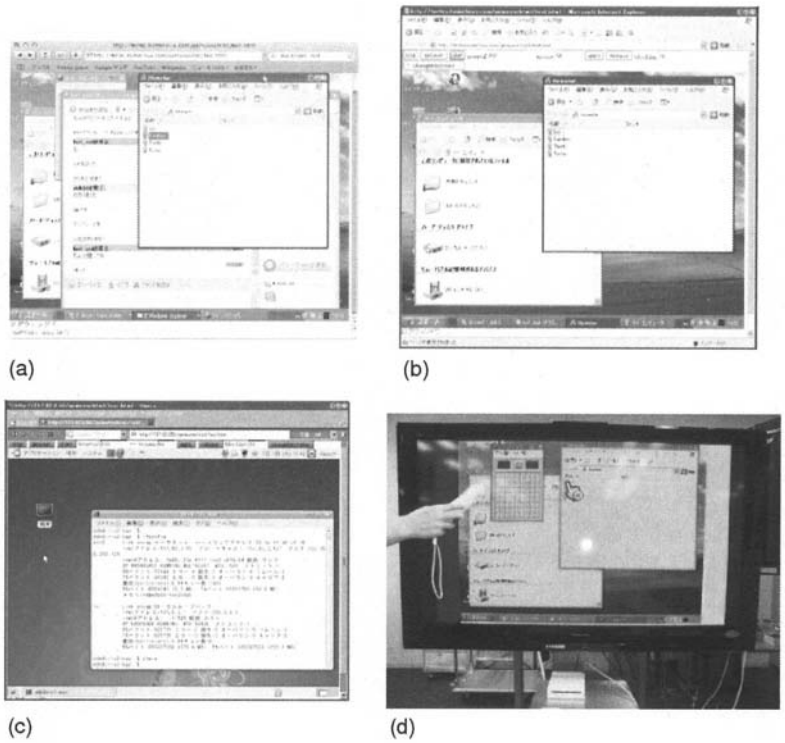

図 1 VoXY 使用例の数々: (a) WindowXP のデスクトップを Safari から. (b) WindowsXP のデスクトップを Internet Explorer から. (c)Ubuntu Linux のデスクトップを Opera から. そして, WindowsXP のデスクトップを任天堂 Wii から見ている.

Fig. 1 A variety of desktop being accessed from different web-browser: (a) a WindowsXP desktop from a Safari, (b) a WindowsXP desktop from a Internet Explorer, (c) a Ubuntu Linux desktop from a Opera, and (d) a WindowsXP desktop using Nintendo Wii

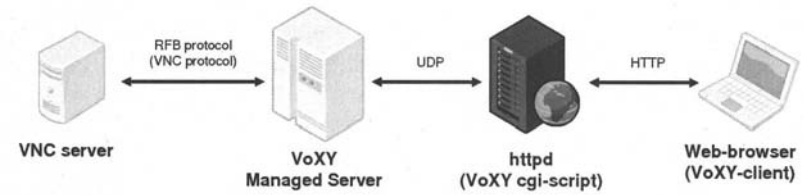

図 2 VoXY アーキテクチャの概念図

Fig. 2 VoXY architecture

3.1.1 VoXY managed-server

VoXY managed-server は, RFB protocol を使用して, 直接 VNC server と通信を行うためのものである. 1 つの managed-server で複数の VNC server と, それを扱う複数のユーザ管理を行うことができる.

managed-server は, VoXY client から送られてくるユーザのイベントを受信すると, RFB protocol を使用して, VNC server へと送信する. VNC server から得た画面データもここにすべて集まり, それから VoXY client へと送られる. 実装は C++で行われており, 4,000 行ほどである.

3.1.2 VoXY cgi-script

VoXY cgi-script は, httpd 上で動作する VoXY client と VoXY managed-server 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. 実装は Common

Lisp で行われており，400 行ほどである．

3.1.3 VoXY client

VoXY client は，画面データの表示，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-script へとデータを送る．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．JavaScript のコードは必要最小限となっており，多くのブラウザ上で動くよう互換性が高く設計されている．

3.2 Web ブラウザへの描画手法

VoXY を実現するにあたっての課題は，いかにして，JavaScript のみで VNC server から送られてきた Pixel 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 Pixel 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．どの手法を取るかによって，VoXY プロトコルの設計が大きく変わってくる．

まず最初に VNC が使用している 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．

3.2.1 (VNC)RFB Protocol の概要

VNC の内部では RFB Protocol（Remote Frame Buffer Protocol）[9] と呼ばれるプロトコルを使用している．現時点で最新の RFB Protocol は version 3.8 である．VoXY は RFB Protocol 3.8 に準拠している VNC server であれば，どれでも使用できる．

