Interfacing High-Definition Displays via the Internet

Gregory G. Finn, Rod Van Meter, Steve Hotz, Bruce Parham
USC/Information Sciences Institute
August 1995

ABSTRACT

Research efforts are underway to interface major subsystems and devices via a gigabit LAN rather than a system bus. As the destination to which most multimedia data is ultimately directed, a color display presents an important test case for this approach to computer system architecture. This report discusses requirements and general research issues for a network attached display, and describes the ISI Netstation Project's efforts to create a network color frame buffer.

1. Introduction

Research efforts are underway to interface major subsystems and devices to one another via a gigabit LAN, and in turn via the Internet, rather than over a system bus [1][2][3].

As the destination to which most multimedia data is ultimately directed, a color display presents an important test case for this approach to computer system architecture. Further, a display may benefit greatly from direct internetwork access. Interfacing a color frame buffer to the network allows multimedia data to be routed across the Internet directly into the frame buffer, alleviating potential bottlenecks for the primary system/CPU resources.

An additional advantage that should result from the development of a network interface to a color frame buffer is standardization. Exporting a standard interface to a prototypical color frame buffer will allow client applications to use a color CRT, AMLCD or FED display, and so on, without requiring that the client maintain special knowledge of each particular display type.

We report here on an effort to create a network color frame buffer as a platform for standardized display interface research and as a demonstration of the Netstation architecture. Section 2 of this report summarizes the differences between a LAN and a bus based system and motivates the use of a network attached display. Section 3 characterizes network performance requirements for a network attached display, based on measurements of a prototype system. Section 4 introduces two issues of data representation and formatting that are potential system bottlenecks. Simulation results illustrate how the Netstation Display will effectively perform these transformation tasks. Section 5 briefly describes the Netstation Display architecture and Section 7 provides a summary of this work.

2. Substituting a LAN for a Bus

Devices in today's computer architectures are commonly memory-mapped. Memory-mapped device control and operation is achieved by addressing device registers and memory via the system bus. The communication-related properties typically presented by a system bus are:

- Small set of interface points or 'slots'
- Single sender at any one time
- Fixed bandwidth channel
- Half-duplex, broadcast transmission
- In-order delivery and data fidelity
  latency circa 100ns
- Destination accessed by memory address
- Optimized for short-length transfers
- Physically isolated set of interfaces that are commonly limited to a span of 50cm or less

This research was sponsored by theAdvanced Research Projects Agency under Contract No. DABT63-93-C-0062. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policies, either expressed or implied, of ARPA, the U.S. Government, or any person or agency connected with them.

Copyright © 1995 by USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292

Internet: Finn@isi.edu

A gigabit packet network of point-to-point channels can offer these properties:

- Large set of interface points or 'hosts'
- Multiple simultaneous senders
- Network bandwidth scales with #of channels
- Full-duplex, point-to-point transmission
- Reliable delivery with latency circa 50µs
- Destination accessed by network address
- Optimized for long-length transfers
- Interfaces are internetwork accessible and distances can span 100s of meters

A gigabit packet network has more attractive scaling properties than a bus, but it is not an efficient mechanism for the transfer of small data items such as memory words or cache lines. To read or write a word or cache line across a bus requires no software intervention and is an operation of very short duration, whereas the procedures executed in a host to reliably send or receive a message across a network may require 50µs or more.

However, longer data transfers across a bus are performed via programmed I/O or DMA. These are carried out in software or moderated by interrupt procedures that require context switching, making their overhead similar to that for message transfer. It is reasonable to consider the substitution of a gigabit network for a system bus in those cases where extremely low latency is not required and where the percentage of communication that requires extremely short transfers between hosts is small.

2.1 Suitability of Attaching Displays to a Network

Two primary issues will affect the performance of a frame buffer that is interfaced via a network: transmission latency and data bandwidth.

The latency induced by the network between the display application and the display itself is not of primary importance if the network bandwidth is sufficient. Human response time is such that a delay of 50ms is acceptable between the pressing of a button or key and the application responding with display updates. Applications that primarily play back video or stills can tolerate substantially longer latencies.

Host-to-host reliable transfer rates as high as 20K packet/second have been achieved across a gigabit LAN. At this rate, a single round-trip transport latency is 50µs, which is very small compared to 50ms response expectations.

