Application and Desktop Sharing

Omer Boyaci and Henning Schulzrinne
Department of Computer Science
Columbia University
{boyaci,hgs}@cs.columbia.edu
November 28, 2007

Abstract

Application and desktop sharing allows sharing 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 ADS which is efficient, reliable, operating system independent, scales well, supports all applications and features true application sharing. With ADS the users can share any application from their desktop and the participants do not need to have the application installed on their systems. [1]

1 Introduction

Application and desktop sharing allows two or more people to collaboratively work on a single document, drawing or project in real-time. They do not need to email the file back and forth for modifications and corrections. Similarly instructors can give a lecture over the Internet using ADS. Students or instructor can record the session and this recording can be played back locally or from a streaming server to multiple users. Application sharing can be enriched with audio and video for online lectures. Moreover ADS can be used for software tutoring where an instructor teaches a particular software like photoshop, dreamweaver, office suites, autocad and visual studio. Students can see what the instructor does and they can participate to the sharing session by remotely using mouse and keyboard. Instructors can request from students to repeat their actions after showing a feature.

[1]This research was supported by FirstHand Technologies.

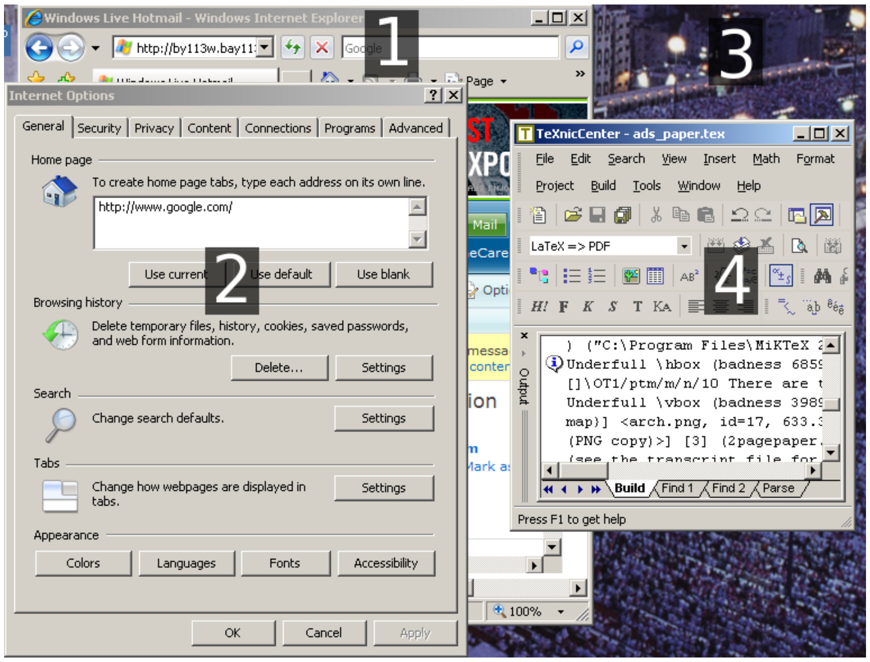
Section 2 gives some background information on different application sharing models. Section 3 discusses the related work and why they are inadequate. System architecture is discussed in Section 4, whereas the details of the ADS protocol is discusses in Section 5. Client and server architectures are explained in Sections 6 and 7 respectively. Section 8 gives information about the performance of the system in terms of bandwidth and processing power requirements. Section 9 talks about future work and finally Section 10 summarizes our work.

2 Background

Application and desktop sharing (*ADS*) allows sharing an application with remote users. All participants see the same screen-view and use the same application. There is only one copy of the shared application and it runs on the server. 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 can use different operating systems. Also, the system should scale well because some sharing scenarios such as e-learning and software tutoring may consist of several simultaneous participants. Systems supporting multicasting scale well but participants will end up with corrupted screen-views if the system is not designed 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 resources while doing this.

