Ekho: Synchronizing Cloud Gaming Media Across Multiple Endpoints

Pouya Hamadanian
MIT CSAIL

Doug Gallatin
Microsoft

Mohammad Alizadeh
MIT CSAIL

Krishna Chintalapudi
Microsoft Research

ABSTRACT

Online cloud gaming platforms stream game media to multiple endpoints (*e.g.,* a television display and a controller-connected headset) via possibly different networks with considerably different latencies. This leads to the media being played out of sync with one another, and severely degrades user experience. Typical approaches that rely on network and software timing measurements fail to reach synchronization goals. In this work, we propose *Ekho*, a robust and efficient end-to-end approach for synchronizing streams transmitted to two devices. Ekho adds faint, human-inaudible pseudo-noise (PN) markers to the game audio, and listens for these markers in the chat audio captured by the player's microphone to measure inter-stream delay (ISD). The game server then compensates for the ISD to synchronize the streams. We evaluate Ekho in depth, with a corpus of audio samples from popular online games, and demonstrate that it calculates ISD with sub-millisecond accuracy, has low computational overhead, and is resilient to background chatter, compression and microphone quality. In end-to-end tests over WiFi and cellular links with frequent packet loss and playback disruption, Ekho maintains human-imperceptible ISD (< 10 ms) 86.8% of the time. Without Ekho, the ISD exceeds 50 ms at all times.

CCS CONCEPTS

• **Networks** → **Application layer protocols**; *Control path algorithms*;

KEYWORDS

Media Synchronization, Cloud Gaming, Inter-device Synchronization, ITU-T P.808

ACM Reference Format:
Pouya Hamadanian, Doug Gallatin, Mohammad Alizadeh, and Krishna Chintalapudi. 2023. Ekho: Synchronizing Cloud Gaming Media Across Multiple Endpoints. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10–14, 2023, New York, NY, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3603269.3604826

1 INTRODUCTION

— Soon Echo was no longer found, Except a faint repeating sound. (Ima Ryma, 2022)

ACM SIGCOMM '23, September 10–14, 2023, New York, NY, USA

ACM ISBN 979-8-4007-0236-5/23/09.
https://doi.org/10.1145/3603269.3604826

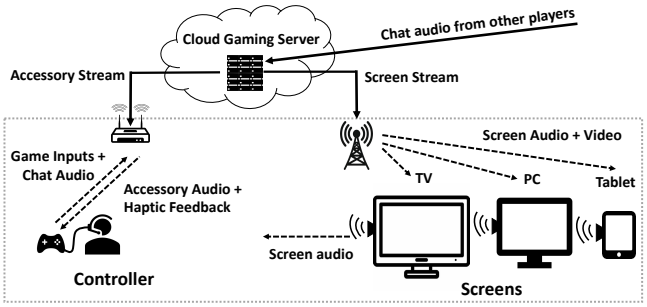


Figure 1: A typical cloud gaming setup

Online cloud game streaming comprises an important class of applications with a $6B market and continues to see an unprecedented growth [13]. In online game streaming players connect to remote gaming servers in the cloud through WiFi/cellular/wired connections, instead of consoles in their home. Gaming servers process gaming inputs (*e.g.,* joystick movements, button presses, *etc.*) generated by the player and respond by rendering game audio, video and, haptic feedback. They also facilitate multi-player coordination through audio chat.

A typical online game streaming system (*e.g.,* XCloud, the cloud streaming service by Xbox) is depicted in Figure 1. The gaming server receives gaming inputs and audio chat streams from gaming accessories such as controllers and headsets. In response it generates two separate media streams simultaneously for the player. First, a *game-screen stream* comprising game audio-video intended for a screen device such as a television, LCD display or tablet. Second, a *game-accessory stream* intended for gaming accessories such as gaming controllers and gaming audio headsets comprising of (1) the game audio sounds mixed with chat from fellow players, intended for players' ears only and played through a headset connected to the controller via an audio jack or a wireless headset; (2) haptic feedback, such as controller vibrations.[1] Given different end points on different devices, these two streams are conveyed over separate network network connections.

In the typical scenario, a gamer listens to the accessory audio stream including game audio and chat from fellow players over headphones while watching the screen video stream on the screen. In addition, a common practice in gaming is for the gamers to have audience while they play. In this case, the screen will also play a

[1]Past approaches used the screen as a proxy to connect the gaming accessories, *e.g.,* controllers to the screen via Bluetooth. However, modern providers prefer a separate direct connection between the server and the accessories to avoid the extra Bluetooth hop that causes 17 to 30 ms of latency [20].

screen audio stream for the benefit of the onlookers. Thus, during game play, a player hears two audio streams simultaneously – the game-screen audio stream playing on the headphones, and the game-accessory audio stream playing on the speakers of the screen device and captured by the player's microphone.

While the screen audio and video playback are synchronized at the screen device, they are not synchronized to the accessory audio stream and haptic feedback which are conveyed over a separate network connection to different gaming accessory devices. This lack of synchronization between the screen and accessory streams can lead to the perception of an echo, a video lag and a sluggish haptic response leading to distraction and a poor gaming experience, especially for professional players and in games that are sensitive to the players' reaction time such as driving and fighting games. As we demonstrate in §3 through our user studies, players can perceive an echo by audio synchronization errors above 10 ms and video synchronization errors over 15-45 ms.

In general these two streams will incur an *Inter-Stream Delay* (ISD) ranging from few tens to over 100 ms depending on how players choose to connect their screen devices and gaming accessories to the cloud server over the internet, the type of end devices, and even the distance of the player from the screen device as we describe in detail in §3.

Network Delay Differences. The common scenario is when the screen is connected via Ethernet to the home router,[2] while the controller is connected to a WiFi access point (AP). Sometimes, the screen and the controller/headset may be connected to different WiFi APs. In another common scenario, in a hotel room, a user might connect their tablet (screen) over a cellular network with latency well over 100 ms, while the WiFi based controller incurs only few tens of ms. Further, the streaming protocol used for these two streams might be different depending on the OS, front-end, and compute constraints of the screen and controller — WebRTC for the screen [14] and URCP [37] for the controller. This will in general lead to different network delays to the WiFi AP. ISPs may prioritize traffic to certain ports over others or even tunnel the streams differently based on their type of IP addresses (IPV4/IPV6).

