DESKTOP SHARING IN VIRTUAL WORLDS
A THESIS
SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FUFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE
MASTER OF SCIENCE
BY
BENJAMIN M. JOHNSTON

ADVISED BY
DOLORES ZAGE

BALL STATE UNIVERSITY
MUNCIE, INDIANA
MAY 2010
## Contents

Chapter 1 – Introduction ........................................................................................................ 4  
   Abstract .......................................................................................................................... 4  
   The Problem .................................................................................................................. 4  

Chapter 2 - Background ....................................................................................................... 5  
   Literature Survey .......................................................................................................... 5  
   Project Lifewerx ........................................................................................................... 8  

Chapter 3 - Current Work .................................................................................................. 12  
   Project Lifeshare, version 1 ......................................................................................... 12  
   Project Lifeshare, version 2 ......................................................................................... 15  
   A Detailed Look at the Current Lifeshare ................................................................... 21  
   The Menu Options ...................................................................................................... 22  
   SAS Hooks .................................................................................................................... 25  
   VNC Server Hooks ....................................................................................................... 26  
   SSH Tunnel .................................................................................................................. 27  
   Lifeshare Stress Test .................................................................................................... 28  

Chapter 4 – Conclusions and Future Work ...................................................................... 31  
   Conclusions .................................................................................................................. 31  
   Future Work .................................................................................................................. 32  

Acknowledgements .......................................................................................................... 34  

Works Cited ...................................................................................................................... 35  

Appendix A : SecurityManager.java .............................................................................. 37  
Appendix B: SecurityCellGLO.java ............................................................................... 44  
Appendix C: LifeShareMenuPlugin.java ......................................................................... 47  
Appendix D: AppLauncher.java ....................................................................................... 52  
Appendix E: LifeShare.java ............................................................................................. 53  
Appendix F: SSHPortForward.java .................................................................................. 57  
Appendix G: Stress Test Data .......................................................................................... 60  

Figure 1 - Excerpt from SecurityManager.java ............................................................... 10  
Figure 2 - Lifeshare Architecture, Version 1 .................................................................. 13  
Figure 3 - Lifeshare Architecture, Version 2 .................................................................. 18
Figure 4 - An Avatar looking at a shared desktop................................................................. 19
Figure 5 - An Avatar looking at a shared desktop including a Wonderland session ....... 20
Figure 6 - The desktop being shared in Figure 4................................................................. 21
Figure 7 - Excerpt from LifeShareMenuPlugin.java .......................................................... 25
Figure 8 - Excerpt from AppLauncher.java ................................................................. 25
Figure 9 - A selection from LifeShare.java ................................................................. 27
Figure 10 - A selection from SSHPortForward.java ......................................................... 28
Figure 11 – Chart detailing average frame rate............................................................. 29
Chapter 1 – Introduction

Abstract

This thesis details the integration of a common collaboration technique, desktop sharing, into a virtual world environment. Previous literature shows no intersection between these fields. This thesis will illustrate that existing collaboration technology can be integrated into virtual worlds with a minimal amount of effort. Outlined in this thesis are the developmental and procedural challenges encountered in demonstrating seamless desktop sharing in a virtual environment and a stress test of the integrated system revealing that the inclusion of desktop sharing resulted in minor performance loss.

The Problem

Current virtual world technology has limited capability for collaboration because of a lack of collaboration tools. This thesis proposes that it is feasible to take a common collaboration tool such as desktop sharing and introduce it into a virtual world.
Chapter 2 - Background

Literature Survey

There are two main areas of research that influence this thesis: desktop application sharing and virtual worlds. Desktop sharing in virtual worlds fundamentally represents the intersection of two unrelated technologies, desktop / application sharing and virtual worlds.

Desktop Sharing

Desktop sharing commonly refers to a remote frame buffer technology. Desktop sharing allows a user to send screen data to be drawn elsewhere and receive input remotely. Its applications vary from remote system administration to accessing virtual machines. There has been much research concerning the use of desktop sharing as a platform for collaboration. A few useful features appear in several papers. The BASS Application Sharing System established the idea of applying a secondary protocol to re-encode video and stream it separately from the frame buffer for any video playing on the screen [1]. Additionally the sharing system supports per-application sharing by removing all non-application specific information from the remote frame buffer. In one system researchers enhanced the Virtual Network Computing (VNC) protocol by adding an additional layer of authentication to allow for view-only or normal interactivity.
connections [2]. Systems that support multicast (multiple people only seeing one screen) tend to use the Binary Floor Control Protocol to determine controllability of the screen at any one point in time [1] [3] [4].

There are also other papers of interest that cover non-desktop sharing collaboration. For example, research on remote pair programming, where two users work on the same code at the same time using shared cursors and synchronized codebases, differs from desktop sharing because both users are still seeing different desktops [5] [6]. Instead of actually visualizing the other collaborator’s desktop, a user of the Sangam tool has a synchronized view and cursor with the other collaborators. This approach works well in very specialized environments such as programming Integrated Development Environments (IDEs) but lacks usability in more general scenarios. Unfortunately no hard research has been done on the efficacy of the technique but it is important to remember that desktop sharing is just one facet of collaboration technology.

Virtual Worlds

Virtual worlds, as related to collaboration technology, have a considerable research base. This thesis will focus on virtual world research specific to corporate environments. Even before 3D virtual environments became more commonplace, research was being conducted on virtual offices. Text and static pictures of human heads led the way toward what could eventually become today’s 3d avatar-based worlds [7]. Another interesting early development was the various techniques used to develop early avatars such as facial mapping or normal texturing [8].
There exists a wide range of research in the area of virtual world design and implementation factoring in many branches of computer science. These range from heavy Human-Computer Interaction (HCI) theory-based interaction models for virtual world design [9] to software engineering theory based design patterns [10]. Other research was done regarding the creation of a team model for virtual environment based collaboration [11]. Kahai et al.’s paper unfortunately lacks any hard data. However, it outlines a solid set of requirements for any future work to test the interaction model. There is also research on (Quality of Service) QoS methodologies for virtual worlds [12]. The QoS research tends to focus on altering quality of streamed videos in response to how “focused” an avatar’s view is on it. The paper by Maybury actually focused on non-virtual world-based virtual collaboration (using shared browsers and webcam feeds). This study contained actual usage data of such a system and provides a list of critical success factors for a virtual collaboration tool. Some of these critical factors are more focused on organizational or process concerns such as possessing a shared outcome between teammates, having a solid leader for the group, organizing physical meetings in addition to virtual ones and having an environment that promotes collaboration. Other main factors are directly relevant to the project at hand, namely infrastructure scaling, tool usability and a secure infrastructure [13].

This thesis’s research is framed in Project Wonderland, a Sun Labs research project. A virtual world project that uses Project Wonderland is WonderDAC, a security system for Wonderland. WonderDAC implemented Linux-style file permissions to add a layer of
access control to the virtual environment. This project was influential in some of the early developments in Project Lifewerx, as will be explained in section 4 [14].

**Project Lifewerx**

Project Lifewerx was a research project administered by the Software Engineering Research Center (SERC) for Rockwell Collins. The research was carried out at Ball State University. The main focus of the project was to develop a standalone secure virtual world collaboration solution. Other candidate virtual worlds existed. However, many of them had a fatal obstruction. They all were sold or offered as a service, not a product. For example, Second Life, a popular virtual world for social use, did not offer an Enterprise (internally runnable) edition until late 2009 [15]. While it would have been possible prior to that to use Second Life as a collaboration tool, all collaboration data would be stored on Linden Lab’s servers. As a defense contractor, Rockwell Collins has many projects where controlling their data is very important, so such a solution was infeasible given the requirements of the company. Given those parameters, our research team decided to use Project Wonderland, a Sun Labs open source Java-based research project. At the time Wonderland was developed primarily as a proof of concept and despite being open source suffered from poor extensibility and lack of documentation.

