

Ethernet Interface for Head-Mounted Displays

Roger Zimmermann and Dwipal A. Desai

Integrated Media Systems Center

University of Southern California

3740 McClintock Avenue • EEB Suite 131 • Los Angeles, CA 90089-2561

[\[zimmerm,dwipalde\]@usc.edu](mailto:[zimmerm,dwipalde]@usc.edu) ♦ TEL: 213.740.7654 ♦ FAX: 213.740.5807

1. Introduction

This report summarizes our investigation of a new concept for the interface of a head mounted display (HMD) to the host workstation. Rather than connect through the video control electronics via high-bandwidth cabling, and use additional wires for motion tracking and/or audio, we would like to explore the use of a high-speed digital network interface to carry all data to the user via a standard Ethernet cable. With Gigabit Ethernet now standard on many PC-based workstations, the hardware is readily available to enable a streaming digital interface over a closed network. This technology will differ from conventional streaming video applications in that there would be no buffering of the packets and potentially no delay. We believe this would be possible over a small LAN with high-speed Ethernet. Eventually an untethered connection should be possible over a wireless link.

The key component of this design would be the real time video server. This hardware would accept high-resolution video in real time, pack it up for network transfer, and send it out over a Gigabit connection. The user would be wearing a battery-powered belt pack with real time client software/hardware to reconstruct the video, audio, and data.

We believe this radical new interface can open up entirely new categories of applications. Consider the following benefits/features:

- Light weight, single cable with extended range.
- Access to network resources, including the Internet.
- Scalable to multi-user environments.
- Scalable to emerging wireless LAN technologies.

We believe the requirements for a state-of-the-art head mounted display go beyond field-of-view and image quality. A new HMD must allow users to roam more freely and eventually, untethered. A digital interface is the first step towards realizing that goal.

2. Approach

In this feasibility study we are investigating the display transmission to an HMD via an Internet Protocol (IP) based digital connection in two stages:

1. System Architecture

We identify the various components required to packetize the video and render it back on the remote system. We studied various techniques that can be used in the system at different stages and analyze their performance using some standard applications. The system is designed such that it is independent of the transmission interface.

2. *Transmission Interface*

We analyze the various wired and wireless interfaces that are commercially available, and make some projections with those that are at the research stage.

2.1 System Architecture

A digital transmission channel over an IP-based network between an HMD and the corresponding workstation requires the following components:

1. **A video acquisition module (AM).** This component acquires the individual video frames from the application that is running on the workstation. There are several options to implement the AM. It could be implemented in hardware with an analog to digital converter that plugs externally into the workstations VGA graphics output connector. If the graphics card provides a digital DVI output, then no analog to digital conversion is necessary. A second option is to implement the AM as a software module that is – for example – an extension of the graphics card driver. In this case, video information is directly captured from the display driver and sent over to the remote side.
2. **A video compression module (CM).** This component may or may not be required. As we will elaborate in more details below, the bandwidth required by high-resolution video can easily exceed the bandwidth supported by current generation networking equipment. Depending on the networking technology used (for example wired Gigabit Ethernet versus a wireless link), little or extensive compression is required. It should be noted that compression algorithms require computations that will cause a delay in the video transmission. Therefore, the type of compression used needs to be carefully selected.
3. **A video encoding module (EM).** The raw video data must be encoded into a suitable format for transmission. Usually this means that the data needs to be packetized and possibly packet header information added to each packet. For a point-to-point application (workstation to HMD) packet headers can be kept minimal. However, if it is desired that the data should be compatible with the general Internet (for farther distance connections that include routers) then standard RTP/UDP (real time protocol [1] and user datagram protocol) must be used. The EM also generally needs to add frame start codes such that the decoding software can synchronize to the frame boundaries.
4. **IP Stack – network transmission hardware – IP Stack.** The IP stack is the software that converts packets to a format suitable for the networking hardware and vice versa. It is required at each end of the transmission channel. High performance IP stacks are available for most networking hardware technologies (wired and wireless).
5. **A video decoding module (DM).** The DM module undoes the encoding performed at module 3 by the EM.
6. **A video decompression module (DCM).** The decompression module reverses the compression performed at module 2 and its output is the raw video data.
7. **A video rendering module (RM).** The raw video data needs to be converted into a format suitable for the actual display hardware in the HMD.

