Toward an Adaptive Screencast Platform: Measurement and Optimization

CHIH-FAN HSU, Academia Sinica, Taiwan
CHING-LING FAN, National Tsing Hua University, Taiwan
TSUNG-HAN TSAI, Academia Sinica, Taiwan
CHUN-YING HUANG, National Chiao Tung University, Taiwan
CHENG-HSIN HSU, National Tsing Hua University, Taiwan
KUAN-TA CHEN, Academia Sinica, Taiwan

The binding between computing devices and displays is becoming dynamic and adaptive, and screencast technologies enable such binding over wireless networks. In this article, we design and conduct the first detailed measurement study on the performance of the state-of-the-art screencast technologies. Several commercial and one open-source screencast technologies are considered in our detailed analysis, which leads to several insights: (i) there is no single winning screencast technology, indicating room to further enhance the screencast technologies, (ii) hardware video encoders significantly reduce the CPU usage at the expense of slightly higher GPU usage and end-to-end delay, and should be adopted in future screencast technologies, (iii) comprehensive error resilience tools are needed as wireless communication is vulnerable to packet loss, (iv) emerging video codecs designed for screen contents lead to better Quality of Experience (QoE) of screencast, and (v) rate adaptation mechanisms are critical to avoiding degraded QoE due to network dynamics. As a case study, we propose a non-intrusive yet accurate available bandwidth estimation mechanism. Real experiments demonstrate the practicality and efficiency of our proposed solution. Our measurement methodology, open-source screencast platform, and case study allow researchers and developers to quantitatively evaluate other design considerations, which will lead to optimized screencast technologies.

Categories and Subject Descriptors: H.5 [**Information Systems Applications**]: Multimedia Information Systems

General Terms: Design, Measurement

Additional Key Words and Phrases: Live video streaming, real-time encoding, performance evaluation, performance optimization

ACM Reference Format:
C. Hsu, 2015. Toward an Adaptive Screencast Platform: Measurement and Optimization *ACM Trans. Multimedia Comput. Commun. Appl.* 7, 3, Article 1 (August 2015), 23 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Wide adoption of heterogeneous computing devices, such as PCs, tablets, smart TVs, and smartphones, urges diverse ways for people to share photos, watch videos, and play games with their family and friends. Most people prefer to use larger or even

Author's address: C.-F Hsu, T.-H Tsai, K.-T. Chen, 128 Academia Road, Section 2, Nankang, Taipei 11574; email:hsuchihfan@gmail.com, zark912@iis.sinica.edu.tw, swc@iis.sinica.edu.tw; C.-F Fan, C.-H. Hsu, No. 101, Section 2, Kuang-Fu Road, Hsinchu, Taiwan 30013; email: yyytr7180@hotmail.com, chsu@cs.nthu.edu.tw; C.-Y. Huang, 1001 University Road Hsinchu, Taiwan 30010; email: chuang@cs.nctu.edu.tw. This work was supported in part by the Ministry of Science and Technology of Taiwan under the grants 103-2221-E-001-023-MY2, 102-2221-E-007-062-MY3, and 103-2221-E-019-033-MY2.

DOI: http://dx.doi.org/10.1145/0000000.0000000

multiple screens to share contents instead of limiting to a single screen. Ubiquitous displays are therefore gradually deployed in homes, schools, offices, shops, and even outdoor squares for experience sharing, educations, presentations,and advertisements. According to market research reports, the global flexible display market is expected to worth $3.89 billion by 2020, growing with high Compound Annual Growth Rate (CAGR) from 2014 to 2020 [Markets 2014]. Moreover, wireless networks have surged in popularity. Featuring displaying screen contents without cable connections to computing devices, wireless displays are expected to grow at a CAGR of 28.03% from 2012 to 2017 [Markets 2012]. These reports show that the *binding* between computing devices and displays becomes more dynamic, leading to flexible and diverse displaying experience.

Such dynamic binding of displays and computing devices can be done via *screencast*, which refers to capturing and sending the audiovisual streams from computing devices over networks to displays in real time. Screencast enables many usage scenarios, including playing multimedia contents over home networks, sharing desktops among colleagues over the Internet, and extending the small built-in displays of mobile and wearable devices over short-range wireless networks, such as Wi-Fi networks. Screencast has attracted serious attention from both the academia and industry because of its rich usage scenarios. For example, several open-source projects [Huang et al. 2014; Chandra et al. 2014] have been initiated to support screencast among wearable and mobile devices as well as desktops, tablets, and laptop computers. There are also proprietary and closed commercial products, such as AirPlay [AirPlay 2014], Chromecast [Chromecast Web Page 2014], Miracast [Miracast 2014], MirrorOp [MirrorOp Web Page 2014], and Splashtop [Splashtop 2014]. Although screencast is gradually getting deployed, the performance measurements on the state-of-the-art screencast technologies have not been rigorously considered in the literature. Current and future developers and researchers, therefore, have to resort to *heuristically* making the design decisions when building screencast technologies.

In this article, we first construct a real testbed to conduct the very first set of detailed experiments to quantify the performance of various screencast technologies under diverse conditions. The conditions are captured by several key parameters, including resolution, frame rate, bandwidth, packet loss rate, and network delay. The performance metrics include video bitrate, video quality, end-to-end latency, and frame loss rate. We evaluate five commercial products [AirPlay 2014; Chromecast Web Page 2014; Miracast 2014; MirrorOp Web Page 2014; Splashtop 2014] and an open-source solution [GamingAnywhere Web Page 2013]. The commercial products are treated as black boxes and general measurement methodologies are developed to compare their performance in different aspects. The open-source solution is a cloud gaming platform, called *GamingAnywhere* (GA) [Huang et al. 2014; GamingAnywhere Web Page 2013]. GA works for screencast, because cloud gaming is an extreme application of screencast, which dictates high video quality, high frame rate (in frame-per-second, fps), and low interaction latency [Chen et al. 2014]. Nevertheless, using GA as a general screencast technology leaves some room for optimization, e.g., it is well-known that popular video coding standards, such as H.264 [Wiegand et al. 2003], are designed for natural videos and may not be suitable to screen contents, also known as *compound images*, which are combinations of computer-generated texts and graphics, rendered 3D scenes, and natural videos [Zhu et al. 2014].

Fortunately, GA [Huang et al. 2014; GamingAnywhere Web Page 2013] is extensible, portable, configurable, and open. Therefore, developers and researchers are free to use GA for systematic experiments to make design decisions for optimized screencast. In this article, we design and conduct several such experiments, e.g., we integrate GA with emerging video codecs [x264 Web Page 2012; HEVC Test Model 2014] in or-

der to conduct a user study using a real screencast setup to quantify the gain of new video codecs. Our sample experiments reveal the potential of using GA for screencast research and developments. More importantly, we demonstrate how to measure the performance of screencast technologies, and how to quantify the pros/cons of different screencast technologies. The screencast measurement setup and design are, therefore, useful on their own rights, because they have not been reported in the literature. One common weakness of the state-of-the-art open-source screencast technologies, GA [Huang et al. 2014; GamingAnywhere Web Page 2013], lack of bitrate adaptation feature, which significantly degrades user experience. To address this limitation, we develop and implement rate adaptation mechanism in GA, which uses video packets to estimate the available bandwidth and adjusts the streaming rate accordingly. The enhanced GA incurs no network estimation overhead and reacts to network dynamic promptly. Evaluation results show that the proposed rate adaptation mechanism is effective and efficient.

The preliminary version of the current article was published in Hsu et al. [Hsu et al. 2015], which contains extensive measurement studies leading to various insights on optimization room of screencast technologies. The main findings are as follows.

— Considering diverse usage conditions and performance metrics, there is no single winning screencast technology, which indicates that there is still room to optimize the state-of-the-art screencast technologies in the coming years.
— Hardware video encoders significantly reduce the CPU usage at the screencast senders, and slightly increase the GPU usage and end-to-end latency; hence are suitable to screencast technologies.
— One way to better adapt to nonzero packet loss rate is to employ the reliable TCP protocol, but TCP protocol does not work well when network latency is long, which is inline with [Calagari et al. 2014]. Therefore, more comprehensive error resilience tools are desired.
— Screen contents are fairly different from natural videos, and adopting emerging video codecs designed for screen contents in screencast technologies leads to better Quality of Experience (QoE).

In the current article, we make the following new contributions on optimizing screencast technologies.

— We design a new, non-intrusive available bandwidth estimator for short-range Wi-Fi networks, which are the most popular networks used in screencast scenarios.
— We propose, implement, and evaluate a practical bitrate adaptation algorithm based on the proposed available bandwidth estimation.
— We conduct extensive experiments on the GA platform and the bitrate adaptation algorithm to show the merits and practicality of the proposed solutions.

The article is organized as follows. We review the literature in Section 2. We customize GA to be a more flexible platform for screencast in Section 3. This is followed by the detailed measurement methodology given in Section 4. We present the GA-based quantitative evaluations and user studies, and we discuss the design considerations for future screencast technologies in Section 5. Section 6 details a rate adaptation mechanism developed by us. Section 7 concludes this paper. In addition, due to the space limitation, we give the measurement results of the state-of-the-art screencast technologies in Appendix B.

2. RELATED WORK

In this section, we survey the literature in the following two directions: (i) screen sharing systems and (ii) performance measurements of screencast platforms. We summa-

Table I. The Comparisons Among Related Work

Related Systems Work	Real-time	Designed for Screencast	General Wi-Fi Support	802.11n Support	Rate Adaptation
Our Work	✓	✓	✓	✓	✓
[Chandra et al. 2014]	✓	✓	✓		
[Lin et al. 2012]	✓	✓	✓		
[Javadtalab et al. 2015]	✓				✓
[Hong et al. 2015]	✓				✓

rize the major differences between related work and our current work in Table I. We also describe prior work on available bandwidth estimation and rate adaptation in Appendix A (due to the space limitations).

