Screencast Dissected: Performance Measurements and Design Considerations

Chih-Fan Hsu[1], Tsung-Han Tsai[1], Chun-Ying Huang[2], Cheng-Hsin Hsu[3], and Kuan-Ta Chen[1]

[1]Institute of Information Science, Academia Sinica
[2]Department of Computer Science and Engineering, National Taiwan Ocean University
[3]Department of Computer Science, National Tsing Hua University

ABSTRACT

Dynamic and adaptive binding between computing devices and displays is increasingly more popular, and screencast technologies enable such binding over wireless networks. In this paper, 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 rooms 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. Furthermore, our measurement methodology and open-source screencast platform 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

Keywords: Measurements; streaming; wireless networks; experiments; optimization

1. INTRODUCTION

Emerging digital display technologies enable larger, less expensive, higher-definition, and more ubiquitous displays to be deployed in homes, offices, schools, shops, and other spaces. For example, flexible displays hit the market by the end of 2014, which come in different sizes and can be used in various applications, such as expanding the limited screen real estate of mobile and wearable devices, constructing roll-out displays for desktop, tablet, and laptop computers, and serving as huge advertisement screens mounted on buildings [3]. In fact, the worldwide market revenue of flexible displays is expected to increase from 0.1 million USD in 2013 to 31.3 billion by 2020 [11]. These next-generation displays will likely come without integrated computing devices, and rely on other computing devices for computations and storage. Therefore, the binding between computing devices and displays will become more dynamic and adaptive than what we are used to nowadays. In some usage scenarios, a display may be concurrently associated with several computing devices, and a computing device may simultaneously leverage multiple displays. A study reports that, on average, people move across 21 different displays every hour [25], which emphasizes the importance of dynamic binding between computing devices and displays, and this number will certainly go up in the near future.

The dynamic binding of displays and computing devices can be done using *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 wireless networks. Because of its rich usage scenarios, screencast has attracted serious attentions from both the academia and industry. For example, several open-source projects [5, 16] have been launched to enable screencast among wearable and mobile devices as well as desktop, tablet, and laptop computers. There are also proprietary and closed commercial products, such as AirPlay [8], Chromecast [1], Miracast [33], MirrorOp [21], and Splashtop [28]. 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 the screencast technologies.

In this paper, we set up a real testbed to conduct the very first detailed experiments to quantify the performance of the 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 5 commercial products [1, 8, 21, 28, 33] and 1 open-source solution [13]. The commercial products are treated

MMSys '15, March 18 - 20, 2015, Portland, OR, USA
Copyright 2015 ACM 978-1-4503-3351-1/15/03 ...$15.00.
http://dx.doi.org/10.1145/2713168.2713176.

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) [13, 16]. 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 [7]. Nevertheless, using GA as a general screencast technology may leave rooms for optimization, e.g., it is well-known that popular video coding standards, such as H.264 [34], 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 [39].

Fortunately, GA [13, 16] 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 paper, we design and conduct several such experiments, e.g., we integrate GA with emerging video codecs [15, 36] in order 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. To our best knowledge, this paper is the first comprehensive work of its kind.

Our experiments on a real screencast testbed lead to the following insights, which are useful for future screencast technologies.

- Considering diverse usage conditions and performance metrics, there is no single winning screencast technology, which indicates that there are still rooms to optimize the state-of-the-art screencast technologies.
- 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, and more comprehensive error resilience tools are desired.
- Screen contents have fairly different characteristics than natural videos, and adopting emerging video codecs designed for screen contents in screencast technologies leads to better Quality of Experience (QoE).
- Most state-of-the-art screencast technologies do not adapt to dynamic bandwidth in a nice way, and thus suffer from negative impacts, including slow responsiveness, blocking features, and frozen screens, which highlights the importance of rate adaptation.

The rest of this paper 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 analyze the measurement results of the state-of-the-art screencast technologies in Section 5. We then present the GA-based quantitative evaluations and user studies, and we discuss the design considerations for future screencast technologies in Section 6. Section 7 concludes this paper.

2. RELATED WORK

Early screen sharing systems, such as thin clients [2, 27] and remote desktops [10, 26], 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 screen, 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. [6]. Readers are referred to the surveys [20, 38] on these screen sharing systems. To cope with such limitations, several companies offer video streaming based cloud gaming systems, such as OnLive [23], GaiKai [12], and Ubitus [30]. Huang et al. propose GamingAnywhere (GA) [16], which is the first open-source cloud gaming system. These cloud gaming platforms also work for screencast scenarios, although there are some optimization rooms to explore. More recently, Chandra et al. [4, 5] 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.

The performance measurements of screen sharing and cloud gaming systems have been done in the literature. For example, Tolia et al. [29] and Lagar-Cavilla et al. [19] analyze the performance of VNC (Virtual Network Computing), and Claypool et al. [9] and Chen et al. [7] study the performance of cloud games. The performance measurements on the state-of-the-art screencast technologies, however, have not received enough attentions in the research community. He et al. [14] conduct a user study on Chromecast [1] with about 20 participants to determine the user tolerance thresholds on video quality (in PSNR [31]), 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 objective performance metrics, (ii) we use real setups for experiments, (iii) we consider multiple screencast technologies [1, 8, 16, 21, 28, 33], 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.