All components from 1 to 7 are inter-dependent and must be carefully balanced to achieve good overall performance. Therefore, we will partition our further discussion into a *wired* and a *wireless* approach and discuss the relevant issues there.

One of the main concerns with a HMD application such as a virtual reality environment (VRE) is how quickly the display can be updated when a user turns her head. If the lag is too extensive the user may perceive the VRE as unstable. The lag depends mostly on two components: the head tracker and the display update. Head trackers are now sufficiently fast that they do not pose a problem. Current research suggests that a lag of approximately 40 milliseconds is tolerable [4]. Consequently, this leaves very little

time for video processing and transmission and any encoding/compression algorithm must be sufficiently fast.

3. Experiments

As part of this feasibility study we have conducted experiments to explore how applicable current technology is to our goal of sending data to an HMD via an IP based transmission channel. We will discuss some of the practical aspects of such a system, and analyze some of the technologies that are available.

We consider an architecture based on the *Host Workstation*, which runs the software that generates the video stream, and a *Display Client* which is a small, wearable device that is connected to HMD. Figure 1 illustrates this setup.

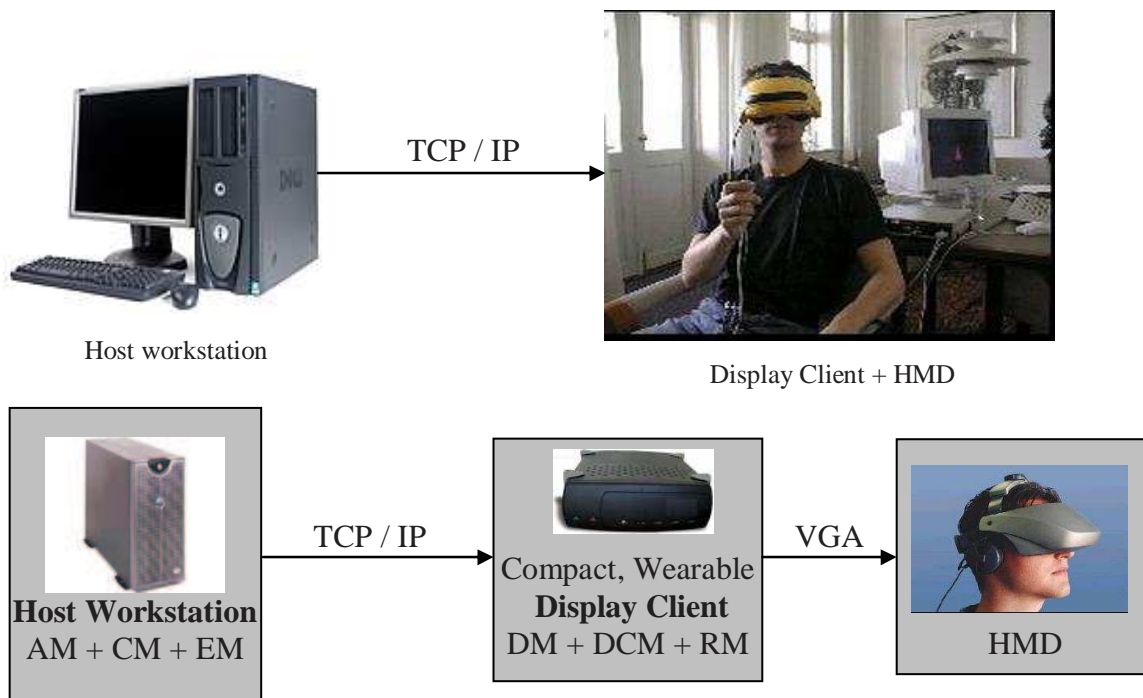


Figure 1: The Host Workstation serializes the video data and transfers it to the Display Client, which renders it on the HMD.