Application sharing differs from desktop sharing in which there is only one shared application for privacy and security. The key challenge is that some other application can sit on top of the shared application and the shared application can open new child windows like options or fonts. A *true* application sharing system should blank all the other applications if they are on top of the shared one and should transfer all the child windows of the shared application. For example, if the user wants to share only the Windows 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” windows. ADS (Figure 3) displays only these two windows with a correct size while blocking the desktop background and the non-shared application. Whereas another application sharing client Mast, which will be discussed later, could not display the shared application in correct size and also could not block the non-shared application and desktop background (Figure 2).

There are two models for application sharing: application-specific and generic. The application-specific model requires the developers to add this feature to their



To create home page tabs, type each address on its own line.

http://www.google.com/

Delete temporary files, history, cookies, saved passwords, and web form information.

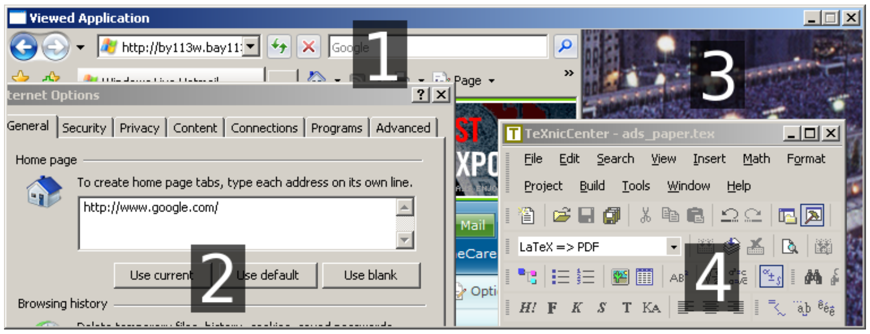
Change search defaults.

Change how webpages are displayed in tabs.

```
) ("C:\Program Files\MiKTeX 2
Underfull \hbox (badness 685
[]\OT1/ptm/m/n/10 There are t
Underfull \vbox (badness 398
map}] <arch.png, id=17, 633.
(PNG copy)>] [3] (2pagepaper.
```

Press F1 to get help

Figure 1: A screenshot of a desktop with overlapped windows



To create home page tabs, type each address on its own line.

http://www.google.com/

Figure 2: Mast client view

applications; for example, the latest version of Microsoft Office has this feature. Also, in order to participate 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 need not have the application installed on their systems. The only disadvantage of generic application sharing is that its generic nature makes it a bit inefficient as compared to the application-specific model in certain scenarios. We have developed ADS based on the generic model therefore, users can share any application without requiring the participants to have the application.

3 Related Work

As we have mentioned in the previous section, application and desktop sharing is a very useful feature therefore, operating system providers, commercial companies and several open source teams try to come up with a solution. Microsoft has Windows Meeting Space for Windows Vista and Netmeeting for Windows XP and older versions. Netmeeting was released in 1999 for Windows 98 and in our tests it fails to display pop-ups and menus. Windows Vista brings application sharing feature with the Windows Meeting Space but all the attendees must use Windows Vista. Apple introduced desktop sharing (no application sharing) with Mac OS X Leopard but again both parties should use Mac OS. These two solutions are operating system dependent. VNC [1] is a cross-platform open source desktop sharing system but it does not have application sharing feature too. UltraVNC [2] 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, Netmeeting, Windows Meeting Space and Leopard Screen Sharing all rely on unicast only so, they do not scale well. Sharing an application via unicast increases the bandwidth usage linearly with the number of participants. For example, Microsoft suggests Windows Meeting Space for a group of 10 users or less. TeleTeachingTool [6] and Multicast Application Sharing Tool (MAST) [7] 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 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 art the capturing methods like mirror drivers which will be discussed in Section 3.1. Also

