Virtual Network Computing

Tristan Richardson, Quentin Stafford-Fraser,
Kenneth R. Wood and Andy Hopper

Reprint from

IEEE Internet Computing
Volume 2, Number 1
January/February 1998

© 1998 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.

DOCKET ALARM

Find authenticated court documents without watermarks at docketalarm.com.

VIRTUAL NETWORK COMPUTING

TRISTAN RICHARDSON, QUENTIN STAFFORD-FRASER, KENNETH R. WOOD, AND ANDY HOPPER*
The Olivetti & Oracle Research Laboratory

VNC is an ultra-thin client system based on a simple display protocol that is platform-independent. It achieves mobile computing without requiring the user to carry any hardware.



The so-called network computer (NC) aims to give users access to centralized resources from simple, inexpensive devices. These devices act as clients to more powerful server machines that are connected to the network and provide applications, data, and storage for a user's preferences and personal customizations. We have taken this idea a stage further. In the virtual network computing (VNC) system, server machines supply not only applications and data but also an entire desktop environment that can be accessed from any Internet-connected machine using a simple *software* NC. Whenever and wherever a VNC desktop is accessed, its state and configuration (right down to the position of the cursor) are exactly the same as when it was last accessed.

In contrast to many recent Internet applications, which have focused on giving users access to resources located anywhere in the world from their home computing environments, VNC provides access to home computing environments from anywhere in the world. Members of the Olivetti & Oracle Research Laboratory (ORL) use VNC to access their personal Unix and PC desktops from any office in our Cambridge building and from around the world on whatever computing infrastructure happens to be available—including, for example, public Web-browsing terminals in airports. VNC thus provides mobile computing without requiring the user to carry any device whatsoever. In addition, VNC allows a single desktop to be accessed from several places simultaneously, thus supporting appli-

*Andy Hopper is also affiliated with Cambridge University Engineering Department.

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

THIN CLIENTS

The Virtual Networking Computing (VNC) system is a thin-client system. Like all such systems, it reduces the amount of state maintained at the user's terminal. VNC viewers are exceedingly thin because they store no unrecoverable state at the endpoint. This contrasts with systems like X Windows, and allows arbitrary disconnection and reconnection of the client with no effect on the session at the server. Since the client can reconnect at a different location—even on the other side of the planet—VNC achieves mobile computing without requiring the user to carry computing hardware.

Of course, VNC is not the only thin-client system. Others include those built around the Citrix ICA protocol (for example, Citrix's Winframe and **Insignia Solutions' Ntrigue**), **SCO's Tarantella**, **Graphon's RapidX**, and Microsoft's Windows-based **Terminal Server** (previously code-named Hydra). The problem with all of these systems except Microsoft's is that, unlike X, they use proprietary protocols, so reliable information about them is difficult to obtain. Citrix's ICA protocol is a popular mechanism for remote interaction with PCs, but it appears to be closely tied to the Microsoft Windows GUI, so it may not be an ideal general-purpose remote display protocol.

Microsoft has developed its own protocol, T.Share, based on the ITU T.120 protocol.[1] This is already used in Microsoft's NetMeeting conferencing software product. Preliminary details suggest that Microsoft's protocol is more like VNC than ICA—the Hydra white paper refers to a "super-thin" client.

We hope that VNC, or something like it, can become an open cross-platform standard for very-thin-client computing.

REFERENCE

1. "Microsoft Windows NT 'Hydra' and Windows-Based Terminals," white paper available at http://microsoft.com/ntserver/guide/hydrapapers.asp.

cation sharing in the style of computer-supported cooperative work (CSCW).

The technology underlying VNC is a simple remote-display protocol. It is the simplicity of this protocol that makes VNC so powerful. Unlike other remote display protocols such as the **X Window System** and **Citrix's ICA**, the VNC protocol is totally independent of operating system, windowing system, and applications (see the sidebar, "Thin Clients"). The **VNC system** is freely available for download from the ORL Web site at http://www.orl.co.uk/vnc/.