We distinguish three different approaches to drive the Display Client:

1. *Transfer the actual video buffer to the Display Client (i.e., rendering takes place on the host workstation).*

This method reads the video buffer on the workstation graphics board and transmits it to the display client. It is a very general solution for remote display and requires not very complex hardware and software on the client side. It is also highly compatible with many applications and is not dependent on special capabilities of the underlying graphics hardware. The client software can be lightweight and be incorporated in an embedded system. The disadvantage of the method is that it requires simple, but very fast transmission and rendering capabilities. Every pixel of the host frame buffer must be transmitted many times per second to the HMD. Furthermore, it fails to utilize features like

hardware acceleration that are provided by newer display cards. Many high performance graphics applications make extensive use of hardware display acceleration to achieve good performance.

2. *Transfer rendering commands and execute them on the Display Client (rendering takes place at the Display Client).*

This method captures the rendering commands that are passed to the display card, for example OpenGL commands, and transmits them to the display client along with the standard display buffer. This approach has a major performance benefit over the first method as the applications can take advantage of hardware rendering. Since it does not have to transmit the entire frame buffer, there is a large reduction in the network traffic.

3. *Use hardware devices to capture the VGA/DVI output and transfer it to the Display Client.*

This method uses specialized hardware devices that can convert the DVI output from the display card to a network data stream, which is then decoded by the display client. It has an advantage over the above methods in that it is completely application and even operating system independent. However such hardware devices are not available commercially.

We have analyzed the software based methods 1. and 2. using two widely used programs, VNC and X Windows.

3.1 Virtual Network Computing (VNC)

VNC utilizes the Remote Frame Buffer (RFB) protocol [16] to provide remote access to a graphical user interface. It requires a special client that can connect to the VNC server, decode the RFB stream and display it. The VNC client (VNC Viewer) works at the frame buffer level and has minimal hardware requirements. It is also available on a wide range of platforms, including Linux, Windows and OS X. It has a very small footprint and can be incorporated in an embedded system. Figure 2 illustrates the software components of such a setup in a Linux environment.

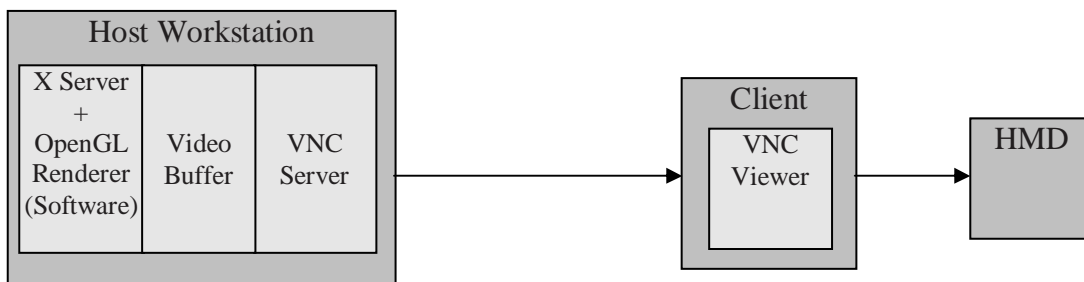


Figure 2: The VNC Server reads the video buffer from the X-Windows server, compresses it and sends it to the VNC Viewer over a network.

VNC compresses the stream before transmitting it. It has a pluggable architecture and supports multiple algorithms to perform this compression. In its default mode, it uses Hextile Encoding [16] to compress the data stream and works best for window-based GUIs. This method splits a given frame into tiles of a predetermined size and then performs compression on each of these tiles using a suitable compression method.

As VNC works by reading the frame buffer from the X-Windows server, it is necessary for the buffer to contain all the display data. However, in the newer display cards, which support hardware based

rendering, this is not always the case. For example, 3D OpenGL based display components are not rendered by the X-Windows server but are rendered directly by the display card graphics processor. Therefore, VNC cannot forward the parts of the screen that contain hardware rendered OpenGL objects and consequently those parts appear “black” on the client display. A workaround for this is to enable software rendering for all OpenGL commands. However, this will affect the performance of the system and it will not be able to take advantage of the features provided by fast display cards.

