Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com/locate/jvlc

An original-stream based solution for smoothly replaying high-definition videos in desktop virtualization systems[☆]

Kui Su, Zonghui Wang[*], Xuequan Lu, Wenzhi Chen

College of Computer Science, Zhejiang University, Hangzhou 310027, China

ARTICLE INFO

Article history:
Received 16 September 2014
Accepted 25 September 2014
Available online 14 October 2014

Keywords:
Desktop virtualization
Remote display
High-definition video
GPU-accelerated
Display driver

ABSTRACT

Existing desktop virtualization systems suffer from a very limited performance in replaying high-definition videos remotely: intolerable CPU and bandwidth consumption, high response delay and poor video quality. In this paper, we propose an original-stream based solution to provide good user experience for replaying high-definition videos in desktop virtualization systems without any modification on applications and support most of prevalent high-definition video formats. In our solution, server's video content is not decoded on server but intercepted and delivered to client in its originally encoded state, so that the video content can be easily stored and transported in computer systems with high quality and low bandwidth. The encoded video content is intercepted in server's display driver, which enables HDR to work seamlessly with existing applications. The extremely CPU-intensive video decoding tasks are executed on client by using GPU-accelerated video decoding technology so that CPU can concentrate on other tasks. The experimental results validate our method and show that this proposed approach measurably outperforms state-of-the-art solutions.

1. Introduction

As an emerging trend, virtualization [1–3] has been widely used in cloud computing [4–6] over the past decade. Among those virtualization applications, desktop virtualization has become an important branch [7,8]. In desktop virtualization environment, all applications and operating system code are executed in a server which lies in a remote data center. End user only needs a thin client which handles display, keyboard and mouse combined with adequate processing power for graphical rendering and network communication. The client no longer has to keep user state and communicate with server by using a remote protocol. The protocol allows graphical displays to be virtualized, and transmits user input from the client to the server [9]. Many productive desktop virtualization systems have been developed and applied to various commercial applications since they provide a lot of advantages for IT enterprises such as reducing maintenance and operating costs and improving resource utilization efficiency.

However, existing desktop virtualization systems still suffer from a number of problems before being widely applied: they cannot provide high fidelity display and good interactive experiences for end users, especially on multimedia applications which are commonly used in desktop computing. Current remote display protocols such as Remote Framebuffer protocol (RFB) [10] and Remote Desktop Protocol (RDP) [11] are widely used in desktop virtualization systems [12]. They are mainly designed for low-motion graphical applications, such as text editors whose graphic changes are minor with low frequency. However, those protocols cannot effectively

[☆] This paper has been recommended for acceptance by S.-K. Chang.
[*] Corresponding author.
E-mail addresses: sukuias12@zju.edu.cn (K. Su), zhwang@zju.edu.cn (Z. Wang), brucelu@zju.edu.cn (X. Lu), chenwz@zju.edu.cn (W. Chen).

http://dx.doi.org/10.1016/j.jvlc.2014.09.009
1045-926X/© 2014 Elsevier Ltd. All rights reserved.

support high-motion scenarios such as video playback and real-time interactions. First, because the transport of multimedia data over those protocols is inefficient, requiring high bandwidth to ensure the delivery of all frames to the client in real time. Second, intensive computation for video decoding imposes a heavy burden on server's CPU, which greatly decreases the overall performance of a desktop virtualization system with increasing clients. The problems become even worse when it comes to replaying high definition (HD) video [13] which has a much larger amount of data than standard definition (SD) video. Real-time re-encoding of the video data can definitely save bandwidth but it is computationally expensive, even with modern CPUs, and it causes high response delay, poor video quality and dropped frames that greatly deteriorate user experience.

To address the existing problems, we propose a high-definition remote rendering system named HDR, which is an original-stream based solution for replaying HD videos in desktop virtualization systems. Combined with virtualization technology, HDR provides great user experiences of replaying HD videos without any changes on applications or the window system. The HDR prototype is implemented in Virtualbox [14], an open-sourced virtualization software. The experimental results show that our system could reach almost 100% video quality and full frame rate in full screen on HD video playback in both 100 Mbps and 10 Mbps network environments while classic systems only achieve no more than 20% quality and very low frame rate which is not enough to replay HD videos smoothly. We have tested most of prevalent video formats such as H264 [15], MPEG-2 and VC-1 with generally used resolutions for HD videos such as 720P, 1080i and 1080P. Besides, a number of popular media players have been tested in our experiments. The results show that all the tested video formats and media player applications can be well supported in HDR while some other systems only support specific video formats and media player applications. Additionally, our solution greatly reduces both server and client's CPU usage by using GPU-accelerated video decoding technology [16].

