Universität Klagenfurt

Fakultät für Wirtschaftswissenschaften und Informatik

Institut für Informationstechnologie

Open-Source Multimedia Session-Management for Thin-Clients based on the X Window System

Diplomarbeit

im Fach Angewandte Informatik

zur Erlangung des akademischen Grades

Diplom-Ingenieur

betreut von Prof. Dipl-Ing. Dr. Hermann Hellwagner

eingereicht von Michael Kropfberger

Ehrenwörtliche Erklärung

Ich erkläre ehrenwörtlich, dass ich die vorliegende Schrift verfasst und alle ihre vorausgehenden oder sie begleitenden Arbeiten durchgeführt habe. Die in der Schrift verwendete Literatur sowie das Ausmaß der mir im gesamten Arbeitsvorgang gewährten Unterstützung sind ausnahmslos angegeben. Die Schrift ist noch keiner anderen Prüfungsbehörde vorgelegt worden.

Klagenfurt, im September 2001

Ich widme diese Arbeit meinen Freunden und meiner Familie, besonders aber meinem Vater, der meine Ausbildung immer förderte.

Einleitung

Moderne Server-Betriebssysteme unterstützen mehrere Benutzer, die zur gleichen Zeit Applikationen ausführen können. Die graphische Ausgabe dieser Applikationen erfolgt zumeist auf entfernte Arbeitsstationen. Diese Arbeitsstationen verrichten ausser der graphischen Ausgabe und dem Empfang von Benutzereingaben keine wesentlichen Dienste und können daher mit schwachen, stromsparenden Prozessoren und ohne Festplatten ausgestattet werden. Dies erlaubt weiters einen lüfterlosen Betrieb, der durch absolute Ruhe ein effizienteres Arbeiten ermöglichen sollte. Diese Arbeitsstationen hielten unter dem Namen "Thin Clients" Einzug in die einschlägige Literatur und werden mittlerweile auch in vielen Firmen eingesetzt.

Wenn sich der Benutzer von der Arbeitsstation entfernt (sei es eine Pause oder Dienstschluss), sollte die laufende Sitzung aus Sicherheitsgründen beendet werden. Diese Arbeitsumgebung muss beim nächsten Mal mühsam wiederhergestellt werden, dies inkludiert zum Beispiel das Starten der benötigten Applikationen, Öffnen der zuletzt bearbeiteten Dokumente inklusive der Anpassung aller zugehörigen Fenster, und die Repositionierung des Cursors zum aktuellen Teilabschnitt des Dokuments.

Bei einem Session Management System kann einfach die graphische Ausgabe abgeschaltet werden, die Applikation läuft aber weiterhin auf dem Server. Bei einer Wiederaufnahme der Arbeitssitzung (auch von einer beliebigen anderen Arbeitsstation aus) wird der begonnene Arbeitsgang genau dort fortgesetzt, wo er davor beendet wurde.

Diese Arbeit zeigt bereits existierende Lösungen, freie sowie kommerzielle. Weiters wird ein neuer Ansatz vorgestellt, der die freie Software XFree86, eine Programmierschnittstelle für die graphische Ausgabe auf UNIX Systemen, mit einem Session Management System erweitert. Zur weiteren Performanzsteigerung wird diese Software mit einem Kompressionsprogramm für XFree86 erweitert, um Sitzungen auch über langsamere Leitungen zügig betreiben zu können. Um diese Performanz zu belegen, folgen ausgiebige Tests und Vergleichsgraphiken.

All diese Anforderungen an ein graphisches Session Management System können (und müssen) auch an die Audioausgabe gestellt werden. Somit werden auch hier die möglichen Alternativen gegenübergestellt und eine neue, bessere Lösung vorgestellt, die allen Bedingungen eines Session Management Systems gerecht wird.

Zuletzt werden noch mögliche zukünftige Erweiterungen und Verbesserungen aufgezeigt, um durch weitere Bandbreiteneinsparungen zum Beispiel auch Video effizient übertragen zu können.

Contents

List of Figures

Chapter 1

Introduction and Rationale

All modern operating systems offer a way to directly log on to the *local* computer and work there with a graphical user interface. When the user has finished his work (at least for the moment), he logs out and the actual running session is closed, maybe including rudimentary storage of opened windows, their positions and opened files. Unix-based operating systems offer the possibility to open a *remote* session with a complete working desktop based on the X Window System[15]. One may redirect the input and output to a low-cost computer (also called thin-client) , while effectively working on a "fat" server, but logging out also means closing the complete X session with all open applications.

Logging out introduces inconvenience by not restoring the completely same desktop view on re-login. And if a partial session management is available, logging out – and in – takes a while, since all applications have to be closed and restarted. When the user is logged out, it is also impossible to let the computer do some useful background work like eg. long-period mathematical calculations.

To circumvent this drawback, more and more products offer "real" session management, where a user reconnects to an always running session, just redirecting its graphical output to his local workstation or thin-client. So logging out only means to close down the local output and logging in means to redirect the graphical output to any workstation, which provides a display, keyboard and mouse for the user. So after a reconnect, even the cursor will be at the same position, everything is in its old state.

This also allows immediate user roaming, eg. from his thin-client at his teleworking place at home to his thin-client in his office, without changing a thing to the already running session on the server, except maybe a perceivable increase of network speed and hereby an increase of “snappiness” of the graphical user interface.

Modern computer work also includes sound, might it be accoustic feedback or streamed music. This also has to be included into the session management, which means that the sound is forwarded to the currently connected working place or – with no connection – stays quiet on the server-side.

Another interesting feature of thin-client computing is that no harddisk and high computing power is needed at the thin-clients. This allows to turn off all internal fans and eliminates the very annoying hiss of harddisks. Hereby working efficiency can be increased because the person in front of the totally silent computer can fully concentrate on his work[2].

The goal of this work was to write an open source program (discussed in Chapter 3), which offers “real” session management, based on the open source X window system implementation XFree86. The display protocol should be highly efficient, extensible and should use a well defined standard. In addition, an already existing open-source program for sound forwarding had to be highly extended to suffice the needs of session management, which will be discussed in Chapter 6. Those two session management facilities for audio and displaying together lead to a multimedia session management ideal for the usage on high-performance thin-clients like CAD computers with modern user interactiveness.

Chapter 2

Multi-User Graphical Session Management

2.1 Requirements for Multi-User Systems

Before starting, we have to define the necessary features of a multi-user system.

First of all, we want to have multiple users completely working on one machine but only redirecting the GUI output to the connected workstations. This disqualifies all older Microsoft Windows products, since their only multi-user ability is defined on shared files or special client/server applications, where multiple other workstations (with their own instance of Microsoft Windows) can access the data of the server. This does not mean connecting to the server, starting any remote applications and redirecting graphical output!

Multi-user ability starts with the Microsoft Terminal Server Edition for Windows NT and Windows 2000. Unix was capable of multiple users since the very beginning: this was a major selling issue for enterprise server operating systems.

To serve multiple users on one server, every user has to have his own *virtual memory area*, so even if the same application is started twice by user A and user B, they run two independant instances and these instances cannot interfere with each other, since they are in different address spaces.

Applications in memory contain binary code and changing data for the running session (numbers, names,...). So it is quite useful to reuse and share the binary code for multiple instances, and only use memory for the different data instances. That is why many applications are linked dynamically against libraries, where these libraries are stored in memory only once.

So if an application starts up for the second time, it only has to copy its data area to newly allocated memory. To be even more efficient, this copying will be postponed until the first write (and hereby a real change of the own data instance) of the application. This is called *copy-on-write*. In our case with multiple sessions, this allows the operating system to save memory by using only one instance of eg. pixmaps of multiply used file browsers.

Hosting multiple users on one server also decreases complexity of authentication, since all user data including passwords is stored on one centralized server, far away from any physical contact of malicious persons. But now we even need stronger precautions to keep away malicious code from the server. Security holes, kernel bugs or hardware failures are even more dangerous, since a server crash would lead to total collapse of all connected terminals. This means to paralyze a whole company department with one blink of an eye.

These concepts are discussed in further detail in any arbitrary introductions and books about operating systems, like Andrew Tanenbaum's "Operating Systems"[1].

2.2 Alternative Remote Display Protocols

In the following section the available software for remote display protocols will be introduced including their session management abilities, their internals and their advantages and disadvantages. We will discuss AT&Ts VNC, Microsoft Terminal Server, Citrix Metaframe, Tarantella, and the SunRay architecture.

2.2.1 VNC

All mentioned programs except VNC do meet the prior discussed features of multi-user systems. VNC can only be used as a redirection of a running session, so this also works with single-user systems like earlier Microsoft Windows versions. It does not extend the underlying operating system with any multi-user features but implements a simple remote display protocol.

VNC stands for Virtual Network Computing and was developed at the AT&T Laboratories[3] in Cambridge. It is licensed under the GNU General Public Licence, so it is free, open-source software.

VNC is based on a very simple bitmap oriented protocol and only connects to the framebuffer layer of the output, so it does not know anything about windows or icons but only their displaying pixels. Therefore it is very easy to implement on different operating systems and for different computer hardware. It allows to start a complete session from various systems like MS Windows, Linux, Apple OS or BeOS and redirect it via TCP/IP on all those systems interchangeably. So it is possible to work locally with a Windows PC and have an open VNC window, which shows a display to a Unix server.

The protocol is called *Remote Framebuffer[5]* (RFB) and is based on only one graphics primitive: "Put a rectangle of pixel data at a given x,y position". Besides that, it also sends keyboard and mouse events from the client to the server.

To increase performance, there are three main features supported by RFB:

- incremental framebuffer update, so only changed areas are sent
- different encoding algorithms for rectangle areas, like raw, hex-tile (a 2D run-length encoding), JPEG or "Copy Rectangle", where an area is just moved at the client-side
- client "pull" approach for framebuffer updates, which means the VNC client

will request a full screen update only when it has received and processed the prior screen update completely.

This "pull" approach may help keeping the network load low when used over slow lines, but also increases latency. This is especially annoying on fast networks or when interactive responsiveness is required. On the other hand, working over a slow line does not slow down the server-side computation and output, since the client-side output is not tightly coupled with the server-side output.

Finally, VNC cannot transfer audio output to the thin-client.

2.2.2 Microsoft Terminal Server

Windows NT was designed in the early 1990s for single-user workstations, and had no option for multi-user operation these days. Around 1995, Citrix licensed the source code of NT 3.51 and changed it to support multiple remote users with their proprietary *Intelligent Console Architecture[9]* (ICA) protocol. They called the new NT *WinFrame*. In May 1997, Microsoft agreed with Citrix to include the basic support for thin-clients and remote desktop PCs into the standard operating system distribution[8].

This *MS Terminal Server Edition* is already built into the newest Microsoft server product, Microsoft Windows 2000. All display transfers use the *Remote Desktop Protocol* (RDP), which is on top of TCP/IP. RDP is based on the ITU's T.120 protocol[10]. This international standard allows up to 64.000 different communication channels over the same connection (eg. keyboard, mouse and video drawings). Microsoft uses one channel and puts everything in it, using the T.128 protocol[10]. Since the RDP protocol itself is kept proprietary by Microsoft, there is no real information about the protocol primtives available.

The application software has not to be aware of the underlying *Terminal Server*, since all applications are treated in separated session spaces with unique names. Only the GUI parts are captured and sent over the wire. Audio is transferred to the connected thin-client within the RDP stream.

The clients use a 1.5 MB local buffer to cache images, toolbar icons and cursors etc. but they are not caching unicode strings because of security issues. The cache is organized with the *least recently used* algorithm (LRU).

On the server-side there is a framebuffer cache, so the display commands are not sent in a bitstream manner but blockwise. When the user is highly interacting with the system (keyboard or mouse inputs), the server sends out screen updates 20 times per second. With no interaction, it slows down to 10 times per second. When there are heavy screen updates, the server sends the changes immediately, so on a fast network also movies should be viewable. With no screen changes on the server-side, the server idles and does not send any updates[7].

2.2.3 Citrix Metaframe

Citrix Metaframe extends the *Microsoft Terminal Server* technology with a protocol plugin instead of RDP, called ICA[9]. Citrix offers multiple clients for all various platforms like Mac OS, DOS, OS/2 and all kinds of Unix and even an ActiveX control to embed a session into a web browser. It is designed to run over industry-standard network protocols, such as TCP/IP, NetBEUI, IPX/SPX, and PPP and industry-standard transport protocols, such as ISDN, Frame Relay and ATM.

It extends the *MS Terminal Server* with large-scale server computing features like application publishing and load balancing on a server farm. To make the basic *MS Terminal Server* usable via low-bandwidth connections, *Citrix Metaframe* also includes some compression algorithms (eg. run-length encoding for bitmaps) and intelligent caching of windows objects, such as bitmaps, brushes, glyphs and pointers.

The ICA protocol surpasses the RDP protocol with client-side features such as client drive mapping, client printer mapping, client serial port redirection and remote clipboard support.

The ICA packet consists of a required one-byte command, followed by the according command data. Reflecting the connection negotiations, there might be multiple preambles, like for error detection and recovery, encryption or compression (see Figure 2.1).

These options are also dependent on the underlying connection, so on TCP/IP, the error detection and recovery preamble will not be necessary. Since ICA is flexible and extensible, other preambles might be added in future versions.

Again, all further protocol details are proprietary and kept away from the public.

Frame Head	Reliable	Encrypt	Compress	COMMAND	COMMAND DATA	Frame Tail

Figure 2.1: Citrix Metaframe Extensible ICA Packet

2.2.4 Tarantella

SCO's *Tarantella*[11] offers a remote desktop via a normal Java-enabled browser, and acts as a middleware between different desktop systems (Windows, Unix, ...) and the thin-client browser. Therefore it is operating systems independent.

On the first connect to the *Tarantella* server, a Java applet is downloaded to support the protocol locally in the browser. This applet is then stored locally, so the next connection will be immediate.

When used with Microsoft's *Terminal Server*, it extends the RDP protocol with virtually the same features as *Citrix Metaframe*, like drive mapping and local printing.

Tarantella deploys the so-called *Adaptive Internet Protocol [12]*(AIP), which informs the middleware about connection speed and latency. Command compression, interlaced images, graphical optimization and delayed updates can be pre-configured on a per-application basis or will be dynamically adjusted to the connection speed or user-based bandwidth limits.

In connection with Unix based X sessions, the *Tarantella* middleware also takes advantage of AIP and session management, but, if the thin-client has a running X server, *Tarantella* can be instructed to directly tunnel the X commands to the client-side X server. But by doing so, all AIP features – most importantly compression and session management – are lost, since the basic X protocol is not capable of these yet. How to circumvent these drawbacks will be discussed in section 3 of this document.

Tarantella offers load balancing of up to 12 *Tarantella* middleware servers, LDAP authentication, and data encryption via SSLv3.

2.2.5 SunRay

Sun Microsystems, historically selling high-end servers, recently came out with the *SunRays*, which are dumb terminals connected to a special *SunRay* server software running on Sun servers with the Solaris operating system. There is a special middleware to redirect also Windows applications via the Sun server to the *SunRay*, using Citrix' ICA protocol (see Figure 2.2)[13].

The server software offers load balancing for applications, and high availability by fail-over for the login servers.

SunRay is the only product that is tightly coupled to the hardware, since the client software is stored in the firmware and loaded on start-up. Therefore no clients exist for any other platform or operating system.

The client unit contains a low-cost 100MHz microSPARC-IIep processor, an ATI Rage 128 graphics card with a maximum of 1280x1024 at 76Hz refresh rate, and 8 MB memory, where only 2 MB are used. This is enough to cope with the simple *SunRay* Hot Desk protocol, which is similar to the RFB protocol used by *VNC*, so it only sends framebuffer contents with pixel commands. But unlike *VNC*, audio output redirection is possible and the audio stream roams from one

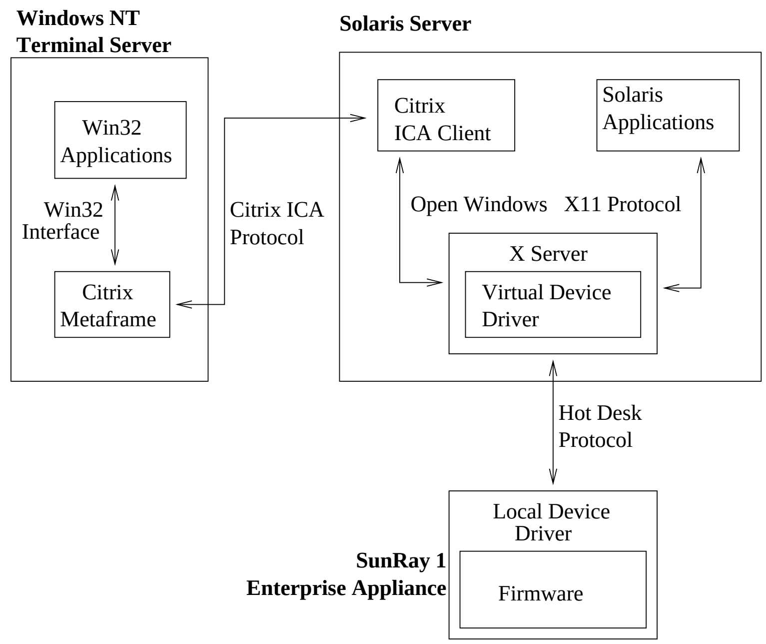

Figure 2.2: SunRay Displaying Microsoft Windows and X11 Applications

SunRay to the other, when the session is disconnected and reconnected from somewhere else.