The second concern is the bandwidth characteristics of the data flow between a typical display application and the display frame buffer. A typical transfer is a rectangle of pixels that is moved between window memory and the frame buffer. Even for a single character glyph that uses an 8-bit pixel, the rectangle that represents it requires approximately 100 bytes.

The lack of sensitivity to latency and the large data block size that is characteristic of transfers between the display application and the display suggest that a color frame buffer is a suitable device to interface directly to a network.[1]

3. Display Application Demands

The first step toward developing a network color frame buffer is to characterize the demands that will be placed upon the network, network interfaces, and software at either end of a connection between the network frame buffer and the controlling software (we refer to the latter as either the application display software or the client software).

To characterize performance demands, a representative display application was developed that simulates accessing its frame buffer across a network. It was instrumented and the traffic that passed between it and the frame buffer was characterized. The client application was an unoptimized version of the X-Windows Server that provided good subjective performance. It executed on a SPARCstation-20 that supported a 1280x1024 frame buffer of 8-bit pixels.

Three tests were run to characterize display applications: the first test examined scrolling behavior, and the second looked at slow-scan video. The first two tests were in a sense "worst case" trials, as they represent short-term peak

[1] The term 'frame buffer' is used here to mean the memory where displayed pixel values are stored. This varies with the type of display technology.

demands that are made by a display application. For most uses today, the frame buffer is static for comparatively lengthy intervals, while the user examines the displayed data before requesting more data.

Scrolling performance is an important determinant of the desired application performance, as it represents a common display operation. The most data intensive operation during scrolling is the BitBlt that moves the majority of the window up or down one line. Transferring the entire updated window between the server and the frame buffer for each scrolled line would require a data rate of approximately 30 Mpixel/sec. Some display servers make use of BitBlt hardware in the display controller to accomplish this task locally. This reduces the amount of data transferred between the server and the frame buffer to one line of an N line window.

Implementing the BitBlt locally within the frame buffer device is an extremely useful, if not necessary, optimization. That was the approach taken in the prototype network color frame buffer discussed here. Using this approach, the display application sends the bottom most (or top most) line of text, and a BitBlt remote procedure call (RPC) across the network.

Scrolling a 350 Kpixel text window that held fifty-five 80 character lines, at the maximum rate of approximately 100 line/second that is allowed by the server, provided the following data.

Xfer/sec:	100
Mean Xfer size:	6.4 Kpixels
Data Rate:	640 Kpixel/sec
BitBlt/sec:	100

The data rate of 640 Kpixel/sec is a relatively low rate for new or emerging LANs. Even for 24-bit/pixel true color, the sustained data rate is under 2 MByte/sec. Of greater interest is the packet transfer rate that this scrolling rate implies.

Packet/Xfer:	5
	(1500 byte max. transfer unit)
Xfer packets:	500 sec
BitBlt packets:	100 sec

Assuming 8-bit pixels and a 1500 byte MTU, a data packet rate of 500 per second must be supported. Since each data packet must be acknowledged, this implies that both the server and the frame buffer network interfaces must be capable of handling 1000 packets per second.

In the second test the *xbench* program was used to measure peak achieved performance. A write rate approaching 6 Mpixel/second was generated by *xbench*. This represents maximum achievable performance of the server and is not a fair estimate of what applications require in practice. However, it does provide a useful design target.

The last experiment involved Internet teleconferencing, in which an MBONE slow-scan compressed video was received, uncompressed by the server, and transferred into the frame buffer.

Xfer/sec:	175
Mean Xfer size:	7 Kpixels
Data Rate:	1.2 Mpixel/sec

Assuming an MTU of 1500 bytes, the packet rate that this represents is 875 per second. With acknowledgments that implies nearly 1800 packets per second.

In practice, the flow of data between a display application and frame buffer is extremely bursty. Substantial quiescent intervals are succeeded by intervals of very-high demand. A peak rate for a display application refreshing an entire window may reach several million pixels per second, but it remains at that peak rate for much less than a second.