2. Related work

Many productive desktop virtualization systems have been developed and applied to various commercial applications since they provides a lot of advantages for IT enterprises such as reducing maintenance and operating costs and improving resource utilization efficiency. VNC [10] and THINC [17] are famous thin-client systems proposed in academic research while in industry there are Microsoft Remote Desktop [11], Citrix XenDesktop [18], VMware View [19], Sun Ray and HP Remote Graphics and so on. However, most of them cannot provide a satisfactory performance for replaying HD videos.

VNC (Virtual Network Computing) is a popular remote display system with RFB protocol. It uses a virtual driver to maintain local copy of the framebuffer state used to refresh its display and forward user input directly to the server. VNC provides a good performance for office applications but not for video, because the “Client-Pull” mode of screen update in VNC 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 that it is not suitable for frequently updated video replay.

THINC and its portable version pTHINC intercepts low-level video driver commands and adopts a push mode to interact with client. Its codec is efficient for UI compression but suffers from compression performance degradation over multimedia content encoding. As a result, it can achieve a great multimedia playback performance with sufficient bandwidth but not for network environments with low bandwidth.

RDP (Remote Desktop Protocol) is widely used in desktop virtualization products such as Microsoft RDS and VMvare view. For office applications, such as a text editor or a spread-sheet, RDP is highly optimized and the display changes are quite small and have a sufficiently low frequency to cope with. However, with the emergence of multimedia applications, existing remote display protocol cannot reach the high levels of crisp. As a result, RDP provides a poor performance for video replay. In recent years, multimedia applications has been playing a significant role in remote display, therefore, Microsoft is taking much more efforts to optimize RDP and VMvare is trying to find a more efficient solution for its desktop virtualization systems.

For the past few years, popular commercial products Xen Desktop and RemoteFx [20] have devoted much effort to the optimization for video replay. They try to deliver video files to the client before they are decoded on the server, this method will achieve good video quality with low bandwidth, but the main drawback is that they capture the video stream from application layer by taking advantage of Windows media foundation so that it only supports certain video formats and media player applications that use the necessary Windows media framework. Considering multitudinous video formats and media player applications and system scalability, these solutions are inappropriate to be used in desktop virtualization systems.

3. Design of architecture

We propose HDR, an original-stream based method to address existing problems of replaying HD videos in desktop virtualization systems. In the HDR, encoded video content and decoding API calls are intercepted from the display driver of the server and delivered to the client through the network. The client re-executes the API calls to display the video on the screen. The intercepted video content is in its originally encoded state, which means that the video has never been decoded in the server. The proposed method has two main goals: (i) improving user experience of watching HD videos remotely in terms of video quality, fluency, bandwidth and generality and (ii) an obvious reduction on both server's and client's CPU utilization and an exclusive use of server's GPU.

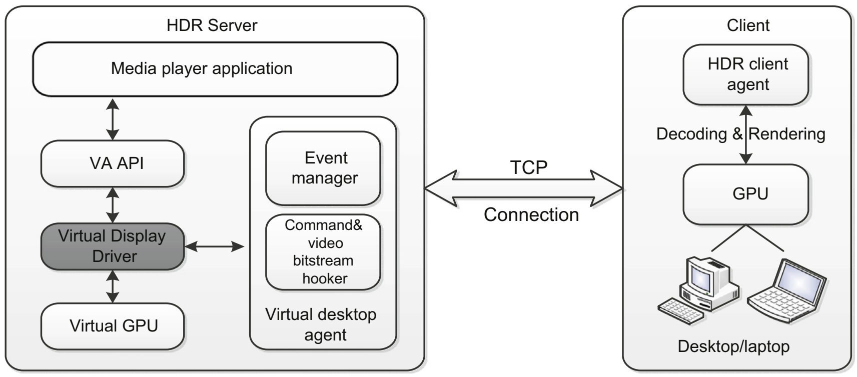

Fig. 1. Overall architecture of HDR.