2.1. Screen Sharing Systems

Early screen sharing systems, such as thin clients [Schmidt et al. 1999; Baratto et al. 2005] and remote desktops [Richardson et al. 1998; Cumberland et al. 1999], allow users to interact with applications running on remote servers. These screen sharing systems focus on developing protocols that efficiently update changed regions of the screens, rather than achieving high visual quality and frame rate, and thus are less suitable to highly-interactive applications, such as computer gaming as reported in Chang et al. [Chang et al. 2011]. Interested readers are referred to the surveys [Yang et al. 2002; Lai and Nieh 2006] on these screen sharing systems. To cope with such limitations, several companies offer video streaming based cloud gaming systems, such as OnLive [OnLive Web Page 2012], GaiKai [GaiKai Web Page 2012], and Ubitus [Ubitus Web Page 2014]. Huang et al. propose GamingAnywhere (GA) [Huang et al. 2014], which is the first open-source cloud gaming system. These cloud gaming platforms also work for screencast scenarios, although there are some optimization room to explore. Chandra et al. [Chandra et al. 2012; Chandra et al. 2014] develop DisplayCast that shares multiple screens among users in an Intranet, where the networking and computation resources are abundant. DisplayCast consists of several components, including the screen capturer, zlib-based video compression, and service discovery, but it lacks of rate control mechanisms. Wi-Fi displays are studied more recently, e.g., Zhang et al. [Zhang et al. 2015] conduct a measurement study on the power consumption of Wi-Fi displays. They model the power consumption of several components separately, including network transmission and codec operations. They then propose optimization mechanisms, such as: (i) adaptive video tail cutting for a better tradeoff between the graphics quality and energy consumption and (ii) energy efficient channel selection. Huang et al. [Huang et al. 2015] develop an open-source smart lens that allow users to preview the captured video on their smartphones via wireless networks. They use the QoE model to enhance the user experience and propose bandwidth estimation for single-hop Wi-Fi networks, but the estimation approach assumes (less realistic) static Wi-Fi networks.

We note that we choose GA [Huang et al. 2014] over DisplayCast [Chandra et al. 2012; Chandra et al. 2014] as the tool to assist design decisions for several reasons, including: (i) GA focuses on the more challenging audiovisual streaming, (ii) GA is arguably more extensible and portable, and (iii) GA has a more active community [GamingAnywhere Web Page 2013]. Nonetheless, readers who prefer to start from DisplayCast [Chandra et al. 2012; Chandra et al. 2014] can apply the lessons learned in this article to DisplayCast as well. Last, a preliminary version of the article is published in Hsu et al. [Hsu et al. 2015]. The current article contains elaborated discussion, additional experiments, and a new rate adaptation mechanism.

2.2. Performance Measurement of Screencast Platforms

The performance measurements of screen sharing and cloud gaming systems have been done in the literature. For example, Tolia et al. [Tolia et al. 2006] and Lagar-Cavilla et al. [Lagar-Cavilla et al. 2007] analyze the performance of VNC (Virtual Network Computing), and Claypool et al. [Claypool et al. 2012] and Chen et al. [Chen et al. 2014] study the performance of cloud games. The performance measurements on the state-of-the-art screencast technologies, however, have not received enough attention in the research community. He et al. [He et al. 2014] conduct a user study on Chromecast [Chromecast Web Page 2014] with about 20 participants to determine the user tolerance thresholds on video quality (in PSNR [Wang et al. 2001]), rendering quality (in frame loss rate), freeze time ratio, and rate of freeze events. The user study is done using a Chromecast emulator. Their work is different from ours in several ways: (i) we also consider the objective performance metrics, (ii) we use real setups for experiments, (iii) we consider multiple screencast technologies [AirPlay 2014; Chromecast Web Page 2014; Miracast 2014; MirrorOp Web Page 2014; Splashtop 2014; Huang et al. 2014], and (iv) our evaluation results reveal some insights on how to further optimize the screencast technologies. Moreover, following the methodologies presented in this paper, researchers and developers can leverage GA to intelligently make design decisions based on quantitative studies.

3. GAMINGANYWHERE AS A SCREENCAST PLATFORM

We investigate the key factors for implementing a successful screencast technology using GamingAnywhere (GA). GA may not be tailored for screencast yet, e.g., unlike powerful cloud gaming servers, the computing devices used for screencast may be resource-constrained low-end PCs or mobile/wearable devices, and thus screencast senders must be lightweight. Moreover, the screen contents of screencast are quite *diverse*, compared to cloud gaming: text-based contents in word processing, slide editing, and Web browsing applications are common in screencast scenarios. In this section, we discuss the customization of GA for screencast, which also enables researchers and developers to employ GA in performance evaluations to systematically make design decisions.

3.1. Support of More Codecs

GA adopts H.264 as its default codec. Currently the implementation is based on *libx264* and is accessed via the *ffmpeg/libav* APIs. However, we found that it is difficult to integrate other codec implementations into GA following the current design. For example, if we plan to use another H.264 implementation from Cisco [OpenH264 Web Page 2015], we have to first implement it as an ffmpeg/libav module, whereas integrating a new codec into ffmpeg/libav brings extra workload. In addition, ffmpeg/libav's framework limits a user to access advanced features of a codec. For example, libx264 allows a user to dynamically reconfigure the codec in terms of, e.g., frame rates, but currently it is not supported by ffmpeg/libav's framework. Therefore, we revise the module design of GA to allow implementing a codec without integrating the codec into the ffmpeg/libav framework. At the same time, we also migrate the RTSP server from ffmpeg to *live555*. As a result, GA now supports a wide range of video codecs that provide the required Session Description Protocol (SDP) parameters at the codec initialization phase. A summary of currently supported codecs and the associated SDP parameters are shown in Table II.

Table II. Supported codecs and the required SDP parameters

Codec	SDP Parameter	Description
Vorbis	configuration	Codec-specific configurations, such as codebooks
Theora	width	Video width
	height	Video height
	configuration	Codec-specific configurations, such as codebooks
H.264	sprop-parameter-sets	SPS (Sequence Parameter Set) and PPS (Picture Parameter Set)
H.265	sprop-vps	VPS (Video Parameter Set)
	sprop-sps	SPS (Sequence Parameter Set)
	sprop-pps	PPS (Picture Parameter Set)

3.2. Hardware Encoder

Screencast servers may be CPU-constrained, and thus we integrate a hardware encoder with GA as a reference implementation. We choose a popular hardware platform, Intel's Media SDK framework [Intel Web Page 2015], to access the hardware encoder. The hardware encoder is available on machines equipped with both an Intel i-series CPU (2^{nd} or later generations) and an Intel HD Graphics video adapter. To integrate the Intel hardware encoder into GA, we have to provide the sprop-parameter-sets, which contains the SPS (Sequence Parameter Set) and PPS (Picture Parameter Set) configurations of the codec. After the codec is initialized, we can obtain the parameters from the encoder context by retrieving SPS and PPS as codec parameters, i.e., calling MFXVideoENCODE_GetVideoParam function with a buffer of type MFX_EXTBUFF_CODING_OPTION_SPSPPS.

The Intel hardware encoder does not support many options. In addition to the setup of bitrate, frame rate, and GoP size, we use the following default configurations for the codec: main profile, best quality, VBR rate control, no B-frame, single decoded frame buffering, and sliced encoding. We also tried to enable intra-refresh feature, but unfortunately this feature is not supported on all of our Intel PCs. We notice that Intel's video encoder only supports the NV12 pixel format. Fortunately, it also provides a hardware-accelerated color space converter. Thus, we can still take video sources with RGBA, BGRA, and YUV420 formats; the video processing engine first converts the input frames into the NV12 pixel format and then passes the converted frames to the encoder. The CPU load reduction due to the hardware encoder is significant, which we will show in the experiments in Section 5.

3.3. Emerging Video Codecs

The revised GA design supports the emerging H.265 coding standard. To be integrated with GA, an H.265 codec implementation has to provide all the three required parameters (VPS, SPS, and PPS, as shown in Table II). We have integrated *libx265* [x265 Web Page 2014] and HEVC Test Model (HM) [HEVC Test Model 2014] with GA. HEVC supports several emerging extensions like Range Extension (REXT) and Screen Content Coding (SCC) [Zhu et al. 2014], which are designed for screencast or similar applications. We note that neither *libx265* nor HM are optimized for real-time applications in our experiments. Longer encoding time, however, is not a huge concern for now, as both implementations are emerging and we consider that the implementations will be optimized before actual deployments. Therefore, in Section 5, we evaluate these emerging codecs, and we focus on their achieved user experience (e.g., graphics quality) by encoding screen contents without considering their running time.

4. MEASUREMENT METHODOLOGY

In this section, we present the measurement methodology to systematically compare the state-of-the-art screencast technologies.

4.1. Screencast Technologies

The following five commercial screencast technologies are considered in our experiments.

— **AirPlay** is a proprietary protocol designed by Apple. AirPlay supports streaming audio, video, photos, and meta-data over wireless channels. Computers running iTunes and devices running iOS 4.2+ can be AirPlay senders, while AirPort Express and Apple TV can be AirPlay receivers. With iOS 4.3+, third-party apps may send compatible audiovisual streams over AirPlay. Besides, there is an open-source implementation [open-airplay: A collection of libraries for connecting over Apple's AirPlay protocol 2014] of the AirPlay protocol, which may turn any computer into an AirPlay receiver.
— **Chromecast** is a digital media player which is capable of directly streaming audiovisual contents via Wi-Fi. For screencast, a user can use Google Cast extension for Chrome, which uses WebRTC API to transmit screen contents from the Web browser or desktop to the Chromecast device.
— **Miracast** is a peer-to-peer wireless standard for screencast over Wi-Fi Direct. Miracast-compatible devices can serve as Miracast senders and receivers. Existing OS's with built-in Miracast support include Android 4.2 or later, BlackBerry 10.2, and Microsoft Windows 8.1. For streaming screens to a device that does not support Miracast, there are also Miracast adapters capable of rendering the screens through HDMI or USB ports.
— **MirrorOp** and **Splashtop** offer pure software solutions, which require the users to install proprietary applications at both the sender and receiver. Although MirrorOp and Splashtop use closed protocols, the developers offer the applications on multiple OS's, including Windows and Mac OS X.