In summary, the network traffic between a typical 8-bit pseudo-color display client application and a color frame buffer is bursty and rarely exceeds 2 MByte/second. This data rate is easily achievable across 100 Mbit/sec or greater LANs. The packet rate that must be supported by both the server and the frame buffer is under 2000 packets per second. This is achievable, but it requires close coupling between the network interface and the frame buffer.

4. Pixel Transfer Between Packet and Frame Buffer

The ideal network interface would allow packets to move between the network and application memory directly. This is extremely difficult to achieve. Most often, a packet is copied at least

once. The case of a packet of pixels that is being sent to a frame buffer is especially complex.

The typical IP packet that arrives at the frame buffer will contain a reliable transport-protocol payload. Within that payload would be the RasterOp RPC that (a) describes the destination region to be modified, and (b) contains the modifying data pixels.

The first step is to examine the packet for validity. Once the packet has been analyzed, the pixels in the trailing portion of the payload can be moved into the frame buffer. The most efficient way to accomplish this is to move them directly, from a packet buffer in the network interface into the color frame buffer without processor intervention. However, this will often require more than a simple data copy. It may also be necessary to transform the data format that is presented in the packet payloads into the physical format supported by frame buffer memory.

4.1 Pixel Mapping

Memory is normally organized as a vector of bytes or words that are addressed individually. A frame buffer is likewise, usually accessed as a contiguous range of addresses. However, a frame buffer represents a two-dimensional surface logically. This causes a conflict between the way that a frame buffer is physically and logically accessed.

In a conventional display, this conflict is typically resolved by the client software maintaining both a one-dimensional and a two-dimensional model of the frame buffer. Pixels in a window are most naturally addressed by [X,Y] coordinates. Software maps between the two-dimensional coordinate model and the one-dimensional memory address model that represents the physical interface to the frame buffer memory.

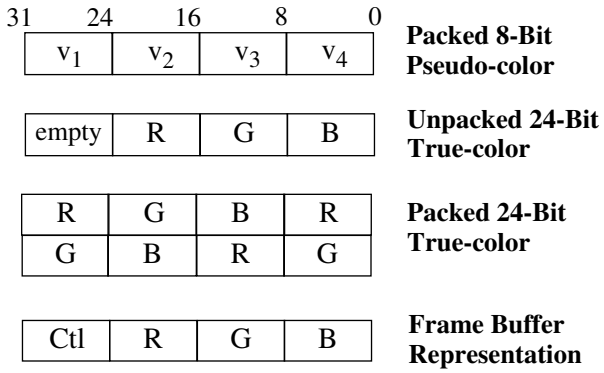
However, in a RasterOp RPC, the pixel payload in a packet is necessarily organized as a sequence of bytes, which reflects the way that they are transmitted. To move a rectangle of pixels from a source client window, across the network, and into a destination frame buffer, requires that the source map that rectangle into a vector and that the destination map that vector back into a rectangle inside the frame buffer.

If pixels are to be moved directly from the packet buffer into the frame buffer without software intervention, the transfer must be managed by an addressing engine that can map from a vector in one region of memory into a rectangle in another.

4.2 Pixel Format Conversion

A further consideration is the format of pixels within the packet. It would be least difficult if the pixels in the packet appeared in precisely the same format that is used by the frame buffer for its pixel representation. This may not be the case, and is even less likely, in light of efforts to produce a standard device independent interface.

A frame buffer is by its nature tightly coupled to its physical display device. A pixel in a frame buffer may have special control bits or a different pixel width than that supported by the client software that wishes to drive that display. The client may wish to use 8-bit pixels and pseudo-color rather than 24-bit true color. Although the physical frame buffer cannot support multiple pixel formats, a network frame buffer should be agile enough to support the common color pixel representations.



A representation of three commonly used pixel formats is shown above, along with the physical pixel representation used in the prototype.

Supporting multiple pixel formats further complicates the design task. In addition to mapping pixels stored in vector order in a packet, into pixels stored in a rectangle within the frame

buffer, the pixels must be converted from the format provided by the client into the format supported by the frame buffer.

Of the examples above, the most difficult conversion task for the prototype discussed here is presented by packed 8-bit pseudo-color format. In this case each 8-bit value must be triplicated to fill a 24-bit true-color pixel and index the color tables for R, G, and B. The color tables would have been loaded previously, to approximate the pseudo-color mapping in the client.