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．

RFB protocol は Client-pull 型のプロトコルである．VNC client が VNC server に向かって FramebufferUpdateRequest メッセージを送ると，VNC server から FramebufferUpdate メッセージが返る．VNC client 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．

3.2.2 JavaScript での描画手法の検討

RFB protocol の仕様を踏まえたうえで，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 Pixel データを描画するいくつかの手法を以下に示す．

(a)1x1 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 Pixel 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，Pixel 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ように，大きな Pixel 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わかった．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 Pixel 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[6] 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 server 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☆，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も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．

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

☆ 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．

☆☆ 現時点での実装は，画面データの横，縦の最大公約数を取った，正方形ブロックである

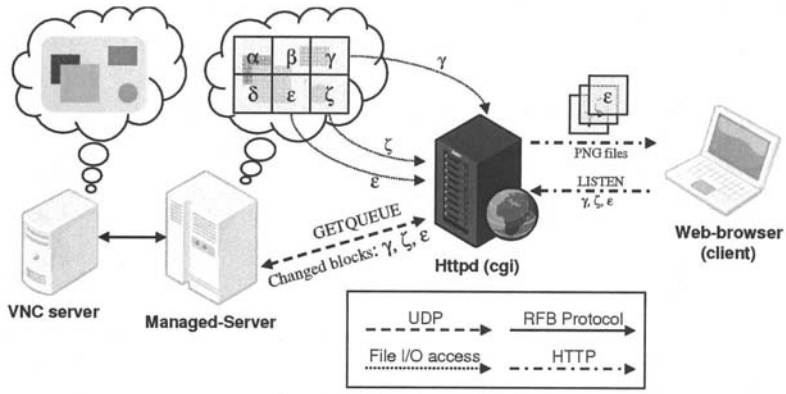

図 3 VoXY プロトコルの概略図
Fig. 3 VoXY protocol overview

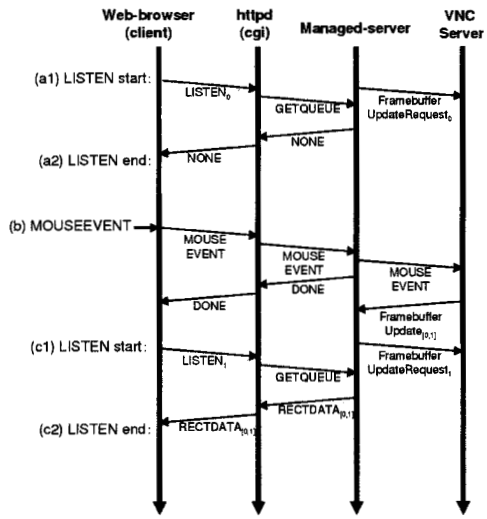

図 4 VoXY プロトコルシークエンス図
Fig. 4 VoXY protocol sequence diagram.

以上，Web ブラウザで使用可能ないくつかの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．

3.3 VoXY プロトコルの設計

3.2 での画面議論を元に 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．

3.3.1 VoXY プロトコルの詳細

Managed server は VNC server へ TCP コネクションを確立する．VNC server 1 台につき 1 つのスレッドを生成して，VNC server に向かって Framebuffer UpdateReqest メッセージを送信する．その後は，VNC server からメッセージが返ってくるまで待つ．VNC server 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 Managed server 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:$\alpha, \beta, \gamma, \delta, \epsilon, \zeta$)．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，画面解像度の縦横の長さの最大公約数を取った．これは，小さすぎるブロックの転送では，Thin-Client としての性能を上げることができなかったからである．Managed Server が VNC server から Framebuffer-Update 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．Managed Server 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．

3.3.2 LISTEN イベントによる polling

図 4 に VoXY プロトコルのシークエンス図を示す．Client 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 polling を行う (図 4 : (a1) の LISTEN end から，(c1) の LISTEN start までが polling の間隔である)．これは，Server 側からの擬似的な Push を実現するためである☆．なお，polling の間隔は 100ms である．polling の際に送信するコマンドは，LISTEN コマンドである．Client

☆ 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

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-scriptはUDPソケットを使用して，Managed Serverへ問い合わせを行う．このときに使用されるコマンドはGETQUEUEである．GETQUEUEコマンドに対し，Managed serverは，QUEUE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Managed Serverが返す内容は，あくまでファイル名だけである．PNGファイルのバイナリデータは，Managed serverがファイルに書き出す．最終的に，Client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．Client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．

3.3.3 ユーザイベント

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100ms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10Hz以下である[11]という結果をもとにした値である．

