# VNC

Where it came from, where it's going

Tim Waugh <tim@cyberelk.net>



This article, which briefly explains what VNC is and what pieces of software have been written that use it and its ideas, was originally published (in Italian) in issue 2 of Red Hat Magazine. It was written using OpenOffice.org, and I have since reformatted it in DocBook. This version is more or less the same as the original; some URL references have been tweaked.

VNC is a thin client technology for using an X desktop (the graphical environment we use everyday). It stands for Virtual Network Computing and can be thought of as a software version of the "network computer" which never quite took off. If you have ever used the UNIX **screen** command, you might like to think of VNC as its graphical equivalent. You can start a desktop session, disconnect from it and then reconnect to it at will, even from a different X terminal. All of the applications are available exactly as before disconnecting.

In contrast, although X itself allows applications to display remotely, it does not provide this persistence; you cannot move an application window from one X server to another. Another thing that VNC allows you to do is to have several viewers all watching the same server. This can be useful in training environments for example.

The original VNC design and reference implementation arose from Olivetti Research Labs in Cambridge where it was used in a "teleport" system, whereby your desktop could follow you around the building, hopping to the nearest convenient terminal (an "active badge" on your clothing made this possible). Since those days AT&T bought the facility and the VNC engineers are now in a venture named RealVNC, a company which provides commercial support for the VNC product. Although it used to be the case that VNC releases were quite infrequent, since RealVNC took off they have been more regular. VNC is free software, licensed under the GNU GPL. This means that any modifications to the software must be made available in source form, a fact which has proven to be useful as we shall see later.

The differences between VNC and X should be made clear at this point, since the terminology can be quite confusing to the uninitiated. Both have clients and servers. For both, the server is where the desktop, with all the application windows, is stored. With X, the server is on the machine in front of you and drives the video card. An X client is an application that wants to display a window on the server, like a web browser or email application. A VNC client, in contrast, is an application that displays the desktop image to the user; normally the job of the X server. Both the VNC server and client perform tasks that the X server would normally do: maintaining the desktop image (in other words, the frame buffer) and displaying that image to the user.

The two independent parts are the client and of course the server and each can be running on a different platform. The protocol is designed to adapt to the amount of bandwidth available. It is also designed to make the client (the "viewer") very simple, without requiring extensive computing resources. This makes it ideal for thin client deployments, diskless if need be, and this perhaps is the most convenient form of network computer made from commodity components.

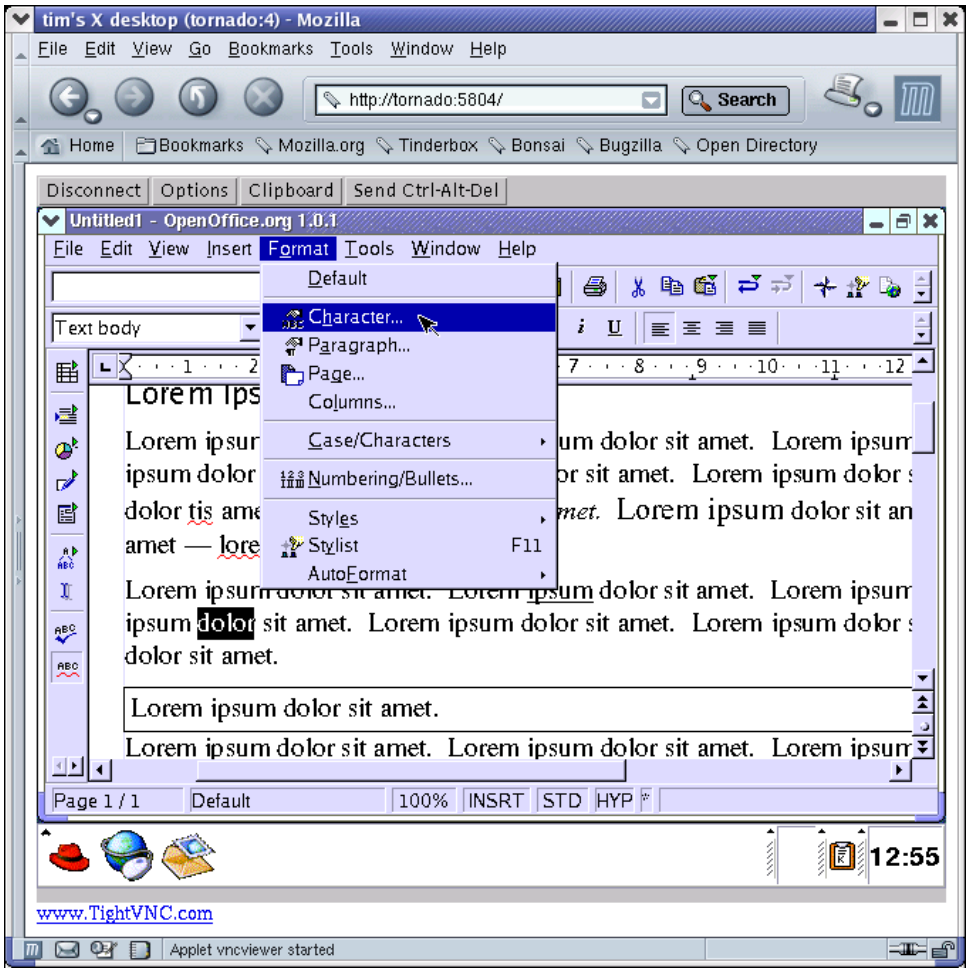
VNC is in fact quite pervasive—it can run on anything from high end servers to hand-held devices such as palm pilots. The uVNC project even has an 8-bit port; remember the Commodore 64?

