A Study of Terminal Server Technology

Mattias Eliasson
I would like to thank my supervisor Viktor Leijon at Luleå University of Technology.
Abstract

Terminal Server technology is a field that promises to solve a lot of problems by moving the desktop into the cloud. As most users are still using PCs we want to examine what it is that holds thin clients back. To answer that we study what the current technology provides, and attempt to point out what it lacks.

To provide background information I have evaluated several terminal servers, which use different protocols. The study shows that this technology is still immature. Multimedia, which is essential to modern desktop use, needs a lot more work. So does 3D graphics which may not be essential but still important to many users.

These problems may require solutions that breaks compatibility with legacy applications, by requiring multimedia and 3D applications to use a network-transparent API, rather than accessing hardware directly like they currently do.

I present a possible solution to the problems with multimedia, by using an API where codecs can be moved to the client. When it comes to 3D the solution may become more complex. The current API:s are to close too hardware and a more abstract scene graph may be needed to provide a sufficient level of abstraction.

As a result of these limitations terminal server technology is only fit for office usage, and not for everyday usage. However, video presentations that may be needed in some offices are becoming more common on the Internet.

I conclude that while the terminal server approach is promising, the existing technology is not mature enough for everyday usage. Most users require multimedia and optionally 3D graphics for a complete desktop experience. Those are the areas that need more attention in order to satisfy the end users.
Contents

1 Introduction.................................................................................................................................4
  1.1 Terminal Servers.................................................................................................................4
  1.2 Purpose and scope...............................................................................................................6
2 Background.................................................................................................................................7
  2.1 Terminal Server Protocols...............................................................................................7
  2.2 Terminal Server Products...............................................................................................9
3 Server test................................................................................................................................11
  3.1 Installation of servers........................................................................................................11
  3.2 Adding a user to the server..............................................................................................11
  3.3 Using the client................................................................................................................12
  3.4 Testing applications..........................................................................................................16
4 NX Server..................................................................................................................................18
  4.1 DXPC (legacy)..................................................................................................................18
  4.2 Nxproxy (NX)....................................................................................................................18
  4.3 Nxagent (NX)....................................................................................................................18
  4.4 Nxssh (NX).......................................................................................................................18
  4.5 Nxesd (NX).......................................................................................................................19
  4.6 Nxserver (FreeNX)............................................................................................................19
  4.7 Nxnode (FreeNX).............................................................................................................19
  4.8 Session management..........................................................................................................19
  4.9 Nx infrastructure...............................................................................................................20
5 Analysis......................................................................................................................................21
  5.1 Seamless desktop integration..........................................................................................21
  5.2 Multimedia.......................................................................................................................22
6 Conclusion..................................................................................................................................27
7 Appendix: Installation of terminal servers.............................................................................28
  7.1 Installation of FreeNX-server.............................................................................................28
  7.2 Installation of LX-Server....................................................................................................28
  7.3 Installation of xrdp.............................................................................................................29
8 References...................................................................................................................................30
1 Introduction

1.1 Terminal Servers

The cost of maintaining and upgrading client computers is high. A modern desktop computer
has an economic lifetime of approximately three years before it is considered too old [1]. In
order to decrease this cost many large organizations have been looking at a revival of the
old system where applications run at a server and the clients where called terminals. An
ordinary personal computer (PC) is now often called thick or fat client while an old-school
terminal is called a thin client.

In a modern PC graphics are handled by a graphics card, that together with a device-specific
driver does most of the graphic-related work. Therefore most of the graphics processing is
done outside of the applications. This could be seen as a client-server setup where the driver
and the graphics card is the server. Now if we move all other local IO to the graphics card and
move it outside the computer we have made a thin client and a terminal server out of it. The
thin client is easier to replace and is reduced to a set of peripheral hardware, like the printer
and the monitor. If it breaks you can just replace it. The PC that is now a terminal server does
not have to deal with graphics and peripherals do not have to be close to it. It can be placed in
the garage or even in another city. It can be accessed by any compatible terminal making it
mobile, without requiring the user to bring any hardware. Thus there are usages for terminal
servers even if they only have one user.

A thin client is totally dependent on a server where the entire user desktop and all applications
are running. Traditional clients that do a lot of client-side operations is often called fat clients,
suggesting that they do more than they have to.

The thin client concept is also applied to client-server software where a minimal amount if
work is done on the client side. An example of a fat client is a mail client. It connects to a mail
server and processes mail locally. A thin client in this context would be a web mail client. The
main code runs on a web server while the web browser only shows the output from the web
server. Sometimes a web application uses JavaScript and other client side code, making it
more of a hybrid between fat and thin client.

