Muse: A Multimedia Streaming Enabled Remote Interactivity System for Mobile Devices

Weiren Yu Jianxin Li Chunming Hu Liang Zhong
School of Computer Science and Engineering, Beihang University, Beijing, China.
Email: {yuweiren, lijx, hucm, zhongl} @act.buaa.edu.cn

ABSTRACT

Recent years we have witnessed the rapid advent of mobile cloud computing, in which remote software is delivered as a service and accessed by mobile device users over the Internet. However, most existing remote display technologies for high motion application (e.g, movie) have defects in latency and bandwidth. In this paper, we designed an adaptive multimedia streaming enabled remote interactivity system, Muse, to utilize remote resources with reduced display update traffic and response latency. A window-aware updating mechanism is designed as an adaptation scheme, which allows users to focus on the current application in use and also enable them to switch between applications on the fly. Besides, a windowed display encoder using H.264 video codec is integrated into the remote frame buffer protocol to achieve high performance in compression to address the high latency limitation of mobile Internet. Experimental results show that the windowed display Muse mechanism can successfully reduce network traffic, loading time and response latency of remote display and interaction. Our system can achieve in average 22fps of 1024*768 desktop multimedia playbacks with good video quality under 1 Mbit/s of bandwidth limitation.

Categories and Subject Descriptors

C.2.4 [Computer-communication Network]: Distributed Systems – *Distributed applications, Client/Server.*

General Terms

Management, Measurement, Performance, Design.

Keywords

Mobile Internet; Thin-client Computing; Remote Display; Remote Access; Desktop Virtualization,

1. INTRODUCTION

The ubiquity of computation and communication capability nowadays has driven information technology and its applications into a data-centered era. In particular, cloud centers have provided scalable storage and computation services, where software can be completely executed, delivered as a service and accessed by mobile device users over the Internet. At the same time, the growing popularity of mobile Internet devices, such as smart phones, pads, netbooks etc. has made a larger market for information system. Accessing the Internet through Wi-Fi and 3G networks has enabled people to work and play anywhere anytime.

Although a great success, the obstacles for the development of mobile Internet and Internet devices have become obvious. First, devices with different operating system and hardware make it hard for an application to adapt to heterogeneous platforms. Second, resource intensive applications have imposed intrinsic limitations of mobile devices in terms of battery life and processing power.

Therefore, software remote execution technology has become a basic solution for these issues. Offloading applications or computationally expensive tasks to resource-rich cloud servers and receiving processing results, the client device could lower its requirements of local hardware capabilities.

One of the software remote execution technologies is the thin-client computing technology, which allows users to access remote resources at the server through remote interactivity protocol. The protocol sends user input to the server and transmit rendered display to the client. Client device only needs to process display and data transmission, as shown in Figure 1. All the applications are executed in the cloud, decoupling applications and device platforms, simplifying application adaptation.

There are three main advantages to thin-client computing solution. 1. **Light weight**: thin client software could run on modest hardware. 2. **Cross-platform**: the applications run in the cloud and wouldn't need extra adaptation effort. 3. **Low maintenance cost**: The client software is stateless. There is no synchronization needed between client and server hence safe for data and easy to maintain. Only the thin-client software itself needs to be tuned. Legacy programs on mature platforms can be re-used. The thin client remote execution solution could take advantage of cloud environment to extend client device capabilities and simplify software adaptation.

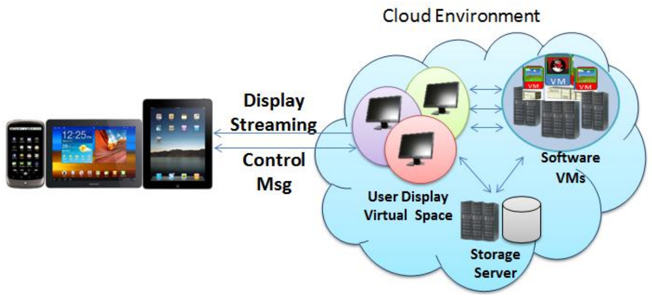


Figure 1. Thin Client System Architecture

Yet mobile thin client computing faces a lot of challenges. First, compared to WIFI or wired network, the 3G wireless network and WAN has a low, unstable bandwidth and uncontrollable latency. Bandwidth and latency as such will severely degrade user experience on interactivity intensive applications. Second, prevalent thin-client systems use a static codec and encoding granularity and lack adaptation to complex desktop scenarios (e.g. desktop with high motion and low motion parts) and varying network conditions in mobile environment. Third, the relatively small screen size of the mobile device will cause disfluent experience of operations.

In this paper, a Multimedia Streaming Enabled remote interactivity system (Muse in short) is designed for mobile devices in a mobile cloud computing convergence. The major contributions of Muse are as follows:

- Based on RFB protocol, we design a protocol that would support windowed display and dynamic region encoding as an adaptation to small device screen size and network conditions. The protocol would allow users to view only the preferred application window and switch quickly to other application windows, reducing unnecessary bandwidth cost incurred by the transmission of all application windows on a desktop.
- We design a windowed display encoder using H.264 video codec to compress screen updates, which will guarantee high fidelity desktop playback in a bandwidth constrained environment. In a high resolution desktop scenario, Muse encoder supports high performance interaction for applications with adequate window size.
- The Muse system prototype is implemented in our iVIC platform [1], and experimental results show that the system could reach video playback at 22fps in 1024*768 resolution.

The rest of the paper is organized as follows. Related works are reviewed in Section 2. The design of Muse is discussed in Section 3. System implementation is shown in Section 4. The system is evaluated in Section 5. A brief conclusion is given in Section 6.

2. RELATED WORK

There are several thin-client systems that have been developed. VNC[2] and THINC[3] etc. are famous thin-client systems proposed in academic research while in industry there are Microsoft Remote Desktop[4], Citrix XenDesktop[5], VMware View, Sun Ray and HP Remote Graphics and so on. Google is creating Chromoting technology for ChromeOS for better user experience to end users. In general, above systems are designed for LAN environment with stable and better network conditions. The bandwidth of current 3G network is still insufficient for these systems. Besides, demand for resource intensive applications like 3D and multimedia processing on mobile devices is growing, putting an even greater demand for technologies that could extend the mobile device's ability in terms of computation.

There are several aspects of thin-client computing technology research: encoding method, transmission optimization and architecture design. The related works are reviewed in these three aspects.