Last, we note that we choose GA [16] over DisplayCast [4, 5] 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 [13]. Nonetheless, readers who prefer to start from DisplayCast [4, 5] can apply the lessons learned in this work to DisplayCast as well.

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

Table 1: 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)

light-weight. 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 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 might not be easy to integrate other codec implementations into GA following the current design. For example, if we plan to use another H.264 implementation from Cisco [22], 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 the 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 1.

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 [17], 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 supports only 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 6.

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 1). We have integrated *libx265* [37] and HEVC Test Model (HM) [15] with GA. HEVC supports several emerging extensions like Range Extension (REXT) and Screen Content Coding (SCC) [39], which are designed for screencast or similar applications. We note that neither *libx265* nor HM are optimized for real-time applications per our experiments. Longer encoding time however is not a huge concern for now, as both implementations are emerging and we consider the implementations will be optimized before actual deployments. Therefore, in Section 6, 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 [24] 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. Some third-party applications claim to be able to mirror the desktop, however, at the time of writing (Sep 2014), none of these applications are available yet.

Table 2: 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%**

- **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.
- **Interactive 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, 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 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, delay is 0 ms, and 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 in given in Table 2, 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 3, 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 1.6 GHz Intel Core i5 processor and 8 GB memory and the receiver 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 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.

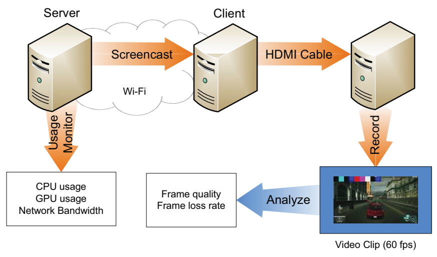
- **Bitrate**. The average amount of data per second transmitted from the sender to receiver, which is important because the wireless spectrum and total bandwidth is limited and shared by all applications/users.
- **End-to-end latency** (latency). The time difference between each video frame is rendered at the sender and

[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.

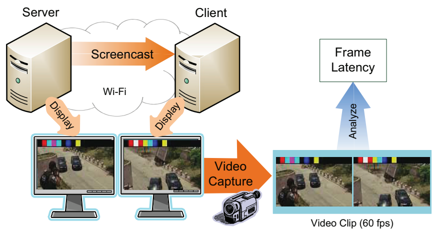
Table 3: Screencast Technologies Considered

Technology		AirPlay	Chromecast	GamingAnywhere	Miracast	MirrorOp	Splashtop
Specification	*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 used	*Sender*	MacBook Pro OS X 10.9.2	Chrome Browser w/ Google Cast Ext. v14.305.0.0 on Windows 8.1 Laptop	Windows 8.1 Laptop	Windows 8.1 Laptop	MirrorOp Sender v2.0.3.2 on Windows 8.1 Laptop	Splashtop Streamer v2.5.8.4 on Windows 8.1 Laptop
	Receiver	Apple TV v6.1.1	Chromecast (firmware v16041)	Windows 7 PC	NETGEAR Push2TV (firmware v2.4.46)	MirrorOp Receiver v0.2.11-4.win on Windows 7 PC	Splashtop Personal v2.4.5.2 on Windows 7 PC

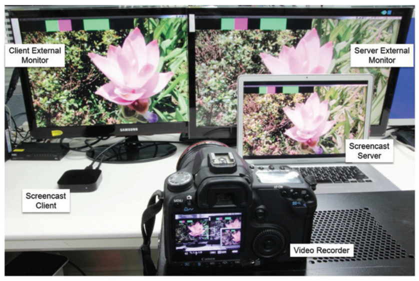
‡ If not otherwise specified, the PC computer is a ThinkCentre M92p, and the laptop computer is a ThinkPad X240.



(a)



(b)



(c)

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

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 [31] and SSIM [32] 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 content 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 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 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, and thus our experiment results are not biased. The actual testbed is shown in Figure 1(c).

5. COMPARATIVE ANALYSIS

We analyze our measurement results in this section. A number of insights are drawn from our measurement results. In summary, we do not observe a single winning screencast technology, which shows that designing an optimized screencast technology remains an open problem.

5.1 Performance under Default Settings

We report the results under the default configurations (see Table 2). Each experiment lasts for 9 minutes 18 seconds, with 33,480 video frames. For each screencast technology, we first calculate the bitrate, latency, and video quality of

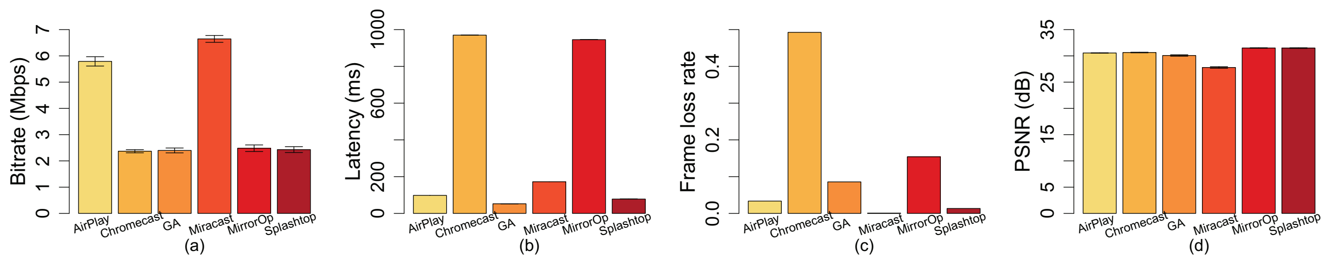


Figure 2: Performance under the default settings: (a) bitrate, (b) latency, (c) frame loss, and (d) quality in PSNR.