There is even a VNC viewer written in Java. This means that you can use it from your web browser on a machine that has had no special set up. Just point the browser at a special port on the server and it will serve up a Java applet ready to go. The port number will be 5800 plus the display number for the VNC server. The display number is used to identify different VNC servers on the same machine, in exactly the same way as X works. In fact, VNC servers and X servers share the same display number range, and so if there is already an X server running (usually on display number 0), a newly started VNC server will get display number 1. To use the Java viewer for that server, point a browser at `http://machinename:5801/`. Note that the Java applet uses the normal VNC port number (5900 plus display number). So, if you want to use a VNC server that is behind a firewall, you might want to open up that port. (More likely you will want to tunnel VNC over SSH—detailed further on.)

Needless to say, VNC runs on Red Hat Linux as well of course. In the latest release (Red Hat Linux 7.3) there is a VNC service that can be started on boot-up. This will start VNC servers for a configured list of users, each with their own assigned display number. To configure it, you need to edit the file `/etc/sysconfig/vncservers` and set which users should have VNC servers started for them, and with which display numbers.

In Figure 1, you can see an OpenOffice.org document being edited using the VNC Java applet running in Mozilla on Red Hat Linux. The only thing running locally is the browser, and any Java-enabled browser on any platform will do.

**Figure 1. Java VNC client**




Although VNC is the name of the technology as a whole, the protocol it uses is called RFB, which stands for Remote Frame Buffer. Conceptually you can think of a VNC client as a sort of abstracted video card. This "video card" happens to be over a network and is accessible only via the RFB protocol.

Compare this with how X traditionally works: the application may or may not be remote but the X server is running on the machine in front of you, showing you the application's window on your monitor. With VNC there is an addition: the X server machine may be different to the one that your monitor is connected to.

The X protocol is itself reasonably low bandwidth but it is difficult to compare it directly to RFB because they are at different levels: where X speaks in terms of lines and text and so on, RFB is only concerned with pixels, and mouse and keyboard events. There is no provision in RFB for storage at the viewer end other than the frame buffer itself, since this makes the viewer a simpler application.

On UNIX, the RealVNC server is essentially a modified XFree86 server, named Xvnc. It differs from VNC on Windows in the way it handles sessions. UNIX can have several X servers running at the same time, each with a different login, and VNC works just as any other X server would (since that is what it is). But Windows only has one session to use and so on that platform the VNC server attaches to that. This means that when you start a VNC server on a Windows machine you are sharing that existing session but on a UNIX machine a new session is started. There are however ways to share an existing session under UNIX, just like with Windows. Both types of sharing are useful in different circumstances.

When a VNC viewer first makes a connection to the server, they negotiate common ground for their capabilities. This allows implementations to be extensible to some degree. It is also at this initial stage that any authentication is performed. It is important to note here that this is only a simple authentication scheme to prevent casual misuse and no encryption is used at all in the mainstream version, although some implementations do have ways to do this.

From then on it is, perhaps surprisingly, the viewer and not the server that initiates messages. In effect the viewer repeatedly queries the server for any screen changes as well as transmitting input events such as typing. This leads to the bandwidth usage being adaptively self-limiting; the slower the network (or viewer), the less frequently updates can be requested. With a fast viewer and network, updates for tiny changes will be requested, but if the network is slow then several changes will be combined into one screen update. Commonly the viewer will only ask for updates every few seconds, or when the user moves the mouse or types on the keyboard. The illusion of constant updates can be maintained by asking for more updates when the screen actually does change; the idea is that usually several changes will happen very shortly after one another.

