Teleporting in an X Window System Environment

Tristan Richardson,* Frazer Bennett,* Glenford Mapp,* and Andy Hopper*†

* Olivetti Research Laboratory
Old Addenbrooke's Site
24a Trumpington Street
Cambridge
CB2 1QA
United Kingdom

† University of Cambridge
Computer Laboratory
Pembroke Street
Cambridge
CB2 3QG
United Kingdom

November 1993

Abstract

Teleporting is the ability to redirect a windowing environment to different computer displays. This paper describes the implementation of a teleporting system developed at Olivetti Research Laboratory (ORL). We outline two particular features of the system that make it powerful. First, it operates with existing applications, which will run without any modification. Second, it incorporates sophisticated techniques of personnel and equipment location which make it simple to use. Teleporting may represent a development in attempts to achieve a ubiquitous, personalised computing environment for all.

1 Introduction

In the near future, communication networks will make it possible to access computing services from almost anywhere in the world. In addition, the number of computers readily available within an office is increasing. We would like to allow individuals to make use of such networked computing facilities as they move from place to place, whilst retaining the familiarity of their own computing environment.

A windowing environment offers a coherent means of interacting with a computer via a screen, keyboard and mouse. Typically, a user would run any number of applications within such a windowing environment. Traditionally, these applications have been bound to one display, offering output at and accepting input from only one fixed point within the building (i.e. the display on which they were started). To make use of the abundance of computing facilities within the office, we would like a means of causing the application's output to disappear from one display and re-materialise on another, as we move around the building.

Moving the windowing environment around in this way is described as *teleporting*. In this paper we examine some issues that surround teleporting, and describe a teleporting system for an X Window System[1] environment. We demonstrate that a means of locating

[1]The X Window System is a trademark of the Massachusetts Institute of Technology

people and equipment within an office becomes an important part of a workable teleporting system.

To begin, we summarise a few basics of the X Window System. Section three examines previous work which has been done on teleporting and related issues. Section four presents the teleporting system that has been developed at ORL. In Section five we describe the location facilities available at ORL and their interaction with the teleporting system. In conclusion, section six summarises our experiences in the use of teleporting, and outlines some pointers to further research in this area.

2 The X Window System

This section briefly describes the X Window System [Scheifler92], and in particular those aspects which affect its use as a platform for teleporting.

2.1 Basic architecture

X is a network-transparent window system using a *client / server* model. A *display* (consisting of a keyboard, mouse and one or more screens) is controlled by an *X server*, and applications are *clients* of this server.

X clients communicate with the server using the *X protocol* over a reliable byte-stream such as that provided by TCP/IP [Postel81.1] [Postel81.2]. A computer may have a number of displays, each with its own X server. The servers will accept connections from X clients on their own well-known TCP port.

Usually, when an X server starts up, a login window is presented. A user begins an *X session* by logging in and starting X clients. The X server offers a *root window* on which the windows of the clients may be arranged. A normal X session consists of many clients, including one special client - the *window manager*. The window manager takes responsibility for arranging the windows of other clients on the root window and provides a mechanism for manipulating these windows.

X servers control what appears on the screen of a display by means of a directly writeable portion of video memory called a *framebuffer*. The number of bits used to represent the value of each pixel is called the *depth* of the framebuffer. Monochrome displays use a 1 bit deep framebuffer, while full colour displays offer a framebuffer up to 32 bits deep. A *colormap* is used to determine the colour and intensity corresponding to each possible pixel value. Some servers have a single, fixed colormap, while others allow writable colormaps where the appearance of a given pixel value can be changed without writing to the framebuffer.

2.2 The X protocol

The X protocol consists of several types of messages sent between client and server. A connection between a client and an X server is established with a *connection setup* packet sent by the client, followed by a reply from the server. Once connected, the client sends *requests* to the server, to which the server may send a *reply*. The server also sends the client *events* (for example when a key is pressed) and *error* packets, when appropriate.

A client creates and manipulates abstract objects within the X server called *resources*. Windows, cursors, fonts, colormaps, graphics contexts and pixmaps are all examples of resources. The server maintains *state* about the client in the form of the current values of the *attributes* associated with these resources.

Although all X servers use the same protocol, their functional characteristics may be significantly different. Such differences include framebuffer size and depth, the fonts available, and the physical characteristics of screens and keyboards. These differences do not affect simple clients, but more complex ones may adjust their behaviour to suit the exact characteristics of the server to which they are connected.