Encoding Method

Encoding method is the most important part in a remote interactivity system. Dealing with unstable network environment, many researches have focused on the optimization of bandwidth adaptation, high latency tolerance and screen content adaptation. THINC and its portable version pTHINC[6] designed a push mode interactivity protocol and achieved a best multimedia playback performance with sufficient bandwidth. Its codec is efficient for UI compression but suffers from compression performance degradation over multimedia content encoding. Huifeng Shen et al [7] designed a compression friendly codec that compresses the text part and the image part of a desktop separately. This could achieve a well balance in compression speed and ratio; it still requires a larger bandwidth than the 3G network could provide for an interactive experience. P. Simons[8][9] developed a hybrid encoding system that switch between H.264 and VNC codec to balance the computation cost of server and client device. The switch is decided by a heuristic algorithm that monitors desktop motion status. This solution is not capable of providing high resolution real time streaming because of the intrinsic performance limitation of video encoder. K. Tan et al [10] improved by dividing a frame into high motion and low motion sections and encode simultaneously with H.264 and VNC and achieved 22.46fps of SIF video playback under 32KByte/s. This solution is possibly capable of high resolution real time streaming if the video is not in full screen mode but the paper didn't provide evaluation on this. Its high/low motion decision algorithm is based on Linux X Window system and cannot be applied universally.

Transmission Optimization

Transmission optimization is also widely applied to reduce system response latency. For display update data, caching [11-13], prefetching[14] methods have been developed. For control data, there are prioritizing[3] and command merging schemes[15]. Vankeirsbilck et al [11] borrowed the idea from video encoding and proposed an optimization method based on history record. The client automatically records the image of a newly open window, sync it with the server and use a residue encoding method for display update. The problem is as more and more history cache is saved, the difference checking step will yield a high cost and the record-residue method is not suitable for display intensive multimedia applications.

Architecture Design

Satyanarayanan[16] identifies the influence of the latency of 3G and WAN network to interactive programs and brings the concept of Cloudlet to enhance user experience. Similar to the idea of CDN(Content Delivery Network, Akamai[17]) , Cloudlet is a resource hosting infrastructure proximate to client devices. It will provide service to clients most adjacent to it thus taking advantage of physical proximity, using low latency network to provide interactive user experience. It provides an insight into a possible solution to network latency though the relationship between cloud and cloudlet is not defined or mentioned.

Besides, to evaluate a thin-client system, there are a few works that provide feasible metrics [18-21]. S. Jae Yang et al [19] defines Video Quality standard and is widely accepted among researchers. Wang et al [18, 23-24] has focused on the research of cloud gaming. First he set up a model that evaluates cloud mobile gaming user experience and mapped them to objective parameters like frame rate, bandwidth, network latency etc. and validated the model. Based on this model, they developed a full frame H.264 encoding system that adjusts to different environment in terms of bitrate, frame rate and response latency. However the metric is game-dependent meaning that for different games the experience model would also be different hence training a new adaptation matrix for every game, let alone other types of applications scenarios like video or office applications.

3. DESIGN OF MUSE

3.1 Overview of Muse

In this section, we introduce the design of Muse.

First, we propose a region encoding protocol based on RFB to allow users to view only the application currently in use and switch between applications to address the limit of device screen size, saving network traffic generated by invisible update beyond the range of the client device screen. This is also an adaptation to encoding granularity to reduce response latency and a support to high performance interaction of applications with adequate window size in a high resolution desktop scenario.

Second, an encoding method using H.264 video codec is introduced for high performance in compression to address the high latency and low bandwidth limitation of mobile Internet. The H.264 video codec is specially optimized for compression of motion pictures and multimedia content hence reducing network traffic greatly.

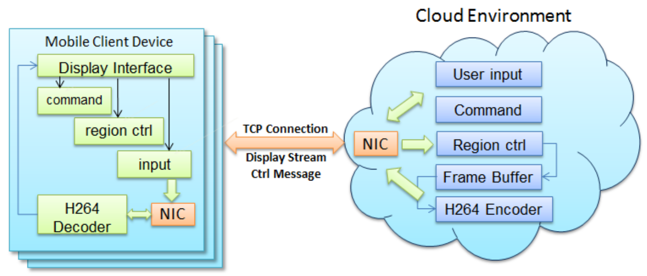


Figure 2. Overall Architecture of Muse System

The overall system architecture is as shown in Figure 2. User would send command, region control messages and input (key, mouse) to the cloud using the display interface. The H.264 Decoder would decode the display stream and update the display. Server takes control messages and provide service accordingly.

3.2 Region Based Update Protocol Extension

In a mobile convergence, the mobile device always has a relatively small screen with lower resolution than the desktop. It would be inconvenient to use desktop applications or to switch between applications. Besides, invisible screen updates are also sent to the client, increasing network traffic. The region-based screen update mode is designed to solve this inconvenience.

In this paper we propose a region encoding protocol based on RFB and implement the region based display scheme. It allows users to select which application window to display and eliminate the display and updates of other applications. It also enables users to switch between applications. Since the performance of H.264 encoder is very sensitive to the input image size, the region based protocol extension could also optimize encoder performance in a high resolution desktop scenario.

The process of the protocol is as follows:

1. Client sends “application switch” request.

2. Server traverses all application windows and sends window handle and name to the client, and window snapshot if possible.

3. The user would select the application needed and send an “application switch” command. Or user could cancel the operation.

4. The server performs the switch, sends screen update and new frame buffer size information.

Following the thin-client principle, all synchronization and state record is performed on the server side. The client won’t be aware of the difference between applications besides its frame buffer size.

To achieve runtime application switching without breaking the connection, we extended the runtime segments of RFB protocol. In practice the server will still support original RFB protocol clients. The extended client will be served by an original server when application switching function is not used.

There are four messages defined for this extension.

Messages from Client to Server:

• Window List Request:

1
0x89

• Window Switch Command:

1	11
0x8A	Wnd Handle

Messages from Server to Client:

• Window List:

1	3	3+35x
0x0D	Length=35x	Wnd Handle | Wnd Name

• Window Coordinates:

1	3	5
0x0D	X Cords.	Y Cords.

“Window list request” message is one byte long; the message type is defined as 137.

When the server gets “window list request” message from the client, it will traverse all open application windows and send back “window list” message. The message is composed of message type, data length and data, one, two, 35*length bytes respectively. The data is composed of unit data of window handle and window name.