A possible extension for VNC would be to enable it to capture the OpenGL commands from the X-Windows server and transfer them along with the pixel data. These commands would need to be decoded on the client side. This approach would make the VNC Viewer more complex as it will now need a fast display card to render OpenGL commands. This method has not yet been implemented in the VNC software.

3.2 The X-Windows System

Based on the X Protocol [18], X-Windows is the most common way of forwarding a display to a remote machine in Unix based computing environments. The server machine runs an X-Windows server which forwards the display to the client, which in turn drives the HMD. The X client is lightweight and can be incorporated in an embedded, wearable system.

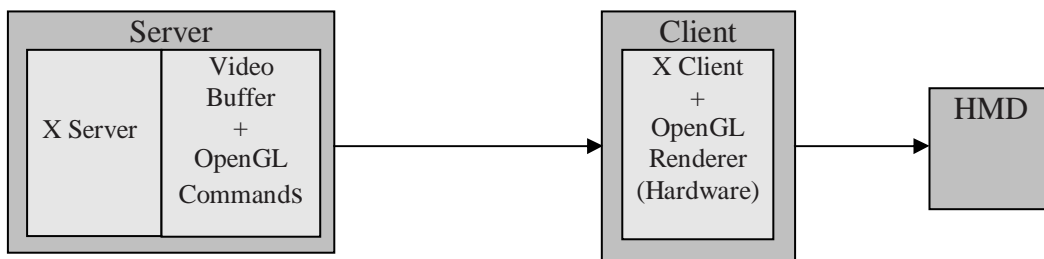


Figure 3: The X Protocol forwards the OpenGL commands to the Client, eliminating the need for software rendering at the server end.

Similar to VNC, the X-Windows system forwards the content of the video buffer to the client, which means that areas with hardware rendering are not visible on the client. However, an “Indirect Rendering” mode for OpenGL is provided, in which the X-Windows server captures OpenGL commands and forwards them along with other display contents. The commands are then rendered by the client system, which must have a video card supporting hardware OpenGL rendering. This method provides multiple advantages over the standard method of forwarding the video buffer contents. First, it allows any graphics application to utilize the power of hardware acceleration for its rendering needs. Second, techniques such as display scaling can be utilized to obtain a higher resolution client display without increasing the CPU utilization or the required transmission bandwidth. OpenGL forwarding is readily available in X-Windows, and thus can be employed without further development. Figure 4 shows an experiment where we forwarded the display of a 3D OpenGL based game to a client rendering machine and Figure 6 illustrates a second, high resolution OpenGL application. The display appears smooth and with low latency (performance numbers are described later on). We used a standard PC for the client system. However, we believe that custom hardware could be built with the necessary capabilities. Specifically, a fast network interface (possibly Gigabit Ethernet) and a 3D capable graphics processor would be necessary.

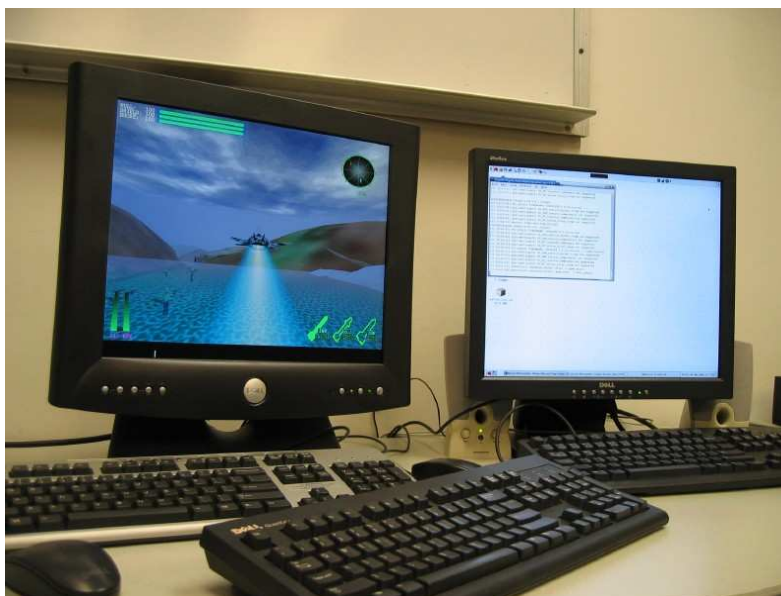


Figure 4: An OpenGL based 3D game [25]. The game is being run on the server (right) with the display exported to the client (left) using OpenGL forwarding.