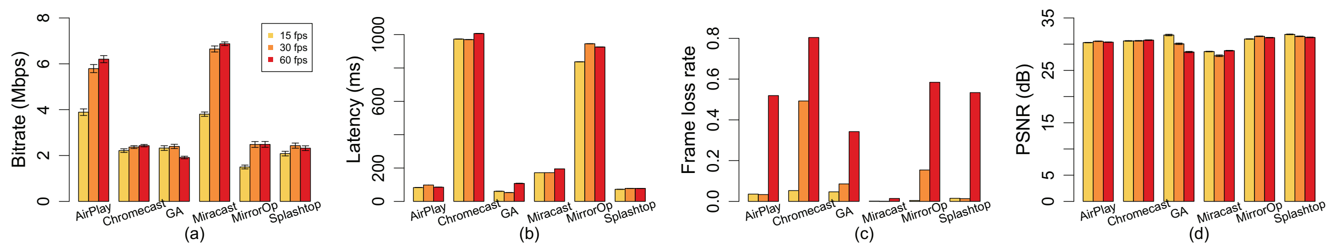


Figure 3: Performance under different frame rates: (a) bitrate, (b) latency, (c) frame loss, and (d) quality in PSNR.

individual video frames rendered by the receiver (i.e., lost frames are not considered) and then compute the mean and standard error of the metrics across all the video frames. We also derive the frame loss rate of each experiment. We plot the results in Figure 2, and make several observations. First, AirPlay and Miracast both lead to high bitrate and low latency, while Miracast achieves much lower frame loss rate. Second, although Chromecast incurs very low bitrate, it suffers from high latency and high frame loss rate. Third, Splashtop and MirrorOp achieve similar bitrate and video quality, but Splashtop leads to lower latency and frame loss rate. Fourth, screencast technologies other than Miracast lead to roughly the same video quality. Last, GA leads to low bitrate, low latency, good video quality, but slightly higher frame loss rate. Figure 2 reveals that most screencast technologies have some weaknesses, e.g., AirPlay and Miracast incur higher bitrate, Chromecast and MirrorOp suffer from high latency, and Chromecast also results in high frame loss rate. In contrast, Splashtop and GA perform fairly well in terms of all metrics. GA's imperfect frame loss can be partially attributed to the default UDP protocol it adopts, and we will take a closer look at the implications of switching to TCP protocol in Section 6.2. We omit the figure of quality in SSIM, because it shows almost identical trends as PSNR.

5.2 Performance under Diverse Workload and Network Conditions

We vary frame rates to generate different amounts of traffics. We plot the performance results in Figure 3, which leads to several observations. First, AirPlay and Miracast incur higher bitrates at 15 fps than other screencast technologies at 30 and 60 fps. Second, higher frame rates generally result in higher latencies and frame loss rates, due to saturated network resources. Third, frame rates impose minor impacts on video quality.

Next, we configure different network conditions in terms of network bandwidth and delay, and plot the observed screencast performance in Figures 4 and 5, respectively. We make some observations on Figure 4. First, AirPlay, Chromecast, and Miracast adjust the bitrate according to the available bandwidth, while GA, MirrorOp, and Splashtop maintain the same bitrate independent to the bandwidth. Second, Chromecast and MirrorOp suffer from excessive latency, while other screencast technologies perform reasonably well. Third, Miracast results in seriously degraded video quality with lower bandwidth, which can be attributed to its over-aggressive bitrate usage. On the other hand, we also make some observations on Figure 5. First, AirPlay and Splashtop are sensitive to delay, because they both reduce the bitrate as the delay increases. Second, higher delay generally results in higher latency and frame loss rate, while GA and Miracast outperform other screencast technologies in these two aspects. Last, only AirPlay and MirrorOp suffer from degraded video quality under longer delay, which we suspect may be partly due to the TCP protocol they adopt (c.f. Table 3).

5.3 Performance Ranking

We study the ranking of these screencast technologies under different conditions. In addition to the default condition, we define *high frame rate* by increasing the frame rate to 60 fps, *lossy network* by setting the packet loss rate to 2%, *high delay network* by setting the network delay to 200 ms, and *low bandwidth network* by setting the bandwidth to 4 Mbps. For each condition, we compute the performance metrics, and rank the screencast technologies on each metric independently[2]. We then plot the results in the form of radar chart in Figure 6, where each of the four axes reports the

[2]We use PSNR as the video quality metric, but SSIM leads to nearly identical ranking.