The client sends “window switch command” to the server when an application is to be called. The message is 11 bytes long, with one byte message type and 10 bytes of window handle data.

The server will then send back “window coordinates” message as a confirmation. The message contains the coordinates of the top left pixel of the window.

The standard RFB protocol has a message that handles resolution changes. The client will reset its frame buffer size accordingly. When the window is minimized, the window size is recorded in the system as 0*0. When the client receives message containing such a window size, the client will send a “window list request” to the server, preventing the client from crashing.

3.3 Screen Encoder Protocol Using H.264

H.264 is currently one of the best codecs in terms of compression ratio. The goal of the system is to utilize H.264 video encoder in screen compression to optimize system performance in high motion scenarios.

The system is based on RFB protocol with region encoding extension. Standard RFB protocol employs a client-driven update mode. The server captures, compresses the screen and sends the update to the client once it receives a request from the client. The system architecture is as shown in Figure 3.

In this design, we keep the client-pull mode of update and replace the RFB codec with H.264 codec. In implementation we keep the original VNC codecs functional for further research and guarantee that both codecs could be utilized for compression.

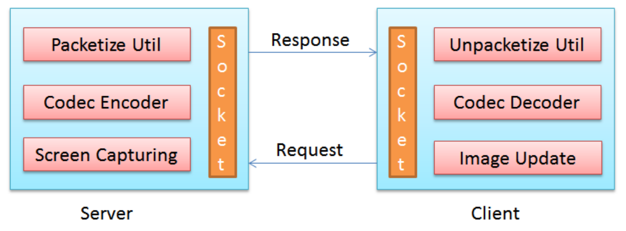


Figure 3. RFB Protocol Update Method

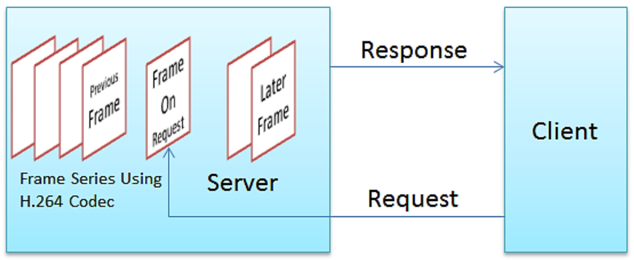


Figure 4. H.264 Application Mode and Video Sequence Definition

In consideration of interactivity performance, client-driven update mode and the inter-prediction feature of H.264, we designed a H.264 screen encoder as such: once the server accepts a screen update request from the client, it will capture the screen and encode it as a frame in the H.264 video sequence, as shown in Figure 4.

In the designed system, every frame is captured when the server accepts a request and the system latency depends on encoding time, network speed and latency. Cache is not added here because the encoder is slower than the decoder hence the time gap between two frames is sufficient for the decoder to process. Besides, encoder speed is less than real time(defined as ⩾25 fps) thus introduces extra system latency.

The system design inherits original “Client Request – Server Response” update mode of VNC system to maintains system consistency in implementation and convenient for subsequent experiments and modifications.

The format of update request doesn’t need to be modified. The client needs to send full screen update request instead of incremental update request because the H.264 encoder takes in fixed size input. The screen update content message needs to be modified to tag H.264 or VNC data. A tag of 2 bytes is inserted after the original message header. The message format is shown as following,

4	6	10	Length+10
RFBUpdate	**Tag**	**Length**	**Data**

4. IMPLEMENTATION OF MUSE

In this section we discuss how Muse system is implemented.

4.1 System Architecture

The detailed architecture of the system is shown in Figure 5. The server and the client communicate through a TCP channel. The server accepts requests from the client and translates the RFB protocol message through event manager. The event manager would send key/mouse input to the selected application, pass region message to region control module and trigger H.264 encoder when a screen update message is received. The region control module would set region parameters according to the client’s request. When H.264 encoder is triggered, it will first read out bitmap from the frame buffer according to the region parameters.

The client would provide functions of region control, command control and keyboard/mouse input. The commands will be passed into the event manager, translated into RFB protocol message and sent to the remote display server. The H.264 decoder is responsible for decoding display streaming data and updating it to client device’s frame buffer to perform display.

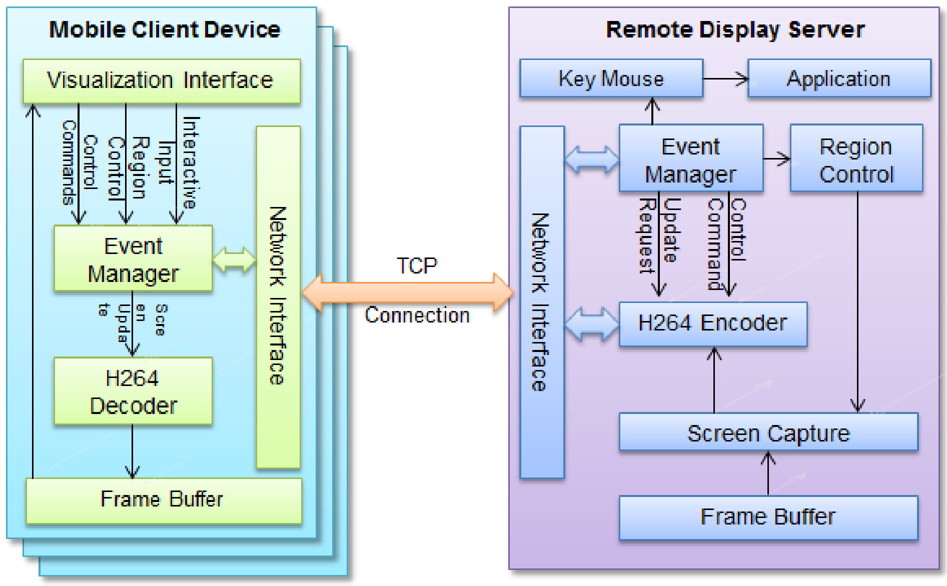


Figure 5. System Architecture

4.2 Server Implementation

The server implementation is based on MetaVNC 0.6.6[25].

(1) Windowed Display and Application Switching

When the server receives a window list message, it will run the implemented `EnumWindows` function to traverse current application windows and return their handles and titles. Then server would encapsulate the defined Window List message and send data to the client.