The real challenge however is to run full desktop applications server-side, even if they are
written to run locally. To do this program output is redirected to the client, while keyboard,
mouse and other peripherals are redirected to the server, in a hardware-independent way.
There are hardware-only thin clients like the Sun Ray and software that converts a PC to a
thin client. Thus an old PC that is useless as a thick client can be reused as a thin client. This
significantly increases the lifetime of a desktop computer, saving money and environmental
impact.

When producing special-purpose hardware for thin clients a lot of resources could be saved
on processing power, memory and local storage. Most thin clients do not have a hard drive but
boot directly from a read only memory (ROM) on the motherboard.
Old time terminals were usually built into the monitors [2], something that is not very common in modern thin clients.

This modularization may even have benefits in a single user scenario. Having components that can be replaced individually and attached/detached in a flexible way provides more usability. Just as an external printer is more flexible than a built in printer, so is this technology.

### 1.1.1 Remote usage

One of the most interesting capabilities from a usability perspective is that remote desktops become completely mobile. As clients become homogeneous, all you need is a client where you can log in and resume your work. If thin clients are placed so that they are available wherever you go, you don't need a laptop or even a USB stick.

### 1.1.2 Management

Thin clients provide a minimalistic hardware and software platform. They therefore require almost no configuration compared to fat clients. Thin clients allows for a homogeneous client environment. You do not have to bother with the individual needs of a department or a specific user. All such problem becomes server side problems.

Thin clients also have a longer theoretical life cycle than fat clients. As processing are done server-side, clients do not have to be upgraded with new hardware.

### 1.1.3 Resource usage

In a multiuser scenario it brings another advantage. Most users do not utilize 100% of their PC resources all the time. These extra resources are wasted when using PC:s. In a terminal server resources are shared, and underutilization means that less hardware is needed than in a fat client solution.

On the other side there are also advantages when users require more resources than provided by a fat client. Applications can run on more than one server by using seamless windowing so Terminal Server (TS)-technology can scale better for demanding users.

Also old hardware does not have to be abandoned. Old and new servers can be used simultaneously, and applications with low system requirements can be run on the old servers.

### 1.1.4 Limitations

Communication between an application and what here is called the terminal can be categorized into three categories. Local on the same computer, Local Area Network (LAN) and Wide Area Network (WAN). Each category introduces further distance between the application and therefor also implies less bandwidth and higher latency. Applications that are not written with LAN usage in mind do not consider latency and expect high bandwidth. They
can therefore become slow on WAN usage even if they are not bandwidth intense, because they expect immediate response on what they do. To combat latency problems asynchronous protocols are required. Some programs work well over WAN connections even if they are only designed for Local usage, while others are very slow. This naturally limits market penetration of terminal-server technology.

Another limiting factor is the lack of a unifying standard. Using different clients for different servers is never a successful solution. There are currently several different protocols being used. Some of them have open source software (OSS) implementations, and those tend to be the more popular, as the others are limited to a specific product.

While open standards and OSS is often overlapping, they are not the same thing. OSS protocols often become open standards, and open standards are often implemented in OSS [3]. Therefore the lack of unity in OSS solutions has a negative impact on standardization.

1.2 Purpose and scope

What is holding Terminal Servers back? Terminal Server technology has great potential, so why isn't everyone using it?

To answer that we need to know what TS-technology can't provide, but a fat client can. And this in turn requires that we know the current status of TS-technology. What standards are emerging and what can they do? This will give an insight in what needs to be added in order to push this technology forward.

- The scope of this thesis is to test the capabilities of some servers.
- There will also be a deeper study of the FreeNX server, the components used in it will be identified.
- The results will then be analyzed and potential solutions to encountered problems will be presented.
2 Background

2.1 Terminal Server Protocols.

2.1.1 Text based

In the beginning there was as many protocols as there was manufacturers of Terminal Servers. With the introduction of Internet a single dominating standard emerged – Telnet. Later a more secure standard largely replaced Telnet – SSH.

2.1.1.1 Telnet

Telnet [4][5] is a simple and fast protocol. It does not standardize character encoding and graphics, however American Standard Code for Information Interchange (ASCII) and various extended ASCII standards was most commonly used. Lately a unifying standard called Unicode Transformation Format (UTF)-8 emerged [6]. Its not yet universal but it is gradually replacing the old character sets. However telnet has a fatal flaw, it is completely unencrypted and therefore insecure.

2.1.1.2 SSH

In response to the security concerns with Telnet a new protocol was needed. The response to this challenge was the Secure SHell (SSH) [7] protocol.

The initial version of ssh was released as freeware in 1995 but became a commercial product. The original freeware version was picked up by the open source community and OpenSSH [8] became the de facto standard for secure remote textbased computing. Telnet still exists and is sometimes used on local networks, however most people use SSH out of habit even when telnet is sufficient. SSH became an Internet Engineering Task Force (IETF) standard in 2006 [9].