2.3 Security

Under normal circumstances, a display and its X server are used by one individual who may run clients from a variety of host computers *(hosts)*, each connecting to this server. It is not desirable, in general, to have clients of other users connected to the same server. With a connection to a server, a client has the ability to kill off other clients, as well as snoop on other clients' input and output. It is clearly important to have control over who has access to the server in this way. There are two common authorisation methods which are used to control connection to a server by clients.

In the first method, the X server keeps a list of hosts from which it will allow connection. If a host appears in this list then any user can run clients on that host which will be able to connect to the server. This scheme has obvious disadvantages - access control is on a *per-host* rather than a *per-user* basis.

The second system introduces per-user authorisation by requiring each client to present to the X server a capability known as a *magic cookie* upon connection. Only clients which present the correct value of this cookie will be allowed to connect. When a user begins an X session, a copy of the cookie is stored in his/her home directory. From here it can then be accessed as necessary by any of the user's X clients. If the home directory is accessible across many hosts, then the user will be able to run clients on any of these hosts, and they will all have authorisation to connect to the server. In addition, as long as the file system is secure, other users will not be able to read the cookie, and their clients will not have access to the server. When the user ends the X session, a new cookie for the server is created, and the old one becomes obsolete.

3 Related Work

There are two approaches which can be taken towards providing a mobile windowing environment. The first involves developing new applications which upon request can redirect their input and output between different displays and thus are aware of their own mobility. This would typically be achieved with the aid of a programmer's toolkit which implements the mechanics of the redirection.

The second approach is to take existing applications which know nothing of mobility and somehow redirect their input and output without their knowledge. In X this can be achieved by introducing a level of indirection between the client and server (usually called a *proxy server*). The proxy server is responsible for redirecting the client's input

and output between different real X servers.

3.1 The toolkit approach

The first approach is that taken by the *Cedar XTk toolkit* [Jacobi92] and *Trestle* [Manasse93]. Applications written using these toolkits can be told to move their windows from one display to another.

The advantage of this approach is that, since the application is responsible for redirecting its output, it can make intelligent decisions about what to show on different displays. For example, when the application is redirected to a smaller screen, non-essential parts of its output may be removed and only shown when explicitly requested by the user.

The major disadvantage of this approach is that there is no method of coping with existing applications which are not written using the toolkit. Another problem is that this approach lends itself best to redirecting applications individually. Redirecting a whole windowing environment such as an X session, with several clients including a window manager, may be more difficult to achieve.

3.2 The proxy approach

The second approach requires that a proxy server be inserted between the clients and the X server. To a client, the proxy server looks exactly like a real X server. To the real X server the proxy server appears to be a normal client (or several clients). Upon request, the proxy server is able to change its connection from one real X server to another, whilst maintaining the connection to its clients. The activities of the proxy server are *transparent* to both server and client.

The advantages of this approach are that it works for existing applications without any modification, and that it lends itself naturally to redirecting a windowing environment as a whole, including the window manager.

The disadvantage is that there is a potential conflict caused by the difference in size, framebuffer depth, input devices, etc. of particular displays. It may be difficult for a proxy server to provide a client with a consistent view of the kind of server it is connected to when the underlying server may change without the client's knowledge.

A number of projects in the area of Computer Supported Cooperative Work (CSCW) have involved proxy servers. These proxy servers provide a means of directing a client's output to several X servers simultaneously, thus allowing windows to be shared between displays. Examples of window-sharing proxy servers are *shX* [Altenhofen90] and *XTV* [Abdel-Wahab91]. A C++ library for constructing proxy servers is described in [Menges93].

3.3 Discussion

If one were designing a windowing system from scratch which allowed applications to move between displays, undoubtedly the redirection mechanism would use an approach which allowed the application to be aware of such redirection.

However, given that there is already a large base of existing X applications, and given the

natural way in which a proxy server encapsulates a whole windowing environment, the proxy approach was chosen. Writing a toolkit together with enough applications to make the system genuinely useful was not considered practical. A proxy server that allowed unmodified X clients to be used in a teleporting environment represented a more suitable approach.

The proxy servers mentioned above have a number of deficiences which make them unsuitable for teleporting. Although these proxy servers can manage connections to several X servers simultaneously, they do not treat these servers equally. There is always a *master server*, the characteristics of which the proxy server adopts. When a client connected to the proxy server requests information about the proxy server's characteristics, the information provided reflects the characteristics of the master server. This scheme makes the job of the proxy server fairly simple, but prevents the proxy server from ever removing its connection to the master server.

