X terminals on SupermaX

Test of a large installation
of Tandberg X-terminals
on the Supermax

External Revision 2.0

Dansk Data Elektronik A/S
Thomas Herlin Jensen
May 1991

CONTENTS

1. Preface 1
   1.1 Audience 1
   1.2 Acknowledgments 1

2. Executive Summary 2

3. Analysis of X Applications 3
   3.1 Introduction 3
   3.2 Memory Usage of X Applications 3
   3.3 The Communicational Behaviour of X Applications 4

4. Characteristics of the tdv6230 Xterminal 10
   4.1 Introduction 10
   4.2 The tdv6230 10
   4.3 The X Server 12

5. The Setup of the Supermax 14
   5.1 Introduction 14
   5.2 The Hardware 14
   5.3 The operating system and TCP 15

6. The Test results 16
   6.1 Introduction 16
   6.2 Booting system 16
   6.3 Character I/O test 16
   6.4 Graphics 19
   6.5 The network 19
   6.6 Fontloading 20

7. Errors and Inexpiencies 21
   7.1 Tdv6230 21

8. X configuration guidelines 22
   8.1 X usage profile 22
   8.2 Guideline table 22
   8.3 Network 23

9. Appendix A References 24

10. Appendix B: DeMax E-CAD Xlib figures 25

11. Appendix C: Supermax config list 26

LIST OF FIGURES

Figure 1. Xterm Client -> Server packet size distribution 8

Figure 2. Xterm Server -> Client packet size distribution 9

Figure 3. Boottime for tdv6230 16

Figure 4. Character output test 17

Figure 5. Graphic performance 19

LIST OF TABLES

TABLE 1. X-applications and their estimated memory usage 4
TABLE 2. X-lib request from client to server 6
TABLE 3. mwm profiling draging from corner to corner 7
TABLE 4. xterm profiling of a *normal* half hour work 8
TABLE 5. The tdv6230 hardware overview 10
TABLE 6. The tdv6230 Software overview 12
TABLE 7. Xbench performance figures 13
TABLE 8. The Supermax Hardware 14
TABLE 9. Network Measurements 20
TABLE 10. Fontloading comparison 20
TABLE 11. Supermax X-terminal configuration table 23

1. Preface

The intention with this report is to test the Tandberg X-terminal (tdv6230) and particularly it's behaviour on the Supermax. It is intended to show the feasibility of large X-terminal installations as well as the bottlenecks in such configurations.

The focus of the test was not primarily on the type of X-terminal, and it's capabilities in e.g. graphics, text, but more the issue of the general X11 behaviour on network and on the Supermax, in terms of memory, cpu usage and MIOC (Multiple I/O Controller) load.

The goal is to find a rules of thumb, or better, a formula to configure Supermax resources and settle a number of X-terminals on a Supermax according to specific needs of the X-installation and X-users.

1.1 Audience

This document is aimed at people interrested in X-windows and it's behaviour on the Supermax. This report will give system administrators, sales consultants and others the guidelines to actually plan and review certain future X configurations. The concept groups in DDE that are considering building X-applications can get a feeling what the Supermax system is capable of.

For the reader of this document it is suggested to have some general knowledge of the X Window System. For those readers interrested in this topic, without preknowledge it is suggested that you refer to [1] and [5] for further information.

This report is devided into several chapters regarding X11 on Supermax and the Tandberg X-terminal tdv6230. Some of the chapters might be read independently, dependant of the interrest of the individual reader.

- **Chapter 3** - gives an analysis of typical X11 applications and their demand of memory as well as their communication to the X-terminal.
- **Chapter 4** - describes the Tandberg X-terminal tdv6230 and discusses the technical data and performance as well as the general impression of the product.
- **Chapter 5** - describes the setup of the Supermax machine and especially the setup of XOS parameters for X11.
- **Chapter 6** - list the test results in the areas text, graphics and networking.
- **Chapter 7** - highligths the errors discovered.
- **Chapter 8** - is a conclusion of the test in a form of a X guideline table that can be used to compare the Supermax with planned X11 installations.

1.2 Acknowledgments

First of all a thank you to Tandberg Data A/S, Oslo. They provided us with 25 tdv6230 X-terminals at no cost. With this gesture it was possible to assemble our first large X-terminal installation.
I also want to thank Peter Holm, Peter Kyllesbeck and the DDE MIOC/TCP team for their help concerning the test environment.

2. Executive Summary

X-terminals are becomming increasingly popular not only in technical environments, but also in commercial use. The graphics capability of the X-terminal and the growing market of X-based applications, makes it possible for X to enter in installations, where it replaces the old character I/O terminal in favor of decentralized PC solutions. To acceive this goal it is neccessary to determine the implications of large X-terminal installions. With the courtesy of Tandberg Data it was possible to make a installation of 24 X-terminals connected to a 3 CPU Supermax (1 additional R3000 CPU was used to measure performance gains). For the networking a MIOC was used, running TCP (256 channels).

Configuring large systems on the Supermax requires attention on the memory and the TCP communication.

A minium memory of 1.5 MByte is required for each X-user. This is due to the fact that X11 applications (especially Motif based) uses large data segments.

The minimum number of TCP channels for each user are 6 channels.

The Tandberg tdv6230 is a X-terminal with a stable X11 Release 4 server, with some minor exceptions. The performance of the X-terminal is medium level and about the same as for a NCD19 from Network Computing devices, which was compared with the tdv6230.

3. Analysis of X Applications

3.1 Introduction

Before we started testing the X11-environment with 24 connected Tandberg tdv6230 X-terminals to a Supermax, we analysed the requirements that X-applications usually have. The goal was primarily to get parameters for the right HW configuration before testing. So we looked for figures of memory size and communication load of average user applications.

We have picked out some typical X applications which are default user setup in our X environment.

- **mwm** - Motif window manager
- **xterm** - terminal emulation program
- **oclock** - analog clock program
- **xbiff** - mail arrival indicator program

As a DDE developped application we have also taken the DeMax E-CAD system.

- **DeMax** Electronic E-CAD for PCB Layout from DDE

3.2 Memory Usage of X Applications

The promoter's of X11 and their counterparts argue a lot about design issues of X11, but one fact remains: *The consumption of large amounts of memory.*

To better understand the problem we have to look at the implementation of X11 libraries and X aplications.