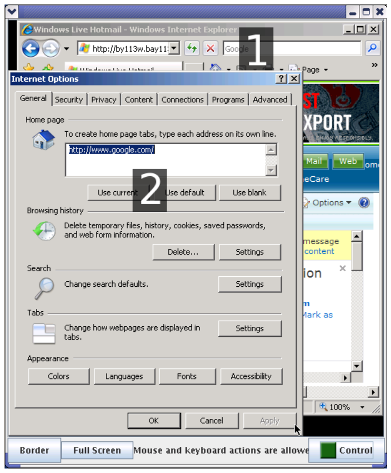


Figure 3: ADS client view

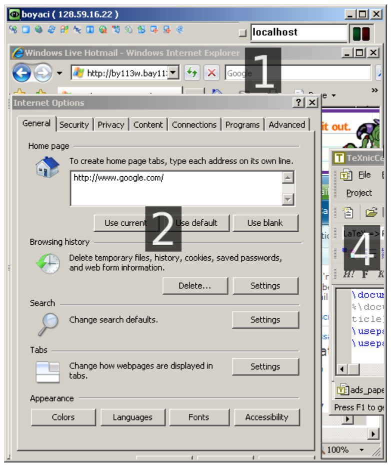


Figure 4: UltraVNC client view

Table 1: Comparison of Related Work

	OS	S	AS	RC	R
VNC	+	-	-	+	+
Windows Meeting Space	-	-	+	+	-
Leopard Screen Sharing	-	-	-	+	-
TeleTeachingTool	+	+	-	-	+
Mast	+	+	-	+	-
ADS	+	+	+	+	+

Legend

OS	Is it operating system independent?
S	Is it scalable?
AS	Does it support application sharing?
RC	Does it support remote control?
R	Does it support recording?

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 and if delivered packets can 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 1 compares the sharing systems discussed so far.

4 System Architecture

This proposed system is based on a client-server architecture (Figure 5). The server is the machine which runs the shared application. The participants use a lightweight Java 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.

We have developed a Java client which works in almost every operating system. The server could not be written with Java due to lack of Java's low level support. 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. Windows XP calls the same drawing functions on both real graphic driver and mirror driver. This mechanism allows the server to learn the updated screen regions without polling. Therefore, the overhead of application sharing is

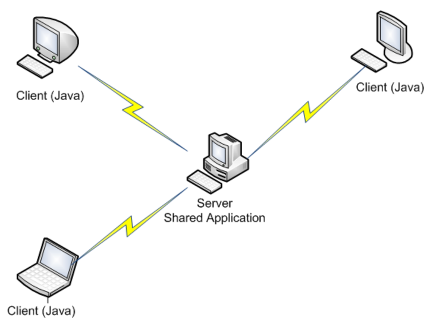


Figure 5: System Architecture of ADS

minimum on the server. Without the mirror driver sharing server should poll the screen state in order to detect the changed regions. 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 packet which consists of a RTP [8] header and a PNG [3] or JPEG [4] 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 are distributed as PNG or JPEG images according to their characteristics. PNG is an open image format which uses a lossless compression algorithm zlib and suitable for computer generated images. JPEG is lossy but suitable for photographic images likes photos or movies.

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. ADS has a NACK based retransmission mechanism for UDP clients. The UDP based clients request the missing packets from the server while storing the received out-of-order packets in a receive buffer. Multicast clients listen the NACK messages from other clients in order not to send multiple NACK requests for the same lost packet. Briefly, the server can share an application to TCP clients, UDP clients and to several multicast addresses in the same sharing session.

Late-joiners should be handled carefully otherwise they can degrade the systems performance. ADS generates a full-screen update for the late-joiners. But if more than one participant join lately within a minute, ADS generates only one full-screen update for the first one and starts all the others from this full-screen update. They will receive the full-screen update first and then all the other partial updates up to current status of the screen.

Although multiple users could receive the screen updates simultaneously, clearly only one of them can manipulate the application via keyboard and mouse events. ADS uses the Binary Floor Control Protocol (BFCP) [8] to restrict the control of the application to a single user. BFCP receives floor request and floor release messages from clients and grants the floor to the appropriate client for a period 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.

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

5 ADS Protocol

ADS Clients and servers communicates using the ADS 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 2). 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 the session. Receiving this FIR message server first transmits "open new window" messages and then the whole screen image via "region update" messages.

Each protocol message consists of a RTP header, message type identifier, window identifier and message specific payload (Table 3). Message specific payload structure and length are determined by the message type. "Close window" message does not have a message specific payload whereas "window size update" message carries new size information in this part. These messages are carried on UDP packets for UDP clients. Neither TCP nor RTP declares the length of the RTP packet therefore, RTP framing [10] is used to split RTP packets for TCP clients. Server and clients should ignore messages with an unknown message type. New message types can be easily added to the protocol.

