THINCing Together : Extending THINC for Multi-User Collaborative Support

Dave Coulthart, Sudip Das, Leo Kim
Department of Computer Science, Columbia University, New York, NY 10027
{davec, sd215, lnk2101}@columbia.edu

Abstract

Commonly-found design elements in collaboration software such as remote display and basic screen-sharing demonstrate a functional intersection with thin-client designs. This intersection of functionality provides a motivation for incorporating a multi-user collaborative environment within a thin-client system. As a result, popular thin-client systems like VNC and Citrix MetaFrame have begun to incorporate collaborative features. In this context, we introduce "THINCing Together" which is an extension of the THINC thin-client protocol that allows for asynchronous and synchronous multi-user sessions. With THINCing Together, we are able to provide a platform for collaboration by implementing centralized cursor management and different modes of screen-sharing. In this paper, we show how we incorporated our design without compromising the thin-client functionality of THINC. We also discuss the issues we encountered with the current architecture of the THINC system while implementing multi-user support. Finally, we demonstrate how our design has a low impact on typical thin-client performance metrics with respect to bandwidth consumption, latency, and CPU utilization compared to a version of THINC without collaborative features as well as VNC.

1 Introduction

The wide acceptance of the Internet in the past decade has fueled the popularity of peer-to-peer and client/server architectures. Collaborative software has benefited from the growth of Internet-based communication by allowing users to collaborate interactively across different geographies. More recently, thin-client computing and remote display technology have received increasing attention[22] [23]. Coupled with progressively more powerful hardware and the rising availability of wide-area broadband networking, thin-client computing has become an enticing alternative to the more prevalent desktop-based computing that exists today.

The ability to view and interactively manipulate a shared resource from a remote location is a feature found in both thin clients and collaborative software. Thin clients are essentially dumb terminals that use a local framebuffer to display remote framebuffer updates. Likewise, many types of collaborative software such as Altiris Vision[1], Apple Remote Desktop[2], NetMeeting[7], Presence-AR Adapter for Excel and Powerpoint[25] and SubEthaEdit[8] provide this same facility to varying degrees. Moreover, some thin-client systems already incorporate collaborative functionality, notably Citrix MetaConferencing Manager[30] and, to a certain extent, Collaborative VNC [4] [29]. Clearly, collaborative features can be integrated with thin-client architectures.

In this paper, we present *THINCing Together*, which is an effort to integrate basic multiple user functionality with an existing thin-client protocol, THINC (short for *THin-client InterNet Computing*) [10]. THINC is a simple and scalable thin-client architecture capable of high performance in both LAN and WAN settings. It achieves high performance by using a simple low-level protocol and applying a range of optimizations that allows it to transparently and efficiently support existing user applications. A novel feature of THINC that separates it from other thin-client systems is that it is able to support real-time full screen video at full framerate.

Our approach to designing THINCing Together was to select a cross-section of some of the common features found in other thin-client-based collaborative systems. We also wished to address any architectural limitations with the existing implementation of THINC in order to enable these features. Compared to other thin-client systems that integrate collaborative features such as Collaborative VNC, we believe that THINCing Together is able to give the user similar functionality without significant modifications to the existing protocol. Most importantly, we prove that the THINCing Together requires no changes to existing thin-client features and adds marginal overhead over an implementation of THINC without our code additions. Our performance results show that a THINC user is capable of sharing a session with several users over a LAN showing

an MPEG video without dropping frames.

This paper is organized as follows. Section 2 discusses some related work in collaborative software and compares THINCing Together to thin-client systems with collaborative functionality. Section 3 gives a brief overview of the THINC protocol features. A description of issues encountered with asynchronous multi-user sessions is given in section 4. Synchronous multi-user session design and implementation is given in sections 5 and 5.2. We show some performance results in section 6 and conclude with suggestions for future work in section 7.

2 Related Work

Much work has been done in the field of CSCW (computer supported cooperative work) and the management of interaction between multiple users. In the designs surveyed for this paper, we got the impression that the means of control passing between users of a shared resource in most collaborative applications is motivated more by intuition than hard research. We found that the dominant theme among all of these designs was the need of some sort of policy to manage control. In particular, numerous ”floor-control mechanisms” similar to the one used by THINCing Together have been proposed to handle control of a synchronous task. Some approaches use a generalized manner where control of a shared application is likened to transaction concurrency [20]. Boyd introduces ”fair dragging” where a user gains control of the floor once the mouse is dragged [16]. Others are more specific to a particular application such as multimedia conferencing [15] [26], document sharing and whiteboarding [28] [14], or control handling among handheld devices [11] [13].

Many collaboration tools are commercially available as well. For example, Altiris Vision and Apple Remote Desktop are designed primarily for teachers and educators and offer screen-sharing, screen-supervising, and remote control of student computers. WebEx [37] is a popular web- and video-conferencing tool. SubEthaEdit is a text editor for Mac OS X that allows multiple users to edit the same document simultaneously by supporting multiple cursors and using different-colored text to represent each user. However, it does not apply any floor control policies and is restricted to sharing only the text editor. Similar functionality to SubEthaEdit is provided by Presence-AR Adapter as a plug-in for Microsoft Excel and Powerpoint. Microsoft NetMeeting is a popular Windows-based application that offers program and screen-sharing, though it does not allow single-user access to a remote desktop.

Still, there has been little in the way of evaluation of these floor-control mechanisms, and results proving the effectiveness of any approach have been conflicting. Greenberg discusses several but notes that ”surprisingly, there has been no attempt to evalute these different methods in existing shared view systems” [32]. Some have suggested that working in parallel (i.e. sharing control of a resource) is not optimal and that users prefer to take turns [33] [34], although at least one finding shows that using multiple cursors without blocking input worked best [12]. As a result, most collaboration tools differ on their approach to floor control in very subtle ways, reflecting disparities of approach to this problem.

For thin-client protocols, collaboration amounts to screen-sharing with floor control as the policy for managing control of the screen. One example of a collaboration-oriented thin-client system is Collaborative VNC. Collaborative VNC is a patch applied to the TightVNC server and client [35] that provides managed collaborative sessions over the RFB protocol. With Collaborative VNC, one user ”has the floor” (i.e. controls the desktop) at any given time. Other users have the power to take control from or give control to other users at any time. Every user’s cursor is displayed, with each cursor assigned one of eight colors. Although this is a nice feature, the necessary bandwidth requirements to maintain multiple cursors does not scale well to a large number of users. THINCing Together takes a single-cursor approach that optimizes bandwidth usage for multiple users.

GoToMyPC [6] and Citrix MetaFrame [9] are Windows-based thin-client systems. GoToMyPC is a web-based thin-client solution that uses a Java applet for display (though hosts can only run on Windows PCs) and offers some basic whiteboard and messaging tools, but it does not offer any collaborative session management beyond these tools. Citrix MetaFrame Access Suite provides collaboration functionality through the Conferencing Manager, which allows a user to initiate a conference on his desktop and invite other users to join the conference. Users can send private messages to other users, view file attachments, and launch multiple applications within the same shared session. Although both Citrix MetaFrame is a sophisticated thin-client/ collaborative solution, it relies heavily on Windows-specific applications such as Outlook for collaborative features whereas THINCing Together does not rely on any other applications to provide its collaborative features. Instead, these features are integrated directly with the protocol. Though THINCing Together does not have any peer-to-peer communication functionality, future additions to the THINC protocol to support this are imminently possible.