An example of the difficulty in extending the old version of Wonderland was the suggestion to create a “virtual poster hall”. Producing a set of static images displayed in-world ended up being a very complex process. It started with looking at the
demonstration world that Sun had provided and identifying poster-like images. These in-world posters were traced back to the original image files and eventually to XML files referencing these images. From the XML files, a fully qualified class name of the object in-world was obtained. Other requirements of images, such as how the in-world renderer was limited to a certain vertical aspect ratio, were also discovered through reverse engineering. After cropping the images to the correct dimensions, new XML files were manually created to define the poster’s in-world locations and rotations. Due to a lack of in-world notification of coordinates, a few known positions were extrapolated to find a wall. From the wall’s coordinates the posters were plotted out on graph paper to their final world positions. Several attempts and adjustments (mostly to the rotation) were necessary to render the poster hall. This process was made more cumbersome since every world reload involved recompiling the server from scratch and waiting for it to come back up.

Despite these hurdles some interesting features were developed, including a graphical avatar configurator and an access control feature. The access control feature was based on requirements from Rockwell Collins. Specifically, such a system could limit access to various data based on access levels. A large portion of development time was spent on searching for a location to insert the new code into the renderer. During this search, an undocumented SecurityManager class was found and extended. The main functional method in SecurityManager, allowedBy(String username, CellGLO cell), went from being a one line “return true” to the following block of code:
public boolean allowedBy(String userName, CellGLO cell) {
    if (cell instanceof AvatarCellGLO) {
        ((TransparencyManager)(AppContext.getDataManager().getBinding("TRANSPARENCY_MANAGER", TransparencyManager.class))).checkCell((AvatarCellGLO)cell);
        if (((AvatarCellGLO) cell).getUserName().equals(userName)) {
            // i can always see myself
            return true;
        } else {
            return checkAvatarIntersection((AvatarCellGLO) cell, userName);
        }
    } else if (cell.getGroup() == null) {
        // make boolean for if cell has been checked after everything is initialized, so this will only check once for each cell // except for avatar cells, so it can update what group it is in
        checkGroups(cell);
    }
    for (SecurityGroup sg : groupList) {
        if (sg.groupName.equals(cell.getGroup())) {
            if (cell.isWhitelist()) {
                return sg.userIsInGroup(userName);
            } else {
                return !sg.userIsInGroup(userName);
            }
        }
    }
    return (true);
}

Figure 1 - Excerpt from SecurityManager.java

Figure 1 evaluates the cell being passed in to determine if the user has permission to view it. The code has a default case of always allowing a user to view themselves but for other avatars the code requests a helper method. After checking for avatar scenarios, the security manager initializes the cell’s security group setting in a “lazy initialization” manner (the code checks to see if a cell has a null group, and only sets the group if necessary). After the check to make sure the cell has a non-null group the code simply determines whether or not the user is in any appropriate groups and bars or allows him accordingly. The full SecurityManager.java is listed in Appendix A. These security
groups were set by in-world objects called SecurityCells. SecurityCells had to be configured externally. However, once in place, users could turn spatially defined areas of the world on or off in real-time. The server-side code for the SecurityCell is available in Appendix B. Several other classes went into making the security manager work however they were mostly boilerplate code. There was a competing feature at Notre Dame named Wonderland Directory Access Control, or WonderDAC for short [14]. WonderDAC offered better granularity of control, allowing for “read-only”, “write” and “execute” permission levels. While appearing more granular, WonderDAC was structured around the In-World-Object to XML backend relationship. Our feature focused the controls more on the geography of the virtual world itself to define boundaries of the secure area. Our tool, however, only had two levels of access control compared to WonderDAC’s three.

These excursions only helped frame what we realized would be a “killer app” feature for this virtual world: desktop sharing. This led to the formation of the subproject dedicated to desktop sharing in a virtual world, Project Lifeshare.
Chapter 3 - Current Work

Project Lifeshare, version 1

Project Lifeshare was originally started to add desktop sharing to the Wonderland environment. It had a few major requirements. It had to be extensible, it had to be secure and it had to multicast desktops. To meet these requirements, a standalone desktop sharing system was developed. It featured a Remote Desktop Protocol (RDP) based viewing client and relied on a VNC server to grab frame buffer data. The VNC server was connected to XRDP, a Linux RDP bridge program. A Windows port was created and the researchers used its VNC->RDP bridge functionality to allow multiple RDP clients to connect into a single VNC session. This bridge solved the problem of multicasting. The entire process was managed by an external client / server relationship that was designed to plug into a variety of other programs and authentication schemes. The manager was designed from the ground up to be customized for the environment (both server and client side) that the user desired. The security was handled by the native encryption in the RDP connections and also by encrypting the manager connections. Lifeshare was a complex solution to what is considered a complex problem.
Unfortunately the complexity of Lifeshare led to a multitude of issues. Integration with Wonderland also proved difficult. Lifeshare was designed more as a standalone solution and accordingly it maintained its own set of permissions, its own user database and was rather monolithic. The question was raised of whether or not it was really desktop sharing in virtual worlds or a desktop sharing application sitting next to a virtual world application. Additionally, since the researchers were the only maintainers of the XRDP port to Windows, it required maintenance between versions. The project felt over-engineered and started to suffer under its own weight.

Figure 2 - Lifeshare Architecture, Version 1

Figure 2 shows the entities and connections of Lifeshare version 1. A desktop sharing connection would be initiated as follows
• A user identifies a friend as the person he wants to share his desktop to by clicking that person's name and clicking the “Share Desktop” button in the Lifeshare GUI.

• The LifeShare server, over the SSL connection sends the message to the receiving friend.

• The friend gets dialog box asking if he wants to receive the desktop. He hits okay.

• The okay message gets relayed to the server. The server tells both sides a randomly generated one-time password.

• The sending client starts a VNC server which only accepts connections on the local loopback.

• The sending client also starts an XRDP session which listens for incoming connections set only to authenticate to the generated password. Additionally if more than a few seconds elapse or it receives an incorrect password XRDP will reject the password.

• The receiving client has a version of the remote desktop protocol client open set to connect to the sending client's IP address and authenticate given the one-time password.

• The remote desktop program from the receiving client connects to the XRDP on the sharing client. XRDP then bridges the connection into a local loopback connection to the VNC.

• The users share their desktop. If the sharing user decides to end the session XRDP and VNC re shut down, disconnecting any currently connected users in the process.
Project Lifeshare, version 2

After a summer reprieve from the project, the project found its second wind, funding from the Ball State University Emerging Media Initiative. Additionally, a new version of Wonderland had been released. This version exhibited a multitude of improvements over the previous edition. The task of putting an image in world required only a drag and drop compared with the previous multi-day arrangement (see section 4). With all the improvements to the base system, a reworking of Lifeshare was not only possible, it was encouraged. The new approach would attempt to better leverage the existing in-world application sharing technology to reduce complexity.

Working with the new version of Wonderland also allowed for many process improvements. Changes that required a complete recompile now only required a short restart. Previously deploying the client to the user was an often complex process. The new version not only built its own client but also distributed it via the Java Net Launch Protocol (JNLP) over an embedded Glassfish web-server. From a programming standpoint, there were a number of improvements. A new architecture allowed for “modules” to be installed without a recompile. This improvement meant that any final versions of Lifeshare could be distributed as a single JAR file to be installed on the server without any programming or build knowledge on the part of the server administrator.

Writing the add-ons also became multitudes easier. What previously involved multiple days of reverse engineering looking for a hook now had a clearly documented (with nice online tutorials) route. Updates that required extended code hunts quickly became wiki
searches. If the previous version of Wonderland had been a proof of concept for virtual worlds, then the new version quickly became a proof of concept for extendable virtual worlds.

The new design of Project Lifeshare concentrated on simplicity. In lieu of the complex xrdp-port, the existing Sun Application Server (SAS) would be used. Due to firewall issues, previous tests had erroneously labeled the SAS as a source of major latency issues. Further testing with the new version showed those assumptions were premature. With the ability to use the SAS for the multiplexing all that remained was frame capture, integration and security.