Device Playback Differences. Besides network delay differences, the devices themselves add variable delays to the streams. The end devices consuming these two streams may have different hardware (GPU/CPU, *etc.*), operating systems, scheduling and buffer management strategies in their applications leading to differences of up to few tens of ms. Playback devices such as TVs add 10-60 ms additional latency before playback due to buffering and hardware post processing.

Sound Propagation Delay. Finally, sound propagation from the screen to the player's ear itself adds a delay of several ms depending on the player's distance from the screen, *e.g.,* 6 ms for a TV screen 6 feet away (1 ms/feet).

In this paper we consider the problem of synchronizing the screen and accessory streams to limit ISD < 10 *ms with the goal of eliminating the perception of echo and audio-video lag for the player.* ISD is dynamic due to changes in network latencies as well as buffering delays at end devices. Thus, any synchronization scheme must continuously measure and track the ISD in order to compensate for it. As discussed in detail in §3, measuring ISD to an accuracy well below 10 ms can be extremely challenging due to two key sources of error. First, network based measurements, *e.g.,* those that rely on Round Trip Time (RTT) measurements may incur errors of several tens of ms due to asymmetry in the forward and backward network paths. Second, many other contributors to latency such as delays from decoding, playback and sound propagation are impractical/impossible to measure either due to the lack of access to APIs and are difficult to profile given the variety of devices.

Prior work on Inter-Device Synchronization (IDES) [12, 23] mainly approach this problem in three ways. **(a)** By using Precision Time Protocol (PTP), *i.e.,* dedicated connections through a shared network component, *e.g.,* a WiFi router, to synchronize clocks [4]. An example of this is WiFi Certified TimeSync [2]. Typical cloud gaming scenarios, however, do not observe (or force) a shared router. **(b)** Using the Network Time Protocol (NTP) to synchronize clocks across devices [21]. However, NTP is known to be as inaccurate as 100 ms [22], particularly in cases with asymmetric network delay. **(c)** Using a recording with all device playbacks, one can measure ISD [8]. However these approaches consider controlled environments with high quality microphones, no background chatter and lossless recordings without compression, which as we shall observe in §6.4, lead to poor synchronization in more than 75% of sample stimuli. Last, such approaches fail in scenarios where the screen audio stream is muted (*e.g.,* to prevent disturbing others).

In this paper we present Ekho, a robust end-to-end system that can measure ISD to sub-millisecond accuracy and use it to synchronize the screen and accessory streams accurately to eliminate echo, video and haptic lag. Ekho comprises of two key components. First, a novel ISD measurement mechanism, *Ekho-Estimator*, that can estimate the latency difference between the streams to sub-millisecond accuracy by embedding pseudo-noise acoustic sequences within the streams. Second, a mechanism *Ekho-Compensator*, that uses this measurement as a feedback to synchronize the streams.

In Ekho-Estimator, the game server adds a Pseudo-Noise (PN) sequence to the rendered screen audio stream that is imperceptible to human ears. As the speakers on the screen play the game audio, it is overheard and captured by the chat microphone in the player's headset possibly mixed with the player's chat as they speak into the microphone. The captured screen audio as well as the accessory stream from the server are time stamped using the local clock of the headset/controller. By correlating the two streams against the PN-sequence and comparing the time-stamps, Ekho is able to estimate the ISD to a sub-millisecond accuracy. The added pseudo-noise sequence allows Ekho to perform accurately despite being mixed with human chat, lossy audio compression encoding that deteriorates recording quality, ambient noise in the game environment, and even on poor quality speakers and microphones on the end devices. The estimated ISD can be negative or positive. The Ekho-Compensator at the cloud game server then either adds extra samples (by adding zeros or audio loss concealment mechanisms) or skips samples depending on positive or negative estimates.

We have evaluated Ekho by implementing and testing it across a large range of headsets varying in quality as well as ambient and human chat noises. To ascertain that the added PN-sequence is imperceptible to humans, we have conducted Degradation MOS

[2]Ethernet provides a more stable connection for cloud gaming compared to wireless connections [27], and is generally preferred.

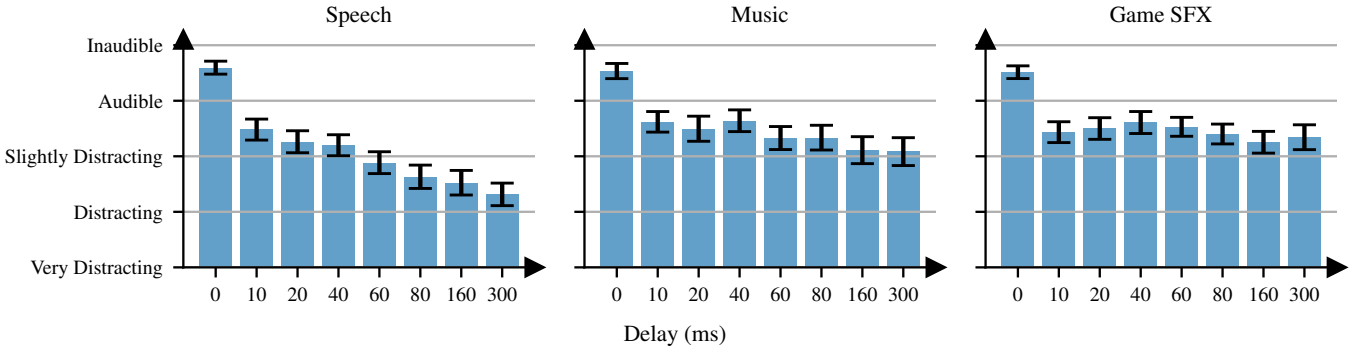


Figure 2: Crowdsourced opinion scores for how echoes affect user experience. Higher delays in the echo visibly affect the speech category, while the effect plateaus for music and game sound effects.

tests, following the ITU-T P.808 standard [24]. In summary, our contributions in this paper are as follows,

- We propose Ekho, a novel system to synchronize gaming streams across multiple endpoints to sub-millisecond accuracy over the public internet and improve user experience in online gaming (§4 and §5).
- In an end-to-end test with WebRTC streams going over WiFi and cellular links, Ekho maintains synchronization below human-perceptible levels (< 10 ms) for 86.6% of the time, whilst without Ekho streams never reach synchronization below 50 ms.
- Through crowd-sourced user tests we demonstrate that our PN-sequence addition technique in Ekho is imperceptible to players (§6).
- We evaluate Ekho extensively across different headsets as well as ambient conditions to evaluate its efficacy (§6).

2 RELATED WORK

Delay Perception. A rich body of work has investigated how playing multiple unsynchronized media affect user perception. The effect varies based on the relationship among these media and their content.