Table 2: ADS Message Types

Server Messages
Open New Window
Close Window
Window Size Update
Region Update
Move Rectangle

Client Messages
Full Intra-frame Request (FIR)
NACK request

Table 3: ADS Message Structure

RTP Header
Message Type
Window ID
Type Specific Payload

6 Client Architecture

The ADS client is very simple compare to the server, receiving screen updates from the server and displaying them to the user. It is completely stateless in the sense that it can disconnect and reconnect to the server. Due to its simplicity, client for different platforms can be developed easily. A Java ADS client has been developed in our lab and can run on any Java supported system.

ADS clients can be passive or active. In active mode, they connect to ADS servers whereas in passive mode they wait for incoming connections. The passive mode can be used as a RGB cable replacement for presentations. Basically, the client becomes a network projector which waits for incoming connections. Presenters can connect to this client and share their applications. A computer can be dedicated as a network projector and only this machine should physically attached to a real projector. The presenters do not need to attach their laptops with an RGB cable to a projector. They just stream their application view to the network projector.

All participants of a sharing session can be view-only or they can request the control from the server. In order to request the control from server, user should press the "control" button in the GUI of the client application (Figure 3). The client will send a floor request message and then the server will reply 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 floor will be granted to the requesters one-by-one automatically. Users can release the floor whenever they want 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.

7 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 works together and communicates 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 application learns the updated regions, prepare region updates and transmit these updates to the participants(Figure 7). The sharing server also receives mouse and keyboard events from participants and executes these requests. These two components are examined in detail in the following sections.

7.1 Mirror Driver

Mirror driver is a display driver that mirrors the drawing operations of a physical display driver. It is the most efficient way to learn screen updates because it learns the exact coordinates of the screen updates automatically from the operating system. We had to develop our own mirror driver for the sharing server because there is no standard, free and open mirror driver. Remote Desktop Connection and VNC also use their own mirror drivers to efficiently learn the screen updates. Stability and correctness is very important for kernel-mode components because they can easily cause restart or blue screen of death. Our mirror driver is completely stable such that we observe no crashes at all in the 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 maps the same region of the memory to their own address spaces. There are four different regions in this shared memory (Figure 6).

- Frame Buffer
- Ring buffer for update records
- Update records current index
- Coordinates of tracking region

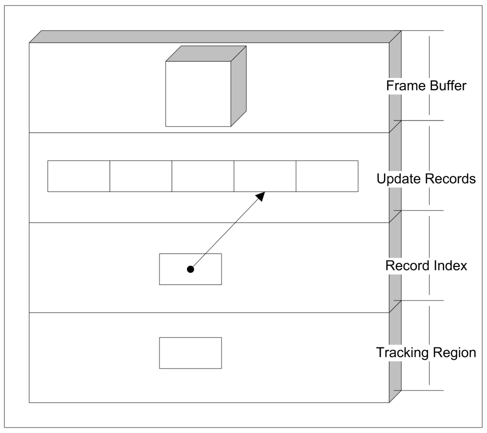


Figure 6: Shared memory regions

Windows calls the physical and mirror display drivers with the same GDI [11] (graphics device interface) commands. Display devices need a frame buffer to keep the screen state. Some display devices have dedicated frame buffers on the physical device itself. Mirror drivers are software only drivers so, they do not have dedicated hardware for graphic acceleration and frame buffer. Therefore we have used some portion the memory as a frame buffer. The memory requirement of the frame buffer is resolutions times bytes per pixel. In our implementation we have used 24 bits per pixel therefore frame buffer uses almost four megabytes for a 1280x1024 shared region.

The second region is 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.

The third region of the shared memory holds a pointer to the last inserted update. Sharing server periodically checks this pointer in order to see whether there is a new update or not.

The last region stores the tracking region and is updated by the sharing server. The sharing 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. Among these four regions the first three are maintained by the mirror driver and the tracking region is updated by the sharing server.