Several papers were written in the early 1990s on screen sharing with the X Windows system that describe implementations of shared screen functionality, multiple cursor operation, floor control, and different levels of sharing granularity (e.g. sharing a single window versus sharing a desktop) by modifying existing X libraries [18] [19] [17]. A number of these concepts have clearly made their way to contemporary collaborative designs, though we have not been able to locate a current version of the systems mentioned in the cited papers. One slightly more modern X-

based application is [39] which is an X client that multiplexes multiple X server sessions to multiple clients and has basic floor management functionality. Unfortunately, we were unable to get XMX to compile properly and the code does not appear to have been updated since 1999.

With THINCing Together, we sought to make the collaborative features as straightforward as possible. Floor control influenced our design in that it was a commonly-found in other collaborative systems and appeared to be the most intuitive. We note that there are many nuances and variants of floor control, and we emphasize that our design takes into account the fact that THINC is a thin-client protocol first and collaborative tool second.

3 Overview of THINC Protocol and Architecture

The design for THINCing Together has three primary goals. The first is to build support for multiple users to run their own remote THINC sessions on the same machine. The second is to allow multiple users to share the same remote THINC session in a cooperative manner. We use the term *asynchronous THINC session* to characterize the first goal, and *synchronous THINC session*, or *screen-sharing session* to describe the second[1]. The third goal is to maintain feature parity; that is, we did not want to change or modify existing features in order to support our design, nor did we wish to unduly impact performance. The multi-user features we sought to implement had implications for both the THINC protocol and architecture. Specifically, the ability to have synchronous sessions depended on the flexibility of the protocol, and enabling asynchronous sessions required a non-trivial architectural investigation. Thus, before going into the multi-user THINC design, some discussion of the THINC system is necessary.

The THINC server is implemented as a loadable module for XFree86 [5] on the Linux operating system to communicate with the X server at the device driver level. An Xlib THINC client interacts with the server using a predefined protocol, which is essentially a messaging specification for handling framebuffer and I/O device updates. During typical operation, the THINC server captures any updates made to the framebuffer on the remote machine. The server encapsulates these updates as a serial combination of messages and data which are sent to a THINC client. The client decodes these messages and displays updates on the user's local machine. The client also sends messages to the server typically in the form of keyboard and mouse events.

Although the protocol itself consists primarily of messages that communicate graphical updates, it is extensibly designed such that a large number of enhancements and optimizations are possible with additional messages[2]. For instance, in order to reduce bandwidth consumption, the protocol allows for a *hardware cursor* mode on the client side. In this mode, the server sends cursor image updates instead of constantly sending raw image data to redraw the mouse pointer each time the cursor is moved. The client continues to send cursor position information to the server. Though this optimization results in an overall reduction in bandwidth usage, it also resulted in some design complexity in accounting for cursor handling when multiple clients were connected. We discuss this in more depth in sections 5.1.3 and 5.2.3.

Overall, we found the THINC protocol to be flexible enough to enable synchronous multi-user support. We were quickly able to determine the basic desired features that would be possible for collaborative THINC. Adding these features to the existing THINC protocol was a simple exercise in defining client and server messages, and determining efficient byte sizes for these messages to keep bandwidth costs low. Our challenge for synchronous multi-user sessions was to come up with a usable design that would minimally impact the performance and normal operation of the current implementation. However, for asynchronous sessions we encountered different design challenges affecting the foundation of the THINC architecture.

4 Design and Implementation of Asynchronous Multi-User THINC

In order to provide tangible benefits over the traditional desktop computing model, a thin-client server must support multiple concurrent client sessions. Like most thin-client systems, THINC leverages the codebase and developer knowlege of a traditional windowing system such as the X Window System [38] to transparently support existing applications. The historical assumption in the design of windowing systems was the scenario of a single local desktop user, only interested in interacting with one screen and set of input devices at a time. XFree86, the dominant implementation of X, does not support concurrent interaction with multiple X servers on the same machine. In order for a THINC server to support multiple clients, it was necessary to identify the specific code within the X server that enforced this design decision and architect a solution to bypass it. Curiously, this proved to be a much more complex problem than it would seem on the surface.

4.1 Problem Description

By interacting with X at the driver layer, THINC is able to utilize interfaces standardized for compatibility while al-

[1] A clarification on terminology: Other research papers use the term "asynchronous collaboration" to mean that users are not interactively collaborating but are still communicating via email and other such methods at different periods of time. Our use of the word "asynchronous" refers to asynchronous *multi-user support* and not *collaboration*.

[2] Currently, a maximum of 255 messages on either the client and server side are supported, although more messages can be added with minimal effort

lowing it to take advantage of potential performance increases by any newly-developed X extensions. The difficulty with this approach for asynchronous THINC is that higher layers of the X server are left managing the interaction between the devices and the rest of the system. The policy of these higher layers is such that only the X server associated with the controlling terminal is permitted to interact with the video card and input devices. This prevents multiple processes from simultaneously writing to VRAM, generating a garbled display image, and saves the X server from having to deal with the complexities of multiplexing input. These assumptions do not apply in the case of the THINC driver, however. Instead, X data is translated into THINC protocol messages and transmitted over the network to remote clients and *not written to local VRAM*.

VNC [27] has faced similar problems interfacing with X. The current stable release (3.3.7) includes a heavily modified version of the XFree86 server known as Xvnc, which acts as a broker, translating between the X and VNC protocols. This version of Xvnc is based on the outdated XFree86 3.3.2, and porting Xvnc to XFree86 4.x in the current VNC development releases has been a non-trivial effort for the VNC developers. The new port still requires a non-trivial patch to be applied to the XFree86 code base in order to compile Xvnc. For THINCing Together, this was not an acceptable solution; we wanted to avoid any modifications to the standard X code that would make THINC difficult to deploy and jeopardize vendor support.

Some related work to address this issue was done by the Linux Console Project [21]. The project consists of the so-called "ruby" patch to support multiple X servers for the purpose of attaching multiple video cards, displays, and other input devices to a single machine, thereby allowing multiple users to run their own X servers simultaneously on the same host. The patch is applied to the Linux kernel and blocks calls made by the X server to disable PCI bus access when another X server is started. While users report that this patch is functional, the THINC driver is not actually interested in local PCI bus access because it does not interact with local devices and only needs to receive proper notification of events for generating THINC protocol messages. Even if the "ruby" patch did work for THINC, a better solution would be desired since the patch is specifically designed for Linux and is thus orthagonal to X's premise of multi-platform support. The acceptability of the patch within the Linux kernel community is also a concern because it tailors kernel code to a specific application - something that kernel developers adamantly discourage.

After determining that none of these alternatives would provide an adequate solution for THINCing Together, we decided that the next step was to tackle the X code ourselves. We set out to identify the code within the X server that was caused the first server to stop responding upon starting another server. Two salient symptoms on the first server after starting the second were that neither the display updated, nor would the cursor change when moving between windows. Key presses did not generate any display updates, though after examining debugging output we noticed that THINC was still receiving the key events. From this we concluded that only display refreshes were not occuring.

Upon further examination of the THINC debugging logs, we found that the most noticeable difference in the codepaths executed before and after starting the second server was related to the cursor. In both cases, `thincHandleMotionEv()` was being called, but `thincSetCursorPosition()` was not being executed after starting the second server. We proceeded to use a debugger to navigate X's heavy use of function pointers and found that a particular function, `xf86CursorMoveCursor()`, checked two boolean variables, `ScreenPriv->SWCursor` and `ScreenPriv->isUp`, to decide whether it should call the hardware cursor functions that invoke `thincSetCursorPosition()`. After examining the booleans, it was clear that after starting the second server, the first server had been modified so that `SWCursor` was set to `TRUE` and `isUp` was set to `FALSE`, resulting in the software cursor implementation being used instead of the desired hardware cursor code.