The SunRay unit comes with a smartcard reader, so every user can log in or reconnect just by plugging in his identifying card.

The protocol is only usable on 100/10 Mbps networks, since there are no outstanding compression algorithms implemented. It does not support encryption or routing capabilities, since it is based on a proprietary network protocol based on UDP/IP.

With the newest version 1.2 of the *SunRay* server software, local printing and serial ports are enabled. Also multi-head operation, based on the X11R6.4 extension XINERAMA[14], is supported. This means spreading one desktop over multiple monitors and the mouse pointer moves from one screen to the other, including drag'n'drop and moving windows.

Since *SunRay* is based on a proprietary protocol, Sun Microsystems keeps away any in-detail information about the exact protocol details.

2.2.6 Comparison Chart

Product	Server OS	Client OS	Display Encoding	Screen Updates	Client Caching	Max. Resolution
MS Terminal Services	Win2000, WinNT	Win2000, WinNT	compressed graphics	server frame buffer, pushed adaptively	glyphs, small bitmaps in RAM, large bitmaps on disk (Cache: 1.5 MB RAM, 10 MB HDD)	1024x768 @8bit
Citrix Metaframe	Win2000, WinNT	Win, OS/2, Linux, Unices, DOS, Mac	compressed graphics	server frame buffer, pushed adaptively	glyphs, small bitmaps in RAM, large bitmaps on disk (Cache: 3 MB RAM, 1% of HDD)	no restriction

Figure 2.3: Comparison of Existing Remote Display Systems

Product	Server OS	Client OS	Display Encoding	Screen Updates	Client Caching	Max. Resolution
SCO Tarantella	Solaris, AIX, SCO, Linux	Java-enabled web browsers (MSIE, Netscape)	compressed graphics, directives or raw (adaptive)	server frame buffer, pushed adaptively	glyphs and bitmaps (only in RAM)	no restriction (depends on middleware server memory) @8 bit (newest development version: 24bit)
At&T VNC	Windows, X, Mac	Win, X, Mac, Java, WinCE	2D run-length encoded pixels	screen differences sent on client pull	None	no restriction
SunRay	Solaris	proprietary device	compressed pixels	update pushed on each window system command	None	no restriction

Figure 2.4: Comparison of Existing Remote Display Systems (cont.)

2.3 The X Window System

The X Protocol[16] splits an application's point of execution from its output by introducing a layer of graphic command directives (see Figure 2.5). This library is called *X Library*[20] (XLib).

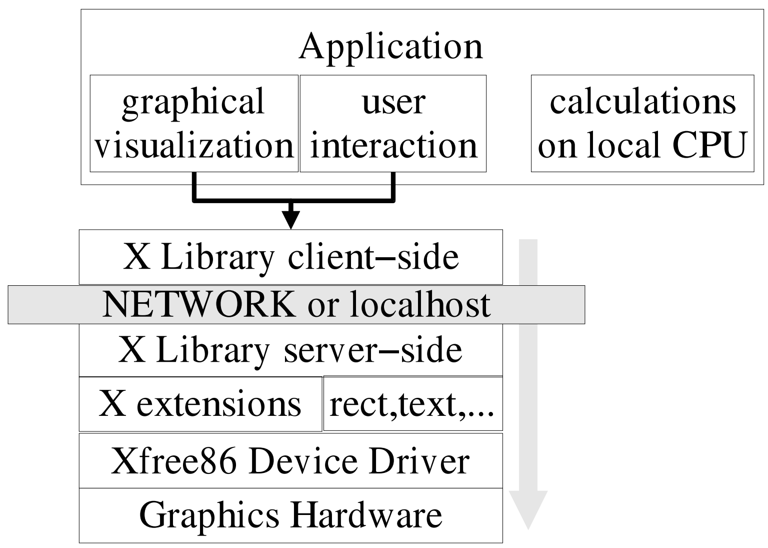

Figure 2.5: Separation of Graphical Output and Application Execution

X is independent of the underlying operating systems' multi-user features since it is only up to the operating system to cope with multiple instances of the same application, in this case displaying on eg. different users' workstations using the X Protocol. The X Protocol has no session management feature per se, but in this work we will introduce an intelligent X proxy named *X-Ray* which will fill this gap.

2.3.1 Different Implementations

XFree86[17] is a freely redistributable open-source implementation of the X Window System [15] that runs on UNIX(R), UNIX-like operating systems (like Linux, the BSDs, Mac OS X and Solaris x86 series) and OS/2.

XFree86 is widely used and comes pre-installed with all flavors of Linux, so

XFree86 is one of the most driving forces in the development of the X Window System, which has a version number of X11R6, so-called Revision 6. The actual XFree86 is version 4.1.0, which is totally compliant to these X11 Revision 6 standards.

Since this work is based on open source software, it uses XFree86 and not one of the commercial X implementations like *eXceed*[18] or *Accelerated X*[19].

The XFree86 Project offers a highly efficient implementation of the X protocol for a wide variety of graphics cards (based on ISA, PCI and AGP bus systems), including 2D and 3D optimizations.

Accelerated X is based on a former version of XFree86 but lead into a commercial, closed-source program with many optimizations for 3D acceleration and laptop displays.

eXceed is a totally independent code base, and runs an X-Server as a Windows program, so it is possible to connect normal X applications from Unix machines to a Windows computer.

2.3.2 X Command Primitives and Objects

The approach for handling graphical contexts like windows or pixmaps is defined by the X Consortium and the concept is highly object oriented. Even though XFree86 is implemented in C, the idea of object orientation is well preserved as far as possible using C.

Every running X-Server offers at least one *screen* with at least one *visual* for each possible color depth. For each *visual* and its corresponding color depth, there is a default *colormap* and a *root window* defined. *Windows* can be created on a *visual*, where each *window* has a parent window, except the very top *root window*, which is administered by the X-Server itself (see Figure 2.6).

X provides graphics, text, and raster operations for *windows*. X does not guarantee to preserve the contents of windows, but sends *Exposure Events* to the applications, so they can redraw the exposed window areas.

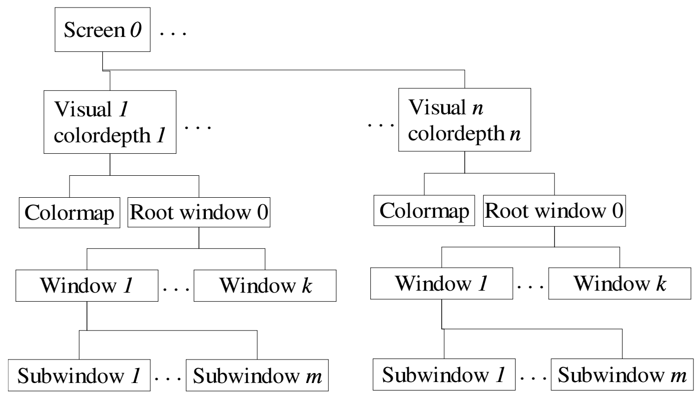

Figure 2.6: The Data Structures of the X Window System

Off-screen storage for graphical objects, called *pixmaps*, is supported in various color depths, depending on the X-Server and the underlying hardware.

The server offers *fonts* from different sources like font files or font servers somewhere in the network. Applications can get handles on those *fonts* and use them for writing on *windows* and *pixmaps*.

There is also support for application specific *colormaps*, including colormaps based on color management tools, which remap the eg. screen-drawn colors to the exact matches on printer output.

Finally getting some pixels, rectangles, circles, text and more drawn on *windows* or *pixmaps* has to happen in conjunction with allocated *graphic contexts (GCs)*. A *GC* holds information about foreground and background colors, tile and stipple pixmaps, line width, fill style and many more. All properties are stored in the GC struct definition (see Section A.1).

It is also possible, to set different *cursor* shapes when eg. moving over buttons or windows.

All these X objects are stored with unique IDs. The pointers to the according data structures are coded into the XIDs (see Figure 2.7).

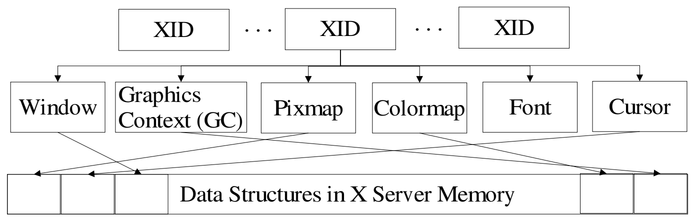

Figure 2.7: XIDs and their Data Structures in Memory

All these manipulations happen on X-Server side, so the application only sends commands like “move window W_i to position x, y” or “draw a line from $x1, y1$ to $x2, y2$ and use GC_j with dashed line mode set”. All these commands are represented by various X directives. This keeps the load of application based calculations on the application side, and graphical based calculations on the X-Server side. Also, applications don’t have to know anything about the displaying on a specific hardware, but rely on predefined X library functions. This allows high optimization (eg. 2D and 3D accelerations, exploiting special hardware features) of the X-Server running on special hardware.

To get some information about mouse movements and keystrokes, the X-Server sends *events* to the connected applications, eg. when a mouse moves into an application’s window or when a mouse button is pressed. It also send information about obscured window areas, resize events and many more[20].

2.3.3 X Compression with ML-View

ML-View[23] is based on the *Differential X Protocol Compressor*[24] DXPC and offers X compression to an average ratio of 60:1[23] with a normal user session of emailing, programming and browsing the internet. The compression is divided in five levels.

1. Caching of X directives as a whole with a ’fingerprint’ and a ’data’ part. The cache hits are between 80% and 90% and the principle is like the IP header compression for low-speed serial lines described in RFC2508[25].

2. If no identical message is in the cache, a differential compression is applied on a field basis. So in a `PutPixel` message maybe only the Y-axis changes from one message to the other and we only send the message fingerprint with the differential change of the Y-axis. This idea is also used in RFC2508.

3. LZO compression on `PutImage` data.

4. ZLIB compression of bursts of multiple messages of the prior levels, which are put together in bigger packets before they are sent.

5. Relevant packets are sent prior to delayable packets, 'groups' of messages are set to meet the exact MTU size of the link.

The real problem with X is not the bandwidth but latency and high roundtrip times. Every X message sent from an application via XLib (see Figure 2.5) has to be acknowledged by the X-Server. The application is blocked until the last sent messages are acknowledged.

2.3.4 Continued Comparison Chart

To see all features of standalone X and in conjunction with compression at a glance, the following table may be compared with the table of alternatives (see Section 2.2.6).

Product	Server OS	Client OS	Display Encoding	Screen Updates	Client Caching	Max. Resolution
X	X-Server (Win, Unices, Linux)	X-Server (Win, Unices, Linux, Mac, Java)	draw primitives	block-wise sending of X commands (eg. 5 in a row)	glyphs, fonts, pixmaps	no restriction
X with ML-View	X-Server (Unices, Linux)	X-Server (Win, Unices, Linux, Mac, Java)	message caching, draw primitives with bitmap compression	block-wise sending of X commands (eg. 5 in a row), intelligent reordering and prioritizing	glyphs, fonts, pixmaps	no restriction

Figure 2.8: Continued Comparison of Remote Display Systems (also see Figures 2.3 and 2.4)

Chapter 3

X-Ray – X-Based Graphical Session Management

Basically, the aim of this work was to write a graphical session management which is not based on a pixel oriented protocol like VNCs *Remote Frame Buffer* (RFB) (see Section 2.2.1), but using the well known X protocol. This offers possibilities like connecting through intelligent application firewalls (which not only open up the well known X protocol TCP/IP ports 6000 and up, but really understand and filter the X protocol itself for possible malicious commands or denial of service attacks). Using the well-defined X protocol also allows connecting to proxies like *ML-View*[23], which compresses pixmap transfers and optimizes X command queues by their importance. On the client-side, having the uninterpreted X commands allows high optimization to the local hardware graphics acceleration or font and color rendering.

X-Ray is based on the XFree86 4.1.0 sources, which are highly reuseable and well documented. There is a very generic code base with lots of libraries. Graphic card drivers overwrite some specific functions to cope with their hardware specialties. In addition to the real graphic card drivers, XFree86 also comes with a purely virtual X-Server *Xvfb* and a nested X-Server *XNest*.

3.1 Xvfb

Xvfb behaves like a "normal" X-Server but only works on a frame buffer in main memory. This screen memory area might be marked as shared memory, so other applications can access the screen contents. There also exists a tool to extract the actual screen contents into a graphic file (screen dump). When you view this file, you can see all opened windows, pictures and even the mouse pointer frozen on its position at the time of the snapshot.

Xvfb offers different resolutions and color depths, so it is ideal for testing.

3.2 XNest

XNest starts on an already running X-Server, comes up as a normal application window, but in this window it can start various other applications including a window manager or screen savers. *Xvfb* goes into the right direction, and would suit best to implement a simple framebuffer oriented protocol with pixel updates. In our case, we have to attach the necessary hooks at a lower level, where the X commands are handled.

XNest already is a kind of a layer between applications and another real X-Server. It internally forwards all X commands directly to the hosting X-Server (see Figure 3.1), thus it is device independent and may also display an application on an X-Server running on another operating system and/or hardware platform.

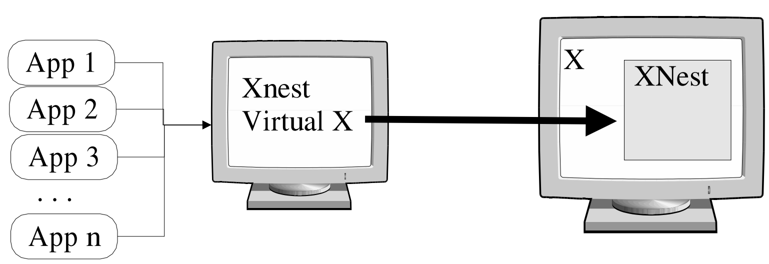

Figure 3.1: XNest Provides a Proxy X-Server for Applications, Forwarding Output to a Remote X-Server

X applications (using the X libraries[20]) always need a running X-Server to connect to, and also during run-time, they need a working output, where they can draw on or read the contents of specific screen areas. At startup, they also negotiate about provided color depths, default screens and other X-Server specific features.

XNest exactly provides the same features as its hosting X-Server, so applications connecting to *XNest* can take full advantage of client-side graphics card features and are displayed on the hosting X-Server system with nearly no delay (see the benchmarks in Chapter 4). The hosting X-Server might be an arbitrary X implementation, namely XFree86 or any (commercial) alternative.

3.3 XRay

To extend arbitrary X applications with session management functionality without interference to their source code, they have to find their well-known environment, which means the X protocol. So the only way to connect and disconnect a remote display during application run-time is to double all application outputs and always leave one output running on a *server-side* computer.

And this is, basically, what *X-Ray* does. On a dedicated session server (called *X-Ray server-side*) all running applications can draw and read screen contents, and – if a thin-client (called *X-Ray client-side*) is connected – all X commands are also sent to the thin-client. Figure 3.2 shows the general idea of the *X-Ray* session management.

For further reading, remember that *X-Ray* is using a hosting X-Server on the *server-side* and *client-side* of whatever X implementation (XFree86 or commercial pendants), the only necessity is their total equality in terms of offered screen sizes and display color depths.

This double draw mode is called *fulldraw* mode, since we double network load to reach both X-Servers for output and input. To optimize display performance, we also offer a *clientdraw* mode, which reduces all draw primitives (draw lines

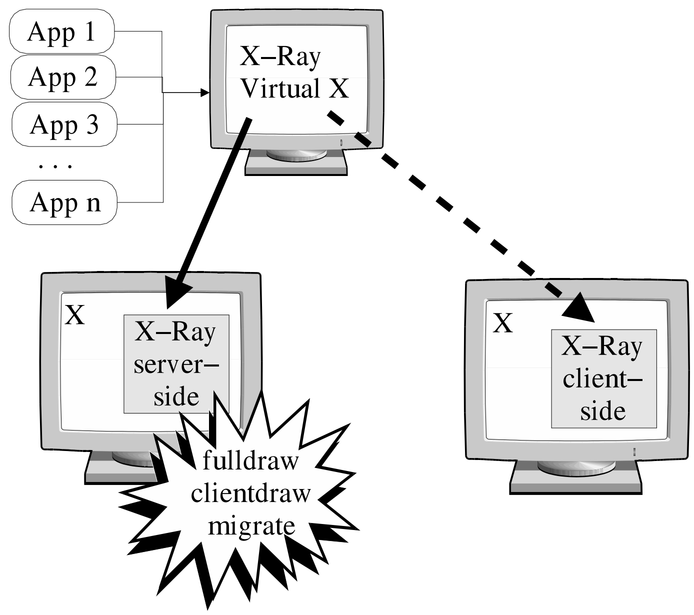

Figure 3.2: X-Ray Draws on a Server-Side and – if Connected – on the Client-Side.

and circles, put pixels...) to either the *X-Ray server-side* or – if connected – to the *X-Ray client-side*. The *clientdraw* mode is partially failsave to an ungraceful disconnect of the *X-Ray client-side* since all X objects like windows and their states are stored on the *X-Ray server-side* too. The only missing information is the screen content painted by draw primitives, but the X-Server can just send a *redraw* event to all applications, so they will repeat their drawing commands.