7.2 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 including connection establishment, update optimization and compression and transmission and keyboard and mouse events handling(Figure 7). The multi-threaded sharing server can serve multiple clients simultaneously. The server can wait for incoming connections and also it can connect to clients directly if instructed by the user. The sharing server has designed considering the following challenges:

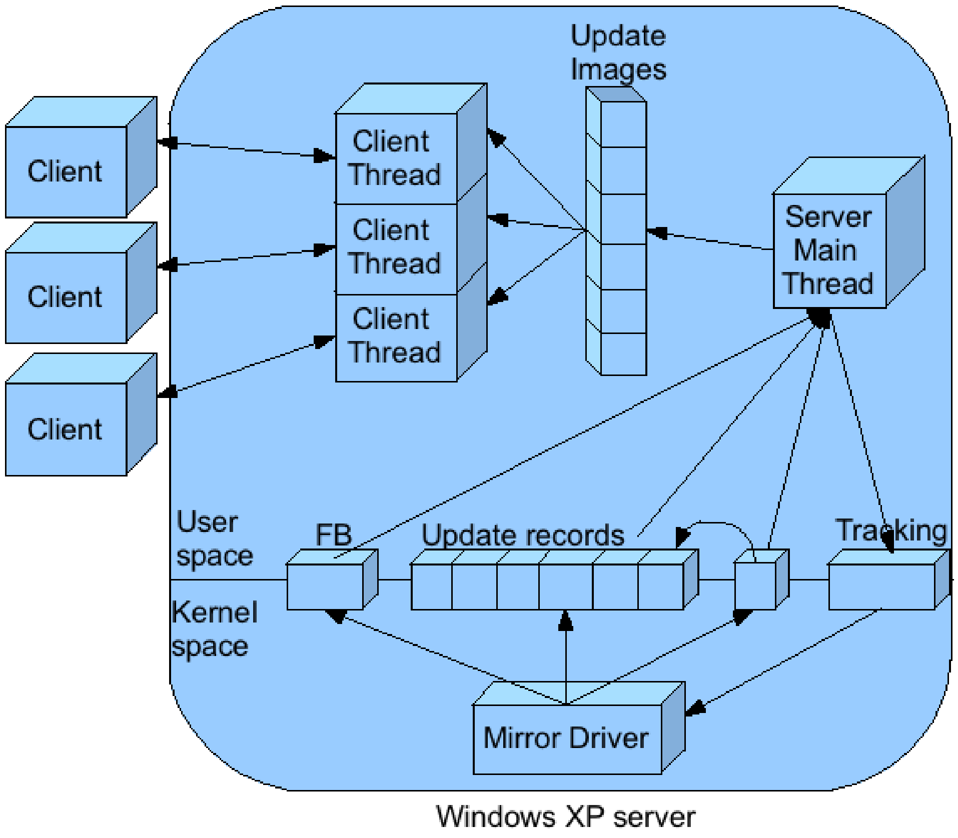


Figure 7: Windows XP server architecture

- Participants may have different bandwidths.
- Some participants may join lately.
- The effects of packet losses

- Reliable multicasting
- Some regions require different encoding

7.2.1 The main algorithm of the server

The sharing server has one main thread, one manager thread and a number of client threads. The main thread periodically checks the updated regions and prepares compressed images of these regions. These images are inserted to the image ring buffer which stores them until they are transmitted by client threads. The algorithm of the main thread is given below(Algorithm 1).

```
while Sharing session continues do
    if There are new updates then
        Find shared process windows;
        Find the overlapping windows of the unshared processes;
        Combine the overlapping or repeating updates into one update;
        Clip the regions considering overlapping unshared windows;
        Prepare a compressed image for each update region;
        sleep x ms (determined by manager thread) before starting again;
    end
    else
        sleep 5 ms and then check again
    end
end
```

Algorithm 1: The algorithm of the main thread

7.2.2 Serving to different bandwidth users

The most CPU intensive part of the server is the image compression which may take up to 500 ms depending on the size of the updated region. Therefore, the main thread tries to optimize the updated regions by combining overlapping regions and ignoring repeated regions. 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 the remote participants. The same region of the screen is updated 24-30 times per second depending on the movie type. Initially the server tries to generate as many updates as possible. If at least one of the clients has enough bandwidth to receive these updates, manager thread allows the main thread to continue with this pace. But if none of the clients can have enough bandwidth to receive these many updates then manager thread slows the main thread by forcing it to sleep between update image 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 and this is properly handled by the manager thread. 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 use an algorithm which skip some of the updates if they are obsoleted by the newer updates. Returning 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.