X11 is implemented with the intention to be as *independant* as possible with regard to operating system as well as communication protocols. This results in implementing interface layers that hide the dependencies to a certain level. Thus this independence results in a *minor* code overhead.

The major bulk of code comes with the nature of window system handling. Going into detail on this subject would be out of scope of this report, but in general the handling of windows and the graphic action in response to incomming events is rather complex, resulting in large amount of code for rather limited programs. For further details in X programmming refer to [5,6,7,8].

To make life easier the X11 toolkit **Xt** reduced the complexity for the programmer, but at a price of adding large library and spending yet more memory. By the introduction of the **Widget,** a compound window element (normally representing a graphical user interface item), the *real* explosion of memory usage began in X11. This time in the data segment of the application, which actually is much more critical, since data can not be shared among applications. The reason for this is the exploitation of the resource management in X11. Every widget is given the freedom of beeing changed and customized via resources, hence they have to use memory for their individual settings.

This is not the end of the memory luting of X11. Industry standards in graphical user interfaces (GUI) have emerged the last few years, giving the Motif user interface good position in UNIX environments. As Motif is build on top of Xt the memory consumption mentioned above also include Motif applications. But as the Motif library introduces an advanced widget set, with a lot of setable resources, along with special (compound) string

handling etc., both text and data segments grow considerable.

So when you end up building Motif based 'hello.world' program, you have programms with a nice user interface but at a price of approximatly 1 MByte[1].

In general the memory usage of X11 is not a major problem, because:

- memory is getting faster an cheaper (and maybe it always will ?)
- you have a workstation with your own 32 MByte, (so you propably are bored reading about the previously mentioned memory consumption)

But memory usage can be a major problem if you work on large UNIX system, such as the Supermax. Typical for large Supermax installations are:

- more than 32 users, on very large even more than 100
- you run large database packages like Oracle
- your users run many large programs like Uniplex,Wordperfect etc.
- you run all kind of I/O services NFS,LM/X,X25,3290, etc.

so the topic of memory is highly relevant when discussing X11 installations on Supermax. The table below show the memory usage of typical applications at DDE. Actually it is assumed that every user has all the listed applications as a minimum.

Client	Lib	Text 68030	Text R3000	Data	Data max
mwm	Motif	602.112	1085.440	ca.350.000	450.000
xterm	Xt	299.008	528.384	ca.270.000	300.000
oclock	Xt	208.896	372.736	ca.131.000	
xbiff	Xt	208.896	376.832	ca.143.000	
total Bytes		1.318.912 1288kByte	2.363.392 2308kByte	894.000 873kByte	

TABLE 1. X-applications and their estimated memory usage

The data size of the applications was measured in active condition, with the **sysdisp** tool and all numbers are pagealigned.

As you can see from the table above the consumption is quite large considering that this is a minimum for one user. If you add more users to the same CPU you will share the text but still have to add data and stack of 941kByte per user.

3.3 The Communicational Behaviour of X Applications

Before you start making a test of a X installation it is important that you have a feeling what actually is communicated between your X-client and the X-server a nd collect the information

1. Demo program hellomotif: text 692kByte, data 356kByte, by the way this program was build with the UIL (User Interface Language)

exchanged via the X-Protocol [2].

To measure the communication we used a tool called **xscope,** which allows you to probe the connection between an X-client and the X-server. This is done by manipulation of network addresses, which means that the xscope program set's itself between the client and server connection.

By this method we have analyzed some typical X applications, mentioned in the introduction, which are frequently used in our X environment.

The benchmark programs presented later on where also analysied, but the programs above give a better idea of what a default user situation is producing.

To give a usefull analysis on the communication trafic, you have to split the communication up into three phases: Initial communication setup, as required for all standard X applications, client specific startup session and the typical runtime traffic.

3.3.1 The Inital Startup Time

The initial startup of a client is actually reduced to three requests:

- XOpenDisplay
- CreateGC (GraphicalContext)
- GetProperty of Resource Manager

These three request produce a total of 60 Bytes data from the client. The server responds to these request with a total of about 3K bytes. This last number is an average and can vary because of the property list of the resource manager. The resource manager keeps the database entries of client resources in the server. The number of set resources can vary between users, but the number shown is the usual default at DDE. When clients extensively use resources like e.g. programs written with the Wcl[2] (Widget Control Library), the replied property list can grow an order of magnitude.

3.3.2 Client Specific Startup

After the initial startup the different clients run their specific startup, which of course varies among the nature of the program. Interresting here is actually not the types of X-protocol messages, but more the amount of data that is exchanged between client and server. The time for startup of applications is dependend of this information flow, which for some applications can take several seconds. Some applications are well aware of their time consuption, so they start to show an image of an hourglas like e.g. the Motif window manager.

The table 2 shows the amount of data exchange before the client is in a usable condition, i.e. before ready for user input.

---

2. Wcl is becomming increasingly accepted, e.g. the new dbx-frontend from Mips is Wcl based.

application	client request	request total	server replies	replies total	comments
mwm	354	9496 Bytes	84	9060 Bytes	
xterm	60	2312 Bytes	?	3660 Bytes	+ ls
oclock	39	1340	3	256	
xbiff	42	1856	8	480	
DeMax	943	19144 Bytes	319	14396 Bytes	13 events

TABLE 2. X-lib request from client to server

The amount of data and the number of requests are not so exiting but give a feeling of the startup time for the application. You can sense the complexity level of the program regarding X by just comparing oclock startup with the DeMax.

The startup time for different X-client's does depend of the load of the overal system, but normally the user only startup application once in their runtime, so waiting is not so critical.

At DDE we run the terminology, setting of virtual terminal, in the users profile, which gives xterm an actual startup time of app. 3.5 sec.

The details of the DeMax profiling can be found in Appendix B for those who are interrested.

3.3.3 Client Runtime Behaviour

The runtime behaviour of a X-client is quite difficult to fix in numbers. This is due to the way the user interferes with the program and how the user set's up his environment for the program. The main reason for this difficulty of defining runtime behaviour is the event oriented method of X11. The user is the event generator, in form of key-press, mouse movements and mouse-button press. These events are then dispatched to the clients which then take action. Sometimes the action can be set/modified by the user resulting in different client behaviour.

