CyberLiveApp: A Secure Sharing and Migration Approach for Live Virtual Desktop Applications in a Cloud Environment

Jianxin Li*, Yu Jia*, Lu Liu+, Tianyu Wo*

*School of computer Science & Engineering
Beihang University, Beijing, China
E-mail: {lijx,jiayu,woty }@act.buaa.edu.cn

+School of Computing and Mathematics
University of Derby, Derby, UK
E-mail: l.liu@derby.ac.uk

Abstract

In recent years we have witnessed the rapid advent of cloud computing, in which the remote software is delivered as a service and accessed by users using a thin client over the Internet. In particular, the traditional desktop application can execute in the remote virtual machines without re-architecture providing a personal desktop experience to users through remote display technologies. However, existing cloud desktop applications mainly achieve isolation environments using virtual machines (VMs), which cannot adequately support application-oriented collaborations between multiple users and VMs. In this paper, we propose a flexible collaboration approach, named CyberLiveApp, to enable live virtual desktop applications sharing based on a cloud and virtualization infrastructure. The CyberLiveApp supports secure application sharing and on-demand migration among multiple users or equipment. To support VM desktop sharing among multiple users, a secure access mechanism is developed to distinguish view privileges allowing window operation events to be tracked to compute hidden window areas in real time. A proxy-based window filtering mechanism is also proposed to deliver desktops to different users. To support application sharing and migration between VMs, we use the presentation streaming redirection mechanism and VM cloning service. These approaches have been preliminary evaluated on an extended MetaVNC. Results of evaluations have verified that these approaches are effective and useful.

Key words: Cloud Computing; Software as a Service (SaaS); Virtual Machine; Secure Accessing; Live Application Sharing and Migration

1. INTRODUCTION

The emergence of Internet-based computing paradigms, e.g., cloud computing [1][2] have allowed the Internet to develop into a data storage and computing centre. Large-scale computing, storage and data service resources can be brought together to build a virtual computing environment. The paradigms provide simple and transparent approaches to enable effective sharing and utilisation of applications over the Internet.

In a traditional computing environment, users need to locally install software under a granted license before use. The limitations of the traditional method have become more apparent as software has proliferated. Software users may be burdened with many complex tasks in terms of software installation, configuration, updating and even troubleshooting when dependency on the host operating system causes compatibility issues. In contrast, many companies, such as Amazon and Google have promoted the relative simplicity of the software as service (SaaS) concept [3]. Software can be installed in VMs with easier encapsulation and secure isolation. Users access software on demand through the Internet without any incumbent update and maintenance issues. For the provider there are two alternative methods of making the SaaS software available. One is to develop or redevelop software (e.g., GoogleDoc) based on Web technologies. This not only requires significant work, but may also encounter compatibility problems with the numerous current and developing browsers. The second approach is based on desktop virtualization, which separates the presentation and execution of applications. This provides a transparent way to deliver an application based remote virtual desktop.

Currently virtual desktops are delivered using remote display protocols such as VNC (Virtual Network Computing) [4][5] and RDP (Remote Desktop Protocol) [6]. These protocols generally provide methods for accessing a remote virtual desktop, users login to a VM and work on the desktop.

Secure collaboration has always been a field of intense interest [7]. As user terminal equipment continues to develop mobility, users want to have access their desktops at anytime and anywhere. This level of mobility is made possible in cloud computing where applications are executed in VMs. Providing a novel flexible collaboration service among live virtual desktop applications (*live application* in short) is the focus of our work. We are presenting some novel scenarios for live application sharing in single or multiple VMs. In a single VM, collaboration scenarios can be supported based on a shared desktop, for example, in a remote teaching system desktop sharing enables the instructor and students to work on the same view. The normal way to authorise remote desktop sharing is to simply share the desktop login accounts and passwords. In a cloud, a virtual machine monitor only provides coarse-grained access control with VM-level granularity. The authorization having only two possible results, success or failure, means users will either all have full access rights with consequent poor privacy and security, or have no concurrent access to the desktop. In particular, it is impractical to share the whole desktop including users' private windows. Thus, a fine-grained access control mechanism with window-level granularity is required to provide security and privacy protection.

Example 1: *Three applications* (*Document editing, Image Viewing and Chatting*) *are running on a VM owned by User A. A wants to share the desktop with all these applications to User B, but block input from B's keyboard and mouse. A also gives the authority of viewing the pictures on his virtual desktop to User C, but does not want to show C the document editing window and the chatting window.*

In a traditional approach for application sharing among multiple VMs, when a user needs to share an application and related data with other users, binary copies of the application are generated. The replication process may involve significant installation and configuration time and may ultimately fail due to system incompatibilities. In a cloud computing environment, this situation can be easily handled through presentation redirection among application desktops or application data clone in the cloud.

Example 2: *In a virtual machine environment, user A is operating a desktop with a drawing application window on it. After finishing his part of the drawing, A intends to let user B continue with it, which means A needs to migrate the application window to B's desktop; While user C is interested in this drawing application and the drawing user A has finished, C hopes to have a complete copy of this application and this drawing, which means user A has to give a clone of the application and user data to user C.*

To meet the above requirements, we have designed a live virtual desktop application sharing and migration system, named CyberLiveApp. The major contributions include:

(1). Multi-user Secure Application Accessing in a single VM. To enable secure live application sharing and collaboration among different users, we have designed a multi-user secure accessing mechanism with application-window-level granularity. A VM owner can configure desktop sharing policies and use a proxy to filter unnecessary or private desktop windows.

- **Multi-user secure desktop accessing approach:** Considering the VM security, sharing and collaboration needs, we have designed and implemented a multi-user secure accessing approach based on MetaVNC, extending the RFB protocol to support multi-user authentication, and provided the ability for users customised views.
- **Window operation event tracking and real-time hidden area computation approach:** As the access control objects, windows generally overlap each other and may change dynamically upon events such as resizing, minimizing, maximizing, and moving. We first track the events which affect the layout of multiple windows, and implement a real-time hidden area computation approach to extract a permitted desktop area on a virtual desktop.

- **Proxy-based window filtering mechanism:** In the existing remote control software, access control mechanisms are generally coarse-grained desktop-level. This does not satisfy the window sharing requirements. We implement a desktop view proxy for desktop delivery among different users. The proxy calculates permitted desktop areas and clips hidden areas based on VM owner's specific security policy.

(2). Application Sharing and Migration among Multiple VMs: To enable collaborations among multiple VMs, we have designed a live application sharing and migration mechanism. Through presentation streaming redirection and VM cloning technology, an application can be easily shared or migrated. We also propose an algorithm for maintaining application state and message consistency during application sharing and migration.

- **Presentation streaming redirection approach:** Using live application presentation streaming, users can subscribe to software services and access the virtual desktop. In a cloud environment, all VMs are located in the same network. Due to the encapsulation and migration capabilities of the VM, and the favourable communication bandwidth among the hosts, sharing and migration of live applications between different clients can be achieved through redirection of application presentation streaming. During the migration and sharing process, a VNC connection on a particular client side can be easily created or closed.
- **Application state and message consistency algorithm:** Ensuring the consistency of all application states during application sharing and migration is a critical problem. It means all involved users should get the same view on a shared application, including the application configuration and all user data (stored in both disks memories). During application sharing, a target user should get a complete copy of the application. We achieve this by cloning the VM in which the target application is running. We have designed an algorithm to coordinate the communication sequences among users, and define application state change rules to ….
- **Virtual desktop application sharing & migration protocols:** To overcome the inflexibility of traditional application copy and desktop sharing, we have designed two protocols for virtual desktop application sharing and migration. In the protocols, we implement a desktop session management approach based on inter-process communication. With these protocols, clients and servers communicate via SOAP messages and application sharing is achieved based on VM cloning.