The issue of frame capture was resolved through a similar route. As in Lifeshare’s previous version a customized version of a VNC server was used. The VNC server was, using the Java Natural Interface (JNI), linked to the Java module to allow for Java-based code to call C code. The customizations to the VNC server entailed creating hooks for this JNI code and resolving threading issues. In the end, all the VNC server needed to do was to serve the desktop and connect to a client. VNC is somewhat unique in that its protocol allows for both client-to-server connections and server-to-client connections. By sending the desktop from a “push” style server, many reverse tunneling issues (several of which the previous version of Lifeshare would have engaged) were cleverly avoided.
Integration became one of the easier tasks. Unfortunately the Wonderland module required some customization. However, Sun had foreseen the requirement to perform runtime-customized command calls to the X11 SAS. A single environment variable must be set on the server to enable runtime-customized commands. This is the only server customization that Lifeshare requires. The Lifeshare module, in the present version of Wonderland, is very easy to launch. Desktop sharing can be initiated by a single menu option which starts the VNC server and instructs the X11 SAS to open a listening VNC client. The server’s VNC client is setup to “listen” as opposed to initiating a connection enabling the user’s VNC server to connect directly to the server’s VNC client. From the Wonderland API’s perspective, the module is a single Wonderland plug-in. In Wonderland terms, a plug-in is a type of module that reacts to startup and shutdown hooks. From the plug-in API, the Lifeshare module attaches a menu option with a listener to start the Lifeshare process.

Security has been another major concern of Lifeshare. Given the project’s roots in the defense industry and the likelihood of Lifeshare being used for sensitive projects, encryption is paramount. VNC, an older standard, has little to offer in the way of innate security. The team has been working on resolving the insecurity by wrapping the VNC connection in a SSL proxy. This proxy will allow for a secure connection between the client and server and will minimize the risk of information leakage. It also adds complexity to the system as SSL relies on various security libraries which must be tweaked into the Lifeshare process and have certificate issues resolved.
Figure 3 illustrates the entities and relationships of a Lifeshare version 2 connection. Assuming the Sending Client and Other Client are already connected to the Darkstar and X11 SAS the process works as follows:

- The user of the Sending Client decides to share his desktop by selecting the menu option. The “Shutdown Lifeshare” menu option becomes available.

- A VNC server that only accepts local loopback connections is started on the sharing user’s machine.

- The user is prompted for username and password for the server. This information is used to authenticate the user and open a SSH tunnel (labeled “Proxy” in Figure 3) to the machine running the X11 SAS. The SSH tunnel connects port 5555 on localhost to a randomly determined port on the X11 SAS machine between 5500 and 5600.

- Darkstar is told to create a SAS based cell with a customized command. The customized command is to start a VNC server listening on the port chosen previously for the SSH tunnel.

- The SAS takes the cell creation and starts a listening VNC client given the instruction from Darkstar.
• After a predetermined delay to allow for the SAS to finish its cell creation, the VNC server on the sharing user's computer is told to connect to 127.0.0.1:5555 (the origin point of the SSH tunnel created before). This VNC connection is routed through the tunnel to the listening vncviewer on the X11 SAS machine. The SAS handles the rest of the viewing.

• If the user on the Sending Client shuts down their share (through the “Shutdown Lifeshare” menu option) the VNC server is told to disconnect all clients and the SSH tunnel is closed. When the vncviewer is disconnected the SAS removes the cell from the world.

Figures 4, 5 and 6 are a series of screenshots of Lifeshare version 0.1.0. Figure 3 shows Ben2’s avatar looking at a shared desktop in world. From this view, the user could easily take control and manipulate the other desktop.

Figure 4 - An Avatar looking at a shared desktop
Figures 5 and 6 are of the shared desktop in-world and a screenshot of that desktop on the originating client.

Figure 5 - An Avatar looking at a shared desktop including a Wonderland session
The current version of Lifeshare acts as a Wonderland plugin. Plugins follow a specific plugin API from Wonderland that gives them a series of hooks to the connection lifecycle. This lifecycle is detailed in the documentation as such [16]:

- **Initialize**: A client first establishes a connection with a Wonderland server. During this phase, all of the needed client modules are downloaded from the server, if not already cached.

- **Activate**: An existing Wonderland connection becomes primary.

- **Deactivate**: An existing primary Wonderland connection is no longer the primary connection.
• **Cleanup**: A client tears down an established connection with a Wonderland server.

The steps are all necessary in the Lifeshare module. The initialize and cleanup steps follow their namesakes by simply creating the necessary objects and cleaning them up afterwards. The activate and deactivate methods are used to insert and remove the GUI hooks as necessary. The interesting step of Lifeshare happens in a step that isn’t directly part of the lifecycle with the primarySession hook. That hook allows you to start code when a connection becomes “primary”, most commonly associated with when a connection has finished authenticating. Users wouldn’t want Lifeshare to be possible outside of authenticated connections. Tying the functional code to the primarySession hook allows for the assurance of authentication. The bulk of the code in these methods is tied into action listeners for the menu items. The researchers used anonymous inner classes for ease of use. There are three menu items that make up Lifeshare: Shutdown Lifeshare, Start Unsecure Lifeshare and Start Secure Lifeshare.

**The Menu Options**

The first menu option, Shutdown Lifeshare, shuts down the share. Finding a clean way to sever the connections has been a complex process. Originally a secondary component approach was reviewed where, after creating the cell (in-world object) on the server, a ShutdownHookComponent would be attached to it. This component would serve as a beacon to let a later code hook into the world system and find these cells to remove them on logout. The shutdown hook component was further
complicated by the issue of finding the tree that stored the cells. Without access to this
tree, getting references to all the objects in-world is nearly impossible. Ultimately, the
shutdown hook component was scrapped and replaced with a cleaner solution. As it
turns out if the VNC server cleanly disconnects the VNC client before shutting down, it
will properly remove the vncviewer from the world.

The Unsecure Lifeshare option was originally included for testing but has proven to be a
useful tool for testing connection issues. The Unsecure Lifeshare menu option simply
starts a push VNC server on the client, a listening vncviewer (a VNC viewing client
configured to listen for incoming connections) on the server, and tells the push server to
connect to the viewer. The next sections will detail the hooks allowing for the
vncviewer and the VNC server in more detail.

The Secure Lifeshare option acts similarly to the Unsecure Lifeshare option but with a
few alterations. As the name implies, the connection is in a secure method and,
therefore, this menu option starts a SSH Tunnel. Additionally, due to the additional
resources necessary for the SSH tunnel, the menu option disables itself (reenabling
when the “Shutdown Lifeshare” option is clicked) to prevent users from attempting
something that would require too many resources. After opening the SSH tunnel,
similar to the Unsecure Lifeshare process, the Secure Lifeshare menu option starts a
VNC server, starts a listening vncviewer on the server and tells the server to connect to
the viewer. The code for a secure lifeshare connection is as follows:

```java
public void actionPerformed(ActionEvent e) {
```
new Thread() {
    @Override
    public void run() {
        try {
            vnc.StartVnc(true);
            Runtime.getRuntime().addShutdownHook(new Thread()
            {
                @Override
                public void run() {
                    if (vnc != null) {
                        try {
                            vnc.StopVnc();
                        } catch (IOException ex) {
                            Logger.getLogger(LifeShareMenuPlugin.class.getName()).log(Level.SEVERE, null, ex);
                        }
                        vnc = null;
                    }
                }
            });
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        AppLauncher al = new AppLauncher();
        int port = r.nextInt(100);
        logger.severe("Starting connection on port "+ port);
        startProxy(5500 + port, remoteHost);
        al.doIt(port);
        try {
            Thread.sleep(2000);
        } catch (InterruptedException ex) {
            Logger.getLogger(LifeShareMenuPlugin.class.getName()).log(Level.SEVERE, null, ex);
        }
        try {
            vnc.applyDefaultSettings();
            vnc.ConnectToLocalHost();
        } catch (IOException ioe) {
            ioe.printStackTrace();
            secureLifeshareMenuItem.setEnabled(false);
            shutdownItem.setEnabled(true);
        }
    }
}.start();
SAS Hooks

The manual creation of cells is not something that is well-documented in Wonderland, so much of the code for the SAS Hooks was developed by reverse engineering existing parts of Wonderland. For example, there is a window in Wonderland that allows a user to select a cell (the technical term for an in-world object) from a list of available cells to create. Code from that window was used as a template to develop the custom app launcher. The derived code is as follows:

```java
// Derived from CellPalette.createActionPerformed
private void createCell(final int port) {
    AppCellXrwServerState state = new AppCellXrwServerState();
    String command = "vncviewer -listen ";
    if (port > 0) {
        command = command + port;
    } else {
        command = command + 5555;
    }
    state.setCommand(command);
    state.setLaunchLocation("server");
    state.setAppName("lifeshare");
    state.setName("Lifeshare");
    logger.info("cell = " + cell);
    try {
        CellUtils.createCell(state);
    } catch (CellCreationException ex) {
        logger.warning("Cannot create cell for app " + displayName);
        return;
    }
}
```

Figure 8 - Excerpt from AppLauncher.java

The complete AppLauncher.java can be found in Appendix D. Adding to the difficulty of this sub-module was the lack of documentation on the SAS’s inability to run custom commands. Additionally, the SAS was provided for users and developers in binary
format only, further complicating efforts in discerning behavior. Thankfully, the error message produced by attempting to use a runtime-configured command was descriptive enough to realize what variable needed to be set. By going into the web interface for editing configuration variables and setting “SasXrwProviderMain.enable.user.commands” on the SAS configuration page to true, the SAS is allowed to use the custom commands Lifeshare was generating. These commands were customized to allow for the port to be selected at runtime.