OpenGL forwarding via the X Protocol substantially lowers the transmission bandwidth requirements. However it produces an uncompressed stream of data which generates enough traffic so that it cannot be transmitted on a wireless link (depending on the application). To reduce the bandwidth, there is an extension to X-Windows server available called **LBX** (Low Bandwidth X) [21], which compresses this stream.

3.3 X-Windows Server + LBX

LBX [21] acts as a proxy to the X-Windows server and compresses the command and data stream before it is transmitted. In our experiments it produced very good compression results and kept the bandwidth below 30 Mb/s. This may be low enough to be usable over a 802.11a wireless link. However, LBX adds a substantial delay to the stream transmission, which makes it most likely unsuitable for real time interactive applications.

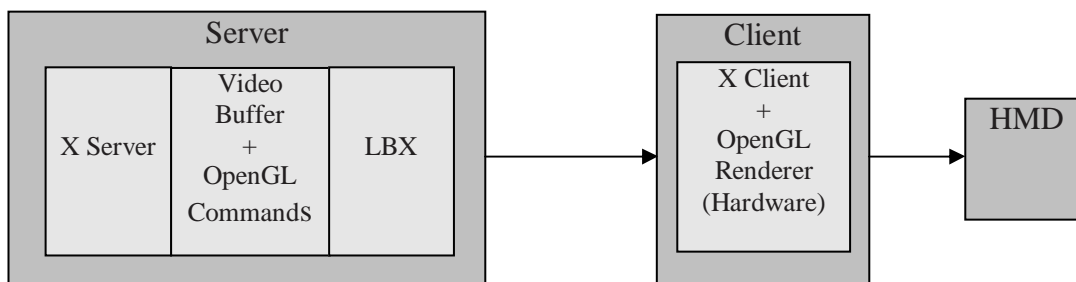


Figure 5: LBX compresses the command and data stream before it is transmitted.

LBX uses real time software compression techniques, due to which its working is limited by the processing power of the host workstation. If the application generates a raw stream of data above a certain bandwidth, LBX will not be able to handle it and will skip some updates which results in a “jerky” display.



Figure 6: A high resolution OpenGL application [24].

3.4 Comparison between Video Buffer Forwarding and OpenGL Forwarding

Table 1 shows a comparison between Video Buffer Forwarding and OpenGL Forwarding.

	<i>Video Buffer Forwarding</i>	<i>OpenGL Forwarding</i>
System Requirements	Requires more processing at server end but will work with a less powerful client.	Requires lower processing at the workstation end, but needs a faster client with hardware rendering support.
Implementation	Easier to implement as only the video buffer has to be taken care of, and also because it can be independent of the underlying hardware.	Both server and client are more complicated as they have to handle OpenGL requests along with the standard requests.
Resolution	Resolution is dependent on the workstation video buffer, and cannot be scaled without affecting video quality.	Scaling can be easily done on the client end in order to render objects at higher resolutions.
Bandwidth Requirements	If uncompressed, this method has very high bandwidth requirements.	It requires a lower bandwidth as it does not transfer the complete video buffer.
Application Compatibility	This method only works on the video buffer, and hence it is compatible with most of the applications.	The OpenGL commands are dependent on underlying hardware which makes this approach less compatible.

Table 1: Comparison between Video Buffer Forwarding and OpenGL Forwarding.

3.5 Comparison between VNC, X and X+LBX

We compared the various methods by running actual applications. The comparison parameters are highly dependent on the application. X-Windows with OpenGL forwarding produced the best quality display with minimal latency.