An example of this user modification is the Motif window manager [3]. By default the Mwm*keyboardFocusPolicy is explict, which means that a window is active and accessable first when you have clicked on the window. By this setting all mouse movements are not interpreted by the window manager, only when clicking the mouse. This has the advantage that the mwm does a minimum of action, and theirby keep the cpu and network load to a minimum. This feature can easily be overruled by user settings that redefine Mwm*keyboardFocusPolicy and enable focusAutoRaise[3]. This change in resource settings will produce a lot more action compared to the previous, because it will activate every client that corresponds to the current mouse position, and place the window on the top if it was covered by another.

We will now take a look at the behavior of our selected clients.

3. Mwm*keyboardFocusPolicy: pointer & Mwm*focusAutoRaise: true

3.3.3.1 mwm - Motif window manager

The window manager is generally the most difficult client to program, because it's ability to stay passive, when the user set's it up to behave that way (as the previous example showed). But generally the window manager takes action on the root window when it menu is activated by a mousebutton press, and on the border of each displayed window. Of course there are other actions like: iconbox, autoraise,pointer focus etc., but they are more specific to user settings. So how do we profile the mwm?
The best thing to do is to pick an everyday action and see what the window manager has to do. For this purpose we picked: a movement of a xterm window from the upper left corner to the lower right. This is something done quite often by users.

communication	**from client**	**from server**	**real time**	**bytes/s**
	17272	4352	3 sec	7208
packet types	REQUEST	REPLY	EVENTS	total packet count
	398	59	80	127
Xlib calls	PolyFillRectangel	PolySegment	QueryPointer	ImageText8
	116	104	55	55
Events types	MotionNotify	**Replies**	QueryPointer	
	68		55	

TABLE 3. mwm profiling draging from corner to corner

The table above show that a total of 400 REQUEST from the mwm and a total of 127 packets are exchanged for this operation. As it was not possible to measure with xscope in real time we estimated the real time to 3 seconds. That means that 7,3 kByte/sec are produced by this action. As users not constantly are moving windows around, this traffic is usually not critical. Some vendors of X-terminals provide a solution to these communication *peaks,* by the ability to run local window managers. Allthough this is done by the price of adding more memory into the terminal, it can become feasible in environments with loaded ethernet segments.

For the people interessted the table 3 also shows the most frequently used Xlib calls and Server events.

3.3.3.2 Xterm

The xterm program is I suppose currently the most frequently used program in X11. This is due to the fact that the amount of fully X based clients are limited, thus a window into the 'old' wellknown world of shell and vi etc. is needed. To profile xterm and give an exact number for it's network trafic, is similar to counting ants, you always loose a few. No, seriously, every user has a different behaviour: some type very fast, some like to cat large files, some just do anything. The best thing to do is to profile a user over a large period of time. In a period of 30 minutes, I have profiled a typical user (myself). In 30 minutes you use several unix commands like ls,cd etc., you use editor like vi. The 30 minutes profiled were active minutes, no coffebreaks etc.. With this waste amount of data the initial startup, as well as the client specific startup was subtracted. For the rest of the time the network traffic, the Xlib request and the events from the server were summed up.

communication	**from client**	**from server**	**bytes/s**	
	140556 bytes	91984 bytes	129.1	
packet types	REQUEST	REPLY	EVENTS	total packet count
	6542	0?	2746	2148
Xlib calls	ImageText8	PolyFillRect	CopyArea	ClearArea
	4192	1102	1115	113
Events types	KeyPress	NoExposure	FocusIn	FocusOut
	1587	1115	9	12

TABLE 4. xterm profiling of a *normal* half hour work

As you see from the table 3 is the amount of data exchanged is quite high compared to the effective characters represented, which were only 9967 characters. Don't even think of comparing this with a serial terminal. If you look at a single KeyPress event generated by a keyboard hit you se that the server sends a 32 Byte packet to the client. This causes the client to respond with an 60 Byte packet echo. But life is not as simple as that because you have generate more packets than theese 92 bytes. When you e.g. have to scroll the window up one position xterm sends a CopyRequest. The essence is that the packet size is variable and the distribution of packet sizes are shown in fig 1, for the xterm client, and in fig 2 for the server. If you look at the fig 2 of the server, you can see that the most frequent package size sent is 32 bytes.

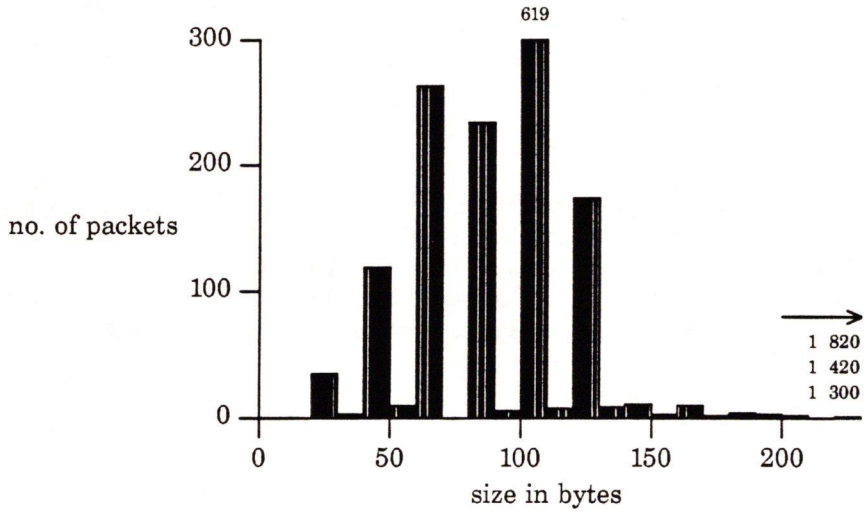

Figure 1. Xterm Client -> Server packet size distribution

The figures 1 and 2 give you a good intuition of the likelihood of xterm data packets. So they could be used e.g to settle issues of which buffers to use in your streams configuration. But this will be described in detail later.