2.1.1.3 GNU Screen

GNU Screen [10] is an popular session manager for text-based applications. It allows the user to have several virtual screens and switch between them. Screen can be detached and reattached, it can even be attached without detaching. This gives both attaching terminals access to the virtual screens. If they watch the same screen they'll have dual command, but they can also view different screens.

2.1.2 Graphical

2.1.2.1 X-Windows

The X Windows system [11] is the most commonly used standard for network-transparent
Graphical User Interfaces (GUI:s), it has been around since 1985 and there are many implementations. Most Unix systems support graphics through an X display server. While it is designed for network transparency several of its newer extensions such as video [12] and OpenGL [13] do not work or work poorly when not used locally. It also lacks sound support. The network protocol is very insecure so tunneling it through SSH is the default way to use it today [14]. In most systems using it without SSH requires reconfiguration, as it is disabled by default, and is strongly discouraged.

The main disadvantage with X-Windows is that it is a synchronous protocol. Therefore network lag slows everything down. X was designed for LAN usage and using it over a WAN connection can be a problem because of the extensive and lag-sensitive synchronization or “round-trips” performed. It also has no compression so it requires a lot of bandwidth, another side effect of being designed for LAN. As much of today’s development of X is focused on local desktop usage, network-transparent usage of X has become even slower.

An important aspect of X11 is that it lacks session management. If a connection dies, so does all windows managed by it.

2.1.2.2 VNC

The Remote Frame-Buffer Protocol (RFB) used by Virtual Network Computing (VNC) is a remote frame-buffer protocol, designed to copy the raw frame-buffer from the video card on the server and transmit it to the client. It also passes mouse and keyboard info from the client to the server. It is not designed for multiuser network-transparent terminal servers, but for remote administration. As such it does not have any need for audio and does not support it. While VNC is the default protocol for remote desktops in OS X it does not support multiple remote desktops. [15][16]

2.1.2.3 RDP

Remote Desktop Protocol (RDP) [17] is originally a proprietary protocol designed for Microsoft Terminal Server. It was reverse engineered by the rdesktop project that created the rdesktop RDP client for Unix[18]. There is both a commercial and an OSS server that was coded using the information provided in the rdesktop project. RDP, like VNC, is a remote frame-buffer protocol but has several functions that VNC lacks. It has support for sound, printers and file sharing in the protocol.

One of the main problems with RDP is that Microsoft often creates new versions of the protocol. This creates problems with old clients. It seems that Microsoft expects users to run Windows and just upgrade the client software. However, there are hardware clients out there and they are not easy to upgrade.

2.1.2.4 NX

NX [19][20] is a proprietary product, like RDP, that has been reverse engineered and there is now an open server called FreeNX [21]. Unlike RDP it is not a frame-buffer protocol. For networking and security it relies on network tunnels. Graphics are handled by a protocol
designed to compress X-windows traffic. This is based on the OSS project DXPC [22] which has been developed further into the program “nxproxy”. On the server NX starts nxproxy as a virtual X11 display server. On the client nxproxy connect to a X11 display server. The actual communication is not the usual X11 protocol but a new one that is less bandwidth-intense and more resistant to network lag. As it is unique in not being a frame-buffer protocol but a protocol that intercepts commands at a higher level, it is one of the most efficient protocols around. There is currently no good free client for NX. While RDP is a single protocol for everything, NX combines several different protocols over SSH [8] tunnels. It uses nxproxy/X11 for graphics. The now antique Enlightenment Sound Deamon (ESD) [23] for sound, Common Unix Printing System (CUPS) [24] for printing and Samba [25] for file sharing. As all of those is derived from OSS projects, their source is available under the GNU General Public License (GPL) license. The FreeNX server is a substitute for the proprietary nxserver and nxnode programs that just gets everything set up. See chapter 4 for more details.

2.2 Terminal Server Products

2.2.1 Sun Ray.

The Sun Ray system uses what is probably the thinnest client on the market. The protocol is the proprietary protocol Appliance Link Protocol (ALP). It is a remote frame-buffer protocol like RDP and VNC. SUN has also introduced RDP support on their new VDI servers [26] which makes RDP and VNC the remaining combatants for being a De Facto standard in the RFB field. As VNC is mainly used for remote control of a physical display, so RDP can be considered the De Facto standard for RFB-based terminal servers.

2.2.2 Citrix XenApp

Citrix uses the proprietary Independent Computing Architecture (ICA) protocol, which sends high-level meta-data from Application Programming Interface (API) calls. It does so by having an alternative implementation of the Windows graphical API. As a result it has a history of incompatibility with many applications, as the Windows API is difficult to implement.