We begin this article by summarizing the evolution of VNC from our work on thin-client architectures. We then describe the structure of the VNC protocol, and conclude by discussing the ways we use VNC technology now and how it may evolve further as new clients and servers are developed.

THE ORIGINS OF VNC

The X Window System allows applications to display a user interface on a remote machine. ORL extended this functionality in our **Teleporting System** by allowing the user interface of a running X application to be dynamically redirected to a different display.[1,2] Teleporting has been in daily use at ORL for several years now. There are, however, several problems with X that restrict its use in the wide area and, in turn, restrict systems based on it, such as Teleporting:

- X requires the display machine to run an X server program. This heavyweight piece of software requires substantial resources, which machines such as NCs and personal digital assistants (PDAs) cannot be expected to run.
- The X security model makes it inherently dangerous to allow a remote machine to use your display. Accordingly, most system administrators stop X traffic from passing in or out of their sites.
- Application startup is extremely slow on high-latency links due to the number of round-trips performed by a typical application (though there are special proxies that alleviate this problem, such as Low Bandwidth X [LBX][3]).

In addition to these technical problems, there is also the nontechnical problem that X is not Windows, and the world is becoming increasingly Microsoft-dominated.

Videotile: An Ultra-Thin Client

In 1994, ORL built the **Videotile** as an experiment in ultra-thin-client technology. The Videotile is a display device with an LCD screen, a pen, and an ATM network connection. It was designed to display good-quality video, but we also wanted to use it to interact with applications. As a first experiment toward this end, we treated a remote computer screen as a video source and simply shipped the user interface as raw video onto the tile. This worked surprisingly well, but used a significant amount of bandwidth.

By adding a little more intelligence at the application side, we were able still to treat the user interface as video, but to send only those parts of the screen that changed. This idea developed into the VNC protocol.

Java: Access Through a Browser

When Sun Microsystems released the alpha version of the Java language and the HotJava browser in 1995, we realized we could implement the Videotile mechanism in Java to access applications through a Web browser. The thin-client paradigm made the adaptation to Java very straightforward. We wrote the original Java client in a day and the resulting class file was a mere 6 kilobytes in size. This eventually became the VNC applet described in more detail elsewhere.[4] Any Java-capable browser could now provide access to a user's desktop, giving the mobility of the Teleporting system, but on a global scale.

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

THE VNC PROTOCOL

The technology underlying the VNC system is a simple protocol for remote access to graphical user interfaces. It works at the framebuffer level and therefore applies to all operating systems, windowing systems, and applications—indeed to any device with some form of communications link. The protocol will operate over any reliable transport such as TCP/IP.

The endpoint with which the user interacts (that is, the display and/or input devices) is called the *VNC client* or *viewer*. The endpoint where changes to the framebuffer originate (that is, the windowing system and applications) is known as the *VNC server* (see Figure 1).

VNC is truly a "thin-client" system. Its design makes very few requirements of the client, and therefore simplifies the task of creating clients to run on a wide range of hardware.

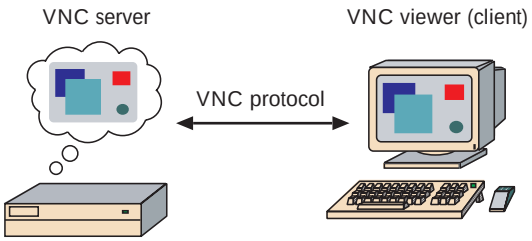

Figure 1. VNC architecture.

A Single Graphics Primitive

The display side of the protocol is based on a single graphics primitive:

> Put a rectangle of pixel data at a given *x*, *y* position.

At first glance this might seem an inefficient way to draw some user interface components. However, allowing various encoding schemes for the pixel data gives a large degree of flexibility in trading off parameters such as network bandwidth, client drawing speed, and server processing speed.