In addition, the open-source GA is evaluated as a screencast technology as well.

4.2. Content Types

We study how the screencast technologies perform when streaming different types of contents. We consider 9 content types in the following 3 categories:

— **Gaming**: including first-person shooter, racing, and turn-based strategy games.
— **Movie/TV**: including dialogue movie scene, car chasing movie scene, and talk show.
— **Applications**: including Google street view browsing, slide editing, and Web surfing in Chrome.

For fair comparisons, we record the screens of different content types into 1280x720 videos. In particular, we extract one minute of representative video for each content type and concatenate them into a single 9-minute long video. We insert 2-second white video frames between any two adjacent content types to reset the video codecs. In this way, the measurement results collected from adjacent content types do not interfere one another.

4.3. Workload and Network Conditions

We also study how the screencast performance is affected under different workload settings and network conditions, which we believe impose direct and non-trivial impacts on screencast quality. Workload parameters are related to the quality of source videos,

Table III. The Considered Parameters

Parameter		Value		
Workload	Frame rate	15 fps	**30 fps**	60 fps
	Resolution	640x360	896x504	**1280x720**
Network	Bandwidth	4 Mbps	6 Mbps	**Unlimited**
	Delay	200 ms	100 ms	**0 ms**
	Packet loss rate	2%	1%	**0%**

Table IV. Screencast Technologies Considered

Technology		*AirPlay*	*Chromecast*	*GamingAnywhere*	*Miracast*	*MirrorOp*	*Splashtop*
Spec.	*Product*	Apple TV	Chromecast	GamingAnywhere	NETGEAR PTV3000	Sender/Receiver	Streamer/Client
	HW/SW	Hardware	Hardware	Software	Hardware	Software	Software
	Connectivity	AP	AP	AP	Wi-Fi Direct	AP/Internet	AP/Internet
	Protocol	TCP	UDP	UDP/TCP	UDP	TCP	TCP
Devices	*Sender*	MacBook Pro OS X 10.9.2	Chrome w/ Google Cast v14.305.0.0 on Win 8.1 Laptop	Win 8.1 Laptop	Win 8.1 Laptop	Sender v2.0.3.2 on Win 8.1 Laptop	Streamer v2.5.8.4 on Win 8.1 Laptop
	Receiver	Apple TV v6.1.1	Chromecast (firmware v16041)	Win 7 PC	NETGEAR Push2TV (firmware v2.4.46)	Receiver v0.2.11-4.win on Win 7 PC	Personal v2.4.5.2 on Win 7 PC

‡ If not otherwise specified, the PC computer is a ThinkCentre M92p, and the laptop computer is a ThinkPad X240.

including *frame rate* and *resolution*. We change the frame sampling rates to generate multiple videos, and set 30 fps as the default frame rate. We also vary the resolutions at 1280x720, 896x504, and 640x480. For the latter two cases, we place the video at the center of the (larger) screen without resizing it. This is because we believe image resizing would cause loss of details and therefore bias our results. As to network conditions, we use `dummynet`[1] to control the *bandwidth*, *delay*, and *packet loss rate* (packet loss) of the outgoing channel of senders. The default bandwidth is not throttled, the delay is 0 ms, and the packet loss rate is 0%.

In our experiments, a parameter of workload and network conditions is varied while all other parameters are fixed at their default values. The list of parameters is given in Table III, with the respective default values in boldface. For screencast technologies that support both UDP and TCP protocols, the default protocol is UDP.

4.4. Experiment Setup

There are several components in the experiment: a sender and a receiver for each screencast technology, and a Wi-Fi AP, which is mandatory for all technologies except Miracast (based on Wi-Fi Direct). The specifications of the screencast technologies are summarized in Table IV, and the detailed experiment setups are given below.

— **AirPlay.** The sender is a MacBook Pro running OS X 10.9.2, with a 2.4 GHz Intel Core i5 processor and 8 GB memory, while the receiver is an Apple TV. They are connected to the same Wi-Fi AP before the sender can discover, connect, and stream screens to the receiver.
— **Chromecast.** The sender is a Lenovo ThinkPad X240 notebook running Windows 8.1, with an 2.6 GHz Intel Core i5 processor and an 8 GB memory with a receiver that is a Chromecast dongle. The only way for screencasting using Chromecast is by Google Cast Chrome Extension. Once the sender is connected to the Wi-Fi AP, it can discover and connect to any available devices in the same Wi-Fi network.
— **Miracast.** We use the Lenovo notebook as the sender. For the receiver, we use a NETGEAR Push2TV Miracast adapter. Miracast is based on Wi-Fi Direct and supported by Windows 8.1. As long as the receiver is placed within the wireless transmission

[1] `dummynet` is a network emulation tool, initially designed for testing networking protocols. It has been used in a variety of applications, such as bandwidth management.

range of the sender, Windows 8.1 provides a simple user interface for screencasting the sender's desktop to the receiver.

— **MirrorOp** and **Splashtop.** The Lenovo notebook serves as the sender, while a PC running Windows 7, with an Intel Core i7 processor serves as the receiver. To use these two services, a user needs to create an account, and run the sender and receiver programs on the respective machines. Once both machines are logged in, they can discover and connect to each other.

In addition, experiments on GA are also conducted using the same setup as MirrorOp and Splashtop. We note that there may be multiple implementations for certain technologies, e.g., Miracast, but we cannot cover all the implementations in this work. We pick a popular implementation for each technology, and detail the measurement methodology so that interested readers can apply the methodology to other implementations.

4.5. Performance Metrics

We measure the following performance metrics that are crucial to screencast user experience.

— **Bitrate**. The average amount of data per second transmitted from the sender to receiver, which is important because the wireless spectrum and total bandwidth is limited and shared by all applications/users.

— **End-to-end latency** (latency). The time difference between each video frame is rendered at the sender and at the receiver, which is especially important for interactive applications. The user experience also drops if the latency jitter (i.e., the variation of latency) is high.

— **Frame loss rate** (frame loss). The fraction of video frames that are not rendered at the receiver, which greatly affects the viewing experience.

— **Video quality** (quality). The video quality rendered at the receiver compared to the original video captured at the sender. We use PSNR [Wang et al. 2001] and SSIM [Wang et al. 2004] to quantify the video quality observed at the receiver.

When presenting the measurement results, 95% confidence intervals of the averages are given as error bars in the figures whenever applicable.

4.6. Experiment Procedure

For each technology, we first connect the sender and receiver, play the video with diverse content types at the sender, and measure the four performance metrics. We repeat the experiment ten times with each configuration (i.e., workload and network parameters). To facilitate our measurements, we have added a unique color bar at the top of each frame of the source contents as their frame id, which can be programmatically recognized (c.f., Figure 1(c)).

To measure the bitrate used by the screencast technologies, we run a packet analyzer at the sender to keep track of the outgoing packets during the experiments. For measuring the video quality, we direct the HDMI output of the receiver to a PC, which is referred to as the *recorder*. The recorder PC is equipped with an Avermedia video capture card to record the videos. To quantify the quality degradation, each frame of the recorded video is matched to its counterpart in the source video, using the frame id. Last, we calculate the PSNR and SSIM values as well as the frame loss rate by matching the frames. This setup is illustrated in Figure 1(a).

To measure the user-perceived latency, we direct the rendered videos of both the sender and receiver to two side-by-side monitors via HDMI (for the sake of larger displays). We then set up a Canon EOS 600D camera to record the two monitors at the

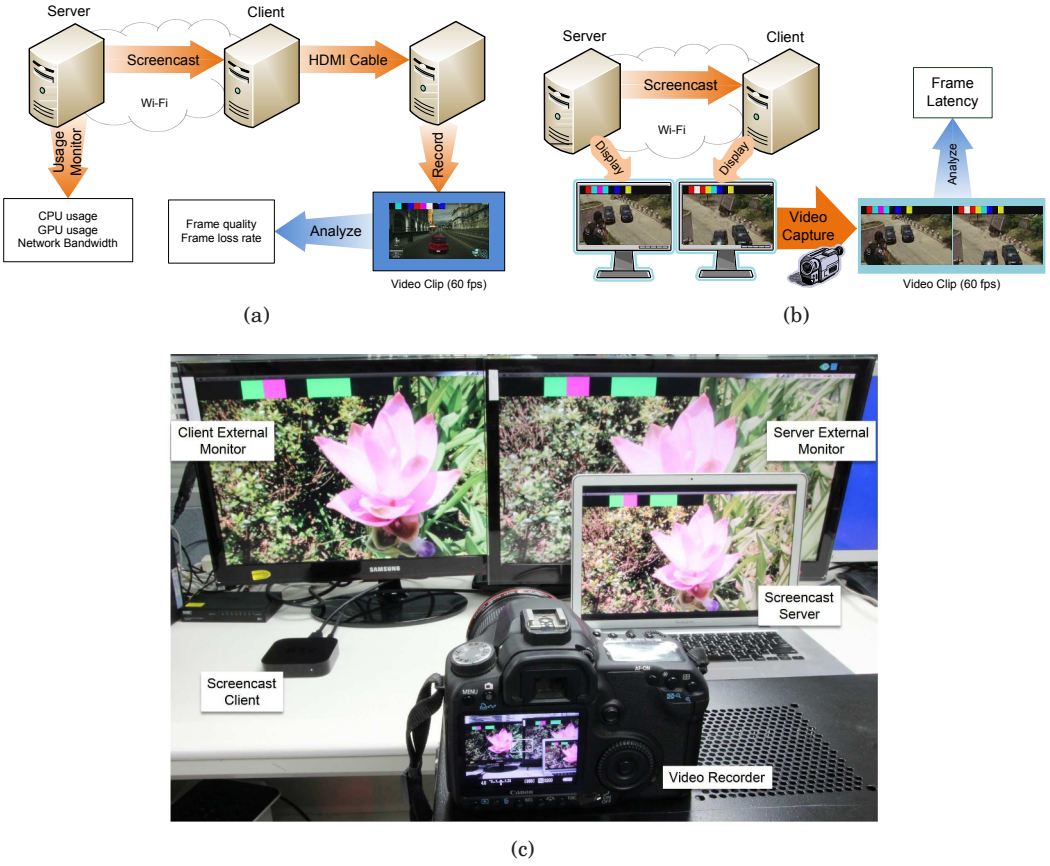


Fig. 1. Experiment setup for: (a) bitrate/video quality and (b) latency; (c) actual testbed for latency measurements in our lab.

same time, as shown in Figure 1(c). To capture every frame rendered on the monitors, we set the recording frame rate of the camera to 60 fps, which equals to the highest frame rate in our workload settings. The recorded video is then processed to compute the latency of each frame, by matching the frames based on frame ids and by comparing the timestamps when the frame is rendered by the sender and receiver. The setup is shown in Figure 1(b).

