# Integrating Synchronous and Asynchronous Collaboration

# With Virtual Network Computing

Sheet Feng Li
*Department of Engineering*♣
*University of Cambridge*
*Cambridge CB2 1PZ, UK*
*sfl20@eng.cam.ac.uk*

Quentin Stafford-Fraser and Andy Hopper♣♠
*AT&T Laboratories Cambridge*♠
*24a Trumpington Street*
*Cambridge CB2 1QA, UK*
*{qsf, ah}@uk.research.att.com*

**Abstract**

*The trend in computing models has changed from thin-client (text terminals) to thick-client (graphics terminals) to standalone personal computing (PCs), and then back to thick-client (client/server applications) and thin-client (Web-based applications) again. This trend is now leading us to the so-called stateless-client computing, which is an ultra-thin-client model that frees the client completely from preserving any application state. This paper explains how we integrate synchronous and asynchronous sharing of workspace with Virtual Network Computing, a stateless-client computing technology developed at the AT&T Laboratories Cambridge. Experiments and applications have demonstrated that our collaborative system is feasible for operation in current and future computing environment.*

## 1. Introduction

In the early days of computing, the communication between users and computers was via some media such as punched cards or paper tapes. In 1963, the Compatible Time-Sharing System (CTSS) that links a large number of users to a single computer via remote terminals was developed at MIT [1]. The terminals and their terminal emulators provide a character interface, and are usually portable in size and limited in functionality.

Text terminals are sometimes referred to as *dumb terminals* because they can't run graphics. The simplest solution to upgrade to graphics terminals is to enable the host computer to send bitmaps continuously across the network. As recently as early eighties, terminals were typically connected to the central host computer via serial lines between 1200 and 19200 baud. For basic terminals such as VT100, a complete screen refresh takes a few seconds over the serial line, but for terminals that display bitmaps, the network load would be unacceptable. A more network-efficient solution is to provide an intelligent terminal that understands how to draw graphics objects without bitmaps. In 1984, MIT created the X Window System [2]. The X terminal or the X server running on Unix workstations, PCs and Macintosh provides a WIMP (Windows, Icons, Menus, Pointer) interface, but requires a significant amount of system resources such as memory and disk space.

In the mainframe model, control is totally centralised and users have little control over data or applications. In the early eighties, microcomputers such as Apple Macintosh and IBM PC also arrived on the market with windowing systems (e.g. Microsoft Windows) as a standard. These personal computers offer a great degree of flexibility and freedom. They not only provide terminal emulators for access to mainframes and workstations, but also transfer the control of resources to local users.

The services offered by standalone desktops could not meet the increasing demands from users and applications. In addition, isolation of resources made enterprise-wide information sharing very difficult. Soon the personal computers and workstations were networked together to provide a more powerful and reliable system, and client/server computing emerged as the method of choice for it preserves the local functions on the desktop, while at the same time, provides centralised control of backend data and applications [5].

As the number of users and applications for personal computers increases, so does the administration and maintenance cost. Since the nineties, a simpler computing model is emerging, driven by the growth of the Internet. Thin-client computing, as embodied in Network Computers (NCs) and Windows-based Terminals, moves most application functionality back to the server and leaves only the user interface on the desktop. This higher degree of centralisation simplifies the administration and maintenance process. The thin-client architecture can be combined with the World Wide Web, where the Web browser itself is a thin client, which makes thin-client computing almost synonymous with *internet computing* [3].

Where is thin-client computing headed and how thin can a client reach? The history of computing models once ruled out the proposal that sends bitmaps continuously across the network from the host computer to the terminals because of the network overload. The fast growth in network speed draws our attention back to reconsider this proposal. The dumb graphics terminal that simply displays bitmaps is what we call a *stateless terminal*, for the entire state of all the applications in the user session is managed in the host computer. Stateless-client computing, which centralises all resources and functionality on the server, seems to be the next logical step in the evolution of computing models. The next section will describe this new computing model in details, using Virtual Network Computing (VNC) as an example [6].

Our research is motivated by the observation of the lack of support for collaborative activities. In our labs, we work closely with our colleagues who may be geographically or temporally separated from us. The Shared Whiteboard Tools and the Shared Application Tools described in [4] can not offer sufficient support as they only operate in synchronous mode. We envision a collaborative system that continuously projects the screen of one computer onto that of another, so that two or more people can share workspace simultaneously. While at the same time, the screen is continuously recorded for future replay. The principle of this collaborative system seems to coincide with that of the stateless-client computing systems where bitmaps of the screen are sent continuously across the network. So the question now becomes, can we make use of stateless-client systems to support workspace sharing? This research is set to explore the potential collaborative features of stateless-client computing, and the paper argues that stateless-client systems can be enhanced to enable the sharing of computer workspace both synchronously and asynchronously.