7.2.3 Encodings

The updates are distributed as PNG images except for movies. PNG is very suitable and efficient for computer generated images. However, it's lossless nature results large increase in compressed image size for photographic images with negligible gain in quality compare to JPEG which is specifically designed for photographic image data. Therefore, using JPEG 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 special characteristic. 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 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 JPEG size is less than quarter of the PNG size, the server switches the default algorithm for this region to JPEG and stores this result in a lookup table for subsequent updates. If the compression ratio of JPEG regions drops below 12:1, 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. For example, we have played a movie which starts with a black screen and then displays some texts on this background and then continues with the movie itself. In this specific case, the server first used PNG and then automatically switched to JPEG after the actual movie starts.

7.2.4 Late joiners

Participants of a sharing session can join anytime and they need the full screen buffer before receiving partial updates. Therefore, the sharing server prepares and transmits the image of the whole shared region for late joiners [12]. Preparing the image of the whole shared session is costly in terms of CPU so, the sharing server stores the generated image for some time. When a participant joins, the server sends the stored image if available and then transmits all the subsequent partial updates. The stored image obsoleted after some period. This also protects the server from malicious or misbehaving clients.

7.2.5 Minimizing the effect of packet loss for UDP clients

UDP is not a reliable transmission mechanism therefore, packets can get lost. Region updates may require several kilobytes or even megabytes and unless designed carefully a single packet loss may destroy the region update completely because 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. And 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 screen update while trying to keep the generated image size below 1500 bytes which the maximum 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 approximately twenty percent increase in bandwidth due to small PNGs.

Transmitting self-contained UDP packets minimize 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 may continue to participate to the session. Figure 8 demonstrates the effect of a single packet loss where the image is displayed with a few missing scan lines. The retransmission mechanism which will be discussed in the next session helps to restore the frame buffer state.

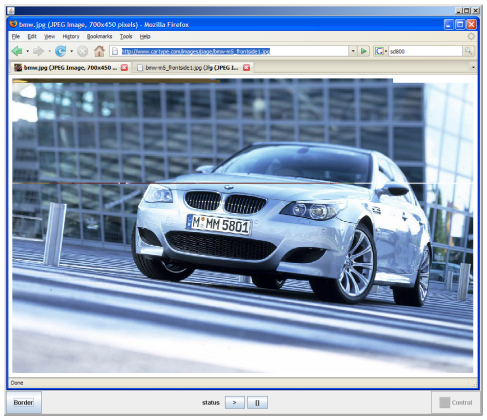

bmw.jpg (JPEG Image, 700x450 pixels) - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Done
Border
status
Control

Figure 8: Minimizing the effect of packet loss

7.2.6 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 retransmit missing packets. When a packet loss occurs, the participant views the rest of the image except missing scan lines(Figure 8). The client application sends a negative acknowledgement (NACK) for this missing packet [12]. Receiving this NACK, server retransmits the requested packet. We have modified the oRTP library which is used in the server side and extended our own Java RTP library in client side to support this feature. The server and the clients do not deal with retransmission and buffering of packets, these are handled by the RTP libraries.

The retransmission mechanism has 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. NACK storms happen when a packet could not reach several multicast clients and they all send NACK requests to server. To solve this problem, clients do not send NACK requests immediately after detection instead they wait for a random amount of time (between 0-100ms). During the waiting period if the client observes a NACK request from another client in the multicast session, it suppresses its NACK request. It is possible that more than one NACK have transmitted due to very close timeouts but this is better than having hundreds of NACKs on the network.