In the server, the Client Thread and Desktop Thread are owned by two virtual windows to implement Window Message passing. The Client Thread is responsible for interacting with the client device. The desktop thread is responsible for all kinds of functions that involve desktop, like screen capture etc., as shown in Figure 6.

When the client device sends an update request, the Client Thread will process first then send it to Virtual Desktop Window to be processed by the Desktop Thread. The Desktop Thread holds a `vncServer` pointer which has a list of all clients through which updates are passed. Using this model, all clients will receive the same screen update.

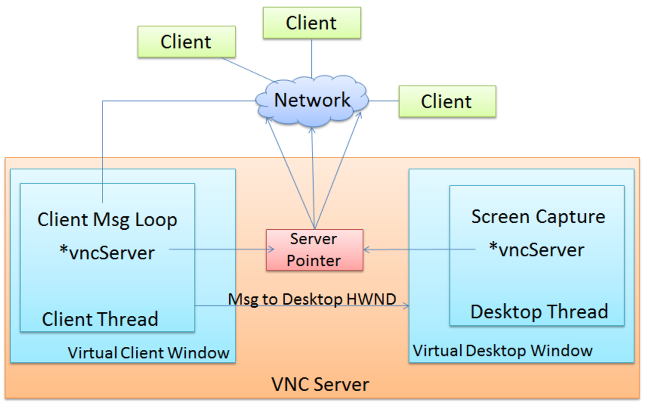


Figure 6. Data Structure of Server

Client thread holds a `vncServer` pointer to get and set current shared screen area information. When the client thread receives a window switch command, it will set screen sharing method to "Windows" opposed to "Desktop", and set the shared region rectangle through `vncServer` pointer.

When the Desktop Thread captures the screen, it will check if the region is the same as the last screen shot. If different, server would send new frame buffer size message. This message is defined by the RFB protocol and used here as a window size update message. Actually, an application window is a "whole desktop" to the user.

We maintain the data structure interface based on `RECT`, which the screen capture function, frame buffer difference checking function and client thread initialization function are all based on.

(2) H.264 Based Encoder

For the H.264 implementation, we use T264 project[26] and implement `T264EncoderUtil` class to encapsulate encoder initialization, RGBA to YUV transcoding and encoding functions. The initialization function is called when the application starts or application switching.

In the VNC system, one update could possibly include cursor shape, cursor postion, `CopyRect` or `LastRect` updates. In the update function of H.264, we added support for these updates to maintain compatibility.

4.3 Client Implementation

(1) Windowed Display and Application Switching

The implementation is based on android VNC Viewer[27].

Implementing window list request is very simple. An item in the menu and its mapping request is added.

Application switching is more complicated. Related data structures are all referenced in the class `VncCanvasActivity`, which handles user interface and its operations, and `VncCanvas`, which encapsulates the canvas view and the RFB message loop. Other data structures include `BitmapData`, which provides functions to draw bitmap and `InputHandler`, which provides different user operation mode.

When the client receives a frame buffer size update message, the processing algorithm is as follows:

1. Modify the frame buffer size held in `RfbProto`.

2. Delete `BitmapData` and reallocate one with new image size.

3. Reallocate `InputHandler`, `Scaling`.

Android Activity's data member cannot be modified in a non-UI thread. The `InputHandler` and `Scaling` couldn't be modified within the RFB message loop, where the algorithm is processed. So we implemented a `windowSwitchListener` class as a callback function to monitor the running status and force the function to be run in UI thread. This technique is also used in server application list display.

In practical use we find that if failed to restore, a minimized window size is 0*0. We use window list request function to handle this exception to prevent crash and maintain continuous experience.

(2) H.264 Based Decoder

The implementation is based on Tight VNC Viewer[28] for Windows.

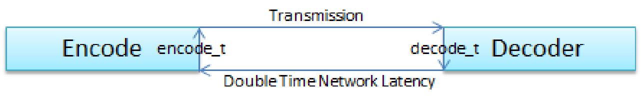


Figure 7. VNC Frame Update Procedure

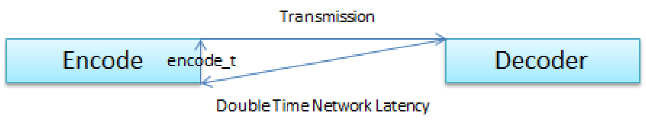


Figure 8. Improved Frame Update Procedure

The original VNC implementation takes t = double_lantency + transmission + decode_t + encode_t to update one frame, as shown in Figure 7. Since it takes H.264 codec more time to encode than to decode, we changed the process into Figure 8 and improved system performance prominently.

To decode H.264 stream, we implement a finite state machine as shown in Figure 9. The state machine starts at receiving data streaming and ends when no data is further fed. We also implemented YUV data extraction from the encapsulated data structure.

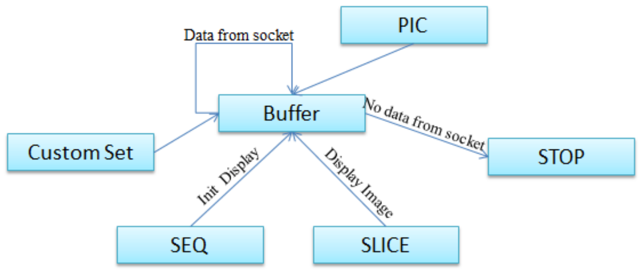


Figure 9. Decoder Finite State Machine

In the original VNC viewer, there is already a set of display methods based on `SetPixel`. There are two main reasons that this is not feasible. First, the efficiency and performance of this function is much lower than requirement. Second, the Bitmap data format used by H.264 decoder is different from the VNC display system. We use `DIBDraw` functions to draw the decoded bitmap, implement a virtual window thread to process window messages and use the original Device Context to embed the display into VNC display region.

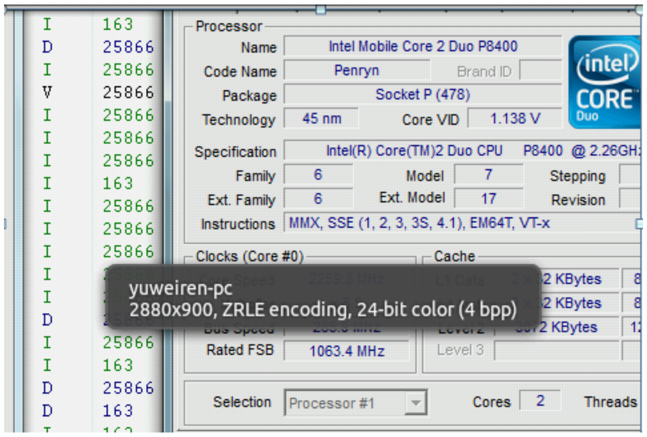
5. EVALUATION OF MUSE

In this section, system performance is evaluated in real application over a wide range of network conditions and prevalent thin-client computing systems are involved for comparison.