Last, we note that we had to repeat each experiment twice: once for bitrate and video quality (Figure 1(a)), and once for the latency (Figure 1(b)). This is because each receiver only has a single HDMI output, but the two measurement setups are quite different. Fortunately, our experiments are highly automated in a controlled environment, thus our experiment results are not biased. The actual testbed is shown in Figure 1(c).

5. DESIGN CONSIDERATIONS

Our performance evaluations on screencast technologies given in Appendix B lead to two main observations: (i) screencast technologies all have advantages and disadvantages and (ii) deeper investigations to identify the best design decisions are crucial. In this section, we present a series of GA-based experiments to analyze several design considerations. We emphasize that our list of design considerations is not ex-

hausted, and readers are free to leverage open-source screencast technologies such as GA [Huang et al. 2014] and DisplayCast [Chandra et al. 2012; Chandra et al. 2014] for similar studies.

5.1. Software vs. Hardware Encoding

We study the implications of switching from software video encoder to hardware encoder in GA, and we compare their performance against the commercial screencast technologies. We use the experiment setup presented in Table IV, and we stream the 9 minutes 18 seconds video using the default settings given in Table III. We consider three performance metrics: CPU usage, GPU usage, and end-to-end latency. For CPU/GPU usage, we take a sample every second, and the end-to-end latency is calculated for every frame. Then, we report the average CPU/GPU usages incurred by individual screencast technologies in Figure 2. In this figure, GA and GA (HE) represent GA with software and hardware video encoders, respectively. Moreover, the numbers above the points are the average end-to-end latency.

We draw several observations from this figure. First, hardware encoder dramatically reduces the CPU usage of GA: less than 1/3 of CPU usage is resulted compared to software encoder. Second, upon using the hardware encoder, GA results in lower CPU usage, compared to MirrorOp, Chromecast, and Splashtop. While AirPlay and Miracast consume less CPU compared to GA with hardware encoder, they achieve inferior coding efficiency as illustrated in Figures 13(a) and 13(d). More specifically, although AirPlay and Miracast incur much higher bitrate, their achieved video quality levels are no better than other screencast technologies. We conclude that AirPlay and Miracast trade bandwidth usage (coding efficiency) for lower CPU load, so as to support less powerful mobile devices, including iOS and BlackBerry. Third, both GA and GA (HE) achieve very low latency: up to 18 times lower than some screencast technologies. Such low end-to-end latency comes from one of the design decisions of GA, i.e., zero playout buffering [Huang et al. 2014], as a cloud gaming platform, which is crucial for highly interactive applications during screencasting. We note that GA (HE) leads to 26 ms longer latency than GA, which is due to the less flexible frame buffer management mechanism in Intel's Media SDK framework [Intel Web Page 2015], which prevents us from performing more detailed latency optimization that are done by us in ffmpeg/libav.

In summary, the hardware video encoder largely reduces the CPU usage, while slightly increases the GPU usage and end-to-end latency. It is therefore quite worthy to consider when developers are building future screencast technologies.

5.2. Comparison of Transport Protocols

The experiment results given in Appendix B indicate that GA is vulnerable to nontrivial packet loss rate. This may be attributed to the fact that GA employs the UDP protocol by default, and a quick fix may be switching to the reliable TCP protocol. Therefore, we next conduct the experiments using GA with the UDP and TCP protocols. We adopt the default settings as above and vary the network bandwidth and delay settings. We consider 3 performance metrics: end-to-end latency, frame loss rate, and video quality in PSNR and report the average results over the 9 minutes 18 seconds video in Figure 3, where two corresponding points (those of UDP versus TCP) are connected by dashed lines. The annotations next to the dash lines are network conditions, and the numbers next to the points are the PSNR values representing the resulting video quality rendered at the client.

We make the following observations. When the network delay is low, TCP always leads to lower frame loss rate: 2% difference is observed. However, when the delay is longer, say ≥ 100 ms, TCP results in even higher frame loss rate, which can be

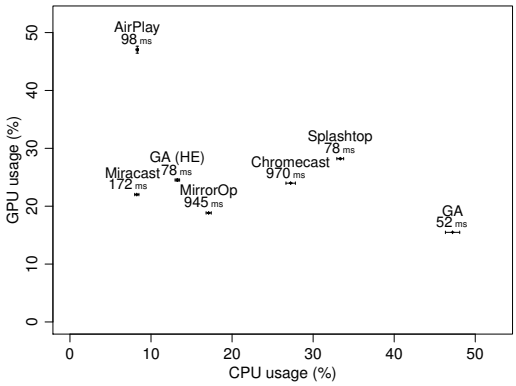


Fig. 2. Hardware encoder reduces the CPU usage of GA. The numbers below screencast technologies represent end-to-end latency, and the length of horizontal (and vertical) line segments represent the 95% confidence intervals of mean CPU usage (and GPU usage), respectively.

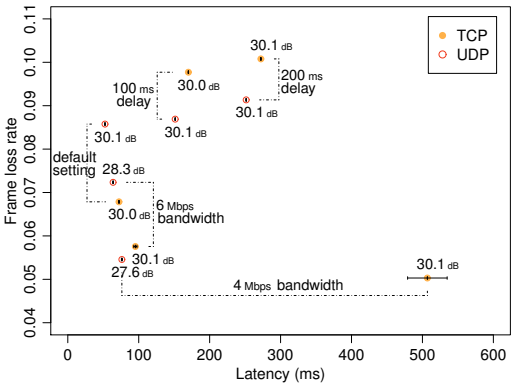


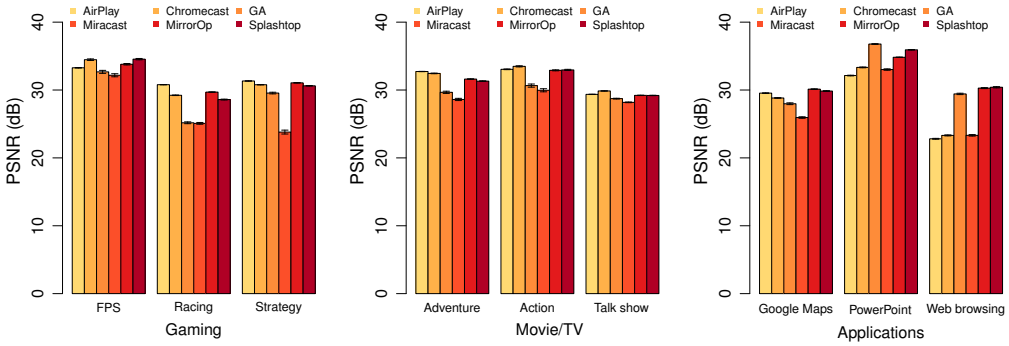
Fig. 3. The impacts of TCP and UDP protocols. The numbers below symbols represent graphical quality in PSNR. Each pair of experiments with identical settings (except for the transport protocol) is connected by dashed lines, and the length of horizontal (and vertical) line segments represent the 95% confidence intervals of mean latency (and frame loss rate), respectively.

attributed to the longer delay caused by TCP, making more packets miss their playout deadlines and are essentially useless. Moreover, TCP usually incurs slightly longer end-to-end latency, except when we set the bandwidth to 4 Mbps, which leads to a much longer latency. On the other hand, under 4 Mbps bandwidth, UDP suffers from higher packet loss rates and thus leads to lower video quality, in particular, UDP results in 2.5 dB lower video quality than TCP.

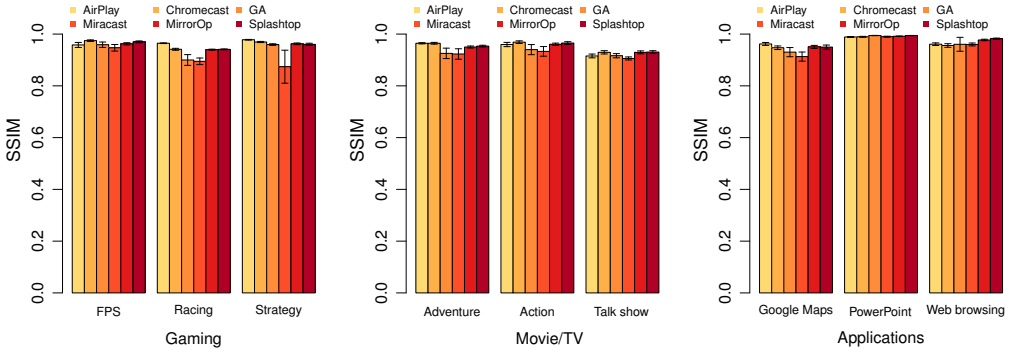
In summary, Figure 3 depicts that the TCP protocol may be used as a basic error resilience tool of GA, but it does not perform well when network delay is longer and when the network bandwidth is not always sufficiently provisioned. This is inline with the well-known limitation on TCP: it suffers from degraded performance in fat long pipes [Kleinrock 1992], due to the widely adopted congestion control algorithms. Hence, more advanced error resilience tools are desired.

