NOMACHINE		Building and using NX Open Source components	
Prepared by: Gian Filippo Pinzari		*N°:* D-110/05-NXS-DOC	
Approved by: Gian Filippo Pinzari	*Signature:*	*Date:* 07/10/2005	*Amended:* D

Building and using NX Open Source components

Index

1. Building and using NX Open Source components 3
   1.1 Building NX compression library and proxy 3
   1.2 Running NX proxy 5
   1.3 Getting X protocol statistics 6
   1.4 Building NX X libraries and the X session agent 7
   1.5 Building RDP and RFB agents 7

1. Building and using NX Open Source components

How to build and use NX X compression software

Follow the instructions below to build nxproxy, nxagent, nxdesktop, nxviewer and the modified NX X libraries. These components are the same used by NoMachine's client and server to compress and deploy KDE, GNOME and Windows RDP sessions through any network connection, even extremely low-bandwidth Internet links.

1.1 Building NX compression library and proxy

The Xcomp library and nxproxy, a stand alone program wrapping the functionalities built into the library, are the basic components you need to compress the X protocol produced by any standard X client.

In its simplest configuration, a session is run on a remote server by having a NX proxy listening to X traffic and compressing such traffic across a TCP/IP connection to a proxy peer running on the local client. X traffic is decompressed by the local proxy and forwarded to the X server.

Besides compression of X protocol, nxproxy is able to tunnel SMB and multimedia channels, so it is possible to listen to music from the remote terminal server or make available local resources of the thin client to the remote session.

A tunnel can be created between two hosts running:

```
> nxproxy -C :1000
```

on the remote machine, and:

```
> nxproxy -S remote_host:1000
```

on the local computer.

Most NX executables need the nxcomp and nxcompext libraries to work. Given that you built the libraries in your $HOME, run:

```
> export LD_LIBRARY_PATH="${HOME}/NX/nxcomp:\
> ${HOME}/NX/nxcompext:${HOME}/NX/nx-X11/exports/lib"
```

nxproxy is able to compress X traffic with ratios ranging from 10:1 to 1000:1 and more. The way Xcomp/nxproxy achieves these results is through extensive caching and differential compression of X protocol requests and replies. Differential compression means that any X protocol message is analysed to find a similar message encoded in the past. If similarities are found, only the differences are sent through the link.

Images are encoded and compressed through a set of different methods. Encoding is usually performed by NX agents, using high-level functions built in Xcompext library. Agents can choose different methods, according to the type of session being run. Encodings exist for JPEG, PNG, raw bitmaps, as well as for special formats like Hextile and Tight, used by VNC sessions, and RDP bitmaps, used by Windows Terminal Server.

Images encoded in one of the available methods are further compressed and cached by nxproxy. Cache populated at run time by local and remote proxies is saved on disk at the time session is shut down. When a new session is negotiated, a list of available caches is sent through the link. If a matching cache is found, proxies load the cache from the disk, so further increasing compression efficiency.

A normal KDE session start-up, running through a modem link, can require as few as 35KB of data to be completed, resulting in a compression ratio of 200:1.

nxproxy is a "specialized" compressor. By "understanding" the X protocol it is able to perform many optimizations on the produced X traffic. It adapts bandwidth consumption according to network conditions and, by only compressing what is not cached by the remote proxy, it reduces the compression overhead to a minimum. By comparison, nxproxy is able to consume 1/10th of the CPU used by ZLIB compression built in SSH, while still compressing better by a factor of 10.

To build nxproxy you need:

nxcomp-X.Y.Z-N.tar.gz

nxproxy-X.Y.Z-N.tar.gz

As for all the other Open Source components, get sources from http://www.nomachine.com/dev_sources.php and follow instructions you find in the README file.

```
> cd ~/NX

> tar zxvf nxcomp-X.Y.Z-N.tar.gz

> cd nxcomp

> ./configure

> make

>

> cd ~/NX
```

> tar zxvf nxproxy-X.Y.Z-N.tar.gz

> cd nxproxy

> ./configure

> make

1.2 Running NX proxy

In the package nxscripts there are some bash scripts made available by NoMachine to let users run NX components in sample configurations, in local-to-local conditions. Scripts can be modified and run as part of SSH remote shell procedures, to let any Linux or Unix host become a mini NX server.

We suggest you put all sources in a NX directory in your home directory. This is the default configuration. Scripts will set the required library paths according to this directory layout.

Scripts are used by NX core developers to test their software in different configurations. They should work without any modification. We suggest you try first to run sessions in local-to-local conditions, using these scripts. You can later modify them and move the remote proxy to a different host.