In the design of a teleporting proxy server, such an arrangement is unsuitable. We do not want a teleporting proxy server to be bound to any real X server, but rather it should be able to move its connections freely from one server to another, or remove them altogether. To achieve this, a teleporting proxy server must store its own set of display and state information, providing a generalised, consistent view to its clients.

Another problem with the existing proxy servers examined is that none had considered how to cope with a window manager and its interaction with other clients. The window manager presents problems because it is given privileges to influence the windows of other clients. In the light of these considerations it was decided to write a teleporting proxy server from scratch.

4 The Teleporting System

The teleporting proxy server developed at ORL provides the mechanism for moving a windowing environment between displays. We discuss the architecture for the system as well as describe some of the problems that have been tackled in its implementation.

Our implementation of teleporting is specific to the X Window System, exploiting the client/server model that X adopts. The notion of teleporting within other windowing systems (for example Microsoft Windows [2]) or between different windowing systems is a perfectly plausible one, although the mechanics of such systems are beyond the scope of this paper.

4.1 System architecture

An outline of the structure of the teleporting system is shown in Figure 1. At the centre is the teleporting proxy server, *tpproxy*. Tpproxy accepts connections from X clients, appearing to these to be just like a real X server. The proxy server has a root window, just like a real X server, and the clients of the proxy server may be arranged on this window. The proxy server can make its root window appear on the root window of a real X server. *Teleporting* occurs when the proxy server moves its root window from the display of one X server to that of another.

[2]Windows is a registered trademark of The Microsoft Corporation.

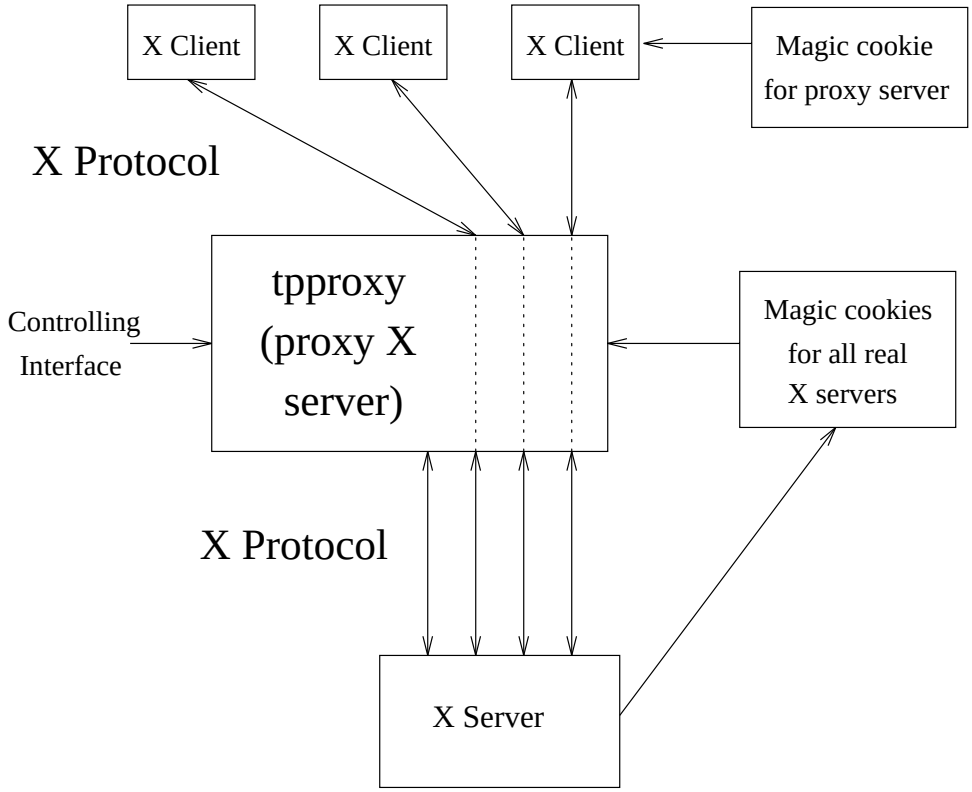

Figure 1: Teleport System Structure

The proxy server's root window may occupy the entire root window of the real X server. Alternatively, if there is already an X session in progress on the real X server, it is possible for the proxy server's root window to occupy just a portion of the real X server's display. This makes it possible for more than one proxy server to be connected to the same X server at any time. This is much like the virtual screen program of [Lin92].