**VNC Server Hooks**

The VNC Server Hooks, written by Robert Miller, represented a large amount of work. Originally, the researchers considered a setup similar to the old Lifeshare, with xrdp serving as some level of go-between, but because of the complexity issues xrdp introduced in the past, it was rejected. The next idea that was pursued involved using the Java Native Interface (JNI) to develop a stripped down version of TightVNC server (an open source VNC server) to allow for a series of C hooks. Those C hooks would bind to Java via the JNI and the VNC server would be controlled through calls to C methods through JNI. This solution, despite being rather complex would have worked. Recent developments in TightVNC allowed for a less complex, alternate method. TightVNC had long had the ability to do pseudo-RPC (remote procedure call) behavior in the Linux version. The pseudo-RPC behavior let any call to the executable control one instance. A recent build in the trunk of TightVNC had ported this functionality to Windows. With this feature in TightVNC, all the researchers had to do was develop C calls to make
console command callouts with specific arguments. The new TightVNC would automatically realize it was already running and apply the arguments to the current session. This was combined with some lightweight code to install TightVNC and together forms the entire VNC server sub-module. An example of one of the main methods for the VNC server sub-module is as follows

```java
/**
 * Connects to a listening client on another host
 * @param ipAddress
 * @param port
 * @throws IOException
 */
public void ConnectToRemoteHost(String ipAddress, String port) throws IOException {
    Runtime r = Runtime.getRuntime();
    r.exec("cmd /C C:\\Temp\\WinVNC.exe -connect " + ipAddress + ":" + port);
}
```

Figure 9 - A selection from LifeShare.java

The code in Figure 9 retrieves the currently in-use runtime and tells VNC to connect to a given ip address and port. The full code for the VNC sub-module is available in Appendix E.

**SSH Tunnel**

The requirement for secure encryption for Lifeshare was one the researchers attempted to face head on. Using code from the older version of Lifeshare, attempts were made to write a SSL-based tunnel from scratch. This approach ran into multiple issues including socket hangup and certificate delivery problems. The manual SSL proxy idea was scrapped when the suggestion of running the VNC traffic through a SSH tunnel was approached. The researchers started compiling a list of various known SSH libraries in C
with the intent to use JNI for integration. It was realized that a pure-Java SSH library, JSch, already existed. With JSch the task of writing a SSH tunnel went from daunting to trivial. The code that actually opens the SSH tunnel is provided below:

```java
int tunnelLocalPort = 5500;
  String tunnelRemoteHost = remoteHost;
  if (username == null || password == null) {
    throw new IllegalArgumentException("No username or password found!");
  }
  JSch jsch = new JSch();
  session = jsch.getSession(username, host, port);
  session.setPassword(password);
  localUserInfo lui = new localUserInfo();
  session.setUserInfo(lui);
  session.connect();
  session.setPortForwardingL(tunnelLocalPort, tunnelRemoteHost, remotePort);
  System.out.println("Connected");
```

Figure 10 - A selection from SSHPortForward.java

The code in Figure 10 simply initializes the JSch library, opens a session given a set of user credentials and calls `setPortForwarding` to start the tunnel. The full code for SSHPortForward.java is available in Appendix F.

**Lifeshare Stress Test**

After the Lifeshare prototype was completed, a stress test was launched against the environment. To perform the test the default “garden-arches” world was loaded and a single desktop was shared from the test client with the avatar in its default location.

Using an open computer lab, the team started logging into the world with additional clients one at a time, allowing each to fully load before going onto the next. The team recorded the average frames per second (FPS) on both the workstation that was sharing the desktop (twin hyperthreaded Xeons at 3 GHz, 8 GB of RAM running Windows XP 64)
and the lab computers that were serving as stress testers (Core 2 Duo at 2.66 GHz, 4GB of RAM running Windows Vista Ultimate 32-bit). These average FPS numbers were gathered using the built-in FPS Monitor in Wonderland. In Figure 6 the results of the stress test are illustrated. The red line is the average FPS of the viewing clients while the blue line represents the average FPS of the client doing the shared desktop. The frame rate data is plotted against a number of increasing clients. This figure illustrates a slow but steady decrease in FPS as you add more clients to the world.

![Frame rate vs Number of Viewing Clients](image)

Figure 11 – Chart detailing average frame rate

Complete data from the stress test is provided in Appendix A. The numbers in the chart in Appendix A are individual frame rate readings with the x axis representing successive readings after more clients were added and the y axis representing those separate clients. Until a client was added it provided no frame rate data, ergo the large number
of blank cells in the table. A control test where all twelve viewing clients and the sharing client were attached to the world without Lifeshare running was also tested. Previous literature on more demanding 3d systems shows a minimal acceptable frame rate of 10-12 frames per second [17] [18]. During the control test the average FPS on the viewing clients went from 17.95 FPS to 18.166 FPS indicating that Lifeshare was only causing a very marginal amount of FPS loss and still keeping the world well above the minimum of 10-12.
**Chapter 4 – Conclusions and Future Work**

**Conclusions**

From the previous work noted in virtual worlds, two conclusions can be derived. First, the focus of this thesis, desktop sharing in virtual worlds, is currently an intersection between two distinct research areas. The topic of desktop sharing in virtual worlds has not been analyzed jointly before. The other conclusion is that there is overlap in the features users desire in a collaborative tool, be it a virtual world, a shared desktop or both. Security, stability, and ease of use are all features users require to minimize the tool’s overhead [13].

The development of Lifeshare has shown that desktop sharing in a virtual environment is a feasible concept. Additionally, through experimentation, several vectors for adding desktop sharing to a virtual world have been explored and a solid, integration-focused solution has been shown to be the most useful. The stress testing has revealed a relatively minimal loss in frame rate as the number of viewers increased, leading to the conclusion that it would be feasible to increase the number of users beyond the scope of the test.
**Future Work**

Lifeshare does not handle single application sharing. Single application sharing is not necessarily a technological restriction, since both the VNC server and the VNC client the program uses can support single application sharing, but the task is more of an interface concern. If a clean method for deciding the application to be shared could be determined, then single application sharing would be feasible.

Currently, despite Wonderland being multi-platform (OS X, Windows and Linux are all supported), Lifeshare only supports sharing Windows desktops. Any client may view a shared desktop. However, the team has only developed Windows XP-based VNC hooks. Lifeshare could share desktops from the other operating systems by developing VNC server hooks for both OS X and Linux. JNI supports multi-platform native code by linking at runtime to platform-specific libraries and as such would work well given the current architecture to allow desktop sharing from Linux or OS X.

The future of Lifeshare could also be altered by an in-world VNC viewer application. Unlike the current listening client to SAS relationship, an in-world VNC viewer could be programmatically executed from within Wonderland, alleviating the reliance on the SAS (and by extension the Linux based server). Additionally removing the necessity of a programmatically defined string for the SAS would also remove one of the configuration steps to install Lifeshare. Currently one of the main steps of installing Lifeshare is to copy a variable into the server’s SAS Configuration. If the need for the SAS was
eliminated, then installing Lifeshare on a Wonderland server could be as easy as “Select the Jar file with the Browse button, then click the Install Plug-in button”.