(3). All approaches mentioned above have been implemented in the CyberLiveApp based on extended MetaVNC and the virtual machine monitor KVM. Furthermore, we have performed simulations to verify that the approaches are effective and useful.

The rest of this paper is organized as follows. We discuss the requirements for live applications in Section 2. Section 3 presents the design of CyberLiveApp, the architecture and protocols. We elaborate implementation experiences in Section 4. The related work is discussed in Section 5. Finally, we conclude the paper in Section 6.

2. Requirements Analysis

In a VM, when a user wants to share applications on his/her virtual desktop to other users, basic requirements for CyberLiveApp are summarized as follows:

Requirement 1: Providing a multi-user accessing mechanism, and controlling viewable desktop areas at application window granularity.

Requirement 2: Providing the capability to enable/disable input devices such as keyboard and mouse. For instance, some users are allowed to browse some windows but not to operate or edit them.

Requirement 3: Preventing a third party from eavesdropping on the content of desktop during transmission, i.e., to ensure confidentiality of transmitted context though an encrypted tunnel.

As shown in Figure 1, there are three users connecting to the same VM to view its virtual desktop, but their permissions are different. Table 1 is an example of permission matrix. User *A* shares all the windows with user *B*, but does not permit *B* to operate the windows via his/her keyboard or mouse. User *A* also shares the desktop to user *C* after hiding the window areas of text editing and chatting program. User *C* is also not allowed to operate *A*'s desktop.

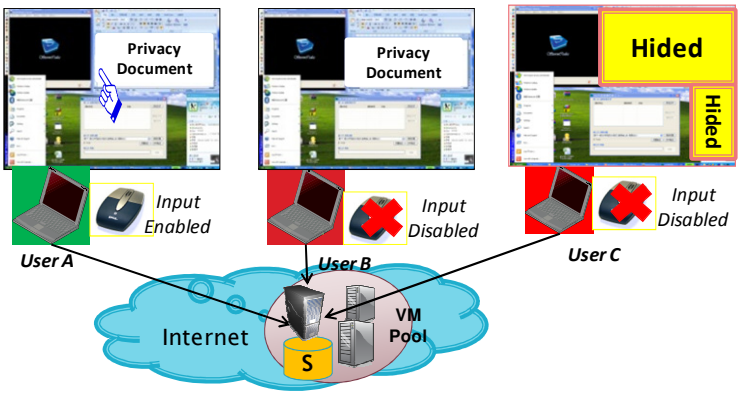


Figure 1: A scenario of secure remote access to a single VM from multiple users

Table 1: Permission matrix for multiple users

	ACDSee Picture Brower	Office Word	MSN	Mouse & Keyboard Input
Owner *A*	Y	Y	Y	Y
User *B*	Y	Y	Y	N
User *C*	Y	N	N	N

CyberLiveApp achieves the display of remote desktop based on MetaVNC, some key technologies are as follows:

- **Secure access control for multiple users**: The virtual desktop can be accessed by multiple users. Every user has a customized view, and can connect to the virtual desktop in a secure way.
- **Configuration of sharing policy among users**: The virtual desktop owner can specify policies for its applications, and configure the permitted users and their permissions as shown in Table 1.
- **The extraction of application windows**: For the convenience of administrators to assign browse permissions for different users, CyberLiveApp extracts all the windows from the operating system (OS), and stores them in a list. It first gets the handles of all the windows on the desktop, then obtains the processes they belong to, and finally extracts the names of the processes.
- **Hiding specific application windows**: CyberLiveApp selects private windows and calculates the hidden area related to these windows in accordance with the sharing policies.
- **The blocking of keyboard and mouse events**: Through intercepting keyboard and mouse events of different users, the server side of CyberLiveApp decides whether or not to block the keyboard and mouse input events.

Among multiple VMs, when a user wants to copy or migrate applications on a virtual desktop, the source and the destination of application sharing or migration should be specified. As shown in Figure 2, in the VM pool, user *A* has two VMs running *App*1 and *App*2, respectively. User *A* can connect to the virtual desktop to view *App*1 and *App*2 through a notebook computer (Client 1). When user A leaves, he can migrate the *App*1 to his mobile terminal (Client 2). User *B* is very interested in *App*2, and user *A* can

easily send this application to user *B*'s client (Client 3).

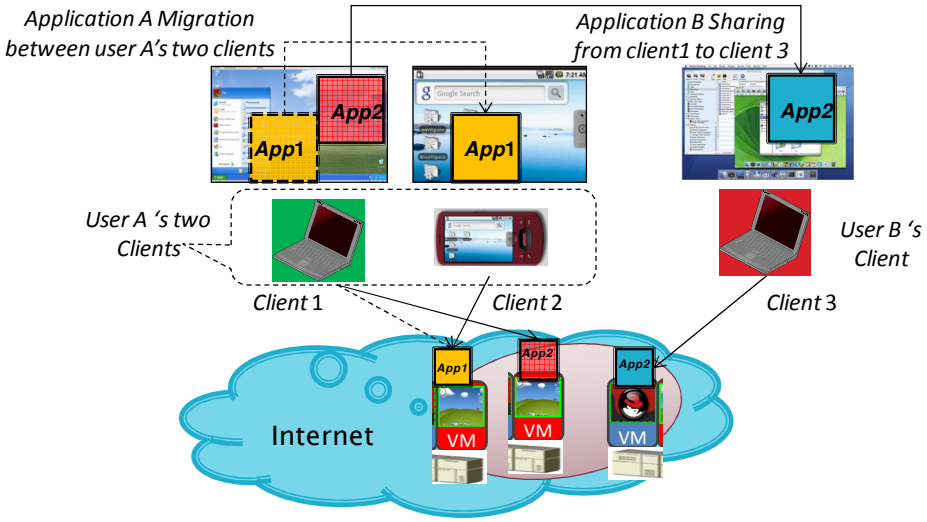


Figure 2: A scenario of live application sharing & migration between deferent VMs

The key technologies of live application sharing and migration among multiple VMs include:

- **Redirection of application presentation streaming among multiple client terminals**: In a cloud environment, all applications are maintained in a VM pool, where the virtual desktops are delivered to client terminals through VNC or RDP protocols. In case of application migration or sharing, application windows can be redirected to a new client terminal.
- **Management of VNC connections on the client side:** CyberLiveApp achieves the goal of remote desktop access based on MetaVNC. During the phase of application sharing and migration, an inter-processes communication technology is applied on each client to automatically create/close a specified VNC connection, thus achieving the redirection of application presentation streaming.
- **Application states consistency guarantee:** CyberLiveApp uses a consistent algorithm to regulate the communication sequence. Two protocols have been designed for virtual application sharing and migration, which will be introduced in section …

3. Design of CyberLiveApp

3.1 Secure Sharing of a Single VM Desktop among Multiple Users

Figure 3 shows the architecture of secure desktop sharing for multiple users. For every VM owned by a user, the display of running applications is delivered to the `vncViewer` on the client sides through a `vncServer` installed in the VM.

