BASS Application Sharing System*

Omer Boyaci and Henning Schulzrinne
Department of Computer Science
Columbia University
{boyaci,hgs}@cs.columbia.edu

Abstract

Application and desktop sharing allows sharing of any application with one or more people over the Internet. The participants receive the screen-view of the shared application from the server. Their mouse and keyboard events are delivered and regenerated at the server. Application and desktop sharing enables collaborative work, software tutoring, and e-learning over the Internet. We have developed an application and desktop sharing platform called BASS which is efficient, reliable, independent of operating system, scales well via heterogenous multicast, supports all applications, and features true application sharing.

1. Introduction

Application and desktop sharing allows two or more people to collaboratively work on a single document, drawing or project in real-time. Some applications like Netbeans and Google Docs are collaboration-aware and allow more than one person to work on the same document at the same time [5, 3]. However, most of the other applications are not collaboration-aware. Fortunately, collaboration features can be added to these applications transparently. There are two models for collaboration-transparent application sharing: application-specific and generic. An application-specific solution allows to share a specific application such as Microsoft Word [24], while a generic one allows to share any application. Application-specific solutions are expensive in terms of engineering cost and they may not allow to use all features of the application [24]. Also, to participate in a sharing session, all participants must have a copy of the shared application. In the generic model, the application can be anything such as word processor, browser, CAD/CAM, Power point or movie editor. Also, the participants do not need to install the application. One disadvantage of generic application sharing is that its generic nature makes it less efficient as compared to the application-specific model in certain scenarios. We have developed an application and desktop sharing system, BASS, based on the generic model.

The main challenges of application and desktop sharing are scalability, reliability, true application sharing, operating system independence, and performance. We believe that an application and desktop sharing system should be operating system independent because participants may use different operating systems. BASS protocol is very simple to implement on any system, and we have developed a Java client for the participants. Scalability is very important because sharing sessions may involve many participants. BASS scales quite well because it supports multicasting. Systems which support multicasting scale well, but participants will end up with corrupted screen-views if the system is not designed reliability in mind. Multicast support of BASS is developed reliability in mind. The sharing system should be efficient in the sense that it should transmit only the changed parts of the screen, and it should not consume all the bandwidth and CPU resources while doing this. BASS uses the most efficient technique, a mirror driver, to detect changed regions of the screen.

Application sharing differs from desktop sharing. In desktop sharing, server distributes any screen update. In application sharing, server distributes screen updates if and only if they belong to the shared application's windows. Some sharing systems such as UltraVNC [11] and MAST [20] claims application sharing support. However, they consider only the boundary of the shared window which is not enough. Other non-shared windows may cover the shared window or shared application may open new child windows such as those for selecting options or font. A true application sharing system must blank all the non-shared windows and must transfer all the child windows of the shared application. For example, if a user wants to share only the "Internet Explorer" application, which has the title "Windows Live Hotmail - Windows Internet Explorer", from the desktop seen in (Figure 1), then the participants should only see the main and the "Internet Options" win-

*This work was supported in part by a grant from FirstHand Technologies.

dows. BASS (Figure 3) displays only these two windows with a correct size while blocking the desktop background and the non-shared windows. MAST, discussed in the following section, could not display the shared application in correct size and could not block the non-shared application and desktop background (Figure 2). Similarly, UltraVNC could not block non-shared windows and could not transmit the child windows correctly (Figure 4).

Current sharing solutions perform poor if the user wants to share photos or movies. They use the same encoding for texts, computer-generated images, movies, and photographic images. Lossless encodings give poor performance for movies and photographic images. Lossy encodings generate visual artifacts around texts and computer-generated images such as straight lines. THINC and RDP can play full motion movies if the bandwidth between the user and participant is tens of Mbps [11, 9]. Due to their high bandwidth requirements, they do not scale well, and they do not perform well under realistic bandwidths such as a few Mbps. BASS is the only system which uses different encodings for different regions of the screen. BASS uses Theora video codec to stream movies [10], JPEG to transmit images, and PNG for the rest.