現時点での実装は，ユーザイベントのコマンドは直ちに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わかった．

4. 評　　価

本章では，VoXYの評価を行った．評価環境は以下のとおりである．

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

Category	Application
Image Processing	KOffice - Kontour
Web Browsing	Gnome nautilus 1.0.4
Word Processing	KOffice - KWord 1.1.1
PIM	KAddressBook

Managed serverとhttpd，VNC serverはともにIntel Pentium D 2.80GHz, 550Mbyte RAM, FastEthernet NICからなるマシン上で動作させた．なお，VNC serverはQEMU 0.9.0のVNC server機能を使用しており，その上でゲストOSとしてCentOS 4を動作させたものを使用している．httpdはApache/2.2.4を使用した．クライアント側の環境はAthlon64 X2 4800+ (2.41GHz相当)．2.00GB RAM, Gigabit Ether net NICからなるマシン上で，Internet Explorer 6を動作させている．クライアントとサーバをつなぐハブはGigabit HUBを使用した．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0.091msである．

次に評価手法について述べる．われわれは，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，それぞれ一回ずつユーザに10分間自由に使ってもらい，画面更新の遅延時間と，転送量の平均値を計測した．このときの，使用解像度は640x480（300Kpixels）である．

4.1 画面更新の遅延

まずはじめに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．**図5**に測定した時間を集計した累積度数分布図を示す．

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nautilusである．これはWebブラウジングを行ったとき，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からである．一方，最も時間が少ないアプリケーションはkword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．

全体的に見てもっとも時間がかかっているnautilusでも，150msを超える更新時間が占める割合は，全体の約10%ほどである．また，nautilusを除いたアプリケーションにおいて，約60%が50ms程度の遅延時間しかなく，80%が100ms以下の遅延時間で画面更新を行えている事が分かる．

人間が遅延を感じ始める時間は，50-150msぐらいから始まるという研究結果がある[12]．上記の結果をこの値に照らし合わせると，ブラウザの画面更新自体は，大枠のアプリケーションにおいて，十分に現実的な値で行えているといえる．

4.2 転　送　量

次に，各アプリケーションにおけるLISTENイベントあたりのデータ転送量の平均値を示した．**図6**に

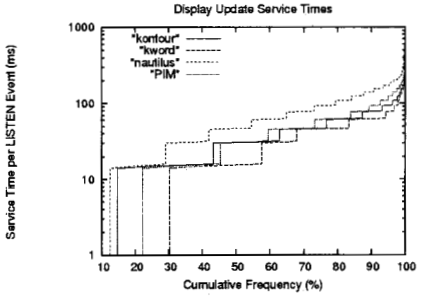

図 5 LISTEN コマンドを発行してから，Web ブラウザ上の画面がアップデートされるまでの時間をまとめた，累積度数分布図．階級は 1ms．

Fig. 5 Cumulative distributions of Web-browser display update service time per LISTEN event. Histogram bucket size is 1 ms.

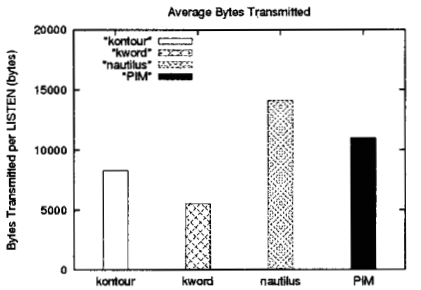

図 6 LISTEN コマンドで転送される画面データの平均バイト数．

Fig. 6 Average bytes transmitted by the benchmark applications.

結果を示す．

最も画面更新量が多い nautilus でも，平均して 14kbytes 程度の画像データしか転送していない．また，kword にいたっては，わずかに 6kbytes 程度の画像データしか転送していないことがわかる．

4.3 LISTEN コマンドの処理コスト

各コンポーネント間におけるボトルネックを探るため，LISTEN コマンドを送ってから，それが返るまでの時間をコンポーネントごとに分析した．ここで示す結果は，あくまで LISTEN イベントのみの測定結果である．LISTEN の結果を受け取った Web ブラウザが，画面データをダウンロードし，画面にレンダリングするまでの時間は含まれていない．

評価方法は，VNC Server 側の計算機上で動画を再生し，その時に LISTEN コマンドを発行してから結果が返ってくるまでの時間を JavaScript 上で測定し平均を取った．測定時間は 5 分間である．また，その際に各コンポーネントごとにかかった時間もそれぞれ個別に算出した．結果を**表 3** と**表 4** に示す．この表から，CGI Script の実行コストが，LISTEN の実行コストの 90% を占めていることがわかる．このことから Common Lisp で書かれた CGI Script の実行コストが非常に大きいことがわかる．なお，Common Lisp で何も処理を行わないプログラムを cgi として実行したところ，LISTEN round trip time の値は 20ms 前後であった．そのため，cgi の処理の中でも特に Common Lisp 処理系のスタートアップコストが大きい事が判明した．