The lowest common denominator is the so-called *raw encoding*, where the pixel data for a rectangle is simply sent in left-to-right scanline order. All VNC clients and servers must support this encoding. However, the encodings actually used on a given connection can be negotiated according to the capabilities of the server and client and the connection between them.

For example, *copy-rectangle encoding* is very simple and efficient, and can be used when the client already has the same pixel data elsewhere in its framebuffer. The encoding on the wire is simply an *x*, *y* coordinate. This gives a position in the framebuffer from which the client can copy the rectangle of pixel data. This encoding is typically used when the user moves a window across the screen or scrolls a window's contents.

Most clients will support copy-rectangle encoding, since it is generally easy to implement, saves bandwidth, and is likely to be faster than sending raw data again. However, in a case where a client cannot easily read back from its framebuffer, the client could specify that it should *not* be sent data encoded this way.

A typical workstation desktop has large areas of solid color and text. One of our most effective encodings takes advantage of this phenomenon by describing rectangles consisting of one majority (background) color and "sub-rectangles" of different colors. There are numerous other possible schemes. We could use a JPEG encoding for efficient transmission of still images or an MPEG encoding for moving images. A pixel-data caching scheme could efficiently encode multiple occurrences of the same text character by referring to the first occurrence.

Adaptive Update

A set of rectangles of pixel data makes a *framebuffer update* (or simply, *update*). An update represents a change from one valid framebuffer state to another. In this sense, an update is similar to a frame of video. It differs, however, in that it usually affects only a small area of the framebuffer. Each rectangle may be encoded using a different scheme. The server can therefore choose the encoding most appropriate for the particular screen content being transmitted and the available network bandwidth.

The update protocol is demand-driven by the client. That is, an update is only sent by the server in response to an explicit request from the client. All screen changes since the client's last request are coalesced into a single update. This gives the protocol an adaptive quality: the slower the client and the network, the lower the rate of updates. On a fast network, for example, as the user drags a window across the screen it will move smoothly, being drawn at all the intermediate positions. On a slower link—for example, over a modem—the client will request updates less frequently, and the window will appear at fewer of these positions. This means that the display will reach its final state as quickly as the network bandwidth will allow, thus maximizing the speed of interaction.

Input

The input side of the VNC protocol is based on a standard workstation model of a keyboard and multibutton pointing device. The client sends input events to the server whenever the user presses a key or pointer button, or moves the pointing device. Input events can also be synthesized from other nonstandard I/O devices. On the Videotile, for example, a pen-based handwriting recognition engine generates keyboard events.

Connection Setup and Shutdown

To establish a client-server connection, the server first requests authentication from the client, using a challenge-response