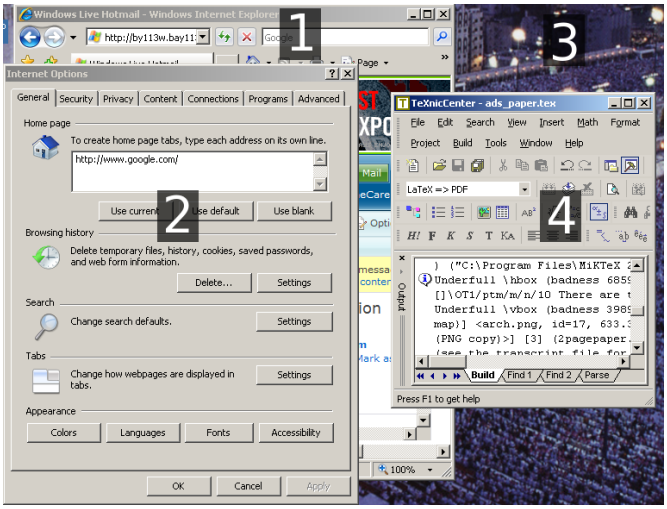

Figure 1. Desktop with overlapping windows

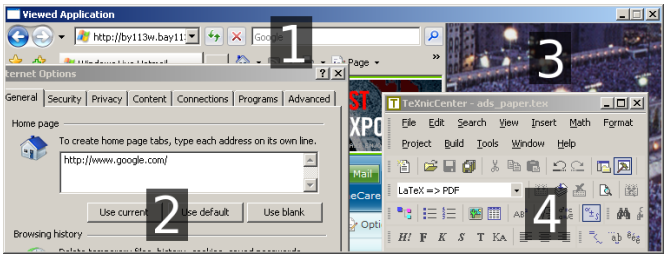

Figure 2. Mast client view

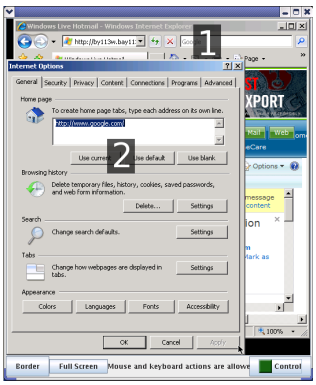

Figure 3. BASS

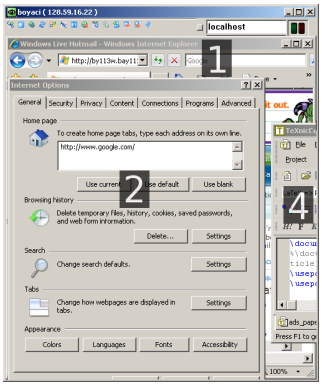

Figure 4. UVNC

Section 2 discusses the related work and why they are inadequate. System architecture is discussed in Section 3, and the details of the BASS protocol are discussed in Section 4. Client and server architectures are explained in Sections 5 and 6 respectively. Section 7 compares the performance of BASS to other systems in terms of bandwidth and frame rate. Finally Section 8 summarizes our work.

2. Related Work

The X Window System is a network-transparent window system. Several X protocol multiplexors such as SharedX, DMX, XMX, and CCFX have been developed [14]. X Window System relies on heavy X servers on the client side. Therefore, multiplexors fail to support heterogeneous X servers and late joiners.

Microsoft has Windows Meeting Space for Windows Vista and Netmeeting for Windows XP. Netmeeting was released in 1999 for Windows 98; in our tests it fails to display pop-ups and menus. Windows Vista introduces application sharing feature as part of Windows Meeting Space, but all the attendees must use Windows Vista. VNC [22] is a cross-platform open source desktop sharing system, but it also does not have application sharing feature. UltraVNC [11] claims to support the application sharing feature, but it has failed in our tests due to following problems: the cursor position did not match, there is no privacy, new windows belonging to same application are not included and long menus are not shown properly. VNC uses a client-pull based transmission mechanism which performs poor compare with server-push based transmissions under high round-trip time (RTT). SharedAppVnc [23] supports true application sharing, but the delay is unacceptable. It uses a lossy codec and does not support multicasting.