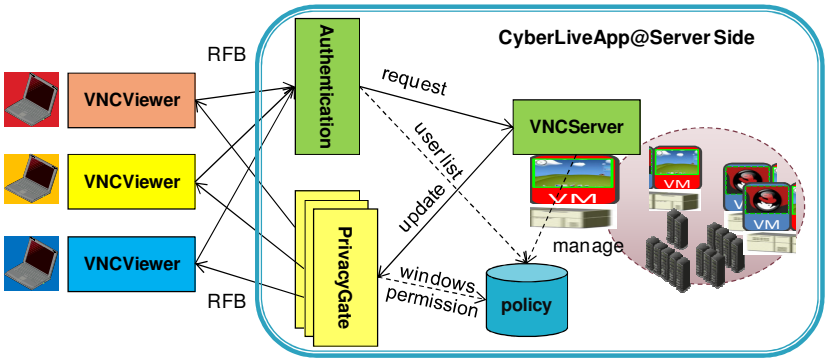


Figure 3: Architecture of secure desktop sharing for multiple users

The basic procedure of secure desktop sharing among multiple users is as follows:

1. **Extraction of application windows in the operating system**: To enable administrators to assign application permissions to users at window granularity, CyberLiveApp needs to extract all related windows from the operating system. Firstly, CyberLiveApp gets all of the windows handlers on a desktop. Secondly, CyberLiveApp uses the windows handlers to obtain related process handles which the windows belong to, by which we can get the application process names. Finally, CyberLiveApp creates an available application list to users for making security policies.
2. **Policy configuration on CyberLiveApp@Server side**. The owner of virtual desktop creates a list of those who are allowed to access his desktop and assign permissions. The policy is de facto an access control list. The policies include two types - one is a *permit policy* which defines the windows that a user can access, and the other is a *forbid policy* which defines the windows that a user cannot access. When a `vncServer` runs in the OS, you can right-click the MetaVNC icon to pop-up a menu. When one chooses `Properties menu`, some configuration options are showed on the tabs, and we add two extra tabs on MetaVNC options dialog to support such configuration.
3. **Authentication of requested users.** When a user wants to access the remote virtual desktop, it will firstly launch the `vncViewer` to send a connection request to the `vncServer` though the RFB protocol. The *authentication* module will authenticate the identity of the requesting user based on the local *policy* database. If the user is authenticated and acquires the permissions, the `vncServer` will build a connection with the `vncViewer`.
4. **The PrivayGate module functions**: In current MetaVNC functions, a `vncClient` object is created when the `vncServer` connection is created. Once some events occur on the server desktop, the `vncServer` will deliver updated desktop to every `vncClient`. Therefore, each client will get the exact same desktop view. In order to allow each user with a customized view, the `vncServer` works on the proxy mode to hide some specified windows according to the policies. The significance of our design is a breakthrough in the current MetaVNC structure. A PrivacyGate is used to send the desktop to each `vncClient`, which means that when the server desktop is changed, it will update the desktop view though a `PrivacyGate` module based on the *permit policy* or *deny policy*. The `PrivacyGate` module will hide the window area that the corresponding user has no browsing permission, and then deliver the filtered desktop view to every `vncViewer`.
5. **Hiding specific application windows of PrivacyGate:** In order to allow every user to have an independant view, the `vncServer` needs to hide the specified windows according to the policy. Firstly, CyberLiveApp gets the names of the invisible windows which administrator configured for the user. Through this name, CyberLiveApp calls a query function to obtain the handlers of the application window and then gets rectangle area of this window. Secondly, this rectangle area is added into the hidden area which could be a simple rectangle or a complex polygon. When setting the hidden area, some overlapped areas with the hidden windows belonging to the permitted top-level windows should be shown.
6. **Blocking input from keyboard and mouse at PrivacyGate:** Since some users are permitted to only browse, not operate, a desktop, CyberLiveApp blocks input of keyboard and mouse from these users. If the *input variable* state is "*`DISABLE`*", the server will block the client's inputs, and the user can only view the desktop windows, but cannot operate the server's windows. If the *input variable* is "*`ENABLE`*", the user's inputs from the remote client will be handled and responded normally. We control the input operations through Windows `hook` function, it intercepts and processes the requests to decide whether to forward the input

events.

3.2 Application Sharing and Migration among Multiple VMs

To realize remote application access in a cloud, we use the VNC protocol to transfer the virtual desktop of a remote VM. The VNC protocol works at the buffer frame layer and supports the remote access to graphical user interfaces, and the mouse or keyboard inputs can be transferred to the remote application, thus achieving a transparent access to the applications. In such a presentation streaming-based software delivery mode, when a client wants to migrate or share an application to another client, the presentation streaming of this application should be redirected and the corresponding VM will be cloned in case of application sharing. Based on these considerations, we have designed the architecture for live application sharing and migration (shown in Figure 4). The key components are as follows:

- **Client Controller.** With a modified `vncViewer` to displayapplication windows from multiple VM machines, the Client Controller manages multiple VNC connections between the `vncViewer` and `vncServers`. Using an inter-process communication technology, the Client Controller can close or create a specified VNC connection.
- **Server Controller:** It maintains information of all live users and applications in the VM pool. This component provides a unified management and processing of all clients' requests, which includes application presentation streaming, and application sharing and migration service.
- **VM Manager:** It provides functionalities of monitoring, stopping, cloning or restarting a specified VM with running applications. VM Manager receives notifications from Server Controller to clone a VM or manage the VM pool.

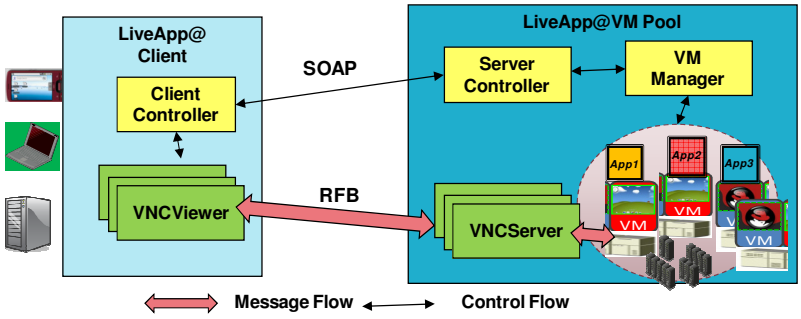


Figure 4: Architecture of application sharing and migration among multiple VMs

3.2.1 A Consistent Algorithm for Application States and Messages

In CyberLiveApp, the Server Controller is eligible to identify all the clients' requests and capable of responding to them by sending control messages to every Client Controller, while the VM Manager manages multiple VM instances in a VM pool. During the phases of application migration and sharing, application states or messages may change frequently. Therefore, we propose a consistent algorithm to ensure the consistency of various application and VM information.

The Server Controller maintains a table of operations of all virtual applications. When a client sends a request to the application, the Server Controller updates its operation. We use the symbol *op* to denote a requested operation to the application. All the operation variables are defined in Table 2.

Table 2 Definitions of all the operation variables

variable op	*Meaning*
*v*1	NORMAL
*v*2	SHARE
*v*3	DISCONNECT
*v*4	MIGRATE
*v*5	CLONE

The *op* field value of an application is one of the five types at one time. When an application's *op='NORMAL'*, that means there is no shared or migration process for this live application. During the

procedure of application sharing and migration, we use a tuple *s* to describe application states between two clients:

$$s=<op_1, op_2>$$

where op_1 denotes the application operation that a client requests, and op_2 is the application operation that another client receives. During the procedure of application migration and sharing, an application's state will be transferred according to the following order as shown in Table 3 and Table 4.

Table 3: Application sate transition during migration

Order	*s=<op₁,op₂>*	*Description*
1	<*v*1,*v*3>	Client1 has no request
2	<*v*4,*v*3>	Client1 sends a migration request to Client2
3	<*v*4,*v*4>	Client2 accepts the migration request
5	<*v*3,*v*1>	Client1 closes the application connection, and Client2 connects to it.