## 2. Virtual Network Computing

The VNC technology pushes the thin-client computing model to an extreme by moving the execution of applications completely to the server, leaving the client stateless. Without changing any windowing system, VNC breaks it into client/server pieces, namely the VNC server, the VNC client and the VNC protocol that connects the previous two.

The VNC server executes all the applications and generates the frame buffer. The VNC client displays the frame buffer and accepts user input. And the VNC protocol defines the mechanism that transfers the frame buffer updates from the server to the client, and the user input from the client to the server. Figure 1 shows the VNC client (also known as the VNC Viewer) running as a Java applet in the Web browser.

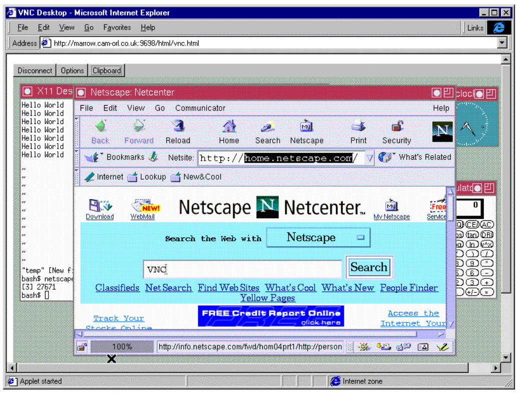


**Figure 1. VNC Viewer operating netscape, xterm, xcalc, and xclock**

The VNC client sends the following messages to the server:

- *Frame buffer update request* notifies the server that the client is ready to receive and display the next frame buffer update. The server responds to this message by sending a *frame buffer update* message.
- *Key event* indicates a key press or release when the client receives key input from the user.
- *Pointer event* indicates either pointer movement or a pointer button press or release when the client receives mouse input from the user.

The VNC server sends the following messages to the client:

- *Frame buffer update* replies to the *frame buffer update request* message from the client. It consists of a sequence of rectangles, each representing a rectangular area of the frame buffer. Together, these rectangles cover the frame buffer area which has been changed since the last update. Each rectangle uniquely identifies a rectangular area of the frame buffer by specifying the x, y coordinates of the area's upper-left corner, and its width and height. The rectangle also specifies the encoding scheme the pixel data of its represented area will follow. In the case of raw coding (i.e. no encoding at all), it lists the pixel values in the scan-line order, i.e. from left to right and from top to bottom.

The VNC protocol is very similar in concept to the Independent Computing Architecture (ICA) protocol used in Citrix Systems [11], the T.128 (formerly known as T.SHARE) application sharing protocol used in NetMeeting [7], and the native protocol used in Sun Ray Hot Desk architecture [8]. The VNC protocol has the advantage of being platform independent, so that we can share X-Window applications on Windows-NT platforms, and Windows applications on Unix platforms.

## 3. System architecture and functionality

The VNC client and server communicate through socket connections. We extend the client/server architecture by seamlessly placing a proxy between the two components to intercept and manipulate the message streams. The proxy merges all the messages from the multiple clients and forwards these messages to the server as if they were from a single client; while on the other hand, it multicasts the messages from the server to each client and enables the clients to share the server simultaneously. It expands a one-to-one client/server into a many-to-one multi-client/server communication channel, where each client receives exactly the same messages from the server as the others participating in the collaborative session. The messages flowing between the clients and the servers are time-stamped and stored, and can be played back at the same rate as they were recorded.

The proxy monitors and coordinates the activities of the VNC clients and servers. It provides the coordination mechanism through the following services:

- **Register** This service allows the clients and the servers to register themselves in the Module Table after they connect to the proxy. The proxy keeps track of all the components currently connected to it and facilitates a simple notification service to the clients. It acts as a session server where the user-related states such as the number of participants are centralised. When there is a change of the shared states, it will notify all the participating clients.
- **Remove** This service allows the clients and the servers to remove themselves from the Module Table before they disconnect from the proxy.
- **Route** This service allows the clients and the servers to send messages to each other. The proxy maintains a Route Table that specifies the destination components for messages sent by each source component.
- **Record** This service allows the clients and the servers to record the messages sent by them. The recording is performed by the proxy and stored in the log files.
- **Request** This service allows the clients to request the floor. After a VNC client obtains the floor, the proxy only routes the user events coming from this particular client and our system enters the *supervision mode*, i.e. only this supervising client can operate the shared work session while the rest of the collaborative clients can only view the session.
- **Release** This service allows the clients to release the floor. When the VNC client releases the floor, our system returns to its normal mode and the collaborative clients can share the view and the control of the remote work session.
- **Reconnect** This service allows the clients to switch to a new VNC server. When this service is invoked, the proxy closes the connection with the current VNC server and reconnects to a new server. The collaborative clients can now decide whose server is to be shared and dynamically switch between various homogeneous and heterogeneous servers in a shared work session.
- **Report** This service generates auditing reports on messages flowing between the clients and the servers.