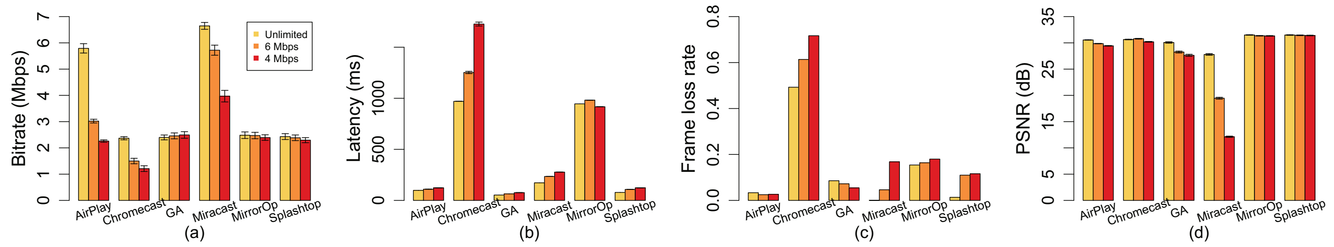


Figure 4: Performance under different bandwidth: (a) bitrate, (b) latency, (c) frame loss, and (d) quality in PSNR.

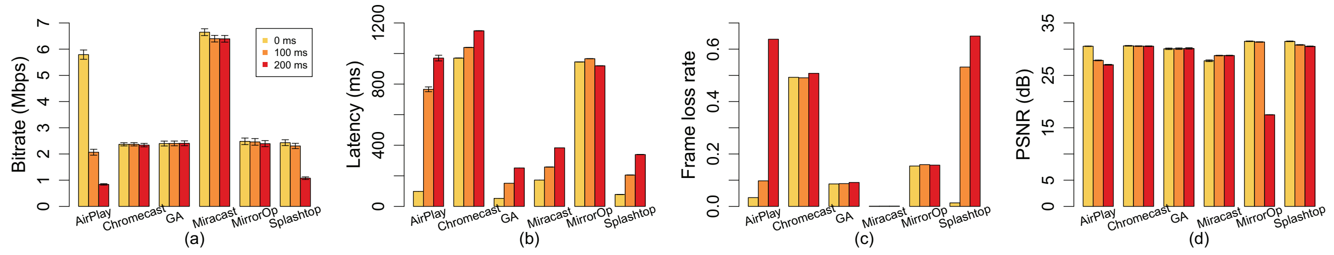


Figure 5: Performance under different delays: (a) bitrate, (b) latency, (c) frame loss, and (d) quality in PSNR.

ranking of screencast technologies in terms of a particular performance metric. This figure reveals that: (i) Splashtop performs the best and balanced in general, and it is never ranked the last in all aspects, (ii) AirPlay and GA perform reasonably well in all aspects, trailing Splashtop, and (iii) Chromecast, Miracast, and MirrorOp lead to inferior performance in general. The figure also reveals potential limitations of individual screencast technologies. For example, under lossy network conditions, GA results in lower video quality and higher latency, which can be mitigated by adding error resilience tools to it.

5.4 Tolerance Ranking

We perform tolerance analysis to quantify how much impact each parameter incurs on each performance metric with diverse screencast technologies. For each screencast technology, we vary a parameter while fixing all other parameters at their default values. We repeat the experiment multiple times, and compute the mean performance of each experiment. For each metric, we compute the *tolerance*, which is defined as one minus the range (i.e., the difference between the maximum and minimum) over the minimum. If the tolerance is smaller than 0, we set it to be 0. Larger tolerance (closer to 1) means more stable performance; smaller tolerance (closer to 0) indicates that the particular parameter affects a performance metric more prominently.

We report the tolerance ranks of latency, frame loss rate, and video quality in Figure 7, where the five axes of each radar chart represent the impact of the five workload/network parameters, and the ticks closer to the origins indicate lower tolerance due to the particular parameter associated with the axis. We make several observations. First, the latency achieved by MirrorOp does not change under different parameters, while latency achieved by other screencast technologies is vulnerable to at least one parameter. For example, AirPlay is vulnerable to changes in network delay and packet loss rate, and Chromecast is vulnerable to changes in bandwidth. Second, the frame loss rates achieved by AirPlay and Splashtop are vulnerable to changes of all parameters, while the frame loss rates of all screencast technologies are vulnerable to changes in the frame rates. Third, most considered screencast technologies achieve stable video quality, except Miracast and MirrorOp, which are sensitive to bandwidth and network delay, respectively. In summary, the frame loss rate is the most vulnerable metric, while all screencast technologies handle video quality quite well. Overall, MirrorOp performs the best, and GA may be enhanced to better adapt to changes in frame rate and delay.

Nevertheless, we need to add that the degree of tolerance needs to be interpreted together with the performance in the evaluation of screencast technologies. For example, MirrorOp performs the best in terms of tolerance. We believe that this is mainly due to its much longer latency (see Figure 2), so maintaining a nearly constant latency and frame loss rate is relatively easy compared to screencast technologies with shorter latencies (such as AirPlay, GA, Miracast, and Splashtop). Thus, tolerance should be the next thing we are looking for only after the performance achieved is satisfactory, and we cannot conclude that one screencast technology is better than others solely based on tolerance comparisons in Figure 7.