	<i>VNC</i>	<i>X-Windows-OpenGL</i>	<i>X-Windows-OpenGL with LBX</i>
System Requirements	Low client requirements (Software rendering on Server)	High client requirements (Fast graphics card for OpenGL rendering)	High client requirements (Fast graphics card for OpenGL rendering)
Resolution	1024x768	Any	Any
Bandwidth	1-50 Mb/s	1-350 Mb/s	1-30 Mb/s
Latency	5-50 ms	0-30 ms	1-2 sec

Table 2: VNC, X and X+LBX comparison.

	<i>VNC</i>	<i>X-Windows</i>	<i>X-Windows + LBX</i>
Web browsing	1 Mb/s	5 Mb/s	700 Kb/s
Video	45 Mb/s	70 Mb/s	20 Mb/s
Interactive 2-D Game	10 Mb/s	30 Mb/s	5 Mb/s
Interactive 3-D Game	-	25 Mb/s	8 Mb/s
High Quality 3-D Graphics Application	-	300 Mb/s	25 Mb/s

Table 3: Bandwidth requirements for different applications.

3.6 Video-based Applications

A high end HMD may provide display resolutions of 1280x1024 pixels with 24 bits/pixel color depth. Assuming a refresh rate of 30 frames per second (fps) the data transfer rate will be approximately 950 Mb/s. For a stereo capable display, twice this bandwidth will be necessary. 950 Mb/s is approaching or exceeding the maximum transfer rate of commercial Gigabit Ethernet. Hence, we investigated various methods for reducing the bandwidth, such as Hextile encoding used by VNC or OpenGL forwarding.

These methods work well with rendered graphics objects, such as 2D or 3D models. If the stream needs to be handled as raw video then both the methods perform extremely poorly. One of our primary concerns while investigating the different approaches was to keep the latency to a minimum. As we need to keep the latencies below 40 milliseconds [4], there is very little time for video processing and transmission and any compression/decompression algorithm must be sufficiently fast. Some compression methods that we considered are:

1. Lossless video compression (2:1 ratio, 475 Mb/s)
2. MPEG-2 compression (20:1 to 50:1 ratio, 50 to 20 Mb/s)
3. MPEG-4 compression (50:1 to 100:1 ratio, 20 to 10 Mb/s)

The MPEG compression algorithms provide very high compression ratios, which are achieved with

temporal inter-frame compression. This requires multiple frames to be buffered and causes a latency in the stream (VCV latency, [10]). Present systems cause latency of 200 to 250 milliseconds while encoding a high-definition MPEG-2 video stream [5, 6, 7]. For HMD applications the delay is most likely too long. The professional HDTV MPEG-2 encoders require special interfaces as input (HD-SDI, serial digital interface). The video data may not be in accordance with those interfaces [7]. MPEG-4 encoder hardware that can support HDTV quality encoding is still not commercially available. Even the lossless compression algorithms like Huffiyuv [3] need some buffering in the stream in order to achieve good compression ratios.

Another issue with using these compression methods is that they are extremely slow, especially in handling high resolution streams. There are just now new chips becoming available with low enough power consumption to be usable for battery-powered devices. For example, the Super ENC-III from NTT is capable of doing either encoding or decoding of 1280x720, 30 fps at < 1.5 watts [6]. These chips are now being used for a new generation of low-cost portable high-definition camcorders. There are no commercial chips currently available that produce HD quality MPEG-4 stream in real time.

At present time, custom hardware would need to be built to perform the compression / decompression. Whether this can be done with a battery-powered device requires further investigation. As mentioned earlier, a major issue with MPEG based encoding/decoding is the additional latency that the compression introduces. As latency is a major concern in HMDs, these methods do not work very well for interactive applications.

4. Transmission Channel

Our system is designed to work on a standard IP based network interface. So, we can select any transmission method according to application bandwidth requirements and the complexity of display client. We studied the various wired and wireless channels to use as our communication interface.

4.1 Wired Transmission Channel

Current commercial wired networking technology realizes cable signaling rates of up to 1 Gigabit per second (Gigabit Ethernet or GigE) with very affordable hardware components. There are currently two transmission technologies widely used: fiber-optic and copper connections. Fiber-optic cables allow longer cable lengths without regenerating the digital signal, but the hardware components are more expensive than for copper based transmissions, which use standard Category 5 networking cables. The throughput performance of both technologies is similar. The usable data bandwidth on GigE can approach 900 or more Mb/s [2] when careful provisions are observed. For example, the maximal transfer unit (MTU) should be set to 9000 bytes per Ethernet frame versus 1500 bytes for standard frames (the larger frames are often called *jumbo* frames).