Finally there exists the *migrate* mode, which means a total migration of the display output from the *X-Ray server-side* to the *X-Ray client-side*, which after disconnect, switches back to the *X-Ray server-side*. The *migrate* mode is not failsave, so if the *X-Ray client-side* is disconnected ungracefully, also the *X-Ray server-side* cannot be kept running since all important information about X objects is missing.

To compress and optimize the X command stream over a low speed network line, it is easy to put ML-View (described in Section 2.3.3) in between (see Figure 3.3).

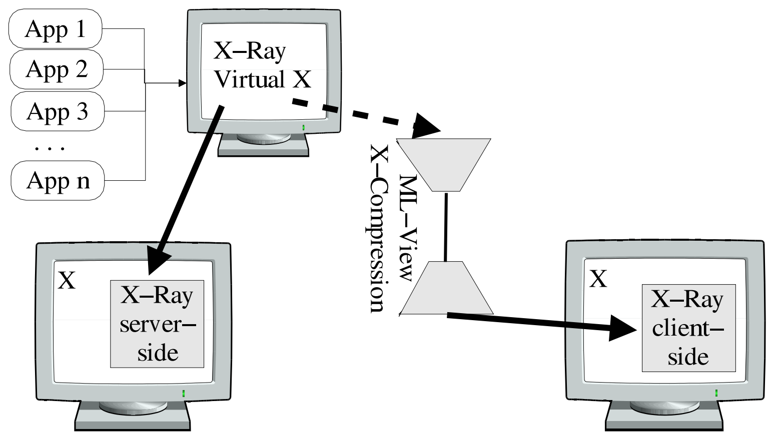

Figure 3.3: X-Ray Client Connection via ML-View

3.4 Memory Footprint and other Server-Side Issues of X-Ray

Every application which displays via X, has to create X objects like windows, pixmaps and so on (see Section 2.3.2). These objects allocate memory on every connected X-Server, so in the case of *X-Ray* this means the same amount of memory on the *server-side* as on the *client-side* (except for the `migrate` mode, where only one connection is active).

When we allow multiple users to start their sessions on the same *server-side*, every user session is started in one big window on the *server-side* running X-server. This big window contains the full session with all subwindows, pixmaps and so on. For functionality, it does not matter if those user session windows are obscured by each other and the number of possible sessions is only limited by the amount of memory available for the X-server on the *server-side*.

To reduce the needed memory for often reused pixmaps or fonts, many desktop environments (like KDE[26]) support pixmap and font caching facilities directly in their APIs, so additional memory is only used for really personalized window data.

3.5 X-Ray in a Large-scale Client-Server Environment

For user acceptance, it is not enough to offer outstanding features like session management. The software also has to be easy to use and – if possible – should not introduce any behavioral and visual change to the existing system at all. To achieve this, we have to analyze a common office (CAD) working place and try to seamlessly fit X-Ray into this system.

Normally a user will go to work in the morning, turns his computer on and waits until boot up, when a login dialog for his username and password appears on the screen. After correct authentication, some scripts are started, the user's home directory is mapped to a local directory, some additional settings are done. Let us assume, we have a distributed client-server approach with application servers, where all users can connect to and start applications, which redirect their output to the local desktop.

When the user goes home or just leaves the office, he has to log out or lock the screen. When he comes back, he has to log in with username and password again.

It doesn't really matter here, if the authentication process is done per typing in a username and password or by sliding a chipcard through a reader, but there definitely will be a standard procedure in every company which at least cares a little about user and company security and privacy. Deploying our X-Ray session management will have to hook into this authentication process seamlessly while providing additional value.

To upgrade the company with X-Ray session management capabilities, we need some *X session servers* with identical graphic cards or with at least an identical number of color depths as the thin clients will support (see section 3.8 for detailed requirements). These *session servers* act as the *X-Ray server-side* session, where one single X server has to be started to serve all user sessions.

The *session servers* may also be used as the *application servers* for user applications, but it is also possible to out-source this job to a special *application server cluster*.

There is one dedicated server, acting as the *connector*, so all thin clients at the offices are configured with the *connector's* host name for session connection. If high availability and load-balancing of a *connector cluster* are also necessary, the easiest would be the DNS standard feature of round-robin DNS entries.

3.5.1 Connection Scheme

When a user turns on the thin client, the system boots out of flash memory and starts the local (*client-side*) X server. Then a login dialog window with empty fields for the username and password pops up. The user authenticates and the thin client sends the username, password and the local IP address to the *connector* (Step 1 in Figure 3.4).

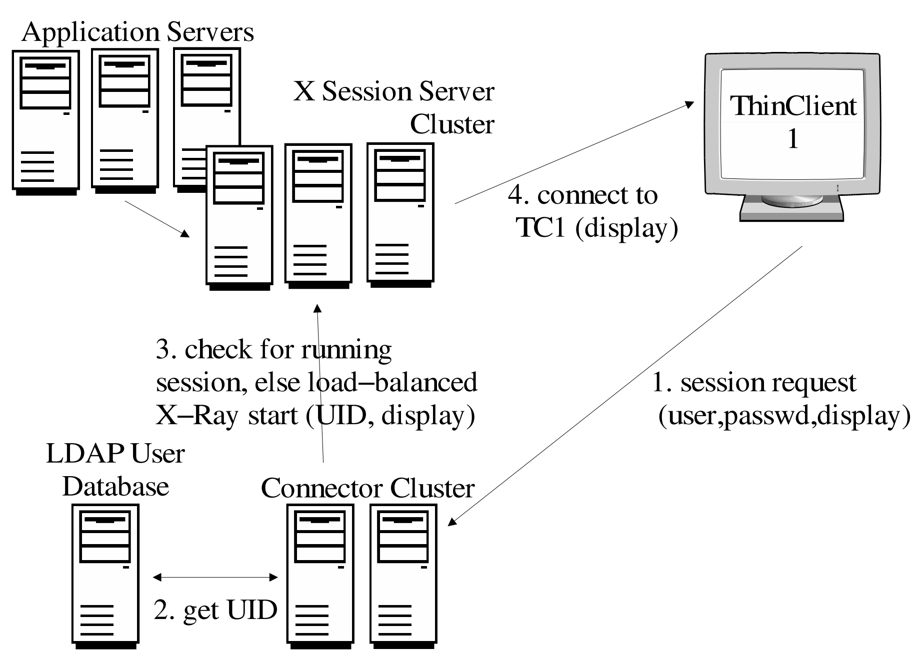

Figure 3.4: X-Ray Connection Scheme in a Large-Scale Client-Server Environment

The *connector* asks the built-in user database (or maybe a the connected LDAP server) to get a unique user-ID for the received username and authenticates it with the given password (Step 2).

After that, the connector checks if there is a running X-Ray session for that user on the *session server cluster* (Step 3). If not, it starts the X-Ray session on a *session server* SS_n with the smallest number of open sessions and starts the desktop environment (window manager, pager, standard applications) on the load balanced *application server cluster*, where all those applications connect to the X-Ray on SS_n. Since X-Ray is based on XNest, it emulates a totally valid X server to all connecting clients. By definition, the X protocol is highly distributed, so each application may be started on different servers of the *application server cluster*, but all are displayed on the user's X-Ray session on *session server* SS_n.

Finally the complete session for the user is up and running, but not yet visible at the *client-side* thin client, where the user is waiting to get his working environment.

The *connector* just has to tell the X-Ray session on SS_n to duplicate its output to the given thin client display (Step 4).

3.5.2 Disconnection Scheme

When the user is done with working and wants to temporarily disconnect, he hits a disconnect button or selects an entry in his desktop menu. This invokes the disconnect script on one of the *application servers*, which immediately sends the username, password and the thin client display's IP address to the *connector* (Step 1 in Figure 3.5).

The *connector* maps the username to his unique user-ID by LDAP or with his local user database and checks for the running X-Ray session in the *session server cluster* (Steps 2 and 3). Then it tells the X-Ray session to disconnect the *client-side* display (Step 4).

On the thin client, the display gets blanked and a new login dialog pops up

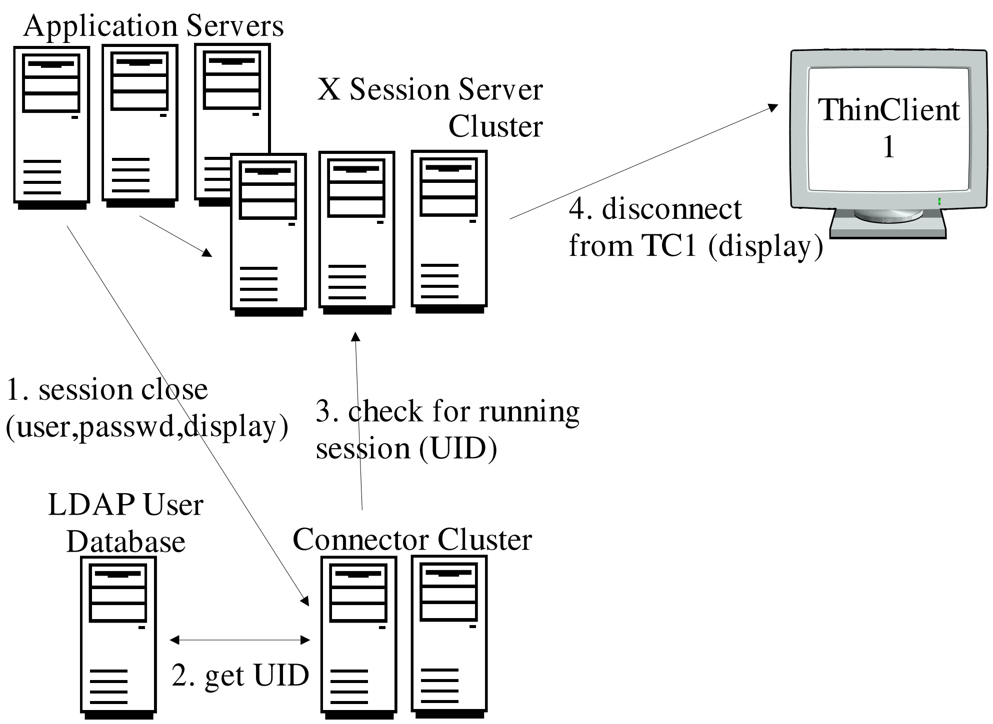

Figure 3.5: X-Ray Disconnection Scheme in a Large-scale Client-Server Environment

for the next user, but the X-Ray session continues to run on the *session server* SS_n. So do all connected applications from the various *application servers*.

To reconnect from any other thin client, the user logs in again, and the *connector* will tell the X-Ray session on SS_n to duplicate its output to the newly given thin client display (Step 4 of Figure 3.4).

When the user is still logged in and wants to really log out and close all applications, he just quits his desktop environment with the normal "logout" button and all connected applications and the X-Ray session on SS_n will be terminated by the login/logout scripts.

3.6 X-Ray Command Line HOWTO

Before we try to deploy *X-Ray* in any real-user environment we have to get familiar with the basic commands for setting up an *X-Ray* session.

X-Ray is distributed on the Web[36] with a patch to the XFree86 sources including all necessary changes to the upper-level makefiles. The filename of the executable *X-Ray* server is `Xray`. The user-space tools `xrayconnect` and `xraydisconnect` and some helper scripts to (dis)connect a session can be found separately on the Web for download.

To use *X-Ray*, we start the *X-Ray* session on the *server-side* display as an X-server with the display `myserver:2` by issuing the following command on the server:

```
Xray :2 -geometry 800x600 -clientdraw
```

This opens a 800x600 *X-Ray* session window on the servers' local X-server. Now we start some applications on the server, which will be displayed in this window:

```
export DISPLAY=myserver:2
/opt/kde2/bin/startkde
```

Finally we connect the *client-side* session to a remote thin-client by typing the following command on the server:

```
xrayconnect myserver:2 client:0
```

After some work we disconnect the *client-side* with

```
xraydisconnect myserver:2 client:0
```

Following the same scheme, we may reconnect as often as we want and redirect the session to wherever we want to.

3.7 Simple (Dis-)Connect Scripts

For easier understanding of the above mentioned large-scale connection schemes, we will show a set of simple shell scripts, which provide the (dis-)connect facility without any high-availability or load-balancing, which could be easily added. But

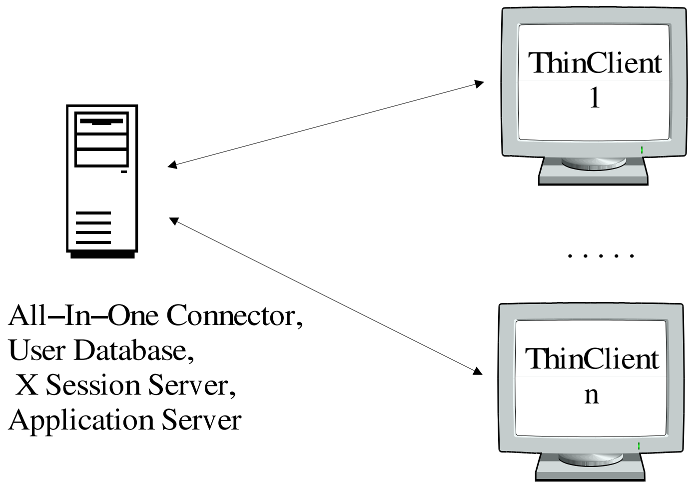

Figure 3.6: Simple Client-Server System Setup with One Server

these scripts are fully functional with the simple client-server system setup shown in Figure 3.6.

One server acts as the *connector* and retrieves the unique user-ID from the `/etc/passwd` file. It runs an X server to provide the *server-side* X host, so it is the *session server*. Multiple users' applications like window managers, Netscape, and task bars are also run on the same server, so it is the *application server* too. This setup should suffice for at least 10 users, since the limiting factor is the *server-side* CPU for the applications and the *server-side* memory for the X-server, its X objects and applications. It will work well for office use with mail and text editors open, but all concurrent users will suffer from high CPU bound and graphics bound applications like CAD or video streaming. To overcome this obstacle of a proportional usage of resources, other schedulers like the lottery scheduler could be installed on the server operating system[27].

```
#!/bin/bash
#####################################
# startscript for Xray-initialization
#
# Author: Michael.Kropfberger@gmx.net
#

BASEDIR=/home/mike/data/uni/dip/src
```

```
XRAYSERVER=kermit
XRAY=$BASEDIR/Xray-wrapper
SIZE=1280x1024
#SIZE=800x600
WINDOWMGR=/opt/kde2/bin/startkde
#WINDOWMGR=wmaker

if [ "$#" -gt "1" ]; then
    USERNAME=$2
    USERID=`grep $USERNAME /etc/passwd | awk -F ':' '{print $3}'`
else
    USERNAME=$USER
    USERID=$UID
    echo "USAGE: $0 THINCLIENT  USERNAME"
    echo "falling back to default user"
fi
LOG=/tmp/xray_UID${USERID}.log

if [ "$#" -gt "0" ]; then
    THINCLIENT=$1
else
    THINCLIENT=$DISPLAY
    echo "USAGE: $0 THINCLIENT  USERNAME"
    echo "falling back to default display"
fi
echo
echo "thinclient display: $THINCLIENT"
echo "real display: $XRAYSERVER:$USERID"
echo "user:  $USERNAME ($USERID)"

#find running instance
if [ -z "`ps aux | grep "$XRAY :$USERID[ ]"`" ]; then
    #start X-Ray session
    if [ $UID == 0 ]; then
        (su - $USERNAME -c "$XRAY :$USERID -geometry $SIZE \
         -display $XRAYSERVER:0 -clientdraw -ac -s 0 " | tee $LOG) &
        (su - $USERNAME -c "export DISPLAY=$XRAYSERVER:$USERID \
          && $WINDOWMGR" 2>&1 | tee -a $LOG)  &
    else
        $XRAY :$USERID -geometry $SIZE \
         -display $XRAYSERVER:0 -clientdraw -ac -s 0 | tee $LOG &
        export DISPLAY=$XRAYSERVER:$USERID \
          && $WINDOWMGR 2>&1 | tee -a $LOG  &
```

```
    fi
fi

#get session to the thin client side
$BASEDIR/xrayconnect $XRAYSERVER:$USERID $THINCLIENT 2>&1 | tee -a $LOG
#force redraw of parent
xrefresh -display $XRAYSERVER:$USERID  2>&1 | tee -a $LOG
echo DONE
```

```
#!/bin/bash
####################################
# stopscript for Xray-initialization
#
# Author: Michael.Kropfberger@gmx.net
#

BASEDIR=/home/mike/data/uni/dip/src
XRAYSERVER=kermit

if [ "$#" -gt "0" ]; then
    USERNAME=$1
    USERID=`grep $USERNAME /etc/passwd | awk -F ':' '{print $3}'`
else
    USERID=$UID
    echo "USAGE: $0 USERNAME"
    echo "fallback to default user"
fi
LOG=/tmp/xray_UID${USERID}.log

echo
echo "thinclient display: $THINCLIENT"
echo "real display: $XRAYSERVER:$USERID"
echo "user: $USERNAME ($USERID)"

#remove session from thin client
$BASEDIR/xraydisconnect $XRAYSERVER:$USERID  2>&1 | tee -a $LOG

#force redraw of parent
xrefresh -display $XRAYSERVER:$USERID  2>&1 | tee -a $LOG
echo DONE
```

3.8 X-Ray Implementation Details

X-Ray seamlessly fits into the XFree86 source code distribution and represents just another possible X-Server in the directory structure at `xc/programs/Xserver/hw/xray` and compiles with the full XFree code by typing `make World` in the upper-most directory `xc`. The *X-Ray* code is heavily based on the *XNest* sources, so the following description of the internal structure also describes *XNest*. A good introduction to implementation details of developing graphics device drivers for X and included data structures, can be found in a paper of Digital Equipment Corp[21].