6. DESIGN CONSIDERATIONS

Thus far we have investigated the performance of considered screencast technologies under a variety of workload and network conditions. Two main take-aways of Section 5 are: (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 sample

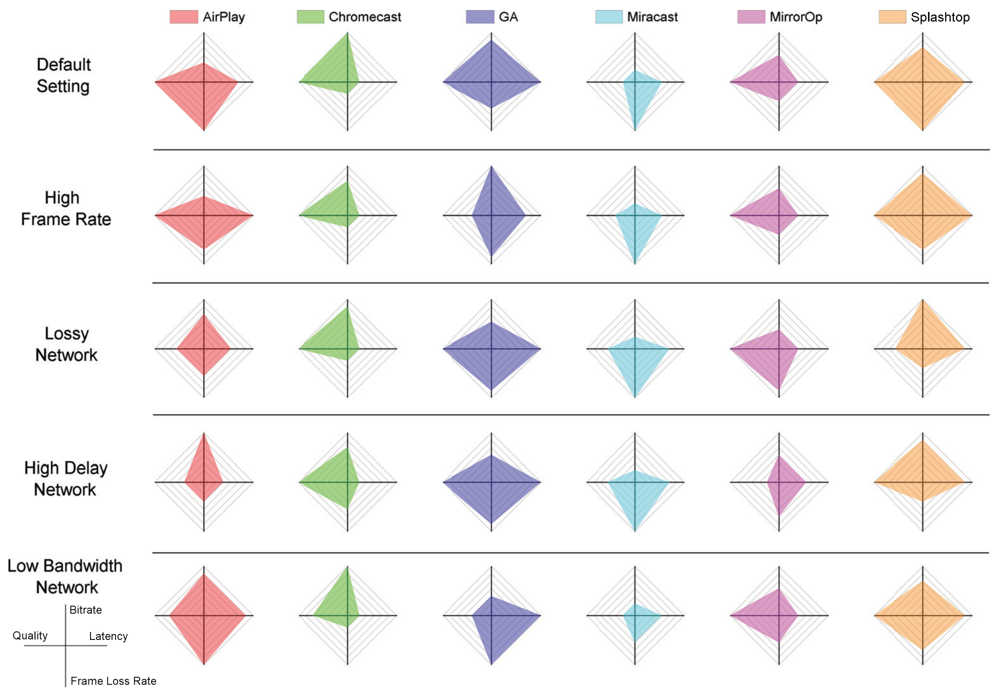


Figure 6: Ranks of different screencast technologies under different conditions. Ticks closer to the origins represent lower ranks (worse performance).

design considerations. We emphasize that our list of design considerations is not exhausted, and readers are free to leverage open-source screencast technologies such as GA [16] and DisplayCast [4, 5] for similar studies.

6.1 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 3, and we stream the 9 minutes 18 seconds video using the default settings given in Table 2. 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 8. In this figure, GA and GA (HE) represent GA with software and hardware video encoders, respectively. Moreover, the numbers in the boxes 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 2(a) and 2(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 [16], as a cloud gaming platform, which is useful 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 [17], which prevents us from performing more detailed latency optimization done 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, which is quite worthy to consider when developers are building future screencast technologies.

6.2 Transport Protocols

The experiment results given in Section 5 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 videos in Figure 9, where two corresponding boxes (those of UDP versus TCP) are connected by dashed lines. The annotations above the boxes are network conditions, and the numbers in the boxes are the PSNR values representing the resulting video quality rendered at the client.

We make the following observations. First, when the net-

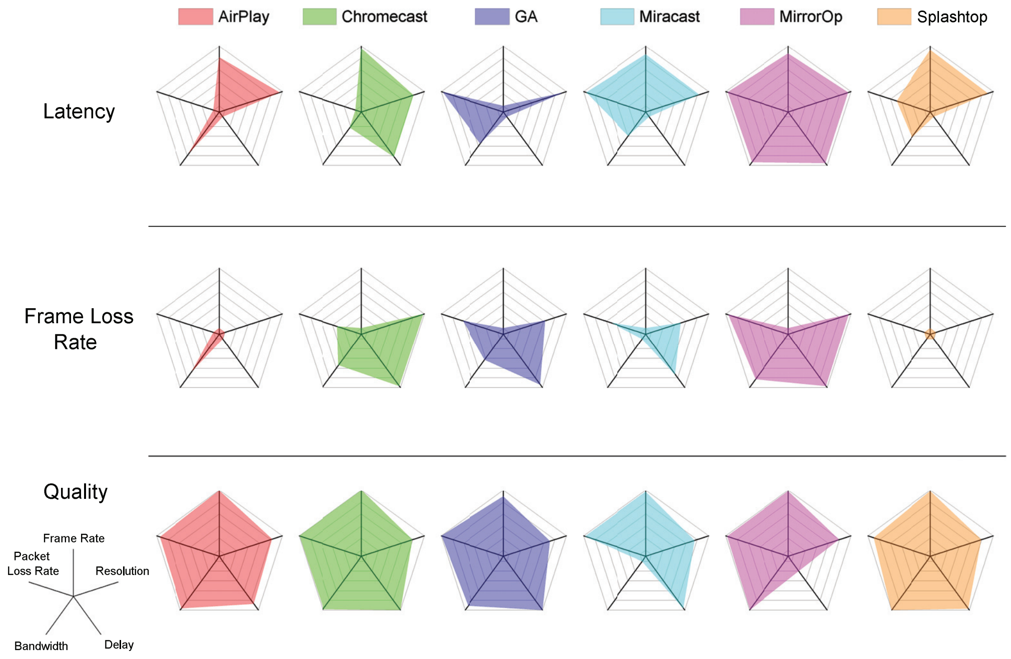


Figure 7: Tolerance of different screencast technologies to different workload and network conditions. Lower tolerance (closer to the origins) means higher vulnerability to dynamic environments.