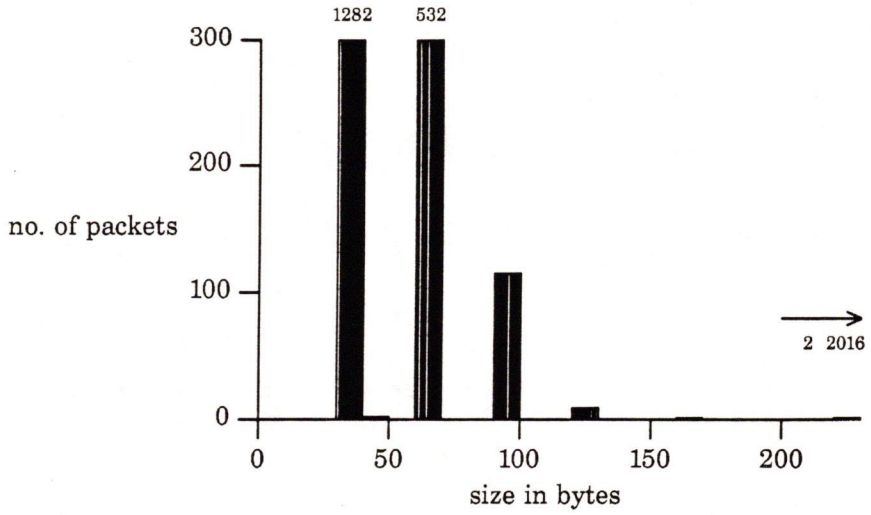

Figure 2. Xterm Server -> Client packet size distribution

3.3.3.3 oclock

The oclock program updates its round clock by default every minute (this can be change by a user option). The runtime behaviour of this client is quite predicable and easy to settle. Every 60 seconds the oclock sends 4 FillPolygon, with a total of 160 Bytes. No server replies are given.

3.3.3.4 xbiff

The xbiff program is unpredictable, because who can know when mail arrives. No seriously the xbiff does only act when mail arives, by sending a bell and loading a bitmap. With a total request of 64 bytes. Xbiff is unexciting for network load, but become interresting when we look at CPU load. The xbiff program is build arround the Athena Mailbox Widget. This widget times out every 60 seconds and loads the CPU quite heavily. This problem will be investigated further, but is not part of this report.

4. Characteristics of the tdv6230 Xterminal

4.1 Introduction

To evaluate X-terminals, you have to focus on several issues like performance, software quality and protocols. But there are of course also the general issues of terminals regarding ergonomy, features and price. In this evaluation we have focused more on the issues mentioned first.

In the following the hardware and the facilities of the tdv6230 will be presented, followed by a desciption of the X-Server and performance measurements.

4.2 The tdv6230

4.2.1 Unpacking

The tdv6230 comes with a monitor, keyboard, mouse and a cartridge with server software. The monitor is a 17 inch monochrom, with the terminal hardware build in on the right side of the monitor. This gives the terminal a rather compact outline. The keyboard is a PS/2 style keyboard with 'softtouch' keys. The mouse is a standard serial mouse with 3 buttons.

Before installing the neccessary cables the hood must be removed from the back, which seemed a bit awkward. The installtion of the cables is quite easy, except for the keyboard key, where the worst position was chosen, on the bottom of the monitor. Due to the eight poled connector it was often impossible to plug it, without turning the monitor upside down.

As an ergonomic feature the monitor can be adjusted not only in angel, but also in height.

4.2.2 Hardware

The Hardware features of tdv6230 are listed in table 5. The Monitor of the tdv6230 has the resolution of 1024x768 pixels, with 1 bitplane. The refresh rate of 78 Hz is above average (compared to other vendors) and gives a excellent display quality.

Monitor					
Type	Bitplanes	Size	Resolution	Refresh	DPI
mono	1	17 inch	1024x768	78 Hz	87
Processor					
Graphics	MHZ	other	Network		
TMS34010	50	none	LANCE		
Memory		**Ports**			
Min	Max	Thin Ether	Thick Ether	Serial	
2	4 (3.5)	yes	yes	2	

TABLE 5. The tdv6230 hardware overview

The hardware of the tdv6230 is build arround a TMS34010 clocked at 50 MHz. This processor seems to be very popular and is e.g. also used by IBM Xstation 120, HDS V19 or HP 700/X. Unlike the tdv6230 the most other X-terminal vendors use an additional processor for networking, keyboard and mouse I/O.

The memory on the tdv6230 was 2 MByte, which is somehow a useable minimum. The best

is to have the terminal equiped with 4 MByte, which is the maximum possible, theirby avoiding memory shortage, when using backing store.

The network connection can either be thin or thick ethernet configured easily by a switch at the connector. Both connection method were tested without any problems. The terminal has two additional serial ports, which can be used for local printing or serial terminal (TDV1211 emulation) connection. Usage of serial X-terminal connection is not possible, as serial transport protocols like SLIP are not supported[4].

4.2.3 Installation

To install the terminal you have to use the internal setup menu. The menu is quite self explaining and the mouse can be used to activate the fields. For those unhappy with english there is a possiblilty to change language of the menu by remote configuration.

Before you can use the tdv6230 you have to know the ethernet address which is listed in the **System Setup** submenu. This submenu is protected by a password, which makes it mandatory to read the installation guide[5]. The ethernet address is then added in bootptab file of the host together with the reference to the bootfile. Bootp is the only boot protocol supported at DDE. The other field/entries in the setup menu can be changed after own purpose and be saved in Non-Voilatile-RAM. For further information please refer to [4]. For filetransfer TFTP [6] is used to download the bootfile (X-server) and the remote configuration file. This file can either be a general configuration file, i.e. for all terminals connected, or a terminal specific (identified via internet address).

The configuration file allows the settings of different areas, which are:

- Networking configuration
- Serial Line configuration
- User preferances
- X-server settings

Most of the configuration fields can be stored in NVRAM. A nice feature is the possibility to include files. This allows a user to include default system settings and add only the few he want's to change like e.g. the Xhosts parameter.

Further aspect on X are described in the next section.

As a additional utility the the tdv6230 supports telnet, with an ansi terminal emulation. We use this utility quite seldom, but it gives you the possibility to connect to other machines if your traditonal xhost is down.

An overview of the software supported is listed in Table 6.