We looked through the file where `xf86CursorMoveCursor()` was defined, `xf86Cursor.c`, to see where these two booleans were being modified. One function that did so was `xf86CursorLeaveVT()`, which also called a driver specific `LeaveVT()` function. Another breakpoint-backtrace iteration through GDB revealed that this function was being called by `xf86VTSwitch()`, which the first X server executes after receiving a signal when the second server starts up. This function manages all of the changes to the state of the X server when changing virtual terminals and starting another server.

Our initial (naive) approach to bypass the `xf86VTSwitch()` function resulted in the second server no longer completing startup. We then analyzed each piece of `xf86VTSwitch()` to identify the variables critical to supporting multiple simultaneous servers and the function calls that were manipulating them would be necessary. Using the debugger to manipulate the values of `SWCursor` and `isUp` during execution resulted in a segfault in `thincRealizeCursor()` because it attempted to access members of a `NullCursor`. Reading through documentation and reference source code, this was inadvertently uncovered a bug in the THINC code because it enables the flag `HARDWARE_CURSOR_SHOW_TRANSPARENT` and must therefore handle the possibility of being passed a `NullCursor`.

After fixing this, there was no longer a segfault, but the cursor now turned transparent upon starting a sec-

ond server. The desired result was actually not to have `thincRealizeCursor()` passed a `NullCursor` at all. Another iteration through the debugger comparing execution paths before and after starting a second X server turned up the difference in `xf86CursorSetCursor()`. Before starting the second X server, the variable `infoPtr->pScrn->vtSema` was set to `TRUE`, resulting in a normal cursor draw. Afterwards, however, `vtSema` was `FALSE`, leading to the undesired `NullCursor`. Looking through `xf86VTSwitch()`, the reason for this became obvious: one of the last pieces to switching away from a VT was to set `vtSema` to `FALSE`. Upon disabling this change to `vtSema`, the change in cursor between windows worked as expected even after starting a second X server.

The next step was to determine how to get the display to update as well. Without this, it was unclear whether additional work would be necessary to also get keyboard and other mouse events to work as well. The function call within `xf86VTSwitch()` which showed the most promise of affecting this was `EnableDisableFBAccess()`. Using GDB one more time to follow function pointers, `xf86EnableDisableFBAccess()` was found to set the clip mask for the root window to 0, which meant that updates to the display were being masked out. Blocking this call to `xf86EnableDisableFBAccess()`, resulted in display updates being received by the THINC client. With this in place, it became clear that both keyboard events and mouse presses were still being processed, and it was in fact possible to run multiple concurrent X servers.

4.2 Solution

The problem was reduced to three specific functions/variables: `xf86DisableVT()`, `xf86EnableDisableFBAccess()`, and `vtSema`. We proceeded to see if it would be possible to override these functions and reset `vtSema` to `TRUE` without modifying the actual core X code. Luckily, the modular driver design meant that hooks were provided so that the driver could supply driver-specific `DisableVT()` and `EnableDisableFBAccess()` functions. The global scope of the `xf86Screens` variable also meant that it was possible to override these functions in the necessary manner.

For `DisableVT()`, the THINC driver simply calls `xf86EnableVT()`, which resets `isUp` to `TRUE` and `SWCursor` to `FALSE`, undoing the undesired changes `xf86DisableVT()` did before calling the THINC driver's `DisableVT()`. For `EnableDisableFBAccess()` a slightly different approach was taken, manipulating the global pointer to jump directly into the THINC driver's `EnableDisableFBAccess()`, which would then only call `xf86EnableDisableFBAccess()`, when the `enable` flag was set to `TRUE`, effectively blocking any call to disable framebuffer access.

The only remaining issue is the modification of the `vtSema` boolean. The modification of this flag within `xf86VTSwitch()` occurs after all calls to any functions that could be manipulated to point to THINC specific code. It should be possible, however, to check `vtSema` within another function such as the wakeup handler, and set it back to `TRUE` before missing any events. If this is not the case, it may be possible to submit a patch to XFree86 to make the change of `vtSema` to `FALSE` optional.

5 Synchronous Multi-User Sessions

5.1 Design

In designing synchronous multi-user sessions for THINC, we wished to give the user the flexibility to enable different levels of screen-sharing. Again, we also tried to make our design transparent with respect to THINC's remote display capabilities. For our design, we have three sharing policies, or *modes of operation*: *single-user*, *unmanaged* and *managed*. A description of these modes is as follows:

- *Single-user mode*. Only a single client is allowed to connect to a single THINC session at a time. Connection attempts by other clients are denied and reported to the user who is interacting with the THINC session. Obviously, this is useful for keeping screen sessions private.

- *Unmanaged mode*. In this mode, multiple clients can connect to a single THINC session. In addition, all clients can control the cursor and keyboard for a given session. The THINC server sequentially processes all input events and re-broadcasts them to the connected clients. This essentially gives the effect of an "anything goes" session where users (unwittingly) fight over control for the cursor.

- *Managed mode*. As in unmanaged mode, multiple clients can connect to a single THINC server. Control of the screen is handled by a *token-passing mechanism* (similar to the floor control mechanisms described before) where only a single client controls the cursor and keyboard during the screen-sharing session. The client in control of the screen is said to *possess the token*. This token can be passed to any one of the clients sharing the screen. Once client A passes the token to client B, client A relinquishes control of the cursor and keyboard to client B.

We selected these three modes for our design as they best captured the basic collaborative functionality seen in other thin-client systems. They also represent a foundation upon which further refinements can be built (e.g.

application-specific sharing, screen-area sharing, etc.) We also adopt a single-cursor model, though it is not beyond the realm of possibility to integrate multiple cursor support with the THINC architecture. As stated previously, Collaborative VNC allows up to eight cursors to be displayed in a screen-sharing session, but doing so consumes additional bandwidth since the VNC server needs to rebroadcast these mouse events to all connected clients in order to update their respective displays. On top of this, Collaborative VNC only allows a single cursor to send mouse button events at a time, which means that much bandwidth and CPU is wasted handling the cursor movements of other connected users. We believe we are better able to optimize bandwidth consumption using a single cursor, and we also believe that this more intuitively reflects the user-desktop interaction.

5.1.1 Unmanaged Collaboration

In unmanaged mode, any number of users can connect to a single THINC server and begin issuing and receiving messages. Mouse input events from all clients are queued by the server and broadcasted to the other connected clients. Keyboard events are also queued and sent to the X server, which in turn sends the event to the application in focus. If the THINC server is configured to handle a hardware-based cursor, it sends a special message to the connected clients indicating that the cursor should be moved to a different location on the screen using specified coordinates. Clients process these mouse events by relocating the cursor to the given coordinates. Otherwise, raw cursor updates are sent to each client.

An unmanaged synchronous session can be useful for quick collaboration involving a small number of users. However, an unmanaged session among many users for extended periods of time may be undesirable. Because the THINC server does not filter events from clients in this mode, users can negate, or ”cancel out” the cursor movements of other users who are moving the cursor at the same time. To allow for a more amicable collaborative environment, a managed session provides a better solution.

5.1.2 Managed Collaboration

Our design for managed sessions is modeled such that only one client controls the remote desktop at a time. We use the notion of a *token* to represent the control of the desktop, whereupon *ownership of the token* gives the client control of the desktop. The client in possession of the token is also called the *control client*. The control client can relinquish control by *passing the token* to another client. Clients that are not in possession of the token merely observe the actions of the the token owner and cannot send any events to the server. Table 1 details the messages used for managed collaboration.