3.1. *Overview of HDR*

Fig. 1 shows the overview of the HDR. In Fig. 1, the clients are connected to the desktop virtualization server through ethernet or wireless network. User's applications are executed in Guest OS virtualized by the server. A HD video file played by a media player of the server will be eventually displayed on the client's screen. Generally, media players call video acceleration (VA) APIs to leverage GPU-accelerated video decoder for better decoding a HD video. Then the encoded video content is passed to the display driver and finally decoded and rendered by GPU. But in the HDR, the video content and API calls will be intercepted in the display driver of the server and delivered to the client by the virtual desktop agent. The virtual desktop agent composes of commands & video bitstream hooker and event manager. Actually the hooker is implemented by modifying the display driver, which is used to intercept video content and API calls. Then the event manager delivers the intercepted data to the client through the network. The hooker and the event manager communicate with each other by shared memory. On the client side, the client agent re-executes the API calls from the server to display the video on the screen.

In the workflow of HDR, there is no need to compress the video data for transmitting to the client, because the HD video is not decoded on the server and it is still at the originally encoded state (e.g. H264, MPEG2, VC-1) which is very suitable for transmitting. No compression and decompression mean little quality loss, low response latency and low CPU usage. Also the GPU of the server is not used. Moreover, HDR intercepts the video data and commands from the display driver layer which is transparent to applications so that it is able to work seamlessly with existing applications without any modifications.

3.2. *Original-stream based streaming*

In the HDR, we deliver HD video content in its originally encoded state instead of highly compressed decoded video data to the client through the network for two reasons: Firstly, HD videos produced by cameras are always encoded to H264, MPEG-2 or VC-1 formats due to the large amount of data so that they can be easily stored and transported in computer systems. For a desktop virtualization system, it is very convenient to deliver and process HD videos in those formats. Secondly, in a desktop virtualization system, the objective is to enable client users to achieve the same desktop experiences as in the local PCs. In traditional solutions, HD videos are decoded on the server and the decoded video data is highly compressed to reduce bandwidth before transmitted to the client. On the client, the compressed video data has to be decompressed before displayed on the screen. The complicated process on video data greatly damages the quality of HD videos, causes high response delay and consumes more CPU power. Therefore, HDR transmits HD video content in its originally encoded state. In the HDR, high compression is not needed for video data on the server and decompression is also eliminated on the client, HD video can be replayed on the client at its full quality with ideal frame rate. User can achieve the same experience as that in the local PCs with low CPU usage of both the server and client.

3.3. *Driver-based video hooking*

In the previous subsection, we discuss that it is much better to deliver video data in its originally encoded state instead of highly compressed decoded video data. In this subsection, we discuss where to intercept the encoded video data. In practice, video data can be intercepted at different layers: media player application, display driver and framebuffer. In framebuffer, the video has been decoded to pixel data which has to be highly compressed for transmitting to client. In application layer, a video file is in the encoded state but often encapsulated into different formats such as MKV, WMV and MP4, etc. Besides, a variety of media player applications have been developed such as Windows media player, MPlayer, KMPlayer, etc. Meanwhile, various development frameworks for media player applications such as Windows media foundation (WMF) and DirectShow have been proposed. In a virtual desktop, it is necessary to support most of popular media player applications and video formats. If we intercept the video data from the application layer, we have to modify those different applications and the client agent will become much more complicated. Therefore, the best choice is to intercept the video data from the display driver layer which is transparent to applications, and in the display driver, the video is also in its encoded state.

It enables HDR to work seamlessly with existing applications without any modifications.

To intercept the encoded video data from the display driver, we assume that all the media players of the server adopt GPU-accelerated technology to decode and render the video. The GPU-accelerated technology guarantees the encoded video data to be passed to the display driver for GPU decoding. GPU-accelerated video decoding technology is very popular and widely used by most of media player applications because that decoding tasks for HD videos are often computation-intensive which imposes a heavy burden on classic low-end CPUs, and high-end CPU is much more expensive than GPU and consume a lot of power.

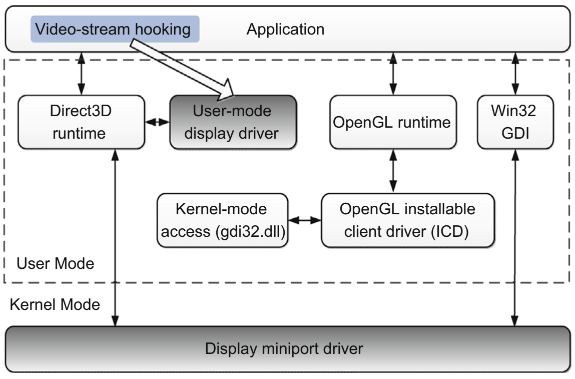

Fig. 2. Video hooking from WDDM.