Our VNC Reviewer reads from the log file of *frame buffer update* messages and replays the captured activities with VCR-like control similar to playing back a video tape. The Reviewer also provides various browsing and searching facilities for users who wish to browse through the captured activities or to search for a particular event or event sequence in the recording [9].

## 4. Experiments

We are trading off bandwidth and storage capacity for a platform independent thin-client architecture. This section explains why such a trade-off is acceptable by measuring the bandwidth and storage consumption of our system with carefully designed experiments.

We designed three tasks to be performed by a number of collaborative users who share workspaces and applications in a work session displayed by Figure 1, and who communicate with each other on telephone. The

VNC server and the proxy are executing on a UNIX workstation at AT&T Laboratories Cambridge. The primary user runs the VNC client as a Java applet within a Netscape Web browser on a Windows-NT workstation from the Engineering Department of Cambridge University with a shared 10Mbps connection to the servers, and performs the three tasks specified below. The secondary users run the VNC clients as Java applications on UNIX workstations at AT&T with shared 10Mbps or 100Mbps connections to the servers, and share the view of the collaborative session.

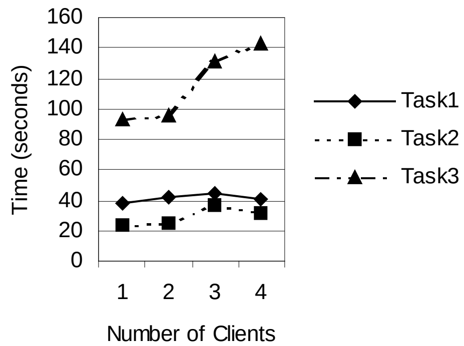
- In task 1, the primary user types 10 lines of “Hello World” in the *vi* editor. This task involves key events only.
- In task 2, the primary user clicks “1+2+…+10=” in the calculator, and drags the calculator to a new location after the result of the calculation appears. This task involves mouse events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

The frame buffer updates are transmitted without any encoding or compression scheme. Graph 1 and 2 show how the time required to complete the tasks and the average data transmission rate for each client change as the number of VNC clients increases from one to four. The results are obtained by taking the average of three consecutive trials.

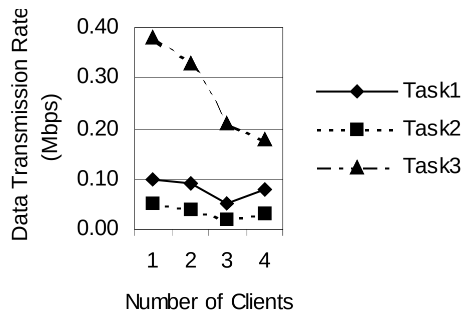
Graph 1 shows that the task completion time tends to increase as the number of clients increases. This is because the network where these tests are carried out does not support multicast by internal mechanism, and multicasting server messages to multiple clients are implemented by multiple biparty or point-to-point connections. This is of course uneconomical, in terms of processing costs at the proxy and the communication cost between the proxy and the clients. It places an unnecessary overload on some segments of the network that have to carry multiple identical flows, and more clients usually cause more congestion and delay. This also explains why task 3, where there are significantly more frame buffer updates than task 1 and 2, suffers more as the number of clients increases.

Most high-level network protocols such as the Transmission Control Protocol (TCP) only provide a unicast transmission service. If a node wants to send the same information to multiple destinations using a unicast transport service, it must perform a replicated unicast, and send multiple copies of the data to each destination in turn. Therefore, similar to audio and video conferencing, the number of VNC clients that can connect to the proxy is usually limited by the network bandwidth. A better way to transmit data from one source to multiple destinations is to provide a multicast transport service [10]. With a multicast transport service, a single node can send data to multiple destinations by making just a single call on the transport service, and the data is only duplicated when the transmission route diverges. Multicast transmission such as IP multicast provides important performance optimisation over unicast transmission in bandwidth-intensive real-time audio and video conferencing, and similarly in our system.