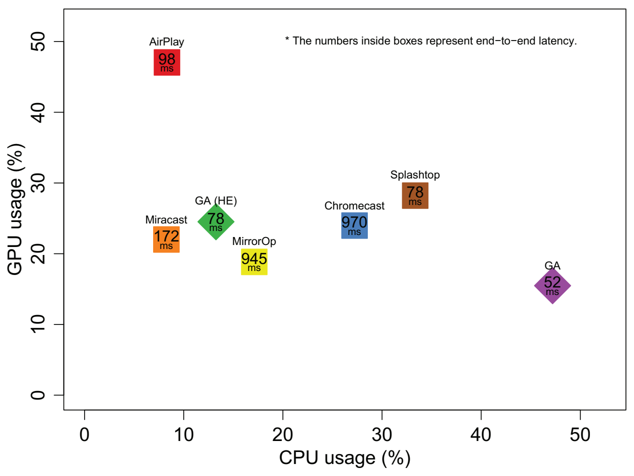


Figure 8: Hardware encoder reduces the CPU usage of GA.

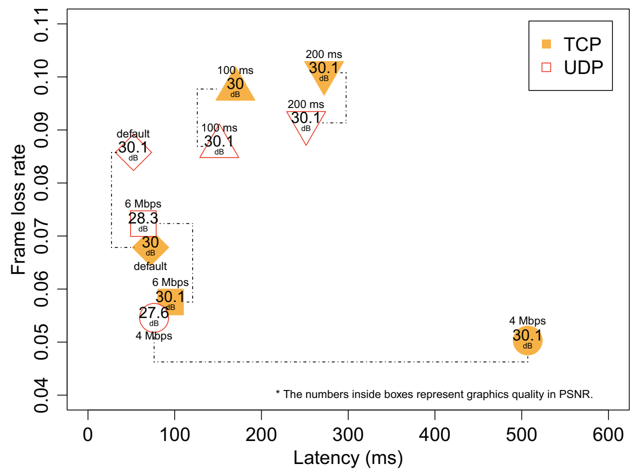


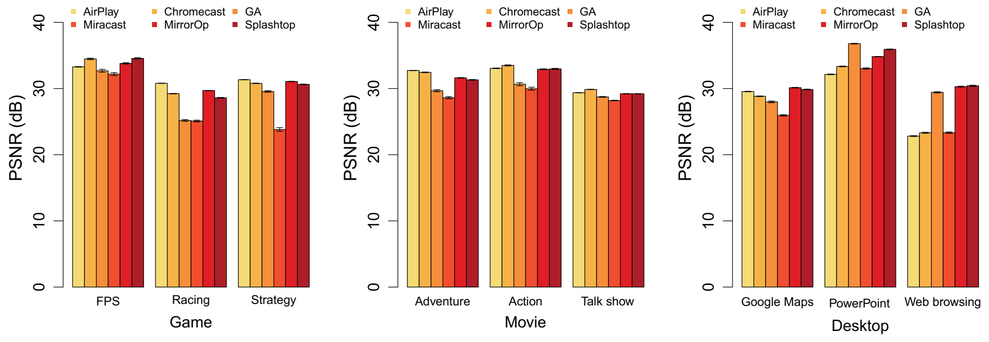
Figure 9: The impacts of TCP and UDP protocols.

work delay is low, TCP always leads to lower frame loss rate: 2% difference is observed. However, when the delay is longer, say $\geq$ 100 ms, TCP results in even higher frame loss rate, which can be attributed to the longer delay caused by TCP, making more packets miss their playout deadlines and are essentially useless. Third, 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 lower video quality, i.e., UDP results in 2.5 dB lower video quality than TCP.

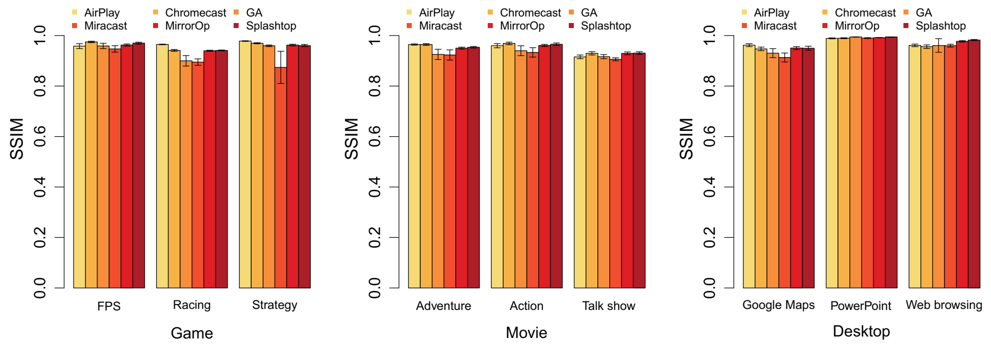
In summary, Figure 9 depicts that the TCP protocol may be used as a basic error resilience tool of GA, but it does not perform well when network delay is longer and when the network bandwidth is not always sufficiently provisioned. This is inline with the well-known limitation on TCP: it suffers from degraded performance in fat long pipes [18], due to the widely adopted congestion control algorithms. Hence, more advanced error resilience tools are desired.