Command	Type	Description
Init	Server	Upon handshake time, informs the client of the number of currently connected clients and the client in ownership of the token.
AddUser	Server	Informs already-connected clients of a newly connected client.
DeleteUser	Server	Informs already-connected clients that a client has disconnected.
PassToken	Client	Passes ownership of the token to another connected client.
PassTokenError	Server	Informs client of an error while attempting to pass the token.
NewMaster	Server	Informs already-connected clients that a new client has received the token.

Table 1: Protocol Messages For Managed Collaboration

A simple description of the normal flow of token passing is as follows. The control client selects a recipient client from a list of connected clients to pass the token to. The control client then sends a message to the server indicating that the token should be passed. The control client relinquishes control at this point. The server then sends a message to the recipient client to inform that it is the new control client. If the recipient client happens to disconnect before the control client’s list of connected clients is updated, the server sends an error message to the control client, and the token does not get passed (i.e. the control client regains control). Once the recipient client receives the message, it takes control of the screen, and the server sends a message to all other connected clients, including the previous client in control, of the new control client. If the control client should somehow lose its connection, the server automatically passes the token to the client that connected *after* the control client.

To facilitate token passing, a list of connected clients must be maintained. Each time a client connects to the server, the server sends a list of currently connected clients along with other initialization data. A header message for this list indicates the number of users connected along with the client currently in control of the session. If other clients are currently connected, then whenever a new client connects the server sends a message to these clients notifying them of the appearance of this new client. Clients then update their list of currently connected users respectively.

5.1.3 Cursor Modes

In order to support unmanaged and managed modes of operation, we also require a mechanism to manage the cursor. As stated previously, the THINC protocol supports the use of a hardware cursor. Having cursor updates handled by the client's hardware cursor saves the server from sending raw updates each time the cursor is moved. However, updates generated by other clients connected to the same session must be reflected on all screens in a synchronous session. Thus, cursor updates using a client's hardware cursor could potentially prevent the user from controlling her computer.

To capitalize on this bandwidth optimization and to provide control of the cursor for the user, we provide the ability to display a *software cursor* that is locally drawn by the client. The software cursor is treated by the server in precisely the same manner as the hardware cursor except it is manually drawn by the client. In addition, we introduce two additional cursor states, *attached* and *detached*, to represent whether the cursor is handled automatically by hardware or drawn by software. When the cursor is attached, the client uses the hardware cursor to interact with the remote framebuffer. When the client is detached, the client frees up the the hardware cursor for the user to control. Framebuffer updates occur as they normally do, and cursor updates are redrawn manually by the client. The user is able to detach her cursor, i.e. enable the software cursor, by simply hitting a pre-defined key combination. To attach her cursor, the user need only click the screen.

5.2 Implementation

The synchronous THINC implementation integrates with most of the aspects of the existing THINC architecture, both on the client and server side. We present details of the implementation in the following sections.

5.2.1 Initialization

The mode of collaboration is set by an option specified in the XF86Config file. At server startup time, configuration settings are parsed by a function called `xf86ProcessOptions`, and the mode is set based on the value provided by the collaboration option (either single-user, unmanaged, or managed). If the option is not set, then the server defaults to single-user mode. The server notifies all clients of the collaboration mode during a handshake phase which takes place upon connection initialization. Clients that connect to the server after this point are then set in this mode throughout their session with the server.

It is possible to allow clients to dynamically change the mode of collaboration during a given session. However, this poses security risks since a user can potentially lose control of her desktop to another user if that "collaborator" decides to change the session from managed/unmanaged mode to single-user mode. There are several issues to consider here, and we discuss them in our future work section (section 7).

Clients supply the login name of the connecting user by calling `getpwuid` and sending it to the server at handshake time. This login name is used to provide a meaningful representation of connected users, though it is possible to provide functionality where user-defined name can be set. The server also records the IP address of connecting clients using the `sockaddr_in` structure upon connection accept. Since multiple clients can connect from the same IP address, the server also assigns a unique 16-bit integer to identify each connected client.

Individual client connections are managed by the server in a data structure containing client state such as framebuffer and cursor location information. The server manages multiple client connections by keeping a linked list of these structures. For each successful client connection, a new client data structure is allocated, several variables are initialized on the data structure including the login name, IP address, and unique ID for the client, and the data structure is added at the head of the list. Once the normal client-server operation commences, each time a server needs to notify a client of a state change (e.g. a framebuffer update), a message is sent to all connected clients by traversing the linked list, retrieving the client's socket descriptor, and sending the message with any accompanying data to the client. It is this basic mechanism that enables the THINC system to provide screen-sharing capabilities.

5.2.2 Handling of Collaboration Modes

For the most part, the server manages the mode of collaboration for all connected clients, though for optimization purposes the client enforces some behavior for managed mode. Clients display the mode of collaboration in the title bar of the window using the `XStoreName` function. If the server is set for managed collaboration, the current client in control is reported in the title bar as well. Using the title bar to report changes in the collaboration session was the most intuitive way to notify the user, and prevented the need for the user to monitor an extra window for this display.

In handling single-user mode, the server allows only one client to connect at a time by checking a lock variable in a global state structure. The server tests against this variable each time a client attempts to connect. If the server is set in single-user mode and no other client is currently connected, the server accepts a single connection and initializes the session for the client attempting to connect. If a client is already connected, additional connections by other clients are accepted, the login names and IP addresses of those clients are recorded by the server, and the connection is closed. In addition, no client structures are allocated for these clients.

As mentioned before, for unmanaged and managed modes, the server collects information at handshake time,

reports the collaboration mode to connected clients, and keeps a local record for each client. In unmanaged mode, the server performs some extra handling of cursor updates (see section 5.2.3) but does little more than queue input events from and send framebuffer updates to connected clients.

In managed mode, the server also keeps track of the ID of the control client to determine which client to process input events from. Also, some additional steps are taken during the client-server handshake for this mode. After reporting the mode of collaboration, the server sends a message to the newly-connected client indicating the unique ID of the control client, the ID assigned by the server to the client itself, and the number of users currently connected. The server then traverses its internal list of clients and sends the login name, IP address, and ID of all connected clients to the newly-connected client using the `AddUser` message. In turn, the server sends the information of the newly-connected client to the clients on the list using the same message type (the server skips sending this information to the newly-connected client). If the newly-connected client is the first client to connect, it is designated the control client and possesses the token.

During normal operation in managed mode, clients that do not possess the token are prevented from sending keyboard and mouse events, although cursor, framebuffer, and user list updates are still received. This is done as an optimization to keep the server from handling unnecessary event messages. The token is passed by an explicit command using the `PassToken` message from the control client to the server, indicating the unique ID of the client that is to receive the token. Once the `PassToken` message is issued, the control client immediately stops sending keyboard and mouse events to the server and resets a local flag telling itself that it is no longer in control. The server processes this message by locating the ID specified by the `PassToken` message in its own list and sending a `NewMaster` message containing the ID to the receiving client. The receiving client recognizes that it is the new control client by matching the ID of the `NewMaster` message with its own ID, and it enables itself to send keyboard and mouse events. The server then issues the same `NewMaster` message to all other clients including the previous control client. To show that the control client has passed the token to another client, the title bars of all connected clients are updated to display the username and IP address of the client now in control. Should a problem occur while passing the token to a particular client, e.g. the receiving client disconnects in the middle of token passing, then the server notifies the delivering client that an error has occurred with a `PassTokenErr` message, and the token is not passed.

In all supported collaboration modes, once a client is disconnected, the server frees its associated data structure in a cleanup function. If the server is in managed mode, the server notifies the remaining clients to update their respective lists using the `DeleteUser` message. If the disconnected client happens to own the token, then the client proceeding it in the linked list of clients kept by the server obtains the token (i.e. the client that connected after the disconnected client).