The proxy server is controlled by a simple textual interface through which it is possible to instruct it to move its root window from one X server to another. This is done by instructing the proxy server to *teleport* to a named server, or *teleport* nowhere, when the root window of the proxy server disappears altogether. However, this interface is still fairly awkward since it requires users to know the names of servers when they wish to teleport, as well as offer some means of authenticating themselves.

4.2 Security

Authorisation for connection by a client to the proxy server is achieved using the magic cookie authorisation technique described earlier. The proxy server has an associated magic cookie, which it places in the user's home directory from where X clients will have access to it. On presenting this cookie to the proxy server, they will be allowed to connect to it. The proxy server also supports the *host list* authorisation mechanism but this is not normally used.

Authorisation of the proxy server by the real X server is more subtle. Within an administrative domain it is desirable for the proxy server to be able to connect to any real server,

whether someone already has an active X session on that server or not. There are two solutions to this problem. Either we remove the need for authorisation altogether, or we run the proxy server in some privileged mode which allows it access to all the servers.

The first solution can be implemented using the *host list* authorisation scheme. Proxy servers are run on a limited set of hosts and all X servers have this set of hosts in their host access list. This has the consequence that any user on one of these hosts may run X clients which will have access to any of the servers. Another disadvantage is that this host list must be manually maintained. If we wish to add a new host on which proxy servers can be run, this host must be separately entered into the host list for every server.

The second solution is clearly more desirable. The technique we have implemented is based on magic cookies and attempts to compromise security as little as possible. Each X server is configured so that its magic cookie is placed into a well-known file, shared across the network. This file is accessible only by proxy servers, enabling them to have connection authorisation to any of the X servers at ORL. Other than the proxy server, no client has access to this set of magic cookies. This solution is effective within our administrative domain.

However, this is clearly not an appropriate authorisation technique for implementing teleporting on a global scale, as it would require a globally accessible file in which to store magic cookies. A different authorisation structure must be contemplated to solve this problem. At ORL, we have experimented with an X server which offers *global login*, allowing, by means of an appropriate password, a proxy server from any remote site to connect to it.

4.3 Proxy server design

During normal operation, the proxy server acts as a bidirectional filter, receiving requests from a client and translating them, before passing them to the real X server. In the reverse direction, it takes replies, events and error packets from the server, similarly translating them and passing them on to the client (see Figure 2 - only a single client is shown for simplicity). In addition, the proxy server records all changes to the state inside the server relevant to its clients. This is done by keeping track of all the resources that these clients have created, and any modifications that they have made to them.

When told to *teleport* to a new display, the proxy server must perform a number of additional tasks, as shown in Figure 3. It replaces the connection to the current X server, if there is one, by a connection to the new server. Initially, the new server knows nothing of the resources which the proxy server's clients may have created. The proxy server has to generate requests in order to bring the new server to a state where each client's resources have been created and exhibit the correct attributes. Once this has been done, the proxy server can return to simply filtering requests and replies.

There is a complex web of possible interdependencies between the resources that clients may create. It is important for the proxy server to re-create these resources and set their attributes in the correct order. Most of these issues were addressed in *XTV* in the context of allowing late-comers to join a window-sharing conference [Chung91]. However, the problem was simpler for XTV, as it is for all window-sharing proxy servers, because of the use of a *master server* which the proxy server uses as its store of some attributes of the clients' resources.