There are major risks for the future of the project. First, Wonderland’s status as a research project implies that it is subject to API changes that range from minor tweaks to complete overhauls requiring redesign of all attached components. An even greater risk is Sun’s recent removal of support for Project Wonderland. The lack of continued corporate support does not necessarily mean the death of Project Wonderland. The original Sun team has expressed interest in continuing development as a purely open source endeavor [19]. However, whether or not this non-corporate funded open-source development will keep pace with previous development is yet to be seen.
Acknowledgements
I would like to acknowledge all the researchers that have worked on or overseen Projects Lifewerx and Lifeshare, Robert Miller, John Zage, Dolores Zage, Wayne Zage, John Dailey and Vinayak Tanksale. I’d also like to thank Wayne and Dolores again for their excellent guidance in the path the research took. I would like to thank Redge Bartholomew from Rockwell Collins for sponsoring the project and the Emerging Media Initiative for bridging the gap between SERC based funding. I would also like to thank the SERC secretaries, Kim Bechdolt and Allison Moore for always being on top of paperwork issues. I’d also like to thank my fellow grad students in the department and my friendly co-workers Spencer Davis and Mary Meadows.
Works Cited


York, 2003, pp. 21-23.


Appendix A : SecurityManager.java

package org.jdesktop.lg3d.wonderland.darkstar.server;

import java.io.Serializable;
import java.util.HashSet;
import java.util.LinkedList;

import javax.media.j3d.BoundingBox;
import javax.media.j3d.Bounds;
import javax.vecmath.Point3d;
import javax.vecmath.Point3f;

import org.jdesktop.lg3d.wonderland.darkstar.server.cell.AvatarCellGLO;
import org.jdesktop.lg3d.wonderland.darkstar.server.cell.CellGLO;
import org.jdesktop.lg3d.wonderland.darkstar.server.cell.MasterCellCacheGLO;
import org.jdesktop.lg3d.wonderland.darkstar.server.cell.WFSCellGLO;
import org.jdesktop.lg3d.wonderland.darkstar.server.cell.WorldRootCellGLO;

import com.sun.sgs.app.AppContext;
import com.sun.sgs.app.ManagedObject;
import com.sun.sgs.app.ManagedReference;

public class SecurityManager implements ManagedObject, Serializable {
    private static final String BINDING_NAME = "SECURITY_MANAGER";

    private LinkedList<SecurityGroup> groupList = new LinkedList<SecurityGroup>();

    // Bounding Boxes of all of the security cells
    private static SecurityBoundsSet securityCellList = new SecurityBoundsSet();

    // Main Setup Class
    public SecurityManager() {
        System.out.println("Security Manager: Starting up");
        HashSet<ManagedReference> cellSet = getCellSet();
        for (ManagedReference rf : cellSet) {
            CellGLO tempCell = rf.get(CellGLO.class);
            checkSecurityCells(tempCell);
        }
        System.out.println("Security Manager: Security Cells Loaded");
        AppContext.getDataManager().setBinding(BINDING_NAME, this);
    }

    /**
     * Return the security manager
     */
    public static SecurityManager getSecurityManager() {
        return AppContext.getDataManager().getBinding(BINDING_NAME, SecurityManager.class);
    }
}
// Adds an new managed Security Group
public void addGroup(String gName) {
    boolean oktoadd = true;
    for (SecurityGroup temp : groupList) {
        if (temp.groupName.equals(gName)) {
            oktoadd = false;
        }
    }
    if (oktoadd) {
        System.out.println("New Managed Security Group: " + gName);
        groupList.add(new SecurityGroup(gName));
        // Don't Leave this here!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        // addMemberToGroup(gName, gName);
    }
}

// Adds unique member to group
public void addMemberToGroup(String gName, String uName) {
    //VoiceHandler voiceHandler = VoiceHandlerImpl.getInstance();
    //String bridgeinfo = voiceHandler.getVoiceBridge();
    //System.out.println("there be stuff ehre" + bridgeinfo);
    for (SecurityGroup sg : groupList) {
        if (sg.groupName.equals(gName) && !sg.groupMembers.contains(uName)) {
            sg.groupMembers.add(uName);
            System.out.println("Added " + uName + " to group " + gName);
            MasterCellCacheGLO.getMasterCellCache().revalidate();
        }
    }
}

// Removes Member from group
public boolean removeMemberFromGroup(String gName, String uName) {
    for (SecurityGroup sg : groupList) {
        if (sg.groupName.equals(gName)) {
            System.out.println("Removed " + uName + " from group " + gName);
            if (sg.groupMembers.remove(uName)) {
                MasterCellCacheGLO.getMasterCellCache().revalidate();
                return true;
            } else {
                return false;
            }
        }
    }
    return false;
}

/**
* Clears all security groups of members
*/
public void clearGroups() {
    for (SecurityGroup Group : groupList) {
        Group.groupMembers.clear();
    }
}

public boolean allowedBy(String usrName, CellGLO cell) {
    if (cell instanceof AvatarCellGLO) {
        ((TransparencyManager)(AppContext.getDataManager().getBinding("TRANSPARENCY_MANAGER", TransparencyManager.class))).checkCell((AvatarCellGLO)cell);
        if (((AvatarCellGLO) cell).getUserName().equals(usrName)) {
            // i can always see myself
            return true;
        } else {
            return checkAvatarIntersection((AvatarCellGLO) cell, usrName);
        }
    } else if (cell.getGroup() == null) {
        // make boolean for if cell has been checked after everything is
        // initialized, so this will only check once for each cell
        // except for avatar cells, so it can update what group it is in
        checkGroups(cell);
    }
    for (SecurityGroup sg : groupList) {
        if (sg.groupName.equals(cell.getGroup())) {
            if (cell.isWhitelist()) {
                return sg.userIsInGroup(usrName);
            } else {
                return !sg.userIsInGroup(usrName);
            }
        }
    }
    return (true);
}

/**
 * This function will take a cell and check against all known
 * SecurityCells
 * to determine if it belongs to a group and just doesn't realize it yet.
 * *
 * @param cell
 * the cell to be checked
 */
private void checkGroups(CellGLO cell) {
    // first parse through list of cells
if (cell instanceof WFSCellGLO)
    return;
if (cell instanceof WorldRootCellGLO)
    return;
    checkForIntersectionWithKnownSecurityCells(cell);
}

private void checkSecurityCells(CellGLO check) {
    if (!check.getChildren().isEmpty()) {
        HashSet<ManagedReference> cellChildSet = (HashSet<ManagedReference>) check.getChildren();
        for (ManagedReference rfid : cellChildSet) {
            CellGLO tempCell = rfid.get(CellGLO.class);
            checkSecurityCells(tempCell);
        }
    }
    // System.out.println(check.getClientCellClassName() + " " +
    // check.getGroup());
    if ((check.getClientCellClassName()
        .equals("org.jdesktop.lg3d.wonderland.securitycell.client.SecurityCell")
    ) ) {
        // if its not in the list add it to thie list
        if (check.getGroup() == null)
            System.out.println("How did a security cell end up empty?!);
        else {
                .getBoundsWorld(), check.getGroup(), check
                .isWhiteList());
            // System.out.println("Bounds are set, adding myself to the
            // manager. BTW I am group : " + check.getGroup() + " and have
            // bounds of " + check.getBoundsWorld());
            addGroup(check.getGroup());
            // System.out.println("Security Group name: " +
            // tempSecurityBounds.getGroup());
            if (!securityCellList.contains(tempSecurityBounds)) {
                securityCellList.add(tempSecurityBounds);
            }
        }
    }
}

private HashSet<ManagedReference> getCellSet() {
    ManagedReference mr = MasterCellCacheGLO.getMasterCellCache().
        getWorldRootRef();
    CellGLO root = mr.get(CellGLO.class);
    HashSet<ManagedReference> cellSet = (HashSet<ManagedReference>) root.
        getChildren();
    return cellSet;
}

private void checkForIntersectionWithKnownSecurityCells(CellGLO cell) {

temp = securityCellList.getList();
for (SecurityBounds scList : temp) {
    if (cell.getBoundsWorld().intersect(scList.getBounds())) {
        cell.addGroup(scList.getGroup());
        cell.setWhiteList(scList.getWhiteList());
    }
}