6.3 Video Codecs

Under the default settings, we report the achieved video quality in Figure 10. This figure shows that MirrorOp and Splashtop achieve good video quality for all content types,



(a)



(b)

Figure 10: Video quality achieved by different screencast technologies on diverse content types, in: (a) PSNR and (b) SSIM.

while other screencast technologies all suffer from degraded video quality for some content types. For example, AirPlay leads to inferior PSNR for desktop contents, and GA results in lower PSNR/SSIM for movie contents. 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 $\sim$ 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 [34]. Recently, Screen Content Coding (SCC) has been proposed [39] as an extension to the High Efficiency Video Coding (HEVC) standard. SCC is built on top of the Range Extension (REXT) of the HEVC standard, which expands the supported image bit depths and color sampling formats for high-quality video coding.

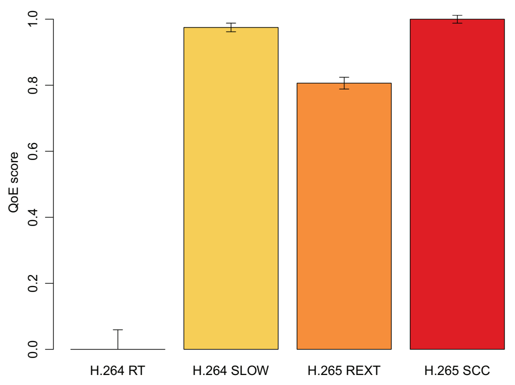
In the following, we conduct a separate study to investigate the benefit of the emerging video coding standards: *H.265 REXT*, which is designed for nature videos, and *H.265 SCC*, which is designed for screen contents. For comparisons, we also include x264 with two sets of coding parameters: the real-time 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

Table 4: The Resulting Video Quality in PSNR (dB)

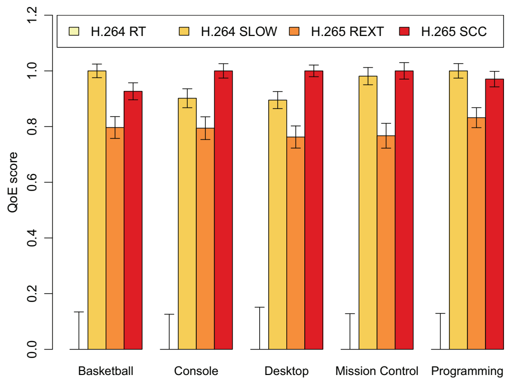
Video	H.264 RT	H.264 SLOW	H.265 REXT	H.265 SCC
BasketballScreen	15.93	33.12	30.83	33.08
Console	15.18	19.35	21.63	22.37
Desktop	13.57	23.08	23.15	28.06
MissionControl3	16.63	34.70	30.46	33.36
Programming	17.29	31.46	31.67	33.36

encode the first 300 frames of each video using the four codecs at 512 kbps on an AMD 2.6 GHz CPU. Table 4 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 $\sim$ 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, and create a website to collect inputs from general publics. We present images encoded by two random codecs side-by-side, and ask viewers to do pair comparison. 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 subjects spent in the study is 27.2 hours. We compute the QoE scores using the Bradley-Terry-Luce (BTL) model [35] and



(a)



(b)

Figure 11: QoE scores of achieved by different codecs: (a) overall scores and (b) individual videos.

Table 5: Observed Negative Impacts due to Imperfect Rate Adaptation (Sampled at 1 Mbps Bandwidth)

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

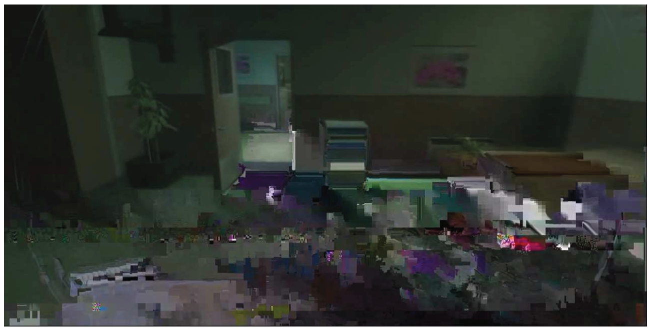


Figure 12: Sample blocking features observed in Miracast.

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 11. We can 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 10 and 11 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.

6.4 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 (see Figure 12), frozen screens, consecutive lost frames, and disconnections (between the sender and receiver) for some screencast technologies once the bandwidth is lower than 3 Mbps. Table 5 presents the sampled 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 Figure 4: 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, GA, MirrorOp, and Splashtop do not adapt their bitrate to the bandwidth at all, and thus sometimes they may under-perform given the available bandwidth and sometimes they may send excessive traffic and suffer from unnecessary packet loss and quality degradation. Hence, these observations clearly manifest that more carefully-designed rate adaptation mechanism is highly demanded in the future screencast technologies.