2.2.3 Windows Terminal Server

Microsoft licensed Citrix technology but not their proprietary protocol ICA. Instead they created their own protocol RDP. By shipping their RDP client with their Windows client operating system it has become the most distributed TS client. The authors of the X11 client rdesktop has successfully reverse engineered the RDP protocol and documented it. There now are several servers based on RDP that can be accessed from windows computers without installing any new software.

2.2.4 NX Server

NX Server uses a protocol based on X11 and is therefore compatible with X11 applications. By intercepting data on a higher level than the frame-buffer it is a highly compressed and
efficient protocol. While it has that in common with ICA the X11 protocol significantly easier to implement with a high level of compatibility. Unlike the Windows API it has always been forced to deal with compatibility issues, as there has been more than one X11 display servers around, and applications has to work with all of them.

### 2.2.5 Xrdp

Xrdp is an X11 RDP server based on the rdesktop client. It is still under development and not ready for production usage.

### 2.2.6 LX server

LX Server is a commercial RDP server for X11 by Thinstuff. It is in many ways similar to Xrdp, but is ready for production usage and has more tools.
3 Server Test

The server test was conducted on my home server running CentOS 5, a Linux distribution based on RedHat Enterprise Linux (RHEL). My client is running Windows XP Pro. The machines have Pentium 4-based Celeron processors, 1,2 GB ram on the server and 2 GB on the client. Therefore my choice of servers is limited to those that work on CentOS 5 and have a Windows client. I have also limited my selection to non-proprietary protocols. Proprietary product-specific protocols do not become standards, unless they are reverse engineered and therefore no longer proprietary nor product-specific – like RDP. Some kind of open specification is a requirement for standardization.

3.1 Installation of servers

I am testing three servers. Two of them are RDP servers, the other uses the NX protocol. LX server is a commercial RDP server, I'm using the free version that logs users out after 30 minutes usage. Xrdp is an RDP backend to Xorg and is open source. FreeNX is a free server that uses components from the commercial NX server to implements its server-side protocol. As NX is based on GPL components, all but the handshaking protocol and some utilities are available with source code. For detailed information on how the server was installed, see the appendix.

3.2 Adding a user to the server.

3.2.1 NX Server

NX uses local authentication. There is also an internal database that can be turned on manually, that was previously mandatory.

3.2.2 LX Server

While users in LX are local system users, access to LX has to been enabled in an internal database. This is done via the built in web server, there are no command line tools missing which is an obstacle when creating scripts. The web interface alsos allow groups to join, so adding an "rdp" group and granting it access is a workaround that simplifies administration.

3.2.3 xrdp

Xrdp uses local authentication.
3.3 Using the client

3.3.1 NX Client

The NX Client is free to download from nomachine, the company that makes the commercial server. Going to http://www.nomachine.com/download.php and choosing the Windows version gets the current version. Download and install it like any other application.

Now we have to configure a "session". Click Start -> Program -> NX Client for Windows -> NX Connection Wizard.

Click next to the first screen and get to this screen where we can enter the first settings. Fill in the session name which can be anything. Then fill in the hostname or IP of the terminal server.
The next screen will allow us to specify desktop environment. The Windows and VNC option can forward the connection to an RDP or VNC server. Leave it unchanged to use the default Unix option. We choose KDE. We also change the size of the desktop from "Available area" to "Full screen".
On the next screen we chose to create a desktop shortcut, that will have the session name. We also chose to show the Advanced Configuration Options dialog. FreeNX does not ship with the same encryption key as the client. So we need to click on the "Key..." button to change it.
Now we have to copy and paste the context of /etc/nxserver/client.id_dsa.key to the box below.

Now we start the session from the desktop icon we created.

3.3.2 RDP Clients

The RDP protocol has two main clients. The Microsoft Terminal Server Client “mstsc.exe” for Windows and the open source rdesktop for X11. Mstsc is included in Windows, to start it open the run dialog box and type “mstsc”.
This is what it looks like in the Swedish version of XP. Type in the address to your server and click connect, or use the enter key. Now the LX or xrdp server login screen will show, depending on what server you are running.

3.4 Testing applications

3.4.1 Office and browser

Office applications and browsers are essential for everyday use of desktop computers. CentOS has OpenOffice and Firefox built in. I also tested using the console and some of the simple built in games in CentOS/KDE. On LX server i had to manually set the console to use UTF-8, but other than that i had no problems.

3.4.2 Printing

Only NX and LX claim to support printing locally on the client. I tested this by pressing Ctrl-P in FireFox. In LX server there was no printer available. In FreeNX there was a printer available but using it gave me a CUPS error message. Printing trough the server naturally works, but that is not an option when the server is far away.