Let's start our first NX tunnel:

> tar zxvf nxscripts-X.Y.Z-N.tar.gz

> cd nxscripts

> ./run-nxproxy

This will run nxproxy listening on DISPLAY port :8. Scripts come with shell echo option -x enabled. This is intended to help you if something goes wrong. To disable this feature just type the 'set -x' line as follows:

> #set –x

By simply using nxproxy you can transport any X application.

> export DISPLAY=:8

> xterm &

> mozilla &

> gnome-session &

> startkde &

Scripts will create C-n/S-n session directories in .nx directory in your home. The session directories are used to store logs, statistics and other relevant files. An initial C means that the directory is storing X-client-proxy files. S means that the directory is storing X-server-proxy files. Letter n is the display number. In the previous case it was 8, so you should find a C-8 and a S-8 directory in your .nx.

Naming follows the X convention. The Client is usually remote, where X Clients are running. The Server is local, where the X server is running.

Persistent caches are created in directory .nx/cache. Multiple cache directories can exist, according to an arbitrary session type, provided by the user or NX server at session start-up.

For nxproxy configuration there are many parameters available. Given that the nxproxy tunnel is presently created as part of the agent startup, the parameters are intended to be passed on the nx display (f.e. nx/nx,link=wan,cache=4M), not on the command line. Use of the '-' switches in nxproxy, except -S and -C, is strongly deprecated. All the available parameters are described with detailed comments, in the ParseEnvironmentOptions() code in nxcomp.

Recent versions on NX pass the parameters in a 'options' file, into the .nx/C|S-session_id directory created at session startup. nxagent gets nxproxy parameters using a X11 protocol pseudo-extension (an extension that, for now, is only targeted at communication with the underlying NX proxy system and is not part of the X11 standard). The interface is described in the nxcompext library and it is only available to applications. nxagent (as well as the other agents) adapts its internal operation according to parameters reported by NXGetControl- Parameters(). For example, to get PNG image encoding instead of JPEG, you can run nxagent as in the following example:

> export DISPLAY=nx/nx,link=modem,pack=16m-png:8

> nxagent :8

Or:

> cd $HOME

> mkdir .nx/C-8

> cat >.nx/C-8/options

link=modem,pack=16m-png:8^D

> export DISPLAY=nx/nx,options=/home/test/.nx/\

C-boezio-1068-A2360A325FA58AC249B9F893B20D7BED/options:1068

> nxagent :8

Options are better passed in a options file as the parameters can change along the life of the session, for example if session is reconnected at later time on a different computer.

1.3 Getting X protocol statistics

X To get a detailed report of X compression statistics do the following:

> cd ~/.nx/S-8

> ps auxw | grep nxproxy

lele 8899 0.6 1.3 7645 6824 ? S 14:18 2:32 nxproxy -S ...

> kill -USR2 8899

> more ~/.nx/S-8/stats

Sending a USR1 signal we tell nxproxy to produce total statistics, i.e. statistics starting from the beginning of the session. By sending a USR2 signal, we let nxproxy produce partial statistics, i.e. statistics starting from the last time a USR2 signal was delivered.

By looking at statistics you will find that even with compression ratios in the order of 50:1, your remote session will still run very slowly. If you look at the bit-rate, running over a modem link, you will find that it rarely exceeds the 500 bytes per second. This is due to the fact most X clients use a large number of requests needing a "reply" from the X server. Such requests are marked with "R" in the NX statistics.

At any request needing a reply, nxproxy must stop compressing the traffic and wait for a response from the remote server. This is called a "round-trip". Round-trips are very expensive on links having a high latency. On a modem link, every round-trip requires nearly 200 milliseconds. If an X client requires 5 replies and nxproxy compresses these

requests and replies at a rate of 1 byte each, you will get a final bit-rate of 10 bytes-per-second.

You can observe thousands of replies even running simple X applications. As you know, roundtrips degrade X performances, regardless of the ability of nxproxy to compress the traffic and regardless of the speed of your X server.

1.4 Building NX X libraries and the X session agent

To get the best from NX X compression technology, you will need nxagent. This is a virtual X server, running with the remote, used to embed and deploy X desktops.

To compile nxagent you need:

nxcomp-X.Y.Z-N.tar.gz
nxcompext-X.Y.Z-N.tar.gz
nxproxy-X.Y.Z-N.tar.gz
nx-X11-X.Y.Z-N.tar.gz
nxauth-X.Y.Z-N.tar.gz
nxagent-X.Y.Z-N.tar.gz

Go to your ~/NX directory and untar, from there, all the previous files. nx-X11 is a modified XFree86 distribution, so, as usual:

```
> make World
```

This will also build nxcomp, nxcompext and all the required libs in nx-X11.

nxagent is usually run by leveraging Xcomp's capability to provide transparent X compressed transport. nx-X11 distribution provides a new Xlib X transport layer with a new semantic of X display.