Table 4: Application sate transition during sharing

Order	***s=<op₁,op₂>***	**Description**
1	<*v*1,*v*3>	Client1 has no request
2	<*v*2,*v*3>	Client1 sends a sharing request to Client2
3	<*v*2,*v*2>	Client2 accepts the sharing request
4	<*v*2,*v*5>	VM Manager clones this application, and another instance will be started
5	<*v*1,*v*3>	Client1 returns its initial state as order 1

A consistent algorithm for application states and messages is implemented in the Client Controller as shown in Figure 5 to illustrate how Server Controller deals with different '*op*' values.

```
monitor(appName){
Input : AppName
1. s = reqeustState(AppName) //gets the application s from Server Controller
2. if(!s.op1.Equals("NORMAL")){
3.    ip = getIPbySWUUID(VM.swUUID); //gets the IP address where the software runs
4.    if(s.op2.Equals("MIGRATE"){
5.        VNCServer.disconnect(Client1); // disconnects the VNC connection
6.        update(s,"DISCONNECT","NORMAL");} //update its state
7.    if(s.op2.Equals("SHARE"){
8.       swUUID=VMManager.clone(); // clone a VM whose ID is swUUID
9.       update(s, "SHARE", "CLONE"); } //update its state
10.   if(s.op2.Equals("DISCONNECT"){
11.       update(s,s.op1,s.op1) ;} //accept the operation of SHARE or MIGRATE
12.   if(s.op2.Equals("CLONE"){
13.       ip2 = getIPbySWUUID(swUUID);// get the IP address of the cloned VM
14.       sendMsg(Client2,"NORMAL",ip2);} // Notify Client2 to connect the cloned VM
15.   } }
```

Figure 5: A consistent algorithm for application state and messages

In this algorithm, `RequestSendMsg()` is responsible for updating the application states, and sending notifications to the clients. We have implemented a SOAPServer in the Server Controller, which receives SOAP messages from the Client Controller and updates the state of application.

3.2.2 Virtual Desktop Application Sharing & Migration Protocols

When a Client Controller requests to migrate or share a live application, both of them will work precisely as the algorithm indicates. Several steps are taken to complete the migration or sharing as shown in Figure 6 and Figure 7.

The steps of application migration are as follows:

Step1: Client1 → SeverController: {*AppName, MigrationSource, MigrationDes*} The Client 1 sends a migration request to ServerContolller with the *AppName*, *MigrationSource* and *MigrationDes*. In

CyberLiveApp, all connected clients firstly register in ServerController, so that the Client 1 can choose the destination from the client list on LiveApp@Client side.

Step2: SeverController → Client2: {*AppName*, *VMInfo*, *MigrationSource*, *MigrationDes*}. The Client2 gets a migration notification from LiveApp@VM Pool side, and the VM IP address and VNCServer port are enclosed in VMInfo, so the VNCViewer can connect to a remote VNCServer to view the application. Sometimes, the Client 2 does not have a public IP address (e.g., it locates in a local network), so Server Controller cannot initialize a connection to Client 2. In CyberLiveApp, we can change the mode of Step 2 through actively querying to ServerController at a certain interval.

Step3: ClientController@Client2 → VNCViewer@Client2: {*VMInfo*}. The Client2's ClientController creates a new Client2's VNCViewer connection to the remote VNCServer running on the VM and the VNCServer will also close the connection with Client1. Therefore, the presentation streaming of Client1 can be redirected to Client2, and the application view and data are exactly the same due to no change in its running environment.

Step4: Client2 → ServerController: {*MigrationFinal*}. The Client2 sends a *MigrationFinal* message to the ServerController.

Step5: ServerController → Client1: {*MigrationComplement*}. The ServerController updates the meta information of VMPool, and sends a *MigrationComplement* message to Client1.

Step6: ClientController@Client1→VNCViewer@Client1: {*VMInfo*}, Client1's ClientContoller clears the connection record of Client1's VNCViewer, and shows a migration success dialog on Client 1.

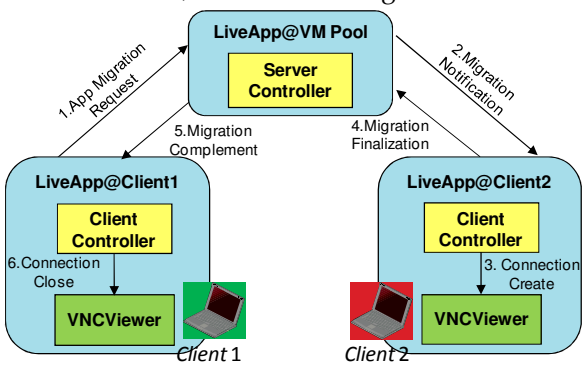


Figure 6: Application migration protocol

The steps of application sharing are as follows:

Step1: Client1→ ServerController: {*AppName*, *SharingSource*, *SharingDes*}. Client 1 sends an application sharing request to ServerContolller with the AppName, SharingSource and SharingDes. In CyberLiveApp, all connected clients firstly register in ServerController, so that Client 1 can choose the destination from the client list on LiveApp@Client side.

Step2: ServerController → Client2: {*AppName*, *SharingSource*, *SharingDes*}. Client2 gets a sharing notification from LiveApp@VM Pool side. Sometimes, Client 2 does not have a public IP address (e.g., it locates in a local network), so Server Controller cannot initialize a connection to the Client 2. In CyberLiveApp, we can change the mode of Step 2 through actively querying the ServerController at an interval.

Step3: Client2 → ServerController: {*CloneRequest*}. Client2 sends a clone request the ServerController after it accepts the sharing message from Client1.

Step4&Step5: ServerController →VMManager:{*VMClone*,*VMInfo*}. ServerController sends a VM clone request to the VMManager, and VMManager clones a VM with `Libvirt` command.

Step6: ServerController → Client2: {*VMInfo*}. The cloned VM IP address and VNCServer port are enclosed in *VMInfo*, so the VNCViewer can connect to a remote VNCServer to view the application.

Step7: ClientController@Client2 → VNCViewer@Client2: {*VMInfo*}. Client2's ClientController creates

a new Client2's VNCViewer connection to the remote VNCServer running on the cloned VM. Therefore, Client2 can view the duplicated application like Client 1, and the initial application view and data are exactly the same.

Step8: Client2 → ServerController: {*SharingFinal*}. Client2 sends a *SharingFinal* message to the ServerController.

Step9: ServerController → Client1: {*SharingComplement*}. The ServerController updates the some meta information of VMPool, and sends a *SharingComplement* message to Client1, and Client1 will unlock its interaction.

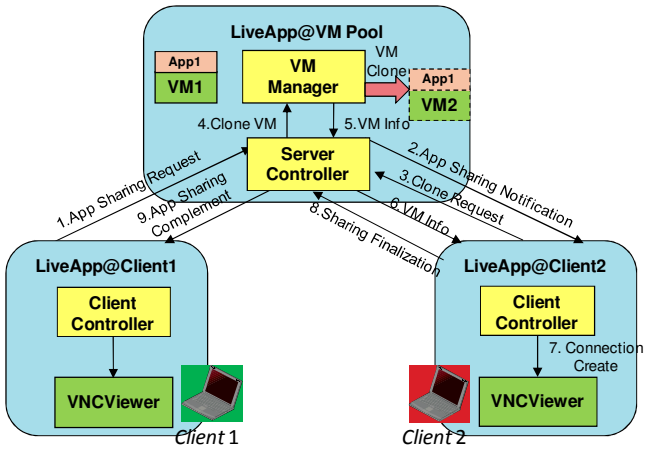


Figure 7: Application sharing protocol among multiple VMs

In the term of application sharing, we intend to provide a completely duplicated application for the two clients. In CyberLiveApp, this requirement is met by cloning the VM in which the application runs. We will discuss the design and implementation of VM cloning in later section. When a Client Controller sends a VM cloning request to the Server Controller, the latter will send the VM id to VM Manager, who will return the new VM IP address after the cloning completes. In this new cloned VM, all the disk and memory data are the same as the original one, thus providing a duplicated application view.

4. System Examples and Simulations

Currently, we implemented a CyberLiveApp prototype system, and verified its viability through some evaluations. The core remote display protocol is based on the RFB implementation of MetaVNC 0.6.6.