4. Implementation

We have implemented a prototype server based on Windows 7 system in a virtual environment created by Virtualbox and a client based on Windows 7 system in real physical machines. In the HDR, we have assumed that all the media players of the server adopt GPU-accelerated technology to decode and render HD videos so that the encoded video data can be intercepted from the display driver. In Windows 7 system, DirectX Video Acceleration (DXVA) [21] technology is the most widely used GPU-accelerated technology. It is a Microsoft API specification for the Microsoft Windows and Xbox 360 platforms that allows video decoding to be GPU accelerated. The pipeline of DXVA allows certain CPU-intensive operations such as IDCT, motion compensation and deinterlacing to be offloaded to the GPU. The DXVA is used by software video decoders to define a codec-specific pipeline for GPU-accelerated decoding and rendering of the codec. The pipeline starts at the CPU which is used for parsing the media stream and conversion to DXVA-compatible structures. DXVA specifies a set of operations that can be hardware accelerated and device driver interfaces (DDIs) that the graphic driver can implement to accelerate the operations.

In the HDR, we intercept encoded video content and decoding API calls by modifying the implementation of DXVA DDIs in server's display driver. However, most of the display drivers especially for Windows OS are commercial proprietary closed, to achieve our goal, we have developed a virtual display driver based on Windows Device Driver Model (WDDM) [22] on the server. As shown in Fig. 2, the display driver based on WDDM consists of user-mode driver and display miniport driver. Video content is intercepted in the user-mode driver and delivered to the event manager (shown in Fig. 1) through the miniport driver.

In order to deliver the intercepted data from the driver to the client, the virtual desktop agent takes advantage of shared memory technology for the communication between the event manager and the modified display driver (i.e., the hooker). First, the event manager creates a shared memory region and maps the region to its process space. Then, the driver also maps the same shared region to its process space so that both the event manager process and the driver process can notify each other to read and write the shared memory by holding an event handle. The driver writes the intercepted data to the shared memory and notify the event manager. The event manager reads the data and delivers them to the client through the network.

Generally, media players need to query the ability information of the local GPU before using DXVA to decode HD videos. This is because that DXVA is only available for suitable GPUs. In the HDR, the actual decoding and rendering operations for HD videos are executed by the client's graphic device. To work compatibly with the server, the client should deliver the ability information of its graphic device to the server in advance. Considering various video cards of clients, we adopt a simple and adaptive method to finish this job. Firstly, when a connection between a client and server is built, the client delivers the ability information of its video card to the server and the server will save the information as a local file which shall be valid until the client disconnects with the server. Once a media player begins to query the ability, the information of the file will be submitted to it. This method effectively solves the ability query problem and dynamically adapts diverse client video cards.

5. Performance evaluation

In this section, system performance is evaluated in real applications under different network conditions to demonstrate the effectiveness of HDR. We mainly evaluate the performance of HDR in terms of bandwidth consumption, CPU usage, video quality and frame rate. Several prevalent remote display systems are involved for comparison. They are TightVNC [23], Microsoft Remote Desktop and THINC.

5.1. *Experimental setup*

In our experiments, we use a 100 Mbps LAN network to emulate different network conditions. The bandwidth emulated by the widely used network emulator WANem [24] is 100 Mbps and 10 Mbps. The Server machine has a 2.66 GHz Intel Core i7-920 processor and 8 GB of RAM. Client_1 is a 2.0 GHz Intel Core II laptop with 1 GB of RAM, Client_2 is a thin client with a 1.6 GHz Intel ATOM N270

