A High-Fidelity and Low-Interaction-Delay Screen Sharing System

DAN MIAO, University of Science and Technology of China
JINGJING FU, YAN LU, and SHIPENG LI, Microsoft Research Asia
CHANG WEN CHEN, State University of New York at Buffalo

The pervasive computing environment and wide network bandwidth provide users more opportunities to share screen content among multiple devices. In this article, we introduce a remote display system to enable screen sharing among multiple devices with high fidelity and responsive interaction. In the developed system, the frame-level screen content is compressed and transmitted to the client side for screen sharing, and the instant control inputs are simultaneously transmitted to the server side for interaction. Even if the screen responds immediately to the control messages and updates at a high frame rate on the server side, it is difficult to update the screen content with low delay and high frame rate in the client side due to non-negligible time consumption on the whole screen frame compression, transmission, and display buffer updating. To address this critical problem, we propose a layered structure for screen coding and rendering to deliver diverse screen content to the client side with an adaptive frame rate. More specifically, the interaction content with small region screen update is compressed by a blockwise screen codec and rendered at a high frame rate to achieve smooth interaction, while the natural video screen content is compressed by standard video codec and rendered at a regular frame rate for a smooth video display. Experimental results with real applications demonstrate that the proposed system can successfully reduce transmission bandwidth cost and interaction delay during screen sharing. Especially for user interaction in small regions, the proposed system can achieve a higher frame rate than most previous counterparts.

CCS Concepts: • **Information systems → Multimedia streaming**

Additional Key Words and Phrases: Interaction delay, video quality, bandwidth consumption, layered structure, screen coding, screen rendering

ACM Reference Format:
Dan Miao, Jingjing Fu, Yan Lu, Shipeng Li, and Chang Wen Chen. 2016. A high-fidelity and low-interaction-delay screen sharing system. ACM Trans. Multimedia Comput. Commun. Appl. 12, 3, Article 44 (May 2016), 23 pages.
DOI: http://dx.doi.org/10.1145/2897395

1. INTRODUCTION

The ubiquity of current computation and communication capabilities provides opportunities for users to share computing and storage resources among multiple computing devices through network connection. One general way to access or control the resource remotely is screen sharing, that is, capturing and sending the screens from remote devices over networks to local devices in real time [Lu et al. 2011]. A variety of screen

This work was partially done while the author was with Microsoft Research Asia as a research intern.
Authors' addresses: D. Miao, Electronic Engineering Department, University of Science and Technology of China, Hefei, Anhui, China; email: miaodan@mail.ustc.edu.cn; J. Fu, Y. Lu, and S. Li, Media Computing Group, Microsoft Research Asia, Beijing, China; emails: {jifu, yanlu, spli}@microsoft.com; C. W. Chen, Computer Science Department, State University of New York at Buffalo, Buffalo, New York, US; email: chencw@buffalo.edu.

sharing applications have been developed to facilitate remote resource access. For example, remote desktop systems, such as Remote Desktop Protocol (RDP) [Microsoft 2016a] and Splashtop [2016], support desktop sharing among devices. Screencast systems, such as Airplay [Apple 2016] and Chromecast [2016], provide the convenience of playing multimedia content among a laptop, a tablet, and a TV over home networks.

A screen sharing system is usually developed based on the thin-client computing architecture, in which the client displays the received screen updates and allows users to interact with screen content by sending the control commands, while the server intercepts, encodes, and transmits screen updates to the client. In order to provide high-fidelity display and responsive interaction for users as if they were using local machines, some important factors should be considered in a screen sharing system:

—*Bandwidth consumption*: Screen sharing systems are expected to achieve acceptable performance in bandwidth-constrained network conditions. However, multiple screen content with multimedia applications and complicated graphical interfaces makes effective screen compression and transmission challenging.
—*Smooth interaction user experience*: During screen sharing, smooth interaction should guarantee that the response screen content is updated within tolerant latency after the interface operation.
—*Cross-platform adaptation*: More and more consumer electronic devices can easily access the Internet. These devices may vary in access bandwidths and operating systems. Therefore, cross-platform adaptation becomes essential for a screen sharing system that can be widely deployed.

The performance of a screen sharing system is highly relevant to the mechanism defining the rules of representing and delivering screen update from server to client. A traditional remote desktop system, X-system [Scheifler and Gettys 1986], utilizes graphical primitives to represent screen update. This method is efficient in representing the display of graphical interface. However, it suffers from performance degradation when representing display-intensive multimedia applications, such as video playback. In addition, the interpretation of graphical commands heavily depends on the operating systems. It is difficult to develop and support cross-platform screen sharing, as servers and clients may be built up on different operating systems with different rendering mechanisms. Some thin-client system, such as Virtual Network Computing (VNC) [Richardson et al. 1998], utilize raw pixel primitives to represent the screen update and support cross-platform implementation. This kind of system lacks efficient compression mechanisms for multimedia application. Recently, a frame-based representation model for screen sharing has attracted much attention and many emerging screen casting and cloud gaming systems are developed based on this model. In this model, the whole screen is captured as a sequence of image frames, which are compressed by video codec and transmitted to the client side. A frame-based model is friendly for cross-platform implementation, since screen update is represented based on screen capture and video compression, and a standard video codec can be easily employed in distinct systems. For example, an open-source cloud gaming system, GamingAnywhere [Huang et al. 2013], which employs x264 as video encoder, is available on Windows, Linux, and OS X. For the systems built up based on the frame-based model, the integrated screen compression scheme is crucial to the system performance. In most of these systems, traditional video codecs are employed for screen compression, which can handle the natural video content well. However, the compression performance of the video codec is not adequate for the text, like text, and graphic content, which are more challenging with respect to the requirements on the high visual quality of screen display and instant user interaction. In this article, we focus on the screen compression efficiency and complexity issues in the screen sharing system with a frame-based representation model.

The main challenges can be summarized as follows: First, the screen is composed of multiple types of content, including text, graphical interface, and natural video. A traditional standard video codec can handle natural video compression well but coding efficiency for typical graphic/text content is inadequate. Thus, the screen coding scheme for the graphic/text compression should be designed and integrated into the system. Second, the coding frame rate determines the screen update frequency in the client device. The requirement of the screen update frequency for smooth screen sharing is various for distinct content. For the multimedia application, such as video playback, the regular frame rate of 30 frames per second (fps) is acceptable for smooth display. While for the user interface (UI), in order to achieve smooth interaction, the screen is expected to be updated at a higher frame rate. For example, a measurement study [Hsu et al. 2015] reveals that a frame rate higher than 30fps (e.g., up to 60fps) can provide a better user experience during interaction with remote display. However, the higher frame rate is challenging for the screen compression scheme. Finally, during screen rendering, the I/O processing, that is, the screen content copying from the decoding buffer to the rendering buffer, is also critical to the system performance. This is because the frequent buffer copy will lead to heavy processor load and increase the system-level delay.

Considering the above issues, we propose a layered screen sharing system based on a frame-based model. In the proposed system, a layered screen coding scheme is designed for efficient screen compression and to further facilitate flexible transmission and high-frame-rate rendering. For layered screen video coding, we have utilized this framework to support the quality scalability [Miao et al. 2013] and high-frame-rate coding [Miao et al. 2014], respectively. In this article, we jointly consider the high frame rate and visual quality enhancement issues in a real system. In the enhanced coding framework, distinct content is partitioned into the appropriate encoders and compressed with an adaptive frame rate. More specifically, the screen frame with natural video content is compressed by a standard video codec at a regular frame rate that is acceptable for smooth video display and the screen frame with interaction content is compressed by a blockwise screen codec at a high frame rate to facilitate a smooth interaction experience. To further improve visual quality, the text/graphic content fed into the video codec is selected and enhanced in the screen codec.

Based on the layered coding structure, we employ several mechanisms at the system level to reduce the bandwidth cost and interaction latency. On the server side, a timer-driven screen update mechanism with multiple thread processing is adopted to schedule the capture, compression, and transmission in an adaptive frame rate based on available bandwidth resources. On the client side, with the assistance of the layered structure, we introduce a fast rendering scheme, in which the partial region update is supported to save the I/O buffer copy cost during screen interaction. To the best of our knowledge, there is little published work that considers rendering optimization in a screen sharing system.

The main contributions of this article can be summarized as follows:

—Based on the frame-based model, the screen update is represented as the whole frame plus small region blocks for the distinct content and a layered coding scheme is proposed to combine the advantages of video and screen codecs for high efficiency coding with high frame rate during interaction.
—Based on the layered coding scheme, an adaptive frame capture mechanism combining with a flexible transmission scheme is proposed on the server side and a fast rendering scheme is adopted on the client side to improve system performance.
—The developed system can leverage existing coding resources and facilitate cross-platform implementation.

The remainder of the article is organized as follows. Section 2 reviews related work. Section 3 presents the architecture of the proposed system. Section 4 introduces the layered structure design on coding and rendering. Following that, Section 5 provides a detailed description on the layered coding implementation. The experimental results of the proposed system are presented in Section 6. Section 7 concludes the article.

2. RELATED WORK

Screen sharing has attracted much attention from both academia and industry due to its rich usage scenarios, and many relevant systems have been developed.