// Creates a tiny bounding box around an Avatar instead of a huge one
private BoundingBox getAvatarAgumentedBounds(AvatarCellGLO cell) {
    Point3f tempPos = new Point3f(0, 0, 0);
    if (cell != null) {
        tempPos = cell.getWorldPosition();
    }
    Point3d tempLower = new Point3d(tempPos.x - 1, tempPos.y - 1, tempPos.z - 1);
    Point3d tempUpper = new Point3d(tempPos.x + 1, tempPos.y + 1, tempPos.z + 1);
    BoundingBox tempBoundingBox = new BoundingBox(tempLower, tempUpper);
    return tempBoundingBox;
}

/**
 * Should you be allowed to view another avatar This should not be used to
 * do active security as it depends on can view to be triggered
 *
 * @param cell
 *          avatar cell to be tested
 * @param userName
 *          username of client asking for permission to view avatar
 * @return
 */
private boolean checkAvatarIntersection(AvatarCellGLO cell, String userName) {
temp = securityCellList.getList();
    boolean insideCellPermission = true;
    boolean observerPermission = true;
    Bounds tempBoundingBox = getAvatarAgumentedBounds(cell);
    for (SecurityBounds scList : temp) {
        if (tempBoundingBox.intersect(scList.getBounds())) {
            for (SecurityGroup sg : groupList) {
                if (sg.groupName.equals(scList.getGroup())) {
                    insideCellPermission = sg.userIsInGroup(cell.getUserName());
                    observerPermission = sg.userIsInGroup(userName);
                }
            }
            if (!scList.getWhiteList()) {
                // Additional code for white list checking
            }
        }
    }
}
insideCellPermission = !insideCellPermission;
observerPermission = !observerPermission;

if (insideCellPermission)
    if (!observerPermission) {
        cell.setWhisperingStatus(true);
        return false;
    }

if (!insideCellPermission)
    if (!observerPermission) {
        cell.setWhisperingStatus(false);
        return true;
    }

if (!insideCellPermission)
    if (observerPermission) {
        cell.setWhisperingStatus(false);
        return false;
    }

if (insideCellPermission)
    if (observerPermission) {
        cell.setWhisperingStatus(true);
        return true;
    }

// This should never happen
return true;

private class SecurityGroup implements Serializable {
    String groupName;
    LinkedList<String> groupMembers = new LinkedList<String>();

    SecurityGroup(String gn) {
        groupName = gn;
    }

    /**
     * This function returns whether or not the user in question is allowed
     * by the security group that is being called This function will
     * properly use white-lists and blacklists. Nifty!
     * @param user
     * the user to check against
     * @return true if the user is on the white-list or not on the
     * blacklist, false otherwise
     */
    boolean userIsInGroup(String user) {

for (String member : groupMembers) {
    if (user.equals(member))
        return true;
}

return false;
}
Appendix B: SecurityCellGLO.java

package org.jdesktop.lg3d.wonderland.securitycell.server;

import org.jdesktop.lg3d.wonderland.darkstar.server.cell.*;
import org.jdesktop.lg3d.wonderland.darkstar.server.setup.*;
import org.jdesktop.lg3d.wonderland.securitycell.common.SecurityCellSetup;
import javax.media.j3d.*;
import javax.vecmath.Point3d;
import javax.vecmath.Vector3d;

import org.jdesktop.lg3d.wonderland.darkstar.server.cell.CellGLO;
import org.jdesktop.lg3d.wonderland.darkstar.server.UserGLO;
import com.sun.sgs.app.ManagedReference;

public class SecurityCellGLO extends StationaryCellGLO implements BeanSetupGLO{
    private float transparancy;
    private float[] bounds = new float[6];
    private String securityGroup;
    private boolean whitelist;
    Bounds CollisionBounds;

    @Override
    public SecurityCellSetup getSetupData(){
        SecurityCellSetup setup = new SecurityCellSetup();
        setup.setTransparancy(this.transparancy);
        setup.setBounds(bounds);
        setup.setSecurityGroup(this.securityGroup);

        return setup;
    }

    public void reconfigureCell(CellGLOSetup setupData){
    }

    @Override
    public String getClientCellClassName(){
        return "org.jdesktop.lg3d.wonderland.securitycell.client.SecurityCell";
    }

    @Override
    public boolean allowed(UserGLO user){

System.out.println(securityGroup);
if (user.getUserName().equals(securityGroup))
{
    System.out.println("Welcome " + user.getUserName());
    System.out.println(this.getParents());
    for(ManagedReference e: this.getParents()){
        System.out.println(e);
    }
    return true;
}
else
{
    System.out.println("Locking out " + user.getUserName());
    return false;
}

@Override
public CellGLOSetup getCellGLOSetup() {
    return new BasicCellGLOSetup<SecurityCellSetup>(getBounds(),
    getOrigin(), getClass().getName(), getSetupData());
}

@Override
public void setupCell(CellGLOSetup setupData) {
    super.setupCell((BasicCellGLOSetup<?>) setupData);
    BasicCellGLOSetup<SecurityCellSetup> setup =
    (BasicCellGLOSetup<SecurityCellSetup>)setupData;
    this.transparancy = setup.getCellSetup().getTransparancy();
    this.bounds = setup.getCellSetup().getBounds();
    this.whitelist = setup.getCellSetup().getWhiteList();
    this.addGroup(setup.getCellSetup().getSecurityGroup());

    Point3d lower = new Point3d (bounds[0],bounds[1],bounds[2]);
    Point3d upper = new Point3d (bounds[3],bounds[4],bounds[5]);
    super.setBounds(new BoundingBox(lower, upper));
    System.out.println("Security CELL: Is a whitelist cell " +
    whitelist);
    System.out.println("Security CELL: Group of : " +
    this.getGroup());
    System.out.println("Security CELL: Cell has bounds of " +
    this.getBoundsWorld());
}
@Override
public Bounds getBoundsWorld()
{
    Vector3d trans = new Vector3d();
    this.getOrigin().get(trans);

    BoundingBox bb = (BoundingBox) this.getBounds();
    Point3d upper = new Point3d();
    Point3d lower = new Point3d();
    bb.getUpper(upper);
    bb.getLower(lower);

    Point3d uptemp = new Point3d(upper);

    upper.x = trans.x + ((upper.x - lower.x) / 2);
    upper.y = trans.y + ((upper.y - lower.y) / 2);
    upper.z = trans.z + ((upper.z - lower.z) / 2);

    lower.x = trans.x - ((uptemp.x - lower.x) / 2);
    lower.y = trans.y - ((uptemp.y - lower.y) / 2);
    lower.z = trans.z - ((uptemp.z - lower.z) / 2);

    return new BoundingBox(lower, upper);
}
}
Appendix C: LifeShareMenuPlugin.java

package edu.bsu.idma.lifeshare.client;

import edu.bsu.idma.lifeshare.vnc.LifeShare;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JMenuItem;
import org.jdesktop.wonderland.client.BaseClientPlugin;
import org.jdesktop.wonderland.client.comms.SessionStatusListener;
import org.jdesktop.wonderland.client.comms.WonderlandSession;
import org.jdesktop.wonderland.client.jme.JmeClientMain;
import org.jdesktop.wonderland.client.login.ServerSessionManager;
import org.jdesktop.wonderland.client.login.SessionLifecycleListener;
import org.jdesktop.wonderland.common.annotation.Plugin;

/**
 * @author Ben
 */
@Plugin
public class LifeShareMenuPlugin extends BaseClientPlugin
    implements SessionLifecycleListener, SessionStatusListener {

    private static Logger logger = 
        Logger.getLogger("edu.bsu.idma.lifeshare.client.LifeShareMenuPlugin");
    private SSHPortForward serv = null;
    private JMenuItem secureLifeshareMenuItem = null;
    private JMenuItem unsecureLifeshareMenuItem = null;
    private JMenuItem shutdownItem = null;
    private LifeShare vnc = null;
    private Random r = new Random();