Note that there is a brief period of time when neither the control client nor the recipient client are in control while the token is being passed, and no clients are capable of sending input events to the server. In the event that both the control client and the recipient client disconnect at the same time during token passing, the server still believes that the original control client is still in control. In this case, the server simply passes the token to the next client in its linked list. This avoids the potential case where no clients can control the screen at once.

Client modifications to support the new collaboration protocol messages were straightforward. Clients keep a local list of connected clients that is centrally managed by message updates from the server. Adding and removing connected clients from this list amounted to simple linked list management. An additional state to the client was added to indicate whether or not the client is the control client, which determines its ability to send input events to the server. Clients are able to view the list of connected users by issuing a key command, which is trapped in the same manner as cursor attach/detach toggling (see section 5.2.4). In managed mode, token passing occurs by keying through the list and selecting the user.

5.2.3 Cursor Management

We use a function called `XWarpPointer()` to update the hardware cursor placement on all connected clients. In both managed and unmanaged modes, each time the cursor is moved the server sends a corresponding event to all connected clients to move their respective hardware cursors. The clients process this event by passing coordinates to `XWarpPointer()` which, in turn, relocates the cursor. Because `XWarpPointer()` issues a mouse event each time it is invoked, the client must suppress the event in order to avoid sending the event to the server. This avoids a ”mirror effect” where the client and server may repeatedly send redundant mouse events to each other. To suppress the event, the client keeps a count of all mouse events that are issued *by the server*. In other words, the count is incremented each time the client must used `XWarpPointer()` to relocate the cursor. If the count is greater than zero, then the mouse event generated by `XWarpPointer()` is discarded and the counter is decremented.

The server must also deal with a potential mirror effect by preventing itself from sending duplicate mouse events to the client that initially sent the event. In addition, it must also detect whether mouse events were actually issued by a client, or if the cursor was relocated by an application.

Each time the server receives a mouse event, the server identifies the local client structure that it associates with the mouse event. It also increments its own counter which tracks all mouse events issued by clients. If the counter is greater than zero, then the server broadcasts the event to all connected clients *except the one that issued the event*. If the counter is zero, then the mouse event is sent to *all* clients.

5.2.4 Cursor Attaching and Detaching

Modifications to support cursor attach and detach occur solely on the client side. To toggle the detach mode, a key command (CTRL-ALT) is carefully trapped. Once the key command is received, the client replaces the hardware cursor with a software-drawn cursor. At this point, the software cursor represents the location of the cursor on the remote framebuffer, and the hardware cursor is free to move as the user wishes. The toggling is triggered only once when either CTRL or ALT are released, but only after both of them are pressed. This prevents any interference with applications that rely on key commands that use a combination of CTRL and ALT plus an additional key[3].

Client complexity increases slightly as a result of handling the software cursor. Cursor image data is represented by two separate images; a source pixmap and a bitmap mask. Normally, the video card processes the image data directly in order to draw cursor image. The client manually draws the software cursor by creating a stipple for both the mask bitmap and the source bitmap and filling the stipples with the background and foreground colors of the cursor respectively. These fills are done directly on the client window, which requires the client to repaint the window from its backing store using the cursor's previous coordinates. Despite having to perform fill operations each time the cursor is moved, the client incurs minimal CPU overhead in drawing the cursor manually.

Nonetheless, the increased client processing cost associated with drawing the cursor is offset by the benefits. The two additional cursor states not only prevent the user from losing control over the cursor, but also allow the user to observe the cursor movements of other connected users while interacting with other applications. Moreover, we are able to provide this functionality without making any additional server changes or any sacrifice of bandwidth.

6 Experimental Results

The goal of our implementation was to transparently provide scalable multi-user support to an existing thin client protocol, in a way that compares favorably to other multi-user thin-client protocols. To exhibit transparency, we compare our implementation of THINCing Together with the original single-user version. We measure scalability by testing THINCing Together with an increasing number of connected clients. We also run comparisons between multi-user THINC and a competitor, Collaborative VNC.

[3]VMWare [36] uses a similar mechanism to detach the cursor from the virtual machine instance. It is also the only application that we know of that uses CTRL and ALT to do the detaching, though it allows the user to specify a different set of key combinations to do so as well.

6.1 Measurement Methodology

Increasing the number of thin clients connected to the server machine puts the burden on the server, in terms of both throughput and CPU usage. As only one thin client is run on each client machine, the burden on any one client machine does not change. Hence, we designed our experiments to put stress on the server.

For each experiment, we monitored the server-side network activity and CPU usage using atsar [3], a system activity report tool. We used a packet monitor to monitor the network activity at each client.

Benchmarks involving multiple clients were generally run at full speed in order to test server scalability, but we also employed slow-motion benchmarking [31] in order to measure the display quality at the client. Slow-motion versions of the benchmarks were run on a single client connected to a single server to obtain reference values of the amount of data that would be transferred in a "perfect" run. If the thin server had an infinite capacity to scale, every client would receive this much data in a fixed amount of time no matter how many clients connect to the server. The amount of data actually received by each client is compared to this figure, and their ratio gives a measure of the display quality perceived by the client.

6.2 Experimental Testbed

We used a semi-isolated experimental testbed to test thin-client performance under controllable network conditions. The testbed consisted of four isolated machines: a thin-client server, a network emulator, a packet monitor, and a web benchmark server. A number of machines from an adjoining computer lab were used to run thin clients (one client per machine) that connected to the thin server via the machine running the network emulator. The network emulator was set to emulate a 100 Mbps LAN network. Details of the testbed machines are summarized in Table 2.

6.3 Application Benchmarks

To measure thin-client performance with respect to display-intensive applications, we used web and video application benchmarks based on those that have been used previously to test THINC.

The web benchmark is based on the Web Text Page Load test from the Ziff-Davis i-Bench 1.5 [40] benchmark suite. It consists of a JavaScript-controlled load of a sequence of 54 web pages from the web benchmark server. This benchmark was modified to allow slow-motion benchmarking by introducing an optional delay of several seconds between

Role / Model	Hardware	OS / Window System	Software
Packet Monitor IBM Netfinity 4500R	933 MHz Intel PIII 512 MB RAM 9.1 GB Disk 10/100BaseT NIC	Debian Linux Testing (2.4.20 kernel)	Ethereal 0.9.13
Web Server IBM Netfinity 4500R	933 MHz Intel PIII 512 MB RAM 9.1 GB Disk 10/100BaseT NIC	Debian Linux Testing (2.4.20 kernel)	i-Bench 1.5 Apache 1.3.27
Thin Server IBM Netfinity 4500R	933 MHz Intel PIII 512 MB RAM 9.1 GB Disk 10/100BaseT NIC	Debian Linux Unstable (2.4.20 kernel) XFree86 4.3.0, TWM 4.3.0	Ethereal 0.9.13 Collaborative VNC 0.4 THINC servers
Network Emulator IBM Netfinity 4500R	933 MHz Intel PIII 512 MB RAM 9.1 GB Disk 2 10/100BaseT NICs	Debian Linux Unstable (2.4.20 kernel)	NISTNet 2.0.12
Thin Clients Dell Dimension 4100	1GHz Intel PIII 256 MB RAM 10/100BaseT NIC	Red Hat Linux (2.4.20 kernel)	Collaborative VNC client THINC clients

Table 2: Testbed machine configurations

the loading of pages. The delay ensures that the thin client receives and displays each page completely, and prevents temporal overlap in transferring the data belonging to two consecutive pages. The benchmark is run using Mozilla 1.4. The browser's memory cache and disk cache are enabled but cleared before each test run. In all cases, the browser window is 1024x768 in size, so the region being updated is the same for all clients.