The server and viewer negotiate an “encoding” to use which they both understand and the specification lists some common ones. They include “hextile” (which breaks screen areas into tiles), “copyrect” (copying another area of the screen), “ZRLE” (a compressed variant) and “raw” for when the server is the same machine as the client. It turns out to be faster to transfer unencoded screen updates when it is all on the same machine.

The RFB protocol can be used in other ways than originally intended as well. One such way is scriptable remote control, which could be used for automating tasks with applications that were not designed with scripting in mind. You can run them in a VNC session and control the session using a program such as rfbplaymacro [http://cyberelk.net/tim/rfbplaymacro]. This takes a script and translates it into RFB input events such as mouse movements, button presses and keyboard typing.

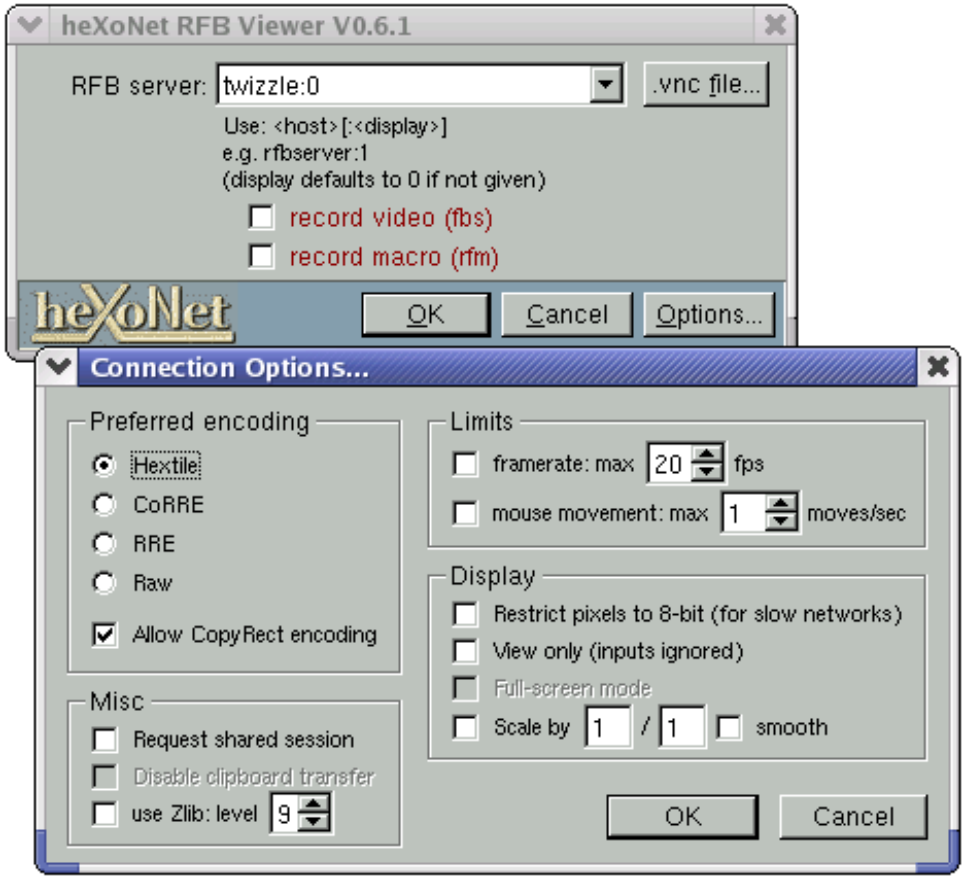
Another use of the RFB protocol is session playback, in other words recording a series of screen updates for later display. You might want to do this if you are setting up a “rolling demo” for a shop display. A simple way to achieve this is just to record the RFB screen updates as they happen into a file with a purpose written application. There are several which perform this function including rfbproxy, which was written for a demonstration machine on a trade show stand, and Jens Wagner’s xrfbplayer (in the rfb package; see later).

The disadvantage of this approach is that only applications which understand the RFB protocol will be able to interpret the saved screen updates. However there is one utility named vnc2swf [http://www.unixuser.org/~euske/vnc2swf/] which converts this format into ShockWave Flash.