3.4.3 Multimedia

This test was performed by installing Adobe Flash Player and surfing you youtube. Flash player is not a part of the CentOS software repository, but can be downloaded from Adobes software repository. A rpm for this can be downloaded from Adobe and installed. The package to provide flash is “flash-plugin”. The instructions below will do this. When Flash is installed this way, it will be updated with system updates.

```
wget http://linuxdownload.adobe.com/linux/i386/adobe-release-i386-1.0-1.noarch.rpm
rpm -i adobe-release-i386-1.0-1.noarch.rpm
yum install flash-plugin
```

On NX Server audio works trough the deprecated esd subsystem. It only works on a fresh
session, when resuming an existing session esd fails. NX Server can handle video, but does so by streaming it as jpeg images, this is very slow. Also audio and video is out of sync. Even if it can randomly be in sync, it would lose sync quickly as video playback is not realtime on my hardware.

RDP has built in sound support and it was enabled in the client. Neither LX nor xrdp could handle sound, even if the LX homepage claims sound support. Video works on both of them, but is slightly slower than NX.

### 3.4.4 Games.

None of the servers have support for 3D graphics. Just for fun I tried running a simple OpenGL game anyway. I expected it to not work at all so I was a bit surprised that it did. It used software rendering and was much slower than video playback. The speed was seconds per frame rather than frames per second.
NX Server

NX server consists of a set of different programs implementing different protocols. Most of them are OSS projects that are shipped with most free operating system, and some of them ship with Mac OS X.

All NX components have the nx- prefix, like nxssh and nxesd. FreeNX consists of two parts. NX which is the OSS components of NX that are used both by the client and the server, and FreeNX which is a set of scripts implementing the server components not included in the “nx” package. Unfortunately, most of this is undocumented. Therefore this section attempts to describe what these components do. Making a complete documentation of the entire suite is outside of the scope of this project, but a partial documentation is a good start.

4.1 DXPC (legacy)

The Differential X Protocol Compression (DXPC) [27] is a now dormant project that NX is derived from [28]. The “dxpc” executable is a proxy that acts as an X11 server on one side and as an X11 client on the other side. It is stateful and does not send redundant data like X does. While it achieves a great level of compression, it is no longer under active development as the original developers no longer are active.

4.2 Nxproxy (NX)

Nxproxy has the same role as dxpc. In the windows client nxproxy is not running as a stand-alone process, but integrated into the X Server that ships with NX Client.

4.3 Nxagent (NX)

The “nxagent” is based on the Xnest X11 server that is intended to run a virtual desktop inside another desktop. This is used to create a complete desktop that can be tunneled to the client. Nxagent has nxproxy integrated into it, so when running a remote desktop there is no nxproxy process running.

4.4 Nxssh (NX)

Nx uses SSH for the actual network connection and encryption. Other protocols are tunneled trough SSH tunnels. In NX 3.X that i have been using nxssh is a fork of OpenSSH with NX-specific code. The NX homepage states that NX 4.0 will change back to OpenSSH and have all NX specific code removed [29]. That makes information about the current patches less interesting. Nxssh is also used only at the client side, on the server side the preexisting ssh server is used.
4.5 **Nxesd (NX)**

ESD is an old and orphaned network-transparent sound system for Linux that is currently being phased out of the mainstream distributions. NX provides its own ESD server in case there is none locally, but requires the desktop environment on the server to have an ESD driver.

4.6 **Nxserver (FreeNX)**

This is a large script with a lot of functionality. When using the internal user database this is the program you run to modify it. The most important role of nxserver is its ability to act as a shell.

The following is a line from `/etc/passwd` that tells us that there is a user `nx` with `/usr/bin/nxserver` as shell. Nxserver in turn uses the smaller script `nxnode` that starts `nxagent` for desktop sessions or `nxproxy` for seamless sessions.

```
nx:x:107:107::/var/lib/nxserver/home:/usr/bin/nxserver
```

4.7 **Nxnode (FreeNX)**

Nxnode is another large script that handles starting, resuming and terminating sessions. While nxserver is run as the `nx` user the `nxnode` script is run as the actual user. This script also configures and starts cups and samba.

4.8 **Session management**

The NX protocol has built in session management. Sessions can be detached and resumed without any major problems. Sometimes this leave artifacts in some windows, which can be fixed by forcing them to repaint.
4.9 **Nx infrastructure**

This illustration describes how a typical NX session works. On the left is the closed source Windows client, and on the right is a FreeNX-server.

*Figure 1: NX Windows client and Linux server.*
5 Analysis

5.1 Seamless desktop integration.

Some terminal servers have the ability to export individual windows rather than whole desktops. This is a good temporary solution to overcome the shortcomings of TS technology. The user can just log in to the server and start Firefox. Firefox then opens up just as if was run locally on the client. Windows theming will look different if you for example have the gnome version of Firefox on the server and view it on a Windows client, it will look like the gnome theme on the server.

The advantage with this is that the user does not have to migrate the entire desktop to the server. Multimedia shortcomings can therefore be compensated by running a local media player.