---

4. Using serial line for X-sessions is anyway not very usefull
5. which is against the nature of 'real' engineers
6. Trivial File Transfer Protocol

Boot		Filetransfer		Utilities
BOOTP	RARP	TFTP	NFS	Telnet
yes	yes	yes	no	yes
X-Server				
version	revision	extensions	XDMCP	local clients
X11R4	2.0 13/2/91	SHAPE	yes	no

TABLE 6. The tdv6230 Software overview

4.3 The X Server

The server supplied was a X11 Release 4 server 2.0 revison. Errors detected are described in chapter: Errors and Inexpencies.

As extension the server supplied the SHAPE feature. The X-server request logon via XDMCP[7] and had no problem connecting to the host.

The only negative remark is that the retry of XDMCP connection does not allways work.

As a special feature the X-server has the capability to cash fonts, which in principle is a nice feature. But the fontcashing must be limitable to a maxium amount of memory, and not use all the memory it can grab.

4.3.1 Benchmark Results

To get a better idea of the performance capability of the tdv6230 we made a benchmark. Everybody knows that benchmarks allways are connected with uncertainty and limits, but everybody does them anyway. For the X-server benchmark we picked the **Xbench** program, which actually is a program used to tune X-servers. The Xbench program was chosen because it gave host independant figures[8].

The results of this x-benchmark is a number of stones in different areas:

- textStones - the ablity to display text, scroll text
- blitStones - ability to show an move images bitblt
- complexStones - ability to create,clear and destroy windows.
- arcstones - ability to draw circles and arc's.

7. X Display Manager Control Protocol
8. The Xbench program was run on a Supermax with a 68030 and 3 times more powerfull R3000, with nearly the same result (only the complexStone had a difference > 10%)

stones	line	fill	blit	arc	text	complex	XStones
tdv6230(r2.0)	30883	19866	22602	402191	25625	13986	23631
ncd19(r2.1)	21108	17104	35363	200798	31281	17254	24320
ratio	1.46	1.16	0.64	2	0.8	0.81	0.97

TABLE 7. Xbench performance figures

Allthough xbench produces a series of numbers (Stones) resulting with an Xstone composed of weighting the individual stones, you have to compare these numbers with an equivalant X-terminal. For this purpose an medium level X-terminal the NCD 19[9] was chosen.

The index shows the performance relative to NCD 19. The only week points are blit and text, which are typical memory copying functions. The performance gain on arcStones is (I presume) related to the force of floating point of the TMS34010 DSP.

For DDE typical usage the textStones and lineStones are the most typically used characteristics. For E-CAD applications the arcStones could also have some interest.

9. Unix Today X-terminal test from the April 1st 1991 picked the same reference machine

5. The Setup of the Supermax

5.1 Introduction

Before going into the details of the testmachine setup I will give a very short introducion to the Supermax. Readers familiar with this topic might glance over this chapter.

The Supermax is a UNIX minicomputer developed by DDE. The Supermax has a scaleable multiprocessor architecture that are loosely coupled, i.e. every processor has it's own memory. The hardware provided is based on Motorola 68030 and MIPS R3000, which both can coexist simultaniously. The I/O subsytem is build upon an intelligent controller basis, i.e. the controller has it's own CPU and can run certain drivers locally.

The multiprocessing at UNIX level is on transparent task basis. This means that processes are executed regardlessly of the CPU number and can interoperate transparently with processes on other CPU's. The CPU selection scheme is based on a loadbalancing algorithm.

The operating system SMOS 3.1 providing the compatibility with AT&T System V 3.1, including some extensions e.g. NFS,symbolic links etc..

5.2 The Hardware

The Supermax chosen was a SupermaxVertical (130) with first 3 CPU's of 68030, then later in the test the machine was enhanced with a R3000 processor module. The Hardware used is listed in table 8.

CPU				
Type	**Clock**	**Memory**	**Swap**	Local Procs
68030	33 MHz	16 MB	26.6 MB	150
68030	33 MHz	16 MB	27.6 MB	150
68030	33 MHz	16 MB	26.6 MB	150
R3000	25 MHz	16 MB	-	150
Controllers				
Type	**Devices**			
DIOC 3 DIOC 2 MIOC MIOC	321 MB disk Floppy Ethernet Ethernet	Streamer	4xtty	

TABLE 8. The Supermax Hardware

The memory on the CPU's was limited to 16 MB per CPU with app. 27 MB swap, which was sufficient for our purpose (because we only ran X related applications). The R3000 had unfortunatly no swap disk, because it was installed later. Please notice that the number of local procs is enlarged to 150, as X-sessions normally have 3-4 times the number of user processes as *traditional* user session.

The MIOC (Multiple I/O Controller) used was (the latest version) equiped with a 20 MHz 68030. The test machine was installed with two MIOC's, with the purpose to measure the gains with two MIOC's compared to one.

5.3 The operating system and TCP

The OS version used for the test machine was a heterogenious OS (capable of running on 68030 and R3000) version 3030.6A from 29/1/91.
The TCP used for this test was a prerelease of the version 3.5 (b) with 256 TCP channels. The rest of the TCP packet, bootp, tftp, rutils complied to this release. During the test no errors were encountered in this package.

6. The Test results

6.1 Introduction

Until now several aspects of X were discussed, the profile of typical applications as well as issues of the X-terminal and X-server. Now we will focus on the test of *real* applications on the Supermax. For this purpose we have split the test up into several categories, which can give an impression of different areas of X-application usage.

6.2 Booting system

The intention with this test is to profile the implications of several X-terminals booting simultaniously. For this purpose we measured the time from when the X-terminal gets it's internet address via BOOTP until the XDM login widget appears on the screen. In this period of time the X-terminal has to access the X-server file and the remote configuration file via TFTP. The size of the files are approxematly 1 MByte. Further more the server has to initialize and run the XDMCP to connect to the host.

The time as function of X-terminals is shown on the figure below.