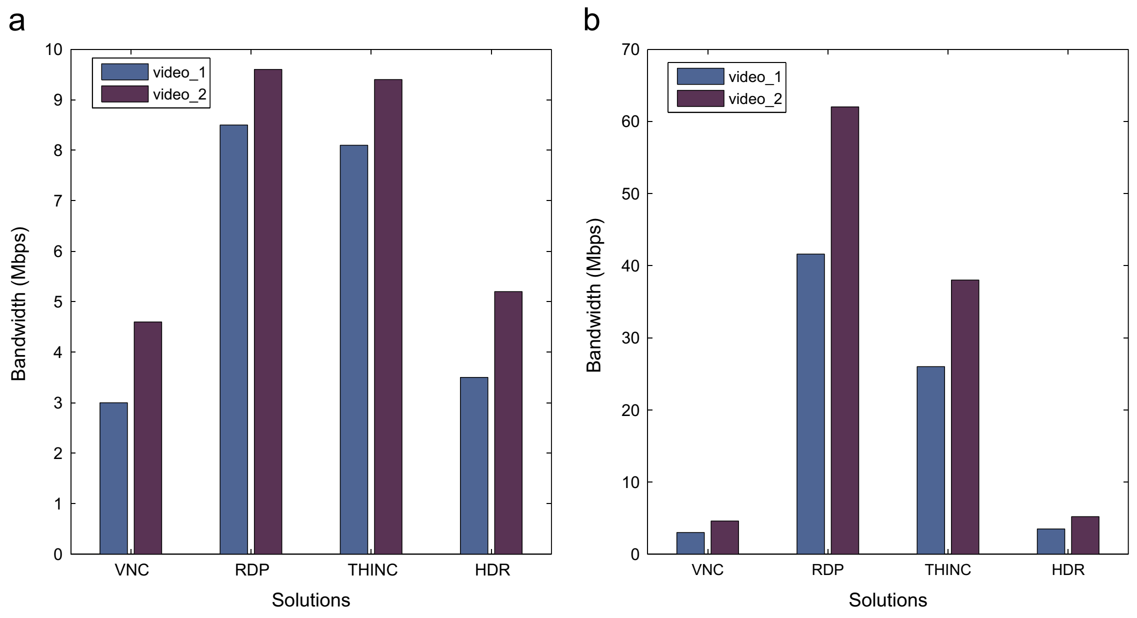

Fig. 3. Bandwidth consumption for different HD videos under different network environments. (a) Bandwidth consumption under 10 Mbps network environment and (b) bandwidth consumption under 100 Mbps network environment.

processor and 512 MB memory. the client_1 runs Windows 7 SP1 system and the client_2 runs Windows Embedded Standard 7 system. For THINC, we use VirtualBox 4.1.16 to run Ubuntu 12.04 system on both server and client hardware. For HDR we use VirtualBox 4.1.16 to run Windows 7 system on server hardware. WANem emulator is also installed on a virtual machine created by VirtualBox 4.1.16 on server hardware. The tested videos include various formats with different resolutions, but in the following, we just give the results of the two H264 HD video clips for the limited space of this paper: (1) Video_1.avi (1280⁎720p, 30 fps, H264 codec, time: 119 s); (2) Video_2.mkv (1920⁎1080p, 30 fps, H264 codec, time: 96 s). The media player used is Windows media player for Windows system, and for Linux we use MPlayer.

5.2. Experimental results

In the following, we introduce our experiments in detail and show the experimental results of all the tested solutions in terms of bandwidth consumption, CPU utilization, video quality and frame rate. Besides, we have also tested that how the FPS in HDR is affected by network delay.

5.2.1. Bandwidth consumption

This experiment is designed to show the detailed bandwidth consumption of each participating system under different network conditions while replaying HD videos of different resolutions. For the 100 Mbps high-bandwidth and 10 Mbps low-bandwidth environments, we compute the average bandwidth consumption during the video replay. We can see the results from Fig. 3(a) and (b), all the solutions consume different bandwidth for HD videos of different resolutions. 1080P video_2 consume much more bandwidth than 720P video_1. HDR and TightVNC consume much less than other solutions under both 10 Mbps and 100 Mbps, but TightVNC does not consume much bandwidth simply because the quality of video is extremely low.

Table 1
CPU utilization for video_1.

Protocol	Role		
	Server (%)	Client_1 (%)	Client_2 (%)
TightVNC	3.7	4.9	21.2
RDP	12.5	11.1	35.4
THINC	10	10.4	38.3
HDR	3.7	2.4	10

Table 2
CPU utilization for video_2.