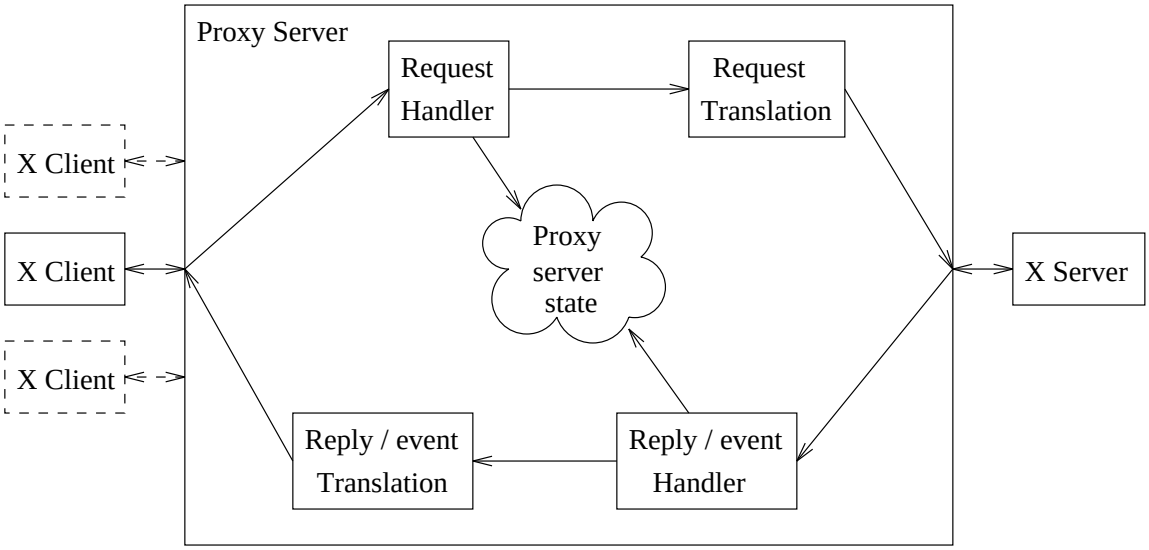

Figure 2: Proxy server in filter mode

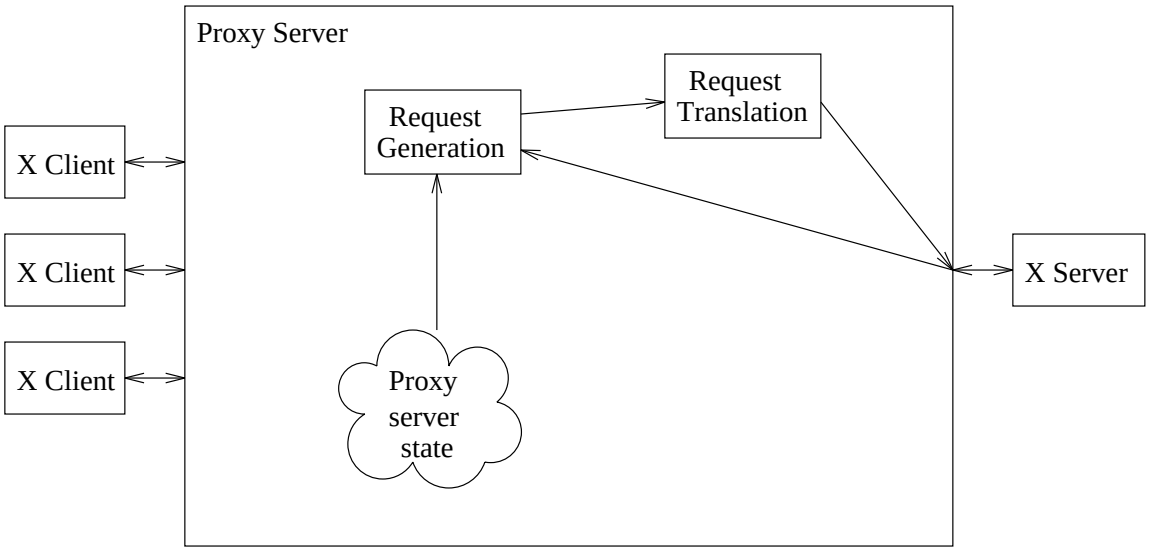

Figure 3: Proxy server in teleporting mode

4.4 The teleport session

The teleport *session* consists of the proxy server, *tpproxy*, and its clients. When connected to a real X server, the proxy server's root window appears encapsulated as one normal window on the screen of the real X server. Within this window the user can interact with clients just like a standard X session. Note that other clients of the real X server to which the proxy server is connected will be shown as normal.

It is possible to instruct the proxy server to move the output of its clients to an X server that no-one is currently using. In this case the only existing window on the X server's display will be the login window. The proxy server removes this window, taking over the whole of the screen, and appears just like a normal X session. When the proxy server's output is sent away, the login window will reappear. The ability to take over an unused display proves very important when using location-controlled teleporting.

The time taken to bring up the teleport session on a new server depends on a number of factors - how many clients there are, the network load, and the speed of the real X server. We have found that it takes only a couple of seconds for an environment with five or ten clients to be re-directed to a new server. Note that this is substantially faster than logging in and starting up the clients from scratch.