Humans are most sensitive to two similar audio streams playing with at a delay with respect to each other. Prior work defines the *echo threshold* as the maximum delay that two identical sounds can be heard by a human without an echo being perceived [11]. The echo threshold is highly affected by the type of audio, equipment and subjects, ranging anywhere from 2 to 50 ms; sudden sharp sounds are discernible with more than 5 ms of delay [18, 30], and complex signals such as speech may be less discernible until 16–22 ms [17]. Game audio streams typically comprises a mixture of all these different kinds of sounds. Further, games differ with respect to the composition of these sounds. In this paper, we are the first to study the effect of echoes in video game audio on user experience, and conclude that delays as low as 10 ms can disturb the user (§3.1).

Video to audio lag is more forgiving. Past work shows that delays as low as 20 ms reduce subject evaluation by statistically significant margins [29]. Standards on video/audio playback allow the audio to precede the video up to 15 ms, and the video to precede the audio up to 45 ms in delay [6].

Users sense the delay between haptic feedback and audio if the delay is above 24 ms [1]. Haptic feedback to video delay is more forgiving, and tolerable up to 30–100 ms [31, 32].

Media Synchronization. We focus on IDES, *i.e.,* synchronizing media across multiple endpoints in close vicinity. These endpoints connect to the same server in the cloud, and we do not assume any further shared network path. For IDES, existing work rely on three techniques:

(a) Provided the devices have accurate clocks, they can exchange when playback timing and synchronize [21]. To synchronize clocks, they can use the NTP, which connects to accurate time servers in the cloud and adjusts the system clock to the timestamp of the server corrected with the RTT. Unfortunately, NTP is known to be as inaccurate as 100 ms [22], and is especially affected by network path asymmetry. Network path asymmetry is common in the internet [26], particularly in wireless networks [36].

(b) Some routers and devices implement PTP, which allows relative clock synchronization for devices connected to the same router. [2]. This is a highly accurate method, and should synchronize media to tolerable delay levels [4]. Unfortunately, this is only useful when the media endpoints share the same router, and it is unclear to us what fraction of existing routers support PTP.

(c) An opportunistic and less common approach to relative clock synchronization is to leverage the physical locality of endpoints directly. If we can record the playback of one device in the other, we can use signal processing based techniques to measure the playback delay. Past work showcases this approach in a controlled environment [8], but real-world limitations hindered it's applicability; low-quality microphones, background audio such as chatter, and lossy uplink audio compression are common, and contribute to poor synchronization in more than 75% of test cases, as we highlight in §6.4. We build on this third approach, and through a clever design that utilizes inaudible markers, we achieve synchronization in more than 90% of test cases.

3 BACKGROUND AND MOTIVATION

In this section we provide the necessary background and motivation for Ekho.

3.1 Time Synchronization Requirements

We first ask the question, *"What is the echo threshold, i.e., the minimum synchronization error threshold between the screen and accessory streams* that the player can perceive as an echo-like artifact for the most popular online games?" Answering this question will inform us as to the required accuracy in synchronizing the streams.

Existing Echo Threshold Studies. The existing corpus of research and studies on echo thresholds [11, 17, 30], suggest that the echo threshold varies greatly with the type of audio (*e.g.,* speech, sudden sharp sounds, music, *etc.*), stimuli, equipment and subjects, ranging anywhere from 2 to 50 ms [18]. Given this rather large range and lack of specificity, we conducted a study specifically targeting echo thresholds in game audio.

Methodology. For our study we adhered to the Degradation Category Ratings (DCR) test methodology as dictated by the ITU-T P.808 standard [24]. We have created a corpus of 30 game audio clips, each 15s long sampled from 15 popular online games (details in Table 2 in the appendix). Our corpus includes first person shooter (FPS) games, sports and racing games, platform games, *etc.* To each of the clips we add a delayed version of the same clip (echo) with varied delays ranging from 10 to 300 ms to generate a large number of audio clips with echo.

Human testers first listened to a reference audio clip without echoes, and then listened to the same clip with echoes with an unknown amount of delay including 0 ms, *i.e.,* no echo. They were asked to select one of many choices indicating the level of distraction arising from the echo. Each tester periodically went through a training phase to ensure a quiet standardized environment by first listening to six clips with varying amounts of echo as training (as per the P.808 standard). We collected a total of 3555 ratings, removed low-quality responses (33% of total data) in accordance with ITU-T P.808 [24]. Each clip was rated $\approx$ 10 times, and we had 30 different clips for each delay level across all games resulting in a total of $\approx$ 296 votes per delay level. The respondent pool consisted of 17% young adults (18-25), 43% adults (26-35), 32% middle-aged individuals (36-50), and 8% seniors (51 and above), and their native language was English. Following ethical guidelines, surveys were anonymous, and we informed respondents of the survey process and type of stimuli before participating. We compensated respondents, and avoided stimuli that could be considered disturbing. Microsoft IRB reviewed and approved this survey.

Results. Since, echo threshold depends on the content of the audio, we categorize audio clips into three categories comprising predominantly – speech, music and game sound effects (car sounds, shooting sounds, *etc.*) Average opinion scores along with 95% confidence intervals for each of these three categories are presented in Figure 2. As seen from Figure 2, across all categories, even a 10 ms echo is perceptible and slightly distracting. Given that gaming sessions typically last 30 minutes to several hours, even slightly distracting in a 15 second sample is considered unacceptable and must be devoid of distractions for complete immersion, especially for professional level players. Speech-based games show a steady increase in annoyance beyond 10 ms delays, while the distraction plateaus for music and game sound effects. *This study suggests that echo threshold and hence ISD should be lower than* 10 *ms.*

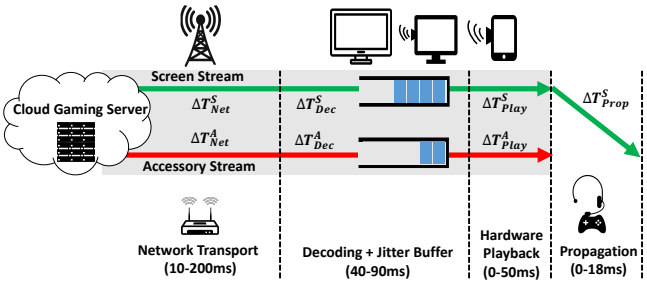


Figure 3: Latency breakdown and synchronization error between screen-stream and accessory-stream in cloud gaming.