Now we will look into the *X-Ray* source code to see what really happens on the `Xray` startup and `xrayconnect` and `xraydisconnect`.

First `Xray` parses all command line parameters. The most important for *X-Ray* in its primary usage are:

-geometry sets the size of the virtual X

-fulldraw sends all X commands to server-side and client-side

-clientdraw sends only to client-side

-migrate completely closes the server-side X when client-side is connected

-display where to display the *X-Ray server-side* (normally omitted)

Internally, a connection is done to the *server-side* X-server. It opens a display and creates a pseudo-root window on the default visual there. Then it creates a colormap for every available visual, a Graphic Context and a default drawable (a pixmap) for every supported color depth and pixmap depth (see `xrayOpenDisplay` in `xc/programs/Xserver/hw/xray/Display.c`).

Every available screen is initialized by overwriting a function pointer array to the various X commands like XCreateWindow, XCreateGC or XGetImage, defined in `struct ScreenRec` (see appendix A.2), so they will call *X-Ray* specific functions. Also a default window is created and mapped on each screen, so this can be used as a pseudo-root window for connected X applications (see `xrayOpenScreen` in `xc/programs/Xserver/hw/xray/Screen.c`).

```
###FILE: xc/programs/Xserver/hw/xray/Screen.c
      xrayDefaultWindows[pScreen->myNum] =
        XCreateWindow(xrayDisplay,
                      DefaultRootWindow(xrayDisplay),
                      xrayX + POSITION_OFFSET,
                      xrayY + POSITION_OFFSET,
                      xrayWidth, xrayHeight,
                      xrayBorderWidth,
                      pScreen->rootDepth,
                      InputOutput,
                      xrayDefaultVisual(pScreen),
                      valuemask, &attributes);

    XMapWindow(xrayDisplay, xrayDefaultWindows[pScreen->myNum]);
```

Properties and Atoms are an interesting additional feature of the X protocol. Since many applications have to communicate with each other about some X state or want to retrieve information from an X-server (like a window's name, size hints,...), they would have to send textual tags with changing values of a certain type back and forth. To optimize this, there are atoms, which represent a textual description, typed with other atoms like *STRING* or *INTEGER*. There are predefined atoms, but an application may create its own. Every atom may be used as a type for other atoms itself, so this system is very flexible and extensible.

We need atoms to connect or disconnect from a running *X-Ray* session. When `xrayconnect` tries to open a *client-side* connection, it triggers *X-Ray* by acquiring the atom `XRAY_CLIENT_CONNECT` and sets its value to the *client-side* X-server.

```
###FILE: xrayconnect.c
  prop = XInternAtom(display, "XRAY_CLIENT_CONNECT", False);

  XChangeProperty(display, DefaultRootWindow(display), prop, XA_STRING, 8,
```

```
                  PropModeReplace, thin,
                  strlen(thin));
```

The *DIX* library is used by all X-Servers and includes the main functionality of every X-Server, eg. the `main.c` with the endless dispatch loop but also atom handling. To support our special atom, `xc/programs/Xserver/dix/property.c` had to be extended. Now if someone changes an atom with no parent window, the `_RootPropertyChange(pProp)` function gets called.

```
###FILE: xc/programs/Xserver/dix/property.c
    /* Additional hook for  Xray server */
    if (pWin->parent == NullWindow) {
      extern void _RootPropertyChange();
      _RootPropertyChange(pProp);
    }
```

In the *X-Ray* server, this function checks for the atom name `XRAY_CLIENT_CONNECT`.

Since the *DIX* library is imported by every X-Server binary (namely `XFree86, XNest, Xvfb`), a `_RootPropertyChange(pProp)` function had to be added to all of them too. But exept for *X-Ray*, the function has an empty body, so a good compiler should optimize and therefore delete it.

X-Ray then checks if there is no other *client-side* connection yet. After connecting, a display is opened to the *client-side* X-Server, and all initialization is done for the *client-side* again. It creates a pseudo-root window on the default visual there. Then it creates a colormap for every available visual, and a graphic context and a default drawable (a pixmap) for every supported color depth and pixmap depth.

The *client-side* **has to report the same** visuals, color depth, screens and pixmap depths as the former initialized *server-side*! This is needed, since every connected application asks for server features at start up, sets them internally and then the *client-side* and *server-side* are hereby interchangeable.

The next step is to find all resources wich connected applications already have acquired. Those resources have to be acquired at the *client-side* X-Server, too.

Since X keeps track of all used resources, it is straight forward to run through the list of all *colormaps, fonts, pixmaps, cursors, GCs* and finally all *windows* (see Figure 2.7).

```
###FILE:  xc/programs/Xserver/hw/xray/Events.c
          /* the order is important! */
          FindAllResourcesByType(RT_COLORMAP,ClInitExistingColormaps,CL_CREATE);
          FindAllResourcesByType(RT_FONT,ClInitExistingFonts,CL_CREATE);
          FindAllResourcesByType(RT_PIXMAP,ClInitExistingPixmaps,CL_CREATE);
          FindAllResourcesByType(RT_CURSOR,ClInitExistingCursors,CL_CREATE);
          FindAllResourcesByType(RT_GC,ClInitExistingGCs,CL_CREATE);

          debug0("init the server scratchGCs\n");
          for (i=0;i < screenInfo.numScreens; i++) {
            FreeScratchPixmapsForScreen(i);
            FreeGCperDepth(i);
            FreeDefaultStipple(i);
            CreateScratchPixmapsForScreen(i);
            CreateGCperDepth(i);
            CreateDefaultStipple(i);
          }
          debug0("init the server scratchGCs: DONE\n");
          FindAllResourcesByType(RT_WINDOW,ClInitExistingWindows,CL_CREATE);
```

Reinitializing pixmaps is a little tricky, since a pixmap normally already has some painted content. So we have to create a *pixmap* on the *client-side* X-Server and then request the pixmap contents from the source, the *server-side*, with `XGetImage`. Finally we write the content back into the new pixmap.

Reinitializing GCs needs to create the pixmaps for *tiles* and *stipples* first, which are used as special brush modes. Since it is possible to set special *shape masks* where a GC can draw on, we have to read the original mask and install it on the *client-side*.

Reinitializing windows includes reinitializing all parent windows recursively first. Again we have to reinit the background pixmap, border pixmap and the window shape mask.

After all that initializations, the *client-side* clone is up and running and the already pending X commands from connected applications can be handled by both *client-side* and *server-side*.

3.8.1 Wrapper Structures

As already stated, *X-Ray* works like a proxy between X applications and a real (*server-side*) X-Server, and – if connected – also to a secondary (*client-side*) X-Server. To achieve this, *X-Ray* has to return **one** handle for every acquirable resource like *colormaps, fonts, pixmaps, cursors, GCs* and *windows* to the requesting application. But internally it has to store **two** handles to the resources of the real X-servers, so all resources are extended by a private structure with external handles.

The nomenclature is straight forward, so a preceeding `cl_` of resource names is the handle to the *client-side* resource.

Only for the *window* resource we have to store not only the `window` and `cl_window` handle, but also a `parent`, general window geometry information and a linked list of all siblings (`sibling_above`). Finally we have to store the `bounding_shape` and `clip_shape`, which describes the outer shape of the window and the inner fields where the window is transparent. Now we have to double all variables for the *client-side* (`cl_ ...`) respectively.

```
###FILE: xc/programs/Xserver/hw/xray/Color.h

typedef struct {
  Colormap colormap;
  Colormap cl_colormap;
} xrayPrivColormap;


###FILE: xc/programs/Xserver/hw/xray/XNFont.h
```

```
typedef struct {
  XFontStruct *font_struct;
  XFontStruct *cl_font_struct;
} xrayPrivFont;


###FILE: xc/programs/Xserver/hw/xray/Pixmap.h

typedef struct {
  Pixmap pixmap;
  Pixmap cl_pixmap;
} xrayPrivPixmap;


###FILE: xc/programs/Xserver/hw/xray/Cursor.h

typedef struct {
  Cursor cursor;
  Cursor cl_cursor;
} xrayPrivCursor;


###FILE: xc/programs/Xserver/hw/xray/XNGC.h

typedef struct {
  XlibGC gc;
  XlibGC cl_gc;
  int nClipRects;
} xrayPrivGC;


###FILE: xc/programs/Xserver/hw/xray/XNwindow.h

typedef struct {
  Window window;
  Window parent;
  Window cl_window;
  Window cl_parent;
  int x;
  int y;
  unsigned int width;
  unsigned int height;
  unsigned int border_width;
```

```
  Window sibling_above;
  Window cl_sibling_above;
#ifdef SHAPE
  RegionPtr bounding_shape;
  RegionPtr clip_shape;
  RegionPtr cl_bounding_shape;
  RegionPtr cl_clip_shape;
#endif /* SHAPE */
} xrayPrivWin;
```

Chapter 4

Performance Measurements

4.1 Benchmarking with `x11perf`

In [28] RDP, Citrix, Laplink, VNC and SunRay are compared against the Ziff Davis *i-bench* Web benchmark, which includes displaying text, pixmaps, scrolling a screen and a flash animation clip. All these tests are done with various bandwidths from ISDN up to 100 Mbps LAN. But because of the totally different approaches and hosting systems, it is hard to get really good benchmarks. Also a Web benchmark does not really reflect the daily work in an average office.

This master's thesis only wants to compare the free competitors, which are native X, VNC and X-Ray in diverse setups. It also compares all three with the same benchmark. The result is not based on measuring a made-up user scenario but the detailed performance factors of all available X library directives.

`x11perf` was used for benchmarking, which comes with the XFree86 distribution and tests all existing X drawing directives as listed in Appendix B. Graphical tests like drawing filled rectangles are always done with sizes from 1x1 pixel, 10x10, 100x100 and 500x500 pixels.

It is definitely very inefficient to draw single dots, rectangles or circles with a diameter of 1 pixel, but these things also happen in real life by badly written applications, or eg. high zoom-out factors of CAD drawings.

These benchmark results cannot be directly mapped and compared with a normal user session, but it is easy to use these results with any application. For instance, a CAD program will be more likely to use `line` and `rectangle` operations than `pixmap` operations, so these values weigh more in this case. It is open to the user to weigh according to the needed application.

4.2 Test Setup

One full run of the benchmark tests every directive for 5 seconds and repeats every directive test for 5 times. Nevertheless, one complete run took 6 to 10 hours. Three computers were connected via a dedicated 100 Mbps line, *srvray* as the server machine, *thray* for throtteling (traffic shaping) and *clray* as the client machine (see Figure 4.1).

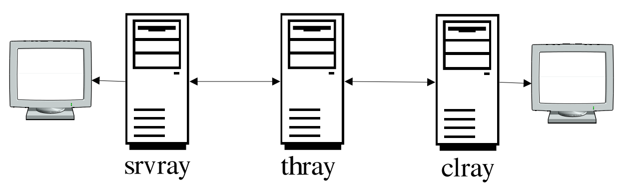

Figure 4.1: Benchmark Setup with the Traffic Shaping Computer in the Middle

All three are equipped with an AMD Athlon 1.3 GHz with 768 MB of RAM, where the two outer computers, *srvray* and *clray*, use a Matrox G-450 graphics card for their output. All tests normally did affect more of *clray's* CPU load because of the X-server displaying the draw primitives on the local display. For the VNC test, CPU power was also needed on the server-side because it had to compress the screen contents before sending them.

Native X to Local Machine

This test is the best case scenario, which just shows ideal values for the benchmark, just testing the Matrox G-450 and CPU without including any network bottlenecks (neither throughput nor latency). Still, in the context of thin-clients this test makes no sense at all, since remote displaying is the main feature of thin-client computing.

Native X to Remote Machine

Here we run the benchmark from *srvray*, displaying everything on *clray*. The network gets a bottleneck only on tests like drawing pixmaps. *srvray's* CPU was only busy at about 10%, whereas *clray's* CPU power was nearly 100% because of the X-Server outputting on the graphics card.

XNest to Remote Machine

Since X-Ray is based on the code of XNest, we run XNest on *srvray* and send the display to *clray*. The benchmark is also run from *srvray*. We recognized the same characteristics for CPU loads, but *srvray's* CPU was minimally more loaded than in the Native X test, because of some execution overhead in XNest.

X-Ray to Remote Machine in Fulldraw Mode

X-Ray was started in `fulldraw` mode on *srvray*, with its *server-side* output to *srvray's* local graphics card. Then the *client-side* was connected to *clray* and the benchmark was started on *srvray*. Both CPUs where highly loaded and, since X always queues multiple X commands to bigger network packets before sending them out, the two displays were updated very infrequently. After sending a block of commands to *srvray*, X-Ray had to wait for *srvray's* acknowledge, then it started to send the same block to *clray*. During that time, *srvray* was waiting for the acknowledge of *clray* and so on.

X-Ray to Remote Machine in Clientdraw Mode

X-Ray was started in `clientdraw` mode on *srvray*, with its *server-side* output to *srvray's* local graphics card. Then the *client-side* was connected to *clray* and the benchmark was started on *srvray*. In `clientdraw` mode, all graphical output to *srvray* is discarded, *srvray* only has to do such things as window handling, and other miscellaneous operations (see Appendix B.2). So for graphical output, *srvray's* CPU load was similar to running XNest, but for the window handling and miscellaneous operations, *srvray's* CPU load and blocking behavior was equally high as for X-Ray in fulldraw mode.

VNC to Remote Machine

The X-based version of VNC was started on *srvray* and a connection was done from *clray*, so all output is directed to *clray*. The benchmark was run from *srvray*. Here it is important to notice, that VNC consumes all CPU power on both *srvray* and *clray* for compressing pixmaps. Also the pull concept for screen updates misleads the benchmark. This means, that the VNC client only requests the next screen update when it has fully received and processed the actual one. So with a very high CPU load or the network load on *clray*, not all display changes where shown on the client side and the benchmark is able to draw to the server-side VNC framebuffer without any latency. Still, this also means that not all screen changes are really shown on the client-side and this leads to a loss in display accuracy. It also leads to a loss of interactiveness even on a fast network because of the roundtrip time for requesting the next screen update.

4.3 Benchmark Results

All following graphs show the relative performance of the different candidates, where the best candidate has a 100% bar height. So the Y-Axis shows the percentage to the best candidate. The basis for the quality is given by the number of operations per second gained from `x11perf` when run with the different X directives as listed in Appendix B. Most of the following graphs are generated from cumulated benchmark results like all rectangle operations in different sizes from 100x100, 300x300 up to 500x500 pixels.