VNC, Netmeeting, Windows Meeting Space and Mac OS X Leopard Screen Sharing all rely on unicast only, so they do not scale well. Sharing an application via uni-

	OS Independent	Scalable	Application Sharing	Remote Control	Recording
VNC	+	-	-	+	+
Windows Meeting Space	-	-	+	+	-
Leopard Screen Sharing	-	-	-	+	-
TeleTeachingTool	+	+	-	-	+
Mast	+	+	-	+	-
BASS	+	+	+	+	+

Table 1. Comparison of Related Work

cast increases the bandwidth usage linearly with the number of participants. For instance, Microsoft suggests Windows Meeting Space for groups of no more than 10 users. The TeleTeachingTool [25] and Multicast Application Sharing Tool (MAST) [20] use multicasting to overcome this limitation. TeleTeachingTool adds multicast support to VNC servers; however, it is developed just for online teaching, so it does not allow participants to control the shared desktop. Also, it does not support true application sharing due to its underlying VNC system. MAST allows remote users to participate in via their keyboard and mouse, but its screen capture model is based on polling which is very primitive and not comparable to current state of the art capturing methods like mirror drivers, discussed in Section 6.1. Also, MAST does not support true application sharing due to problems which we have mentioned in the previous section (Figure 2). Although both TeleTeachingTool and MAST use multicasting for scalability, they do not address the unreliable nature of UDP transmissions. UDP does not guarantee delivery of packets. Even if the packets are delivered, they may be out of order. In order to compensate for packet loss, the TeleTeachingTool and MAST periodically transmit the whole screen which increases the bandwidth and CPU usage. Table 2 compares the sharing systems discussed so far. Nieh et al compared sharing systems in detail [18].

3. System Architecture

BASS is based on client-server architecture. The server is the computer which runs the shared application. The participants use a lightweight Java [4] client application for connecting to server, and they do not need the shared application. Clients receive screen updates from the server and send keyboard and mouse events to the server.

The Java client works in almost every operating system. The server could not be written with Java due to lack of Java's low level support. For example, Java does not have OS level windowing information and can not learn screen updates from the OS. Therefore, there should be a server for each operating system and we have developed a Windows XP server and mirror driver. Mirror driver is the best known technique for capturing screen update events and will be discussed in detail at Section 7.1. The mirror driver runs in kernel mode and notifies the user mode server when it detects changes in the GUI of the shared application. The server then prepares a RTP packet containing encoded image of the updated region. RTP allows the clients to re-order the packets, recognize missing packets and synchronize application sharing with other media types like audio and video. The screen updates can be encoded with PNG [8], JPEG [12] or Theora according to their characteristics. PNG is an open image format which uses a lossless compression algorithm [17, 16] and more suitable for computer generated images. JPEG is lossy, but more suitable for photographic images. Theora is an open video codec comparable to H.264 and suitable for movie encoding.

The server supports both multicast and unicast transmissions. For unicast connections, either UDP or TCP can be used. TCP provides reliable communication and flow control. Therefore, it is more suitable for unicast sessions. TCP clients may have different bandwidths, so we have developed an algorithm which sends the updates at the link speed of each client. For UDP clients, the server controls the transmission rate because UDP does not provide flow and congestion control. Several simultaneous multicast sessions with different transmission rates can be created at the server. Briefly, the server can share an application to TCP clients, UDP clients, and several multicast addresses in the same sharing session.

Participants of a sharing session can join in anytime, and they need the full screen buffer before receiving partial updates. Therefore, they send a RCTP-based feedback message, Full Intra-frame Request (FIR) [21], after joining in the session. The server prepares and transmits the image of the whole shared region after receiving a FIR message. Preparing a full screen update is costly in terms of CPU, so the sharing server stores the generated image for some time.

Although multiple users could receive the screen updates simultaneously, clearly only one of them can manipulate the application via keyboard and mouse events. BASS uses the Binary Floor Control Protocol (BFCP) [15] to restrict the control of the application to a single user. BFCP receives floor request and floor release messages from clients; and then it grants the floor to the appropriate client for a pe-

riod of time while keeping the requests from other clients in a FIFO queue. All BFCP messages, keyboard and mouse events are transmitted directly to the server using TCP. For mouse and keyboard events Java's own key-codes are used because these events are captured from a Java client and regenerated by a Java component at the server. These key-codes are publicly available because Java is an open-source project.

We have also added a recording feature to BASS. Participants can record the sharing session to a file. This file may be used for watching the session locally or streaming to multiple receivers simultaneously. This feature is very useful for preparing lectures or software tutorials for future use.

4. The BASS Protocol

BASS clients and servers communicate using the BASS protocol which defines the packet structures and message types. The protocol defines five messages from server to client and two messages in the reverse direction (Table 4). Each new client needs one "open new window" message for each shared window. For TCP clients server transmits this message right after the connection establishment. UDP-based clients send a "full intra-frame request" to the server when they join in the session. Receiving this FIR message, the server first transmits "open new window" message for each shared window. Then, it transmits complete image of each shared window via "region update" messages.