Protocol	Role		
	Server (%)	Client_1 (%)	Client_2 (%)
TightVNC	4.8	5.6	22.5
RDP	15.1	12.9	46.8
THINC	12	14.5	44
HDR	4.1	3	13.2

5.2.2. CPU consumption

For desktop virtualization systems, CPU consumption is an essential metric for system performance. In this experiment, we have tested the average CPU consumption of video replay on the two clients and the server. Tables 1 and

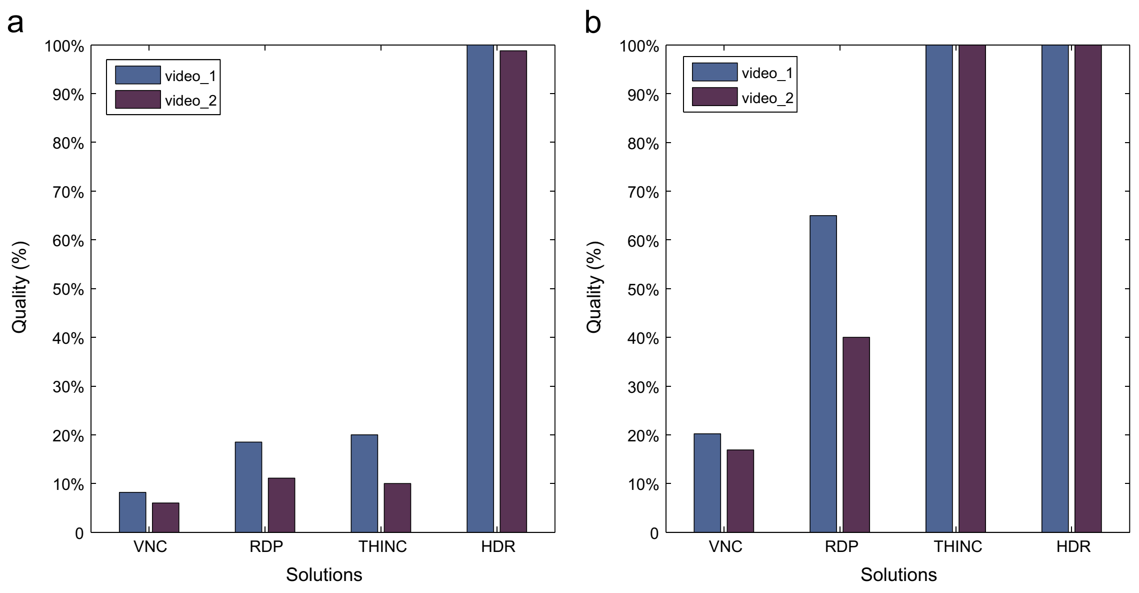

Fig. 4. Video quality for different HD videos under different network environments. (a) Video quality under 10 Mbps network environment and (b) video quality under 100 Mbps network environment.

Table 3
Frame rate under different network environments.

Video	Protocol			
	TightVNC	RDP	THINC	HDR
10 Mb/s (*video*_1)	4.1	6.6	11.2	30
10 Mb/s (*video*_2)	3.7	5.3	6.2	28.8
100 Mb/s (*video*_1)	6.2	12.4	20.8	30
100 Mb/s (*video*_2)	5.8	10.1	16.1	30

2 show the CPU utilization of the participating systems for the two videos, respectively. As shown in the tables, video_2 consume much more CPU than video_1 for the same solution. HDR performs better than RDP and THINC, because HDR delivers video data to the client without high compression which costs much CPU resource and the client's CPU does not need to decompress the video data. Additionally, video decoding on the client is executed by GPU and CPU is just used to process network I/O requests in HDR.

5.2.3. Video quality

Video quality is measured by using slow motion technique [25], which takes both playback delays and frame drops into consideration. Define Video Quality (V.Q.), the video quality is calculated according to formula 1. 100% video quality is the optimal quality, which means all video frames are played at real-time speed. Fig. 4(a) and (b) show the results of video quality for the two videos under 100 Mbps and 10 Mbps network environments, respectively. Among the tested solutions, TightVNC provides the worst quality of video. RDP performs better than TightVNC, it can achieve almost 65% for 720P video_1 but only 40% for 1080P video_2 under 100 Mbps. It becomes much worse while the network bandwidth is reduced to 10 Mbps. THINC does very well under 100 Mbps but it only achieve the video quality no more than 20% under 10 Mbps even for 720P video_1. By contrast, HDR provide 100% quality of the two videos under 100 Mbps, and the only quality loss is for 1080P video_2 under 10 Mbps. Thus, our proposed solution outperforms all the tested solutions in terms of video quality.