4.3 Multi-Dimensional DMA

An addressing engine can be designed to perform the source to destination mappings and conversions discussed above. Advances in VLSI now allow a sophisticated addressing engine of this type to be incorporated into a processor.

The Texas Instruments TMS320C80 MVP processor organizes its memory conventionally as a sequence of bytes. However, it includes as a part of its memory-bus interface a transfer controller [5]. The transfer controller is capable of performing the pixel-format conversions outlined in Section 4.1as it maps a source vector of pixels into its destination within a frame buffer.

A transfer controller of this type makes the necessary mappings and conversions available as part of a block data transfer. This allows a sequence of color pixel values to be moved from within an incoming network packet buffer into a frame buffer without directly involving the CPU in the data movement.

4.4 Transfer Controller Performance

The 320C80 development environment includes both an extensive chip simulator and an in-circuit emulator [5]. The simulator correctly simulates the processor, cache, transfer controller and memory system interactions. This was used to evaluate alternative memory architectures.

Of particular interest was how effectively the transfer controller could perform the mappings and format conversions. Accordingly, for a number of differing memory architecture choices the simulator was used to determine how many processor cycles are consumed to transfer 1024 pixels from a packet buffer SRAM into the VRAM of the frame buffer. These tests assumed nominal 40 MHz processor clock for the 320C80 and a 20 MHz bus clock. Both the VRAM and SRAM supported a two-cycle bus access. The processor memory bus size was 64 bits.

Pixel Format	ns/pixel
Packed 8-bit Pseudo-color	310
Unpacked 24-bit True-color	210
Packed 24-bit True-color	220

Table 1: No VRAM Burst

A higher performance design is achieved without raising clock rates by allowing burst-mode access to the VRAM. This change produces the timings in Table 2.

Pixel Format	ns/pixel
Packed 8-bit Pseudo-color	210
Unpacked 24-bit True-color	110
Packed 24-bit True-color	120

Table 2: VRAM with Burst

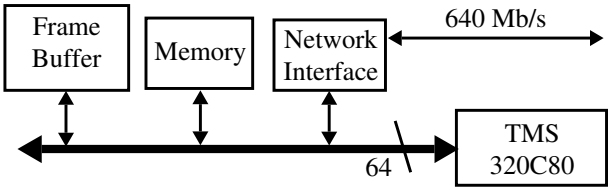
These tests indicate that a vector of pixels in packed 8-bit format can be moved into a 24-bit true color frame buffer at a rate of 4.7 Mpixel/second. That is well above the demonstrated need of 1.2 Mpixel/second required by MBONE video conferencing and approaches the SPARCstation-20 maximum rate generated by *xbench*. For applications that send packed or unpacked 24-bit true-color, 9 Mpixel/second can be transferred.

5. Network Display Architecture

A CRT was chosen for the display technology of this prototype display network peripheral. This decision reflected a desire to avoid focussing resources on new display technologies. CRT support technology is well developed.

Instead, the primary focus was the development of a color frame buffer architecture that was suited to receiving its pixel values via Internet packets received over a gigabit network. The Myrinet LAN was chosen as the network [6]. Myrinet provides 640 Mb/s point-to-point channels. The packet buffers within the network

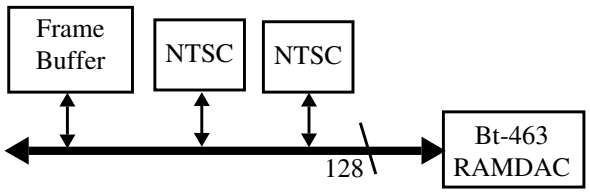
interface are visible to the transfer controller that is located inside the 320C80 chip.



A secondary goal was to provide capabilities to ease multi-party video teleconferencing applications. This strongly suggested adoption of true-color pixels, rather than 8-bit pseudo-color. Two full-rate JPEG decoders were included, as was audio output capability. Compressed JPEG data arrives over its own transport-layer connection and is transferred into the JPEG decoder FIFO.

In order to achieve sufficient bandwidth to the frame buffer RAMDAC from commonly available VRAM parts, the video-bus was made four pixels wide. The control bits within each quad-pixel that is read from the frame buffer are combined to create 32-bit control field. A key control function is the ability to choose which of three possible sources drives a particular pixel value into the RAMDAC. This is done by controlling tri-state drivers at each source.