Messages from server to client
Open New Window
Close Window
Window Size Update
Region Update
Move Rectangle

Messages from client to server
Full Intra-frame Request (FIR)
NACK request

Table 2. BASS Message Types

RTP Header	Message Type	Window ID	MS Payload

Table 3. BASS Message Structure

Each protocol message consists of a RTP header, message type identifier, window identifier, and message specific payload (Table 4). The message specific payload structure and length are determined by the message type. Neither TCP nor RTP declares the length of the RTP packet. Therefore, RTP framing [19] is used to split RTP packets for TCP clients.

5. Client Architecture

The BASS client is very simple and lightweight compare with the server. It receives screen updates from the server and displays them to the participant. It is completely stateless in the sense that it can disconnect and reconnect to the server. Due to its simplicity, clients for different platforms can be easily developed. A Java BASS client has been developed in our lab. BASS clients can listen for or initiate connections. Participants of a sharing session can be view-only or they can request input control from the server. To request input control from server, a user presses the "control" button in the GUI of the client application (Figure 3). The client sends a floor request message, and then the server responds with a "granted" or "request queued" message. The server grants the floor immediately if nobody else currently controls the floor. Otherwise, the request will be queued in a FIFO queue, and the floor will be granted to the requesters one-by-one automatically. Users can release the floor by pressing the "control" button again. After the server grants the floor, the client captures all keyboard and mouse events locally and transmits them to the server via RTP messages.

6. Windows Server Architecture

Windows server allows Windows XP users to share an application with other participants. Windows server has two main compenents: kernel-mode mirror driver and user-mode sharing server. These two components work together and communicate via shared memory. Basically, the mirror driver tracks the updated regions of the screen, and notifies the user-mode sharing server about these updates. The user-mode sharing server learns the updated regions, prepares region updates, and transmits these updates to the participants(Figure 5). The sharing server also receives and regenerates mouse and keyboard events from participants. These two components are examined in detail in the following sections.

6.1. Mirror Driver

Mirror driver is a display driver that mirrors the drawing operations of a physical display driver. Windows calls the physical and mirror display drivers with the same GDI (graphics device interface) commands. It is the most efficient way of learning screen updates because operating system tells the exact coordinates of screen updates. We had to develop our own mirror driver for the sharing server because there is no free and open source mirror driver. Re-

mote Desktop Connection and VNC also use their own mirror drivers to efficiently learn the screen updates. Stability and correctness are very important for kernel-mode components because they may easily cause restart or blue screen of death. Our mirror driver is completely stable such that we have not observed any crashes at all for last two years.

We have used the shared memory approach to establish a communication channel between mirror driver and the sharing server. Both mirror driver and the sharing server map the same region of the memory to their own address spaces. The shared memory consists of a frame buffer to keep the screen state and a ring buffer which is used by the mirror driver to insert update commands and coordinates. There are two types of commands BitBlt and MoveRect. Windows notifies the mirror driver for an update, and then mirror driver inserts this update to the ring buffer with command type and the coordinates of the region. In case of application sharing, the server computes the bounding rectangle of the shared application windows and informs the mirror driver about the tracking region. The mirror driver only tracks this specific region instead of the whole desktop which decreases CPU overhead.