4.1 Implementation Experience

4.1.1 Secure Sharing of a Single VM Desktop among Multiple Users

The MetaVNC allocates an unoccupied slot as the `ClientID` for the arrived client request according to the array `vncClient` **m_clientmap*[`MAX_CLIENTS`]. Then, a `vncClient` object will be initialized for this connection, and its attributes are set according to the passed parameters. After that, -MetaVNC calls `vncClient::Init` method to pass the `clientID` instance pointer. Finally, this user will be added into the unauthorized user list of `vncServer` through `m_clientmap[clientid] = client`.

(1) Multi-user Authentication

In the authentication entry function `vncServer::Authenticated (vncClientId clientid)` of MetaVNC, it firstly gets the `vncClient` object from the unauthorized list, and removes it from the `unauth list`. If the user is the first `vncServer` client, a `vncDesktop` object is created by calling `m_desktop->Init(this)`. Next, it allocates a `vncBuffer *buffer = new vncBuffer(m_desktop)`, and sets this buffer by calling `vncClient::SetBuffer` to add this user into `auth list`.

Multi-user authentication feature is implemented in the vncClient.cpp file of MetaVNC project. There are two classes named vncClient and vncClientThread in vncClient.cpp. vncClient is used to send updated image to the client while vncClientTrhead is the server's service thread which is used to receive message sent from the client. Authentication function is implemented in vncClientThread because there are lots of interactions with the client. CyberLiveApp implements the Authentication's function in *BOOL* vncClientThread::InitAuthenticate() at line 209. We added a new security type named rfbSecTypeMultiUser. If the client supports this security type, it will be used; otherwise the default authentication method will be used. In the new authentication process, client returns the challenges and username to the server. When the server receives this username, it checks whether this user exists. If so, the server compares the received password hash with the hash value stored locally. If the two usernames and passwords match, the authentication is successful, then the client and the server will further negotiate to prepare for sending remote desktop. Some source codes are shown in Figure 8.

```
(1)  list[3] = (CARD8)rfbSecTypeMultiUser; //support for Multi-user extension
(2)  if (type == (CARD8)rfbSecTypeMultiUser) {
(3)    vnclog.Print(LL_INTINFO, VNCLOG("enabling Multi-user protocol extensions\n"));
(4)    m_client->m_protocol_tightvnc = TRUE;
(5)    m_client->m_protocol_multiUser = TRUE;
(6)    if (!NegotiateTunneling())
(7)       return FALSE;
(8)       if (!NegotiateAuthenticationForMultiUser(rfbSecTypeMultiUser))
(9)         return FALSE;
(10)         if (secType == rfbSecTypeVncAuth)
(11)           secType = rfbSecTypeMultiUser;
(12)    }
(13) switch (secType){
(14)    case rfbSecTypeNone:
(15)      vnclog.Print(LL_CLIENTS, VNCLOG("no authentication necessary\n"));
(16)      return AuthenticateNone();
(17)    case rfbSecTypeVncAuth:
(18)      vnclog.Print(LL_CLIENTS, VNCLOG("performing VNC authentication\n"));
(19)      return AuthenticateVNC();
(20)    case rfbSecTypeMultiUser:
(21)      vnclog.Print(LL_CLIENTS, VNCLOG("performing VNC multi-user authentication\n"));
(22)      return AuthenticateForMultiUser();
(23)    }
(24)
```

Figure 8: Part of source codes in InitAuthenticate()

(2) Proxy-based Windows Filtering

In the protocol of MetaVNC, the client thread in vncServer is used to receive requests from the clients and send update information to them, and this is one of the most predominant functionalities of vncServer. The vncClient class is responsible for sending data, which is implemented in the function vncClient::SendUpdate. The basic procedure is like this: firstly,vncClient calls SendRFBMsg, then calls the function SendCursorShapeUpdate to send the mouse shape update, and calls function SendCursorPosUpdate to send the mouse position update, and then calls function SendTpDecRgn to reset the region. Finally, vncClient calls SendRectangles to send the related data for updated rectangle.

The existing service threads can only request display update or send display update, which cannot meet the requirements of multiple user accesses. The major work of CyberLiveApp includes:

- Firstly, the access control module reads some configuration information, such as username, password and windows list. The information will be sent to other modules, and we define the functions of BOOL WritePrivateProfileString and DWORD GetPrivateProfileString to

modify the policy file.

- We can only get the window's name from the authentication module, but we ultimately need to get the permitted window area. Therefore, we implement a transformation function in `SendUpdate` of `vncClient` object. Firstly, we get the application path name of the current windows on the desktop, and further get the windows rectangle area based on the application windows.

Two dialogs shown in Figure 9 are extended on metaVNC to set user permissions and windows permissions.

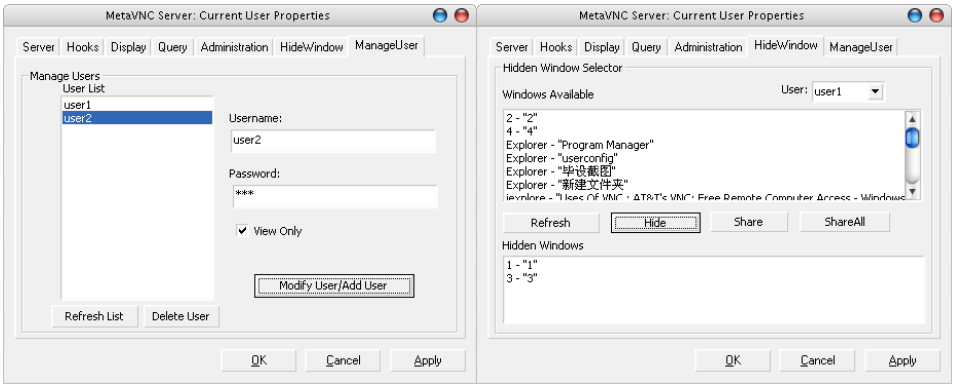


Figure 9: The snapshot of MangerUser and HideWindows Tab

(3) Windows Region Hide

We need to deal with layer and overlap relations of multiple windows, so as to calculate the hidden areas. We implement this in the function `SendUpdate` of the `vncClient` object. Firstly, we use `GetForegroundWindow` function to get the handle of the top window, and we set a flag variable. If the top window is a hidden window, it will be easy. If the top window is not a hidden window, we should get the window's size and remove this rectangle area from the hidden area with `tpRegion.SubtractRect`, so that user can view the top window which he is using.

There is a `vncRegion` object named `toBeSent` in the function of `SendUpdate()` in `vncClient.cpp`. We call `addRect` function in `vncRegion.cpp` to add the updated area into `toBeSent`. Then, we call `Rectangles()` function to get the `rectList` which includes the information of the rectangle area to be sent. Finally we call `SendRectangles(rectList)` function to send the update rectangle.

Figure 10 shows an example in which three clients connecting to the same virtual desktop and they gets different desktop views due to various permissions.