For applications with lower bandwidth requirements, we can also use the standard 100 Mb/s Ethernet, or the IEEE-1394 [23] Firewire interface. We tested the 1394 interface performance between two machines by using IP over 1394 [20] modules, and got a throughput of 135 Mb/s. IP over Firewire is still in the early development stages, and is currently not very efficient. Using this interface, we can theoretically achieve a throughput of 350 Mb/s [22, 23].

4.2 Wireless Transmission Channel

The ultimate goal for HMD applications is an untethered, wireless video connection. Wireless interfaces have only recently been commercialized. Current wireless interface based on IEEE 802.11a/802.11g provides a maximum bandwidth of 54 Mb/s (30 Mb/s in practice).

However, wireless digital transmissions are the focus of much research these days and future technologies like Firewireless (100 Mb/s or 400 Mb/s) [9] and 13 GHz digital wireless transmission system for VGA signals [8] are emerging.

As current wireless interfaces have very low bandwidth compared with wired interfaces, we need to lower down bandwidth requirements of our application using techniques like LBX. Using LBX with OpenGL forwarding allows us to produce high resolution display using very low bandwidth. Latency is again an issue here, as LBX buffers a large part of the stream before compressing and transmitting it.

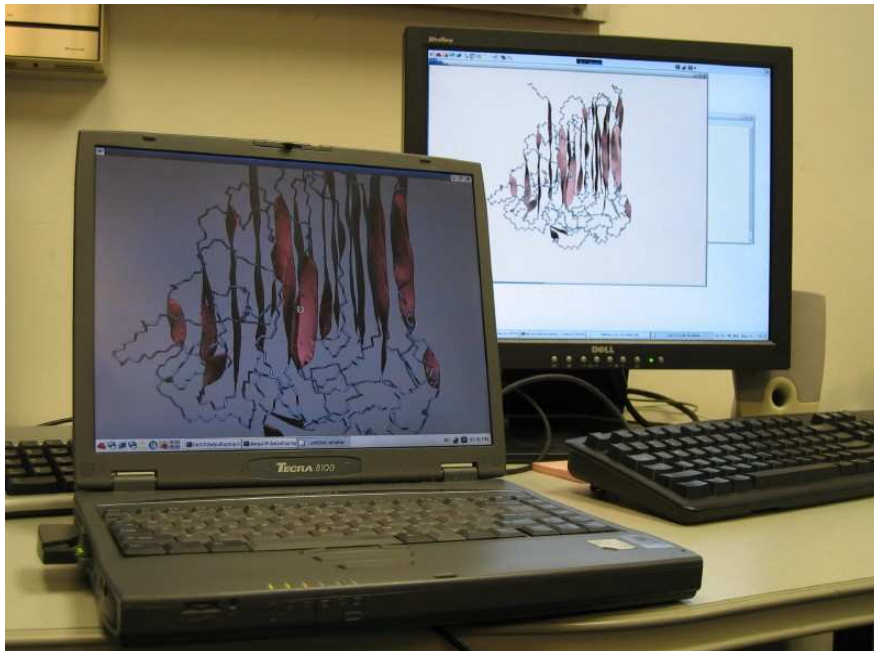


Figure 7: Application using an 802.11b wireless connection.

4.3 Transmission Channel Comparison

Table 4 shows the actual bandwidth utilized by a typical application [24] with some of the interfaces that are currently available.

<i>Interface</i>	<i>Actual Bandwidth Achieved</i>
100 Mb/s Ethernet	94 Mb/s
Gigabit Ethernet	> 350 Mb/s
Wireless 802.11b	3.5 Mb/s
IP over Firewire	130 Mb/s

Table 4: Transmission channel comparison.