4.5 Variation between X servers

Many of the differing characteristics of X servers are not hidden from their clients. This causes a problem for a proxy server which must hide these differences from its clients. It must present its clients with a consistent set of characteristics at all times, whilst mapping these onto the differing characteristics of the real servers. For window-sharing proxy servers, the characteristics presented to the client are usually those of the *master server*. If a client makes use of characteristics which are incompatible with those of any server other than the master server (for example using a writable colormap), it will not be possible for the window-sharing proxy server to distribute the client's output to that server correctly.

In the case of a teleporting proxy server, there is no master server. *Tpproxy* has to invent and present a consistent set of characteristics to all clients. These characteristics must be chosen to be as general as possible so as to allow them to be mapped onto the characteristics of a wide range of real X servers. We discuss below the way in which some specific problems in this area have been tackled.

4.5.1 Framebuffer depth and format

The differences in framebuffer depth between different servers must be tolerated by the teleporting proxy server. The proxy server appears to its clients to have a framebuffer with 32 bits per pixel (the maximum allowed by the X protocol). The proxy server translates this framebuffer depth to and from the various depths required by real X servers. This works well on most colour servers. In the case of a monochrome server, where only black and white are available, applications which make use of several colours may appear unsatisfactorily. This problem can usually be avoided by configuring the applications to make sure that important contrasting colours will map to black and white.

Some clients send and receive image data to be loaded to and from the framebuffer directly. Such data must be translated by the proxy server between the format it has declared to the client and the particular format required by the real server. This translation process operates on each pixel individually so it can be somewhat time-consuming. Some clients which manipulate images directly can be noticeably slower under a proxy server.

Another problem can be caused by clients which use logical functions such as exclusive-or on pixel values. This is often done to cause particular pixels to change colour temporarily. Such operations may not translate correctly through the proxy server, and unpredictable colours may result. In the worst case, the resulting colour would render the object indistinguishable from its background.

4.5.2 Screen size

The proxy server tells all clients that the screen has a fixed size (in pixels) which can be specified on startup. When teleported to an X server whose screen size is smaller than this (or when the user shrinks the proxy server's root window), the window shown on the screen appears as a *viewport* offering a partial view onto the proxy server's root window. It is possible to move this viewport over the proxy server's root window using the mouse. This feature is similar to the facility provided by virtual desk-top window managers such as *tvtwm*.

Often the user may want certain windows such as an icon manager and a clock to always be visible through this viewport. This effect is achieved by marking such windows as *sticky windows*. As the viewport is panned over the proxy server's root window, these windows remain in fixed positions on the viewport. If the viewport is resized, sticky windows remain at the same position relative to the corner of the viewport nearest to them.

5 Location-Controlled Teleporting

The mechanisms we have outlined for controlling a teleporting proxy server are sufficient to make such a system useful, but, as previously mentioned, awkward to use. Integration of teleporting with the *Active Badge Location System*[3] [Want92.1] [Want92.2] [Harter94] has introduced a level of automation to this procedure, making teleporting a trivial activity - far easier, indeed, than using X servers in a conventional manner.

In this section we discuss this integration and profile some of the ways in which teleporting has benefited from it.

5.1 The Active Badge system

The Active Badge system provides a means of locating individuals within a building by determining the location of their Active Badges. This small device worn by personnel transmits a unique infra-red signal every 10 seconds. Each office is equipped with one or more networked sensors which detect these transmissions. The location of a badge (and hence its wearer) can thus be determined on the basis of information provided by

[3]Active Badge is a registered trademark of Ing. C. Olivetti & C., S.p.A.