2.1. Screen Sharing System

Existing screen sharing systems span a range of differences in several aspects, such as the policies for updating the client display, algorithms for compressing screen updates, supported display color depth, and transport protocol used. Among these factors, a method to represent and compress the screen update plays an important role in a screen sharing system.

2.1.1. Region Update Model. Some thin-client systems, such as VNC [Richardson et al. 1998] and THINC [Baratto et al. 2005], represent screen updates with 2D primitives and compress the update with run-length encoding (RLE) or Lempel-Ziv Welch (LZW). Such a mechanism allows the server to simply forward graphics to be updated into the compressors and discard other stable regions directly. On the client side, the screen presenter renders the decoded graphics and overlays rectangular areas of pixels in destined regions. Although this method is efficient in representing the display of graphical interface, it suffers from performance degradation, especially when representing display-intensive multimedia applications, such as video playback. This is because the temporal correlation in natural video cannot be exploited by the above coding schemes. Recently, a screen sharing system named DisplayCast [Chandra et al. 2012] has been proposed for the intranet environment. In this scheme, the screen update is represented as a pixmap, which is captured with distinct frequency for different screen content and compressed by a modified Zlib lossless coding scheme. Although the screen update rate can be significantly improved for interaction scenarios, the compression efficiency is still inadequate for video display.

2.1.2. Frame-Based Model. With the increase of multimedia applications appearing in screen sharing scenarios, many screencast systems have been developed in the industry, such as Miracast [Wi-Fi Alliance 2016], Airplay [Apple 2016] and Chromecast [2016]. Screencast treats the screen as a fixed resolution video while capturing and encoding them at the low level of the protocol stack and usually employs hardware video codec for screen compression. A video codec integrated into the system can support efficient compression for natural video content and the implementation through hardware can also reduce the CPU usage at the expense of slightly higher GPU usage. In addition to multimedia applications, screencast systems are also developed and utilized in many other scenarios, such as NCast [2016], which is designed to support screen streaming in the presentation scenario. In this system, rich screen content, such as natural video, images, and slides, are compressed by H.264 video encoder and transmitted to users. When a screencast system is used for sharing multiple types of screen content, a standard video codec can achieve efficient coding on natural video content but may not be suitable to other text and graphic content.

Cloud gaming can also be considered a remote screen sharing scenario in which the computer games run on cloud servers and users interact with games via thin clients. Providing a good user experience in cloud gaming is not an easy task, because users expect both high-quality video and low response delay. Generic desktop systems, such as

LogMeIn [2016] and TeamViewer [2016], can support cloud gaming. However, a measurement study [Chang et al. 2011] reveals that these thin clients achieve low frame rates, on average 13.5fps, which lead to sluggish game plays. A better user experience is possible with thin clients specifically designed for cloud gaming, for example, OnLive [2016] and StreamMyGame [2016]. Nevertheless, another measurement study [Chen et al. 2011] demonstrates that these cloud gaming systems also suffer from non-trivial response time. In addition to cloud gaming development, many research works have been attempted in this area. For example, in Wu et al. [2015b], the authors guarantee the delivery quality of high-frame-rate video in mobile cloud gaming scenarios based on a novel transmission scheduling framework [Wu et al. 2015a]. Huang et al. [2013] propose the first open-source cloud gaming system, GamingAnywhere, based on a video streaming mechanism, and focus on the modularized system design and optimization. This system can achieve good performance of video streaming during screen sharing. But high frame rate is not supported very well during interaction. To better evaluate the performance of cloud gaming solutions, some measurement schemes [Choy et al. 2012; Jarschel et al. 2011; Chen et al. 2014] are also proposed.

Shen et al. [2009] develop a screen sharing platform based on a pure compression model, in which the whole screen frame is compressed by a blockwise screen video codec and transmitted to a thin client. In this codec, the screen frame is partitioned as text blocks and image blocks. The text block is compressed in pixel domain while the image block is compressed by a JPEG encoder. This platform performs well on pure screen content sharing, but the bandwidth cost increases dramatically when natural video with high motion is embedded in the screen.

2.1.3. Hybrid Protocol. Several screen sharing platforms have been developed based on the hybrid remote desktop protocol, in which the Remote Framebuffer (RFB) protocol employed in VNC is leveraged with a video streaming mode to transport the rendered images of multimedia applications. Deboosere et al. [2007] and Simoens et al. [2008] develop a hybrid encoding scheme that switches between H.264 and VNC codecs. The switch is decided by a heuristic algorithm that monitors desktop motion status. The frame with high motion is fed into video encoder while the frame with low motion is relayed through the VNC-RFB protocol. In this scheme, the motion detection algorithm plays an important role. The more precise the motion detection algorithm, the lower the necessary bandwidth between clients and servers. As the authors claim in the article, the content partition scheme should be refined to deal with the frame mixed with video and still content. Tan et al. [2010] improve this algorithm by splitting each remote frame to be parts of low motion and high motion. However, its high/low motion decision algorithm is based on the Linux X Window system and cannot be applied universally. Moreover, this kind of system suffers from inadequate performance. For example, the system in Tan et al. [2010] achieves 22.46fps of video playback with low resolution in a low-bandwidth condition and drops to less than 10fps through a VNC protocol if the partition failed. Note that the compression scheme in the proposed system can be also considered a combined scheme with a video codec and blockwise screen codec. Compared with the above schemes, the content partition scheme based on a fast global motion detection algorithm is more efficient and robust and the screen codec in the proposed scheme is also more efficient than VNC.

2.2. Screen Video Coding Scheme

Considering that the screen compression scheme plays an important role in the screen sharing system, several works have contributed to screen compression. To improve coding efficiency on the distinct content in screen, some layer-based coding frameworks are proposed, in which the distinct content is segmented at the pixel level or block level and compressed by different algorithms. For example, the mixed raster content model (MRC) segments the screen into a foreground layer with text and graphics, a background layer with natural images and white spaces, and a mask layer to indicate the pixels of each layer. Based on the MRC, the DjVu scheme [Haffner et al. 1998] compresses the two layers with a wavelet-based algorithm (IW44) and compresses the mask layer with JBIG2. It can achieve a high compression ratio but suffers from high computational complexity due to the overhead on pixel segmentation. The distinct content can also be classified at the block level. Pan et al. [2013] partition the screen frame into image and text blocks. JPEG is employed for image block compression. Text blocks are entropy encoded after quantization in pixel domain. This scheme performs well on pure screen content, but its performance decreases dramatically when natural video with high motion is embedded.

In order to leverage a standard video codec, some new coding methods have been designed for screen compression. For example, Ding et al. [2007] introduce a new intra mode in a pixel domain based on H.264/AVC to better exploit spatial correlation in the text region. Although rate distortion performance is improved, the compatibility with a standard codec is destroyed due to the modification on coding syntax. Zhang et al. [2013] propose an arbitrary-sized motion detection algorithm instead of traditional block-based motion estimation for large motion regions in screen video. This scheme can reduce both encoding time and bit rate in terms of the implementation in H.264. But the spatial correlation in text/graphic content is not exploited.

Recently, some screen coding schemes have been proposed that aim to guarantee coding compatibility with a standard video codec, so existing coding resources can be utilized without modification. For instance, Wang et al. [2012] propose a layered hybrid screen video codec. In their work, the video region is segmented and padded as a whole frame, which is compressed by a video codec. The rest of the content is encoded by a screen coding scheme. Although high compatibility with the standard is achieved, it is difficult to reconstruct the whole screen from the video codec with only the video region available. Moreover, a high coding frame rate for a smooth interaction is not considered in this scheme. In our previous works [Miao et al. 2013, 2014], two layered screen coding schemes were proposed. In Miao et al. [2013], we achieve a quality scalability leveraging standard video codec in terms of a layered coding structure. Due to the support on the low-quality bitstream extraction from a standard video codec, the high coding frame rate cannot be guaranteed in this scheme. In Miao et al. [2014], the layered coding scheme can support high-frame-rate coding for interaction content. Since the screen frames with a large region content update are compressed by the standard video codec, visual quality for the text content can be further improved.

3. SYSTEM ARCHITECTURE

We develop a screen sharing system that employs a frame-based representation model. The system architecture is illustrated in Figure 1. On the server side, the whole screen frame is captured from the display buffer and fed into the encoder. The generated bits are transmitted to the client side via a network. On the client side, the datastream received is decompressed as a screen frame, which is copied from the decoding buffer to the rendering buffer for screen update. Meanwhile, the user input data is encrypted and sent to the server side to enable remote interaction.