7. CONCLUSION

The performance of the state-of-the-art screencast technologies has not been rigorously studied, and researchers and developers have to heuristically make design decisions when building the future screencast technologies. In this paper, we have developed 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 also 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 also identified some open problems via the GA-based experiments, such as the importance of well-designed rate adaptation mechanisms for dynamic wireless networks.

References

[1] AirPlay–play content from iOS devices on Android TV, 2014. `https://www.apple.com/airplay/`.

[2] R. A. Baratto, L. N. Kim, and J. Nieh. Thinc: a virtual display architecture for thin-client computing. In *Proceedings of the twentieth ACM symposium on Operating systems principles*, SOSP '05, pages 277–290, New York, NY, USA, 2005. ACM.

[3] Bendable displays are finally headed to market, 2014. `http://www.technologyreview.com/news/529991/bendable-displays-are-finally-headed-to-market/`.

[4] S. Chandra, J. Biehl, J. Boreczky, S. Carter, and L. Rowe. Understanding screen contents for building a high performance, real time screen sharing system. In *Proc. of ACM Multimedia'12*, Nara, Japan, 2012.

[5] S. Chandra, J. Boreczky, and L. Rowe. High performance many-to-many Intranet screen sharing with DisplayCast. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 10(2), Feb 2014.

[6] Y.-C. Chang, P.-H. Tseng, K.-T. Chen, and C.-L. Lei. Understanding the performance of thin-client gaming. In *Proceedings of IEEE CQR 2011*, May 2011.

[7] K.-T. Chen, Y.-C. Chang, H.-J. Hsu, D.-Y. Chen, C.-Y. Huang, and C.-H. Hsu. On the quality of service of cloud gaming systems. *IEEE Transactions on Multimedia*, Feb 2014.

[8] Chromecast web page, 2014. `http://www.google.com/chrome/devices/chromecast/`.

[9] M. Claypool, D. Finkel, A. Grant, and M. Solano. 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*, Nov 2012.

[10] B. C. Cumberland, G. Carius, and A. Muir. Microsoft windows nt server 4.0 terminal server edition technical reference. Microsoft Press, 1999.

[11] Flexible display market to reach nearly 800 million unit shipments by 2020, 2013. `http://tinyurl.com/lmkfmca`.

[12] Gaikai web page, 2012. `http://www.gaikai.com/`.

[13] GamingAnywhere: An open source cloud gaming project, 2013. `http://gaminganywhere.org`.

[14] Y. He, K. Fei, G. Fernandez, and E. Delp. Video quality assessment for Web content mirroring. In *Proc. of Imaging and Multimedia Analytics in a Web and Mobile World*, Mar 2014.

[15] HEVC Test Model (HM) documentation, 2014. `http://hevc.hhi.fraunhofer.de/HM-doc/`.

[16] C.-Y. Huang, K.-T. Chen, D.-Y. Chen, H.-J. Hsu, and C.-H. Hsu. GamingAnywhere: The first open source cloud gaming system. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 10(1), Jan 2014.

[17] Intel Media Client Solution web site. `https://software.intel.com/en-us/media-client-solutions`.

[18] L. Kleinrock. The latency/bandwidth tradeoff in gigabit networks. *IEEE Communications Magazine*, 30(4):36–40, 1992.

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

[20] A. M. Lai and J. Nieh. On the performance of wide-area thin-client computing. *ACM Trans. Comput. Syst.*, 24:175–209, May 2006.

[21] MirrorOp web page, 2014. `http://www.mirrorop.com`.

[22] OpenH264. `http://www.openh264.org/`.

[23] OnLive web page, 2012. `http://www.onlive.com/`.

[24] open-airplay: A collection of libraries for connecting over Apple's AirPlay protocol, 2014. `https://github.com/jamesdlow/open-airplay`.

[25] People swap devices 21 times an hour, says OMD, 2014. `http://www.campaignlive.co.uk/news/1225960/`.

[26] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2:33–38, January 1998.

[27] B. K. Schmidt, M. S. Lam, and J. D. Northcutt. The interactive performance of slim: a stateless, thin-client architecture. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*, SOSP '99, pages 32–47, New York, NY, USA, 1999. ACM.

[28] Splashtop home page, 2014. `http://www.splashtop.com`.

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

[30] Ubitus web page, July 2014. `http://www.ubitus.net`.

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

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

[33] Wi-Fi certified Miracast, 2014. `http://www.wi-fi.org/discover-wi-fi/wi-fi-certified-miracast`.

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

[35] C.-C. Wu, K.-T. Chen, Y.-C. Chang, and C.-L. Lei. Crowdsourcing multimedia QoE evaluation: A trusted framework. *IEEE Transactions on Multimedia*, pages 1121–1137, July 2013.

[36] x264 web page, 2012. `http://www.videolan.org/developers/x264.html`.

[37] x265 web page, 2014. `http://x265.org`.

[38] S. J. Yang, J. Nieh, M. Selsky, and N. Tiwari. 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*, pages 131–146, Berkeley, CA, USA, 2002. USENIX Association.

[39] W. Zhu, W. Ding, J. Xu, Y. Shi, and B. Yin. Screen content coding based on HEVC framework. *IEEE Transactions on Multimedia*, 16(5), August 2014.