There are draw-backs with this. The obvious disadvantage is that you don't have the entire desktop online. Mobility and really thin clients is not a part of this concept. Usually you can move a session to another desktop by closing the client and logging in at the other desktop, but then only the applications running on the server as applications running at the clients desktop is not mobile.

Another problem is desktop integration. Launching applications on the server is usually not an easy task with this solution. No client I have tried can integrate start menus, quick start buttons, desktop icons and other things a user is used to. Resources such as printers and files are not homogeneous. It can be confusing for the user that the remote and local applications have different files and printers. You also can't set the remote Firefox as your default browser or the remote OpenOffice to open your documents.

To improve desktop integration I suggest that the use of a local launcher and an extension of the protocols so that the launcher can send a command to the server and get an application running.

In order to open a document remotely some kind of filesystem integration or sharing must occur. This is not as simple as just launching the application, as it can only access files on the computer where it is actually running. One solution could be to make the launcher upload the file to the server, and download it when its saved on the server. This is however not as simple as it sounds, it has to monitor the application and synchronize the file. You may also want a filesize limit on such a feature. It would be nice to open an Office document this way, but opening a DVD iso file would probably be to slow.

In a future scenario, client side desktop integration is not an issue. We simply do not want a local desktop at all, we want everything on the server. On the server it may however be useful, while we want to run the desktop on a server, we may also want to run some applications on other servers and make them appear on that desktop.

Files, printers and other resources do not have to be shared by the TS software. Configuring
resources that are shared among servers is something that Unix/Linux administrators are already used to. The difference between such a server and a client is usually about hardware, so setting up several TS servers that mount user directories from a file server is no different than setting up several clients to do this.

5.2 Multimedia.

5.2.1 Sound.

The solution for grabbing and streaming sound used today is to emulate a sound device and let the applications play sound to them in Pulse-code modulation (PCM) format. This is sometimes compressed before transmission.

This means that all sound is transcoded before transmission. Transcoding uses a lot of cpu, memory and sometimes bandwidth, if the transport stream has less compression than the original file.

In Figure 2 the application has its own codecs and outputs decompressed data to the sound subsystem, in NX this is ESD. When the sound is sent over the network this may include compressing it again, and therefore in fact be transcoded before transmission.

If this traffic is compressed outside of the application, lossy compression of decoded material usually results in bad quality. Transcoding from a lossy format to another lossy format is generally a bad idea.

A better solution would be to provide client-side codecs and just stream files to them. When playing a format not provided by the client transcoding would be required, but user habits and software can adapt to it. If the client supports ogg but not mp3 the player running on the server could offer the user to convert files before they are played. Or if they are transcoded they can be saved to disk while playing.

A network-transparent codec chain will not only minimize transcoding, cpu and bandwidth usage, it will also minimize quality loss as a result of unnecessary transcoding. When a file is transcoded from one compressed format to another the loss of quality could be substantial.

Another advantage is that a client side codec can use hardware for decoding, something that is available in most desktop computers today.
In Figure 3 we show network-transparent decoding using GStreamer [30] which is a popular library for decoding data outside of the application. If GStreamer could be made network-transparent, it is possible that applications that use it may be runnable with little or no modification.

If the API needs to change for this to happen, a change in the upstream library that is not exclusive to network-transparency would be the best solution. That would make developers adapt their program even if they do not have network-transparent usage in mind.

5.2.2 Video

Like sound video is currently handled by grabbing frames and sending them frame by frame. While this actually works for sound, the result on video is not that great. Decoding mpeg to frames and sending them in a jpeg stream is generally a bad idea. If you have a lot of CPU and bandwidth, it may be possible to do this without to much problems, but the amount of data used in video is a lot more than the amount of data used in audio, and that makes this solution use a lot of resources.

As with sound it would be possible to move decoding to the client, freeing a lot of resources on the server.

A client side codec could use hardware acceleration to play High-definition (HD) video and therefore be much more efficient than a server side codec.

Another problem is using sound and video together. If you are watching a movie you generally want sound and video synchronized with each other. The current solution does not do this. When designing a network-transparent codec chain it is important to cover this area, as audio and video are currently independent from each other using two different streams.

5.2.3 3D and Games

5.2.3.1 GLX

Most modern games use OpenGL, commonly abbreviated as GL, or DirectX as the graphics backend. These libraries provide a very thin layer on top of the hardware and the application has a render loop that renders frame by frame. The OpenGL extension for X, or GLX, is
network-transparent. When using GLX over a remote connection the commands are sent over the network. While the actual rendering is done on the display server, equivalent with a TS client, it is still done by a render loop in the application. Most OpenGL programs were not designed to be network-transparent giving them bad performance on remote GLX, if they work at all. Many OpenGL extensions require hardware graphics acceleration and does that by talking directly to hardware, a technique called GLX forking. This makes most games break when they are run remotely.