As discussed in the introduction, system performance is highly relevant to screen coding efficiency, screen update frame rate, and I/O processing efficiency. In order to improve system performance, a layered screen video scheme is developed for efficient screen compression and to further facilitate flexible transmission and high-frame-rate rendering. In addition, a timer-driven screen update mechanism with multiple thread processing is adopted on the server side to schedule the capture, compression, and

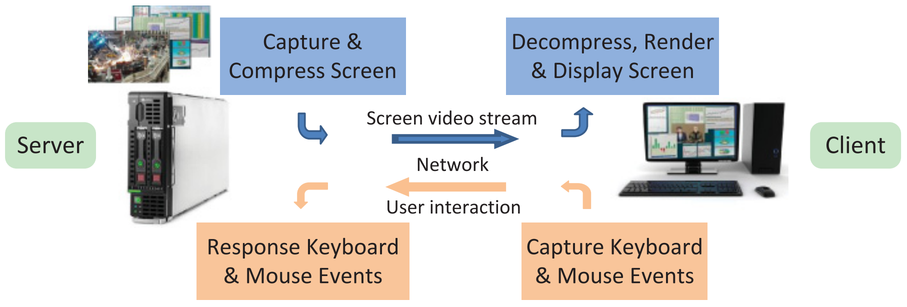

Fig. 1. The architecture of the interactive screen sharing system.

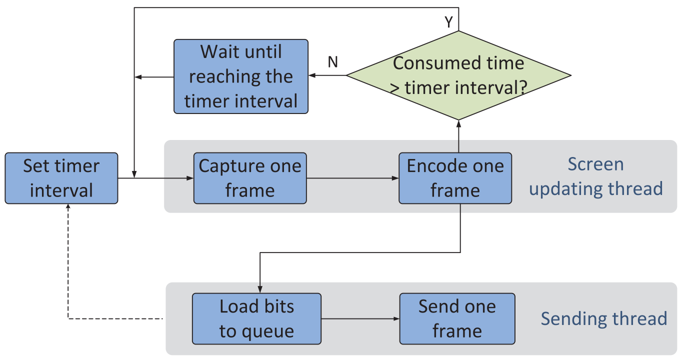

Fig. 2. The multi-thread screen update process based on timer-driven mechanism.

transmission in an adaptive frame rate based on available bandwidth resources. On the client side, with the assistance of the layered structure, a fast rendering scheme is employed, in which a partial region update is supported to save the buffer copy cost during screen rendering.

The screen frames are compressed by the proposed layered coding scheme. The basic idea is that screen frames are partitioned to different layers based on the temporal update property and compressed by distinct codecs. For example, a screen frame containing considerable natural video content is very likely extracted and compressed by video codec. The screen update during interaction is presented at the block level and compressed by a blockwise screen codec with a low complexity to support the high-frame-rate screen update. Under this layered coding structure, an enhancement coding scheme is proposed to improve the visual quality of the text/graphic among the natural video content.

Screen capture, transmission, and rendering should be jointly optimized with screen compression. In the screen sharing system, the server can respond immediately to the request and the screen can update up to 100 times per second. Although the screen can be captured once the update is detected, this mechanism may generate a number of screen frames fed to the encoder and lead to a heavy load on compression and transmission, which may slow down the overall pipeline speed. To schedule the operations of screen capture, compression, and transmission, we adopt a timer-driven screen capture mechanism with multi-thread processing on the server side. The work flow is illustrated in Figure 2. A minimal time interval is set between two captured consecutive frames. During processing, if the consuming time of capture and compression for one frame is within the time interval, then the screen updating thread will wait

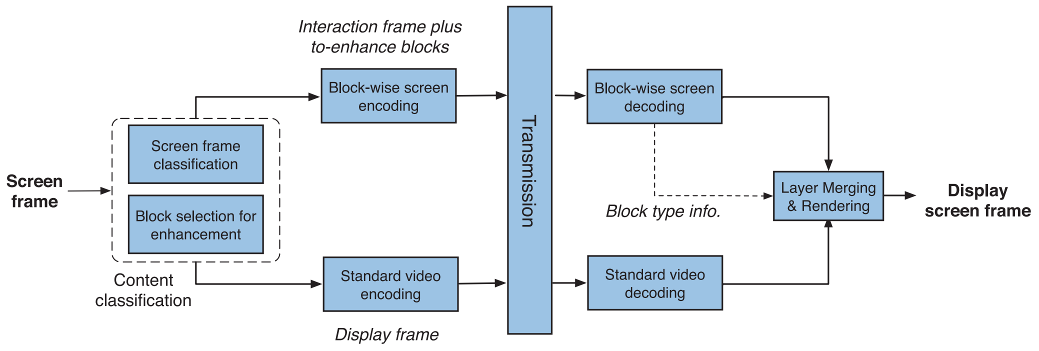
Interaction frame plus to-enhance blocks
Block-wise screen encoding
Screen frame
Screen frame classification
Block selection for enhancement
Content classification
Standard video encoding
Display frame
Transmission
Block-wise screen decoding
Block type info.
Layer Merging & Rendering
Display screen frame
Standard video decoding

Fig. 3. Framework of the layered screen coding and rendering.

until the next timepoint to capture a new frame. Otherwise, the next frame will be captured after compression is finished. In this way, we can control the screen update frequency and avoid the overloading of frames into the processing pipeline to eat up large amounts of computing and communication resources. On the other side, given the resource constraint, we can set the time interval as small as possible to support a high-frame-rate screen update.

Parallel with the screen-updating thread, a background sending thread fetches bits from the to-send queue and sends them to the client. Considering the interaction between the two threads shown in Figure 2, if the coding efficiency is inadequate, then massive coding bits will slow the sending thread processing. Meanwhile, if the coding complexity is high, then the coding processing will slow the compression thread and the sending thread must wait for it. Thus, screen compression plays an important role in the architecture, the efficiency and complexity of which will determine the thread processing speed. Moreover, to match the two threads, we also need to set the screen update frequency to guarantee that the encoded bits of the captured frame can be sent using the available bandwidth. Considering all of the above, we propose a layered screen compression scheme to support high coding efficiency and low coding complexity. Based on the layered coding scheme, we can estimate the bitrate cost of each layer and dynamically determine the time interval to achieve a high screen update frame rate under the bandwidth constraint. In the proposed system, the data are transmitted through a TCP channel, and a TCP-friendly rate control (TFRC) scheme [Handley et al. 2002] is employed to estimate the available bandwidth.

On the client side, to display the screen content, the decoded frames are copied from the decoding buffer into the rendering buffer and then rendered on the screen. The frequency of this buffer copy will partially determine the interaction delay and frame rate on the client side. Based on the layered coding structure, we can skip the data copy operation in the stable content during interaction to reduce the rendering complexity and improve the system performance. Meanwhile, the clients send the input events as client-to-server (C-S) messages to the remote server when the user presses/releases a key in the keyboard or clicks/moves the pointing device.

4. LAYERED SCREEN COMPRESSION AND RENDERING

In this section, we discuss the layered structure of screen coding and rendering, the diagram of which is shown in Figure 3. During screen sharing, the requirement of the frame update frequency is diverse for distinct screen content. For natural video content, the regular frame rate is acceptable for smooth display. For interaction content, the screen frame is expected to be updated at a higher frame rate to achieve smooth

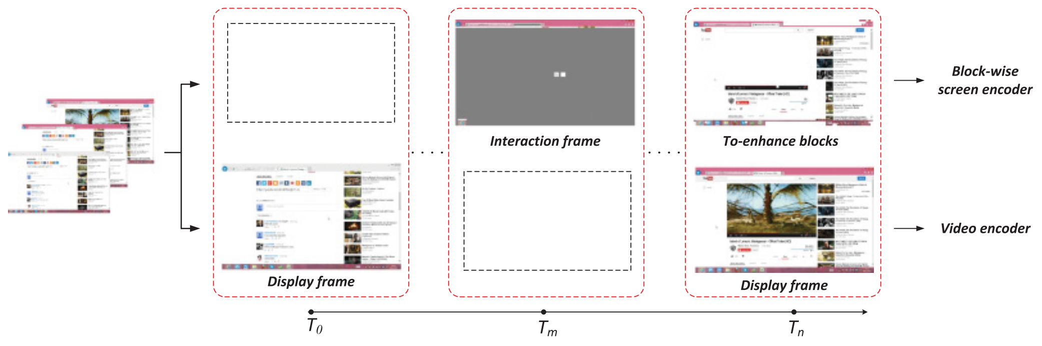

Fig. 4. An example of content classification for layered coding.

interaction. Thus, a screen frame with different content is partitioned into distinct layers to achieve an adaptive coding frame rate. A frame with natural video is labeled as a *display frame* and fed to a standard video encoder compressed at a regular frame rate. A screen frame with interaction content is labeled as an *interaction frame* and compressed by a blockwise screen encoder at a high frame rate. Meanwhile, the visual quality of the display frame can be improved by overcoming two types of coding artifacts. One is the chromatic aberration distortion caused by the data format conversion for YUV420. The other is the coding artifacts introduced by a transform-based video encoder on text/graphic content. In the proposed scheme, the blocks sensitive to the video quality degradation in the display frame will be selected and compressed in the screen encoder again. An example of content classification is shown in Figure 4. The screen content is partitioned as two layers in terms of frame type classification and enhancement block selection. The partitioned content will be compressed in each layer independently. On the client side, the frames are decoded based on the bitstream in each layer and then merged as the complete content in the temporal domain. The enhanced blocks are padded into the decoded frame to obtain quality improvement. After decoding, the rendering is performed with the buffer copy from decoding buffer to rendering buffer and data format transformation. With the side information from the layered structure, we can skip the operation on stable content in a large region which greatly reduces interaction delay during user interaction on the client side.