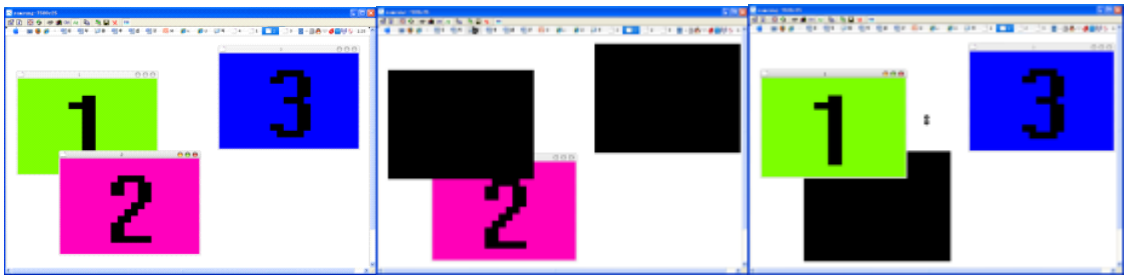


Figure 10: The snapshot of a desktop access by three clients

4.1.2 Application sharing for multiple VMs

In CyberLiveApp, the Client Controller manages multiple `vncViewer` processes, and the inter-process communication is based on Windows `COPYDATA` messages. We have used some Windows

predefined messages such as `VM_CLOSE`, `VM_USER`, `VM_COPY-DATA`, as well as messages `VM_NEWCONNECTION`, `VM_CON_END`, `VM_KILL_ALL`, `VM_SEND_SUBS_HWND` defined by ourselves. For instance, `VM_NEWCONNECTION` is a message to create a new VNC connection, while `VM_CON_END` is a message to close a VNC connection. Since the original `vncViewer` project does not provide a function for closing a specified connection, we add a function `KillConnection(TCHAR *host, int port)` in the file `VNCviewerApp32.cpp`, and it calls function `kill()` to close a specified connection., while on the side of Client Controller, we use `DllImport("user32.dll")` to send a `COPYDATA` message to the `vncViewer` process though `SendMessage(int hWnd, int Msg,int wParam,int lParam)`.

VM Manager manages all VMs in the VM pool, and its main functions are to create, start or clone a specified VM with the support of the `Libvirt` library. `Libvirt` library is a Linux® API realizing the virtualization functions of Linux, it supports a variety of virtual machine monitors including Xen and KVM [12] (we use KVM in CyberLiveApp). `Libvirt` provides a mechanism of saving the memory mirroring of a live VM and starting a VM based on the memory mirroring. When the VM Manager wants to clone a VM, it will firstly saves the memory mirroring and disk mirroring of this VM, which is set to *Suspend* state during the cloning, and restoring it after the cloning. The procedure of VM cloning is shown as follows.

Step1: Copying the original VM disk mirroring file. The user can also configure a copy-on-write (COW) to reduce the image copying time.

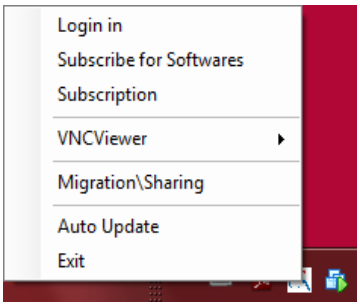
Step2: Saving the VM memory mirroring.

Step3: Restarting the original VM based on the memory mirroring.

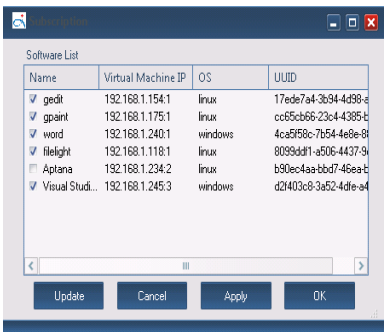
Step4: Modifying the network configuration and the disk mirroring path in the memory mirroring file.

Step5: Restoring a new VM based on the modified memory mirroring and copied disk mirroring file.

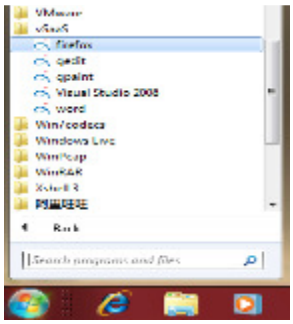
Figure 11 shows some snapshots of CyberLiveApp for application migration and sharing. When we start a Client Controller, it will display an icon on the task bar, and provide an easy way to subscribe for applications. All of these applications are maintained in the VM pool, after subscription, the Client Controller will automatically create shortcuts for these subscribed applications in the Windows Start menu. When a user sponsors to migrate or share a running application with another client, he firstly selects the target client from the online client list registered on the Server Controller. After migration, the application presentation streaming is closed in the sponsor client and redirected to the migration target client. However, after sharing, the presentation streaming will not be closed in the sponsor client, and the sharing target client will get an application streaming from the cloned VM. Therefore, they get the same application and user data at the same time within different clients, but then they can run independently.



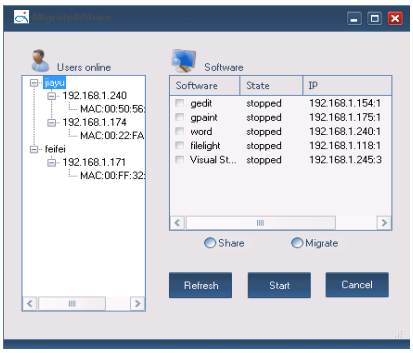
(a) the menu of ClientController



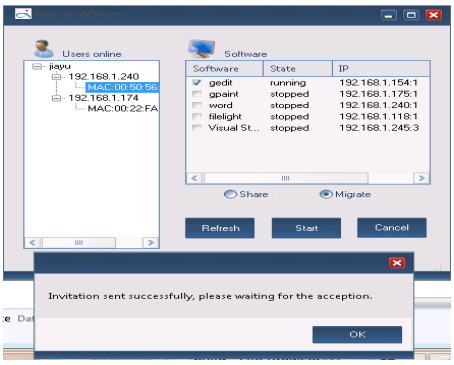
(b) the software list subscription



(c) the subscripted software shortcut on Start menu



(d) the client and terminal list connected the Server Controller



(e) the application migration request

Figure 11: Snapshots of a client controller

4.2 Simulation Results

We next present an experimental study of the CyberLiveApp prototype. We examine the following performance aspects: (a) overhead of multiple users' authentication in CyberLiveApp, (b) network traffic of multiple users' access for a VM, (c) cost of live application migration, and (d) cost of VM clone for application sharing. The simulation scenarios are shown in Figure 12.

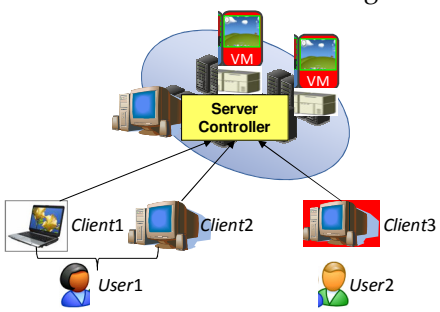


Figure 12: Simulation environment setup

Simulation Environment Setup: The Server Controller, Client2 and Client3 running on machines with Intel(R) Core(TM) 2 CPU 6400 @2.13GHZ, 4GB RAM, Ubuntu 9.04 (Linux 2.6.28) operating system. Client1 is an IBM T61 computer with an Intel(R) Core(TM)2Duo CPU 2.2GHz, 4GB RAM, Windows 7 operating system. These machines are with 100Mbps Internet connection. Client1 and Client2 are owned by User1. Unless specified separately, each experiment is executed five times and the average value is chosen.

4.2.1 Desktop Sharing for a VM

***Simulation* 1:** *In this experiment, we measure the total time of VNCServer update for multi-user access control in CyberLiveApp. We simulate multiple clients on the three computers to connect to the VNCServer on the side of ServerController. The total VNCServer update time is recorded by increasing the number of concurrent clients from 1 to 5.*

The experimental results are shown in Figure 13, which shows that the overall VNCServer update time almost increases linearly with the number of concurrent clients, and multi-user desktop access for a VM in CyberLiveApp is scalable. The bar chart also indicates that the VNCServer update time in CyberLiveApp is higher than in MetaVNC, but the overhead incurred by the multi-user authentication mechanism is less than 20%.