The video benchmark is a MPEG1 video clip that is played on a server using Mplayer 1.1 [24], a media player that runs on Unix-based systems. The video file is 5.11 MB, and it contains a 34.75 second clip that consists of 834 video frames with an ideal frame rate of 24 frames/sec. The video benchmark is run on the server and is thus displayed on any connected clients. Slow-motion benchmarking is used by monitoring the amount of data traffic at two playback rates, 1 frame per second (fps) and 24 fps. The 1 fps playback rate ensures that all data packets from the server to a client are recorded, in order to establish the reference data size transferred from the server to the client that corresponds to a perfect playback. This is measured for a single client connected to a single server. To measure the video quality for multiple clients, we monitor the packet traffic delivered to each thin client at the normal playback rate (this is estimated by computing the total amount of data sent by the server divided by the number of clients) and compare the total data transferred per client to the reference data size. This ratio multiplied by 24 fps then gives the effective frame rate of the playback.

6.4 Measuring Synchronous Sessions

We would like to measure the following quantities with respect to synchronous sessions. First, we want to measure the client processing overhead due to using a software-drawn cursor rather than a hardware-drawn one. Second, we want to measure the server bandwidth and processing overhead with respect to the number of clients, to see how well the server scales. We do this for both the original single-user THINC and our multi-user THINC, to see what additional costs result from our implementation, and also for Collaborative VNC, to see how well we compare with a competitor product.

6.4.1 Cursor Draw

As discussed earlier, multi-user THINC uses a hardware-drawn cursor when the cursor is attached but switches to a software-drawn cursor when the cursor is detached. The use of a software cursor increases the processing overhead at the client machine but does not affect the server, as the same messages are still sent back and forth with respect to cursor movement.

To measure the client processor overhead, we instrumented THINC to report the number of CPU clock cycles were used for each cursor update. The overhead was measured for two experimental setups. In the first experiment, two clients were connected to a server. One client used the default hardware-drawn cursor, while the other client detached its cursor so that its THINC cursor was software-drawn. The `XWarpPointer()` function was run on the server 5000 times at a rate of ten times per second, using

Movement Type	Soft cursor	Hard cursor	Ratio
warp	28032	1651	17.0:1
smooth	23785	1508	15.8:1

Table 3: Overhead due to software-drawn cursor

randomly generated coordinates to jump around the screen. In the second experiment, three clients were connected to a server. The first client was used to control the cursor, and the other two clients were as before. The mouse of the controlling client machine was used to move the cursor quickly around the screen for one minute, resulting in 5640 cursor moves. The results of both experiments are tabulated in Table 3.

The results show that using a software cursor is about 17 times more expensive to the client CPU than using a hardware cursor when the pointer warps to a new location on the screen. When the pointer moves more smoothly, the software cursor performs slightly better. The overhead could become a factor on slower client systems, but in general it should not be a problem.

6.4.2 Web Benchmark

The i-Bench web benchmark was run on three versions of THINC as well as on Collaborative VNC. The three versions of THINC were as follows: ”branchpoint” is the single-user version that we started with, ”multiuser” is the multi-user version that we developed based on the branchpoint version, and ”newglyph” is a more recent single-user version with additional optimizations. Though the single-user versions do not support collaboration between clients, they do display the desktop on all connected clients, so it is possible to run display tests with multiple clients for all three versions.

We ran the web benchmark for each platform as follows. The appropriate server was run on the thin server machine. A number of corresponding clients were connected to the server. Mozilla was executed on the server, and the web benchmark was initiated by the controlling client. Runtime, data traffic, throughput, and server CPU utilization were measured by running atsar on the server machine. The results for each platform are depicted in Figures 1 to 4.

We observed that the multiuser version performs as well as the branchpoint version with respect to all measured quantities. This implies that the added functionality of multi-user THINC produces negligible server overhead. Cursor movement was not tested here, but that should have little effect on server performance, since the changes in cursor handling are at the client end.

In comparing THINC with Collaborative VNC, some patterns emerged that applied to both platforms. We observed that, for both platforms, data traffic increased su-