Video and Tactile Lag Perception Studies. Studies show that if audio precedes video by as little as 20 ms, subject evaluation loss is statistically significant [29]. Standards on tolerable audio/video lag state that audio must not precede video by more than 15 ms, and video must not precede by more than 45 ms [6]. Past research shows that users can perceive audio-to-haptic and video-to-haptic lag at delays as low as 24 ms and 30 ms [1, 31, 32]. Hence, we believe that synchronizing the screen and accessory audio streams to within 10 ms to eliminate echo perception is sufficient to synchronize the video and haptic feedback.

3.2 Inter-Stream Delay

In this section, we model the (ISD) between the screen and accessory game streams as heard by the player by analyzing the various latencies incurred along the path as depicted in Figure 3 and listed in Table 1.

Network Latency. This part of the latency captures the time taken by the audio packets to traverse the path from the cloud server to the end device (ΔT^A_{Net} for the accessory stream and ΔT^S_{Net} for the screen stream). The latency can range anywhere from 10 to 200 ms. Latency beyond 200 ms renders the majority of games unplayable [7].

The most accurate ways to measure network latency is by synchronizing the physical clocks on each device using GPS or dedicated shared hardware (PTP). This approach however is not practical for devices such as gaming controllers, headphones, TVs, *etc.* Network-based time synchronization techniques (NTP) over public internet incur several 10s of ms of errors and consequently are inadequate for our goal of synchronizing under 10 ms. A common approach is to estimate network latency by measuring RTT and estimate latency as RTT/2 at the server. This approach relies on the assumption that network latency is identical in both forward and reverse directions – server to device and device to server. Prior studies [26] however, have shown that asymmetry in network path latencies is common and that estimates relying on symmetry can be off by up to 60 ms.

Decoding Latency. Audio is typically compressed for bandwidth efficiency using encoders such as OPUS [35]. Upon their receipt from the transport layer, audio packets must be decoded to the native Pulse-Code Modulation (PCM) audio format for playback at the end device. The end devices are typically embedded devices and the decoding times (typically $0-10$ ms) vary depending on available

Latency Part	Notation	Typical Range (ms)	Measurement Error Source	Measurement Error Range (ms)
Network Path	ΔT^S_{Net} & ΔT^A_{Net}	$10-200$	RTT asymmetry	$0-60$
Decoding and Buffering	ΔT^S_{Dec} & ΔT^A_{Dec}	$40-90$	3rd party APIs	$0-80$
Hardware Scheduling	ΔT^S_{Play} & ΔT^A_{Play}	$0-50$	Immeasurable	$0-50$
Sound Propagation	ΔT^S_{Prop}	$0-18$	Immeasurable	$0-18$
Total	-	$50-360$	-	$0-210$

Table 1: Latency breakdown in cloud gaming, source of error, and ranges.

computational resources (*e.g.,* GPU/CPU), operating system, load due to other applications sharing resources, and the encoding audio format.

Once the audio is decoded, a jitter buffer receives and controls playback rate. Buffer management strategies try to ensure smooth playback in the face of network jitter. Both the jitter as well as the buffer management algorithms vary from device to device, and can incur up to 80 ms in typical streams. We represent the time taken to decode for the screen-stream as ΔT^S_{Dec} and that for the accessory-stream as ΔT^A_{Dec}. While in theory, these delays can be profiled by careful measurements on the target device and audio format type, in practice, given the sheer variability in the number of devices, it is impractical to profile all varieties of end devices. Also, most devices do not provide APIs to measure jitter buffer delays.

Hardware Latency. The jitter buffer schedules audio for playback at specific times. The actual playback time, however, varies depending on the device's hardware and operating system buffers. For example, screen devices (*e.g.,* TV, tablet) may delay playback to try and synchronize audio to corresponding video frames going through post-processing, while headsets do not. We represent the latency incurred in the hardware as ΔT^A_{Play} for the accessory-stream and ΔT^S_{Play} for the screen-stream. Outside of profiling, such latencies can not be measured.

Sound Propagation Latency. Sound waves travel at a speed of ~ 330 m/s, or ~ 1 ms/foot. Players may be located at a distance of 2 feet, to as far as 19 feet, from the TV.[3] Thus, the screen-stream audio emanating from the speakers of the screen device will incur a propagation delay (represented as ΔT^S_{Prop}) up to 18 ms before reaching the ears of the player. Measuring this component of delay is not feasible.

Thus, we can characterize ISD as:

$$\begin{aligned} ISD = \Delta T^S_{Net} + \Delta T^S_{Dec} + \Delta T^S_{Play} + \Delta T^S_{Prop} \\ - \Delta T^A_{Net} - \Delta T^A_{Dec} - \Delta T^A_{Play} \end{aligned} \quad (1)$$

Conclusions. The variation in these different components of latencies can add up to several tens of milliseconds to even over 100 ms. Coupled with the inability to measure the various latencies accurately, it is not possible to estimate the synchronization error and synchronize audio streams to within 10 ms using network-based measurements.

[3]The maximum rated distance for an Xbox Wireless controller is 19ft. [33]

3.3 ISD variation

A further complexity is that latency components will vary over time, rendering prior ISD measurements obsolete. We can broadly categorize latency variation in three groups:

No variation. The hardware lag from when audio is given to (taken from) the Digital-to-Analog Converter (DAC) (Analog-to-Digital Converter (ADC)) modules is a low-level hardware lag that does not change.

Low-frequency. These variations are substantial (up to several 100 ms), but occur rarely. Examples include network latency changes due to path changes in the IP layer, decoding latency changes due to CPU/GPU throttling, or changes in the sound propagation latency due to player movement.

High-frequency. Network latency can change on a per packet basis, due to congestion, queuing and competing traffic. If packets are streamed in 20 ms frames, any latency fluctuation above 20 ms will lead to the frame arriving too late. To circumvent this, a jitter buffer is typically implemented that takes in frames, and only serves them once a the buffer level surpasses some threshold, say 60 ms. In theory, any fluctuation up to this threshold will be absorbed by the jitter buffer. However, fluctuations above this threshold will still deplete the buffer, and lead to a change in ISD.

Conclusion. Although the jitter buffers isolate stream playback from modest delay fluctuations, we still need to synchronize the accessory and screen audio streams (i) at the start of the playback and (ii) again, anytime when delay fluctuations cause a jitter buffer to deplete. Ekho measures and attempts to correct ISD periodically. The ISD measurement frequency determines how fast we can respond to changes. In practice, we believe that a measurement at most every 15 seconds will suffice to handle most ISD changes.