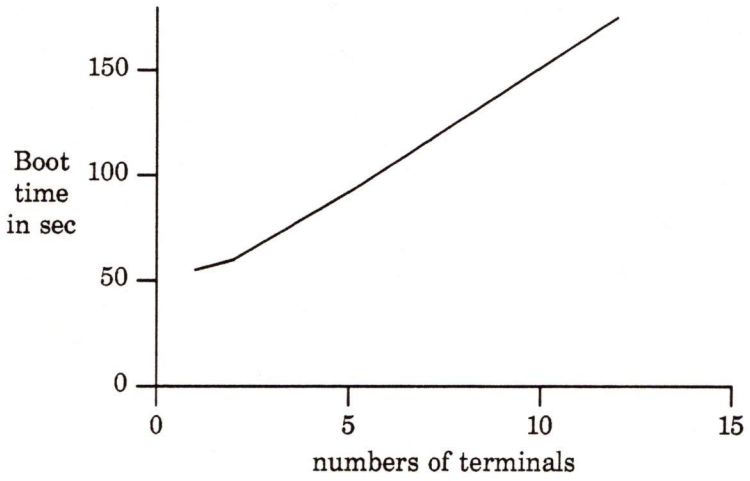

Figure 3. Boottime for tdv6230

The number of X-terminals was limited to 12 more for pratical reasons (as it was quite awkward measuring on more). As the time seems to be linear, it would make little sense too.

As NFS was not supported by the tdv6230 it was not possible to test the difference in performance of these two filetransfer methods.

6.3 Character I/O test

The character I/O capability is a very important for the comparison of X-terminals versus traditional terminals. As long as most installations rely on applications that are character based, the obvious comparison must emphasis on terminal emulation programs, because these programs allow a backwards 'compatability'. Such a program is the allready mentioned xterm client. Other terminal emulation programs exsist, but are not concidered in this

report. For the purpose of testing the character I/O capability of xterm it would have been preferable to either have 24 users tapping on their keyboards or possibility to use some server input extensions, for simulation of keyboard input. As we unfortunatly didn't have any of these options, we used a model instead, which was combining character output test with character input tests.

6.3.1 Character Output

For the character output we use shell scripts that repeatately cat'ed the content of a file onto the xterm window. The different output sessions on the terminals were started almost simultaniously, via delay. The content of the file was exactly 620 bytes, and chosen in a way, that it fitted on one single screen. This is quite important, as xterm cut's away output if it is non visible. After each session the screen was cleared.

In parallel the **load**[10] program was started to meassure the CPU load, together with the **miocidle** program to determin MIOC usage.

In the figure below you see the result for the 68030 MCU for 100 output sessions.

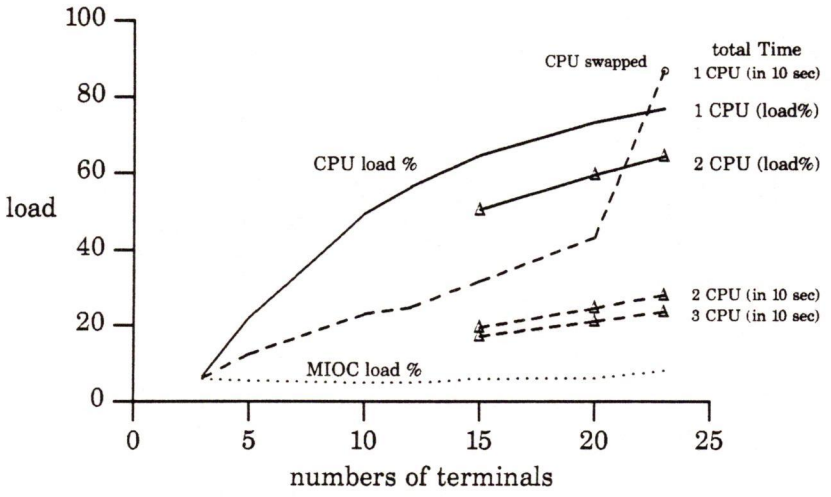

Figure 4. Character output test

The load of the CPU is astonishing high, compared to the actual work beeing done. If you sum up the character output you get an avarage of 2800 characters/sec for 1 CPU. Distributing the load over more CPU's gains of course a speedup. The speedup from 1 to 2 CPU's is about 70 %. The speedup from 2 to 3 CPU's were measured as relatively low app 15%. This was due to the fact that the added CPU was not chosen by the kernel distribution mechanism, because numerous processes (daemons) were running and occupying memory on the added CPU. The difference in CPU load to the other processes were 50%.

10. The **load** program fork's low priority children on all available CPU's and accumulates their runtime usage.

6.3.2 Character Input

As mentioned previously does the keypress of a user, produce a message of 32 byte data, which returns an 60 byte echo message. The maximum input of characters on a tdv6230 seems 3 keystrokes/sec. So when using curser keys for scrolling you typically would produce 276 bytes/sec, excluding the xterm operations for scrolling.

For the test we did not have the possibility to simulate keyboard input, but Tandberg data promised to look at this topic, with some X-server input extensions. This could be the topic of a further investigation.

6.4 Graphics

For the test of graphic performance their was used a standard animation program the ico demo. This program gives the opportunity to evaluate simple continous graphical operations, that can be changed to a certain limit, e.g the ico can show a polyhedra as wireframe, as well as full surface. The ico program was changed to permit time measurement. A further advantage of this program is the use of floating point, which is typical for larger graphical applications.

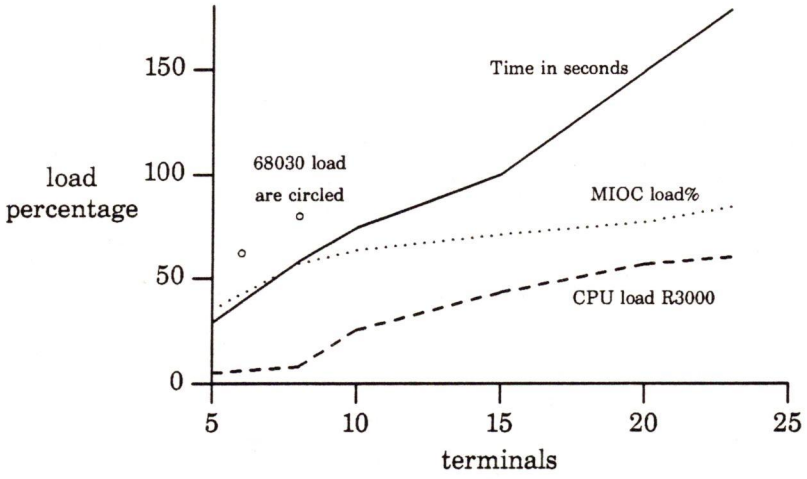

Figure 5. Graphic performance

The graph above shows the load for the RISC R3000. The 68030 was only used for a few test's, as it very quickly became saturated (see circles). The performance gain for the R3000 was a factor of 3.4, for this application. Again we noticed that the system time for the R3000 was larger than the 68030, about 30%.