5.1 Windowed Display And Application Switching

5.1.1 Prototype

In experiment an Android 2.2 tablet device is used as client and a Windows7 Ultimate x86 PC as server. From Figure 10 we can see that the original server resolution is 2880*900 while the device resolution being 800*480. The whole desktop is being transferred to the client and only part of it is visible.



```
I   163
D   25866
I   25866
V   25866
I   25866
I   25866
I   25866
I   163
I   25866
I   25866
I   25866
I
I
D
I   25866
I   163
D   25866
D   163
```

Processor
Name: Intel Mobile Core 2 Duo P8400
Code Name: Penryn Brand ID
Package: Socket P (478)
Technology: 45 nm Core VID: 1.138 V
Specification: Intel(R) Core(TM)2 Duo CPU P8400 @ 2.26GH
Family: 6 Model: 7 Stepping
Ext. Family: 6 Ext. Model: 17 Revision
Instructions: MMX, SSE (1, 2, 3, 3S, 4.1), EM64T, VT-x
Clocks (Core #0) Cache
2 KBytes
2 KBytes
2 KBytes
Rated FSB: 1063.4 MHz Level 3
Selection: Processor #1 Cores: 2 Threads

yuweiren-pc
2880x900, ZRLE encoding, 24-bit color (4 bpp)

Figure 10. Server Desktop Information

Figure 11 shows the application list menu and the windowed display image. Users can switch applications through the menu.

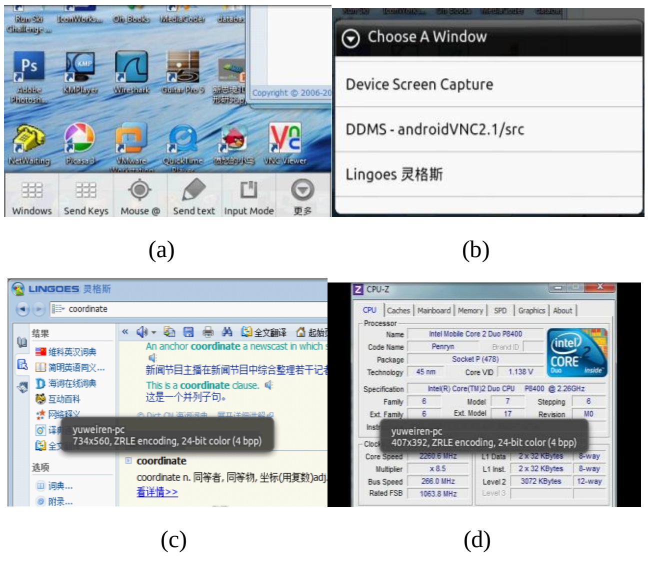


Figure 11. Windowed Display and Application Switching

5.1.2 Bandwidth And Response Time Optimization

For mobile devices, Internet traffic is an expensive resource. To reduce traffic load and boot system performance, we propose the region based encoding scheme. In experiment, we log onto the VNC server using original method and region based method respectively, compare the two methods and record their traffic. Results show that the new method boosts system response performance and reduces Internet traffic.

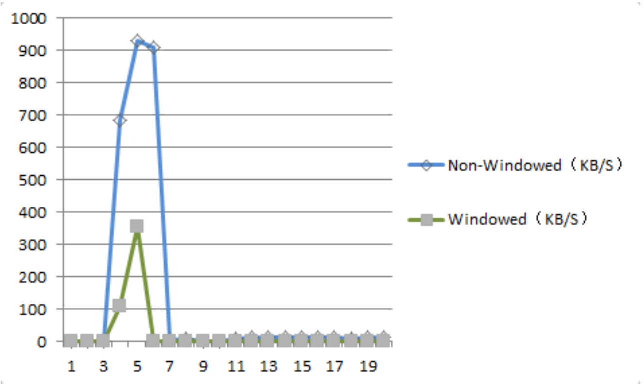


Figure 12. Traffic of MSPaint

Figure 12 and Table 1 shows the traffic record of the first 20 seconds while loading MSPaint software. The original method produced a total traffic of 2676.3KB while the region based method 463.7KB.

The first frame size is reduced so that the traffic and start time is reduced. In slower network the latency reduction could be more prominent. We can also see that there is background traffic in the original method, as shown in Table 1. This is because other applications produce screen changes and updated to the client.

Experiment test bed:

Server: INTEL Q9550 @2.83GHZ, 3GB RAM, Windows 7 Ultimate X86.

Client: HTC G3 CPU@526MHz, 190MB RAM, Android 2.2.

Network: Server in 1Gb/s 1ms LAN, Client in WiFi.

Table 1. Traffic of MSPaint **unit: KB/s**

No	Non-windowed	Windowed	No	Non-windowed	Windowed
1	0.3	0.9	11	7.6	0
2	0.8	0.6	12	14.9	0
3	1	0.1	13	13.6	0
4	684.4	107.4	14	14	0
5	930.7	354.6	15	13.6	0
6	910	0.1	16	14.9	0
7	3.4	0	17	13.7	0
8	10	0	18	10.8	0
9	2	0	19	14.9	0
10	0.9	0	20	14.9	0

5.2 Video Playback Performance

This experiment evaluates the video playback performance and traffic load of the involved systems over a wide range of network conditions. The involved thin-client systems are THINC, VNC, RDP for Windows7 and the system proposed by MSRA. The above systems differ in update mode, screen encoding algorithms and network protocol etc.

5.2.1 Test Bed Setup

(1) Network environment:

We used a 1Gb/s, 1ms latency LAN network to emulate different network conditions. The bandwidth we emulated is 1Gb/s, 10Mb/s and 1Mb/s.

Network emulation is performed on server side. For Windows system, we used `Shunra VE Desktop Client free` version. In Linux we used `netem` command. Wireshark 1.2.8 is used to sniff and record network traffic.