4 EKHO

In this section we provide the intuition and an overview of our proposed scheme Ekho that can measure the inter-stream latency difference with a sub-millisecond accuracy and uses these measurements as feedback to synchronize the screen and accessory audio streams.

4.1 Overview of ISD Measurement in Ekho

Figure 4 illustrates the central idea behind the measurement of ISD in Ekho. The server streams the accessory stream directly to the headset. At the same time, the screen audio streamed to the screen device (*e.g.,* TV) is played by its speakers. This screen audio stream

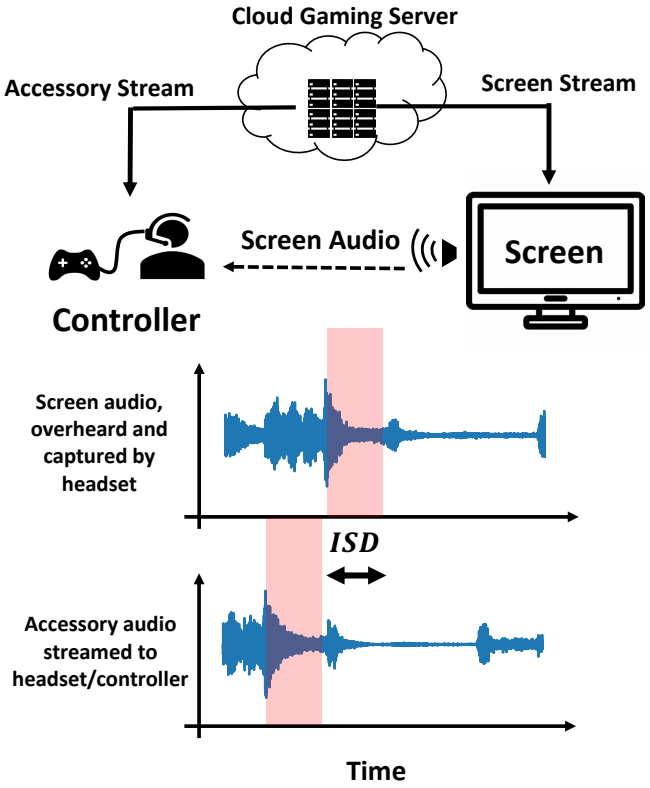


Figure 4: Intuition behind measuring ISD in Ekho

is overheard and captured by the player's headset microphone as a part of the chat audio stream. Thus, ISD can be estimated by correlating the overheard screen game audio content in the audio chat to that in the accessory-stream received directly from the cloud server. This estimated ISD is an end-to-end estimate, as the latency difference of the overheard screen stream and the received accessory stream is precisely what causes the perception of an echo. This key intuition forms the basis of the design of Ekho's ISD measurement mechanism.

Poor auto-correlation of game audio. Ekho's ISD measurement based on the above intuition faces a key challenge. Unlike signals like chirps or pseudo-noise sequences, game audio does not possess sharp auto-correlation properties. While signal processing techniques like Generalized Cross-Correlation PHAse Transform (GCC-PHAT) [5] perform significantly better than standard correlation based schemes in correlating game audio, they incur several tens of ms error as we evaluate in §6.4. This is due to several alterations that the audio signals undergo on their path to the headset. The overheard version of the game audio stream from the speakers in the headset microphone are an order of magnitude fainter than the speech of the player directly speaking into it. It is affected by the echoes and ambient sounds in the room. The speaker and microphone themselves introduce distortions in the waveform depending on their quality. Furthermore, both audio streams undergo deterioration due to lossy compression by the server and the accessory audio stream is also mixed with chat audio from other players.

Pseudo-Noise Markers. To provide the game audio stream with sharp cross correlation properties, Ekho adds identical pseudo-noise (PN) sequences as markers periodically at a regular interval (1 second in our implementation) into the screen audio stream at the cloud server. These markers can be detected within the streams by correlating with the known PN-sequence marker. This allows Ekho to continuously estimate and track the ISD between the streams. The PN-sequence requires a careful design to ensure that it does not deteriorate the quality of the game audio perceptibly to humans. Our choice of PN-sequences rather than other sequences like chirps is motivated by human imperceptibility.

4.2 Reliable marker detection in Ekho

There are three key challenges in designing markers and their detection in our ISD measurement scheme. First, as discussed earlier, the PN-sequence marker should be inaudible to humans. Second, our PN-sequence marker detection scheme must have a very low false positive rate, *i.e.,* it must not detect spurious peaks despite the presence of human speech and being altered by compression, microphones and speakers. This is because spurious peaks can cause very large estimation errors, and when synchronization algorithms compensate for the erroneous ISD, they cause jitter and unpleasant audio artifacts at the end user. At 48,000 samples/sec, a false positive rate of 10^{-6} per sample will result in a spurious detection every 20 seconds; in practice, we require a significantly lower false positive rate. Third, we must periodically estimate ISD to keep up with the changes in ISD due to changes in network latencies [10] as well as playback latencies (as buffer sizes vary due to buffer management schemes).

We now outline the sequence of steps that Ekho uses for the addition and detection of the markers in detail.

Marker Addition. Through several tests we found that often chat audio is typically encoded at Super Wide Band (SWB), *i.e.,* up to 12KHz in the frequency spectrum. Further, most significant audio content generally lies below 6KHz. Thus, Ekho PN-sequences are band-limited between 6-12KHz, to avoid either limitation. Generating PN-sequences is simple; we create a vector of L Gaussian Normal variables, and then apply a band-pass filter between 6-12KHz on this vector. The amplitude of game audio is dynamic and varies significantly on the timescale of few tens of ms. Audibility of the marker depends on the relative amplitudes of the markers to the game audio. Since, the game audio amplitude is dynamic and varies significantly, in order to ensure imperceptibility of the added PN-sequence marker, Ekho continuously tracks the game audio amplitude and adjusts the amplitude of the added PN-sequence as a moving average to ensure that the ratio of amplitudes of the PN-sequence to that of game audio is constant.

Let $x[t]$ be the game audio and $w[t]$ the PN-sequence of length L. Then $x_m[t]$, the audio stream with added markers, is computed as,

$$\begin{aligned} x_m[t] &= x[t] + C.\alpha_k.w[t] \\ \alpha_k &= \gamma \times \alpha_{k-1} + (1 - gamma) \times \rho(x[(k-1).T : k.T]) \end{aligned} \tag{2}$$

The choice of L affects both the rate of ISD measurement and the maximum measurable ISD, as we describe later on, and for most experiments, we set $L = 48,000$ (corresponding to 1 second of audio).