5.3. Comparison of Video Codecs

Under the default settings, we report the achieved video quality in Figure 4. This figure shows that MirrorOp and Splashtop achieve good video quality for all content types, while other screencast technologies all suffer from degraded video quality for some content types. For example, AirPlay leads to inferior PSNR for Applications, and GA results in lower PSNR/SSIM for Movie/TV. Furthermore, we observe that several screencast technologies suffer from lower video quality, especially in PSNR, for some content types. For example, for Web browsing, AirPlay, Chromecast, and Miracast lead to ~ 22 dB in PSNR, which can be caused by the different characteristics of Web browsing videos: the sharp edges of texts are easily affected by the ringing artifacts in the standard video codecs, such as H.264 [Wiegand et al. 2003]. Recently, Screen Content Coding (SCC) has been proposed [Zhu et al. 2014] as an extension to the High Efficiency Video Coding (HEVC) standard. SCC is built on top of the Range Extension



(a)



(b)

Fig. 4. Video quality achieved by different screencast technologies on diverse content types, in: (a) PSNR and (b) SSIM.

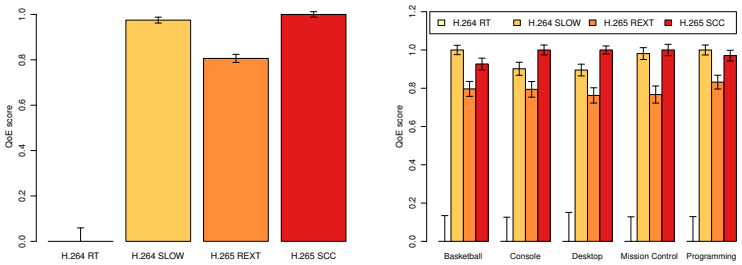
(REXT) of the HEVC standard. REXT expands the supported image bit depths and color sampling formats for high-quality video coding.

In the following, we conduct a separate study to investigate the benefits of the emerging video coding standards: *H.265 REXT*, which is designed for nature videos, and *H.265 SCC*, which is designed for screen contents. For comparisons, we also include x264 with two sets of coding parameters: the real-time parameters used by GA, which is denoted as *H.264 RT*, and the high-quality parameters with most optimization tools enabled, which is denoted as *H.264 SLOW*. In particular, we select 5 screen content videos: BasketballScreen (2560x1440), Console (1920x1080), Desktop (1920x1080), MissionControl3 (1920x1080), and Programming (1280x720) from HEVC testing sequences for SCC.We encode the first 300 frames of each video using the four codecs at 512 kbps on an AMD 2.6 GHz CPU. Table V gives the resulting video quality, which reveals that, H.264 RT results in inferior video quality. With optimized tools enabled, H.264 SLOW leads to video quality comparable to H.265 REXT, which is outperformed by H.265 SCC by up to ~ 5 dB. This table shows the potential of the emerging H.265 video codecs.

We next conduct a user study to get the QoE scores achieved by different codecs. We randomly pick 40 frames from each video, and extract these frames from the reconstructed videos of the 4 codecs. We save the chosen frames as lossless PNG images,

Table V. The Resulting Video Quality in PSNR (dB)

Video	H.264 RT	H.264 SLOW	H.265 REXT	H.265 SCC
BasketballScreen	15.93	33.12	30.83	33.08
Console	15.18	19.35	21.63	22.37
Desktop	13.57	23.08	23.15	28.06
MissionControl3	16.63	34.70	30.46	33.36
Programming	17.29	31.46	31.67	33.36



(a) (b)

Fig. 5. QoE scores of achieved by different codecs: (a) overall scores and (b) individual videos.

and create a Web site to collect inputs from general publics. We present images encoded by two random codecs side-by-side, and ask viewers to do pair comparisons. We conducted the user study in September 2014, including 126 paid subjects, who completed 180 sessions with 7,200 paired comparisons, and the total time the subjects spent in the study is 27.2 hours. We compute the QoE scores using the Bradley-Terry-Luce (BTL) model [Wu et al. 2013] and normalize the scores to the range between 0 (worst experience) and 1 (best experience). We plot the overall average and per-video QoE scores in Figure 5. We make a number of observations on this figure. First, H.265 SCC outperforms H.265 REXT for all videos, demonstrating the effectiveness of H.265 SCC. Second, the H.264 RT codec results in very low QoE scores, while the H.264 SLOW codec results in video quality comparable to H.265 SCC. However, a closer look at the H.264 SLOW reveals that the encoding speed can be as low as < 1 fps, turning it less suitable to real-time applications such as screencasting.

In summary, Figures 4 and 5 depict that different contents require different video codecs, e.g., the emerging H.265 SCC codec is more suitable to screen contents, comprising texts, graphics, and nature images.

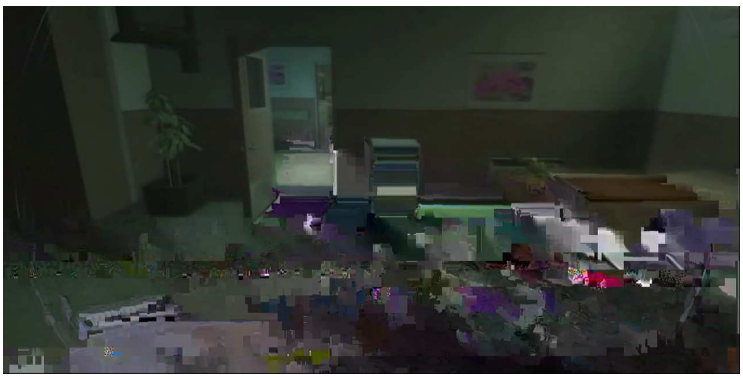


Fig. 6. Sample blocking features observed in Miracast.

Table VI. Negative Impacts due to Imperfect Rate Adaptation (Sample Results at 1 Mbps)

Technology	Slow Responsiveness	Blocking Features	Frozen Screens (a few secs)	Consecutive Lost Frames	Disconnections
AirPlay	✓				
Chromecast	✓		✓	✓	✓
GA	✓	✓		✓	
Miracast	✓	✓		✓	
MirrorOp	✓		✓	✓	
Splashtop	✓			✓	

5.4. Necessity of Rate Adaptation

We repeat the experiments under different bandwidth settings: between 4 Mbps and 1 Mbps. We observe various negative impacts, including slow responsiveness, blocking features, frozen screens, consecutive lost frames, and disconnections (between the sender and receiver) for some screencast technologies once the bandwidth is lower than 3 Mbps. Figure 6 gives a sample screen with serious artifacts. Table VI presents the observed negative impacts under 1 Mbps bandwidth, which clearly shows that most screencast technologies suffer from at least two types of negative implications. AirPlay performs the best, which is consistent with our observation made in Appendix B.2 : AirPlay actively adapts its bitrate to the changing bandwidth. On the other hand, although Chromecast and Miracast also actively adapt their bitrate, they do not survive under low bandwidth. Furthermore, (ordinary) GA, MirrorOp, and Splashtop do not adapt their bitrate to the bandwidth at all, and thus sometimes they under-utilize the available bandwidth. Moreover, they may sometimes send excessive traffic and suffer from unnecessary packet loss and quality degradation. These observations clearly manifest that more carefully-designed rate adaptation mechanism is highly demanded for screencast technologies. Thus, we develop and implement rate adaptation mechanism in open-source GA and conduct experiments to evaluate the performance and efficiency of our proposed mechanism. The details are given in the next section.

6. AN ADAPTIVE SCREENCAST PLATFORM

The crux of adaptive screencasting is an accurate and efficient available bandwidth estimation algorithm. However, most existing algorithms are not suitable for screencast platforms because they send probing packets that generate extra traffic. This is not a problem for quickly estimating the available bandwidth before streaming commences, but less suitable to continuous bandwidth estimation. To better understand the overhead, we configure Iperf [Iperf Web Page 2015] and WBest+ [Farshad et al. 2014] to estimate the available bandwidth for 2 minutes. We then plot the probing traffic overhead in Figure 7. Iperf (OT) and WBest+ (OT) only send probing packets one time at the beginning of the streaming, and thus are not able to dynamically estimate the available bandwidth for rate adaptation. On the other hand, we configure Iperf (P) and WBest+ (P) to send probing packets every 20 seconds for adaptive estimation. This figure shows that Iperf results in much higher probing overhead, especially for dynamic estimation, which prevents screencast platforms from sending videos at higher bitrates. WBest+ introduces about 3.4 Mbits probing overhead for each estimation. In contrast, we develop a non-intrusive and flexible algorithm, inspired by WBest+, that leverages video packets as probing packets. Indicated as GA in this figure, our algorithm (detailed below) leads to no probing overhead.

6.1. Overview on Adaptation Supports

Figure 8 gives a high-level overview of our proposed rate adaptation mechanism for GA [Huang et al. 2014] screencast platform. A typical screencast platform consists of several components that are presented below. The server contains: (i) a frame buffer that is the display memory holding the screen contents, (ii) a video capturer that repeatedly retrieves the screen contents from the frame buffer, (iii) a video encoder that

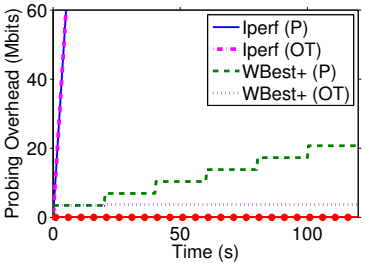


Fig. 7. Overhead of Iperf and WBest over non-intrusive GA.

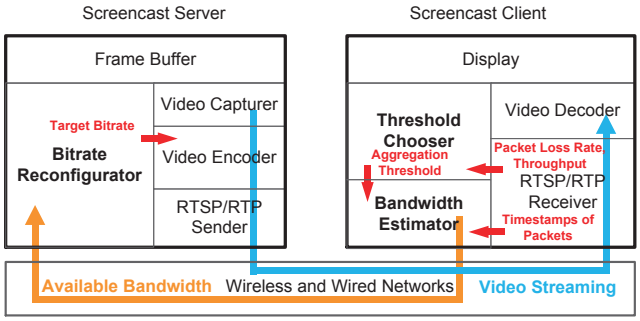


Fig. 8. The overview of the components for rate adaptation.

encodes the captured video, and (iv) an RTSP/RTP sender that sends the coded video to the client via wired or wireless networks. The client contains: (i) an RTSP/RTP receiver that receives the coded video from the networks, (ii) a video decoder that decodes the coded video to screen contents, and (iii) a display to render the screen contents to user. These 7 components are standard to real-time video streaming applications, including screencast platforms.