As shown below the RAMDAC can be driven by pixels from the 1280x1024 frame buffer or from either of two NTSC frame buffers that are the target for independent decompressed JPEG streams. Additional control bits indicate where the scan-out of an NTSC row begins within the larger row of the frame buffer.



6. Related Work

Several examples exist of frame buffers that have been attached to networks. The need for scientific visualization has led to the development of high-speed storage devices and color displays as network attached peripherals. The Avaika Networks HIPPI color frame buffer is an example of this approach.

The Avaika color frame buffer is closely associated with the underlying link-layer packet format [7]. The frame buffer assumes that pixels within a packet are in frame-buffer format. Top-of-frame and start-of-data are indicated by a field within the HIPPI packet. Once a frame has been started, data from successive packets is used to fill the frame.

This design provides the ability to move pixels from the link-layer packet into the frame buffer very efficiently. A transfer rate of 19.7 Mpixel/second can be reached. However, this approach is tied to the HIPPI network specifically and to the particular pixel format supported by the frame buffer. It is not compatible with the Internet.

The approach being taken by the DAN project is to attach a framestore to an ATM network [8]. The DAN framestore in addition to acting as a frame buffer provides windowing support. The VCI of an ATM connection is associated with a particular window within the frame store. The implementation of the DAN framestore appears to be ATM-specific, with pixel data copied from individual incoming ATM cells into the framestore.

7. Conclusion

There are several issues to consider in order to interface a color frame buffer directly to a network. First, we characterize the demands placed upon the network, network interfaces, and protocol software by a typical display application. We found that peak pixel transfer rates as high as a few million per second may be required. Assuming 8-bit pseudo-color pixels, this implies a data rate of 25 Mb/s. A high-speed network is therefore a pre-requisite. For a 1500 byte MTU, the corresponding data packet rate is 2000/second.

That rate at which packets from a display application can be processed is limited by how fast the pixel values inside the packets can be moved into the frame buffer. The design must consider: (1) transfer of a vector of pixels from within a packet buffer into a rectangle within the frame buffer, and (2) conversion from the pixel format

received into the format implemented in the frame buffer.

Hardware assistance for the mapping and format conversion of pixel data is preferable to performing those operations in software. A sophisticated DMA controller can perform both operations simultaneously. Advances in VLSI now allow such controllers to be designed and incorporated into processors. The Texas Instruments 320C80 MVP processor includes a transfer controller of this type.

For a range of color pixel representations, chip-level simulations were undertaken to determine at what rate the 320C80 could perform mapping and format conversion when moving pixels from a packet buffer into a frame buffer. Rates from 3 to 9 Mpixel/second were achieved for typical clock rates and frame buffer memory architectures. This rate matches or exceeds the display application demands for current workstation displays.

8. References

[1] Finn, G.
An Integration of Network Communication with Workstation Architecture
ACM Computer Communication Review, Vol. 21, No. 5, October 1991.

[2] Hayter, M., McAuley, D.
The Desk Area Network
ACM *Transactions on Operating Systems*, October 1991, pp. 14–21.

[3] Houh, H., Adam, J., Ismert, M., Lindblad, C., Tennenhouse, D.
The VuNet Desk Area Network: Architecture, Implementation and Experience
IEEE Journal on Selected Areas in Communications, Vol. 13, No. 4, May 1995.

[4] TMS 320C80 MVP Transfer Controller User's Guide
Texas Instruments, 1995.

[5] TMS 320C80 MVP C Source Debugger User's Guide
Texas Instruments, 1995.

[6] Seitz, C. L., et. al.
Myrinet - A Gigabit/Second Local-Area Network
IEEE Micro, Vol. 15, No. 1, pp. 29-36.
February 1995.

[7] HIPPI Frame Buffer
http://www.io.com/~webpub/avaika
Avaika Networks Corporation, February 1995.

[8] Barham, P., hayter, M., McAuley, D., Pratt, I..
Devices on the Desk Area Network
IEEE Journal on Selected Areas in Communications, Vol. 13, No. 4, May 1995.