    @Override
    public void cleanup() {
        getSessionManager().removeLifecycleListener(this);
        secureLifeshareMenuItem = null;
        unsecureLifeshareMenuItem = null;
        try {
            vnc.disconnectClients();
            vnc.StopVnc();
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
        vnc = null;
        super.cleanup();
    }
}
@Override
public void initialize(ServerSessionManager sessionManager) {
    logger.severe("Reached init");
    sessionManager.addLifecycleListener(this);
    shutdownItem = new JMenuItem("Shutdown Lifeshare");
    shutdownItem.setEnabled(false);
    secureLifeshareMenuItem = new JMenuItem("Secure Proxied Lifeshare");
    unsecureLifeshareMenuItem = new JMenuItem("Proxy-free Lifeshare");
    vnc = new LifeShare();
    super.initialize(sessionManager);
}

@Override
protected void activate() {
    logger.severe("Reached activate");
    JmeClientMain.getFrame().addToToolsMenu(secureLifeshareMenuItem);
    JmeClientMain.getFrame().addToToolsMenu(unsecureLifeshareMenuItem);
    JmeClientMain.getFrame().addToToolsMenu(shutdownItem);
    super.activate();
}

@Override
protected void deactivate() {
    if (serv != null) {
        serv.turnOff();
    }
    JmeClientMain.getFrame().removeFromToolsMenu(secureLifeshareMenuItem);
    JmeClientMain.getFrame().removeFromToolsMenu(unsecureLifeshareMenuItem);
    JmeClientMain.getFrame().removeFromToolsMenu(shutdownItem);
    serv = null;
    super.deactivate();
}

private final void startProxy(int remotePort, String remoteHost) {
    serv = new SSHPortForward(remotePort);
    serv.setRemoteHost(remoteHost);
    try {
        serv.go();
    } catch (Exception ex) {
        Logger.getLogger(LifeShareMenuPlugin.class.getName()).log(Level.SEVERE, null, ex);
    }
}

@Override
public void sessionCreated(WonderlandSession session) {
    //Do nothing
}
public void primarySession(WonderlandSession session) {
    if (session != null) {
        session.addSessionStatusListener(this);
        if (session.getStatus() == WonderlandSession.Status.CONNECTED) {
            connectClient(session);
        }
    }
}

final String remoteHost = session.getServerInfo().getHostname();
shutdownItem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            vnc.disconnectClients();
            vnc.StopVnc();
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
        if (serv != null) {
            serv.turnOff();
        }
        shutdownItem.setEnabled(false);
        secureLifeshareMenuItem.setEnabled(true);
    }
});
secureLifeshareMenuItem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        new Thread() {
            @Override
            public void run() {
                try {
                    vnc.StartVnc(true);
                    Runtime.getRuntime().addShutdownHook(new Thread()
                    {
                        @Override
                        public void run() {
                            if (vnc != null) {
                                try {
                                    vnc.StopVnc();
                                } catch (IOException ex) {
                                    Logger.getLogger(LifeShareMenuPlugin.class.getName()).log(Level.SEVERE, null, ex);
                                }
                                vnc = null;
                            }
                        }
                    })
                }
            }
        }
    }
});
AppLauncher al = new AppLauncher();
int port = r.nextInt(100);

logger.severe("Starting connection on port " + port);
startProxy(5500 + port, remoteHost);
al.doIt(port);
try {
    Thread.sleep(2000);
} catch (InterruptedException ex) {
    Logger.getLogger(LifeShareMenuPlugin.class.getName()).log(Level.SEVERE, null, ex);
}

try {
    vnc.applyDefaultSettings();
vnc.ConnectToLocalHost();
} catch (IOException ioe) {
    ioe.printStackTrace();
}
secureLifeshareMenuItem.setEnabled(false);
shutdownItem.setEnabled(true);
}
}.start();
}

unsecureLifeshareMenuItem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        shutdownItem.setEnabled(true);
        try {
            vnc.StartVnc(true);
            Runtime.getRuntime().addShutdownHook(new Thread() {
                @Override
                public void run() {
                    if (vnc != null) {
                        try {
                            vnc.StopVnc();
                        } catch (IOException ex) {
                            Logger.getLogger(LifeShareMenuPlugin.class.getName()).log(Level.SEVERE, null, ex);
                        }
                    }
                }
            });
        }
    }
});
vnc = null;
}
});
} catch (IOException ioe) {
    ioe.printStackTrace();
}

try {
    Thread.sleep(2000);
} catch (InterruptedException ex) {
    ex.printStackTrace();
}

AppLauncher al = new AppLauncher();
int port = r.nextInt(100);
al.doIt(port);
try {
    vnc.applyDefaultSettings();
    vnc.ConnectToRemoteHost(remoteHost, "" + (port + 5500));
} catch (IOException ioe) {
    ioe.printStackTrace();
}

@Override
public void sessionStatusChanged(WonderlandSession session, Status status) {
    switch (status) {
    case CONNECTED:
        connectClient(session);
        return;

    case DISCONNECTED:
        disconnectClient();
        return;
    }
}

private void connectClient(WonderlandSession session) {
}

private void disconnectClient() {
}
Appendix D: AppLauncher.java

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package edu.bsu.idma.lifeshare.client;

import java.util.logging.Logger;
import org.jdesktop.wonderland.client.cell.utils.CellCreationException;
import org.jdesktop.wonderland.client.cell.utils.CellUtils;
import org.jdesktop.wonderland.modules.appbase.client.cell.App2DCell;
import org.jdesktop.wonderland.modules.xremwin.common.cell.AppCellXrwServerState;

/**
 * @author Ben
 */
public class AppLauncher {

    private App2DCell cell;
    private String displayName;
    private static Logger logger = Logger.getLogger("edu.bsu.idma.lifeshare.client.AppLauncher");

    public void doIt(int port) {
        createCell(port);
    }

    // Derived from CellPalette.createActionPerformed
    private void createCell(final int port) {
        AppCellXrwServerState state = new AppCellXrwServerState();
        String command = "vncviewer -listen ";
        if (port > 0) {
            command = command + port;
        } else {
            command = command + 5555;
        }
        state.setCommand(command);
        state.setLaunchLocation("server");
        state.setAppName("lifeshare");
        state.setName("Lifeshare");
        logger.info("cell = " + cell);
        try {
            CellUtils.createCell(state);
        } catch (CellCreationException ex) {
            logger.warning("Cannot create cell for app " + displayName);
            return;
        }
    }
}
Appendix E: LifeShare.java

```java
package edu.bsu.idma.lifeshare.vnc;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
public class LifeShare {

    private static String TEMP_Path = "C:\Temp";
    private static String VNC_File = "\WinVNC.exe";
    private static String Hooks = "\VNCHooks.dll";
    private static String RegFile = "\RegFile.dll";

    public static void main(String[] args) {
        LifeShare temp = new LifeShare();
        temp.applyDefaultSettings();
        try {
            temp.StartVnc(true);
            Thread.currentThread().sleep(1000);
            temp.ConnectToLocalHost();
            Thread.currentThread().sleep(6000);
            //temp.StopVnc();
            //temp.ConnectToLocalHost();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public LifeShare() {
        try {
            getFile("WinVNC.exe", TEMP_Path, VNC_File);
            getFile("VNCHooks.dll", TEMP_Path, Hooks);
            getFile("WinVncSettings.reg", TEMP_Path, RegFile);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```
public void applyDefaultSettings(){
    Runtime r = Runtime.getRuntime();
    try {
        r.exec("regedit /s " + TEMP_Path + RegFile);
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

public void StartVnc(boolean showIcon) throws IOException{
    Runtime r = Runtime.getRuntime();
    if (showIcon) {
        r.exec(TEMP_Path + VNC_File);
    } else {
        r.exec(TEMP_Path + VNC_File + "-silent");
    }
}

public void StartVnc() throws IOException{
    StartVnc(false);
}

public void StopVnc() throws IOException{
    Runtime r = Runtime.getRuntime();
    r.exec(TEMP_Path + VNC_File + "-kill");
}

/**
 * Experimental -> Untested
 * @param windowName
 */
public void ShareWindow(String windowName) throws IOException{
    Runtime r = Runtime.getRuntime();
    r.exec(TEMP_Path + VNC_File + "-sharewindow " + windowName);
}
/**
 * Kill all Clients
 * @throws IOException
 */
public void disconnectClients() throws IOException
{
    Runtime r = Runtime.getRuntime();
r.exec(TMP_PATH + VNC_FILE + "-killallclients");
}

/**
 * Connects to listening client on local host on the loop back
 */
public void ConnectToLocalHost() throws IOException
{
    ConnectToRemoteHost("127.0.0.1");
}

/**
 * Connects to a listening client on another host
 * @param ipAddress
 */
public void ConnectToRemoteHost(String ipAddress) throws IOException
{
    ConnectToRemoteHost(ipAddress, "5500");
}

/**
 * Connects to a listening client on another host
 * @param ipAddress
 * @param port
 * @throws IOException
 */
public void ConnectToRemoteHost(String ipAddress, String port) throws IOException
{
    Runtime r = Runtime.getRuntime();
r.exec("cmd /C C:\Temp\WinVNC.exe -connect " + ipAddress + ":" + port);
}

private static void getFile(String filename, String path, String localPath) throws IOException
{
    InputStream t = LifeShare.class.getResourceAsStream(filename);
    InputStream in = t;

    File file = new File(path);
    if (!file.exists())
    {
        file.mkdirs();
    }

    InputStream s = new FileInputStream(filename);
    FileChannel fc = s.getChannel();
    fc.transferTo(0, fc.size(), new FileOutputStream(localPath));
}
```java
file = new File(path + localPath);
if (!file.exists())
{
    file.createNewFile();
}