For applications that require graphic capability, in the areas of CAD, it is suggested to use the R3000.

6.5 The network

6.5.1 Network measurements

The Network was measured by using a PC and the **Spider Monitor** network monitoring tool. With this tool we repeated some of the previous test scenarios. The table below shows the network load of the MIOC, as well as some typical X-terminal situations.

action	peak	average	comments
Host actions			
Max X-load 1 MIOC	15.6%	-	Full saturation 1512 sync. X-calls
Max X-load 2 MIOC	25.2%	-	Full saturation 2478 sync. X-calls MIOC 1 app. 10,7% and MIOC 2 app. 14,5%
X-terminal actions			
Keypress	0.25%	0.08%	
Mousemove-mwm	0.40	0.25	mwm window wireframe
Mousemove	0.30	0.10	draging mouse accross the screen
Fontloading	0.33	-	TFTP requests

TABLE 9. Network Measurements

The consideration of when the network becomes a bottleneck, was not possible to achieve with one machine, as it was not capable of loading the network more than 25% (2 MIOC's). In the last chapter X Configuration guidelines, we have tried to sum up some characteristics regarding the number of X-terminal on an ethernet segment.

6.6 Fontloading

The fontloading capability of X-terminals is often used in application that work with constantly changing font like desktop publishing packages. For the fontloading test a program was used that loaded 36 different fonts.

The table below show the capability of several platforms.

Type	Time sec	Time/font sec	comments
tdv6230	46.5	1.3	caching disabled
tdv6230	55.7	1.5	caching enabled first
tdv6230	2	0.05	caching enabled following
NCD17C	36.4	1	no caching
NCD19	34.8	0.97	no caching
PC-33MHz	14	0.39	Unix PC with local X-server

TABLE 10. Fontloading comparison

In the table above you see that the tdv6230 has font caching capability, which gives a quite good speedup. But the price is loosing precious memory. I noticed that the tdv6230 did not have litmits for the caching of fonts. Which means that after the run of the test program the tdv6230 had 0.5 MByte filled with fonts. This is not very meaningfull since you can get into memory shortage problems, especially on the 2 MByte version. A limit to a certain amount of memory e.g. 100 KByte (app. 7 fonts) should be a default maximum. Otherwise you get memory shortage and will be unable to open more windows.

7. Errors and Inexpiencies

This section is intended for the developers responsible for the listed topics and might be skipped.

7.1 Tdv6230

Hardware
From the very start of the test one terminal did not boot. Although the terminal was set up correctly it was not possible to boot either automatically or manually. I persume it is an error in HW, maybe due to transportation.

Memory management
The terminals used for testing were equipped with 2MByte. This is regarded as an usable minimum, if some rules of thumb are kept. The default of **Backing-Store** must be changed to Always_off instead of On_request. Their were some nasty **Out of Memory - Closing some windows** messages at the start of the test.
The font memory management is also part of this topic. As mentioned previously it memory shortage could occur if multiple fonts are loaded, due to the font caching. A user seteable limmit, with a default maximum set could be the solution here. I would regard this matter as very important, as we at DDE have noticed that some applications browse through the font register in order to determin which are available to the server.
Another solution is of course the proposed scaling of fonts according to the X Logical Font Description (XLFD), this could avoid spending app. 20K for saving each font. XLFD is suggested as a part of X11 Release 5.

Communication
At a point where one terminal was quite heavily loaded with 6 active windows, an error occured as **mget: corrupted mbuf pool** The must be related to TCP layers. As it only occured on one specific terminal, it could be the Network interface.
Sometimes the tdv6230 displayed a message when booting, **Warning: init processed failed** or **Illegal Opcode.** Typical communication error problems. It seems like the TFTP protocol implementation is quite unrelyable, concerning the contents of it's packages. Maybe some checksum's would help[11].

11. DDE's other X-terminals types did not have communication problems quite as often as the Tandberg tdv6230.

8. X configuration guidelines

8.1 X usage profile

This chapter is intended to give you guidelines to configure X-terminal installations. The typical configuring parameters are memory, TCP channels and CPU power and network. In the following we want to list usefull configuration for different installation types. But first we must define some typical areas of usage. For this reason we have defined 4 X user level:

- Level 0 - **Commercial secretary usage**
  The X terminal is more a replacement of character terminals in fields of offices, where 1-3 applications are the daily use. These applications could be wordprocessing, database or special dedicated applications where the common denominator is character I/O.
- Level 1 - **Advanced commercial usage**
  The X applications are no longer limited to characters, but applications like desktop publishing, spreadsheets etc. are used with windowing as a demand.
- Level 2 - **Technical CAD usage**
  The X applications are used in the CAD areas where 2D-graphics[12] is needed. But the usage of the terminal is limited to 'production' sites, where the number of specialized applications are 2-3.
- Level 3 - **SW developpers usage**
  Broad and unpredictable users, that have more windows than they need, to do several things in parallel. In other words, typical software development environments.

8.2 Guideline table

For the 4 types of users, we want to set up a table of requirements.

The table below lists the requirements of the different X-user levels.
Index 1-4 are the X-applications that are used, where the *dedicated* applications are specific X-applications for the integrated solution, e.g. in Level 2 it could be a advanced stockbroker system with integrated spreadsheet. Index 5 sums up the requested TCP channels.
As the maximum number of TCP channels on a MIOC is 256[13] , but as not all channels are available for X usage, we have to set a utilization factor for TCP channels (index 6). This determins the number of X-terminals, from a static point of view.
As the usage of X is quite different we have to define a **X-divider.** The X-divider is the number of estimated X-requests/per second times the number of previously calculated X-terminals divided by the maximum number of X-requests of the MIOC, which is 1500/sec. If the X-divider becomes less then 1 it is rounded to 1. The index 9 shows the X-divider calculated for X-user level. As you might notice has the CAD X-user a large X-divider, as the communication bandwidth of CAD is multiple higher than X-character I/O.