4.1. Content Classification in Temporal Domain

Screen content with natural video usually updates in a large region during video display. In contrast, many types of user interactions cause screen updates in a small region, such as text input or cursor moving. To verify this update pattern, we measure the content variation for distinct screen content. We first classify the block in each frame as one of two types: *stable block* and *update block*. For each block, if it is exactly the same as an original region in the previous frame, this block is identified as a stable block. Otherwise, it is identified as an update block with content updates. We measure the update block ratio in each frame captured from the screen sequences shown in Figure 5 with a resolution of 1360×768. Then the distribution of the update block ratio in each frame is shown in Figure 6, which reflects the content update percentage in each frame. From the results, we can observe the natural video content updates in almost the whole frame and the webpage video causes a screen update in a certain percentage that is related to the size of video window. The screen update ratio of the interaction content is within a small range.

Based on the measurement results, we can simply classify screen content in terms of the update block ratio. If the update block ratio is within a threshold, then this

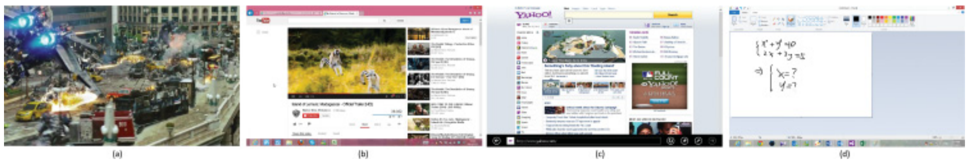

Fig. 5. Sample frames of distinct scenarios used for screen sharing in experiment. ((a) and (b)) Video display as full-screen video and webpage video; ((c) and (d)) user interaction as webpage browsing and editing on the board.

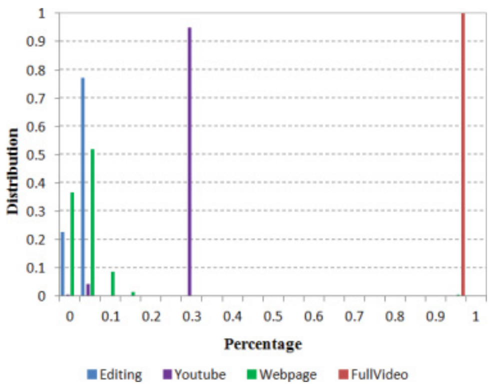

Fig. 6. The distribution of the update block percentage in screen sequence.

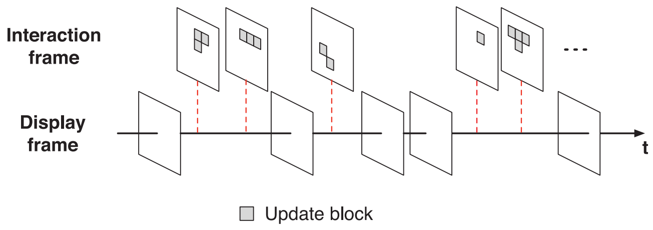

Fig. 7. An example of screen frame layer classification.

frame is considered an interaction frame and fed to the screen encoder. Otherwise, it is considered a display frame, which is fed to the video encoder as shown in Figure 7. An example of screen frame classification can be observed in Figure 4. The initial frame is fed to the video encoder at first, and then the frames with a small region update during webpage scrolling are fed to a screen encoder. After the webpage video display, the frames are fed to the video encoder.

4.2. Content Selection for Quality Improvement

To maintain the compatibility with the optimized standard video codec, the data format of the display frame should be transformed as YUV420 in the video coding layer. The downsampling in the chroma channel may introduce distortion in the high-contrast region. Moreover, the text/graphic content may appear in a display frame, such as text/graphic around the webpage video or in large region interaction during scene changes. The transform-based coding scheme cannot compress text/graphic content efficiently. To address these issues and improve the visual quality further, the block sensitive to quality degradation is selected and the original data are compressed independently in the screen encoder again.

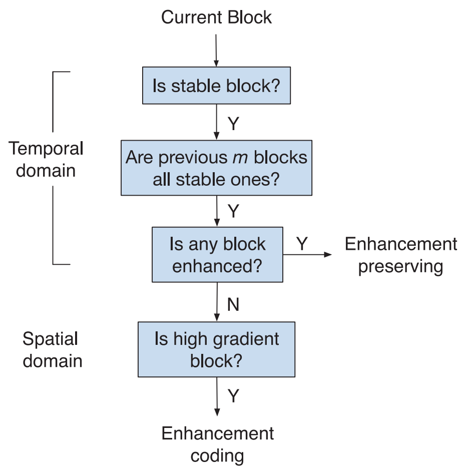

Fig. 8. The flow of block selection algorithm for quality enhancement.

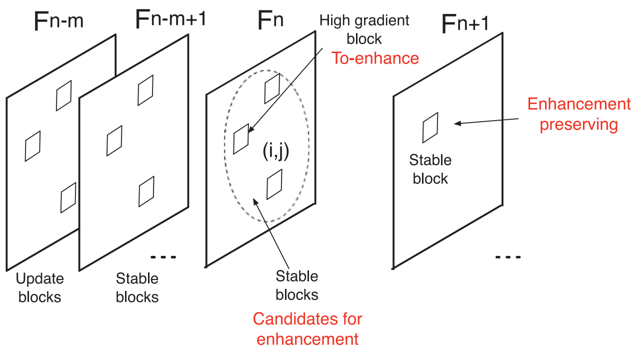

Fig. 9. An example of enhancement block selection.

We evaluate the content's impact on the visual quality in both temporal and spatial domains. The temporal impact is determined by its duration. If the screen content is stable, then quality enhancement at the beginning will be preserved in the following stable frames by global motion compensation. Accordingly, the overall video quality can be improved with a limited bitrate cost on enhancement coding at only one frame. For the active region with frequent content change, although video quality can be enhanced frame by frame, quality improvement can hardly be perceived due to fast changes in content. Thus, we select the stable content as enhancement candidates. In the spatial domain, the impact on video quality is evaluated by inherent texture of content. Since the human visual system is more sensitive to the distortion in a high-contrast region than a smooth region, the high contrast blocks will be enhanced.

In combination with temporal and spatial selection rules, the flow of the proposed algorithm is illustrated in Figure 8. The stable content duration is measured by the number of successive stable blocks in adjacent frames. For a block region, a stable period is considered detected if m consecutive blocks are all stable in this location ($m > 1$), as shown in Figure 9. Notice that this algorithm is an on-line scheme during the encoding process. What we can obtain is only the block type in the current and previous frames. For one block in the current frame F_n detected as a stable block, if the blocks in the co-located position of the previous $m - 1$ frames are all stable, then the content within this position is stable. Then, if no block has been enhanced after the nearest update block in the $(n-m)$-th frame, the current block will be regarded an enhancement coding candidate. If there is any stable block enhanced in this position, then the following stable blocks will be labeled as the enhancement preserving block and reconstructed by copying from the previous enhanced block on the decoder side to preserve the quality improvement in this stable period. After temporal filtering, the gradient value of each pixel in block is calculated. If the number of pixels with a high gradient value is beyond a threshold, then the block is considered a high-contrast block with text/graphic or image edge content. We will perform enhancement coding on it. An instance of enhancement block selection is shown in Figure 4. In this example, the blocks with high-contrast content, such as text/graphics around the video region in a webpage, are selected and compressed again in the screen encoder.

4.3. Timer-Driven Screen Capture

On the server side, the frequency of screen capture is determined by the time interval we set in the system. We can achieve a high frame rate by setting a small time interval. Meanwhile, we need to control the time interval to guarantee that the captured screen frames will not overload the bandwidth and block the transmission pipeline. In this subsection, we will dynamically determine the screen frame update frequency based on the estimated bitrate cost of the layered codec and the available bandwidth resource.