Hardware:

Server: INTEL Q9550 @2.83GHZ, 3GB RAM, Windows 7 Ultimate X86;

Client: INTEL P8400 @2.26GHZ, 2GB RAM, Windows 7 Ultimate X86。

For THINC，we used VMWare 7.0 to run Ubuntu 9.04 system on server hardware.

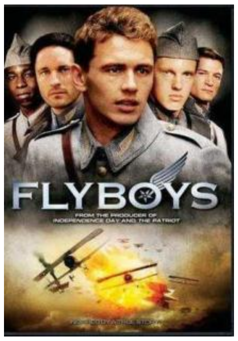


Figure 13. FLYBOYS

(2) Systems and protocols used for comparison:

VNC: Tight VNC with Mirror Driver;

RDP: RDP pre-installed on Windows 7 Ultimate X86;

THINC: As downloaded from [29].

Muse (***ours***):H264 codec as implemented in T264 project.

(3) Test video sequence:

Flyboys fighting scene, 1000 frames in total.

5.2.2 Video Quality Benchmark

We borrowed the video quality evaluation method developed by Yang et al [32]. But Yang's method is applied to screen encoding algorithms that doesn't exploit inter-frame redundancies; we modified the method and proved that it is valid for horizontal comparison.

The original method is based on traffic. The server plays a video sequence slow enough that every frame is completely processed. The traffic for each frame is recorded as a reference traffic load for a 'perfect' playback. Then play the video at normal speed and record the traffic load. The same number of frames is played each time. Define Video Quality（V.Q.） as such:

$$V.Q. = \frac{\dfrac{DataTransfered(24fps)/PlaybackTime(24fps)}{IdealFPS(24fps)}}{\dfrac{DataTransfered(1fps)/PlaybackTime(24fps)}{IdealFPS(1fps)}}$$

Video Quality formula could be transformed:

$$V.Q. = \frac{\dfrac{DataTransfered(24fps)}{Total\ Frame\ Numbers}}{\dfrac{DataTransfered(1fps)}{Total\ Frame\ Numbers}} = \frac{Avg.\ \ Frame\ Datasize(24fps)}{Avg.\ \ Frame\ Datasize(1fps)}$$

Or:

$$V.Q. = \frac{\dfrac{DataTransfered(24fps)}{Total\ Frame\ Numbers}}{\dfrac{DataTransfered(1fps)}{Total\ Frame\ Numbers}} = \frac{DataTransfered(24fps)}{DataTransfered(1fps)}$$

We can see from the above formula that Video Quality is defined as ratio of average frame data size in real-time display and reference playback cases. When the frames are independent to each other, the result is most accurate, because their frame size, seen as weight, is within small differences, which is the case of prevalent thin-client system. H.264 encoder exploits inter-frame redundancies which causes dependencies between frames and the size of frames are vastly diverse. So we modified the encoder formula as such:

$$V.Q. = \begin{cases} reference\ formula & RDP,\ VNC\ etc. \\ \dfrac{frameNums(24fps)}{Total\ Frame\ Nums} & Ours \end{cases}$$

We count the actual frame number received by the client and define the frame number ratio as video quality, which guarantees that each element in the formula has equal weight.

5.2.3 Video Quality Comparison

As shown in Table 2 and Figure 14, different colors stand for experiment result from different network bandwidth. The server screen size is 1024*768.

Table 2. Video Quality Detailed Data

Protocol \ bandwidth	1Mb/s	10Mb/s	1Gb/s
VNC	1.9%	3.8%	3.8%
MSRA Paper	20%	65%	72%
RDP	2.2%	5.9%	88.4%
THINC	1%	20%	100%
Muse (ours)	74.2%	74%	80.3%

From the result we can infer that the system proposed in this paper is the most insensitive for bandwidth changes and has the best performance in low bandwidth environment and maintains a high quality of video playback. In a 3G mobile internet convergence, the network bandwidth is 1Mb/s-2Mb/s. The proposed system could satisfy remote video playback in terms of bandwidth consumption and user experience. We can also see that the performance of the system should be improved in high bandwidth network environment.

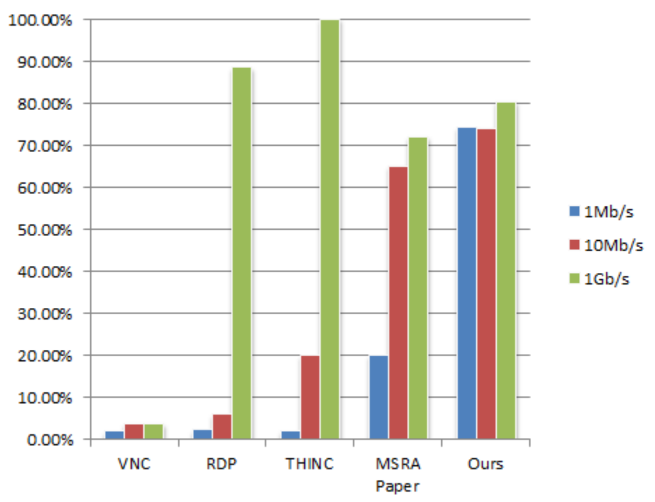


Figure 14. Video Quality Comparison

5.3 Encoder Efficiency

5.3.1 Network Traffic Load Comparison

This experiment is designed to show the detailed bandwidth consumption of each participating system under different network conditions.

Experimental results are shown in Figure 15 and Table 3. The unit of vertical axis is Mb/s. The bandwidth consumption for RDP in the 1Gb/s environment is 37.02Mb/s.