In Equation (2), $\rho(x[(k-1).T : k.T])$ is the signal power in the 6KHz to 12KHz spectrum, computed over T samples. We chose $T = 960$ corresponding to 20 ms of samples at 48KHz. Our choice stems from practical ease of implementation considerations, since each OPUS encoded audio packet at the cloud server comprised 20 ms of audio. Thus, choosing 20 ms, allowed us to adjust amplitude on a per packet basis. α_k is adjusted every T samples as a moving average of this power. Empirically, we found $\gamma = 0.4$ to be effective. We control the relative volume of the marker to the original game audio with C. We study the effects of C on audibility and ISD measurement in §6.

Marker Cross-Correlation. Given the headset recording $x_{rec}[t]$, its cross-correlation $Z[t]$ (depicted in Figure 5a) can be computed with the marker as,

$$Z[t] = \sum_{i=1}^{L} x_{rec}[t+i].w[i] \tag{3}$$

$Z[t]$ will have sharp peaks, when the markers in $x_{rec}[t+i]$ and $w[i]$ align. This is depicted in Figure 5a as seen by the spikes in the $Z[t]$. However, game audio amplitude varies significantly over time, which affects the amplitude of $Z[t]$. In order to reliably detect a peak in the face of changing game audio levels, we normalize the correlation by dividing it by the power in the cross-correlation signal over a signal history window size S. Further to account for sampling offsets that result in negative and positive cross-correlation peaks we use the absolute value of the normalized correlation as,

$$Z^*[\tau] = \left| \frac{Z[\tau]}{\sqrt{\frac{1}{S}\sum_{i=1}^{S} Z[\tau+i]^2}} \right| \tag{4}$$

As seen from Figure 5b, $Z^*[t]$ has an almost constant envelope due to normalization and the peaks are more pronounced even in sections where the game audio amplitude was low. In practice we found that $S = 100$ ms works well, perhaps because typical game sounds such as game effects and speech phonemes last 100–250 ms.

Cross-Correlation Peak Filtering and Extraction: Peaks in a signal are the local maxima in the signal envelope. Thus, in order to detect the peaks, we first compute the envelope of $Z^*[t]$ (depicted in Figure 5c) as,

$$R[t] = \begin{cases} |Z^*[t]| & if\ R[t-1] \le |Z^*[t]| \\ \beta.R[t-1] & otherwise \end{cases} \tag{5}$$

In Equation (5), to compute the envelope we chose the decay parameter β as 0.99995 to ensure that the envelope of an impulse decays to about 0.1 within 1 second. This choice was motivated by the interval of our markers which is 1s. We then identify, points where the gradient of the envelop changes from positive to negative as peaks. After normalization $Z[t]$ has a standard deviation of 1. Thus, to pick reliable peaks, we choose only peaks above a minimum peak threshold of $\theta = 5$ as indicated by the orange spots in Figure 5c in $R[t]$. This choice of θ is carefully driven by an analytical model, described in detail in Appendix §A. Thus, the peaks function $P[t]$ (as indicated in Figure 5c) is computed as,

$$P[t] = \begin{cases} R[t] & if\ R[t-1], R[t+1], \theta < R[t] \\ 0 & otherwise \end{cases} \tag{6}$$

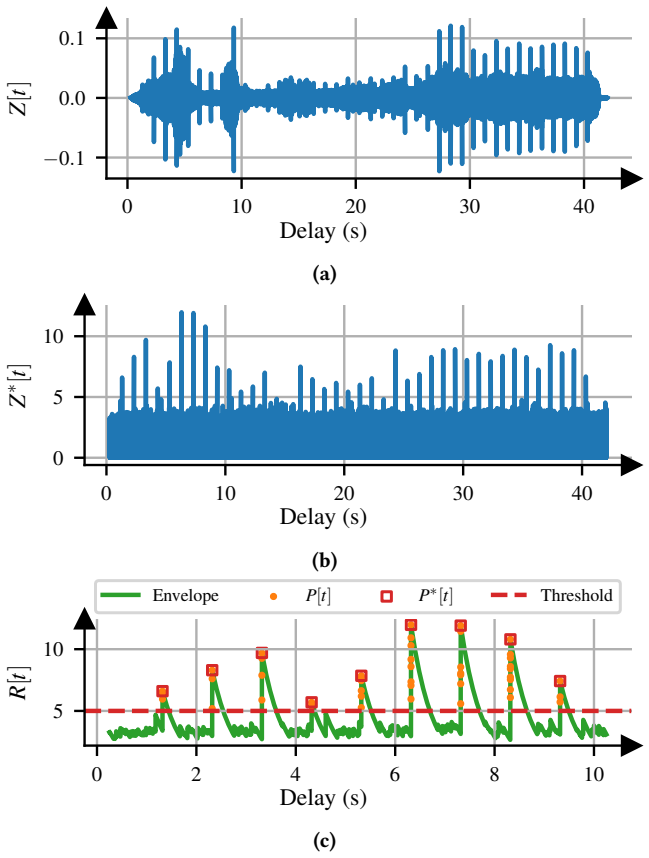


Figure 5: (a) Unnormalized cross-correlation, (b) normalized cross-correlation, and (c) the envelope.

Filtering based on consecutive peaks and local maxima. We filter peaks that have larger peaks than them less than δ samples away. We also expect peaks to periodically recur every L samples. Thus, we remove all peaks that do not have a peak L samples later allowing for an error of up to δ samples (shown in Figure 5c) as,

$$P^*[t] = \begin{cases} P[t] & if\ P[t] = \max_{j\in[-\delta,\delta]} P[t+j] > 0 \\ & and\ \max_{j\in[-\delta,\delta]} P[t+L+j] > 0 \\ 0 & otherwise \end{cases} \tag{7}$$

In Equation (7), $L = 48000$, the number of samples in 1s worth of audio data. The peaks found by $P^*[t]$ are extremely reliable with almost zero false positives as modeled in §A and evaluated in §6.