表 3 各コンポーネントのベンチマーク 1

Table 3 Component benchmark result 1

Component	Cost (average)	Percent
LISTEN round trip time	33.386 ms	100%
CGI Script	30.238 ms	90%

表 4 各コンポーネントのベンチマーク 2

Table 4 Component benchmark result 2

Component	Cost (average)	Percent
CGI Script	30.238 ms	100%
Managed Server	5.020 ms	16%

5. 関連研究

本章では VoXY といくつかの既存ソフトウェアとの比較を行う．

FLOZ : Free Linux OS Zoo[5] は，QEMU と Java Applet 版 VNC Client （VNC Viewer）を使用し，Web ブラウザ上にてゲスト OS をブラウザ上で動作させることのできるシステムである．FLOZ は，Java Applet 版 VNC Client を，直接 QEMU が操る VNC Server の TCP ポート（5900 番以降）に直結するという形を取っている．そのため，通信プロトコルは通常の VNC Client と同様であり，HTTP のみを使用して通信が可能である VoXY とは性質が異なる．

Ajax VNC[4] は Web ブラウザ上でリモートデスクトップを実現するためのソフトウェアである．Ajax VNC の通信は VoXY と同様に HTTP のみを使用している．Ajax VNC は複数の VNC Server 管理したり，マルチユーザの管理を行うことはできない．本システムはあくまで単一計算機のリモートコントロールソフトウェアという位置づけが強い．

6. おわりに

本論文では Thin-Client である VNC プロトコルを HTTP に変換する VNC proxy，VoXY の提案と実装を行った．これにより制約の多いネットワーク環境下においても，HTTP を経由して VNC を使用することができる．Thin-Client としての性能評価では，選定したアプリケーションにおいて，描画遅延時間の大枠が 150ms 以内に収まっており，この値を既存研究[12] が提示する閾値に照らし合わせて，十分に実用的であ

ることが示された．また，別の評価実験では，VoXY のプロトコルの処理の中でも CGI script の実行コストが高いことも判明した．

今後の課題は，まず，3.2.2 章にて議論した Web ブラウザ上への描画手法のうち，Canvas タグでの評価実験を行う必要があると考える．また，3.3.3 にて議論した，サーバからの更新情報をどのように扱うかについての更なる検討も今後の課題である．また，CGI script の実行コストが高いことが判明したため．より負荷の低い，CGI script 以外の技術で置き換えることも今後の課題である．今回の評価実験は，正確なインタラクティブ性能の評価を行ったとはいえない．そのため今後は VoXY が Thin-Client としてどれだけ使いやすいのかを評価する，インタラクティブ性能評価が必要であると考える．

参 考 文 献

1) 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).
2) Bellard, F.: *QEMU*. http://bellard.org/qemu/.
3) Boca Research, Inc: *Citrix ICA Technology Brief*, Boca Raton, FL (1999). Technical White Paper.
4) Chan, H.: *Ajax VNC*. http://sourceforge.net/projects/ajaxvnc.
5) FreeOsZoo project: *Free Live OS Zoo (FLOZ)*. http://www.oszoo.org/wiki/index.php/Free_Live_OS_Zoo.
6) Michael Smith, W3C: HTML 5 Publication Notes, http://www.w3.org/TR/2008/NOTE-html5-pubnotes-20080610/ (2008).
7) Microsoft Corporation: *Comparing MS Windows NT Server 4.0, Terminal Server Edition, and UNIX Application Deployment Solutions*,, Redmond, WA (1999). Technical White Paper.
8) Nye, A.: *X Protocol Reference Manual*, O'Reilly, MA (1995).
9) Richardson, T.: The RFB Protocol, http://www.realvnc.com/docs/rfbproto.pdf (2007).
10) 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).
11) Schmidt, B. K., Lam, M. S. and Northcutt, J. D.: The interactive performance of SLIM: a stateless, thin-client architecture, *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*, New York, NY, USA, ACM, pp. 32–47 (1999).
12) Shneiderman, B.: *Designing the User Interface: Strategies for Effective Human-Computer Interaction. (3rd ed.)*, Addison-Wesley, MA (1998).
13) VMware, Inc.: *VMware*. http://www.vmware.com/.
14) WebShaka, I.: YouOS, https://www.youos.com/ (2006).