Based on the layered coding structure, the overall bitrate cost will be

$$R = \alpha \cdot f_d \cdot R_d + (1-\alpha) \cdot f_i \cdot R_i, \tag{1}$$

where f_d, f_i are the frame rates of the display frame and the interaction frame, respectively. R_d, R_i are the average bitrate costs for each display frame and interaction frame. α is the display frame ratio in a screen sequence. In the proposed system, if the consuming time of the capture and compression for one frame is within the time interval, the screen updating thread will wait until the next timepoint to capture a new frame. Otherwise, the next frame will be captured after compression is finished. In the layered coding scheme, the display frame is compressed by a video codec in a regular frame rate, which is lower than the high frame rate set in the system. Thus, the display frame rate f_d can be estimated as a regular frame rate. Given the quantization step-size set in the video codec, the bitrate R_d can be estimated by the rate distortion model [Chiang and Zhang 1997]. For the screen coding layer, the bitrate is mainly consumed by intra-block coding. Thus, the upper bound of the bitrate can be estimated as:

$$R_{i_{upper}} = N \cdot Th_0 \cdot r_{update}, \tag{2}$$

where r_{update} is the average bitrate cost for each update block in the screen codec. Th_0 is the update block ratio threshold. N is the block number in each frame. $N \cdot Th_0$ will be the maximum number of update blocks possibly appearing in an interaction frame. The average bitrate cost r_{update} is updated by the previous encoded update blocks. We employ this upper bound to perform the estimation on capture frequency, so we can avoid the capture overloading caused by estimation error. Considering that the screen content has a temporal correlation, we use the frame type information of the previous neighboring frames to estimate the display frame ratio in this period. Based on the TCP rate control scheme, we can estimate the available bandwidth. Given the bandwidth constraint R_0 and Equation (1), we can dynamically determine the update rate for the interaction frame f_i. To support this high frame rate, we can set the screen capture frame rate f_0 as:

$$f_0 = min(\bar{f_i}, f_{max}), \tag{3}$$

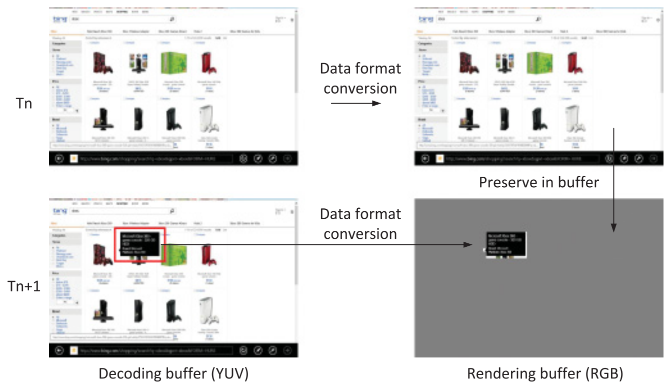

Fig. 10. An example of layered screen rendering. The gray region will be preserved without processing in the buffer.

where $\bar{f}_i$ is the estimated rate of the interaction frame and f_{max} is an upper bound frame rate we set to guarantee that the coding frame rate in screen layer can achieve or close to this frequency. Then the time interval T_0 will be set as $1/f_0$.

4.4. Layered Screen Rendering

On the client side, the decoded frame should be copied from the decoding buffer to the rendering buffer with the data format transformed from YUV to RGB for display. The rendering processing complexity will affect interaction delay on the thin client side. The frequent I/O buffer copy and data format conversion for large amounts of pixels will be very costly in terms of computing resources and reduce the screen frame update speed. In this system, we introduce a rendering scheme in terms of the layered structure aiming to reduce rendering complexity. As we know, there is no content change for the stable content between the neighboring decoded interaction frames, and content can be preserved without updates to the rendering buffer. Thus, during rendering, we extract block type information from the layered screen codec. For the interaction frame, update blocks will be processed and rendered for display while stable blocks with zero motion will be skipped from buffer copy as well as data format conversion, and the content in the buffer is preserved without change, as shown in Figure 10. Considering that the interaction frame contains update blocks in a small region and stable blocks in a large region, the numbers of buffer copy and data format conversion operations can be saved with computing complexity reduction.

5. SCREEN CONTENT CODEC DESIGN

In this section, we will introduce the implementation of the layered coding scheme including the coding mode in screen encoder and layered syntax design.

5.1. Blockwise Screen Content Coding

In order to compress interaction content with high visual quality and low complexity, the screen encoder is designed as an open-loop coding structure. Four coding modes are introduced, which are skip mode, text mode, image smooth mode, and image edge mode. For the stable block, temporal data correlation is exploited by skip mode without residual coding and reference frame reconstruction on the encoder side. A fast global motion detection [Christiansen and Schauser 2002] is conducted based on original frames. For the detected stable block, only block type and motion vector are transmitted to the decoder. In this way, the abundant redundancy within stable content can be efficiently

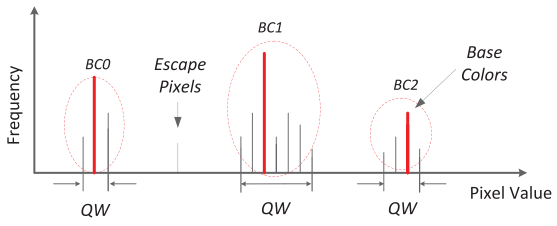

Fig. 11. The quantization in pixel domain for mode decision.

exploited with very low complexity and the main computation cost is allocated to the update blocks in the small region.

For the update block, we first perform content classification. Considering that a text/graphic region has high contrast and a limited number of base colors, a histogram-based quantization algorithm is employed to classify the text/graphic region from the image one. The colors with peak values in the histogram are selected as base colors, shown in Figure 11. Then, an equal size window is used to range the close colors that are quantized to the base color in the same window. There might also be some pixels that escaped from the ranging windows. If the escaped pixel number is within a threshold, then the block is considered a text/graphic. Otherwise, it is set as an image block. The image content is then further classified as a smooth region and an edge region based on the block gradient value. If the block gradient value is beyond a threshold, then the block is considered an image edge block. Otherwise, it is an image smooth block.

For the text/graphic block, encoding without transform is an efficient coding scheme for content with complicated and irregular edges. Thus, we employ a pixel domain coding scheme [Miao et al. 2013]. Based on the above quantization scheme, the text block can be represented by several base colors, escaped colors, and an index map, which indicates that each pixel corresponds to a particular color value. These color values and index map are entropy encoded directly. For the image block, the mature JPEG encoder is adopted. In the edge region, considering that chroma downsampling will introduce chromatic aberration distortion, UV channel data will be compressed without downsampling. In the smooth region, the chroma channel is downsampled for efficient coding.

For enhancement coding, since the blocks selected for enhancement all have a high gradient value, the to-enhance blocks will be classified as a text/graphic one or image edge one and compressed by the text mode or image edge mode, respectively. The stable blocks following the to-enhance one in the co-located position will be compressed by the skip mode to preserve quality improvement.

5.2. Syntax of the Layered Coding Scheme

To combine the standard video codec and the blockwise screen codec under the layered coding structure, we design the coding syntax as illustrated in Figure 12. For each encoded screen frame, the bitstream starts with the frame type label that may be the *display frame* or *interaction frame*. Following that, the slice level bitstream is written. Three types of slices are introduced, which are *video slice*, *interaction slice*, and *quality enhancement slice*.

The video slice contains the bitstream of display frame encoded by a standard video encoder. To achieve the tradeoff between the coding efficiency and coding complexity, H.264/AVC is adopted with the real-time implementation of x264 as an encoder and FFmpeg as a decoder. Thus, in the video slice, the H.264/AVC bitstream of one frame is written following the slice type. The interaction slice contains the bitstream of the interaction frame from the screen encoder. Following the slice type, the bitstream of

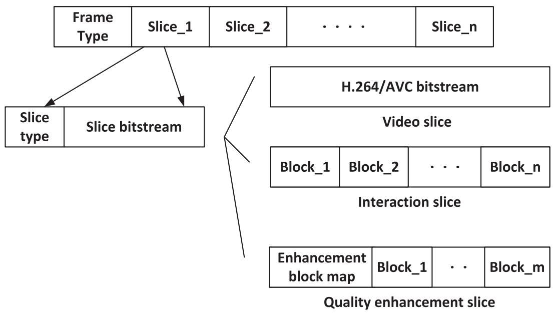

Fig. 12. Syntax of the layered coding.

each block is written with a horizontal scan order. The quality enhancement slice contains the coding bitstream of enhanced blocks. Since partial blocks are selected for quality improvement, following the slice type, there is an enhancement map written into the bitstream to identify the location of the block. Three coding modes, including text mode, image edge mode, and skip mode, are involved for the enhancement coding.

On the decoder side, the decoded frames from two layers are merged as the complete screen video first. For the interaction frame, the reconstructed update block is padded to the decoded frame directly. For stable blocks, the reconstructed content is copied from the previous decoded frame in the completed screen sequence. Then, the enhanced blocks are padded to a decoded frame to improve quality, and enhancement preserving blocks are reconstructed by copying from the previous frame with enhanced content in terms of global motion.

6. EXPERIMENTAL RESULT

We implement the proposed screen sharing system and evaluate its performance on the real application over a wide range of network environments. Considering that this system is developed based on software implementation, we compare it with several popular software-based screen sharing systems, including RDP, VNC, THINC, DisplayCast [Chandra et al. 2012], AnyDesk [2016a], Splashtop [2016], and Gaming Anywhere [Huang et al. 2013].

6.1. Tested Setup

The test-bed consists of two desktop computers within various network conditions. The server is running on Microsoft Windows 8 and equipped with an Intel Core i5-3210M processor with two 2.50GHz cores. The client is running on Microsoft Windows 8 and configured with an Intel Core i5-3317U processor with two 1.70GHz cores. The display resolutions on both the server and client are set as 1360×768, which is one option in Windows. The conditions of the relevant screen sharing systems are as follows:

—*RDP*: We use the pre-installed version of Windows 8.
—*VNC*: We use the TightVNC downloaded from TightVNC [2016].
—*THINC*: The system is downloaded from THINC [Software Systems Laboratory 2016].
—*DisplayCast*: The system is downloaded from Github [DisplayCastSourcecode 2016].
—*AnyDesk*: The system is downloaded from the webpage [AnyDesk 2016a].
—*Splashtop*: The system is downloaded from the webpage [Splashtop 2016]. To use this service, we create an account and run the sender and receiver programs on the respective machines.