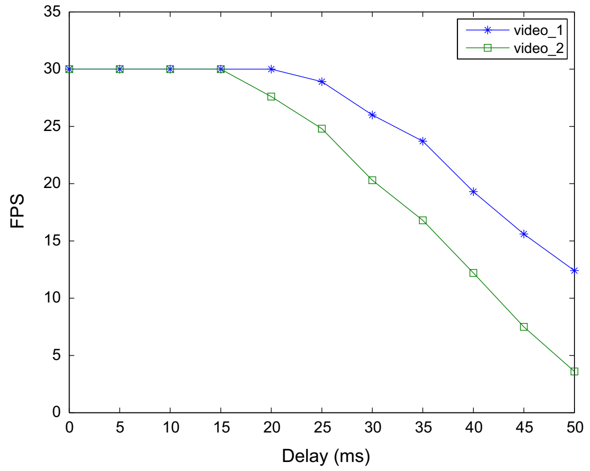

Fig. 5. FPS in HDR under different network delay.

$$V.Q. = \frac{\dfrac{(DataTransfered(30\text{ fps}))/(PlaybackTime(30\text{ fps}))}{IdealFPS(30\text{ fps})}}{\dfrac{(DataTransfered(1\text{ fps}))/(PlaybackTime(30\text{ fps}))}{IdealFPS(1\text{ fps})}} \tag{1}$$

5.2.4. Frames per second (FPS)

In this experiment, the evaluated factor is FPS for video replay on the clients, we compute the average FPS during the video replay. For video_1 and video_2, the full FPS is 30. We have tested these videos in a local PC, all of them can achieve 30 FPS, but in the participating systems, not all of them can make it. As shown in Table 3, HDR can achieve an ideal FPS for both the two videos under 100 Mbps and video_1 under 10 Mbps, and it reaches 28.8 even for 1080P video_2 under 10 Mbps. However, among the other solutions, only THINC achieves a FPS greater than 20 for 720P video_1 under 100 Mbps, all the others have a very low FPS especially for 1080P video_2 under 10 Mbps.

5.2.5. *Network delay*

The above experiments are done under LAN network with ideal latency. Although we have limited network bandwidth, latency is not taken into account. In practice, propagation delay between server and client can also affect system performance. The above experimental results show that HDR performs very well under both 100 Mbps and 10 Mbps environments, now we test the performance under different network delay. We use WANem to emulate a 100 Mbps network with different delay to test the average FPS of video replay. As shown in Fig. 5, when the delay is lower than 20 ms, the FPS is still ideal, but as the delay increases, the FPS decreases rapidly. When the delay is more than 50 ms, the FPS is too low for users to watch HD videos smoothly. Though the above experiment results indicate that HDR is an excellent solution for replaying HD videos in desktop virtualization systems and performs much better than existing solutions, high network latency limits the performance due to the large amount data of HD videos.

6. Conclusion

Desktop virtualization has been widely applied and multimedia applications play a significant role in it. Existing desktop virtualization systems provide good performance for general-purpose applications but still have some challenges for multimedia applications, especially for replaying HD videos which have large amount of data and frequent display updates. We introduce HDR which transmits video content to the clients in its originally encoded state so that video is replayed on the client with ideal FPS, none of quality loss, low CPU cost and network bandwidth. The encoded video content is intercepted from display driver layer on the server, which enables HDR to work seamlessly with unmodified media player applications, do not depend on any multimedia framework such as Windows media framework and support most of prevalent video formats and high definition resolutions. Besides, HDR uses an adaptive method to dynamically adapts various clients with different video cards.

We have measured HDR's performance on HD video replay in terms of bandwidth consumption, CPU usage, video quality and frame rate under 100 Mbps and 10 Mbps network environments and make a comparison with classic remote display systems. From our experimental results, we can see that HDR costs much less resource (CPU, network bandwidth) and provides better user experience (FPS, video quality) than other systems. It shows that HDR is a very favorable method, which far outperforms other state-of-the-art methods. In HDR, client users will achieve the same good experience as that in a local PC. However, there are still some limitations in our prototype system. In the future, we plan to expand HDR to more client devices such as smartphones and PDAs. We will also make more optimizations to reduce the complexity of client agents and lower the processing time on the server to adapt for network environments with high latency.