6.2. Server Architecture

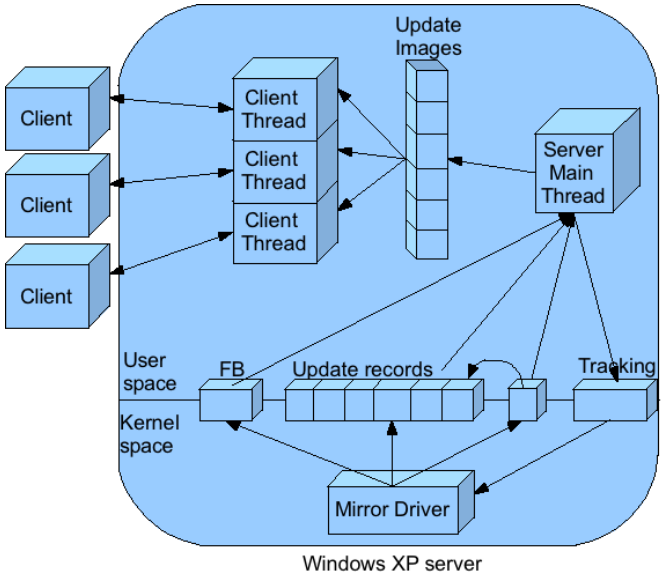

Figure 5. Windows XP server architecture

The Windows XP sharing server is a user-mode process which complements the mirror driver. Mirror driver keeps track of the frame buffer and the list of updated regions. The sharing server does rest of the duties. It handles connection establishment. It optimizes, compresses and transmits screen updates. Keyboard and mouse events are also handled by the sharing server (Figure 5). The multi-threaded sharing server can serve multiple clients simultaneously. The server can wait for incoming connections and it can also connect to clients directly if instructed by the user. The sharing server has been designed considering the following challenges. Participants may have different bandwidths and they can join in anytime. UDP-based multicast and unicast sessions should be reliable even though UDP does not provide reliability. Some regions or windows may require different encoding for better performance.

The sharing server has one main thread, one manager thread, and a number of client threads. The main thread periodically checks for updated regions and prepares encoded images of these regions. These images are inserted to the image ring buffer which stores them until they are transmitted by client threads.

6.3. Serving window updates to different bandwidth users

Encoding screen updates are CPU-intensive operations, so there is no point in generating lots of updates if the clients do not have enough bandwidth. The manager thread observes each clients bandwidths and throttles the main thread according to the highest bandwidth client. This technique can be easily explained with a movie playing example. Assume that the user shares a movie with remote participants. The same region of the screen is updated 24-30 times per second. Initially, the server tries to generate as many updates per second as possible. If at least one of the clients has enough bandwidth to receive these updates, the manager thread allows the main thread to continue this pace. But if none of the clients has enough bandwidth to deal with that update rate, then the manager thread slows the main thread by forcing it to sleep between update generations. Therefore, the main thread will generate a single update by combining several updates into one. The manager thread tries to equalize the update generation rate to the fastest client's bandwidth speed. This technique prevents unnecessary CPU usage in the sharing server. The effective bandwidths of clients may change during the sharing session, this is properly handled by the manager thread because it checks clients' effective bandwidths periodically. Briefly, the main thread combines several updates into one update in order to decrease the bandwidth requirement. Similar technique is utilized by the low bandwidth client threads. The fastest one transmits the generated updates in order, however other clients may not transmit all the generated updates due to their low bandwidths. These low bandwidth clients skip some of the updates if they are obsoleted by the newer updates. Going back to video player example, the fastest may transmit 12 frames per second, whereas the others may transmit some of these generated frames permitted by their bandwidths.

6.4. Encodings

The updates are distributed as PNG images except for movies or photos. PNG is very suitable and efficient for computer-generated images. However, its lossless nature results large increase in compressed size of photographic images with negligible gain in quality, compare with JPEG and Theora which are specifically designed for photographic images. Therefore, using JPEG or Theora for photographic images and PNG for the others is the best solution. But the server does not know whether an updated region contains photographic or computer generated content; because, mirror driver runs at the frame buffer level and at this level there is only bits and no metadata. Fortunately detecting movie playing is very easy due to its specific characteristics. Different from other applications, movies generate 24-30 updates per second in a specific region of the screen. Benefitting from this characteristic, we have developed an algorithm to detect movie playing in order to use JPEG/Theora encoding for this region. Consecutive updates to a specific region trigger the detection. Detection algorithm uses the JPEG encoding on the region and compares the compressed image size between JPEG and PNG. If the JPEG size is less than a quarter of the PNG size, the server switches the default algorithm for this region to Theora and stores this result in a lookup table for subsequent updates. Theora is the default encoding for movies because it is four times more bandwidth efficient than JPEG. However, encoding Theora is costlier than JPEG especially for high resolution movies. JPEG uses four times more bandwidth but gives fifty percent more frames. User can switch the server's default encoding for movies to JPEG if all the participants have enough bandwidths. If the compression ratio of JPEG/Theora regions drops below 12:1 during the session, the server deletes the stored encoding information for this region from the lookup table. Similarly regions which are marked as PNG regions periodically rechecked. Periodic rechecking allows the server to adapt the dynamic nature of the content.