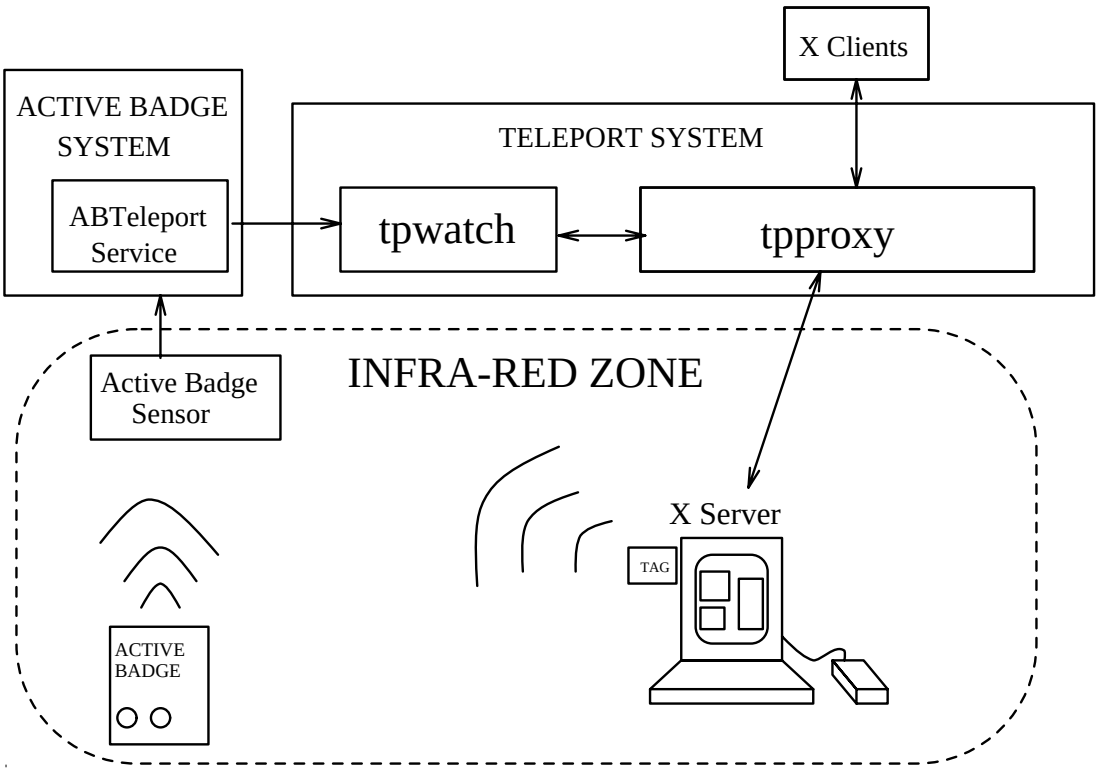

Figure 4: Location-controlled teleporting

these sensors. The badge also has two small push-buttons which when pressed cause it to transmit immediately. The buttons on the badge can be used to convey simple input information to the system.

A simpler version of the Active Badge with a longer battery life has been developed for tagging equipment. The *Equipment Tag* has been widely deployed at ORL including being used to tag all of the X displays in the building. When an Active Badge is detected in an office, it is possible to deduce which X displays are also in that office.

The Active Badge System can be used, then, to provide a sophisticated location service to the teleporting system. The buttons on the badge act as a simple interface to the proxy server, and a dynamically changing database of equipment provides information about the location of the X displays within the building. The interaction between the proxy server and the Active Badge system is outlined in Figure 4. We discuss below some of the components of this system.

5.2 The abteleport service

The Active Badge system offers *abteleport* as a service to the teleporting system. This service provides the necessary information to make it possible to control a proxy server using an Active Badge. It does this by providing the proxy server with notification when a user presses a button on his/her badge, as well as when they change location. The indication of when a user changes location is provided to make it possible for the user's teleport session to disappear when the user leaves the room. The information provided by

abteleport includes a list of X displays which are at the user's current location.

5.3 Tpwatch

Tpwatch is the process which controls *tpproxy*. It does this by interpreting the Active Badge sightings received from *abteleport*, using the textual interface to tpproxy to switch the teleport session between different displays.

Pressing the badge's middle button initiates a selection procedure in which the user can choose an X display within the room on which to teleport. With successive presses of the middle button, X displays in the room will beep and the border of their screens will flash one after the other. Upon reaching the desired X server, pressing the badge's side button will confirm the selection, and teleporting will take place. The proxy server will remove its output from the display with another press of the middle button, or if the badge is seen outside of the room. It is clearly possible to impose further heuristics on the choice of display and the method and order of selection based on user preference and the suitability of each display.

6 Conclusion

We conclude with a summary of our experiences with the teleporting system at ORL, and focus some of the ways that this notion of a portable computing environment may impact future issues in mobility.

6.1 Experiences

A complete teleporting system has been in everyday use at ORL since September 1993. For some, it has completely replaced the standard computer working environment, and for others it represents an invaluable complementary tool.

Immediate acceptance came about because of the way that the proxy server eliminated the traditional login and setup procedure as well as offering the convenience of a truly mobile desk-top. Nearly all X applications run satisfactorily in a teleport session. Loss of performance is only apparent in certain clients which manipulate image data directly, and even then the loss is usually quite acceptable. The only other fundamental problem is with applications using features which are unavailable on a particular real server - such as a writable colormap.

Aspects of collaborative work have benefited from the system. It is now much easier to share computing resources and ideas when one's desk-top may be called to appear on any X display in the building.

