Application and Desktop Sharing

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

Abstract

Application and desktop sharing allows sharing any application with numerous 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 has true application sharing.

1 Introduction

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 scales 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 is different than 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 other applications if they are on top of the shared one and should transfer all the child windows of the shared application.

There are two models for application sharing: application-specific and generic. The application-specific model requires the developers to add this feature to their applications; for example, the latest version of Microsoft Office has this feature. Also, in order to have 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.

Windows Vista brings application sharing with the Windows Meeting Space but all the attendees should use Windows Vista. Similarly, Mac OS X Leopard will bring desktop sharing but again both parties should use Mac OS. These two solutions are operating system dependent and they do not scale well due to their unicast nature. Sharing an application via unicast increases the bandwidth usage linearly. For example, Microsoft suggests Windows Meeting Space for a group of 10 users or less. TeleTeachingTool [1] and Multicast Application Sharing Tool(MAST) [2] use multicasting in order to built a scalable sharing system. TeleTeachingTool adds multicast support to VNC servers however, it is developed just for online teaching so it does not allow participants to use the shared desktop. Also, it does not support real 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 is discussed in the next section. Although both TeleTeachingTool and MAST systems use multicasting for scalability, they do not provide a real solution to 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 the packet losses the TeleTeachingTool and MAST periodically transmit the whole screen which increases the bandwidth and CPU usage.

We are proposing a mirror driver based efficient, OS independent, scalable solution which has true generic application sharing. Not only we come up with a new protocol but also we have developed a running system. We will discuss the architecture of the system in next section.

2 Architecture

This proposed system is based on a client-server architecture (Figure 1). The server is the machine which runs the shared application. The clients use a very simple and small Java application for connecting to server and they do not need the shared application. They receive screen updates from the server and send keyboard and mouse events to the server.

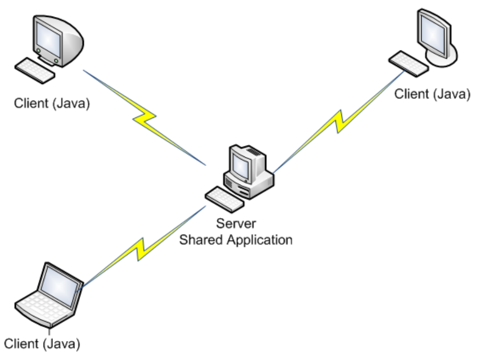

Figure 1: System Architecture of ADS

We have developed a client which has been written in Java and naturally 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. 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 minimal 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 the kernel mode and notifies the user mode server when it detects some changes in the GUI of the shared application. Server then prepares a packet which consists of a RTP [3] header and a PNG 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. PNG is an open standard for images and it uses a lossless compression algorithm zlib which is necessary to have a clear and undistorted image of the shared application on the clients. Our initial measurements shows that the bandwidth usage of the ADS is comparable to the current sharing systems like VNC. We will conduct more measurements to obtain more detailed comparison results.

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. The bandwidths of TCP clients can be different so we have developed an algorithm which sends the updates at the link speed of each client. UDP does not provide flow control therefore, server transmits for a particular bandwidth. Several simultaneous multicast sessions with different transmission speeds 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. We will do some measurements to compute the estimated overhead of NACK protocol. 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) [3] 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 added 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.

3 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] P. Ziewer et al., Transparent TeleTeaching. ASCILITE 2002, Auckland, NZ, December 2002.

[2] M. Hasan et al., Multicast Application Sharing Tool for the Access Grid Toolkit. UK e-Science All Hands Meeting, Nottingham, UK, 2005.

[3] G. Camarillo et al., The Binary Floor Control Protocol (BFCP). RFC 4582, IETF, November 2006.

[4] H. Schulzrinne et al., RTP: A Transport Protocol for Real-Time Applications. RFC 3550, IETF, July 2003.