5. Conclusions

In this report we have identified the different components that need to be carefully designed for an IP-based (e.g., Ethernet) digital video transmission from a workstation to a HMD. We have also presented some of the challenges that would need to be overcome (low latency, compression) and proposed some solutions for them. We evaluated several different technologies for the system architecture and identified possible issues with them. It appears that an X-Windows based approach with OpenGL command forwarding can provide high resolution and low latency. For an actual implementation a wearable, battery powered client device would need to be built with a high-speed (Gigabit Ethernet) network interface and a graphics processor with OpenGL rendering capabilities. In light of the capabilities of current generation laptop computers, this seems doable. However, this solution would be restricted to OpenGL compatible applications.

6. References

- [1] **RTP: A Transport Protocol for Real Time Applications.** H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson. RFC1889, 1996, URL: <http://www.ietf.org/rfc/rfc1889.txt>.
- [2] **Performance Evaluation of Copper-Based Gigabit Ethernet Interfaces.** P. Gray, A. Betz. 27th Annual IEEE Conference on Local Computer Networks (LCN'02), November 06 - 08, 2002, Tampa, Florida, p 0679.
- [3] **Huffyuv v2.1.1.** Huffyuv is a very fast, lossless video codec. URL: <http://www.cdr.cz/link/568>.
- [4] **Tolerance of Temporal Delay in Virtual Environments.** Robert S. Allison, Laurence R. Harris, Michael Jenkin, Urszula Jasiobedzka, James E. Zacher. Virtual Reality 2001 Conference (VR'01) March 13 - 17, 2001, Yokohama, Japan.
- [5] **MPEG-2 HDTV I & P Encoder,** http://www.xilinx.com/ipcenter/catalog/search/alliancecore/duma_hdtv.htm.
- [6] **Super NTT III HDTV-MPEG-2 Encoder/Decoder,** <http://www.ntt.co.jp/news/news03e/0302/030213.html>.
- [7] **Standalone HDTV MPEG-2 Encoder,** <http://www.saeurope.net/downloads/pictor.pdf>.
- [8] **13 GHz digital wireless transmission system for VGA signals,** <http://www.vtq.de>.
- [9] January 27th, 2000, **NEC unveils IEEE 1394 'Firewireless' home LAN,** <http://www.1394ta.org/Press/2000Press/jan.htm>.
- [10] **Levels for MPEG-4 Visual Profiles,** <http://www.m4if.org/resources/profiles/index.html>.
- [11] **High-Definition Video Capture System,** <http://www.zaxel.com/digitalcinemasolutions/high-definition.html>
- [12] **Amphion MPEG-4 Decoder,** <http://www.us.design-reuse.com/cgi-bin/TRACKING/exit.pl?url=http://www.amphion.com/video.html>
- [13] **HDTVxpress: PCI Based Real Time HDTV Compressor,** http://www.lsilogic.com/news/product_news/pr20030401.html
- [14] **Standalone HDTV MPEG-2 Encoder,** <http://www.saeurope.net/downloads/pictor.pdf>
- [15] **Sigma Designs HDTV MPEG-4 Decoder,** http://www.sigmadesigns.com/news/press_releases/030108.htm
- [16] **VNC Remote Frame Buffer (RFB) Protocol,** <http://www.uk.research.att.com/vnc/rfbproto.pdf>
- [17] **Palm VNC,** <http://www.wind-junkie.de/PalmVNC>

- [18] **XFree86**, <http://www.xfree86.org/>
- [19] **X Server for Windows with OpenGL Support**, <http://www.starnet.com/products/>
- [20] **Ethernet over IEEE 1394 (eth1394) for Linux**, <http://www.linux1394.org/eth1394.html>
- [21] **Low Bandwidth X**, <http://www.linux.org/docs/ldp/howto/mini/LBX.html>
- [22] **IEEE 1394 Trade Association**, <http://www.1394ta.com/>
- [23] **IEEE 1394 OHCI Specification**,
http://developer.intel.com/technology/1394/download/ohci_11.htm
- [24] **Astral** (OpenGL Demo), <http://astral.scene.hu>
- [25] **Reaper** (OpenGL Based 3-D Game), <http://sourceforge.net/projects/reaper3d>