5.2.3.2 VirtualGL

VirtualGL intercepts local OpenGL rendering by modifying calls to libGL so that they render trough the graphics card and then sent back to VirtualGL instead of the display. VirtualGL then compresses each frame as a JPEG and send it over the network. This works far better than GLX but requires graphics hardware on the server.

5.2.3.3 Realtime performance

When playing video and music, network lag can be compensated by buffering. This is not true for games, they require real time response. Even a solution like VirtualGL may be too slow for games. While hardware rendering at the server is cool, it is also expensive. There is still a huge performance loss. Graphics cards that support feedback is also more expensive than the gaming cards. As most desktops have hardware accelerated OpenGL or DirectX capabilities, the best solution would be to use them.

5.2.3.4 Game specific clients

Client/Server multiuser games are very common today. World of Warcraft, Counter-Strike, Second Life and are some examples of 3D games that are widely used. What they all have in common is that the main game logic is performed on the server. When game logic is performed on the client it opens up a window for cheating. If, for example, a players health would be stored in local memory, there would be applications that could make the player immortal, by making sure that the health is always at its maximum value. There are a lot of examples of such cheats for games that are stand-alone or online games that does much logic in the client. Diablo is a classical example as it has a lot of anti-cheating code. This helped for a while but was eventually broken.

Game clients can therefore be seen as program-specific TS clients. This is naturally not a good solution for a TS environment where clients are supposed to be thin and require a minimum amount of management.

5.2.3.5 General purpose gaming

In order to make a protocol that allows general purpose gaming the clients for network games make a good starting point. The general design goal when building such a client is to minimize network traffic. More traffic means more sensitivity for network problems and more load on game servers.
5.2.3.6 3D Scene Graphics

A Scene Graph API like Java3D provides an abstraction compared to Open3D and DirectX. Instead of having a render loop the application creates a scene with an object-oriented interface. The render loop itself exists within the library and can therefore be moved to the client. Designing a client-server scene graph API can reduce bandwidth a lot compared to network transparent OpenGL or DirectX implementations, while taking full advantage of client side hardware acceleration.

A scene graph library should have support for playing audio and video and is perhaps the best place to solve those issues. Most API:s for 2D graphics are being reduced to being a 2D layer on top of 3D libraries like OpenGL and Direct3D, as 2D is a subset of 3D. This simplifies hardware acceleration and reduces complexity in API code as it does not need its own rendering mechanisms.

Figure 4 Shows how an OpenGL application normally renders graphics locally. The render loop run inside the application, and is optimized for local performance.

Figure 5 Shows how it is usually done when network-transparent OpenGL is used. The render loop still runs in the application and is optimized for local performance.

Figure 6 Illustrates a scene graph library. The library handles the rendering loop and the application is abstracted from hardware. This is not optimized for local usage or the specific game. It is designed to simplify application development and make applications portable.
In Figure 7 we see how a protocol for network-transparent scene graph could not only be used to move the data-intense frame loop to the server. It could also be framework- and hardware-transparent. If both Object-Oriented Graphics Rendering Engine (OGRE) and Java3D implement the protocol, each of them could be either frontend or backend. Also, the Unix application can render through the DirectX backend in Java3D, or trough the DirectX backend in OGRE.

While this will probably not be as efficient game specific client, it is significantly more efficient than streaming OpenGL-commands or compressed frames. It may still use quite a bit of bandwidth, but not so much that it is unusable on a LAN.

In Figure 8 a setup where a game-specific client runs on a local terminal server is a solution that may bring some of the advantages of terminal servers to gaming. It provides centralized administration among several users and lowers the administrative demand on the clients.

5.2.3.7 Applets

Another solution may be to allow small downloadable chunks of bytecode that changes that some run in a virtual machine and changes the clients behavior. Like Java Applets or Silverlight run in a browser. While this may solve problems with application-specific code that need to be run close to metal, it also introduces a lot of problems. First of all, something that downloads and runs non-generic code is not strictly a thin client. It introduces unpredictable CPU overhead. This kind of technology introduces the kind of unpredictability that is currently a large problem with web browsers. However, this kind of trick would allow games to perform game-specific tasks on the client, this can significantly improve performance.

5.2.4 Non-transparency

The main problem with some of the suggestions above is that all of them require specialized software that is aware if the software stack. Most media players have built in codecs, so moving decoding to the client requires reducing those players to a GUI around the network-transparent solution. A network-transparent 3D scene graph library could be used as a back end by other scene graph libraries like OGRE and Java3D. However, most applications have their own application-specific optimized scene graph code as a thin layer between the application and OpenGL/DirectX. Those applications may require a lot of work to port to a generic scene graph library.
6 Conclusion

One problem has been finding scientific studies regarding terminal servers, particularly when it comes to cost and usability. Therefore it would be useful with studies that look into these areas. While the arguments for using terminal servers are convincing, they lack hard evidence.