To support the rate adaptation mechanism, we add 3 new software components in the screencast server/client. They are highlighted by the bold font in Figure 8. These components are: (i) *bandwidth estimator*, (ii) *threshold chooser*, and (iii) *bitrate reconfigurator*. Their interactions are as follows. When the screencast server starts, the screen contents are captured by the video capturer and sent to the video encoder, which is then streamed to the screencast client by the RTSP/RTP sender. Upon receiving video frames from the server for a period of time T, the RTSP/RTP receiver at the client calculates the current packet loss rate ρ and throughput θ and then reports them to the threshold chooser. The threshold chooser selects the *aggregation threshold* a, which is the percentile of inter-arrival times of received packets in T. Adjacent packets with inter-arrival time smaller than the aggregation threshold are considered as parts of the same aggregated frame in 802.11n. The bandwidth estimator uses the timestamps of video packets sent from the RTSP/RTP receiver and the empirical chosen percentile a sent from the threshold chooser to estimate the available bandwidth $\bar{c}_a$ by calculating the results of the packet size over the dispersion time among clustered adjacent probing packets. After that, the bandwidth estimator sends the estimated bandwidth to the server for the bitrate reconfigurator that maps the estimated bandwidth to the target encoding bitrate b. The target encoding bitrate is then sent to the video encoder. We present the detailed designs of the added components in the next section.

6.2. Design of Major Adaptation Components

We build our bandwidth estimator on top of the state-of-the-art WBest+ [Farshad et al. 2014], which is proposed to estimate available bandwidth for 802.11n. However, they did not specify the actual value of the aggregation threshold, and we address this limitation. In particular, we first create a Cumulative Distribution Function (CDF) of inter-arrival times, and then determine the percentile a to best match the average bandwidth to the ground truth given by Iperf. We conduct experiments to find aggregation threshold a in our labs with live interference traffic. We consider the following two scenarios: (i) both the server and the client are launched on End Devices (EDs), which are connected via a Wi-Fi Access Point (AP), and (ii) the server and the client

are launched on an ED and an AP, respectively. We vary the distances between the server and the client to be at 1, 2, 4, and 8 m. In our experiments, the server streams captured video to the client for 3 minutes at each distance, and we compute the aggregation thresholds in Table VII. This table shows that the aggregation thresholds are diverse and dynamic, e.g., the difference can be higher than 10%. Therefore, the threshold chooser that dynamically adjusts the aggregation threshold is proposed to enhance our bandwidth estimator. The bitrate reconfigurator then reconfigures the encoding bitrate, which is proportional to the estimated bandwidth.

We also conduct experiments to determine the best system parameters: ρ_f and b_f for adjusting the aggregation threshold and the encoding bitrate, respectively. We vary $\rho_f, b_f \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$ and measure the throughput and the packet loss rate. We found that the difference between the highest and the lowest throughput and the packet loss rate under different b_f values are only 1.9% and 1.5%, respectively. Therefore, we choose the median, 0.7, as our default b_f value. Similarly, the difference between the highest throughput and the lowest throughput under different ρ_f values is only 2.7%. Thus, we choose 0.7 as the default ρ_f value for its minimum packet loss rate of 3.11% among all considered ρ_f values.

Table VII. Different Thresholds Under Different Environments

Distance	1 m	2 m	4 m	8 m
ED-ED	32%	37%	40%	43%
ED-AP	33%	34%	35%	33%

Threshold chooser. The threshold chooser considers the packet loss rate and throughput to choose the aggregation threshold, e.g., high packet loss rate indicates that the aggregation threshold should be risen for lower available bandwidth estimation. In contrast, if the throughput is high and there is no packet loss, the aggregation threshold should be reduced. In our implementation, the aggregation threshold a is adjusted with step size δ once every T seconds as follows. We increase the aggregation threshold if the current packet loss rate is higher than the average packet loss rate ρ_{avg} multiply a system parameter ρ_f. If not otherwise specified, we let $\rho_f = 0.7$ according to our preliminary experiment. We note that the actual bitrate of coded video is typically different from the target encoding bitrate, depending on the amount of information to code and inaccuracy of the rate control. Therefore, we introduce a new parameter θ_f to accommodate such reality in the following way. We reduce the aggregation threshold if: (i) the current packet loss rate is lower than the minimum packet loss rate ρ_L and (ii) the achieved throughput is higher than the factor of target bitrate $\theta_f \times b$. We set $\rho_L = 0\%$, $\theta_f = 95\%$ if not otherwise specified.

Bandwidth estimator. Let s_p and t_p represent the packet size and the received time of packet p. The packet dispersion time, which is the inter-arrival time between two adjacent received packets, can be represented as $t_p - t_{p-1}$ and the instantaneous bandwidth is calculated as $s_p/(t_p - t_{p-1})$. We first sort all the packet dispersion times collected in T seconds in the ascending order, and use a to cluster the received packets for estimating the available bandwidth $\bar{c}_a$.

Bitrate reconfigurator. The bitrate reconfigurator obtains the available bandwidth $\bar{c}_a$ from the bandwidth estimator and then reconfigures the encoder with encoding bitrate b, which is a function of $\bar{c}_a$. We let $b = b_f \bar{c}_a$ to be conservative according to our preliminary experiment if not otherwise specified.

Pseudocode of our algorithm. Algorithm 1 gives the pseudocode of our rate adaptation algorithm. Lines 3–7 belong to the threshold chooser. Line 3 checks whether the achieved throughput is high and the packet loss rate is low. If it passes, line 4 reduces

Algorithm 1 Rate Adaptation Algorithm

1: **for** every T seconds **do**
2: Compute ρ and θ
3: **if** $\theta > \theta_f \times b$ and $\rho \leq \rho_L$ **then**
4: $a = a - \delta$
5: **else if** $\rho > \rho_{avg} \times \rho_f$ **then**
6: $a = a + \delta$
7: **end if**
8: Compute $\bar{c}_a$
9: $b = b_f \times \bar{c}_a$
10: Reconfigure video encoder with b
11: **end for**

the aggregation threshold for higher estimated available bandwidth. Line 5 checks if the current packet loss rate is high. If it passes, line 6 increases the aggregation threshold. Line 8 is part of the bandwidth estimator, which computes the available bandwidth as the average value of all $s_p/(t_p - t_{p-1})$, where $(t_p - t_{p-1}) > a$. The server then reconfigures the encoding bitrate in line 9. Instead of dynamically selecting the aggregation threshold for bandwidth estimation and encoding bitrate, Hong et al. [Hong et al. 2015] choose the mean capacity among all packets as the effective capacity, while Huang et al. [Huang et al. 2015] model the percentile capacity for single-hop bandwidth estimation. Compared to the current article, they are less flexible in diverse and dynamic wireless networks. Javadtalab et al. [Javadtalab et al. 2015] estimate the available bandwidth changes during video streaming using weighted inter-arrival time of video packets. Their method is not tailored for 802.11n or other short-range wireless networks, which is the most common environment for screencast.

6.3. Experiments

We have implemented our proposed components for rate adaptations in GA server/client. The AP is build on a Linux box using a wireless adapter with Atheros chip. Our proposed system is meant to be used in real world. Thus, we conduct experiments in the lab, which is one of the typical scenarios for screencast. Our experiments last within 2 to 3 hours, and we believe that the wireless network conditions do not fluctuate too much within such a short time period. In our experiments, both the server and client run on Thinkpad Linux laptops. We randomly choose a content type from each of the three categories (see Section 4.2) and play the videos on the server. We save the videos at both server and client into YUV files, which are used for quality assessment. We run Wireshark at the server for network related results, such as bitrate, video quality, frame loss rate, *estimated bandwidth*, *throughput*, and *packet loss rate*. The estimated bandwidth is the available bandwidth reported by the bandwidth estimator. The throughput is the number of bits per-second received at the client. The packet loss rate is the fraction of lost packets. In addition, we derive PSNR by first comparing the pre-inserted color bars in video frames to match the frames at the server- and client-side YUV files. For lost frames, we replay the previously decoded frames at the client side to mimic the basic error concealment approach. We then compute the PSNR values accordingly.

Functionality of our adaptation algorithm. We fix the distance between the server and the client at 4 meters, which is inline with one of the most popular screencast scenarios: streaming presentation slides over a Wi-Fi network with a projector in a conference room. The server plays different content types and streams captured videos to the client for 3 minutes in each experiments. Figure 9 shows the sample results from the Movie/TV category (Talk Show), which is more complex. We first plot

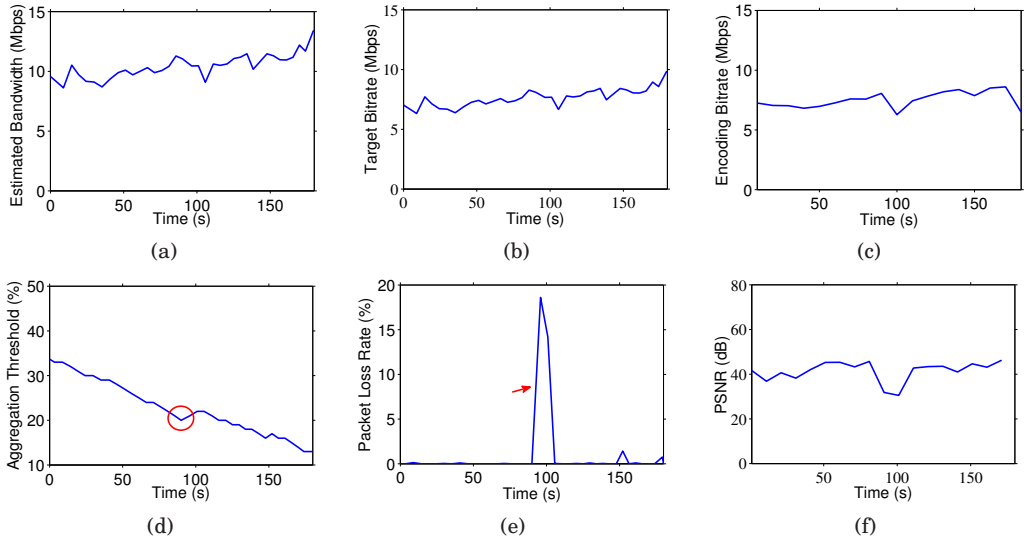


Fig. 9. Sample results from Movie/TV: (a) estimated bandwidth, (b) target bitrate, (c) encoding bitrate, (d) aggregation threshold, (e) packet loss rate, and (f) quality in PSNR.

