What You See Is What I Saw: Applications of Stateless Client Systems in Asynchronous CSCW

S. F. Li
Computer Laboratory
University of Cambridge
Cambridge CB2 3QG, UK
sfl20@cl.cam.ac.uk

A. Hopper
Department of Engineering
University of Cambridge
Cambridge CB2 1PZ, UK
ah@orl.co.uk

Abstract

Thin-client systems are becoming favourable in today's computing, as today's working environment tends to be mobile and ubiquitous. Stateless-client systems not only conform to the thin-client paradigm, but also extend to free the client completely from preserving any system state. In the stateless-client systems, the application processing takes place in the server, only changes to the user interface are sent to the client. Therefore, the client can connect to and disconnect from the server arbitrarily from different endpoints, and every time it connects, it receives the user interface with exactly the same state as when the system was last accessed. This technology of preserving all system state in the server and sending only the user interface to the client in the form of pixel images has motivated us to incorporate it in our CSCW (Computer Supported Cooperative Work) tools to support asynchronous as well as synchronous collaboration. Our paper will focus on how we make use of the stateless-client systems to support asynchronous CSCW on the World-wide Web by recording the changes to the user interface during a work session and replaying the session for any collaborative participant who is temporally separated. The recorded medium is a stream of rectangles containing pixel data of the user interface. Experiments show that the new lossless medium in general consumes less storage space than required by an equivalent lossy MPEG (Moving Picture Experts Group) video.

1. Introduction

The first client/server revolution chain-sawed mainframe applications into client and server pieces where the client managed the user interface and the server managed the shared resources. In [1], Orfali et al. predicted that during the second revolution, when client/server computing evolved from the Ethernet era to the intergalactic era, the clients and the servers that ran on the LAN (Local Area Network) would roam the intergalactic WAN (Wide Area Network). Today we see Web clients, in the form of Java applets, roam the Internet. To ensure client mobility, it is import that the clients remain thin in size so that they are Internet download-able. However, code mobility alone does not guarantee mobile computing. We need somewhere to maintain the computing state, and this is done in the server. The new trend of client/server computing is to move as much computation and state information from the client to the server as possible to keep the client thin to such a degree that the client requires minimum human resources to develop and minimum hardware resources to run. In other words, the clients become stateless. ORL's (Olivetti and Oracle Research Laboratory) VNC (Virtual Network Computing) [2], Citrix WinFrame [3] and Microsoft's Terminal Server code-named Hydra [4] are examples of such stateless client systems.

Now that the client is reduced to simply present rather than to manage the user interface, it brings us benefits such as *low total cost of ownership*, *zero administration* [4], etc. It does not take us long to incorporate this new technology in our CSCW tools. We experimented sending a desktop user interface to several clients simultaneously to enable workspace and application sharing in a work session. We also recorded changes to the user interface during the session and replayed for participants who worked at a different time to enable work session reviewing [5]. In this paper, we focus on how we extend ORL's VNC system to support asynchronous CSCW. In Section 2, we will describe the protocol upon which the VNC system is built; in Section 3, we'll explain how we perform recording and replaying of a work session. Section 4 will present the results of our applications. Finally in Section 5, we'll compare our work with similar work carried out at other research sites.

2. The RFB (Remote Frame Buffer) protocol

The protocol underlying the VNC system is the RFB protocol [2, 6]. This is a client/server protocol for remote access to the GUI (Graphical User Interface) of windowing systems. The remote endpoint where the GUI is displayed and where the user interacts with the GUI is called the RFB client and the endpoint where the GUI is generated or where the frame buffer originates is the RFB server.

The RFB client sends keyboard and mouse events to the RFB server, as well as regular requests to update its frame buffer.