—*GamingAnywhere (GA)*: This is an open-source cloud gaming platform download from GamingAnywhere [2016].

In the proposed system, the mature software x264 [VideoLAN 2016] is integrated as the encoder in video coding layer and the default mode is used for encoding configuration. Considering that the hierarchical b-frame will increase decoding delay, we disable the b-frame prediction and employ IPPP coding structure with one reference frame to reduce the decoding delay. The GOP size is set as 16. The mature software FFmpeg [2016] is used as a video decoder.

In the screen coding layer, the number of base colors in the pixel domain is set as 4, since four clusters can cover most color values in the text block by quantization. It also achieves a good tradeoff on bitrate cost between the index map and escaped colors. Based on the investigation result in Section 4.1, the update block ratio threshold is set as 20% for the frame classification. Under this setting, most video display cases including full-screen video and webpage video can be classified as natural video content. This parameter can be adjusted in terms of distinct screen sharing scenarios or screen resolutions. The upper bound frame rate we set in the capture module is equal to 60fps based on the coding complexity of the screen coding layer.

We use a 1Gb/s LAN network, with 1ms latency, which is the maximum granularity in the network. A software-based network emulation is performed on the server side by Shunra VE Desktop Client to emulate the different network conditions. Wireshark 1.2.8 [Wireshark 2016] is used to monitor and record network traffic. The tested network environments vary among 100Mbps FastEthernet LAN, 10Mbps Ethernet, and 5Mbps and 2Mbps low-bandwidth 802.11g, which are all produced by the software-based network emulator.

In our experiment, two types of scenarios are tested, which are video display and interaction. For video display, the scenario is further classified as full-screen video display, as shown in Case (a) in Figure 5, which is a trailer of "Transformer 3," and webpage video display as shown in Case (b) in Figure 5, in which the video is clicked following an input interaction. For the interaction scenario, two cases are tested shown as Cases (c) and (d) in Figure 5, which are webpage browsing and writing on the board.

6.2. Latency and Frame-Rate Test

In this experiment, we measure the end-to-end screen updating latency between the server and the client, which is the time difference between the screen update appearing at the server side and the update displaying at the client side. The end-to-end latency can be calculated as the sum of the encoding, transmission, decoding, and other processing times. Given that some reference systems are close source and it is difficult to measure the time of each module, a photo-capture-based method is employed in this article, which is used in both academia and industry for performance measurement on screen sharing [Hsu et al. 2015; measurement in AnyDesk 2016b]. In this method, the screen with a timer showing in the corner of the screen is sent from the server to the client. Both screens on server and client sides are recorded in one image by a Fujifilm x100s camera at the same time. The difference between times shown on two screens is considered the latency. We measure the latency for video display and interaction scenarios. Each scenario has two cases, as shown in Figure 5. We repeat the experiment 10 times for each case given the bandwidth condition.

The average latency results for distinct scenarios are shown in Table I. From the results, we can reach several conclusions. First, the latency will increase as the bandwidth decreases, mainly due to the increase of transmission time. Second, the latency of the video display is larger than that of the interaction for the distinct systems. This is mainly because the natural video content update will cause more compressed bits

Table I. Average Latency Comparison Results During Interaction and Video Display (MS)

	Interaction				Video			
Systems	100Mbps	10Mbps	5Mbps	2Mbps	100Mbps	10Mbps	5Mbps	2Mbps
RDP	46	61	93	145	60	198	271	383
VNC	36	48	73	137	67	194	298	410
THINC	32	40	74	138	52	116	213	373
AnyDesk	26	41	92	117	43	73	176	275
DisplayCast	23	34	53	86	63	156	231	398
Spalashtop	37	45	85	124	47	84	187	352
GamingAnywhere	33	46	83	135	35	57	132	201
Proposed	**24**	**36**	**47**	**78**	**38**	**72**	**142**	**228**

Table II. Frame Rate Comparison Results During Interaction and Video Display (FPS)

	Interaction				Video			
Systems	100Mbps	10Mbps	5Mbps	2Mbps	100Mbps	10Mbps	5Mbps	2Mbps
RDP	24	22	20	16	18	13	11	8
VNC	35	32	27	23	17	14	13	11
THINC	37	35	29	25	23	19	15	12
AnyDesk	43	38	35	32	27	25	20	15
DisplayCast	50	45	36	30	23	16	13	8
Spalashtop	31	29	28	25	23	21	16	13
GamingAnywhere	35	33	31	29	31	25	23	16
Proposed	**49**	**43**	**37**	**35**	**28**	**24**	**21**	**14**

than the interaction update in a small region and will cost more time on compression and transmission accordingly. Third, in the interaction scenario, the proposed system and DisplayCast can achieve lower latency than other systems. In the proposed system, the latency can be preserved within 80ms, even if the bandwidth is limited. Compared with GamingAnywhere, in which x264 is employed for screen compression, the latency can be reduced about 40% in low-bandwidth conditions. This is mainly because the encoding, decoding, and transmission times can be reduced due to the high-efficiency and low-complexity coding in the proposed scheme. Moreover, the rendering time can be further reduced by the partial region update during interaction. When bandwidth is sufficient and the transmission cost introduces less impact on latency, the proposed system can also achieve about 20% improvement compared with GamingAnywhere. For video display scenario, GamingAnywhere can outperform others and the proposed system can achieve a performance comparable to that of GamingAnywhere. Compared with VNC, the proposed system can reduce the latency significantly in various bandwidth conditions.

Since the frame rate will influence user experience, we measure the frame rate on the client side by using the API function of DirectX to monitor the screen update while the screen sharing system is running. Once the screen changes, the time is noted and the frame rate is recorded. Considering that the requirement for screen update varies for distinct content, we measure the frame rate of video display and interaction, respectively. For each scenario, two cases, shown in Figure 5, are tested. The average results for each scenario are shown in Table II. We can observe that, in the interaction scenario, the proposed system can outperform most other systems. The frame rate can achieve up to 50fps in the high-bandwidth condition and about 35fps in the low-bandwidth condition. The high frequency of screen update can be also supported well in DisplayCast when the bandwidth is sufficient. Compared with GamingAnywhere, the proposed system can achieve an approximately 35% higher frame rate in the high-bandwidth condition and about a 20% improvement in low-bandwidth condition. This is mainly because the open-loop coding structure in screen

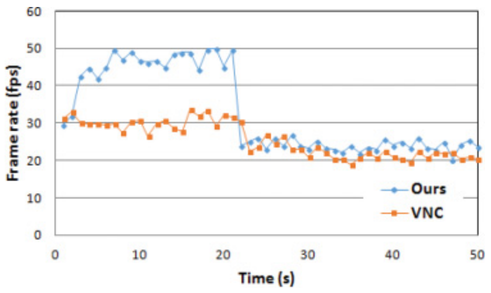

Fig. 13. Frame-rate variation during webpage video interaction and display.

codec can reduce the encoding and decoding time. The rendering cost can also be saved in the stable region during interaction. In contrast, the close-loop coding structure in video codec and whole frame rendering operation introduce higher coding and rending complexity in the reference system. Even though the transmission time can be reduced in the high bandwidth condition, the frame rate in local devices cannot be improved further due to the bottleneck caused by the coding and rendering complexity.

For the video display scenario, GamingAnywhere can achieve the best performance, followed by the proposed system. In these two systems, a standard video codec can support real-time streaming when the bandwidth is beyond 5Mbps. The frame rate in the system based on the region update model is inadequate for video display. For example, the frame rate in VNC for webpage video can achieve about 20fps, while it drops to about 10fps for full-screen video display. This is because the screen is updated as the form of updated pixels and the frame rate will decrease significantly if the amount of the update pixels is as large as the whole screen. Similarly to VNC, DisplayCast is also based on pixel update and suffers from the same issue for full-screen video display. Note that the frame rate in the proposed system is also inadequate for video display when bandwidth is limited. We will improve it further for the bandwidth-limited condition.

We also test the frame-rate variation in a webpage video display case shown as Figure 5(b). In this case, the user first typed the video name in the search bar on the Youtube webpage and then clicked a video for display. The frame-rate variation result under the bandwidth of 100Mbps is shown in Figure 13. From the result, we can see that during the user interaction, the frame rate in the client side can achieve up to 50fps. After video display, the frame rate is beyond 20fps. The proposed coding scheme can provide an adaptive frame rate for distinct content. We can also see that the frame rate in VNC can also achieve about 20fps during webpage video display. As mentioned above, the screen in TightVNC is represented as rectangles of pixels, which are compressed by a Hextile scheme in this experiment. The compression efficiency and complexity are relevant with the size of rectangle. When the screen update ratio is about 30% as webpage video display, a frame rate of about 20fps can be supported in TightVNC.