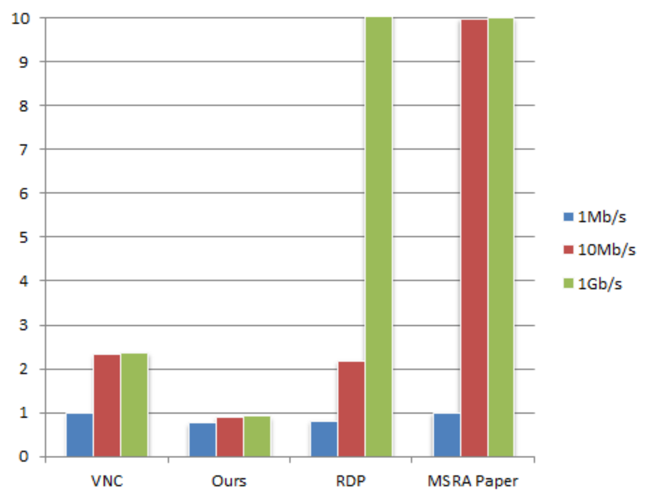


Figure 15. Traffic Load

Table 3. Traffic Usage Detailed Data

Unit: Mb/s

Protocol \ Bandwidth	1Mb/s	10Mb/s	1Gb/s
VNC	0.986	2.33	2.36
Muse (ours)	0.76	0.89	0.904
RDP	0.81	2.18	37.02
MSRA Paper	1	10	10

Results show that the proposed system is insensitive to network bandwidth changes and consumes lower bandwidth than other solutions. The bandwidth utilization rate is relatively low due to TCP protocol. This could be solved by using UDP protocol. We believe that the results could potentially be better. Combining the video quality results we can see the proposed system has a best video playback performance.

The experiments are done under LAN network with 1ms latency. Although we have limited network bandwidth, latency is not taken into account. The "Client-Pull" mode of screen update is very sensitive to network latency. It takes encoding time, data transmission time and round trip latency time for every frame to be fully processed. Latency could be reduced by employing push update mode with traffic control or using physically proximate hosts.

5.3.2 CPU Utilization Rate

This experiment is designed to show the CPU utilization rate of both the server and the client. The result is as shown in Table 4.

Table 4. CPU Utilization

Protocol \ Role	Server	Client	FPS
VNC	12%	Near 100%	0.95
Muse (ours)	80%	60%	20
RDP	10%	invisible	22
THINC	10%	10%	25

The above result is recorded when each system is at their best performance displaying FLYBOYS movie. The VNC client CPU is fully occupied because of implementation. In Tight VNC, the display update function is `SetPixel`, which has low efficiency and would cost a lot computation power.

H.264 codec has high CPU utilization rate when providing high quality video play back. Another experiment that records the CPU utilization rate of the server at different frame rates has been performed. The result is shown in Table 5. We can see that the CPU utilization rate is nearly linear to frame rate.

Table 5. Server CPU Usage at Different Frame Rate

CPU	Frame Rate	CPU	Frame Rate
45%	8	67%	16
54%	10	72%	18
58%	12	80%	20
61%	14	85%	22

5.4 Small Region Update Performance

This experiment compares small region update performance of the proposed system and the VNC system. VNC codec represents the image based encoding method. We selected the sample that could be fully processed by both systems. As shown in Figure 16, the cyan large bold rectangle is a large area slide while the small green bold rectangle indicates a low motion gif animation.

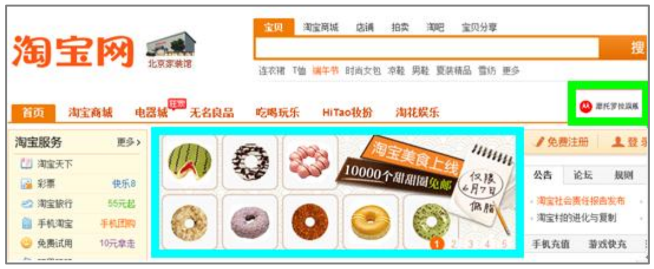


Figure 16. Large Region and Small Region

In experiment, we measure animation playback traffic and frame rate. There are two animations in the display as said above. Both animations are played in Group 1 while only the smaller animation is played in Group 2. The test bed is the same as video quality experiments.

The VNC system uses Tight Encoding method. Both systems use Mirage Driver for better screen capture performance. The result is shown in Table 6.

Table 6. Small Region Update Experimental Result

Group/ Codec	1		2	
	Data(Mb/s)	Speed(fps)	Data(Mb/s)	Speed(fps)
VNC	1.428	26.5	0.241	29.3
H264	0.277	22.6	0.14	22.3

There are three main aspects as follows for comparison,

1. Sensitivity to screen changes

H.264 encoding has high algorithm complexity and insensitive to screen changes with a more stable frame rate. The traffic is vastly different under the two scenarios. VNC tight encoding is very sensitive to desktop changes with a drop of 3fps for average frame rate. This is actually an abrupt experience because there is a large gap between the switching of each slide.

2. Frame rate and response time

From the average frame rate we can know that for small updates VNC system has a very high frame rate. In terms of user experience for small region update, VNC system is better than the proposed system because of the higher frame rate.

3. Bandwidth consumption

Though greatly different, the absolute traffic load of both are not great. This gives us a hint that in compromise we could have a screen codec that combines the advantages of both codecs to adapt to mobile Internet environment.

A video encoding algorithm exploits inter-frame redundancies while a typical desktop scenario has many discontinuous screen changes like switching applications. The video encoder has a very stable performance. Further experiments show that H.264 encoder is not sensitive even to full screen changes but sensitive to frame size. The proposed system could reach an average frame rate of 20 -22fps at 1024*768 while VNC system is very sensitive to large screen updates.

We can conclude that VNC has an advantage over the proposed system for small region updates like mouse hover effects or small region of animation while the video encoder is more suitable for display intensive scenarios. We can also see that none of the prevalent thin-client systems have used video encoder for screen updates, because of its sensitivity to input frame size and high computation cost. The advantage of VNC codec is actually the advantage of all image-based codecs.

6. CONCLUSION

The ubiquity of computation and communication nowadays has driven information technology and its applications into data-centered era. The popularity of mobile Internet devices is growing fast and will bring endless possibilities to the future of computing. Yet application adaptation and resource limitation are two main obstacles for the development of mobile Internet devices.

Thin-client computing is a feasible solution and the Muse system is designed to provide interactive user experience over mobile Internet.

The proposed system has three advantages:

1. Windowed display could optimize the traffic cost, application loading time and interactivity latency and help users focus on one application. The application switching mechanism would allow users to switch between applications and adapt encoding granularity according to user.

2. The proposed H.264 based encoding method has achieved better performance in terms of video play back quality, bandwidth consumption and frame rate in comparison with state-of-the-art thin client systems. It is a significant step towards a mature remote interactivity system over mobile Internet. We also notice the system's relatively high CPU cost.

3. The proposed system could achieve interactivity performance in common desktop scenarios and the experiments show that the video codec has advantage in stability and quality of service. Through experiment we proposed a feasible way to improve system performance.