Upon request, the RFB server sends frame buffer updates to the RFB client. A *frame buffer update* message has the following format:

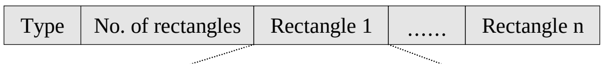


Figure 1a. Frame buffer update message

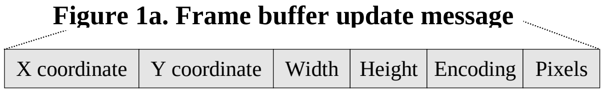


Figure 1b. An update rectangle (raw encoding)

- The *Type* specifies the message type, i.e. *frame buffer update* in this case (Figure 1a).
- The *No. of rectangles* specifies the number of rectangles contained in this update message. A *frame buffer update* consists of a sequence of rectangles, each representing a rectangular area of the frame buffer. Together, these non-overlapping rectangles cover the frame buffer area which has been changed since the last update (Figure 1a).
- An *update rectangle* uniquely identifies a rectangular area of the frame buffer by specifying the X, Y coordinates of its upper-left corner, and its width and height. *Encoding* specifies the encoding scheme the pixel data contained in the rectangle will follow. In the case of raw coding (i.e. no encoding at all), *Pixels* is an array of bytes containing the pixel values within the rectangle in the scan-line order, i.e. from left to right and from top to bottom (Figure 1b).

The RFB protocol is a generic protocol, which is independent of hardware platforms, operating systems or windowing systems and can be implemented in programming languages such as Java or C/C++.

3. Our applications

In our applications, we make use of the RFB (Remote Frame Buffer) protocol to support work session reviewing. This is a two-stage process, which involves recording the work session during synchronous collaboration and replaying it at a later stage for asynchronous collaboration.

3.1 Recording

Recording of a work session is achieved by seamlessly placing a proxy server between the RFB client and the RFB server [5]. This proxy server can intercept, store and forward messages between the RFB client/server without changing either of them. To facilitate future replaying, the *frame buffer update* messages are time-stamped and saved to a log file. The log file contains a sequence of records shown in Figure 2.

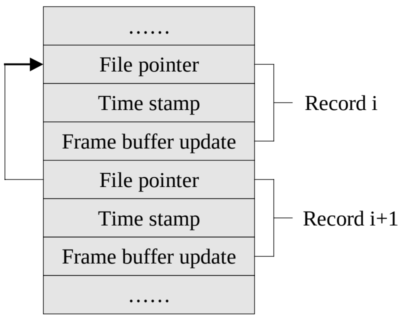


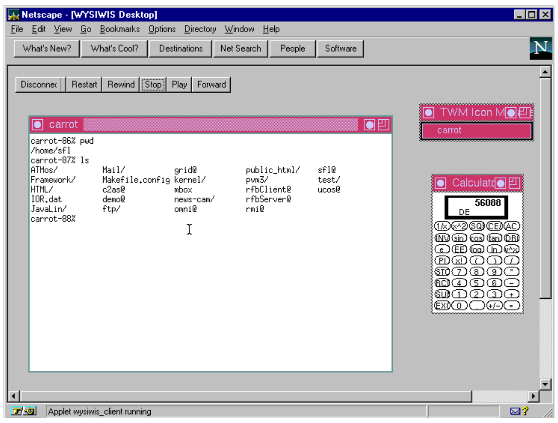
Figure 2. Records in the log file

Each record has three attributes, the *file pointer*, the *time stamp* and the *frame buffer update* message. The *file pointer* always points to the previous record in the sequence; the *time stamp* records the time at which the following message is saved; and the *frame buffer update* message is the original message sent by the RFB server to the RFB client during the work session. The order of the messages sent by the RFB server is maintained in the log file.

The rectangles in the *frame buffer update* messages contain absolute pixel values. These rectangles are sufficient for replaying but insufficient for rewinding. As the new pixel value replaces the old in the frame buffer, the latter will be lost and no longer recoverable unless we trace back to the nearest past rectangle that contains this pixel. To support VCR-like control (i.e. fast forward, rewind, play, stop etc.) in a more efficient way, we convert these rectangles so that they contain relative pixel values (or delta values) instead before they are stored. Such conversion can be easily done with the `XOR` (Exclusive OR) operator. For example, if the old value is the byte `01010101` and the new value `00001111`, the relative value or delta value can be calculated as `01010101 XOR 00001111 = 01011010.` With the help of the delta value, during replaying we can calculate the new value from the old: `01010101 XOR 01011010 = 00001111`; and during rewinding we can recalculate the old value from the new: `00001111 XOR 01011010 = 01010101`.

3.2 Replaying

The replaying of the work session is performed in a tool called the Reviewer. A screen snapshot of the Reviewer running as a Java applet in the Netscape Web browser is shown in Figure 3.