Acknowledgment

This work is supported by the National Science and Technology Major Project of the Ministry of Science and Technology of China under Grant 2013ZX03003010-002.

References

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield, Xen and the art of virtualization, ACM SIGOPS Oper. Syst. Rev. 37 (5) (2003) 164–177.
[2] Rich Uhlig, Gil Neiger, Dion Rodgers, Amy L. Santoni, Fernando C.M. Martins, Andrew V. Anderson, Steven M. Bennett, Alain Kagi, Felix H. Leung, Larry Smith, Intel virtualization technology, Computer 38 (5) (2005) 48–56.
[3] Irfan Habib, Virtualization with KVM, Linux J. 2008 (166) (2008) 8.
[4] Peter Mell, Tim Grance. The NIST Definition of Cloud Computing. National Institute of Standards and Technology, vol. 53(6), 2009, p. 50.
[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al., A view of cloud computing, Commun. ACM 53 (4) (2010) 50–58.
[6] Ajay Gulati, Ganesha Shanmuganathan, Anne Holler, Irfan Ahmad, Cloud-scale resource management: challenges and techniques, in: Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing, USENIX Association, 2011, pp. 3–3.
[7] Karissa Miller, Mahmoud Pegah, Virtualization: virtually at the desktop, in: Proceedings of the 35th Annual ACM SIGUCCS Fall Conference, ACM, 2007, pp. 255–260.
[8] Xiaofei Liao, Hai Jin, Liting Hu, Haikun Liu, Towards virtualized desktop environment, Concurr. Comput. Pract. Exp. 22 (4) (2010) 419–440.
[9] Jiewei Wu, Jiajun Wang, Zhengwei Qi, Haibing Guan, Sridesk: a streaming based remote interactivity architecture for desktop virtualization system, in: 2013 IEEE Symposium on Computers and Communications (ISCC), IEEE, 2013, pp. 281–286.
[10] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, Andy Hopper, Virtual network computing, IEEE Internet Comput. 2 (1) (1998) 33–38.
[11] Windows Remote Desktop Protocol (RDP), ⟨http://msdn.microsoft.com/en-us/library⟩.
[12] Charles Border. The development and deployment of a multi-user, remote access virtualization system for networking, security, and system administration classes, in: ACM SIGCSE Bulletin, vol. 39, ACM, 2007, pp. 576–580.
[13] Hd-video, ⟨http://en.wikipedia.org/wiki/high-definition_video⟩.
[14] Virtualbox, ⟨http://www.virtualbox.org/wiki/vbox_vs_others⟩.
[15] Thomas Wiegand, Gary J. Sullivan, Gisle Bjontegaard, Ajay Luthra, Overview of the h. 264/AVC video coding standard, IEEE Trans. Circuits Syst. Video Technol. 13 (7) (2003) 560–576.
[16] Guobin Shen, Guang-Ping Gao, Shipeng Li, Heung-Yeung Shum, Ya-Qin Zhang, Accelerate video decoding with generic GPU, IEEE Trans. Circuits Syst. Video Technol. 15 (5) (2005) 685–693.
[17] Ricardo A Baratto, Leonard N Kim, Jason Nieh, THINC: a virtual display architecture for thin-client computing, in: ACM SIGOPS Operating Systems Review, vol. 39, ACM, 2005, pp. 277–290.
[18] Hdx of citrix xendesktop, ⟨http://hdx.citrix.com/hdx⟩.
[19] Vmwareview, ⟨http://www.vmware.com/products/horizon-view⟩.
[20] Remotefx of microsoft, ⟨http://technet.microsoft.com/⟩.

[21] Directx video acceleration (DXVA), 〈http://msdn.microsoft.com/en-us/library/windows/desktop〉.
[22] Windows display driver model (WDDM), 〈http://msdn.microsoft.com/en-us/library〉.
[23] Tightvnc, 〈http://en.wikipedia.org/wiki/tightvnc〉.
[24] Hemanta Kumar Kalitay, Manoj K. Nambiar, Designing WANEM: a wide area network emulator tool, in: 2011 Third International Conference on Communication Systems and Networks (COMSNETS), IEEE, 2011, pp. 1–4.
[25] Nieh Jason, S.Jae Yang, Naomi Novik, Measuring thin-client performance using slow-motion benchmarking, ACM Trans. Comput. Syst. 21 (1) (2003) 87–115.