7.2.7 Better movie support (Experimental)

As mentioned before we have developed an algorithm to detect movie playing. Server starts using JPEG encoding for the detected region which increases frame per second and also decreases bandwidth usage dramatically without any distinguishable loss by eye from the original. This technique can be improved by bypassing the rendering to frame buffer.

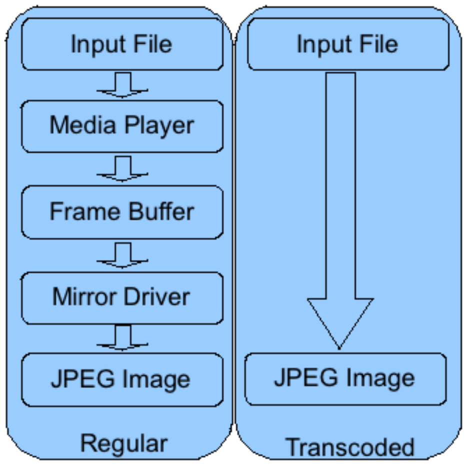


Figure 9: Better movie support

Figure 9 demonstrates the necessary steps for each of these techniques. In the regular approach user plays the movie in the server with any movie player and the frames are captured from frame buffer and compressed into JPEG images. In the transcoded case, the user selects the movie he want to share with participants and the frames are directly read from input file and transcoded on-the-fly into JPEG images in real-time. These JPEG images are transmitted into participants according to their bandwidths such that low bandwidth users get less fps compare to high bandwidth users. For on-the-fly transcoding we have used FFMPEG-Java which is a sub-project of Freedom for Media in Java, FMJ in short. FFMPEG-Java is a Java wrapper around FFMPEG, using JNA. The architecture of the implemented system is given in Figure 10. The results with this technique is better than the regular one due to low overhead on the system. The detailed performance results can be found in the following section.

8 Performance Results

TBD later...

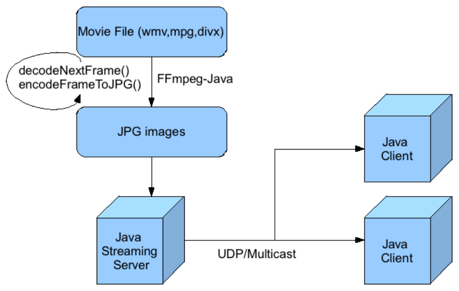


Figure 10: Transcoding a movie into JPEGs

9 Future Work

TBD later...

10 Conclusion

We have developed an operating system independent, scalable and efficient application and desktop sharing platform. ADS 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. The CPU usage of ADS is very low thanks to the mirror driver. The bandwidth usage of ADS is similar to other solutions like VNC. ADS allows collaborative working on a single document or project. It can also be used for e-learning and software tutoring. People can develop clients and servers compatible with ADS by implementing its open protocol.

References

[1] Tristan Richardson , Quentin Stafford-Fraser , Kenneth R. Wood , Andy Hopper, Virtual Network Computing, IEEE Internet Computing, v.2 n.1, p.33-38, January 1998

[2] UltraVNC Win32Server 0.5.2 Release Build Jun 18 2006 14:56:09

[3] Portable Network Graphics, http://www.libpng.org/pub/png/

[4] Joint Photographic Experts Group, http://www.jpeg.org, ISO 10918-1, 1992

[5] Java Technology, http://java.sun.com/

[6] Peter Ziewer and Helmut Seidl. Transparent TeleTeaching. In Proceedings of ASCILITE 2002, Auckland, NZ, Dec. 2002.

[7] Mehmood Hasan, Gareth Lewis, Vassil Alexandrov, Dove Martin and Tucker Matt. Multicast Application Sharing Tool for the Access Grid Toolkit. In Proceedings of the UK e-Science All Hands Meeting, Nottingham, UK, 2005.

[8] G. Camarillo, J. Ott, K. Drage. The Binary Floor Control Protocol (BFCP). RFC 4582, IETF, November 2006.

[9] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, IETF, July 2003.

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

[11] Microsoft Windows graphics device interface, http://msdn2.microsoft.com/EN-US/library/ms536795.aspx

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