6.5. Minimizing the effect of packet loss for UDP clients

UDP is not a reliable transmission mechanism and packets can get lost. Region updates may require several kilobytes or even megabytes. Unless designed carefully, a single packet loss may destroy the region update completely. Region updates are compressed and they cannot be decompressed without complete data. In order to minimize the effect of packet loss, we have developed an algorithm which generates several small PNG images for a given update region. Blindly generating a PNG image for each scan line may increase the bandwidth usage because a new zlib compressor object should be created for each new PNG. Creating a new zlib compressor decreases the compression ratio, so the bandwidth usage increases. The developed algorithm tries to maximize the number of scan lines included in a single UDP packet while trying to keep the packet size below 1500 bytes, which the MTU for ethernet. Due to its adaptive nature, it may feed tens of lines for a text document whereas it may feed only a single line for a photographic image. We have observed an increase of approximately twenty percent in bandwidth due to small PNGs.

Transmitting self-contained UDP packets minimizes the effect of packet loss. Instead of losing the complete region update, participants may lose only a few scan lines in case of a packet loss. They may end up with imperfect frame buffer due to packet losses, but they can continue to participate to the session. The retransmission mechanism discussed in the next section helps to restore the frame buffer state.

6.6. Reliable multicast

In the previous section, we described our algorithm which minimizes the effect of packet loss. In this section, we explain another supplementary technique which retransmits missing packets [13]. When a packet loss occurs, the participant views the rest of the image except missing scanlines. The client application sends a negative acknowledgement (NACK) for this missing packet [21]. Receiving this NACK, server retransmits the requested packet. We have modified the oRTP library [7] which is used in the server side and extended our own Java RTP library on the client side to support this feature. The client/server applications do not deal with retransmission and buffering of packets, these are handled by the RTP libraries.

The retransmission mechanism has been developed considering the two issues: malicious or corrupted client behavior and NACK storms. In order to protect itself from malicious clients, the server does not retransmit a packet more than three times. If a packet can not reach to several multicast clients, they all send a NACK back to the server causing a NACK storm. Instead of sending a NACK right after detecting a packet loss; clients wait for a random amount of time (0-100 ms). If a client observes a multicast NACK from another client while waiting, it suppresses its own multicast NACK request.

6.7. Exclusive movie support

BASS is able to detect the regions of screen where a movie is playing, and it uses Theora or JPEG encoding for these regions. Encoding in real-time is computationally expensive. Although a Pentium 4 can encode 426x320 movie in full motion, it can only encode 6-10 frames for 852x480 movie. Therefore, we implemented another fea-

ture into BASS which enables to stream full motion movies to participants regardless of the resolution. The user of the BASS copies the movie file into BASS directory. BASS automatically detects the movie and displays it in the GUI. If the movie is not encoded in Theora, BASS transcodes the movie into Theora using ffmpeg2theora [2]. This preprocessing takes 30 seconds for a 20 seconds 852x480 MPEG-4 movie. After the transcoding user can stream the movie to participants using negligible CPU power.

7. Performance Results

We compared sharing systems for web browsing in terms of bandwidth usage. We also compared them for playing movies in terms of both bandwidth usage and frame rate. All sharing systems use 24-bit per pixel except RDP which uses 16-bit per pixel. If RDP used 24-bit, it would consume fifty percent more bandwidth. For all tests, we used a Pentium 4 3GHz CPU and 1GB memory as the server and a Athlon XP 2600+ CPU and 1GB memory as the client. Server and client are connected over an 100Mbps LAN. For the movie playing comparison over a low bandwidth measurement, we restricted the bandwidth of the client to 3 Mbps using NetLimiter [6]. To count the frame rate, we used a Canopus TwinPact100 scan-line converter. This box takes the RGB output of the client as input, and it outputs a digital movie stream via firewire cable. We recorded this movie stream using the iMovie application of a macbook pro. We then counted the individual frames one by one to find the actual frame rate.