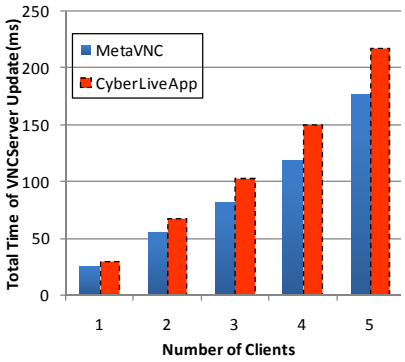


Figure 13: The number of clients v.s. the total update time of VNCServer

***Simulation* 2:** *In this experiment, we want to measure the network traffic impact due to Windows actions. We use a software macro to run two types of windows operations, in which one is moving the windows quickly, and another is maximizing/minimizing windows. We respectively record the total traffic in a period of time on MetaVNC and CyberLiveApp, where the total network traffic is recorded with Wireshark.*

The experimental results are shown in Table 5 and Figure 14.We can see that these two systems almost have the same network traffic under the two different types of operations. Moreover, the frequent updating of a window will dramatically increase the network traffic. In all, the traffic overhead due to multi-user authentication in CyberLiveApp is fewer.

Table 5: Network traffic comparison between MetaVNC and CyberLiveApp

Operation / System	Moving Windows	Maximizing/Minimizing Windows
MetaVNC	15781 KB	1151 KB
CyberLiveApp	15990 KB	1173 KB

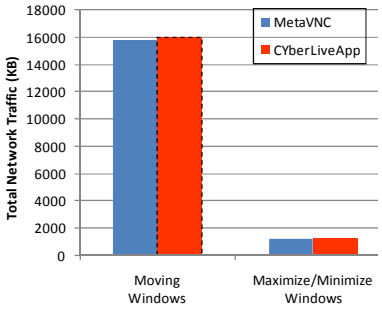


Figure 14: Network traffic comparison between MetaVNC and CyberLiveApp

4.2.2 Application migration and clone for multiple VMs

The two simulations below mainly focus on the time cost for application migration or sharing. We analyzed and compared the time consumed for application's migration or sharing under different configurations.

***Simulation* 3:** *In this experiment, we use two clients to simulate a scenario of live application migration. If a client has a public IP address, the total migration time is similar to the interpretation of Setp*2 *in Figure* 6. *However, in a cloud computing environment, the client generally does not have a public IP address, so the client should use a query mechanism to get the notification of ServerController. We test the application migration time for this scenario under different query intervals.*

The experimental results are shown in Figure 15 and Figure 16. Figure 15 shows the migration time tested for 10 times, and the migration time is less than 200 ms if the query interval time is 100 ms. Figure 16 shows the average migration time for different query intervals tested for 10 times, and the application migration time almost increases linearly with the time of query interval.

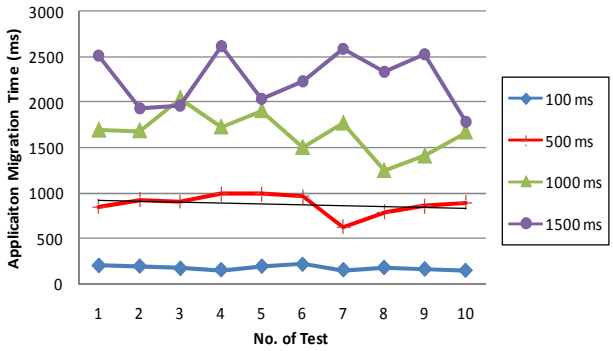


Figure 15: Application migration time for different notification interval time

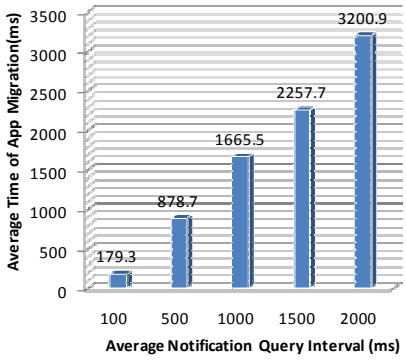


Figure 16: Average application migration time for different notification interval time

***Simulation* 4:** *In this experiment, we also use two clients to simulate a scenario of live application sharing. The environment is the same, and the client does not have a public IP address. Based on our designing, the sharing time will add an extra VM cloning time compared with the application migration scenario. Therefore, we mainly test the VM clone time under different size of VM image.*

When a VM is cloned, VM Manager needs to save the VM's memory mirroring, copy the disk mirroring and restart the VM. Therefore, the clone time consists of three parts:

- T_1: Time for saving the memory mirroring file.
- T_2: Time for copying VM disk COW (copy-on-write) file.
- T_3: Time for restarting the VM.

The selected Windows XP VM memory is 512M in our experiment. After twenty times of repeated tests, we concluded the average time cost for T_1 is 9.308s and T_3 is 0.879s. The time of T_2 is related to the size of VM disk mirroring file. We tested the time cost for copying COW files in size of 64M, 128M, 256M, 512M and 1024M respectively, and the average time cost is shown in Table 7 and Figure 16 below.

Table 6: Average Time Cost for copying disk mirroring file in different sizes

Size	64M	128M	256M	512M	1G	2G
T2b	0.122s	1.611s	3.805s	8.973s	17.391s	42.713s

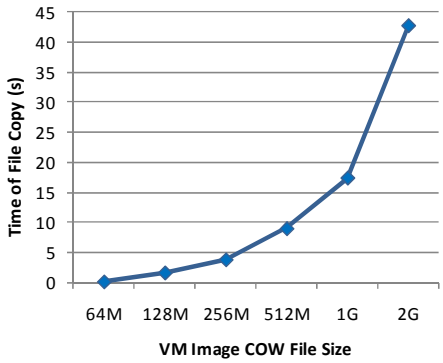


Figure 17: Time cost for copying disk mirroring file in different image sizes

As shown in Figure 17, T_2 is increased linearly with the increasing of size of COW disk mirroring file. In summary, the virtualized application's sharing time is mainly related to the following two factors: notification query time interval in the Client Controller and the size of disk COW image of VM.

5. RELATED WORK

Desktop virtualization technology provides access to cloud computing environments from anywhere in the world, on whatever operating systems. It has become an irresistible trend, and is ranked the second among the ten hottest technologies selected by InfoWorld in 2010 [8]. Forrester predicted: starting from 2010, the desktop virtualization technology will be gradually adopted by large enterprises.

Microsoft has launched the *Client-Hosted Desktop Virtualization* and the *Server-Based Desktop Virtualization*. The latter virtualization technology allows the separation of software execution and presentation by adopting some remote desktop protocols, such as RDP. *Virtual Desktop Infrastructure* (*VDI*), a desktop delivery model developed by Microsoft, allows users to access desktops running in the datacenter. *VMware View* is developed by VMware to achieve the isolation of operating system, applications and user data, which avoids problems brought by the tightly-coupled architecture. Citrix [6] also launched *XenDesktop* to achieve the desktop virtualization. A *FlexCast* technology is used to meet the different demands of desktop environment for different users in an enterprise. These produces break the traditional tightly-coupled software execution environment, and provide flexible desktop access approaches. However, they do not meet the demands for live application sharing and migration or secure sharing of a virtual desktop. Besides, these products also face a compatibility problem under different operation systems.

The reprehensive projects on desktop virtualization include THINC [8][12][13], Citrix XenDesktop [6][15], Microsoft Terminal Service [6] and some VNC systems [5][14]. THINC is a remote display system architecture for high-performance thin-client computing in both LAN and WAN environments. THINC enables higher-level graphics primitives used by applications to be transparently mapped to a few simple low-level primitives that can be implemented easily and efficiently. Citrix provides full VDC (Virtual Desktop Computing) using their ICA protocol in parallel with Ardence image and provisioning manager and desktop server hypervisor. Recently, XenClient extends the benefits of desktop virtualization to mobile users offering improved control for IT with increased flexibility for users. RDP enhancements in Windows Server 2008 and in recent MS client Operating Systems will also address some of the problems identified in relation to video and other graphics-intensive applications over RDP. VNC [4][16][17][18] is based on the PRB protocol which is a simple and powerful remote display protocol. Unlike other remote display protocols such as the X Window System and Citrix's ICA, the VNC protocol is totally independent of operating system, windowing system, and applications. RealVNC [14]proposes different remote display solutions for the client access, the software is executed at remote servers; user client just gets the presentation desktop. This solution only focuses on the

separation of execution and presentation, but does not involve the software deployment and execution related fields. MetaVNC [5] pursues a remote desktop environment that users can control applications on different hosts seamlessly. MetaVNC is a window aware VNC, and it merges windows of multiple remote desktops into a single desktop screen.