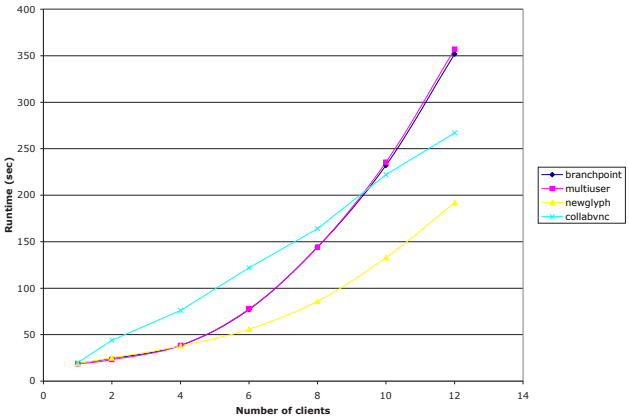

Figure 1: Web Test - Runtime

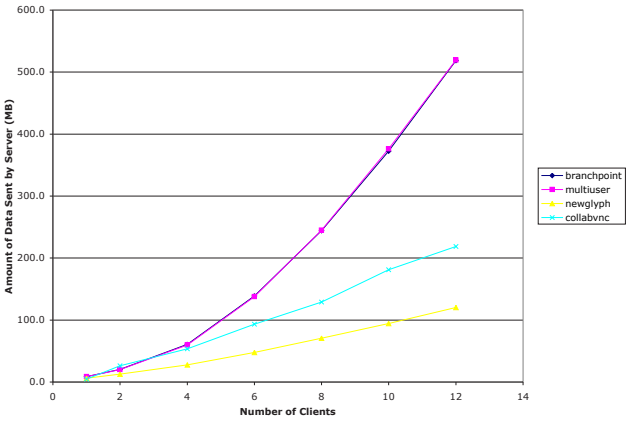

Figure 2: Web Test - Data Transferred

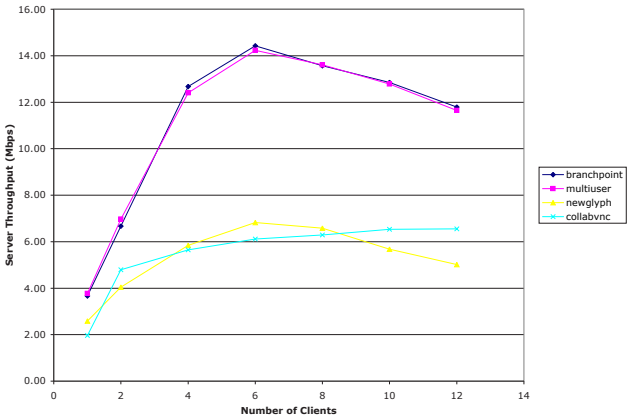

Figure 3: Web Test - Server Throughput

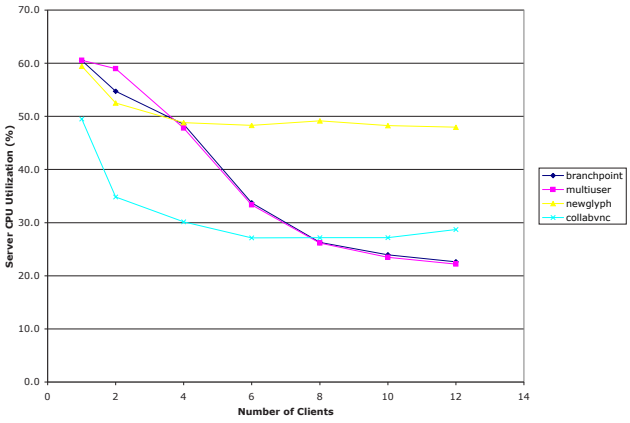

Figure 4: Web Test - Server CPU Utilization

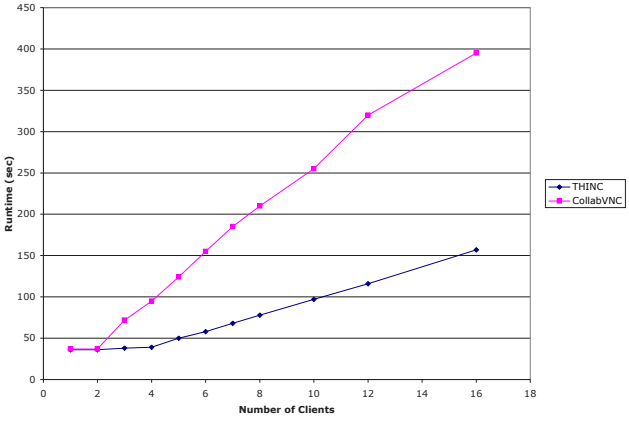

Figure 5: Video Test - Runtime

perlinearly with respect to the number of clients (Figure 2). This means that the amount of data sent to each client increased with an increase in the number of clients. THINC suffered from this problem more than Collaborative VNC, probably because VNC compensates and reduces the amount of data traffic by having the server wait to send screen updates to a client until that client is ready to receive them. Still, the newglyph version of THINC completed runs sooner than Collaborative VNC (Figure 1).

The results pertaining to server utilization were disappointing for both thin clients. Neither of them utilized more than a small fraction of the available bandwidth or CPU (Figures 3 and 4). The poor performances may be due to TCP issues, as the packet monitor did detect that many packets were being dropped. Alternatively, they may be due to some issue with Mozilla. Perhaps increasing the TCP window or using another browser would produce better results.

6.4.3 Video Benchmark

The video benchmark was run on the newglyph version of THINC as well as on Collaborative VNC. The other two versions of THINC could not be tested, because the branchpoint version (and thus also the multiuser version) does not support displaying video on multiple clients. The newglyph version uses XVideo for video playback.

For each platform, the appropriate server and clients were run, and Mplayer was executed from the server. Runtime, data traffic, throughput, and server CPU utilization again were measured by running atsar on the server machine. The results for each platform are depicted in Figures 5 to 9.

Video playback is perfect for three THINC clients, and is

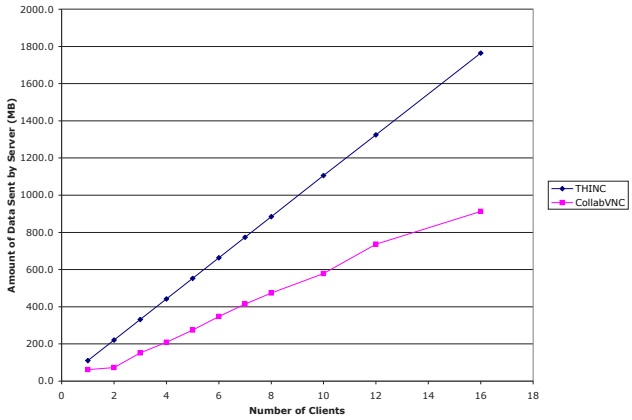

Figure 6: Video Test - Data Transferred

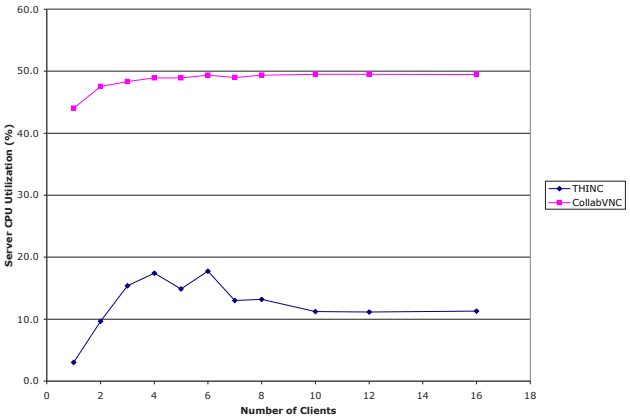

Figure 8: Video Test - Server CPU Utilization

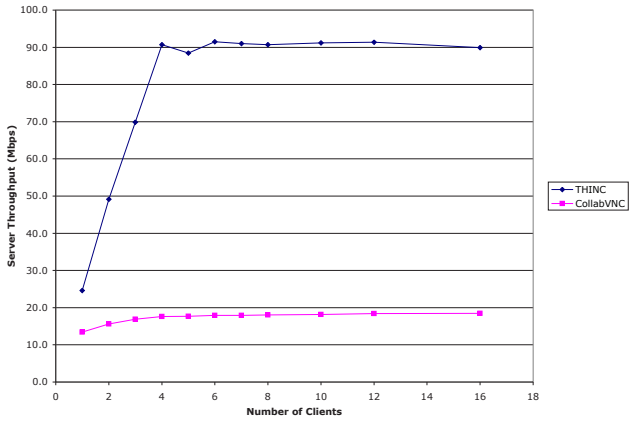

Figure 7: Video Test - Server Throughput

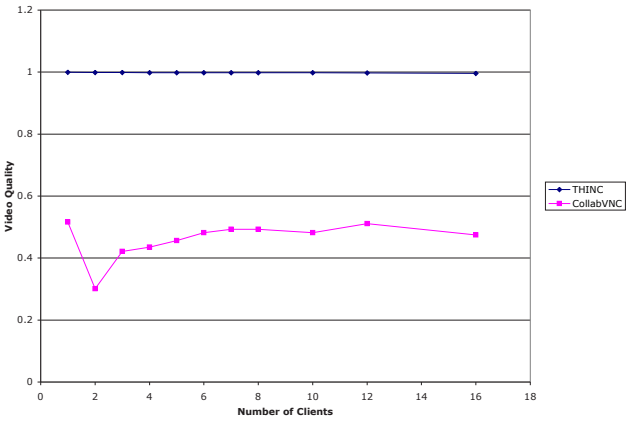

Figure 9: Video Test - Video Quality

almost perfect with four clients (Figure 5). Beginning with five clients, the runtime increases linearly with the number of additional clients, at a rate of 9 seconds per client (or approximately one-fourth of the length of the video). The slowdown after four clients is attributable to reaching the throughput limit, as the server utilizes more than 90% of the available bandwidth at that point (Figure 7). Server CPU utilization is not an issue in this case (Figure 8). The amount of data transferred (Figure 6) is about 110 MB per client regardless of the number of clients. This almost matches the amount of data (110.7 MB) resulting from a perfect slow-motion run of the video, thus the video quality is almost 100% (Figure 9).

Collaborative VNC plays the video in real time for two clients (Figure 5). The playback is not perfect, though, as it skips some frames to preserve real-time playback. Beginning with three clients, the runtime increases linearly at about 30 seconds per additional client (or approximately the length of the video). By skipping frames, VNC is able to reduce the amount of data transferred by the server, to about 60 MB per client. This is about half the amount of data (120.2 MB) that results from a perfect slow-motion run of the video that does not skip frames. Hence the video quality is about 50% (Figure 9). It utilizes only about 20% of the available bandwidth (Figure 6) and about 50% of available CPU (Figure 8).