Netscape - [WYSIWIS Desktop]
File Edit View Go Bookmarks Options Directory Window Help
What's New? What's Cool? Destinations Net Search People Software
Disconnec Restart Rewind Stop Play Forward
carrot
TWM Icon M
carrot
Calculat
56088
DE
Applet wysiwis_client running

```
carrot-86% pwd
/home/sfl
carrot-87% ls
ATMos/          Mail/           grid@           public_html/    sfl@
Framework/      Makefile.config kernel/         pvm3/           test/
HTML/           c2as@           mbox            rfbClient@      ucos@
IOR.dat         demo@           news-cam/       rfbServer@
JavaLin/        ftp/            omni@           rmi@
carrot-88%
```

Figure 3. Work session review on the Web

The Reviewer provides the user with VCR-like control (i.e. fast forward, rewind, play, stop etc.) of the recorded session. It communicates through sockets with a Review Server that manages the session log files. The Review Server is responsible for tracing the time-stamped *frame buffer update* messages forward and backward (Figure 2.), and for sending these messages, together with their corresponding time-stamps, to the Reviewer in the correct sequence. This sequence depends on whether the update requests sent by the Reviewer are for replaying or rewinding. Like the RFB client (also known as the Viewer), the Reviewer just displays the frame buffer updates received, except that the update messages now come from the Review Server rather than the RFB server and require simple calculations to convert delta pixel values to absolute values. With the time-stamps, the Reviewer can replay the messages at the same rate as when they were recorded. These time-stamps will be ignored during fast forwarding or rewinding. Due to the security restrictions, the Reviewer, when running as an applet, cannot open sockets to a computer other than the one from which the applet is downloaded [7]. Therefore, we encapsulate our multi-threaded Review Server in an HTTP (Hyper Text Transfer Protocol) server which processes HTTP requests from the Web user for applet downloading and connects the applet to the Review Server after downloading is completed.

The Reviewer can playback the session according to a selected resolution. If the user selects a lower resolution than full screen, only a subset of the pixels in the screen image (or frame buffer) will be displayed. For example, the user can choose to display one pixel from every 2×2 pixel square in the screen image reducing the image width and height by a factor of 2 respectively. By doing so, the user will be able to browse through work sessions with trade-offs between the transmission bandwidth required, the display area available, and the details of the session.

To speed up the playback, the Reviewer can retrieve the log file to its local disk by means of, say the FTP (File Transfer Protocol) and then replay the session locally by reading directly from the retrieved file. However, Java applets cannot read or write files on the Web user's disk for security reasons [7], so the Reviewer has to run as a Java or C/C++ application if a local replay is desirable. Future releases of Java will provide more flexibility on security controls.

4. Experiments

To evaluate our applications, we designed three tasks for recording and replaying.

- In task 1, the user types 10 lines of "Hello World" in the *vi* editor.
- In task 2, the user presses "1+2+…+10=" in the calculator, and drags the calculator to a new location after the result of the calculation appears.
- In task 3, the user launches a Netscape browser with the geometry 600×400, then goes to the bookmarked AltaVista search engine page and searches for the word "CSCW". Once the search results start to appear, the user presses the "Page Down" key to view the list of the first 10 matches.

The clock on the screen is ticking while the user performs the above tasks.

The table below shows the performance statistics of the applications. During all the tasks, the servers including the RFB server, the proxy server and the Review Server are all executing on a UNIX workstation at ORL. And for each task, we run a local test and a remote test. In the local test, the clients including the Viewer and the Reviewer run as Java applications on a second UNIX workstation at the same site. While in the remote test, they run as Java applets within a Netscape Web browser on a Windows NT workstation from the Computer Laboratory of Cambridge University. The results are obtained by taking the average of three consecutive trials.

The *Record Size* shows the amount of data (in megabytes) transmitted from the RFB server to the RFB client and stored by the proxy server during the task sessions. The pixel data transmitted and recorded follows raw encoding, i.e. no compression scheme is applied. The *Record Time* is the time (in seconds) taken by the user to complete the tasks. The *Replay Time* is the time required for the user to replay the recorded sessions in the fast forward mode.