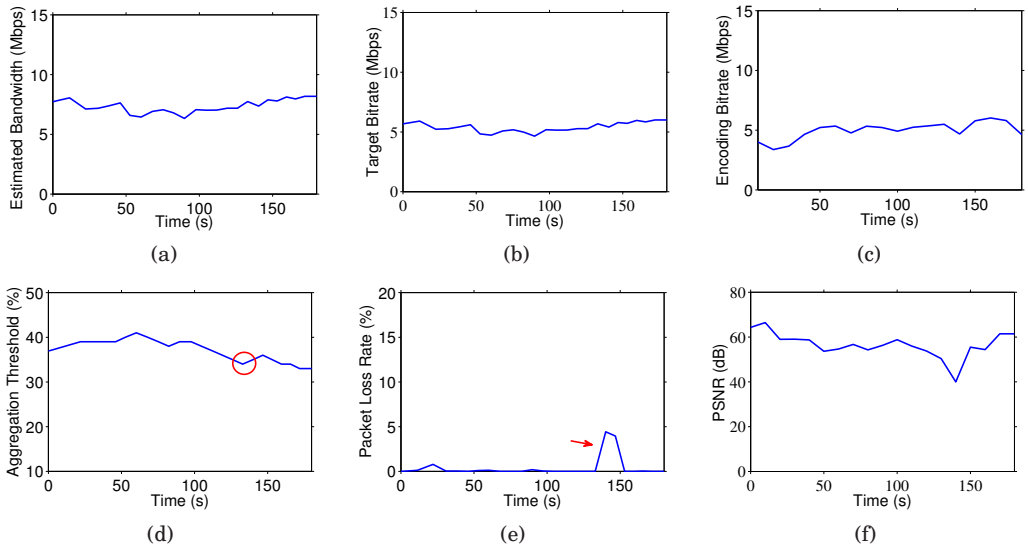


Fig. 10. Sample results from Application: (a) estimated bandwidth, (b) target bitrate, (c) encoding bitrate, (d) aggregation threshold, (e) packet loss rate, and (f) quality in PSNR.

the estimated available bandwidth over time from Movie/TV in Figure 9(a). The target encoding bitrate is then decided by a function of $\bar{c}_a$, and the values are illustrated in Figure 9(b). Figure 9(c) shows the actual encoding bitrates at the server. The average encoding bitrate of Movie/TV is about 7.3 Mbps, which is close to the target bitrate. Figure 10 gives similar results from the Application category (PowerPoint). However, we observe that its average encoding bitrate is 0.6 Mbps lower than the target bitrate. A deeper investigation indicates that it is because Applications have lower complexity and are easier to be compressed.

The dynamics of the aggregation threshold. Figure 9(d) plots the trend of the adaptive aggregation threshold over time. In our experiments, the initial aggregation threshold is empirically set to 0.36. The aggregation threshold in Figure 9(d) tends to be continuously decreased until $t = 90$ s. The aggregation threshold continuously decreases because the throughput approaches the target bitrate and the packet loss rate is zero. This indicates that the aggregation threshold should be reduced for higher bandwidth estimation. Figure 9(e) plots the packet loss rate over time from Movie/TV. Comparing Figures 9(d) and 9(e), the threshold chooser is aware of high packet loss rate at $t = 90$ s, and increases the aggregation threshold accordingly. Figures 10(d) and 10(e) show similar behavior at $t = 135$ s. The threshold chooser adjusts the aggregation threshold to accommodate to the current network environments, and helps to quickly recover from the video quality degradation due to high packet loss rate.

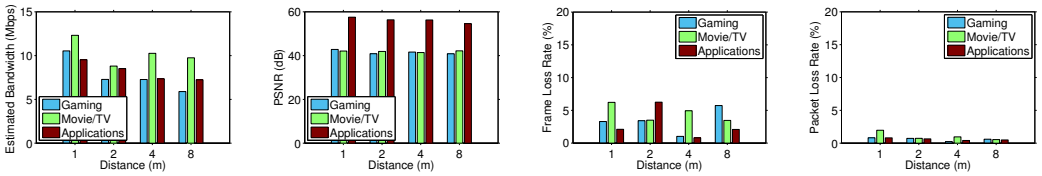
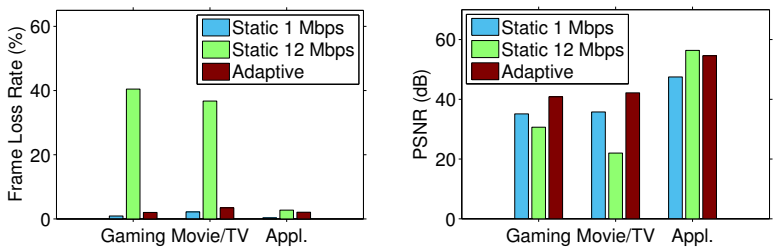


Fig. 11. Performance at different distances: (a) estimated bandwidth, (b) quality in PSNR, (c) frame loss rate, and (d) packet loss rate.

Implications of distances on performance. Screencasting is more likely to happen in conference rooms, homes, and classrooms. Thus, we consider the distance between the server and the client at 1, 2 , 4, and 8 m. We plot the average of the estimated available bandwidth with different content types at diverse distances in Figure 11(a). This figure shows that the estimated available bandwidth has a trend of decreasing as the distance increases. This is consistent with our intuitions. Figure 11(b) plots the video quality at different distances. This figure shows the scalability of our proposed rate adaptation: the PSNR values are always higher than 40 dB[2]. Besides, the PSNR values of Applications are higher than the other 2 categories by about 15 dB on average. This can be attributed to the lower complexity of Applications. Figures 11(c) and 11(d) report the frame loss rate and the packet loss rate, respectively. We note that frame loss rate is usually higher than packet loss rate. This is because each frame is typically segmented into multiple packets and losing any packets (of a frame) leads to a frame loss. Movie/TV usually suffers from higher loss rate than others, because it has higher complexity, and hence produces more packets.

Effectiveness of our rate adaptation mechanism. We conduct experiments to compare our proposed algorithm to static bitrate configurations: (i) 1 Mbps and (ii) 12 Mbps at 8 m. We report the frame loss rate and PSNR in Figures 12(a) and 12(b). We observe that 12 Mbps configuration suffers from high frame loss rate, thus leads to inferior video quality except for Applications. A closer look at the encoding bitrate of Applications video reveals that the contents are encoded at 6.6 Mbps on average. Therefore, even when the available bandwidth reduces, the video quality of Applications may not be affected too much. Last, although conservatively setting the static bitrate at 1 Mbps avoids high frame loss rate, it also results in lower video quality. In

[2]The PSNR value of our proposed GA in Figures 11(b) and 12(b) are higher than that in Figure 4 because these two figures are from two independent experiment setups.



(a) (b)

Fig. 12. Comparisons between static and adaptive bitrate: (a) frame loss rate and (b) estimated bandwidth.

particular, our adaptation algorithm outperforms the 1 Mbps configuration by at least 5 dB in PSNR in our experiments.

7. CONCLUSION

Although screencasting is becoming increasingly popular, researchers and developers resort to ad-hoc design decisions, because the performance measurement of screencast technologies has not been throughly studied. In this article, we have developed a comprehensive measurement methodology for screencast technologies and carried out detailed analysis on several commercial and one open-source screencast technologies. The presented methodology is also applicable to other and future technologies, such as screencast products and non-Intel hardware codec SDKs. Our comparative analysis shows that all screencast technologies have advantages and disadvantages, which in turn demonstrates that the state-of-the-art screencast technologies can be further improved by making educated design decisions, based on quantitative measurement results. Exercising different design decisions using commercial screencast technologies is, however, impossible, because these technologies are proprietary and closed. We have presented how to customize GA for a screencast platform, which enables researchers and developers to perform experiments using real testbed when facing various design considerations. Several sample experiments related to actual decision considerations have been discussed, e.g., we have found that hardware video encoders largely reduce the CPU usage, while slightly increase the GPU usage and end-to-end latency. We have acknowledged the importance of well-designed rate adaptation mechanisms for dynamic wireless networks, and designed, implemented, and evaluated a practical rate adaptation algorithm on GA. The lessons learned in our research shed some light on building next generation screencast technologies.

Future work. The current article can be extended in several directions. For example, we plan to study more ubiquitous multi-display usage scenarios, where several handheld device users share these displays in a nature fashion. Another possible extension is an alternative transport protocol, such as Quick UDP Internet Connections (QUIC) [Roskind 2012], which leverages Forward Error Correction (FEC) to mitigate packet losses.

REFERENCES

AirPlay 2014. AirPlay–Play Content from iOS devices on Android TV. (2014). https://www.apple.com/airplay/.