6.3. Video Quality Test

In this section, we present the evaluation of video quality at the codec level and system level, respectively. We first capture four offline screen video sequences under the distinct scenarios shown in Figure 5. Each sequence has 200 frames with a resolution of 1360×768. These four video sequences are compressed by the proposed layered codec with different bitrates. Considering that x264 is widely employed in many screen sharing systems, such as Miracast [Wi-Fi Alliance 2016] and GamingAnywhere [Huang et al. 2013], we compare the proposed layered codec with x264. The encoding

Table III. Performance of the Proposed Scheme Compared with x264 Evaluated by PSNR and SSIM

Sequence	ΔPSNR(dB)	ΔBitrate(%)	ΔSSIM	ΔBitrate(%)
Full-screen video	−0.09	1.46	−0.0012	1.23
Webpage video	0.56	−10.76	0.0082	−11.38
Web browsing	1.24	−22.54	0.0138	−23.34
Editing	0.63	−13.52	0.0094	−13.47

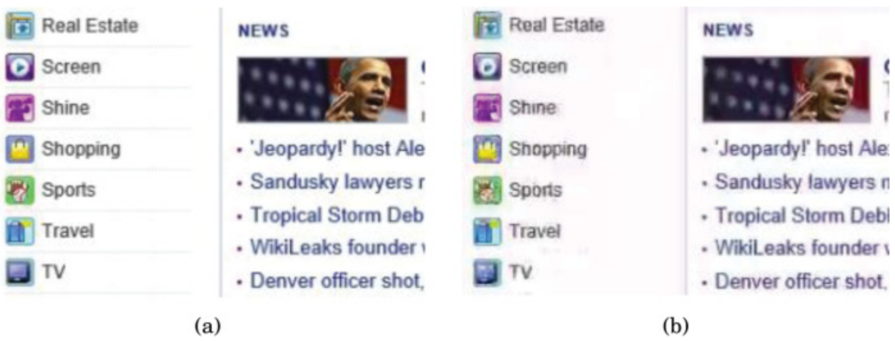

Fig. 14. Visual quality comparison between the proposed layered coding scheme and x264. (a) Reconstructed from the proposed scheme (bitrate: 1.83Mbps). (b) Reconstructed from x264 (bitrate: 1.84Mbps).

configuration is set the same as the video codec integrated in the video coding layer, and the main setting is introduced in Section 6.1. The data format in x264 is YUV420 while it is YUV444 in the screen coding layer of the proposed scheme. Considering that the chroma data are processed and compressed in distinct ways in the proposed scheme and reference scheme, we evaluate the coding performance in the RGB color space. The peak signal-to-noise ratio (PSNR) and structural similarity (SSIM) values are calculated as the average value of each channel in RGB color space. The rate distortion performance is shown in Table III.

For the natural video display in full screen, the coding performance of the proposed coding scheme is almost the same as x264. This is because most screen frames are distributed into the video coding layer. The limited coding lost is caused by the layered coding overhead. For the webpage video content, the proposed scheme achieves better rate-distortion performance than x264 due to the fact that the natural video content is compressed by the video codec and the text/graphic content is enhanced by the proposed screen coding. For the screen with full text/graphic content, the coding performance of the proposed scheme is much better than x264. This is because the pixel domain coding is more efficient for text content, and chromatic aberration distortion is avoided by full chroma data compression without downsampling. The visual quality comparison is shown in Figure 14, in which a region of screen is cut from the webpage. The bitrate is controlled similarly for the two schemes. We can observe that there are artifacts around the boundary of text in the image reconstructed from x264. In contrast, the reconstruction result from the proposed scheme has a clear boundary of text.

We compare the video quality at the system level using a similar method employed in Hsu et al. [2015]. We hack and record the screen frames from the display buffer in both server and client to calculate the frame quality based on PSNR. To facilitate the comparison, we add a frame identifyer at the corner of each frame for matching. Since the frame rates may be mismatched between the server and the client, the number of the received screen frames on the client side could be less than that on the server side. We copy the previous received frames on the client side to guarantee

Table IV. Video Quality Comparison During Interaction and Video Display Based on PSNR Value (dB)

	Interaction				Video			
Systems	100Mbps	10Mbps	5Mbps	2Mbps	100Mbps	10Mbps	5Mbps	2Mbps
RDP	31	27.6	26	24.5	26.8	23.4	21	15
VNC	32.4	31	28.9	26	26	24.4	22.5	18.4
THINC	31	28.5	26.8	24	31	28.5	26	21
AnyDesk	34	32.5	29	27	32	30.5	28.2	22
DisplayCast	36	33.5	32	28.4	27.2	25.4	22.3	19
Spalashtop	32	30.7	29	27.5	29.5	26.2	24.5	21
GamingAnywhere	33	32.1	30.6	27.3	33	32	30.5	23.5
Proposed	**36.5**	**34**	**32.4**	**29**	**32.5**	**31.1**	**29.5**	**23**

Table V. Small Region Update Performance

	Webpage		Editing	
Protocol	Data(Mbps)	Speed(fps)	Data(Mbps)	Speed(fps)
VNC	1.61	31	0.90	36
Proposed	0.71	43	0.25	49

that the total numbers of screen frames are equal for comparison. Thus, the video quality in this test is mainly determined by two factors. One is the received video quality and the other is the frame rate on the client side. If the frame rate on the client side is much lower than that on the server side, then the quality will decrease dramatically. The comparison results are shown in Table IV. In the interaction scenario, the video quality of the proposed system is better than other system under various bandwidth conditions. Compared with the screen sharing system employing x264, such as GamingAnywhere, the proposed system can provide 5%–10% performance improvement in low- to high-bandwidth conditions. The larger improvement in high bandwidth is mainly caused by the higher frame rate supported in the proposed system. Compared with the systems based on the region update model, such as VNC and THINC, the video quality improvement is caused by both the quality improvement in the text content and the frame-rate preservation on the client side. In the video display scenario, the video quality in this system is similar to that of GamingAnywhere, since the screens are compressed by x264 in both systems. Compared with the systems based on a region update model, the video quality improvement is mainly caused by frame-rate preservation on the client side.

We also compare the small region update performance between the proposed system and VNC using Cases (c) and (d) shown in Figure 5. In the experiment, we measure the transmission rate and frame rate shown in Table V. Due to the high coding efficiency on text/graphic content, the bitrate of the proposed system is lower than VNC. Since the complexity on blockwise coding and layered rendering can be preserved in a low level, the updating rate in the proposed system on a small region can be higher than a regular frame rate.

6.4. Complexity Test

This experiment is designed to measure the CPU utilization on both the server and client sides. A Performance Monitor tool [Microsoft 2016b] is used to monitor the CPU usage and the results are shown in Table VI.

From the results, we can observe that the CPU usage of the proposed system is comparable with other systems. For the distinct scenarios, the CPU utilization for full-screen video display is highest when the video codec is working for the screen compression. The interaction case consumes the least amount of CPU resources, since the complexity of the screen codec is lower than that of the nature video codec.

Table VI. CPU Utilization Comparison Results

Protocol	Full-screen video		Webpage video		Interaction	
	Server	Client	Server	Client	Server	Client
VNC	16%	12%	13%	17%	14%	12%
RDP	35%	32%	35%	22%	27%	19%
THINC	21%	19%	18%	15%	16%	15%
AnyDesk	29%	23%	21%	18%	17%	15%
DisplayCast	39%	32%	33%	25%	24%	21%
Spalashtop	31%	27%	26%	19%	23%	21%
GamingAnywhere	33%	18%	27%	17%	19%	12%
Proposed	**37%**	**15%**	**32%**	**16%**	**12%**	**10%**

Table VII. Complexity of the Rendering Operation (MS/Frame)

Scenario	Reference	Proposed	Ratio
Full-screen video	6.82	**6.87**	1.01
Webpage video	6.71	**4.49**	0.67
Interaction on small region	6.51	**1.35**	0.21

Note: Ratio = Proposed/Reference

We also test the complexity of the rendering operation with buffer copy and data format conversion. Considering that it is difficult to measure screen rendering complexity in other closed-source systems, we test the rendering cost compared with a reference system integrated with pure video codec for screen coding, that is, x264 software as the screen encoder and FFmpeg as the decoder. The result is shown in Table VII, in which we can observe that the rendering cost in the reference system is almost fixed. This is because the data format conversion and buffer copy are performed for the whole frame. In the proposed system, the rendering cost can be reduced, since the stable content in each frame skip the rendering operation during screen update. The reduce proportion corresponds to the stable content ratio in each scenario. For the interaction in teh small region, the rendering cost can be saved up to 80% compared with the reference system, whereas, for the natural video display, the rendering cost is almost the same as the reference.

7. CONCLUSION AND DISCUSSION

In this article, we have presented a novel screen sharing system. In this system, we develop a low-level compression-friendly architecture to represent and compress the display information. Within the model, we further propose a layered screen compression scheme for screens with multiple types of content. The proposed screen compression scheme employs the standard video codec to compress the screen with multimedia content at a regular frame rate and the specially designed blockwise screen codec to compress interaction content at a high frame rate. The experimental results show that the proposed system delivers superior performances compared with several popular screen sharing systems.

In order to improve system performance and extend application scenarios, we shall focus our future work on hardware implementation. In the future research, we plan to implement hardware encoding and decoding of video codec in the system to reduce the end-to-end latency as well as CPU usage in server and client. Moreover, with the benefit of hardware implementation, we shall be able to support more scenarios, such as screencast from a laptop to a TV through a developed screen sharing box, in which the screen is decoded and sent to a TV through an HDMI port. More comparisons with hardware-based screen sharing systems, such as Miracast and Airplay, will also be conducted in the future.