Quite predictably, in the remote tests it takes longer to complete the same tasks than in the local tests, as the

	Task 1		Task 2		Task 3	
	Local	**Remote**	**Local**	**Remote**	**Local**	**Remote**
Record Size (Mbytes)	0.99	0.87	0.62	0.54	4.63	3.59
Record Time (Sec)	39	41	25	55	100	150
Replay Time (Sec)	14	39	12	50	27	100
X Time (Sec)	29		17		45	
MPEG Size (Mbytes)	2.62		3.50		4.99	

Table 1. Performance statistics

network connection between the client and the server is slower. However, the size of the data transmitted and recorded in the remote tests appears to be smaller. This is because the RFB client is adaptive to its computing environment, i.e. when it detects a slower connection, it sends less update requests to the RFB server. In other words, there exists a trade-off between the network bandwidth and the granularity of the frame buffer updates.

For comparison, we have recorded the tasks using the equivalent MPEG video, and the results are as listed in the last two rows of Table 1. The *X Time* is the time taken by the same user to complete the same tasks on a local X display and the *MPEG Size* is the size of the resulting MPEG stream generated by an MPEG encoder [16]. Our frame buffer has a width of 796 and a height of 576 pixels, the subsampling ratio is 4:1:1, the frame rate is in the range of 5 to 15 fps (frames per second) and the MPEG encoder parameters are set to their corresponding default values [16]. In general, our recording consumes less storage space than required by the equivalent MPEG video and the savings depend on the complexity of the scenes. When the screen images are not complex, as in Task1 and Task2, our medium can save up to 80% of the storage space.

We observe that it takes a longer time to complete the tasks with our applications than with a local X display, sometimes three times as much (Table 1). This is partially because of the heavy traffic involved between the RFB client/server, and partially because of the slow drawing mechanism of the Java programming language [6]. When the network bandwidth between the RFB client/server is guaranteed, when the speed of Java increases in the later versions, and when we apply various compression schemes to the pixel data, our performance will improve. The RFB client (i.e. the Viewer) implemented in C has already approached the X display performance, as it takes the user 28, 19 and 46 seconds to complete the three tasks respectively.

5. Related work

CSCW (Computer Supported Cooperative Work) systems can be classified according to the time/space matrix [8]. They are often referred to as either synchronous or asynchronous (same time/different time), and either colocated or remote (same place/different place).

The majority of the CSCW systems supporting shared workspace and applications operate in the synchronous mode, determined by the WYSIWIS (What You See Is What I See) paradigm [10]. They require the real-time presence of all the participating individuals. However, a significant portion of the collaborative activities in practice occurs in an asynchronous manner. This is due to the fact that globally distributed teams may work in different time zones and colocated groups may work in shifts [9]. The additional difficulties of sharing workspace and applications asynchronously can be reduced by recording and replaying the collaborative session for absent participants, so that they can review the work done in order to continue the incomplete task. The video technology seems to be a natural choice for recording and replaying. Our applications differ from this traditional technology in the following ways:

- Video capture requires extra devices such as cameras, video capture cards, etc. Although these devices are becoming more integrated with computers, they are not ubiquitous, i.e. not all the workstations are equipped with them. In our applications, both the Viewer and the Reviewer can be downloaded to and executed within a Web browser, and the recording and replaying can be performed ubiquitously and seamlessly without extra devices, hence answering to Robert Johansen's call for "any time, any place" support [11].
- Video technology is widely used in CSCW systems such as video conferencing (synchronous collaboration) [12] and video mail (asynchronous collaboration) [13], however, in the context of workspace and application sharing, the technical limitations of video taping computer screens make the screen very difficult to read during playback [14]. Our applications adopt the idea of screen capture rather than video capture and the resulting medium is a stream of rectangles containing pixel data of the screen images.
- Although Hiroshi Ishii et al. identified video as the most powerful medium for fusing a variety of traditionally incompatible workspaces to create an open shared workspace [14], our workspace and application sharing is purely computer-based, therefore we can take advantage of the spatial and temporal correlation of the frame buffers. Most of the time when user interactions take place, only a small area of the screen is affected, so that we only need to

record the area which has been changed since the last screen update, hence keep the data recorded to its minimal. Our experiments show that our medium with lossless encoding consumes less storage space than required by an equivalent MPEG video with lossy compression.

- Video technology alone only supports viewing of the workspace and applications, it does not support remote operations. After replaying the session, we may want to access and control the same workspace and applications seen in the recording to carry on the work. In our applications, the synchronous and the asynchronous collaborations are integrated: we use the Reviewer for replaying (asynchronous collaboration) and the Viewer for remote operations (synchronous collaboration). These operations (e.g. key presses), when saved, can be used to automatically index a recording, to provide searching facilities during playback for queries such as "what happened last time when the user typed the command *xterm*?"