OutputStream out = new java.io.FileOutputStream(file);
int nextChar;
while ( ( nextChar = in.read() ) != -1 )
{
    out.write( nextChar );
    out.flush();
}

out.close();
t.close();
```
Appendix F: SSHPortForward.java
package edu.bsu.idma.lifeshare.client;

import com.jcraft.jsch.JSch;
import com.jcraft.jsch.Session;
import com.jcraft.jsch.UserInfo;
import java.awt.event.WindowAdapter;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JDialog;
import org.jdesktop.wonderland.client.jme.JmeClientMain;

/**
 * This class uses jsch to open a simple SSH Port Tunnel
 * @author Ben
 */
public class SSHPortForward extends Thread implements UserPasswordStore {
    private int remotePort = 5555;
    private String remoteHost = "";
    private String username = null;
    private String password = null;

    /**
     * This sets the port to tunnel to on the remote machine
     * @param remotePort the target port
     */
    public void setRemotePort(int remotePort) {
        this.remotePort = remotePort;
    }

    /**
     * This sets the host to tunnel to
     * @param remoteHost the target machine
     */
    public void setRemoteHost(String remoteHost) {
        this.remoteHost = remoteHost;
    }

    /**
     * This constructor builds the ssh port forward with a set remote port
     * @param remotePort the port to forward to
     */
    public SSHPortForward(int remotePort) {
        this.remotePort = remotePort;
    }

    private Session session = null;
    public void go() throws Exception {
        String host = remoteHost;
        JDialog inputDialog = new UserPassInputDialog(JmeClientMain.getFrame().getFrame(), true, this);
    }
inputDialog.setEnabled(true);
inputDialog.setVisible(true);
while (inputDialog.isEnabled()) {
    Thread.sleep(50);
}

int port = 22;

int tunnelLocalPort = 5500;
String tunnelRemoteHost = remoteHost;
if (username == null || password == null) {
    throw new IllegalArgumentException("No username or password found!");
}
JSch jsch = new JSch();
session = jsch.getSession(username, host, port);
session.setPassword(password);
localUserInfo lui = new localUserInfo();
session.setUserInfo(lui);
session.connect();
session.setPortForwardingL(tunnelLocalPort, tunnelRemoteHost, remotePort);
System.out.println("Connected");
}

@Override
public void run() {
    try {
        this.go();
    } catch (Exception ex) {
        Logger.getLogger(SSHPortForward.class.getName()).log(Level.SEVERE, null, ex);
    }
}

/**
 * This stops the SSH tunnel
 */
public void turnOff() {
    session.disconnect();
}

public String getUsername() {
    return username;
}

public String getPassword() {
    return password;
}

public void setUsername(String username) {
    this.username = username;
}
```java
public void setPassword(String password) {
    this.password = password;
}

class localUserInfo implements UserInfo {
    String passwd;

    public String getPassword() {
        return passwd;
    }

    public boolean promptYesNo(String str) {
        return true;
    }

    public String getPassphrase() {
        return null;
    }

    public boolean promptPassphrase(String message) {
        return true;
    }

    public boolean promptPassword(String message) {
        return true;
    }

    public void showMessage(String message) {
    }
}

```
### Appendix G: Stress Test Data

<table>
<thead>
<tr>
<th>Number of Clients</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sharing Client</td>
<td>10.2</td>
<td>9.4</td>
<td>9.6</td>
<td>9.3</td>
<td>8.8</td>
<td>8.7</td>
<td>8.4</td>
</tr>
<tr>
<td>View 1</td>
<td>21.8</td>
<td>21.9</td>
<td>21.4</td>
<td>20.8</td>
<td>20.6</td>
<td>20.2</td>
<td></td>
</tr>
<tr>
<td>View 2</td>
<td>21.5</td>
<td>21.2</td>
<td>20.8</td>
<td>20.4</td>
<td>20.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>View 3</td>
<td>21.2</td>
<td>20.7</td>
<td>20.5</td>
<td>20.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View 4</td>
<td></td>
<td>20.9</td>
<td>20.9</td>
<td>20.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View 5</td>
<td></td>
<td></td>
<td>20</td>
<td>19.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>19.8</td>
</tr>
<tr>
<td>View 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View 9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View 11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>View 12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg Viewer</td>
<td>21.8</td>
<td>21.7</td>
<td>21.26667</td>
<td>20.8</td>
<td>20.48</td>
<td>20.06667</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.3</td>
<td>8.8</td>
<td>8.1</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>7.9</td>
</tr>
<tr>
<td>19.6</td>
<td>19.5</td>
<td>19.1</td>
<td>18.5</td>
<td>18.5</td>
<td>18.1</td>
<td>18.2</td>
</tr>
<tr>
<td>19.7</td>
<td>19.4</td>
<td>19</td>
<td>18.6</td>
<td>18.4</td>
<td>18.1</td>
<td>18.1</td>
</tr>
<tr>
<td>19.7</td>
<td>19.4</td>
<td>18.9</td>
<td>18.6</td>
<td>18.3</td>
<td>18</td>
<td>18.1</td>
</tr>
<tr>
<td>20.1</td>
<td>19.7</td>
<td>19.4</td>
<td>18.8</td>
<td>18.7</td>
<td>17.6</td>
<td>18.4</td>
</tr>
<tr>
<td>19.4</td>
<td>19.1</td>
<td>18.7</td>
<td>18.4</td>
<td>18</td>
<td>17.4</td>
<td>17.6</td>
</tr>
<tr>
<td>20</td>
<td>19.6</td>
<td>19</td>
<td>18.9</td>
<td>18.4</td>
<td>18.2</td>
<td>18.3</td>
</tr>
<tr>
<td>19.7</td>
<td>19.4</td>
<td>18.9</td>
<td>18.8</td>
<td>18.4</td>
<td>17.9</td>
<td>18.2</td>
</tr>
<tr>
<td>19.4</td>
<td>19.1</td>
<td>18.6</td>
<td>18.5</td>
<td>18.1</td>
<td>18.2</td>
<td></td>
</tr>
<tr>
<td>18.9</td>
<td>18.5</td>
<td>18</td>
<td>17.8</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18.5</td>
<td>18.4</td>
<td>18.1</td>
<td>18.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18.4</td>
<td>18.1</td>
<td>18.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>18.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19.74286</td>
<td>19.4375</td>
<td>19</td>
<td>18.62</td>
<td>18.36364</td>
<td>17.95</td>
<td>18.16667</td>
</tr>
</tbody>
</table>