Besides, some products and research work emerged to address the software service requirements for mobile equipment in recent years. Microsoft Application Virtualization (App-V, named SoftGrid previously) is a core component of the Microsoft desktop optimization pack for software assurance, it transforms applications into centrally managed virtual services that are never installed and do not conflict with other applications. Progressive Deployment System (PDS) [19] , Yang's work [20] and FVM [21] employee OS-level virtualization technology to reduce the deploying, updating and management labor cost of IT as well as the execution environment isolation. All the virtual software packages are managed at central server sites. When a user wants to use some software, the software package will be delivered to the local machine in a streaming way. MobiDesk is a mobile virtual desktop computing hosting infrastructure, and it transparently virtualizes a user's computing session by abstracting underlying system resources in three key areas: display, operating system, and network. It provides a thin virtualization layer that decouples a user's computing session from any particular end-user device, and moves all application logic to hosting providers.

In summary, there are some desktop virtualization approaches to providing remote access to a cloud computing environment. However, these approaches only focus on displaying the remote desktop, but do not consider flexible collaboration for live application sharing and migration.

6. CONCLUSION

In a cloud computing environment, users can utilise SaaS subscriptions instead of traditional software vendors perpetual-use licenses. We have developed a dynamic prototype system named CyberLiveApp to support application sharing and migration on-demand between multiple clients. CyberLiveApp provides the two key services: secure multi-user sharing service for virtual desktop of a VM and multi-VM application sharing and migration. We have designed a mechanism for tracking windows operation events and hidden window areas are rapidly computed. To filter various windows, a proxy-based filtering mechanism is used to deliver a custom desktop to different users. To support multi-VM applications sharing, we have developed a presentation streaming redirection approach for virtual desktops; an application state consistency algorithm and two protocols are proposed. All of these methods have been implemented in CyberLiveApp based on extended MetaVNC and virtual machine monitor KVM. We have experimentally verified that these approaches are effective and useful. Several extensions to this work are being planned for future development. We are currently developing a virtualization-based software as a service platform, and we are exploring methods to integrate the CyberLiveApp into cloud-based computing environments for flexible collaboration.

Acknowledgment

The authors gratefully acknowledge the anonymous reviewers for their helpful suggestions and comments, and thank Shuang Yang, Yanmin Zhu, Liang Zhong, and Jin Li for their helps on this work. This work is partially supported by Program for New Century Excellent Talents in University 2010, National Nature Science Foundation of China (No. 60903149), China 863 Program (No. 2009AA01Z419), and China 973 Fundamental R&D Program (No. 2011CB302603).

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," 2009

[2] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, Ivona Brandic: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Comp. Syst. 25(6): 599-616 (2009)

[3] M. Turner, D. Budgen, P. BreretonTristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Turning software into a service, IEEE Computer, Vol. 36, No. 10. pp. 38-44. October 2003. http://www.cl.cam.ac.uk/Research/DTG/attarchive/pub/docs/att/tr.98.1.pdf.

[4] Tristan Richardson; Quentin Stafford-Fraser; Kenneth R. Wood; & Andy Hopper. "Virtual Network Computing. IEEE Internet Computing 2 (1): 33–38. January/February 1998. http://www.cl.cam.ac.uk/Research/DTG/attarchive/pub/docs/att/tr.98.1.pdf.

[5] MetaVNC, a part of the Collective at Stanford University, http://metavnc.sourceforge.net/

[6] T. W. Mathers and S. P. Genoway, Windows NT Thin Client Solutions: Implementing Terminal Server and Citrix MetaFrame, Macmillan Technical Publishing, Indianapolis, IN, Nov. 1998.

[7] Jianxin Li, Jinpeng Huai, Chunming Hu, Yanming Zhu. A Secure Collaboration Service for Dynamic Virtual Organizations, Elsevier, Information Sciences, v 180, n 17, p 3086-3107, September 1, 2010.

[8] Info World Test Center staff. InfoWorld's 2010 Technology of the Year Awards[Z]. Info world,2010.

[9] Baratto, R.A., Nieh, J. & Kim, L. Thinc: A remote display architecture for thin-client computing. In Proc. 20th ACM Symposium on Operating Systems Principles (SOSP), 2004.

[10] Ricardo Baratto, Leonard Kim, and Jason Nieh, "THINC: A Virtual Display Architecture for Thin-Client Computing," Proceedings of the Twentieth ACM Symposium on Operating Systems Principles (SOSP 2005), ACM New York, NY, USA, pp. 277-290, October,2005

[11] S. Jae Yang, Jason Nieh, Matt Selsky, and Nikhil Tiwari, "The Performance of Remote Display Mechanisms for Thin-Client Computing", Proceedings of the 2002 USENIX Annual Technical Conference, Monterey, CA, June 10-15, 2002, pp. 131-146.

[12] Albert Lai, Jason Nieh, Bhagyashree Bohra, Vijayarka Nandikonda, Abhishek P. Surana, and Suchita Varshneya, "Improving Web Browsing on Wireless PDAs Using Thin-Client Computing", Proceedings of the 13th International World Wide Web Conference (WWW 2004), New York, NY, May 17-22, 2004, pp. 143-154.

[13] Albert Lai and Jason Nieh, "On the Performance of Wide-Area Thin-Client Computing", ACM Transactions on Computer Systems (TOCS), 24(2), pp. 175-209. May 2006

[14] RealVNC - VNC® remote control software, http://www.realvnc.com/

[15] Citrix Systems - Virtualization, Networking and Cloud. http://www.citrix.com/

[16] Tom Wall, Virtualisation and Thin Client : A Survey of Virtual Desktop environments, Technical Report, Dublin Institute of Technology, 2009, http://arrow.dit.ie/ahfrcart/5/

[17] Taylor, C., Pasquale, J. Improving video performance in VNC under high latency conditions, 2010 International Symposium on Collaborative Technologies and Systems (CTS), pp.26 - 35, 17-21 May 2010

[18] Oren Laadan, Ricardo Baratto, Dan Phung, Shaya Potter, and Jason Nieh, "DejaView: A Personal Virtual Computer Recorder", Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP 2007), Stevenson, WA, pp. 279-292. October 14-17, 2007

[19] Alpern, Bowen, Joshua Auerbach, et al., "PDS: A Virtual Execution Environment for Software Deployment," Proceedings of the First ACM/USENIX International Conference on Virtual Execution Environment, March, 2005.

[20] Yu, Yang, Fanglu Guo, Susanta Nanda, Lapchung Lam and Tzi-cker Chiueh, "A Feather-weight Virtual Machine for Windows Applications," Proceddings of the Second ACM/USENIX Conference on Virtual Execution Environments(VEE'06),June,2006.

[21] C.Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, J. Norris, M. S. Lam, and M. Rosenblum, "Virtual appliances for deploying and maintaining software," Proceedings of Seventeenth USENIX Large Installation System Administration Conference, October, 2003.

[22] Ricardo A. Baratto, Shaya Potter, Gong Su, and Jason Nieh. 2004. MobiDesk: mobile virtual desktop computing. In Proceedings of the 10th annual international conference on Mobile computing and networking (MobiCom '04). ACM,

New York, NY, USA, 1-15.