Running applications that are not graphics intense works to a sufficient degree. Session handling is acceptable but could use a bit more polishing. Local printing is supported by at least one server, but is do not work in my setup because of bugs. What needs a lot more attention is multimedia and graphics. By moving codecs to the client we may reduce the server side media players role to streaming data and showing the user interface. This may solve most problems with multimedia, including synchronization.

3D graphics is another problem area that may be more difficult to solve. At least we must move the render loop to the client in order to have some efficiency. This may prove problematic since rendering is usually hardcoded in games, so they have to be aware of any such technology. An efficient solution may require the ability to run some kind of bytecode locally, compromising the ideal solution where applications are run server side. However without such a trick, it may prove very hard to perform as well as a game-specific client. With such a trick, the client may be adaptable to time-saving, game-specific algorithms.

Because of those limitations terminal server technology is currently only fit for office usage. However, video presentations that may be needed in some offices are becoming more. A trend is that corporations present information about themselves in video, and distribute information to employees in video form. If that is the case employees must be able to watch them.

While the terminal server approach is promising, the existing technology is not mature enough for everyday usage, only limited office usage. Most users require multimedia and optionally 3D graphics for a complete desktop experience. This is the areas that needs more attention in order to reach the end users.
7 Appendix: Installation Of Terminal Servers

7.1 Installation of FreeNX-server

As FreeNX is included in CentOS the installation procedure is quite simple.

```
yum -y install freenx
```

Use the sample configuration.

```
cp /etc/nxserver/node.conf.sample /etc/nxserver/node.conf
```

7.2 Installation of LX-Server

First create a directory for the server and change to it.

```
[root@localhost ~]# mkdir lxserver && cd lxserver
```

Download LX server. See www.thinstuff.com for download location of current version.

```
[root@localhost lxserver]# wget http://www.thinstuff.com/releases/lxserver-1.2.1-5366.sh
```

Change file mode so the installer can run.

```
[root@localhost lxserver]# chmod +x lxserver-1.2.1-5366.sh
```

Run the installer. Excess information have been removed.

```
[root@localhost lxserver]# ./lxserver-1.2.1-5366.sh
```

Please choose one of the following actions:

- [I]nstall: This deletes your current version (1.1.2, build 3546) and installs 1.2.1, build 5366. Any settings made will be lost.
- [E]xit: Exit installer without changing anything.

We chose to install, accept the license and don't send info to thinstuff.

```
Please enter choice (E/I/R): I
Do you accept the license (yes/no) ? yes
Would you like to send anonymous system information (yes/no) ? no
```

The installer could not find groupadd and useradd. CentOS has those in /usr/sbin.

```
application groupadd not found, please specify the full path: /usr/sbin/groupadd
application useradd not found, please specify the full path: /usr/sbin/useradd
```

Then the installer asks us what desktop environment we want. After that it tries to log in as the
user i su:ed root from. This doesn't work, so i chose to continue with t wm.

Configuring default session program. Please choose one of the following options:

[G]nome
[K]DE

Please enter choice (G/K/T): K
Password (for user ********): ERROR: logon failed for user "******" at server "localhost:2911"
Error reason: Access denied.
setting session type failed with error code 2
Do you want to continue or retry (c/r)? c
If you ignore the problem and continue, the newly installed version will have its session set to the built-int windowmanager (twm).
Do you want to continue (yes/no)? yes

We do not currently support your system for automatic installation of an init.d script. A sample init.d script is provided as /opt/thinstuff/rdpserver/scripts/rdp-server.init.d-template for manual installation.

We are told that there is a init.d template we can use. So we copy it to the appropriate location and start the server.

[root@localhost lxserver]# cp /opt/thinstuff/rdpserver/scripts/rdp-server.init.d-template /etc/init.d/rdp-server
[root@localhost lxserver]# /etc/init.d/rdp-server start

The server is now installed and running.

### 7.3 Installation of xrdp

First we download xrdp from sourceforge.

[root@localhost xrdp]# wget http://dfn.dl.sourceforge.net/sourceforge/xrdp/xrdp-0.4.1.tar.gz

Then we unpack the archive.

[root@localhost xrdp]# tar -zxvf xrdp-0.4.1.tar.gz

Change to the directory extracted from the archive.

[root@localhost xrdp]# cd xrdp-0.4.1

Xrdp has no configure script, so we just run make and make install.

[root@localhost xrdp]# make
[root@localhost xrdp]# make install

Then we start xrdp with the provided start script.

[root@localhost xrdp]# /usr/local/xrdp/xrdp_control.sh start
8 References


http://en.wikipedia.org/wiki/NX_technology