The index 10 lists the total number of X-terminals. As shown previously did the addition of a MIOC increase the X-message throughput with 60%. This ratio could be used in installation with 2 MIOC's.

---

12. In the very near future also 3D-graphics must be considered; a) because it with the PEX standard becomes a part of X11 Release 5 and b) because some PEX-terminals are becomming available.
13. limited through the device minor number

Index	Requirements	Level 0	Level 1	Level 2	Level 3
1	Std. Applications CLOCK/XBIFF/MWM	3	3	3	3
2	XTERM	1	2	1	3
3	DESKTOP MANAGER	-	1	-	1
4	dedicated app.	1	4	3	3
5	TCP channels (incl. XDM)	6	11	8	12
6	TCP utilization	80%	80%	80%	70%
7	**X-terminals/ MIOC**	**34**	**19**	**26**	**16**
8	X-request/sec	20	100	200	80
9	X divider	1	1.3	3.5	1
10	**X-terminals total**	**34**	**15**	**8**	**16**
11	Memory SWAP x 3	1.5 MB	2.5 MB	2.5 MB	2 MB
12	MIPS	0.5	2	4	2
13	No of 68030's or	3	3	-	3
14	R3000	2	2	2	2

TABLE 11. Supermax X-terminal configuration table

The amount of memory (index 11) is estimated for the various fields. Especially for the level 1 and 2 can dedicated X-application become quite large, thus the estimated memory is regarded as a minimum.

Due to the invasion of RISC the CPU performance in general has become less a problem[14], more a marketshare parameter. As X11 arised from the workstation arenas the demand for performance were more than traditional character I/O, but not extreme. The index 12 estimates the MIPS usage per user. Again we have to state an uncertainty in the area of CAD, as certain imulation packages could double up the demand.

8.3 Network

Normally you would install your X-terminals on a network that is department or company wide. This is of course the most natural thing to do, if you keep some figures in mind. One X-terminal has a average producing load of 0.2% of the overall network bandwidth. But as X-clients get triggered by the X-terminal action, the host produces 3 times (in average) the corresponding load, which is 0.6%. As you can see would then 100 X-terminals together load 80% of the network. Without going into detail of certain configuration, I would suggest that 100 X-terminals are quite feasible on the same ethernet segment. If a lot of nettraffic is made by e.g. NFS, a solution with decoupled networks could be made using a dual MIOC configuration.

14. Now the struggel of performance should focus on I/O systems

9. Appendix A References

[1] O'Reilly, Tim, ed., "X Window System User's Guide, Volume 3," O'Reilly and Associates, 1990 3rd edition.

[2] Scheifler, Robert W., "X Window System Protocol X Version 11, Release 4", MIT Laboratory for Computer Science, Part of X11R4 distribution

[3] "OSF/Motif Programmer's Reference." Revision 1.1. Open Software Foundation. 1990.

[4] Tandberg, "TDV 6230 X Terminal, Installation and Operation Guide", Tandberg Data 1990.

[5] Jones, Oliver, "Introduction to the X Window System," Prentice Hall, 1989.

[6] Nye, Adrian, "Xlib Programming Manual, Volume 1" and "Xlib Reference Manual, Volume 2," O'Reilly and Associates, 1988.

[7] Nye, Adrian, and Tim O'Reilly, "X Toolkit Programming Manual, Volume 4," O'Reilly and Associates, 1989.

[8] Young, Doug. "The X Window System: Applications and Programming with Xt (Motif Version)," Prentice Hall, 1989 (ISBN 0-13-497-074-8).

10. Appendix B: DeMax E-CAD Xlib figures

This section is not included in this revision of the report

11. Appendix C: Supermax config list

```
Master MCU:              1          Filedescr:              500
Lockelems:             300          Global processes:       500
Opens:                1200          Shared memory descr:    200
Message queues:        150          Semaphore descr:        200
Maxio:                  32


dump  disk:            /dev/dsk/u14c3s0
root  disk:            /dev/dsk/u13c28s0
Initial program:       /etc/init
console:               /dev/term/u14c0w1
                       9600 baud              8 data bits
                       Send one stop bit      Parity disable


Streams:
Event cells:   512            Queue pairs:   1024
Links:          32            Message blocks:2048
Low fraction:   80            Med fraction:    90

Number of message blocks:
   4 bytes: 128          16 bytes: 512          64 bytes: 1024
 128 bytes: 512         256 bytes: 256         512 bytes: 256
1024 bytes:  56        2048 bytes:  36        4096 bytes:   2

NFS mounts:   0

Module definitions:
timo    32      0
ioct    0       0
mioc    250     250
sp      255     0
vti     105     5

MCU  type      inst.     allow.     local  text   part    items    swapdisk
 #             memory    memory     procs  desc.  desc.           /dev/dsk
 1  M68030    16.00 M  no limit     150    128    400     800 k    u13c13s5
 2  M68030    16.00 M  no limit     150    128    400     804 k    u13c13s4
 3  M68030    16.00 M  no limit     150    128    400     800 k    u13c13s6
 4  R3000     16.00 M  no limit     150    128    400     800 k      -

Dioc3 #13:
Disks:
 1: Floppy
13: Hard disk, length: 321.00 MB
                subdisk  0: 122880 k bytes
                subdisk  1: 122880 k bytes
                subdisk  2:   1024 k bytes
                subdisk  3:   1024 k bytes
                subdisk  4:  27648 k bytes
                subdisk  5:  26624 k bytes
                subdisk  6:  26624 k bytes
28: Mirror hard disk (channel  8/9 ), length: 321.00 MB
                subdisk  0: 122880 k bytes
                subdisk  1: 122880 k bytes
                subdisk  2:  77824 k bytes
                subdisk  3:   2560 k bytes
                subdisk  4:   2560 k bytes
29: Mirror hard disk (channel 10/11), length: 321.00 MB
                subdisk  0: 122880 k bytes
                subdisk  1: 122880 k bytes
                subdisk  2:  82944 k bytes

Dioc2 #14:
Terminals:
 0: normal terminal/printer          1: normal terminal/printer
 2: normal terminal/printer          3: normal terminal/printer
Disks:
 3: First  560 KB 5.25" floppy
 7: Streamer tape length: 120.25 MB
20: First  360 KB  floppy - 40 tracks, 9 sectors

Mioc #6:
```