Here we see that THINC scales well with the number of connected clients. The THINC server uses almost all of the available bandwidth. Had there been more available bandwidth (say B Mbps), the video would play perfectly with $\lfloor \frac{B}{25} \rfloor$ connected clients. Collaborative VNC, on the other hand, suffers from problems similar to those seen with the web benchmark. That is, it converges to a relatively low server throughput and CPU utilization, so it is wasting available resources. Again, this is likely due at least in part to the way VNC handles screen updates.

6.5 Measuring Asynchronous Sessions

We very recently developed a working implementation of asynchronous multi-user sessions. We have not yet had a chance to run performance tests for such sessions. In the future, we plan to run experiments similar to those we used to test synchronous sessions.

7 Conclusions and Future Work

Coupled with the existing THINC system, we believe that THINCing Together provides a solid multi-user environment and lays an architectural foundation for further collaborative features to be added. In implementing our design, we were able to address a major architectural issue by allowing multiple versions of the THINC server to run concurrently. We proved that the THINC protocol design was flexible enough to allow for synchronous sessions, and we were able to implementation them without modifying existing features. We also show that THINCing Together has a negligible impact on performance over a version of the THINC system without multi-user functionality, and compares favorably to Collaborative VNC.

Further refinement of the collaboration features offered by THINCing Together can be done in a number of ways. The development of a graphical user interface to facilitate token passing among clients participating in a synchronous session would be particularly useful. We also wish to experiment with the ability to share specific windows and screen regions so that host clients can ”hide” areas of the desktop from other users. Ongoing, a stronger security and authentication model will be needed to prevent malicious users from invading a host client’s desktop and arbitrarily executing commands.

8 Acknowledgments

Many thanks go to Ricardo Baratto (for his attentive assistance and helpful advice), and to Jason Nieh (for innumerable reasons).

References

[1] Altiris Vision. `http://www.mastersolutionus.com/vision.php`.

[2] Apple Remote Desktop. `http://www.apple.com/remotedesktop/`.

[3] atsar. `http://freshmeat.net/projects/atsar/`.

[4] Collaborative VNC. `http://benjie.org/software/linux/vnc-collaborate.html`.

[5] Documentation for XFree86[tm] version 4.3.0. `http://www.xfree86.org/4.3.0/`.

[6] GoToMyPC. `http://www.gotomypc.com`.

[7] NetMeeting. `http://www.microsoft.com/windows/netmeeting/`.

[8] SubEthaEdit. `http://www.codingmonkeys.de/subethaedit/`.

[9] Citrix ICA Technology Brief. Technical White Paper, Boca Research, Boca Raton, FL, 1999.

[10] R. Baratto and J. Nieh. THINC: A Remote Display Architecture for Thin-Client Computing. In preparation, 2004.

[11] Brad A. Myers. Using Handhelds and PCs Together. In *ACM Communications of the ACM*, volume 44, pages 34–41, New York, NY, 2001.

[12] Brad A. Myers et al. Floor Control in a Highly Collaborative Co-Located Task. Unpublished.

[13] Brad A. Myers, Herb Stiel, Robert Gargiulo, et al. Collaboration Using Multiple PDAs Connected to a PC. In *Proceedings CSCW'98: ACM Conference on Computer-Supported Cooperative Work*, volume 44, pages 285–294, Seattle, Washington, 1998.

[14] Elin Ronby Pedersen, Kim McCall, et al. Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 391–398, Amsterdam, The Netherlands, 1993.

[15] Hans-Peter Dommel and J. J. Garcia-Luna-Aceves. Floor Control for Multimedia Conferencing and Collaboration. *Multimedia Systems*, 5(1):23–38, 1997. `http://citeseer.ist.psu.edu/dommel97floor.html`.

[16] John Boyd. Floor Control Policies in Multi-User Applications. In *INTERACT '93 and CHI '93 Conference Companion on Human Factors in Computing Systems*, pages 107–108, Amsterdam, The Netherlands, 1993.

[17] John Menges and Kevin Jeffay. Beyond Window Sharing Hacks: Support for First-Class Window Sharing. `citeseer.ist.psu.edu/294458.html`.

[18] John Menges and Kevin Jeffay. Inverting X: An Architecture for a Shared Distributed Window System. `http://citeseer.ist.psu.edu/menges94inverting.html`.

[19] John Menges and Kevin Jeffay. On the Partitioning of Function in Distributed Synchronous Collaboration Systems. `http://citeseer.ist.psu.edu/5220.html`.

[20] Jonathan P. Munson and Prasun Dewan. A Concurrency Control Framework for Collaborative Systems. In *Computer Supported Cooperative Work*, pages 278–287, 1996. `citeseer.ist.psu.edu/munson96concurrency.html`.

[21] The Linux Console Project. `http://linuxconsole.sourceforge.net`.

[22] Thin-Client Market to Fatten Up, IDC Says. `http://news.com.com/2100-1003-5077884.html`.

[23] Worldwide Enterprise Thin Client Forecast and Analysis, 2002-2007: The Rise of Thin Machines. `http://www.idcresearch.com/getdoc.jhtml?containerId=30016`.

[24] MPlayer. `http://www.mplayerhq.hu/`.

[25] Advanced Reality: Presence AR Adapter. `http://www.advancedreality.com/`.

[26] Radhika Malpani and Lawrence A. Rowe. Floor Control for Large-Scale MBone Seminars. In *Proceedings of the Conference On Multimedia '97*, pages 138–145, Seattle, WA, 1997.

[27] RealVNC. `http://www.realvnc.com`.

[28] Richard C. Davis, James A. Landay, et al. NotePals: Lightweight Note Sharing By the Group, For the Group. In *Proceedings of the SIGCHI Conference on Human Factors in computing Systems*, pages 338–345, Pittsburgh, PA, 1999.

[29] S. F. Li and Q. Stafford-Fraser and A. Hopper. Integrating Synchronous and Asynchronous Collaboration With Virtual Network Computing. In *Proceedings of the First International Workshop on Intelligent Multimedia Computing and Networking*, volume 2, pages 717–721, Atlantic City, NJ, February 2000.

[30] S. Jae Yang. Citrix Moves Beyond Thin Clients. *PC Magazine*, 23(2), February 2004. `http://www.pcmag.com/article2/0,1759,1436546,00.asp`.

[31] S. Jae Yang and Jason Nieh and Naomi Novik. Measuring Thin-Client Performance Using Slow-Motion Benchmarking. In *2001 USENIX Annual Techincal Conference*, pages 35–49, Boston, MA, June 2001.

[32] Saul Greenberg. Sharing Views and Interactions with Single-User Applications. In *Proceedings of the ACM/IEEE Conference on Office Information Systems*, pages 227–237, Cambridge, MA, 1990.

[33] Saul Greenberg. Personalizable Groupware: Accommodating Individual Roles and Group Differences. In *Proceedings of the ECSCW '91 European Conference of Computer Supported Cooperative Work*, pages 17–32, Amsterdam, The Netherlands, 1991.

[34] Steve Whittaker, B. O'Conaill. Characterizing, Predicting, and Measuring Video-Mediated Communication: A Conversational Approach. In *Video Mediated Communication*, pages 107–132, Mahwah, NJ, 1997.

[35] TightVNC / Bandwidth-Efficient VNC Distribution. `http://www.tightvnc.com`.

[36] VMWare. `http://www.vmware.com`.

[37] WebEx. `http://www.webex.com`.

[38] The X Window System. `http://www.x.org/X11.html`.

[39] XMX. `http://www.cs.brown.edu/software/xmx/`.

[40] ”i-bench version 1.5”. http://i-bench.zdnet.com.