REFERENCES

Apple. 2016. Apple Airplay. Retrieved from http://www.apple.com/appletv/airplay/.

AnyDesk. 2016a. Homepage. Retrieved from http://anydesk.com/remote-desktop.

AnyDesk. 2016b. Remote Desktop Software Benchmark. Retrieved from http://anydesk.com/benchmark/anydesk-benchmark.pdf.

Ricardo A. Baratto, Leonard N. Kim, and Jason Nieh. 2005. THINC: A virtual display architecture for thin-client computing. *ACM SIGOPS Operat. Syst. Rev.* 39, 5 (2005), 277–290.

Surendar Chandra, Jacob T. Biehl, John Boreczky, Scott Carter, and Lawrence A. Rowe. 2012. Understanding screen contents for building a high performance, real time screen sharing system. In *Proceedings of the 20th ACM International Conference on Multimedia*. ACM, New York, NY, 389–398.

Yu-Chun Chang, Po-Han Tseng, Kuan-Ta Chen, and Chin-Laung Lei. 2011. Understanding the performance of thin-client gaming. In *Proceedings of the 2011 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR)*. IEEE, 1–6.

Kuan-Ta Chen, Yu-Chun Chang, Hwai-Jung Hsu, De-Yu Chen, Chun-Ying Huang, and Cheng-Hsin Hsu. 2014. On the quality of service of cloud gaming systems. *IEEE Trans. Multimed.* 16, 2 (2014), 480–495.

Kuan-Ta Chen, Yu-Chun Chang, Po-Han Tseng, Chun-Ying Huang, and Chin-Laung Lei. 2011. Measuring the latency of cloud gaming systems. In *Proceedings of the 19th ACM International Conference on Multimedia*. ACM, New York, NY, 1269–1272.

Tihao Chiang and Ya-Qin Zhang. 1997. A new rate control scheme using quadratic rate distortion model. *IEEE Trans. Circuits Syst. Video Technol.* 7, 1 (1997), 246–250.

Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. 2012. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*. IEEE, 1–6.

Bernd Oliver Christiansen and Klaus Erik Schauser. 2002. Fast motion detection for thin client compression. In *Proceedings of the 2002 Data Compression Conference*. IEEE, 332–341.

Chromecast. 2016. Homepage. Retrieved from http://www.google.com/chrome/devices/chromecast/.

Lien Deboosere, Jeroen De Wachter, Pieter Simoens, Filip De Turck, Bart Dhoedt, and Piet Demeester. 2007. Thin client computing solutions in low-and high-motion scenarios. In *Proceedings of the 3rd International Conference on Networking and Services, 2007.* IEEE, 38–38.

Wenpeng Ding, Yan Lu, and Feng Wu. 2007. Enable efficient compound image compression in H. 264/AVC intra coding. In *Proceedings of the IEEE International Conference on Image Processing (ICIP) 2007*, Vol. 2. IEEE, 337–340.

DisplayCastSourcecode. 2016. Homepage. Retrieved from https://github.com/displayCast/.

FFmpeg. 2016. Homepage. Retrieved from http://ffmpeg.org/.

GamingAnywhere. 2016. GamingAnywhere: An Open Cloud Gaming System. Retrieved from https://github.com/chunying/gaminganywhere.

Patrick Haffner, Paul G. Howard, Patrice Simard, Yoshua Bengio, Yann Lecun, and others. 1998. High quality document image compression with DjVu. *J. Electron. Imag.* 7, 3 (1998), 410–425.

Mark Handley, Sally Floyd, Jitendra Padhye, and Jörg Widmer. 2002. *TCP Friendly Rate Control (TFRC): Protocol Specification*. Technical Report.

Chih-Fan Hsu, Tsung-Han Tsai, Chun-Ying Huang, Cheng-Hsin Hsu, and Kuan-Ta Chen. 2015. Screencast dissected: Performance measurements and design considerations. In *Proceedings of the 6th ACM Multimedia Systems Conference*. ACM, New York, NY, 177–188.

Chun-Ying Huang, Cheng-Hsin Hsu, Yu-Chun Chang, and Kuan-Ta Chen. 2013. GamingAnywhere: An open cloud gaming system. In *Proceedings of the 4th ACM Multimedia Systems Conference*. ACM, New York, NY, 36–47.

Michael Jarschel, Daniel Schlosser, Sven Scheuring, and Tobias Hofeld. 2011. An evaluation of QoE in cloud gaming based on subjective tests. In *Proceedings of the 2011 5th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*. IEEE, 330–335.

LogMeIn. 2016. Homepage. Retrieved from https://secure.logmein.com/.

Yan Lu, Shipeng Li, and Huifeng Shen. 2011. Virtualized screen: A third element for cloud-mobile convergence. *IEEE Trans. Multimed.* 18, 2 (2011), 4–11.

Dan Miao, Jingjing Fu, Yan Lu, Shipeng Li, and Chang Wen Chen. 2013. Layered screen video coding leveraging hardware video codec. In *Proceedings of the 2013 IEEE International Conference on Multimedia and Expo*. IEEE, 1–6.

Dan Miao, Jingjing Fu, Yan Lu, Shipeng Li, and Chang Wen Chen. 2014. High frame rate screen video coding for screen sharing applications. In *Proceedings of the 2014 IEEE International Symposium on Circuits and Systems*. IEEE, 2157–2160.

Microsoft. 2016a. Remote Desktop Protocol (RDP). Retrieved from http://msdn.microsoft.com/en-us/library/aa383015(v=vs.85).aspx.

Microsoft. 2016b. Using Performance Monitor. Retrieved from https://technet.microsoft.com/en-us/library/cc749115.aspx.

NCast. 2016. Homepage. Retrieved from http://www.ncast.com/.

OnLive. 2016. Homepage. Retrieved from http://www.onlive.com/.

Zhaotai Pan, Huifeng Shen, Yan Lu, Shipeng Li, and Nenghai Yu. 2013. A low-complexity screen compression scheme for interactive screen sharing. *IEEE Trans. Circuits Syst. Video Technol.* 23, 6 (2013), 949–960.

Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. 1998. Virtual network computing. *IEEE Trans. Internet Comput.* 2, 1 (1998), 33–38.

Robert W. Scheifler and Jim Gettys. 1986. The X window system. *ACM Trans. Graphics* 5, 2 (1986), 79–109.

Huifeng Shen, Yan Lu, Feng Wu, and Shipeng Li. 2009. A high-performanance remote computing platform. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications, 2009*. IEEE, 1–6.

Pieter Simoens, Paul Praet, Bert Vankeirsbilck, Jeroen De Wachter, Lien Deboosere, Filip De Turck, Bart Dhoedt, and Piet Demeester. 2008. Design and implementation of a hybrid remote display protocol to optimize multimedia experience on thin client devices. In *Proceedings of the Australian Telecommunication Networks and Applications Conference, 2008 (ATNAC'08).* IEEE, 391–396.

Software Systems Laboratory. 2016. THINC. Retrieved from http://systems.cs.columbia.edu/projects/thinc/.

StreamMyGame. 2016. Homepage. Retrieved from http://streammygame.com/smg/index.php.

StreamMyGame. 2012. Homepage. Retrieved from http://streammygame.com/smg/index.php.

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, and Jiun-In Guo. 2010. A remote thin client system for real time multimedia streaming over VNC. In *Proceedings of the 2010 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 992–997.

VideoLAN. 2016. x264. Retrieved from http://www.videolan.org/developers/x264.html.

TeamViewer. 2016. Homepage. Retrieved from http://www.teamviewer.com/.

TightVNC. 2016. Homepage. Retrieved from http://www.tightvnc.com/.

Shiqi Wang, Jingjing Fu, Yan Lu, Shipeng Li, and Wen Gao. 2012. Content-aware layered compound video compression. In *Proceedings of the 2012 IEEE International Symposium on Circuits and Systems*. IEEE, 145–148.

Wireshark. 2016. Homepage. Retrieved from https://www.wireshark.org/.

Wi-Fi Alliance. 2016. Wi-Fi CERTIFIED Miracast. Retrieved from http://www.wi-fi.org/discover-wi-fi/wi-fi-certified-miracast.

Jiyan Wu, Bo Cheng, Chau Yuen, Yanlei Shang, and Junliang Chen. 2015a. Distortion-aware concurrent multipath transfer for mobile video streaming in heterogeneous wireless networks. *IEEE Trans. Mobile Comput.* 14, 4 (2015), 688–701.

Jiyan Wu, Chau Yuen, Ngai-Man Cheung, Junliang Chen, and Chang Wen Chen. 2015b. Enabling adaptive high-frame-rate video streaming in mobile cloud gaming applications. *IEEE Trans. Circuits Syst. Video Technol*. 25, 12 (2015), 1988–2001.

Tao Zhang, Xun Guo, Yan Lu, Shipeng Li, Siwei Ma, and Debin Zhao. 2013. Arbitrary-sized motion detection in screen video coding. In *Image Processing (ICIP), 2013 20th IEEE International Conference on*. IEEE, 1943–1947.

Received August 2015; revised February 2016; accepted February 2016