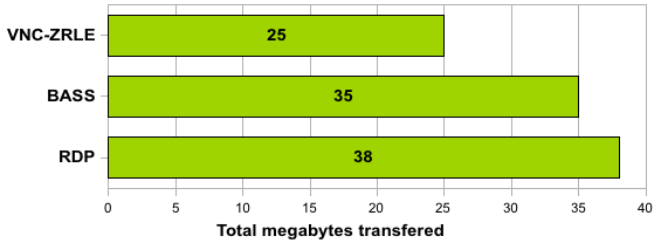

Figure 7. Web browsing performance

Figure 7 compares RDP, VNC and BASS for web browsing. During the measurement, server automatically visited the home-pages of twenty most popular webpages (according to alexa.com). We developed and used a Java-based automated testing application, available at [1], which visits each website for ten seconds. Some of these websites have animations and advertisements. Therefore, the bandwidth usage depends on how eagerly a particular sharing system transmits updates. We can say that all systems consume almost the same bandwidth which is around 1 Mbps. Although VNC and BASS use similar compression techniques, VNC consumes less bandwidth because it uses a single compressor during the session, while BASS uses a separate compressor for each update. Using a new a compressor for each update allows BASS to compress each update only once regardless of the number of participants. However, VNC has to compress the same update for each participant because each participant has a different compressor. In case of more than one participant, VNC consumes more CPU, while CPU usage of BASS remains constant.

We measured the multimedia performances of sharing systems by playing a movie over both an unlimited bandwidth link and a 3-Mbps bandwidth link. The movie is a 20 seconds soundless 852x480 24-fps MPEG-4 encoded trailer of Warren Miller's Higher Ground. The BASS server can be configured by the user to use JPEG or Theora for movies. BASS-T and BASS-J represent BASS systems which use Theora and JPEG for movies, respectively. BASS-M represents BASS's exclusive Theora streaming feature, discussed in section 6.7, instead of playing them in default media player. Figure 6 compares sharing systems over an unlimited bandwidth link. BASS-M and THINC are able to play the movie in full motion, however THINC consumes 112 Mbps, while BASS-M consumes only 1.6 Mbps. RDP gives the second high frame rate, but consumes 45 Mbps. BASS-J gives 9-fps consuming just 2-Mbps, and BASS-T gives 6-fps consuming less than 1-Mbps. VNC is the worst performer in terms of frame rate. In conclusion, BASS gives acceptable frame rate while using less than 2-Mbps.

Comparing sharing systems in unlimited bandwidth environments is not very realistic because some participants may have low bandwidths. We repeated the same experiments over a 3-Mbps link (Figure 8). Frame rates of all sharing systems dropped less than a frame per second except BASS whose frame rate remained the same. BASS-M is able to play full motion movies over an 1.6-Mbps link. In conclusion, over low bandwidth links all three BASS configurations give at least six times more frame rate than the other sharing systems.

8. Conclusion

We have developed an application and desktop sharing system which is scalable, efficient, and independent of operating system. BASS supports all applications due to its generic model, and transmits only the shared application and its child windows. We have used industry standards like RTP, BFCP, and PNG. BASS uses very little CPU thanks to the mirror driver. BASS, VNC and RDP consume roughly the same bandwidth for web browsing. However, BASS uses several times less bandwidth than the others for playing movies. Others can develop clients and servers compatible with BASS by implementing its open protocol.