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

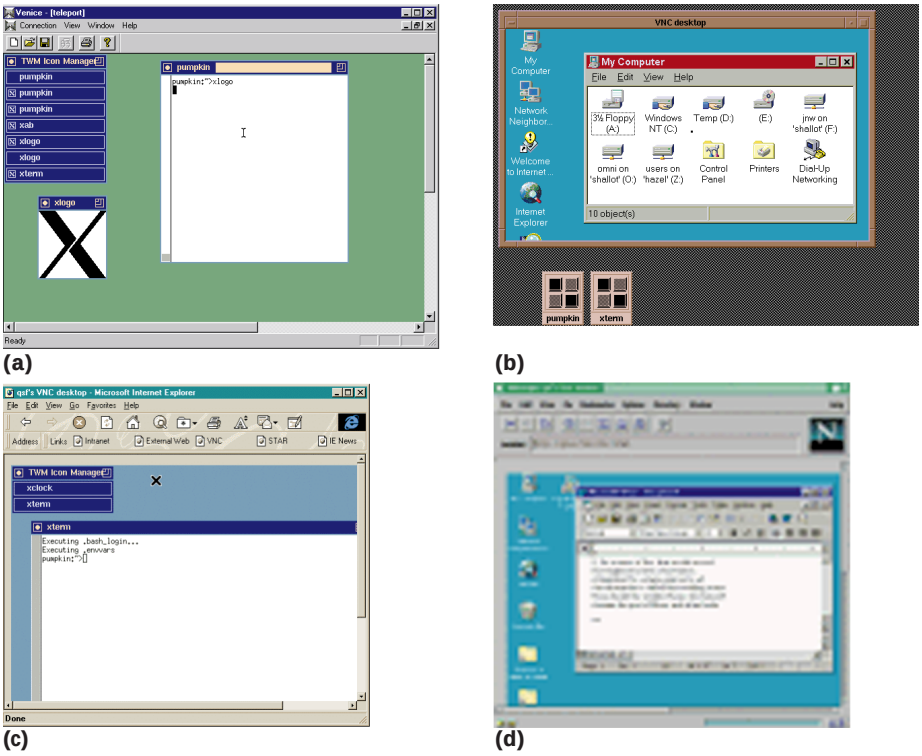

Figure 2. A variety of desktops being accessed from different viewers: (a) a Unix desktop from a Windows viewer, (b) a Windows 95 desktop from an X viewer, (c) a Unix desktop from a Java applet within Internet Explorer, and (d) a Windows desktop using Netscape on Unix.

scheme; the client typically requires the user to enter a password at this point. The server and client then exchange messages to negotiate desktop size, pixel format, and encoding schemes. The client requests an update for the entire screen, and the session begins. Because of the stateless nature of the client, either side can close the connection at any time without adverse consequences.

VNC Viewers

In day-to-day use, we prefer the more descriptive term *viewer* to the rather overloaded word *client*. Writing a VNC viewer is a simple task, as indeed it should be for any thin-client system. It requires only a reliable transport (usually TCP/IP), and a way of displaying pixels (either writing directly to the framebuffer or going through a windowing system).

We have written viewers for all the networked display devices available at ORL. These include the Videotile (the original VNC viewer), an X-based viewer (which runs on Solaris, Linux, and Digital Unix workstations), a Win32 viewer that runs on Windows NT and 95, and a Java applet that runs on any Java-capable browser (including Sun's JavaStation). Members of our lab use these viewers on a daily basis to access their personal computing environments.

The images in Figure 2 show a variety of X and Windows desktops being accessed from both Java and native X and Windows viewers.

VNC Servers

Writing a VNC server is slightly harder than writing a viewer. Because the protocol is designed to make the client as simple as possible, it is usually up to the server to perform any necessary translations (for example, the server must provide pixel data in the format the client wants). We have written servers for our two main platforms, X (that is, Unix) and Windows NT/95.

The X-based server was the first one we developed. A single Unix machine can run a number of VNC servers for different users, each representing a distinct VNC desktop. Each desktop is like a virtual X display, with a root window on which several X applications can appear.

The Windows VNC server was a little more difficult to create. Windows has fewer places to insert hooks into the system to monitor display updates, and the model of multiuser operation is less clearly defined. Our current server simply mirrors the real display to a remote client, which means that only a single VNC desktop is available from any one PC.

The X-based server, the X viewer, the Win32 server, and Win32 viewer can all fit on a single floppy disk.

We have also created "thin" servers which produce displays other than desktops, using a simple toolkit. A "VNC CD player," for example, generates a CD player user interface using VNC directly without any reference to a windowing system or framebuffer (see figure 3 on the following page). Such servers can run on very simple hardware, and can be accessed from any of the standard VNC viewers.

ANY USER INTERFACE, ANYWHERE

At ORL, we have used VNC to add mobility to workstation GUIs, where the concept of at least some form of remote interaction is not new. But the protocol's simplicity could allow it to be used on a much wider range of hardware. Consumer electronics devices, such as CD players, usually have a highly specialized user interface and typically employ customized phys-

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.