You can run:

```
> ./nxagent -display nx/nx,link=modem:1000
```

In this case the modified Xlib, linked by nxagent, will create a proxy process that will listen for connection from a proxy peer running on the user's client. In this way any X client can be NX-ized. Local (thin client) proxy is run with:

```
> ./nxproxy -S remote_host:1000
```

The local nxproxy will connect to the nxagent process and forward the X traffic to the current display.

Special NX-ized clients, called "agents", were developed by NoMachine to handle X, RDP and RFB sessions. The agent providing X transport of X session is nxagent. This is based on the well known Xnest "nested" server. nxagent, like Xnest, is an X server for its own clients, and at the same time, an X client for the real X server. The main scope of nxagent is to eliminate X round-trips or transform them into asynchronous replies. nxproxy does not make any effort to minimize round-trips by itself, this is demanded of nxagent. Being an X server, nxagent is able to resolve locally all the property/atoms related requests, ensuring that the most common source of round-trips are nearly reduced to zero.

As we have seen, good compression would be useless without a comparable effort dedicated to reducing the X round-trips. Single applications running through an nxproxy tunnel cannot currently leverage round-trip suppression offered by nxagent. A rootless nxagent (that is a nxagent that is able to mix its own windows with windows opened by local clients on the real X server) will be made available in the future.

Concerning the original Xnest, distributed by the XFree86 project, NoMachine has implemented significant improvements. By integrating frame-buffer functionalities, we have eliminated many GetImage requests that were needed by original Xnest. A lot of work was devoted to font handling to solve the problems you can find described in the Xnest main page.

nxagent implements clipboard integration with the host X server and internal handling of expose events. This last feature permits nxagent to greatly speed up repaints as it does not have to wait for expose events coming from the real X server.

nxagent can be considered a better Xnest and it is, infact, able to run much faster than Xnest, even without X compression functionalities provided by other NX components.

Xcompext library is used to tie together NX agents with compression capabilities of Xcomp/nxproxy. Xcompext adds some Xlib functions and some primitives to the X protocol that let agents compress images and perform X protocol operations in a way that helps Xcomp to carry out its job.

For example, Xcomp provides streaming of big images in small chunks, so nxagent can put demanding clients to sleep, and wake them up as soon as images (or other big requests) have been completely transferred to the remote side. These interfaces are implemented in Xcompext. This makes it very simple to develop further agents leveraging similar functionalities.

1.5 Building RDP and RFB agents

Besides X support, built in nxagent, special X clients provide support for RDP, the protocol used by Windows Terminal Service, and RFB, the remote frame buffer protocol powering VNC. These agents are based on RDesktop and TightVNC.

nxdesktop and nxviewer run on the remote terminal server. They act as gateways, translating RDP and RFB protocol requests to X-Window requests. This brings a significant advantage as only an X server and the NX compression software needs to be installed on the clients. Updating software to a newer version of the agent on a server handling hundreds of clients is just a matter of copying a file. No modification is needed to client software.

RDP and RFB protocols are known to be very network efficient, at least on reasonably fast network connections. Their compactness and simplicity comes from the fact that they make large use of bitmaps to encode screen updates. By comparison, X is a much higher level protocol. It uses abstractions like Graphic Contexts and Pixmaps to avoid bitmap encodings. X encourages developers to revert to image encoding only when strictly needed.

nxdesktop and nxviewer use NX compression techniques to compress, cache and transport RDP and RFB screen updates (mostly images) in their original format to the client. Screen updates are then decompressed and translated in X protocol by NX proxy toward the X server. There is almost no overhead in respect of running rdesktop or vncviewer directly on the client, with the advantage of leveraging NX compression and the higher level X protocol primitives, when possible.

nxdesktop and nxviewer can reach compression ratio ranging from 2:1 to 10:1, compared with the bandwidth usage of the same session run using the original RDP or RFB protocols.

To compile nxdesktop and nxviewer you need all the previous packages, more:

nxdesktop-X.Y.Z-N.tar.gz
nxviewer-X.Y.Z-N.tar.gz

These two packages are derived from RDesktop and VNC viewer and they follow their original build procedure. Untar in NX directory and run:

> cd nxdesktop

> ./configure

> make

>

> cd nxviewer

> make World

As usual, you can find the run-nxviewer and run-nxdesktop sample scripts in the nxscripts package.