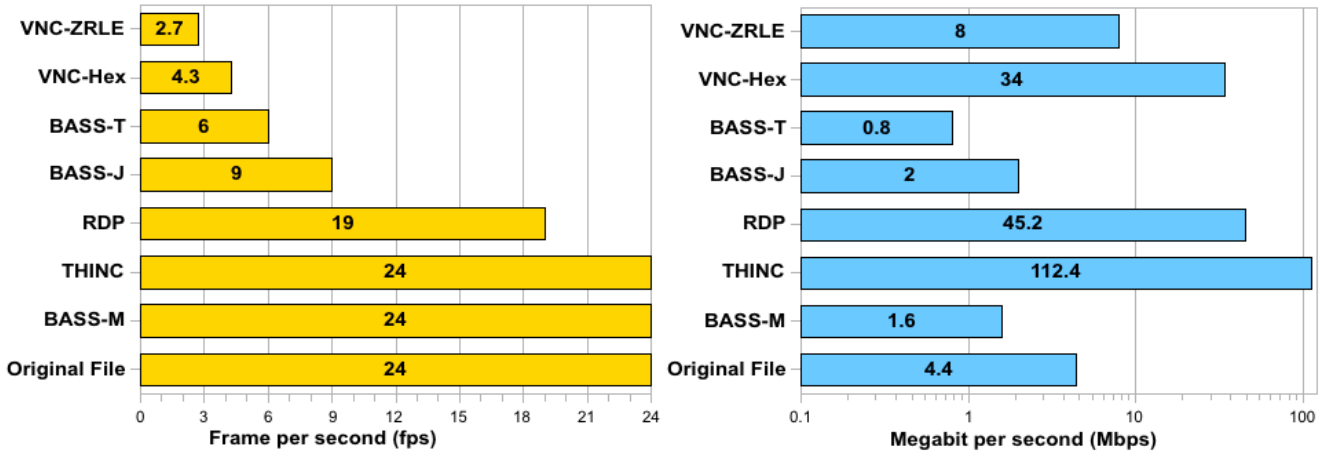

Figure 6. Comparison of sharing systems in terms of movie performance (unlimited bandwidth)

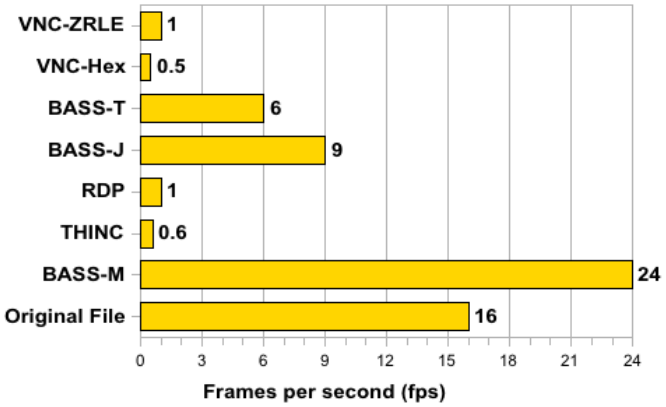

Figure 8. Movie performance (3Mbps)

References

[1] Bass. http://www.cs.columbia.edu/~boyaci/.

[2] ffmpeg2theora. http://v2v.cc/~j/ffmpeg2theora.

[3] Google docs. http://docs.google.com.

[4] Java technology. http://java.sun.com/.

[5] Netbeans. http://www.netbeans.org/.

[6] Netlimiter. http://www.netlimiter.com/.

[7] ortp. http://freshmeat.net/projects/ortp/.

[8] Png. http://www.libpng.org/pub/png/.

[9] Rdp. http://www.microsoft.com/windowsserver2003/techinfo/overview/termserv.mspx.

[10] Theora open video codec. http://www.theora.org/.

[11] Ultravnc. http://www.ultravnc.com.

[12] Joint photographic experts group. http://www.jpeg.org, 1992. ISO 10918-1.

[13] B. Adamson, C. Bormann, M. Handley, and J. Macker. Negative-acknowledgment (NACK)-Oriented Reliable Multicast (NORM) Protocol. RFC 3940, Nov. 2004.

[14] J. E. Baldeschwieler, T. Gutekunst, and B. Plattner. A survey of x protocol multiplexors. *SIGCOMM Comput. Commun. Rev.*, 23(2):16–24, 1993.

[15] G. Camarillo, J. Ott, and K. Drage. The Binary Floor Control Protocol (BFCP). RFC 4582, Nov. 2006.

[16] P. Deutsch. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951, May 1996.

[17] P. Deutsch and J.-L. Gailly. ZLIB Compressed Data Format Specification version 3.3. RFC 1950, May 1996.

[18] A. M. Lai and J. Nieh. On the performance of wide-area thin-client computing. *ACM Trans. Comput. Syst.*, 24(2):175–209, 2006.

[19] J. Lazzaro. Framing Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport. RFC 4571, July 2006.

[20] G. Lewis, S. M. Hasan, V. N. Alexandrov, and M. T. Dove. Facilitating collaboration and application sharing with mast and the access grid development infrastructures. In *E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, 2006.

[21] J. Ott, S. Wenger, N. Sato, C. Burmeister, and J. Rey. Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF). RFC 4585, July 2006.

[22] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 02(1):33–38, 1998.

[23] G. Wallace and K. Li. Virtually shared displays and user input devices. In *ATC'07: 2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, 2007.

[24] S. Xia, D. Sun, C. Sun, D. Chen, and H. Shen. Leveraging single-user applications for multi-user collaboration: the coword approach. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, 2004.

[25] P. Ziewer and H. Seidl. Transparent teleteaching. In *ASCILITE*, pages 749–758. UNITEC Institute of Technology, Auckland, New Zealand, 2002.