VNC is currently going in several directions. The fact that historically the Olivetti/AT&T releases were so slow has led to several independent VNC development efforts springing up. It will be interesting to see how these progress, now that the original VNC developers are working at RealVNC, a company entirely devoted to that project. I personally hope that some of the development forks will be merged back into the main RealVNC project. This is mainly possible because of the GNU GPL license, of course.

One development effort was started by Jens Wagner with his rfb project [http://download.hexonet.com/software/rfb/]. This project is also licensed under the GNU GPL, although it is mostly a re-implementation rather than a modification. There are two main programs: x0rfbserver and xrfbviewer. The first is an implementation of the VNC server that can make VNC on UNIX act as it does on Windows. The way it works is that it shares your existing X session by watching the screen for updates. Unfortunately it can be quite slow to do this since it has to continually poll the screen (the same is true for the original Windows VNC server). The other program is a re-implementation of the VNC viewer application. It looks quite a lot prettier than the original—see Figure 2. It can also record sessions and play them back in the same way and using the same format as rfbproxy, and can create scripts that rfbplaymacro can use as well.

**Figure 2. xrfbviewer**



Another notable offshoot of VNC is the Tight VNC project [http://www.tightvnc.com/] started by Const Kaplinsky. This is a modification of the original VNC software, and so is also available in source form under the GNU GPL. Its original purpose was to enhance the original program by implementing a new encoding named “tight”. This new encoding was designed by analysing the bandwidth problems of the existing encodings, and for real sessions can provide much better bandwidth usage. Const was able to directly compare encoding efficiency with the standard encodings by using a saved session provided by the rfbproxy program and so could fine-tune its performance. Now, Tight VNC includes several other enhancements as well as bug fixes. Among them is JPEG compression (for lossy compression, if you do not mind slight loss of detail but want to use less bandwidth), local cursor handling (to prevent data transfer when you are just moving the mouse), and automatic SSH tunnelling for security. Currently Red Hat Linux ships with Tight VNC in preference to the original VNC.

The enhancements from the Tight VNC project are still largely not incorporated into the main VNC project, and this gave an opening for a company named Tridia VNC. By incorporating features from Tight VNC and the rfb project, adding a Windows installer and offering commercial support, they provided a central place for features and enhancements to go, when it looked like the Olivetti/AT&T VNC development had stagnated.

There has even been a VNC server implementation written for 8-bit microcontrollers, such as toasters and microwave ovens. The author of uVNC [http://www.sics.se/~adam/uvnc/] envisions devices like light dimmer switches and thermometers embedding a VNC server, and being able to control them over a network! For two weeks Adam Dunkels (its author) had hooked up a Commodore 64 to the Internet running uVNC on its 6510 processor. As far as I have been able to tell, there are no VNC-enabled toasters available as yet, but I’ll keep my fingers crossed.

One of the problems with the VNC servers we have seen so far is that it is difficult to embed them into your own application if you need to. Also, if a bug is found in one, it most likely needs fixing in several others since they share the same heritage. One obvious way to make this better is to separate out the protocol engine and share it between all of the implementations; in other words make it into a library. One such library is libvncserver [http://libvncserver.sourceforge.net/]. To see why this is so useful, take the simple rfbproxy program I wrote as an example. It is so simple that it only works if the viewer is using the same resolution and colour depth as when the original recording was done. This is because it literally just remembers the bytes that were transferred originally, rather than reconstructing the screen as it was and acting as a VNC server for that. Had libvncserver existed when I wrote rfbproxy I would certainly have used it and rfbproxy would have been a better program for it. Perhaps one day I will re-write it that way.

One application using the libvncserver library is a program that acts as a gateway between the Windows Terminal Server protocol (RDP) and the protocol used by VNC. This program, rdp2vnc, was in fact the reason libvncserver was originally written. With it, you can use Windows Terminal Server just like a VNC server.

Another application that uses libvncserver is the KDE Desktop Sharing service [http://www.tjansen.de/krfb/], which is due to appear in KDE 3.1. This is a small application which allows you to share your existing KDE desktop. A similar feature is planned for the GNOME desktop environment as well. This style of desktop sharing gives very good integration into the rest of the desktop and will make it much easier for new users. However, as I understand it, it works in the same way as x0rfbserver and needs to poll the screen for updates. What is really needed is for the X server to be able to notify applications of changes to the screen, and in fact an X extension for tracking screen changes has been proposed recently. Another approach is to have an X module for VNC sharing, so the X server itself becomes the VNC server, eliminating the need for slow polling.