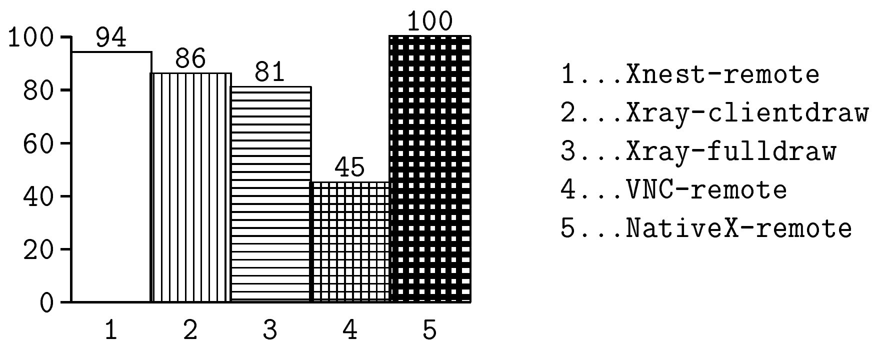

Figure 4.2: Comparing 100 Mbps Benchmarks Containing All Keywords

Figure 4.2 shows an overall benchmark on a 100 Mbps network, cumulating all possible `x11perf` benchmarks. We see the best result for native X directly connecting to *clray*. *X-Ray* in `clientdraw` mode is slightly slower than the equally designed *XNest*, since it introduces a second branch for the second display. *VNC* lies far behind at half the possible speed of native X.

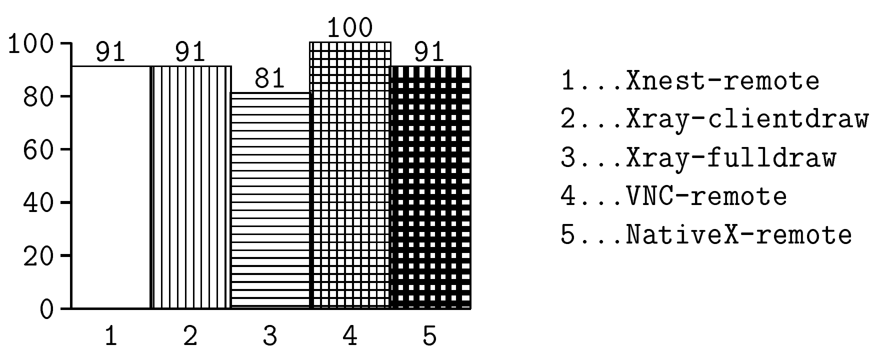

Figure 4.3: Comparing 100 Mbps Benchmarks Containing Keywords: "1x1 1-pixel dot"

To see the big advantages of *VNC*, refer to Figure 4.3, where we combined the benchmarks of single dots, 1 pixel diameter circles and 1x1 rectangles. This all leads to an enormous message overhead in the X protocol, but *VNC* handles it without problems because of its `pull` approach and framebuffer compression.

As soon as we look at larger rectangles, lines, circles and image operations with a size of 100x100, 300x300 or 500x500, *VNC* is highly outperformed by all X based approaches (see Figure 4.4). We also see, that *X-Ray* in `fulldraw` mode

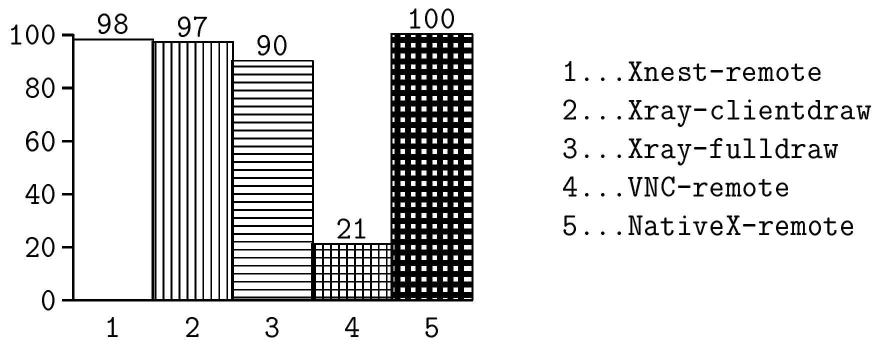

Figure 4.4: Comparing 100 Mbps Benchmarks Containing Keywords: "500x500 500-pixel 300x300 300-pixel 100x100 100-pixel"

is the slowest of all X based approaches, because it has to draw doubly.

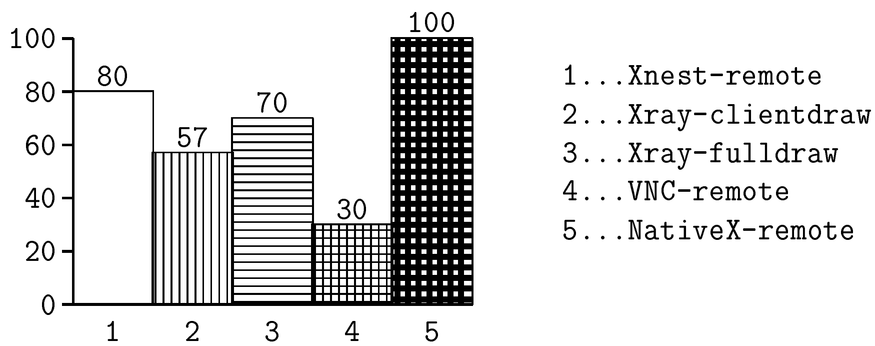

Figure 4.5: Comparing 100 Mbps Benchmarks Containing Keywords: "noop atom pointer prop context subwindow unmap via move resize circulate"

In Figure 4.5 we compare only administrative operations like creating, mapping, moving, resizing and destroying windows, including circulating subwindows and diverse others like No-Operation, pointer operations and changing graphics context. *VNC* performs very badly again, but compared to native X, all others also suffer from severe overhead by sending twice the output to the two attached X-Servers on the *server-side* and *client-side*. In this case, *X-Ray* in both `fulldraw` and `clientdraw` mode has to send all commands to both their connected computers. Still, `fulldraw` is slightly better in this scenario, since it can query some properties from the *server-side* display directly, which normally is on the same computer where *X-Ray* is started. So it has not to wait for any acknowledge arriving via the network.

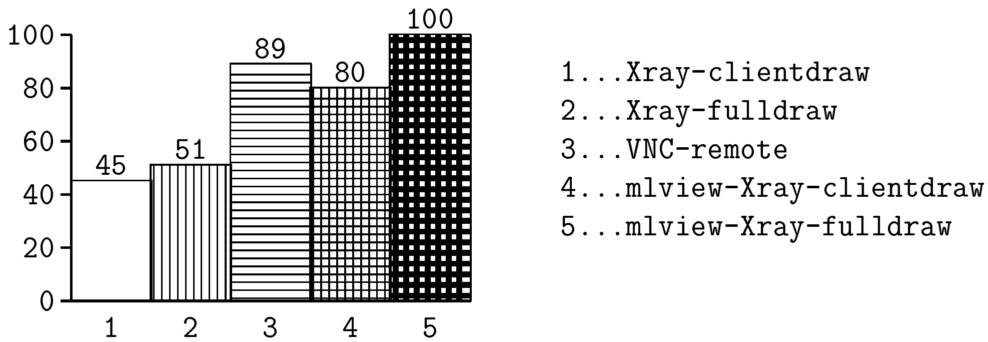

Figure 4.6: Comparing 10 Mbps Benchmarks With and Without X Compression Containing All Keywords

Figure 4.6 shows the overall benchmark on a 10 Mbps network. *X-Ray* – substitutional for all X based approaches – is outperformed by *VNC*. This is mainly because the network is now the bottleneck and *VNC* has enough CPU power on both sides to (de-)compress the sent framebuffer changes. Further, *VNC* now requests fewer display updates because of the `pull` approach. The *VNC* client queries the *VNC* server only when it fully received and painted the actual screen update. This leads to an extremely higher performance, since the *VNC* server runs on *server-side* and paints all output only to a virtual memory framebuffer without any synchronization with the *VNC* client. But this performance gain is highly misleading, since a user doesn't see screen changes at a constantly high frame-rate and hereby it leads to a loss in interactiveness and even a loss of information (eg. a movie is still only shown with 10 fps instead of 24 fps).

X-Ray in connection with *ML-View* offers a great solution, because *ML-View* compresses all sent pixmaps and caches multiple but identical messages (for more details on X compression refer to Section 2.3.3). Now, *X-Ray* in `fulldraw` mode is the fastest candidate. This makes sense, because every X command is queued by the X library internally. The X library tries to send the X command to the connected X-Server, which takes its time over a network. So the next X command is sent only when the last one is acknowledged by the X-Server, depending on network throughput and latency. In our case, the *ML-View* server is also run on the local machine, so the acknowledgment arrives in virtually no time. This is valid for both `fulldraw` and `clientdraw` mode. But `fulldraw` mode is extremely faster in querying X-Server states or getting images, since there is a fully valid instance running on the local machine.

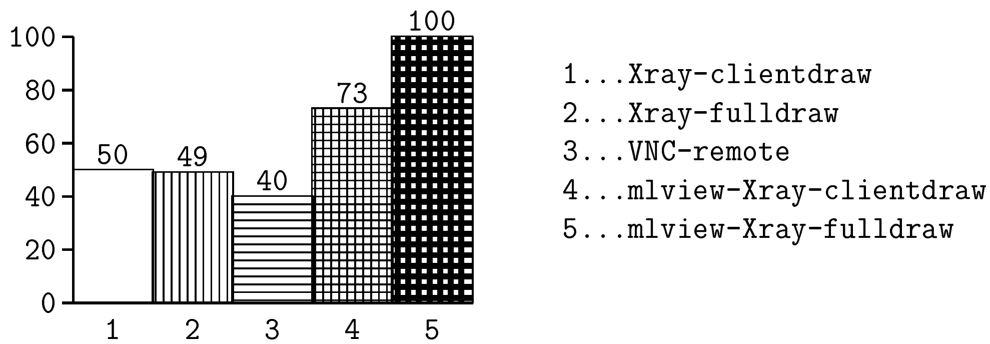

Figure 4.7: Comparing 10 Mbps Benchmarks With and Without X Compression Containing Keywords: "500x500 500-pixel 300x300 300-pixel 100x100 100-pixel"

When we are looking only at larger rectangles, lines, circles and image operations, VNC is also behind *X-Ray* without any compression, since the X message overhead is diminishing in relation to the graphical work to be done on the *client-side* (see Figure 4.7).

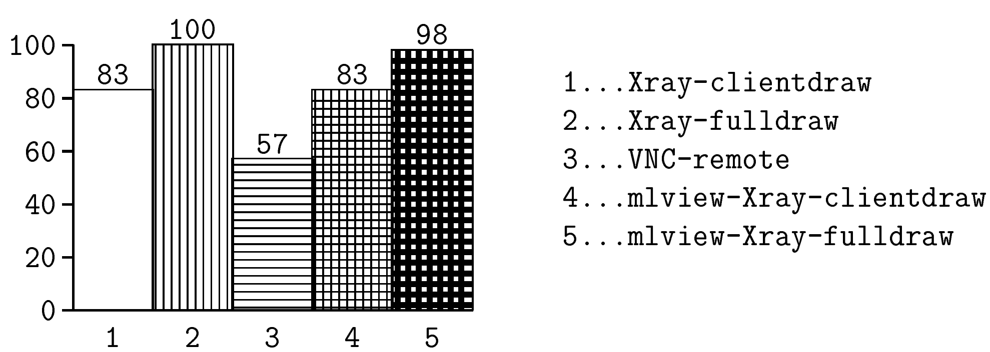

Figure 4.8: Comparing 10 Mbps Benchmarks With and Without X Compression Containing Keywords: "noop atom pointer prop context subwindow unmap via move resize circulate"

Concentrating on administrative operations, *ML-View* does not improve performance, but only increases *server-side* CPU load (see Figure 4.8). *VNC* is very slow and *X-Ray* in `fulldraw` mode again gains performance because of locality.

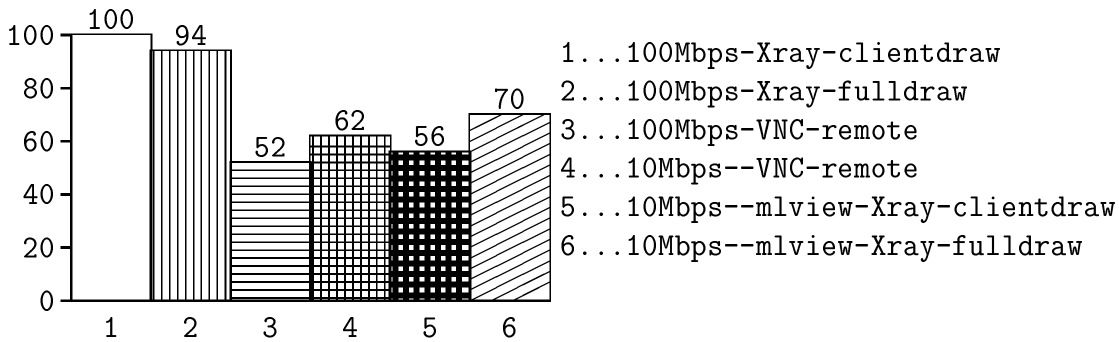

Figure 4.9: Comparing 10 Mbps and 100 Mbps Benchmarks Containing All Keywords

Now we want to compare the 100 Mbps network and the 10 Mbps network. On the 100 Mbps network, we take the *X-Ray* measurements without *ML-View* compression, since the network is definitely not the bottleneck. Figure 4.9 shows that the ten times faster network only doubles the overall performance. This is because of local CPU and graphics card load. *VNC* seems to perform better on the 10 Mbps network, but again, this is only an artifact due to the `pull` approach.

Concluding the latter measurements, *X-Ray* session management offers a much better solution than *VNC* in terms of graphics performance, including user interactiveness and guaranteed displaying of screen changes. This is valid especially on a local area network (LAN) with 100 Mbps or 10 Mbps. On slower lines this is only dependent on the *ML-View*[23] package. This package is primarily built to compress normal X sessions. It is under intense development and there will be optimizations for session management. So there have to follow more in-depth measurements of various X sessions using *ML-View* over connections as low as ISDN and up to 2Mbps. These measurements are not directly in the scope of this work, but (assuming a different, more portable benchmark suite) they would allow direct comparison between X and other low-speed line protocols like Citrix Metaframe.

Chapter 5

Sound Forwarding

A modern desktop environment not only offers a graphical user interface (GUI) with all bells and whistles like animated toolbars, senseless jumping paperclips and multiple themes with backgrounds, cursors and so on, but it also offers user feedback with played sound samples for eg. opening windows, error messages, shutting down programs and more. It is also very common to listen to music from the Internet or from the local hard disk. This may also include video, Internet radio or just streamed MP3 or .wav files.

5.1 Audio Access under Linux

Digital sound quality is defined by a sampling rate (44.1 kHz is CD quality), the granularity of each sample taken (8 bit, 16 bit and in professional studios up to 24 bit) and of course the number of channels, normally mono or stereo.

Modern soundcards offer a variety of different modes starting from CD quality (44.1 kHz, 16 bit, stereo) down to very low qualities like 11 kHz, 8 bit, mono. They also have built-in mixer features like volume control, balance, bass, treble, digital echo effects and more. All these modes have to be set in a soundcard-specific and proprietary way defined by the producer. Also the low-level sound data transfer of the raw sound samples is rather complicated, since this has to happen via interrupt-driven DMA buffers.

Linux combines the support for many of these different cards into one generalized driver and allows applications to access a generic sound device via two well-defined character devices `/dev/dsp` and `/dev/mixer`. Like all Linux devices, these two understand special IOCTLs to set the different sound modes (sampling rate, bit rate, channels) and mixer settings like the volume.

An application now only has to open `/dev/dsp` and `/dev/mixer` and sets the wanted sound mode and volume via the according IOCTLs. Then it just writes the raw sound samples into `/dev/dsp` and the kernel driver takes care of the rest. As long as this application has opened and hereby locked `/dev/dsp` and `/dev/mixer`, no other application can access it.

Modern sound cards all support CD quality and all sound applications try to play music in this quality. To get an idea of needed bandwidth for 44.1 kHz, 16 bit in stereo, we can easily calculate $44100 * 2 * 2$ which results in 172 KByte/sec just for the raw data samples, excluding any IOCTLs sent. According to this, if we want to send sound over the network, we need at least 1.3 Mbps just for the raw sound samples in CD quality of the available network bandwidth!

5.2 Requirements for Multi-User Audio Session Management

Since we want to offer these audio features in our session management system, we first have to overcome some severe obstacles. All problems stem from one single fact: All applications run on the server-side but are displayed on a (roaming) client-side. Also audio output is generated on the server-side, but has to be forwarded to the connected thin-client. During a session, the user may change his working thin-client, and hereby require to redirect the output. This has to happen without user intervention and even without bothering and changing the already running sound applications.

So we can compile a list of all needed features:

- Multiple users can share the same server-side sound device, which is somehow multiplexed to their connected thin-clients.

- Even when there is no connected thin-client, the sound session has to be kept alive somehow. On reconnect, the old state of the device (sampling frequency, mono/stereo, data format) has to be reset on the newly connected thin-client.

- Sound applications have to believe to access a regular `/dev/dsp` and `/dev/mixer`.