6.2 Further Work

We expect to improve our teleporting system in several ways. First, we will tackle some of the subtleties of more complex clients. It should be be possible to cope with clients which use features like a writable colormap so that they function correctly on real servers which have such a feature but also behave sensibly on a real server which does not. It should also

be possible to improve the performance of clients which manipulate image data directly, at least on real servers which themselves have writable colormaps.

Second, we are considering how applications which run through the proxy server can be made aware of where they are being displayed, in a similar way to the toolkits described earlier. This will hopefully combine the best of both worlds, allowing existing applications to teleport without modification, but allowing new, more complex applications to take advantage of knowing where they are being displayed.

Third, we hope to develop a mechanism for expanding the notion of teleporting to other computing devices. This might involve making it possible to *teleport* between different window systems, for example between X and Windows NT. [4]

We recognise two key issues in making full use of the wealth and variety of computing facilities available - authentication and configuration. An authentication mechanism provides a means of authorising the use of available facilities. Our authentication procedure has become fully automated by the use of the Active Badge. Configuration becomes necessary when we try to make use of differing pieces of computing equipment. Again, our configuration mechanisms have become automated by the use of equipment tags and the dynamic equipment database. The configuration process may be personalised with hints about a user's work patterns and preferences.

When the authorisation issue has been addressed, it will be possible to teleport between different sites around the world. At ORL a *teleworking* system has been in operation for three months, offering an ISDN connection between the office and the home. Through this connection, it is possible to teleport between work and home as easily as around the office.

6.3 Summary

Teleporting is simple, adaptable, and powerful. Simple, because it presents no change to a user's existing working environment whilst offering enhanced functionality which may be employed without effort. Adaptable, because it will transparently and intelligently accommodate the differences that exist between different computers. It is powerful since these additional features offer us a new paradigm for *mobile* and *nomadic* computing. With the integration of networked computing devices, people can be less dependent upon portable computing devices within the office environment. They will be able to make use of appropriate equipment that is already to hand.

7 Acknowledgements

We would like to thank Andy Harter and Damian Gilmurray for providing useful discussion during the development of the teleporting system. We are also grateful to those members of ORL who have been willing to experiment with the system.

[4] Windows NT is a registered trademark of The Microsoft Corporation.

References

[Abdel-Wahab91] Hussein M Abdel-Wahab and Mark A Feit. *XTV: A Framework for Sharing X Window Clients in Remote Synchronous Collaboration.* Proceedings of IEEE TriComm 91: Communications for Distributed Applications & Systems, Chapel Hill, North Carolina, April 1991.

[Altenhofen90] Michael P Altenhofen. *Using and Porting shX.* Digital Equipment Corporation, CEC Karlsruhe, Germany, November 1990

[Chung91] Goopeel Chung, Kevin Jeffay and Hussein M Abdel-Wahab. *Accommodating late-comers in a distributed system for synchronous collaboration.* Technical report TR91-038, Department of Computer Science, University of North Carolina at Chapel Hill, October 1991.

[Harter94] Andy Harter and Andy Hopper. *A Distributed Location System for the Active Office.* IEEE Network, Special Issue on Distributed Systems for Telecommunications, January 1994.

[Jacobi92] Christian P Jacobi. *Migrating Widgets.* Proceedings of the 6th Annual X Technical Conference, January 1992.

[Lin92] Jin-Kun Lin. *Virtual Screen: A Framework for Task Management.* Proceedings of the 6th Annual X Technical Conference, January 1992.

[Manasse93] Mark S Manasse. *The Trestle Toolkit.* Proceedings of the 7th Annual X Technical Conference, January 1993.

[Menges93] John Menges. *The X Engine Library: A C++ Library for Constructing X Pseudo-servers.* Proceedings of the 7th Annual X Technical Conference, January 1993.

[Postel81.1] Jon Postel. *Transmission Control Protocol.* RFC 793, September 1981.

[Postel81.2] Jon Postel. *Internet Protocol.* RFC 791, 1981.

[Scheifler92] Robert W Scheifler and Jim Gettys. *X Window System.* Digital Press, Bedford, Massachussetts, 1992.

[Want92.1] Roy Want, Andy Hopper, Veronica Falcao and Jonathan Gibbons. *The Active Badge Location System.* ACM Transactions on Information Systems, January 1992.

[Want92.2] Roy Want and Andy Hopper. *Active Badges and Personal Interactive Computing Objects.* IEEE Transactions on Consumer Electronics, February 1992.