Periodic Marker Addition In order to continuously track ISD, we must periodically add markers to audio streams. Two considerations dictate the choice of periodicity. First, the period should be small enough to respond to changes in the ISD. Second, it should be larger than twice the maximum possible value of ISD, to avoid accidentally estimating ISD with the incorrect marker (as discussed in §4.3). In our implementation we chose an interval of 1s assuming conservatively a maximum possible ISD below 500 ms. Finally, the longer the PN-sequence, the higher its detection rate. Thus, we

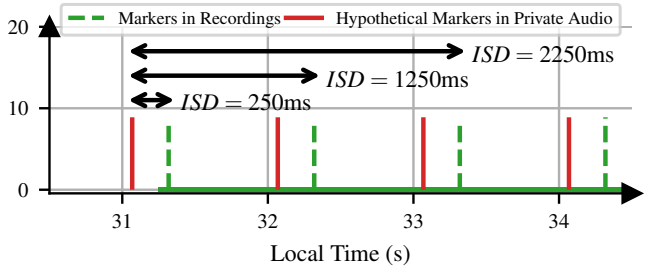


Figure 6: Marker matching, when marker length L is 1 second. If the marker length is long enough, the smallest delay is ISD.

chose our PN-sequence length to be 1 second, *i.e.,* 48000 samples long.

4.3 ISD Estimation

The headset/controller records time T_i^{chat} when chat audio frame i was captured, and logs the playback time $T_j^{accessory}$ when the j^{th} accessory-stream audio frame was played at the headset. Let us define $t = 0$ at $P[t]$ to correspond to local time T_0^{chat}, at the controller. If a marker was found at sample τ (corresponding to $P^*[\tau] = 1$ from Equation (7)), then this implies that it was detected at local time $T_0^{chat} + \frac{\tau}{F_s}$ at the headset/controller (green lines in Figure 6). Assuming the server informs the headset/controller that a marker was added at the beginning of audio packet j in the screen audio stream, then we know $T_j^{accessory}$ is the local time at which the accessory-stream would have had a hypothetical marker (red lines in Figure 6). The final step is to align these two signals. This is done by looking for the smallest shift in time that aligns them and we can estimate ISD with this shift. In order for this scheme to function correctly, the interval between two markers should be greater than twice the greatest possible ISD since ISD can be negative or positive. As discussed in §4.2 we chose a duration of 1 second, under the assumption that ISD does not exceed 500 ms.

4.4 Ekho Delay Compensation

Using the ISD measured as feedback, Ekho synchronizes the two streams. This synchronization is achieved by either skipping samples in the stream with higher latency or slowing down the stream with lower latency by injecting a silence period equal to the ISD once. Thus, ISD is measured once every second and continuously eliminated to keep the streams synchronized. Since injecting silence periods can deteriorate audio quality, a better alternative is to use packet loss concealment techniques [16, 19] and add interpolated audio instead of silence periods. We leave this enhancement to future work.

5 DESIGN AND IMPLEMENTATION

Often headsets/controllers are limited in terms of compute and power resources to compute correlations in real-time. Consequently, Ekho estimates ISD at the cloud server rather than at the headset/controller. Further, operations such as marker addition and delay compensation are also better implemented as server side operations. Thus, Ekho adopts an architecture that enables compute intensive operations to be performed in the cloud.

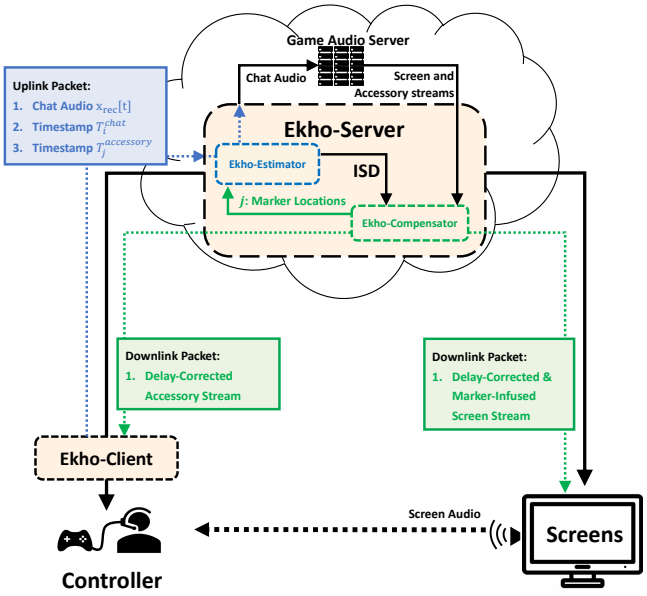


Figure 7: Systematic overview of Ekho.

We leverage the underlying architecture of existing online gaming system. In all cloud gaming systems, chat audio is packetized and sent to the game server. Thus, chat audio destined to the gaming server can be intercepted in the cloud to detect markers in them. Similarly, delay compensation in Ekho is achieved by modifying the audio streams by either adding extra zero samples (silence) or skipping game audio samples. This can be done by intercepting the audio streams generated by the gaming server in the cloud and modifying it prior to sending them to the player's devices.

5.1 Architecture

Ekho comprises two main components; Ekho-Server located at the cloud gaming server, and Ekho-Client, situated at the controller with a headset. Ekho-Server itself is composed of two modules: Ekho-Compensator for delay compensation and marker addition, and Ekho-Estimator for ISD measurement. Figure 7 visualizes the design of Ekho.

Ekho-Compensator. Ekho-Compensator intercepts the screen and accessory streams from the gaming server and modifies them in two ways before forwarding them to the player's devices. First, it receives ISD measurements from Ekho-Estimator, and compensates for them. When ISD is positive, *i.e.,* the accessory stream is playing before the screen stream, we inject extra 'silence' frames to the accessory stream. The reverse is done when ISD is negative. When an already compensated delay shrinks, *e.g.,* accessory stream preceded by 80 ms before, but now precedes by 60 ms, we need to revert corrected delay. Reverting corrected delay is done by skipping frames (or temporarily faster playback) at the streaming device that was previously delayed. The second role of Ekho-Compensator is that

it adds PN-sequence markers to the screen audio stream and logs audio frame IDs containing the start of the markers.

As a minor optimization for stability, correction is only initiated when the current synchronization error is markedly large. ISD can change slightly due to network reasons, encoding changes or hardware throttling and we suggest 5 ms as a minimum difference before correction is needed. When ISD correction begins, it will take several seconds before it reflects in measurements. During this temporary period, Ekho-Compensator ignores any new ISD measurements.

We expect a preceding screen audio stream to be rare; the screen audio stream accompanies a bandwidth-hungry video stream that adds transport latency, and must compensate for video decoders. In contrast, the accessory stream only has audio, tactile and input streams, and the headset/controller will usually include fast hardware decoding. As such, the one-way latency to the accessory stream is expected to be smaller. The rare exceptions are when the screen stream goes over a faster network than the accessory stream. For example, screen stream going over broadband, while the controller is connected to a public WiFi access point with many active users.