But there are more desired features and open questions, which will be partially supported by the following approach:

- Buffers on server and client have to be large enough to cope with network congestions.

- Buffers on server and client have to be small enough to not lose too much data on dis-/reconnect, or the server has to communicate with the client about how much data is really processed already, so on a reconnect from another client, the server has to resend the not yet processed data from its internal buffers.

- The server and client should be capable of compressing the data stream in real-time, adapting to various network speeds. This also includes feedback messages from the client so the quality can be reduced by the server to accommodate with changing network speeds or even client-side output quality (why send 44.1 kHz 16bit stereo, when the client is only capable of 11 kHz with 8 bit mono?)

  This compression could happen with the MPEG-1 Layer 3 (aka MP3) audio compression algorithm, which is lossy, but it benefits from humanoid acoustic hearing disabilities. This heavily increases server-side CPU usage for compression and client-side CPU usage for decompression, so to support multiple users on the server, maybe hardware compression should be used (eg. available on specialized Internet telephony multi-channel PCI cards).

- When a client is not connected, should the sound be halted or played without output?
- Sound recording is needed for eg. Internet telephony from a user's workplace. This means an extra audio stream in the reverse direction from the (possibly roaming) client to the server.

The following approach will describe a solution, which implements the reconnect feature, allows multiple users on one server, and blocks running applications on disconnect to a certain degree. It is not capable of sound recording and (adaptive) compression, but these features could be easily added, since the necessary hooks in the source code are available.

5.3 Non-satisfying Sound Forwarding Alternatives

There are numerous alternatives for redirecting sound via a network in a client/server manner. But all of the following programs mostly lack the mostly desired features in a session management environment, which are basically the reconnect and keep-alive of running sessions.

5.3.1 NAS – The Network Audio System

NAS[37] wraps a daemon over the soundcard's `/dev/dsp` device and listens on port 8000 for remotely incoming streams, which are then played on the local `/dev/dsp` device. It works operating system independent and comes with a variety of application plug-ins like the free MP3-player `xmms`, where the output host is configurable.

Unfortunately it is not usable in our context, since the server-side application should not be bothered with reconfiguration for each reconnect from a different thin-client.

5.3.2 EsounD – Enlightened Sound Daemon

EsounD[38] works in a very similar way. Like NAS, it offers network independent simultaneous sound playback, but it cannot reconfigure an application on the fly to automatically redirect the playback to another thin-client's `/dev/dsp` device.

5.3.3 Rplay – Remote Play

Rplay[39] also allows cross-network sound playing and adds an interesting caching feature. It can access sound bits from a local directory where they are numbered and a remote application only tells the `rplayd` on the client to play a certain sound bit by its referencing number. `rplayd` is able to cache sound bits from other hosts' `rplayd` and plays sound files of various formats like AU, AIFF, WAV, VOC, UB, UL, G.721 4-bit, G.723 3-bit, G.723 5-bit, or GSM.

The biggest disqualification issue is that `rplayd` doesn't emulate a `/dev/dsp` device but there is only a client program to be called from other applications to play sounds. So there are almost no applications like MP3-players or desktop environments, which would support `rplayd`.

5.3.4 afwd – Audio Forwarder

At a first glance, afwd[40] seemed to be the right approach for our problem: a kernel module emulates valid `/dev/dsp` and `/dev/audio` devices to server-side sound applications, but passes the sound data to a server-side running server. Now any client can connect to the server via the network, and the sound data is forwarded to the client, which sends the sound data directly to the real client's `/dev/dsp` device.

The project is in a very early state, so even the server name is still hard-coded in the sources and – which is more of a problem – it only works for one user per faked `/dev/dsp` device. The first sound application blocks the fake `/dev/dsp` and no other application can open it until the first application releases the device.

The naive solution would be to create multiple `/dev/dsp{UID}` devices and an environment variable like $AUDIODEVICE to start all sound applications pointing to that device. But this is not easily possible because of some limiting factors of the Linux kernel, which are very few possible free major and minor numbers for devices to serve a large number of simultaneous users.

5.3.5 dsproxy

Finally, dsproxy[41] provided most of the needed features. Like *afwd*, it also sets up a kernel module which fakes valid `/dev/dsp` and `/dev/mixer` devices, but it internally copes with multiple users at a time.

But unfortunately the server/client software in the dsproxy version 0.2.17 was very rudimentary, so it just offered the same functionality as *afwd*, which means no multiple users and no reconnect without restarting of the server and the client (and hereby losing the connected sound application because `write()` returns the broken pipe error `-EPIPE`). Since no reconnect is possible anyway, it does not save and restore the state of the sound device over the duration of a full desktop session.

Chapter 6

A Sound Forwarding System for Multiple Thin-Clients

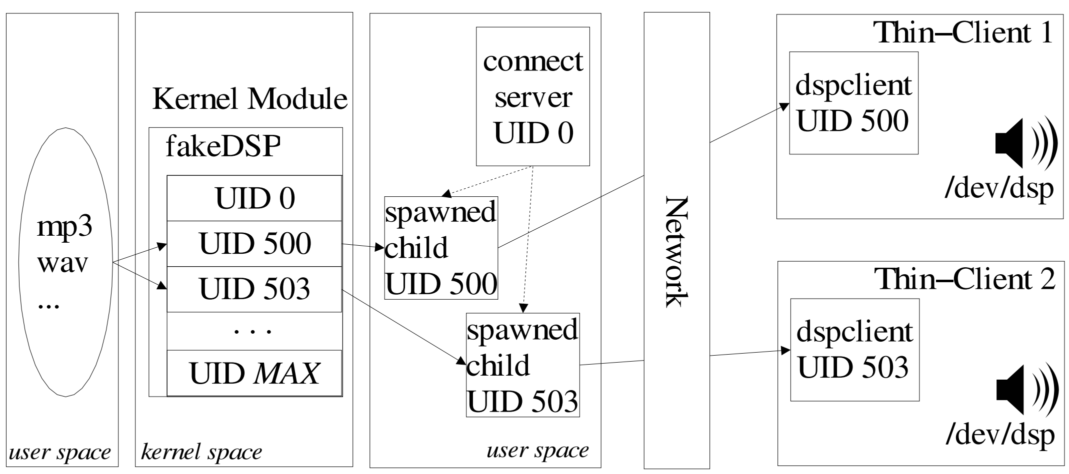

Figure 6.1: Overview of the *dsproxy* system

The *dsproxy* system was extended to meet the requirements of session management and multiple users starting sound applications and hereby sharing one `/dev/dsp` device on the same server.

6.1 The Extended *dsproxy* Server

To behave like a real daemon, the server was changed to run with UID 0, running with root rights. When a client connects to the server on port 9138, the server forks itself and the client sends its own UID. Using that, the spawned child server changes its UID to the received userID by using `setuid()` and goes into a big loop, waiting for any data sent from sound applications, which has to be forwarded to the connected client.

6.2 The Extended *dsproxy* Client

On startup, the client immediately connects to the server and sends its UID. Then it goes into an endless loop, waiting for data from the server. The data (which may contain IOCTLs for the *dsp* or *mixer* device, but also raw sound data) is redirected to the real `/dev/dsp` and `/dev/mixer` devices on the local client system.

The client reacts on the signals SIG_INT and SIG_KILL by sending a final "BYE" packet to the server and then exits cleanly.

6.3 The Extended *dsproxy* Kernel Module

The kernel module works on two emulation devices `/dev/dsp` and `/dev/mixer` with the major number 121 and the minor numbers 2 and 3. So first one has to change the files in `/dev` as follows:

```
[root@arjuna ram]# cd /dev
[root@arjuna ram]# mv dsp dsp_x
[root@arjuna ram]# mknod dsp c 121 2
[root@arjuna ram]# mv mixer mixer_x
[root@arjuna ram]# mknod mixer c 121 3
```

If a spawned server tries to open "its" instance for a certain UID for the first time, a new `struct dsproxy_s` is created (see Appendix A.3). If a client just reconnects, the kernel module reuses the structure and pushes some IOCTLs to the user's device FIFO. These IOCTLs will set the old *dsp* and *mixer* state, which is still stored in the user specific structure. Against all rules, we have to push the IOCTLs in front of the FIFO, so this is the first data the new client receives. So the server immediately reads the initializations and sends them to the newly connected client, which – now initialized to the correct values – may immediately continue playing pending data.

6.4 Arbitrary Sound Applications

A sound application does not care which major and minor numbers the sound devices have; it just connects to `/dev/dsp` and `/dev/mixer` and expects a correct behavior.

Still, a connect of an application will only be successful, when a server and a client is already connected to the faked `/dev/dsp`, since many audio applications do some tests on the sound device's capabilities like maximum frequency or bit rate by sending IOCTLs for requesting information.

Also, different applications react differently on losing a client-side connection. When a sound application has opened the faked `/dev/dsp` and is writing data to it, it uses the normal `write()` on a file descriptor pointing to `/dev/dsp`. This `write()` returns the number of written bytes. If the client disconnects during a write phase, the kernel module just returns zero bytes written, and the error code `-EAGAIN`, so the sound application believes that the `/dev/dsp` data buffer is full.

Some applications will wait and continue when a write is successful (eg. `cat mysound.wav > /dev/dsp`), some more sophisticated applications have internal timers, so they just stop playing if there is a too high time difference of buffered, sent and pending data (eg. `xmms`). The third reaction is based on a "real-time" approach, so these applications just keep dumping data and ignoring every feedback about really written data (eg. `gqmpeg`, `mpg123`). This also means, the playing speed increases rapidly since there is no slow down, because the `write()` call returns immediately instead of relatively conforming to the playing speed.

`xmms` is one of the mainly used MP3 players and offers a high modularity, even on the output plug-ins. So it should be quite easy to write a specialized plugin which does not stop playing, but discards the output in a timely manner or blocks until a client-side reconnect happens.

The only MP3 player that perfectly worked out of the box with this blocking approach was the KDE1 MP3 player `kmp3`, which wrote out this error message for a million of times:

```
Ouch ... error while writing audio data: : Resource temporarily unavailable
```

But after the client-side reconnect, it continued to play exactly at the right position. So a quick work around is to start the KDE player with `kmp3 2>/dev/null` or to comment out the error output in the source code and recompile it.

Chapter 7

Open Questions

7.1 Video Forwarding

In the future, thin-clients will have to be capable of playing videos, since it is very forseeable, that TVs will be replaced by normal computers. To do well with videos, some optimizations have to be done to reduce the network traffic.

Normally, a video application will decompress a video stream (eg. MPEG-4) on the server-side, paint its output into the application window with play, pause and stop buttons. Then these pixels are transferred over the network, which logically happens uncompressed with high overhead (see Figure 7.1).

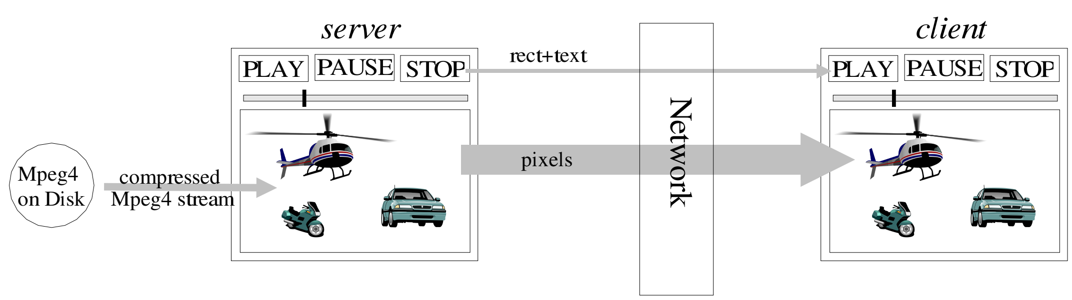

Figure 7.1: Uncompressed Video Transfer

The only satisfying solution would be a splitted player software, which locally draws a window with the play, pause and stop buttons and sends the compressed video stream to the connected thin-client over a parallel connection. The stream is decompressed on the client-side and exactly drawn into the predefined window coordinates of the player software (see Figure 7.2).

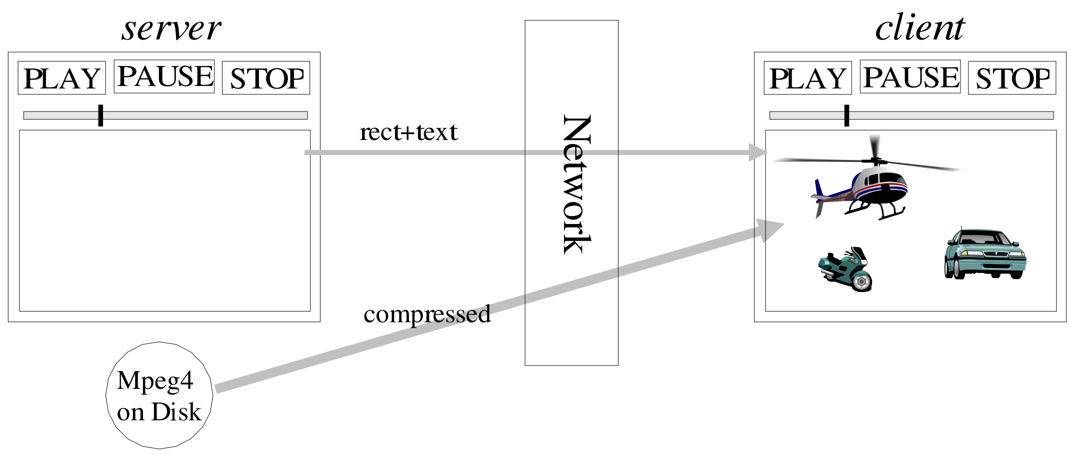

Figure 7.2: Compressed Video Transfer in a Second Stream

All this could be solved with the open-source software *xmps*[42]. `xmps` takes advantage of various input/output plug-ins and is capable of various video encoding protocols like MPEG-1, MPEG-2, FLI, AVI, DivX, QuickTime and more.

Instead of displaying the video on the server side, a display plugin has to forward the compressed video stream to the thin-client, where a decompression server is waiting for input. This decompression server receives exact information about the window position of the player, so the uncompressed video can be mapped in the exact blank spot. By communicating with the player, it is also easy to move the window or even show the decompressed video in full screen mode.

It would also be possible to take advantage of the X-Server extension *DRI* (Direct Rendering Interface) to directly access the graphics card and write the pixels into the graphics memory.

7.2 Optimized X Compression and Session Management

We already stated a viable solution in connection with ML-View[23], a native X-server and the XNest-based *X-Ray*. But to present an optimal X-based solution, we have to melt the session management features directly in every hardware-optimized X server, so no extra layer and no visible server-side connection is necessary. Also the compression features of ML-View for pixmaps and reordering of X commands by their importance (with respect to network latency) should be directly included in the XFree86 code. Finally, adaptive compression for sound forwarding would increase acceptance because of acoustic user feedback also on slow networks.

Chapter 8

Conclusion

This diploma thesis shows the necessity and feasibility of features like multimedia session management and should help to extend future X protocol versions to reflect the new needs of centralized thin-client computing. If we believe in the research of well-known institutes like the Gartner Group[2], the modern thin-client approach will be very successful in the next years, mostly because of reducing the *total cost of ownership* (TCO).

Even in this early stage of these "working prototypes" for proof of concept, connecting *X-Ray* with *ML-View* X compression and sound forwarding offer a free alternative for all companies, schools or universities not willing to administer hundreds of client-side computers, but prefer a controllable server-side configuration approach.

Besides, this will also increase user efficiency because of absolutely silent thin-client computers with no hard disks or CPU fans. These were the key features I wanted to have available at home: UNIX-based, open-source, silent and fast – achieving that, was the main reason why I chose this topic for my diploma thesis...