Similar to our applications, Lotus ScreenCam [15] uses dynamic screen captures to create movies for training and customer support. It delivers instructions and procedures with the exact visual cues that illustrate what to do, hence the concept of WYSIWYD (What You See Is What You Do). However, a work session can only be recorded and replayed on Windows platforms even though StreamCam, a streaming server technology from the same company, enables the movie playback directly from a Web page. Our approach is more generic, i.e. we can record a Unix session and replay it on a Windows PC or vice versa.

6. Conclusion

In this paper, we have presented a new medium, i.e. the stream of rectangles containing pixel data of the screen images, generated by a Stateless Client System known as the VNC. With this medium, we can record and replay a work session to enable reviewing involved in asynchronous collaboration, introducing the concept of *What You See Is What I Saw*.

Acknowledgement

Sheng F. Li is sponsored by a CASE grant from the *Engineering and Physical Sciences Research Council (EPSRC)* and the *Olivetti and Oracle Research Laboratory (ORL)*. Andy Hopper is the professor of Communications Engineering and the director of ORL. The authors would like to thank Quentin Stafford-Fraser, Tristan Richardson and Sai-Lai Lo for their support, and Stuart Wray and Antony Rowstron for their comments.

Reference

[1] R. Orfali, D. Harkey and J. Edwards. "Intergalactic Client/Server Computing", *Byte*, April, 1995

[2] T. Richardson, Q. Stafford-Fraser, K. R. Wood and A. Hopper, "Virtual Network Computing", *IEEE Internet Computing*, Vol. 2 No. 1, January/February 1998

[3] "ICA Technical Paper", Citrix Systems, Inc., http://www.citrix.com/technology/icatech.htm, March 1996

[4] "Microsoft Windows NT 'Hydra' and Windows-based Terminals", Microsoft Corporation, *http://www.microsoft.com/ntserver/info/hydra.htm*, September 1997

[5] S. Li and A. Hopper, "A Framework to Integrate Synchronous and Asynchronous Collaboration", *IEEE Seventh International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'98), IEEE Computer Society Press*, June 17-19, 1998.

[6] K. R. Wood, T. Richardson, F. Bennett, A. Harter and A. Hopper, "Global Teleporting with Java: Toward Ubiquitous Personalised Computing", *IEEE Computer*, vol. 30, no. 2, February, 1997

[7] M. Morrison, et al., "Java 1.1 Unleashed Third Edition", *Sams.net Publishing*, ISBN 1-57521-298-6, page 187-188, 1997

[8] Francois Fluckiger, "Understanding Networked Multimedia", *Prentice Hall*, ISBN 0-13-190992-4, page 113-114, 1995

[9] I. Gorton, I. Hawryszkiewycz and K. Ragoonaden, "Collaborative Tools and Processes to Support Software Engineering Shift Work", *BT Technology Journal*, vol. 15, no.3, July 1997

[10] W. Reinhard, J. Schweitzer, G. Völksen and M. Weber, "CSCW Tools: Concepts and Architectures", *IEEE Computer*, vol.27, no.5, May 1994

[11] J. Grudin, "Computer-Supported Cooperative Work: History and Focus", *IEEE Computer*, vol.27, no.5, May 1994

[12] T. King, "Pandora: An Experiment in Distributed Multimedia", *Olivetti and Oracle Research Laboratory Technical Report 92-5*, *Proceedings of Eurographics*, 1992

[13] S. Wray, T. Glauert and A. Hopper, "The Medusa Applications Environment", *Olivetti and Oracle Research Laboratory Technical Report 94-3*, *Proceedings of the International Conference on Multimedia Computing and Systems*, Boston, MA, May 1994

[14] H. Ishii and N. Miyake, "Toward An Open Shared Workspace: Computer and Video Fusion approach of TeamWorkStation", *Communications of the ACM*, vol. 34, no.12, December 1991

[15] "Lotus ScreenCam", Lotus Development Corporation, http://www.lotus.com/products/screencam.nsf, July 1997

[16] A. Hung, "PVRG-MPEG CODEC 1.1", *Portable Video Research Group, Stanford University*, November 1993