R. Baratto, L. Kim, and J. Nieh. 2005. THINC: a virtual display architecture for thin-client computing. In *Proc. of ACM symposium on Operating systems principles (SOSP'05) (SOSP '05)*. ACM, New York, NY, USA, 277–290. DOI:http://dx.doi.org/10.1145/1095810.1095837

K. Calagari, M. Pakravan, S. Shirmohammadi, and M. Hefeeda. 2014. ALP: Adaptive loss protection scheme with constant overhead for interactive video applications. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM'14)* 11, 2 (2014), 25.

S. Chandra, J. Biehl, J. Boreczky, S. Carter, and L. Rowe. 2012. Understanding Screen Contents for Building a High Performance, Real Time Screen Sharing System. In *Proc. of ACM Multimedia'12*. Nara, Japan, 389–398.

S. Chandra, J. Boreczky, and L. Rowe. 2014. High Performance Many-to-Many Intranet Screen Sharing with DisplayCast. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM'14)* 10, 2 (Feb 2014).

Y. Chang, P. Tseng, K. Chen, and C. Lei. 2011. Understanding The Performance of Thin-Client Gaming. In *Proc. of IEEE CQR'11*.

K. Chen, Y. Chang, H. Hsu, D. Chen, C. Huang, and C. Hsu. 2014. On the Quality of Service of Cloud Gaming Systems. *IEEE Transactions on Multimedia* (Feb 2014).

Chromecast Web Page 2014. Chromecast Web Page. (2014). http://www.google.com/chrome/devices/chromecast/.

M. Claypool, D. Finkel, A. Grant, and M. Solano. 2012. Thin to win? Network performance analysis of the OnLive thin client game system. In *Proc. of ACM Workshop on Network and Systems Support for Games*. DOI:http://dx.doi.org/10.1109/NetGames.2012.6404013

B. Cumberland, G., and A. Muir. 1999. Microsoft Windows NT server 4.0 terminal server edition technical reference. Microsoft Press. (1999).

A. Farshad, M. Lee, M. Marina, and F. Garcia. 2014. On the impact of 802.11 n frame aggregation on end-to-end available bandwidth estimation. In *Proc. of IEEE International Conference on Sensing, Communication, and Networking (SECON'14)*. IEEE, 108–116.

GaiKai Web Page 2012. GaiKai Web Page. (2012). http://www.gaikai.com/.

GamingAnywhere Web Page 2013. GamingAnywhere: An Open Source Cloud Gaming Project. (2013). http://gaminganywhere.org.

Y. He, K. Fei, G. Fernandez, and E. Delp. 2014. Video Quality Assessment for Web Content Mirroring. In *Proc. of Imaging and Multimedia Analytics in a Web and Mobile World*.

HEVC Test Model 2014. HEVC Test Model (HM) Documentation. (2014). http://hevc.hhi.fraunhofer.de/HM-doc/.

H. Hong, C. Hsu, T. Tsai, C. Huang, K. Chen, and C. Hsu. 2015. Enabling adaptive cloud gaming in an open-source cloud gaming platform. *IEEE Transactions on Circuits and Systems for Video Technology* (Jun 2015). Accepted to appear.

C. Hsu, T. Tsai, C. Huang, C. Hsu, and K. Chen. 2015. Screencast dissected: performance measurements and design considerations. In *Proc. of ACM Multimedia Systems Conference (MMSys'15)*. ACM, 177–188.

C. Huang, K. Chen, D. Chen, H. Hsu, and C. Hsu. 2014. GamingAnywhere: The First Open Source Cloud Gaming System. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM'14)* 10, 1 (Jan 2014).

C. Huang, C. Hsu, T. Tsai, C. Fan, C. Hsu, and K. Chen. 2015. Smart Beholder: An Open-Source Smart Lens for Mobile Photography. In *Proc. of ACM Multimedia'15*. Brisbane, Australia.

Intel Web Page 2015. Intel Media Client Solution Web Page. (2015). https://software.intel.com/en-us/media-client-solutions.

Iperf Web Page 2015. Iperf Web Page. (2015). https://iperf.fr/.

A. Javadtalab, M. Semsarzadeh, A. Khanchi, S. Shirmohammadi, and A. Yassine. 2015. Continuous one-way detection of available bandwidth changes for video streaming over best-effort networks. *IEEE Transactions on Instrumentation and Measurement* 64, 1 (2015), 190–203.

R. Kapoor, L. Chen, L. Lao, M. Gerla, and M. Sanadidi. 2004. CapProbe: A Simple and Accurate Capacity Estimation Technique. In *Proc. of SIGCOMM'04*. Portland, OR, 67–78.

L. Kleinrock. 1992. The Latency/Bandwidth Tradeoff in Gigabit Networks. *IEEE Communications Magazine* 30, 4 (1992), 36–40.

H. Lagar-Cavilla, N. Tolia, E. Lara, M. Satyanarayanan, and D. O'Hallaron. 2007. Interactive resource-intensive applications made easy. In *Proc. of the ACM/IFIP/USENIX International Conference on Middleware*.

M. Lai and J. Nieh. 2006. On the performance of wide-area thin-client computing. *ACM Trans. Comput. Syst.* 24 (May 2006), 175–209. Issue 2. DOI:http://dx.doi.org/10.1145/1132026.1132029

M. Li, M. Claypool, and R. Kinicki. 2008. WBest: a Bandwidth Estimation Tool for IEEE 802.11 Wireless Networks. In *Proc. of IEEE Conference on Local Computer Networks (LCN'08)*. Montreal, Canada, 374–381.

Y. Lin, W. Xie, L. Jin, and R. Shen. 2012. Content-adpative H. 264 rate control for live screencasting. In *Visual Communications and Image Processing (VCIP'12)*. IEEE, 1–6.

Markets 2012. Global Market for Wi-Fi/WLAN, Wireless Display/Video, Mobile WiMAX & LTE (4G) and ZigBee Chipsets in Consumer Electronics & Automation Applications worth $20.4 Billion by 2017. (2012). http://www.marketsandmarkets.com/PressReleases/wireless-communication-chipsets.asp.

Markets 2014. Flexible Display Market worth $3.89 Billion by 2020. (2014). http://www.marketsandmarkets.com/PressReleases/flexible-display.asp.

Miracast 2014. Wi-Fi Certified Miracast. (2014). http://www.wi-fi.org/discover-wi-fi/wi-fi-certified-miracast.

MirrorOp Web Page 2014. MirrorOp Web Page. (2014). http://www.mirrorop.com.

OnLive Web Page 2012. OnLive Web Page. (2012). http://www.onlive.com/.

open-airplay: A collection of libraries for connecting over Apple's AirPlay protocol 2014. open-airplay: A collection of libraries for connecting over Apple's AirPlay protocol. (2014). https://github.com/jamesdlow/open-airplay.

OpenH264 Web Page 2015. OpenH264 Web Page. (2015). http://www.openh264.org/.

V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell. 2003. PathChirp: Efficient Available Bandwidth Estimation for Network Paths. In *Proc. of Passive and Active Monitoring Workshop (PAM'03)*, Vol. 4. San Diego, CA.

T. Richardson, Q. Stafford-Fraser, R. Wood, and Hopper. 1998. Virtual Network Computing. *IEEE Internet Computing* 2 (January 1998), 33–38. Issue 1. DOI:http://dx.doi.org/10.1109/4236.656066

J. Roskind. 2012. *QUIC (quick UDP Internet connections), multiplexed stream transport over UDP*. Technical Report. Google technical report,[online].

K. Schmidt, M. Lam, and J. Northcutt. 1999. The interactive performance of SLIM: a stateless, thin-client architecture. In *Proceedings of the seventeenth ACM symposium on Operating systems principles (SOSP '99)*. ACM, New York, NY, USA, 32–47. DOI:http://dx.doi.org/10.1145/319151.319154

Splashtop 2014. Splashtop Home Page. (2014). http://www.splashtop.com.

N. Tolia, D. Andersen, and M. Satyanarayanan. 2006. Quantifying interactive user experience on thin clients. *Computer* 39, 3 (2006), 46–52.

S. Tursunova, K. Inoyatov, and Y. Kim. 2010. Cognitive passive estimation of available bandwidth (cPEAB) in overlapped IEEE 802.11 WiFi WLANs. In *Proc. of IEEE International Conference on Network Operations and Management Symposium (NOMS)*. IEEE, 448–454.

Ubitus Web Page 2014. Ubitus Web Page. (July 2014). http://www.ubitus.net.

Y. Wang, J. Ostermann, and Y. Zhang. 2001. *Video Processing and Communications*. Prentice Hall.

Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (April 2004), 600–612.

T. Wiegand, G. Sullivan, G. Bjntegaard, and A. Luthra. 2003. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology* 13, 7 (July 2003), 560–576.

C. Wu, K. Chen, Y. Chang, and C. Lei. 2013. Crowdsourcing Multimedia QoE Evaluation: A Trusted Framework. *IEEE Transactions on Multimedia* (July 2013), 1121–1137.

x264 Web Page 2012. x264 Web Page. (2012). http://www.videolan.org/developers/x264.html.

x265 Web Page 2014. x265 Web Page. (2014). http://x265.org.

K. Xu, K. Tang, R. Bagrodia, M. Gerla, and M. Bereschinsky. 2003. Adaptive bandwidth management and QoS provisioning in large scale ad hoc networks. In *Military Communications Conference, 2003. MILCOM'03. 2003 IEEE*, Vol. 2. IEEE, 1018–1023.

S. Yang, J. Nieh, M. Selsky, and N. Tiwari. 2002. The Performance of Remote Display Mechanisms for Thin-Client Computing. In *Proceedings of the General Track of the annual conference on USENIX Annual Technical Conference*. USENIX Association, Berkeley, CA, USA, 131–146. http://portal.acm.org/citation.cfm?id=647057.713852

C. Zhang, X. Zhang, and R. Chandra. 2015. Energy Efficient WiFi Display. In *Proc. ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'15)*. Florence, Italy.

W. Zhu, W. Ding, J. Xu, Y. Shi, and B. Yin. 2014. Screen Content Coding Based on HEVC Framework. *IEEE Transactions on Multimedia* 16, 5 (August 2014).

Received August 2015; revised December 2015; accepted December 2015