Bibliography

[1] A. Tanenbaum, A. Woodhull, "Operating Systems – Design and Implementation", second edition, 1997 Prentice Hall International, Inc. ISBN 0-13-630195-9

[2] The Gartner Group

http://www.gartnergroup.com

[3] VNC – Virtual Network Computing, AT&T Laboratories Cambridge

http://www.uk.research.att.com/vnc/

[4] T. Richardson, Q. Stafford-Fraser, K. R. Wood, A. Hopper, "Virtual Network Computing", IEEE Internet Computing, Vol.2 No.1, Jan/Feb 1998 pp33-38

[5] T. Richardson, K. R. Wood, RFB – Remote Framebuffer Protocol for VNC, "The RFB Protocol", January 1998

http://www.uk.research.att.com/vnc/protocol.html

[6] Microsoft Corp, Technical White Paper, Windows NT Server, Terminal Server Edition, "Comparing Terminal Server and UNIX Application Deployment Solutions", Redmond, WA, 1999

[7] Microsoft Corp, Technical White Paper, Windows NT Server, "Terminal Server Edition, version 4.0: An Architectural Overview", Redmond, WA, 1998

[8] Mark Russinovich, Windows NT Magazine, Focus, "Inside Microsoft Terminal Server", July 1998

`http://www.winntmag.com/Articles/Index.cfm?IssueID=54&ArticleID=3594`

[9] Citrix, "ICA Technology Brief", March 1996

`http://www.kios.de/datenblaetter/ica/icatech.htm`

[10] ITU T.120 Protocol with Upper and Lower Level Layers, International Multimedia Telecommunications Consortium

`http://www.imtc.org/t120.htm`

[11] SCO Inc., Tarantella White Paper, "Tarantella Web-Enabling Software: One World, One Network, One Answer", May 2001

[12] SCO Inc., Tarantella White Paper, "Tarantella Enterprise 3 Technical Overview", May 2001

`http://www.tarantella.com/whitepapers/overview/`

[13] Sun Microsystems Inc., "SunRay 1 Enterprise Appliance Overview and Technical Brief", August 1999

[14] XINERAMA Extension of XFree86 Release 6 Version 4.0

`http://www.linuxhq.com/ldp/howto/Xinerama-HOWTO.html`

[15] X Consortium for the X Window system

`http://www.x.org`

[16] Introduction to the X Window System Protocol

`http://www.x.org/about_x.htm`

[17] The XFree86 Project, Free X Window System

`http://www.xfree86.org`

[18] Hummingbird, eXceed

`http://www.hummingbird.com`

[19] X_i Graphics Inc., Accelerated X

`http://www.xig.com`

[20] James Gettys, Robert W. Scheifler, “Xlib - C Language X Interface Reference”, X Consortium Standard, X Version 11, Release 6.4

[21] Digital Equipment Corp, Developers Manual, “Writing a Graphics Device Driver and DDX for the Digital Unix X Server”, June 1997

`http://www.tru64unix.compaq.com/docs`

[22] H. Schulzrinne, S. Casner, R. Frederick,and V. Jacobson, RFC 1889: “RTP: A Transport Protocol for Real-Time Applications”, January 1996

[23] ML-View by Medialogic s.r.l., Via G. Giovannoni, 76 - 00128 - Rome (Italy), Tel. +39 06 50797484

`http://www.medialogic.it`

[24] Differential X Protocol Compressor (DXCP)

`http://vigor.nu/dxpc`

[25] S. Casner, V. Jacobson, RFC 2508: “Compressing IP/UDP/RTP Headers for Low-Speed Serial Links”, February 1999

[26] K Desktop Environment

http://www.kde.org

[27] Carl A. Waldspurger, "Lottery and Stride Scheduling: Flexible Proportional-Share Resource Management", Ph.D. Thesis, September 1995

http://citeseer.nj.nec.com/waldspurger95lottery.html

[28] Jason Nieh, S. Jae Yang, Naomi Novik, "A Comparison of Thin-Client Computing Architectures", Technical Report CUCS-022-00, Network Computing Laboratory, Columbia University, November 2000

http://www.ncl.cs.columbia.edu/publications/cucs-022-00.pdf

[29] Jason Nieh, S. Jae Yang, "Measuring the Multimedia Performance of Server-Based Computing", Columbia University and PC Magazine Labs, 2000

http://citeseer.nj.nec.com/311704.html

[30] Brian K. Schmidt, Monica S. Lan, J. Duane Northcutt, Stanford University and Sun Microsystems Laboratories, "The interactive performance of SLIM: a stateless, thin-client architecture", 17th ACM Symposium on Operating Systems Principles (SOSP'99), Published as Operating Systems Review, 34(5):32-47, December 1999

[31] Steven R. Balmer, Cynthia E. Irvine, "Analysis of Terminal Server Architectures for Thin Clients in a High Assurance Network",Department of Computer Science, Naval Postgraduate School, 2000

http://citeseer.nj.nec.com/329988.html

[32] John Danskin, Pat Hanrahan, "Profiling the X Protocol", Proceedings of the 1994 conference on Measurement and modeling of computer systems May 16 - 20, 1994, Nashville, TN USA, May 1994

http://graphics.stanford.edu/papers/profiling/

[33] Thomas Gutekunst, Daniel Bauer, Germano Caronni, Hasan, Bernhard Plattner, “A Distributed and Policy-Free General-Purpose Shared Window System”, published in IEEE/ACM Transactions on Networking, February 1995

[34] Jachim Bleser, Edmund Lang, “Balkendiagramme in LaTeX Dokumenten”, TH Darmstadt, Hochschulrechenzentrum

[35] Werner Almesberger, “Linux Traffic Control – Implementation Overview”, EPFL ICA, November 1998

`http://diffserv.sourceforge.net/`

[36] Michael Kropfberger, X-Ray Sources and Documentation, August 2001,

`http://www.kropfberger.n3.net/xray.html`

[37] NAS – The Network Audio System

`http://radscan.com/nas.html`

[38] EsounD – Enlightened Sound Daemon

`http://www.tux.org/~ricdude/EsounD.html`

[39] Rplay – Remote Play

`http://rplay.doit.org`

[40] Pete Wyckoff, afwd – Audio Forwarder

`http://www.osc.edu/~pw/afwd.html`

[41] Tony Bybell, dsproxy

`http://www.linux-workshop.com/bybell/dsproxy`

[42] Damien Chavarria, xmps - X Movie Player System

`http://xmps.sourceforge.net`

Appendix A

Data Structures

A.1 GC: xc/lib/X11/Xlib.h

```
###FILE: xc/lib/X11/Xlib.h
/*
 * Data structure for setting graphics context.
 */
typedef struct {
        int function;           /* logical operation */
        unsigned long plane_mask;/* plane mask */
        unsigned long foreground;/* foreground pixel */
        unsigned long background;/* background pixel */
        int line_width;         /* line width */
        int line_style;         /* LineSolid, LineOnOffDash, LineDoubleDash */
        int cap_style;          /* CapNotLast, CapButt,
                                   CapRound, CapProjecting */
        int join_style;         /* JoinMiter, JoinRound, JoinBevel */
        int fill_style;         /* FillSolid, FillTiled,
                                   FillStippled, FillOpaeueStippled */
        int fill_rule;          /* EvenOddRule, WindingRule */
        int arc_mode;           /* ArcChord, ArcPieSlice */
        Pixmap tile;            /* tile pixmap for tiling operations */
        Pixmap stipple;         /* stipple 1 plane pixmap for stipping */
        int ts_x_origin;        /* offset for tile or stipple operations */
        int ts_y_origin;
        Font font;              /* default text font for text operations */
        int subwindow_mode;     /* ClipByChildren, IncludeInferiors */
        Bool graphics_exposures;/* boolean, should exposures be generated */
        int clip_x_origin;      /* origin for clipping */
        int clip_y_origin;
```

```
        Pixmap clip_mask;       /* bitmap clipping; other calls for rects */
        int dash_offset;        /* patterned/dashed line information */
        char dashes;
} XGCValues;

/*
 * Graphics context.  The contents of this structure are implementation
 * dependent.  A GC should be treated as opaque by application code.
 */

typedef struct _XGC
#ifdef XLIB_ILLEGAL_ACCESS
{
    XExtData *ext_data; /* hook for extension to hang data */
    GContext gid;       /* protocol ID for graphics context */
    /* there is more to this structure, but it is private to Xlib */
}
#endif
*GC;
```

A.2 ScreenRec: xc/programs/Xserver/include/scrnintstr.h

###FILE: xc/programs/Xserver/include/scrnintstr.h

```
typedef struct _Screen {
    int                 myNum;  /* index of this instance in Screens[] */
    ATOM                id;
    short               width, height;
    short               mmWidth, mmHeight;
    short               numDepths;
    unsigned char       rootDepth;
    DepthPtr            allowedDepths;
    unsigned long       rootVisual;
    unsigned long       defColormap;
    short               minInstalledCmaps, maxInstalledCmaps;
    char                backingStoreSupport, saveUnderSupport;
    unsigned long       whitePixel, blackPixel;
    unsigned long       rgf;    /* array of flags; she's -- HUNGARIAN */
    GCPtr               GCperDepth[MAXFORMATS+1];
                        /* next field is a stipple to use as default in
                           a GC.  we don't build default tiles of all depths
                           because they are likely to be of a color
```

```
                           different from the default fg pixel, so
                           we don't win anything by building
                           a standard one.
                        */
    PixmapPtr           PixmapPerDepth[1];
    pointer             devPrivate;
    short               numVisuals;
    VisualPtr           visuals;
    int                 WindowPrivateLen;
    unsigned            *WindowPrivateSizes;
    unsigned            totalWindowSize;
    int                 GCPrivateLen;
    unsigned            *GCPrivateSizes;
    unsigned            totalGCSize;

    /* Random screen procedures */

    CloseScreenProcPtr          CloseScreen;
    QueryBestSizeProcPtr        QueryBestSize;
    SaveScreenProcPtr           SaveScreen;
    GetImageProcPtr             GetImage;
    GetSpansProcPtr             GetSpans;
    PointerNonInterestBoxProcPtr PointerNonInterestBox;
    SourceValidateProcPtr       SourceValidate;

    /* Window Procedures */

    CreateWindowProcPtr         CreateWindow;
    DestroyWindowProcPtr        DestroyWindow;
    PositionWindowProcPtr       PositionWindow;
    ChangeWindowAttributesProcPtr ChangeWindowAttributes;
    RealizeWindowProcPtr        RealizeWindow;
    UnrealizeWindowProcPtr      UnrealizeWindow;
    ValidateTreeProcPtr         ValidateTree;
    PostValidateTreeProcPtr     PostValidateTree;
    WindowExposuresProcPtr      WindowExposures;
    PaintWindowBackgroundProcPtr PaintWindowBackground;
    PaintWindowBorderProcPtr    PaintWindowBorder;
    CopyWindowProcPtr           CopyWindow;
    ClearToBackgroundProcPtr    ClearToBackground;
    ClipNotifyProcPtr           ClipNotify;
    RestackWindowProcPtr        RestackWindow;

```

```
/* Pixmap procedures */

CreatePixmapProcPtr         CreatePixmap;
DestroyPixmapProcPtr        DestroyPixmap;

/* Backing store procedures */

SaveDoomedAreasProcPtr      SaveDoomedAreas;
RestoreAreasProcPtr         RestoreAreas;
ExposeCopyProcPtr           ExposeCopy;
TranslateBackingStoreProcPtr TranslateBackingStore;
ClearBackingStoreProcPtr    ClearBackingStore;
DrawGuaranteeProcPtr        DrawGuarantee;
/*
 * A read/write copy of the lower level backing store vector is needed now
 * that the functions can be wrapped.
 */
BSFuncRec                   BackingStoreFuncs;

/* Font procedures */

RealizeFontProcPtr          RealizeFont;
UnrealizeFontProcPtr        UnrealizeFont;

/* Cursor Procedures */

ConstrainCursorProcPtr      ConstrainCursor;
CursorLimitsProcPtr         CursorLimits;
DisplayCursorProcPtr        DisplayCursor;
RealizeCursorProcPtr        RealizeCursor;
UnrealizeCursorProcPtr      UnrealizeCursor;
RecolorCursorProcPtr        RecolorCursor;
SetCursorPositionProcPtr    SetCursorPosition;

/* GC procedures */

CreateGCProcPtr             CreateGC;

/* Colormap procedures */

CreateColormapProcPtr       CreateColormap;
DestroyColormapProcPtr      DestroyColormap;
InstallColormapProcPtr      InstallColormap;
```

```
    UninstallColormapProcPtr    UninstallColormap;
    ListInstalledColormapsProcPtr ListInstalledColormaps;
    StoreColorsProcPtr          StoreColors;
    ResolveColorProcPtr         ResolveColor;

    /* Region procedures */

#ifdef NEED_SCREEN_REGIONS
    RegionCreateProcPtr         RegionCreate;
    RegionInitProcPtr           RegionInit;
    RegionCopyProcPtr           RegionCopy;
    RegionDestroyProcPtr        RegionDestroy;
    RegionUninitProcPtr         RegionUninit;
    IntersectProcPtr            Intersect;
    UnionProcPtr                Union;
    SubtractProcPtr             Subtract;
    InverseProcPtr              Inverse;
    RegionResetProcPtr          RegionReset;
    TranslateRegionProcPtr      TranslateRegion;
    RectInProcPtr               RectIn;
    PointInRegionProcPtr        PointInRegion;
    RegionNotEmptyProcPtr       RegionNotEmpty;
    RegionBrokenProcPtr         RegionBroken;
    RegionBreakProcPtr          RegionBreak;
    RegionEmptyProcPtr          RegionEmpty;
    RegionExtentsProcPtr        RegionExtents;
    RegionAppendProcPtr         RegionAppend;
    RegionValidateProcPtr       RegionValidate;
#endif /* NEED_SCREEN_REGIONS */
    BitmapToRegionProcPtr       BitmapToRegion;
#ifdef NEED_SCREEN_REGIONS
    RectsToRegionProcPtr        RectsToRegion;
#endif /* NEED_SCREEN_REGIONS */
    SendGraphicsExposeProcPtr   SendGraphicsExpose;

    /* os layer procedures */

    ScreenBlockHandlerProcPtr   BlockHandler;
    ScreenWakeupHandlerProcPtr  WakeupHandler;

    pointer blockData;
    pointer wakeupData;

```

```
    /* anybody can get a piece of this array */
    DevUnion    *devPrivates;

    CreateScreenResourcesProcPtr CreateScreenResources;
    ModifyPixmapHeaderProcPtr   ModifyPixmapHeader;

    GetWindowPixmapProcPtr      GetWindowPixmap;
    SetWindowPixmapProcPtr      SetWindowPixmap;
    GetScreenPixmapProcPtr      GetScreenPixmap;
    SetScreenPixmapProcPtr      SetScreenPixmap;

    PixmapPtr pScratchPixmap;           /* scratch pixmap "pool" */

#ifdef PIXPRIV
    int                 PixmapPrivateLen;
    unsigned int                *PixmapPrivateSizes;
    unsigned int                totalPixmapSize;
#endif

    MarkWindowProcPtr           MarkWindow;
    MarkOverlappedWindowsProcPtr MarkOverlappedWindows;
    ChangeSaveUnderProcPtr      ChangeSaveUnder;
    PostChangeSaveUnderProcPtr  PostChangeSaveUnder;
    MoveWindowProcPtr           MoveWindow;
    ResizeWindowProcPtr         ResizeWindow;
    GetLayerWindowProcPtr       GetLayerWindow;
    HandleExposuresProcPtr      HandleExposures;
    ReparentWindowProcPtr       ReparentWindow;

#ifdef SHAPE
    SetShapeProcPtr             SetShape;
#endif /* SHAPE */

    ChangeBorderWidthProcPtr    ChangeBorderWidth;
    MarkUnrealizedWindowProcPtr MarkUnrealizedWindow;

} ScreenRec;

typedef struct _ScreenInfo {
    int         imageByteOrder;
    int         bitmapScanlineUnit;
    int         bitmapScanlinePad;
    int         bitmapBitOrder;
```

```
    int         numPixmapFormats;
    PixmapFormatRec
                formats[MAXFORMATS];
    int         arraySize;
    int         numScreens;
    ScreenPtr   screens[MAXSCREENS];
    int         numVideoScreens;
} ScreenInfo;

extern ScreenInfo screenInfo;
```

A.3 dsproxy_s

```
###FILE: dsproxy/dsproxy.h


struct dsproxy_s {
  struct dsproxy_s *next;
  uid_t uid;    /* uid of writer */
  int clients;
  int subdevice;

  struct semaphore sem, wr_sem, ioctl_sem, mixer_sem;
  struct semaphore forkread, forkwrite;
/* bracket r/w ops with these to make forked reads/writes/ioctls sequential! */

  int r_inuse;
  int w_inuse;

  /* Audio */
  int fragsize;
  int numfrags;
  int bufsiz;
  char frag_explicitly_set;
  char send_reset;

  int format;
  int freq;
  int channels, bytes_per_sample;

  /* buffers */
  char vmbuf[DSPROXY_FIFO_SIZE];
```

```
  /* this is for programs that don't look at the return value for write()
     as they should! */
  char faultbuf[DSPROXY_FAULTBUF_SIZ+DSPROXY_FRAG_SIZ];
  int offset, lowwater;

  int fifoget, fifoput, fifocount;
  int pcm_outstanding;

  /* statistics */
  unsigned long long bytes_processed;
  unsigned long long hdrskip;

  /* sleep */
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,4,0)
  wait_queue_head_t inq, outq;
#else
  struct wait_queue *inq, *outq;
#endif

  /* timing */
  struct timeval firstwrite, etime;

  /* mixer */
  int volume;
  int bass;
  int treble;
  int pcm;
  int ogain;

  int dirty;    /* set when there are dirty bits in the mixer */

  /* kernel */
  struct inode *inode;
};
```

Appendix B

x11perf Benchmark Details

B.1 Graphical Directives Commands

Test Name	Description
-dot	Dot.
-rect1	1x1 solid-filled rectangle.
-rect10	10x10 solid-filled rectangle.
-rect100	100x100 solid-filled rectangle.
-rect500	500x500 solid-filled rectangle.
-srect1	1x1 transparent stippled rectangle, 8x8 stipple pattern.
-srect10	10x10 transparent stippled rectangle, 8x8 stipple pattern.
-srect100	100x100 transparent stippled rectangle, 8x8 stipple pattern.
-srect500	500x500 transparent stippled rectangle, 8x8 stipple pattern.
-osrect1	1x1 opaque stippled rectangle, 8x8 stipple pattern.
-osrect10	10x10 opaque stippled rectangle, 8x8 stipple pattern.
-osrect100	100x100 opaque stippled rectangle, 8x8 stipple pattern.
-osrect500	500x500 opaque stippled rectangle, 8x8 stipple pattern.
-tilerect1	1x1 tiled rectangle, 4x4 tile pattern.
-tilerect10	10x10 tiled rectangle, 4x4 tile pattern.
-tilerect100	100x100 tiled rectangle, 4x4 tile pattern.
-tilerect500	500x500 tiled rectangle, 4x4 tile pattern.