This problem is solved by Alan Hourihane’s xf4vnc [http://xf4vnc.sourceforge.net/]. This is exactly what is needed: a VNC server implementation as an XFree86 module, in particular for XFree86 4.2.0. This also solves another VNC problem, namely that XFree86 has made several advances since the version VNC was based on (3.3.x), and so VNC has been lagging behind ever since. In fact 3.3.x is no longer maintained at all which leaves VNC users in a bad situation. As well as providing an X driver as a drop-in replacement for the old Xvnc server, xf4vnc provides a module which can be used for VNC-enabling an existing session, in the same way that the KDE Desktop Sharing service works. In fact it will be better than the Desktop Sharing service since it will not need to poll the screen and as a result will be faster. The xf4vnc project is licensed under the GNU GPL, although that is incompatible with the license of XFree86 and so it will never be merged (but it can of course be used by everyone). It is quite a recent development and so is not being shipped by any free software vendors yet to my knowledge. It is based on the original VNC and therefore unfortunately does not have many of the enhancements of some of the other VNC projects.

VNC can be used as a teaching tool, for either teleteaching where the students are physically remote from the teacher, or class teaching in the same room. In this mode generally the teacher's screen is viewable by the students but they may not control it. When there are just a few students, all is well and good, but VNC does not scale to large numbers of viewers. One approach to solving the problem is to use multicast where the same IP packet is sent to multiple machines, rather than the unicast method (one IP packet per machine) that appears in standard VNC. This modification has been made in the MulticastVNC program [http://www.informatik.uni-trier.de/~ziewer/MulticastVNC/].

With some teaching environments, such as computer troubleshooting, the teacher needs to be able to see and control the text console as well as the graphical environment. Imagine for example the class entitled "setting up an X server" or "recovering from boot-up problems". There is hardware available to solve this problem, but it is quite expensive. An adequate but not as comprehensive solution can be made from software. One attempt to do this is the vtgrab project [http://cyberelk.net/tim/vtgrab]. This aims to be a text mode equivalent of VNC for the actual text console of a Linux machine. It can replicate the server machine's console on the teacher's machine in a window and is designed to be usable over a serial link (so that it does not rely on a working network connection). The teacher can switch between consoles and can see console switches made by the student; the teacher can also type at the remote console. If the student's machine is switched to a console running X, then it automatically starts x0rfbserver on the student's machine and a VNC viewer on the teacher's machine. As far as that goes it works well enough but there is plenty more to do and it will never fully replace a hardware solution. It can not start until the operating system is booted, whereas a hardware solution would allow remote BIOS setting changes for example.

There are several areas in which the current VNC efforts are lacking. One often requested feature is the ability to print things remotely and transfer files easily as part of VNC. Solutions to these problems, so the argument goes, exist already independently of VNC—network printer sharing, in the printing case, and FTP or scp in the file transfer case—but it is asked for often enough that there obviously needs to be some better integrated way of doing it. Development efforts in this area are sporadic but frequent although there is not much progress to report yet. It seems clear though that there is no need to invent another way of transferring print jobs between machines for example; all that is needed is integration to an existing method. The eSVNC project [http://perso.wanadoo.fr/samfd/esvnc/] seems to be taking steps in this direction.

One interesting possible future direction would be the ability to transfer sound as well as images. In a shared session it would be advantageous to be able to talk to the person whose session you are sharing. Several low bandwidth methods are possible and with the (admittedly gradual) rise in popularity of Ogg Vorbis perhaps someone will find enough incentive to attack this problem. Even an out-of-band way of writing text to each other would be a good first step in this direction.

Perhaps the most often requested feature, and surely the most needed, is encryption. In the reference implementation from RealVNC, a challenge-response authentication scheme is possible. However even this is unfortunately vulnerable to a man-in-the-middle attack. This is in spite of the fact that it is possible to do authentication securely as well as encrypting all communications to boot (for example SSH). VNC was designed to get its job done on a trusted network and so security was never really a goal. In this day and age though, it is more and more common for people to want to share desktops across the Internet using their broadband connections making security much more of an issue.

Luckily it is possible to "tunnel" a normal VNC session over a properly authenticated connection such as SSH or Zebedee [http://www.winton.org.uk/zebedee/]. In the short term this is adequate, and not all that difficult to set up, but it is a huge shame that you need this extra step.

There are many more VNC-related projects underway than those I have mentioned here, and features which I have not pointed out. If there is something particular that you are after it may already exist; the links in this article are probably a good place to start looking.