Experimental results show that the Muse mechanism has successfully reduced network traffic, loading time and response latency of remote display and interaction. The proposed system could play "Angry Bird" webpage edition under 1Mbit/s bandwidth with interactive performance at 1024*768.

We have identified the intrinsic limitation of H.264 encoding, including high resolution encoding performance degradation, high computing cost, slow response to small region update etc. The future work will focus on adaptive encoding and architectural design of a new remote interactivity system with regards to these problems.

7. ACKNOWLEDGMENTS

The authors gratefully acknowledge Dr. Yan Lu in MSRA for his suggestions and helps, and Dr. Mingjing Ai and Lili Zhao for their aid on H.264 codec. This work is also partially supported by National Nature Science Foundation of China (No. 61170294), and China 973 Fundamental R&D Program (No. 2011CB302602) and Program for New Century Excellent Talents in University 2010.

8. REFERENCES

[1] Jianxin Li, Yu Jia, Lu Liu, Tianyu Wo. CyberLiveApp: A secure sharing and migration approach for live virtual desktop applications in a cloud environment. Future Generation Computer Systems. August 2011.

[2] RFB Protocol. http://www.realvnc.com/docs/rfbproto.pdf

[3] R. A. Baratto, L. Kim, J. Nieh, THINC: A Virtual Display Architecture for Thin-Client Computing, Proceedings of 20th ACM Symposium on Operating System Principles(SOSP 2005), Brighton, UK, October 23-26, 2005. Pp. 277-290.

[4] RDP, http://en.wikipedia.org/wiki/Remote_Desktop_Protocol

[5] CITRIX, http://www.citrix.com

[6] Joeng Kim, Ricardo A. Baratto, and Jason Nieh, pTHINC: A Thin-Client Architecure for Mobile Wireless Web. Proceedings of the 15th International World Wide Web Conference(WWW 2006), May 23-26. Edinburgh, Scotland. 2006,143-152.

[7] Huifeng Shen, Yan Lu, Feng Wu, Shipeng Li, "A High-Performance Remote Computing Platform", Proceedings of the 2009 IEEE International Conference on Pervasive Computing and Communications.

[8] L. Deboosere, J. De Wachter, P. Simoens, F. De Turck, B. Dhoedt, and P. Demeester, "Thin Client Computing Solutions in Low- and High-Motion Scenarios," Proc. IEEE Third International Conference on Networking and Service, pp. 38-43, June. 2007.

[9] P. Simoens, P. Praet, B. Vankeirbilck, and J. De Wachter, "Design and Implementation of a Hybrid remote display protocol to optimize multimedia experience on thin client devices," Proc. IEEE Telecommunication Networks and Applications Conference, pp. 22-25, Dec. 2008.

[10] Kheng-Joo Tan, Jia-Wei Gong, Bing-Tsung Wu, Dou-Cheng Chang, Hsin-Yi Li, Yi-Mao Hsiao, Yung-Chung Chen, Shi-Wu Lo, Yuan-Sun Chu, Jiun-In Guo, A remote thin client system for real time multimedia streaming over VNC, ICME 2010, pp.992-997, 2010 IEEE International Conference on Multimedia and Expo, 2010.

[11] B. Vankeirsbilck, P. Simoens, J. De Wachter, L. Deboosere, F. De Turck, B. Dhoedt, P. Demeester, "Bandwidth Optimization for Mobile Thin Client Computing through Graphical Update Caching", ATNAC 2008.

[12] M. Mitrea, P. Simoens, B. Joveski, J. Marshall, A. Taguengayte, F. Preteux, B. Dhoedt, "BiFS-based approaches to remote display for mobile thin clients," in Proceedings of the SPIE - The International Society for Optical Engineering, vol. 7444, 2009 2009, p. 74440F (8pp.).

[13] A. Boukerche, R. W. N. Pazzi, and J. Feng, "An end-to-end virtual environment streaming technique for thin mobile devices over heterogeneous networks," COMPUTER COMMUNICATIONS, vol. 31, no. 11, pp. 2716–2725, JUL 15 2008.

[14] R. W. N. Pazzi, A. Boukerche, and T. Huang, "Implementation, measurement, and analysis of an image-based virtual environment streaming protocol for wireless mobile devices," IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT, vol. 57, no. 9, pp. 1894–1907, SEP 2008.

[15] P. Simoens, B. Vankeirsbilck, L. Deboosere, F. A. Ali, F. De Turck, B. Dhoedt, and P. Demeester, "Upstream bandwidth optimization of thin client protocols through latency-aware adaptive user event buffering," Int. J. Commun. Syst., Accepted for publication [available online], 2010.

[16] Mahadev Satyanarayanan, Paramvir Bahl, Ramon Caceres, Nigel Davies. The case for VM-based Cloudlets in Mobile Computing, IEEE Pervasive Computing, November 2009.

[17] Akamai – http://www.akamai.cn

[18] S. Wang, S. Dey, "Modeling and Characterizing User Experience in a Cloud Server Based Mobile Gaming Approach", GLOBECOM 2009.

[19] S. J. Yang, J. Nieh, M. Selsky, N. Tiwari, "The Performance of Remote Display Mechanisms for Thin-Client Computing", Proceedings of the 2002 USENIX Annual Technical Conference, June 2002.

[20] Niraj Tolia, David G. Andersen, and M. Satyanarayanan, "Quantifying Interactive User Experience on Thin Clients", COMPUTER 2006.

[21] Y. Chang, P.Tseng, K. Chen, C. Lei, "Understanding The Performance of Thin-Client Gaming", CQR Workshop 2011.

[22] Y. Endo, Z. Wang, J. Chen, M. Seltzer, "Using Latency to Evaluate Interactive System Performance", OSDI 1996.

[23] S. Wang, S. Dey, "Rendering Adaptation to Address Communication and Computation Constraints in Cloud Mobile Gaming", GLOBECOM 2010.

[24] S. Wang, S. Dey, "Addressing Response Time and Video Quality in Remote Server Based Internet Mobile Gaming", WCNC, 2010.

[25] MetaVNC, http://metavnc.sourceforge.net/

[26] T264, http://sourceforge.net/projects/t264/

[27] Android VNC Viewer, http://code.google.com/p/android-vnc-viewer/

[28] TightVNC, http://www.tightvnc.com/

[29] THINC Project, http://systems.cs.columbia.edu/projects/thinc/