Test Name	Description
-oddsrect1	1x1 transparent stippled rectangle, 17x15 stipple pattern.
-oddsrect10	10x10 transparent stippled rectangle, 17x15 stipple pattern.
-oddsrect100	100x100 transparent stippled rectangle, 17x15 stipple pattern.
-oddsrect500	500x500 transparent stippled rectangle, 17x15 stipple pattern.
-oddosrect1	1x1 opaque stippled rectangle, 17x15 stipple pattern.
-oddosrect10	10x10 opaque stippled rectangle, 17x15 stipple pattern.
-oddosrect100	100x100 opaque stippled rectangle, 17x15 stipple pattern.
-oddosrect500	500x500 opaque stippled rectangle, 17x15 stipple pattern.
-oddtilerect1	1x1 tiled rectangle, 17x15 tile pattern.
-oddtilerect10	10x10 tiled rectangle, 17x15 tile pattern.
-oddtilerect100	100x100 tiled rectangle, 17x15 tile pattern.
-oddtilerect500	500x500 tiled rectangle, 17x15 tile pattern.
-bigsrect1	1x1 stippled rectangle, 161x145 stipple pattern.
-bigsrect10	10x10 stippled rectangle, 161x145 stipple pattern.
-bigsrect100	100x100 stippled rectangle, 161x145 stipple pattern.
-bigsrect500	500x500 stippled rectangle, 161x145 stipple pattern.
-bigosrect1	1x1 opaque stippled rectangle, 161x145 stipple pattern.
-bigosrect10	10x10 opaque stippled rectangle, 161x145 stipple pattern.
-bigosrect100	100x100 opaque stippled rectangle, 161x145 stipple pattern.
-bigosrect500	500x500 opaque stippled rectangle, 161x145 stipple pattern.
-bigtilerect1	1x1 tiled rectangle, 161x145 tile pattern.
-bigtilerect10	10x10 tiled rectangle, 161x145 tile pattern.
-bigtilerect100	100x100 tiled rectangle, 161x145 tile pattern.
-bigtilerect500	500x500 tiled rectangle, 161x145 tile pattern.
-eschertilerect1	1x1 tiled rectangle, 215x208 tile pattern.
-eschertilerect10	10x10 tiled rectangle, 215x208 tile pattern.
-eschertilerect100	100x100 tiled rectangle, 215x208 tile pattern.
-eschertilerect500	500x500 tiled rectangle, 215x208 tile pattern.

Test Name	Description
-seg1	1-pixel thin line segment.
-seg10	10-pixel thin line segment.
-seg100	100-pixel thin line segment.
-seg500	500-pixel thin line segment.
-seg100c1	100-pixel thin line segment (1 obscuring rectangle).
-seg100c2	100-pixel thin line segment (2 obscuring rectangles).
-seg100c3	100-pixel thin line segment (3 obscuring rectangles).
-dseg10	10-pixel thin dashed segment (3 on, 2 off).
-dseg100	100-pixel thin dashed segment (3 on, 2 off).
-ddseg100	100-pixel thin double-dashed segment (3 fg, 2 bg).
-hseg10	10-pixel thin horizontal line segment.
-hseg100	100-pixel thin horizontal line segment.
-hseg500	500-pixel thin horizontal line segment.
-vseg10	10-pixel thin vertical line segment.
-vseg100	100-pixel thin vertical line segment.
-vseg500	500-pixel thin vertical line segment.
-whseg10	10-pixel wide horizontal line segment.
-whseg100	100-pixel wide horizontal line segment.
-whseg500	500-pixel wide horizontal line segment.
-wvseg10	10-pixel wide vertical line segment.
-wvseg100	100-pixel wide vertical line segment.
-wvseg500	500-pixel wide vertical line segment.
-line1	1-pixel thin (width 0) line.
-line10	10-pixel thin line.
-line100	100-pixel thin line.
-line500	500-pixel thin line.
-dline10	10-pixel thin dashed line (3 on, 2 off).
-dline100	100-pixel thin dashed line (3 on, 2 off).

Test Name	Description
-ddline100	100-pixel thin double-dashed line (3 fg, 2 bg).
-wline10	10-pixel line, line width 1.
-wline100	100-pixel line, line width 10.
-wline500	500-pixel line, line width 50.
-wdline100	100-pixel dashed line, line width 10 (30 on, 20 off).
-wddline100	100-pixel double-dashed line, line width 10 (30 fg, 20 bg).
-orect10	10x10 thin rectangle outline.
-orect100	100-pixel thin vertical line segment.
-orect500	500-pixel thin vertical line segment.
-worect10	10x10 wide rectangle outline.
-worect100	100-pixel wide vertical line segment.
-worect500	500-pixel wide vertical line segment.
-circle1	1-pixel diameter thin (line width 0) circle.
-circle10	10-pixel diameter thin circle.
-circle100	100-pixel diameter thin circle.
-circle500	500-pixel diameter thin circle.
-dcircle100	100-pixel diameter thin dashed circle (3 on, 2 off).
-ddcircle100	100-pixel diameter thin double-dashed circle (3 fg, 2 bg).
-wcircle10	10-pixel diameter circle, line width 1.
-wcircle100	100-pixel diameter circle, line width 10.
-wcircle500	500-pixel diameter circle, line width 50.
-wdcircle100	100-pixel diameter dashed circle, line width 10 (30 on, 20 off).
-wddcircle100	100-pixel diameter double-dashed circle, line width 10 (30 fg, 20 bg).
-pcircle10	10-pixel diameter thin partial circle, orientation and arc angle evenly distributed.
-pcircle100	100-pixel diameter thin partial circle.
-wpcircle10	10-pixel diameter wide partial circle.
-wpcircle100	100-pixel diameter wide partial circle.

Test Name	Description
-fcircle1	1-pixel diameter filled circle.
-fcircle10	10-pixel diameter filled circle.
-fcircle100	100-pixel diameter filled circle.
-fcircle500	500-pixel diameter filled circle.
-fcpcircle10	10-pixel diameter partial filled circle, chord fill, orientation and arc angle evenly distributed.
-fcpcircle100	100-pixel diameter partial filled circle, chord fill.
-fspcircle10	10-pixel diameter partial filled circle, pie slice fill, orientation and arc angle evenly distributed.
-fspcircle100	100-pixel diameter partial filled circle, pie slice fill.
-ellipse10	10-pixel diameter thin (line width 0) ellipse, major and minor axis sizes evenly distributed.
-ellipse100	100-pixel diameter thin ellipse.
-ellipse500	500-pixel diameter thin ellipse.
-dellipse100	100-pixel diameter thin dashed ellipse (3 on, 2 off).
-ddellipse100	100-pixel diameter thin double-dashed ellipse (3 fg, 2 bg).
-wellipse10	10-pixel diameter ellipse, line width 1.
-wellipse100	100-pixel diameter ellipse, line width 10.
-wellipse500	500-pixel diameter ellipse, line width 50.
-wdellipse100	100-pixel diameter dashed ellipse, line width 10 (30 on, 20 off).
-wddellipse100	100-pixel diameter double-dashed ellipse, line width 10 (30 fg, 20 bg).
-pellipse10	10-pixel diameter thin partial ellipse.
-pellipse100	100-pixel diameter thin partial ellipse.
-wpellipse10	10-pixel diameter wide partial ellipse.
-wpellipse100	100-pixel diameter wide partial ellipse.
-fellipse10	10-pixel diameter filled ellipse.
-fellipse100	100-pixel diameter filled ellipse.
-fellipse500	500-pixel diameter filled ellipse.
-fcpellipse10	10-pixel diameter partial filled ellipse, chord fill.
-fcpellipse100	100-pixel diameter partial filled ellipse, chord fill.

Test Name	Description
-fspellipse10	10-pixel diameter partial filled ellipse, pie slice fill.
-fspellipse100	100-pixel diameter partial filled ellipse, pie slice fill.
-triangle1	Fill 1-pixel/side triangle.
-triangle10	Fill 10-pixel/side triangle.
-triangle100	Fill 100-pixel/side triangle.
-trap1	Fill 1x1 trapezoid.
-trap10	Fill 10x10 trapezoid.
-trap100	Fill 100x100 trapezoid.
-trap300	Fill 300x300 trapezoid.
-strap1	Fill 1x1 transparent stippled trapezoid, 8x8 stipple pattern.
-strap10	Fill 10x10 transparent stippled trapezoid, 8x8 stipple pattern.
-strap100	Fill 100x100 transparent stippled trapezoid, 8x8 stipple pattern.
-strap300	Fill 300x300 transparent stippled trapezoid, 8x8 stipple pattern.
-ostrap1	Fill 10x10 opaque stippled trapezoid, 8x8 stipple pattern.
-ostrap10	Fill 10x10 opaque stippled trapezoid, 8x8 stipple pattern.
-ostrap100	Fill 100x100 opaque stippled trapezoid, 8x8 stipple pattern.
-ostrap300	Fill 300x300 opaque stippled trapezoid, 8x8 stipple pattern.
-tiletrap1	Fill 10x10 tiled trapezoid, 4x4 tile pattern.
-tiletrap10	Fill 10x10 tiled trapezoid, 4x4 tile pattern.
-tiletrap100	Fill 100x100 tiled trapezoid, 4x4 tile pattern.
-tiletrap300	Fill 300x300 tiled trapezoid, 4x4 tile pattern.
-oddstrap1	Fill 1x1 transparent stippled trapezoid, 17x15 stipple pattern.
-oddstrap10	Fill 10x10 transparent stippled trapezoid, 17x15 stipple pattern.
-oddstrap100	Fill 100x100 transparent stippled trapezoid, 17x15 stipple pattern.
-oddstrap300	Fill 300x300 transparent stippled trapezoid, 17x15 stipple pattern.

Test Name	Description
-oddostrap1	Fill 10x10 opaque stippled trapezoid, 17x15 stipple pattern.
-oddostrap10	Fill 10x10 opaque stippled trapezoid, 17x15 stipple pattern.
-oddostrap100	Fill 100x100 opaque stippled trapezoid, 17x15 stipple pattern.
-oddostrap300	Fill 300x300 opaque stippled trapezoid, 17x15 stipple pattern.
-oddtiletrap1	Fill 10x10 tiled trapezoid, 17x15 tile pattern.
-oddtiletrap10	Fill 10x10 tiled trapezoid, 17x15 tile pattern.
-oddtiletrap100	Fill 100x100 tiled trapezoid, 17x15 tile pattern.
-oddtiletrap300	Fill 300x300 tiled trapezoid, 17x15 tile pattern.
-bigstrap1	Fill 1x1 transparent stippled trapezoid, 161x145 stipple pattern.
-bigstrap10	Fill 10x10 transparent stippled trapezoid, 161x145 stipple pattern.
-bigstrap100	Fill 100x100 transparent stippled trapezoid, 161x145 stipple pattern.
-bigstrap300	Fill 300x300 transparent stippled trapezoid, 161x145 stipple pattern.
-bigostrap1	Fill 10x10 opaque stippled trapezoid, 161x145 stipple pattern.
-bigostrap10	Fill 10x10 opaque stippled trapezoid, 161x145 stipple pattern.
-bigostrap100	Fill 100x100 opaque stippled trapezoid, 161x145 stipple pattern.
-bigostrap300	Fill 300x300 opaque stippled trapezoid, 161x145 stipple pattern.
-bigtiletrap1	Fill 10x10 tiled trapezoid, 161x145 tile pattern.
-bigtiletrap10	Fill 10x10 tiled trapezoid, 161x145 tile pattern.
-bigtiletrap100	Fill 100x100 tiled trapezoid, 161x145 tile pattern.
-bigtiletrap300	Fill 300x300 tiled trapezoid, 161x145 tile pattern.
-eschertiletrap1	Fill 1x1 tiled trapezoid, 216x208 tile pattern.
-eschertiletrap10	Fill 10x10 tiled trapezoid, 216x208 tile pattern.
-eschertiletrap100	Fill 100x100 tiled trapezoid, 216x208 tile pattern.
-eschertiletrap300	Fill 300x300 tiled trapezoid, 216x208 tile pattern.
-complex10	Fill 10-pixel/side complex polygon.
-complex100	Fill 100-pixel/side complex polygon.
-64poly10convex	Fill 10x10 convex 64-gon.
-64poly100convex	Fill 100x100 convex 64-gon.

Test Name	Description
-64poly10complex	Fill 10x10 complex 64-gon.
-64poly100complex	Fill 100x100 complex 64-gon.
-ftext	Character in 80-char line (6x13).
-f8text	Character in 70-char line (8x13).
-f9text	Character in 60-char line (9x15).
-f14text16	2-byte character in 40-char line (k14).
-tr10text	Character in 80-char line (Times-Roman 10).
-tr24text	Character in 30-char line (Times-Roman 24).
-polytext	Character in 20/40/20 line (6x13, Times-Roman 10, 6x13).
-polytext16	2-byte character in 7/14/7 line (k14, k24).
-fitext	Character in 80-char image line (6x13).
-f8itext	Character in 70-char image line (8x13).
-f9itext	Character in 60-char image line (9x15).
-f14itext16	2-byte character in 40-char image line (k14).
-f24itext16	2-byte character in 23-char image line (k24).
-tr10itext	Character in 80-char image line (Times-Roman 10).
-tr24itext	Character in 30-char image line (Times-Roman 24).
-scroll10	Scroll 10x10 pixels vertically.
-scroll100	Scroll 100x100 pixels vertically.
-scroll500	Scroll 500x500 pixels vertically.
-copywinwin10	Copy 10x10 square from window to window.
-copywinwin100	Copy 100x100 square from window to window.
-copywinwin500	Copy 500x500 square from window to window.
-copypixwin10	Copy 10x10 square from pixmap to window.
-copypixwin100	Copy 100x100 square from pixmap to window.
-copypixwin500	Copy 500x500 square from pixmap to window.

Test Name	Description
-copywinpix10	Copy 10x10 square from window to pixmap.
-copywinpix100	Copy 100x100 square from window to pixmap.
-copywinpix500	Copy 500x500 square from window to pixmap.
-copypixpix10	Copy 10x10 square from pixmap to pixmap.
-copypixpix100	Copy 100x100 square from pixmap to pixmap.
-copypixpix500	Copy 500x500 square from pixmap to pixmap.
-copyplane10	Copy 10x10 1-bit deep plane.
-copyplane100	Copy 100x100 1-bit deep plane.
-copyplane500	Copy 500x500 1-bit deep plane.
-putimage10	PutImage 10x10 square.
-putimage100	PutImage 100x100 square.
-putimage500	PutImage 500x500 square.
-putimagexy10	PutImage XY format 10x10 square.
-putimagexy100	PutImage XY format 100x100 square.
-putimagexy500	PutImage XY format 500x500 square.
-getimage10	GetImage 10x10 square.
-getimage100	GetImage 100x100 square.
-getimage500	GetImage 500x500 square.
-getimagexy10	GetImage XY format 10x10 square.
-getimagexy100	GetImage XY format 100x100 square.
-getimagexy500	GetImage XY format 500x500 square.

B.2 Window and Misc Directives Commands

Test Name	Description
-noop	X protocol NoOperation.
-atom	GetAtomName.
-pointer	QueryPointer.
-prop	GetProperty.
-gc	Change graphics context.
-create	Create child window and map using MapSubwindows.
-ucreate	Create unmapped window.
-map	Map child window via MapWindow on parent.
-unmap	Unmap child window via UnmapWindow on parent.
-destroy	Destroy child window via DestroyWindow parent.
-popup	Hide/expose window via Map/Unmap popup window.
-move	Move window.
-umove	Moved unmapped window.
-movetree	Move window via MoveWindow on parent.
-resize	Resize window.
-uresize	Resize unmapped window.
-circulate	Circulate lowest window to top.
-ucirculate	Circulate unmapped window to top.