Graph 1 Task Completion Time




Graph 2 Average Data Transmission Rate For Each Client




Graph 2 shows that the average data transmission rate for each client decreases as the number of clients increases. The cause of this decrease in rate per client is twofold. Firstly, the VNC protocol has an adaptive quality in the client’s computing and networking environment: the slower the client and the network, the lower the rate of update. This is because frame buffer update is demand-driven by the client. That is, the server sends an update only when the client sends an explicit request, and all screen changes since the client's last request are coalesced into a single update. At the proxy where the update requests are merged, the faster clients have to wait for the slower clients. Therefore the more the clients, the lower the rate of requests, and the fewer the updates to transmit. Secondly, with more clients, it takes a longer time to complete the tasks as the result of increasing congestion and delay. Although the data transmission rate per client drops as the number of clients increases, the total data transmission rate at some segments of the network actually grows because multiple point-to-point connections share these segments and hence the data transmission rate there is multiplied.

To summarise, the system requires the network bandwidth to support the data transfer rate of up to 4Mbps at peak time during our daily activities. This requirement is further reduced by various encoding schemes we apply to the pixel data.

Table 1 shows the amount of data (in megabytes) transmitted from the VNC server to the VNC clients and stored by the proxy server during the task sessions with two collaborative users. *Raw size* shows the recording size when the frame buffer updates are transmitted and saved without any encoding or compression scheme while *encoded size* is the recording size when various encoding schemes are applied to the pixel data. The details of the encoding schemes can be found in [6]. The recording sizes shown do not include the size of the initial whole screen image transmitted and stored (which is 0.44 Mbytes if the data is not encoded) when the user first signs into the work session. When users first connect to a work session, they will receive a whole screen image reflecting the current state of the session. The following frame buffer updates only cover the part of the screen that is changed since the last update, and their sizes are usually much less than the size of the whole screen.

***Table 1*. Recording size**

| | **Task1** | **Task2** | **Task3** |
|---|---|---|---|
| **Raw size (MB)** | 0.40 | 0.10 | 2.86 |
| **Encoded size (MB)** | 0.03 | 0.06 | 0.77 |

## 5. Conclusion

VNC supplies a live and mobile desktop that can be accessed from any Internet-connected machine. Built on the VNC technology, our system enables this desktop to be shared for synchronous collaboration and stored for asynchronous collaboration.

The simplicity of the VNC protocol allows it to be applied to a wide range of hardware devices. We have demonstrated in [6] the VNC server that runs on Unix workstation and produces an X-Window desktop, and the VNC server that runs on Window-NT workstation and produces a Windows desktop. VNC servers can also be programmed to run on any hardware devices other than computer workstations, e.g. on CD players, telephone answering machines, etc. The servers generate display interfaces for hardware devices directly without any reference to a windowing system or frame buffer, and the display interfaces can be accessed from any VNC Viewer anywhere.

Thus, we envisage a "virtual desktop", where users can decide on the components to be included in their computing environment. They can choose to run a number of VNC Viewers that access remote computer workstations and hardware devices with a real-time view, and some of the views are shared with other users. They can also run a number of VNC Reviewers that access stored computer sessions with a non-real time or delayed view. This desktop of desktops (and other display interfaces) offers much flexibility in creating a new collaborative computing environment.

## Reference


[1] T. Vleck, "The IBM 7094 and CTSS", http://www.multicians.org/7094.html, 1997.

[2] N. Mansfield, The Joy of X: An Overview of the X Window Systems, Addison-Wesley, ISBN 0-201-56512-9, 1993.

[3] P. Taylor, "Internet reshapes world computing", Financial Times, Wednesday November 3, 1999.

[4] F. Fluckiger, *Understanding networked multimedia*, Prentice Hall, 1995

[5] R. Orfali, D. Harkey, and J. Edwards, "The Essential Client/Server Survival Guide", Third Edition, John Wiley & Sons, 1999.

[6] T. Richardson, Q. Stafford-Fraser, K. R. Wood and A. Hopper, "Virtual Network Computing", *IEEE Internet Computing*, vol.2, no.1, Page 33-38, January/February, 1998

[7] Microsoft Corporation, *NetMeeting 2.0 Reviewers Guide*, June 1997, *http:// www.microsoft.com/netmeeting/*

[8] Sun Microsystems, *Deploying the Sun Ray Hot Desk Architecture*, August 1999, *http://www.sun.com/sunray1/hotdesk.html*

[9] S. Li, M. Spiteri, J. Bates and A. Hopper, "Capturing and Indexing Computer-based Activities", *Proceedings, 2000 ACM Symposium on Applied Computing (to appear)*, Como, Italy, 19-21 March, 2000.

[10] R. Steinmetz and K. Nahrstedt, Multimedia: Computing, Communications and Applications, ISBN 0-13-324435-0, page 412, 1995.

[11] J. P. Kanter, Understanding Thin-Client/Server Computing, Microsoft Express, ISBN 1-57231-744-2, 1998.