Ekho-Estimator. Ekho-Estimator receives chat audio packet with ID i from the controller/headset along with timestamps from Ekho-Client T_i^{chat}. It also receives the time-stamps corresponding to the playback time for each accessory-stream packet with ID j, $T_j^{accessory}$ from Ekho-Client. Meanwhile, as described earlier in this section, Ekho-Compensator already logs the audio frame IDs j where a marker was added to the screen audio stream. Ekho-Estimator measures ISD by following the technique described in §4.2 and §4.3. Ekho-Estimator needs 2-5 seconds of recording before a robust ISD can be measured. During this initial 2-5 second period, the streams are not fully in sync.

Ekho-Client. Ekho-Client is a lightweight component on the controller that communicates with Ekho-Server and sends back chat audio, appended with two sets of timestamps; **(1)** Chat audio local timestamps T_i^{chat}, and **(2)** Accessory audio playback timestamps $T_j^{accessory}$. Note that in most situations, we do not have access to the screen device API (such as when streaming to a WebRTC screen client), and thus we design Ekho without any cooperation from the screen device API in mind.

5.2 Implementation

We implement Ekho-Server in ~ 1000 lines of C++ code. In our implementation, we use FFTW 3.3.10 [9] for signal processing. The bulk of the compute involved with Ekho is done in Ekho-Server, and our implementation utilizes a modest 2.5% of a single core of an Intel 2.3GHz core i9 processor (I9-9880H) and allocates a maximum of 83MiB of memory at peak.

Ekho measures an end-to-end ISD, except for T_{Play}^{P}, *i.e.,* the hardware latency. This is not an issue, however, as controllers have low hardware latency (sub-millisecond). Furthermore, the controller designer can measure the hardware latency and inform Ekho, as we do in our end-to-end test.

6 EVALUATION

In this section, we aim to evaluate Ekho, and analyze the effect of certain conditions on its accuracy. Specifically, we answer the following questions:

(1) **§6.1.** Does Ekho successfully synchronize audio streams in an end-to-end system?
(2) **§6.2, §6.3**. What marker volume is sufficient for accurate ISD estimation while being imperceptible to humans?
(3) **§6.4.** How does marker-assisted ISD measurement fare against prior work?
(4) **§6.5.** Can Ekho be used for video to audio synchronization when the screen audio stream is muted?

We include further ablations on Ekho's marker estimation with respect to various encoding and microphone type in §B and §C.

6.1 End-to-end synchronization

We evaluate Ekho with synchronizing two WebRTC streams. In this test, a central game server streams the same audio to two different devices. One device, acting as the screen and connected to a cellular network, simply plays back the audio. The other device, acting as the controller and connected to a campus WiFi endpoint, plays the audio back, sends back recordings from a microphone to the server, and also sends back timestamps of the recordings and playback. We measure the ISD of the streams throughout the test. We use an Xbox Stereo Headset as the microphone connected to the controller.

At the start of each session, the streams will be out of sync, and Ekho will synchronize them in several seconds. As such, we shall ignore the first 5 seconds in each stream in our evaluation. If there was no packet loss that depletes the jitter buffer and changes ISD, the streams would remain in sync indefinitely. In our tests however, packet loss and jitter buffer depletion was common, and led to Ekho having to resynchronize the streams frequently, with a response time of 4 to 6 seconds.

Setup. We use a Python implementation of the WebRTC library [15]. To play audio or record from a microphone, we use PyAudio [28], a python binding of PortAudio [3]. We use input ADC and DAC timestamps for T_i^{chat} and $T_j^{accessory}$, respectively. The controller device is a MacBook Pro 2019, the screen device is an ASUS ROG G551JW, the game server is an AWS c4.large instance in the East 1 or East 2 region. All audio is encoded with OPUS [35] in 20 ms frames. We also correct ISD by adding or skiping 20 ms frames in this implementation, which means we can have errors up to 10 ms. Note that a more involved implementation could add or skip fractions of frames, and synchronize below the 10 ms bound.

ISD Measurement Methodology. To calculate the end-to-end ground-truth ISD, we have both devices log the time at which they play each audio frame. Since the device clocks are not in sync, we need one ISD measurement from another source to synchronize these logs. To achieve that, we add a 2KHz to 5KHz chirp (a frequency sweeping audio) to the start of the screen audio, and a 5KHz to 2KHz chirp to the start of the controller audio. A microphone from a third device listens to the playback from both devices, and by correlating each chirp to the recording, we extract the initial ISD, which then synchronizes the two device's logs.

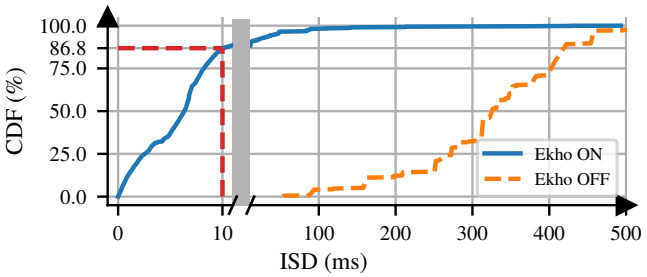


Figure 8: CDF of ISD throughout several tests. Despite consistent jitter, Ekho maintains ISD below 10 ms more than 86% of times.

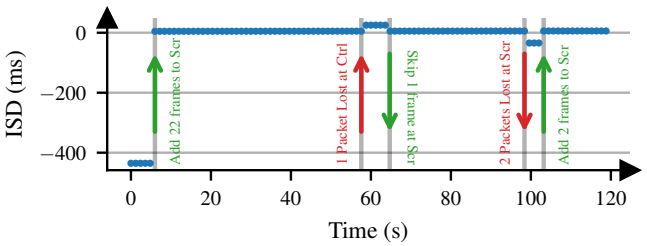


Figure 9: ISD in an example WebRTC session with Ekho.

Results. Figure 8 displays the end-to-end ISD with and without Ekho, across 6 sessions, each of 5 minutes length, for a total of 30 minutes of streaming. Ekho maintains synchronization with the 10 ms bound for 86.8% of the time across all streams. The remaining 13.2% is due to 0.03% packet loss, each of which leads to several seconds of out-of-sync playback. Note that without Ekho, the streams never reach this bound. As an illustrative example, Figure 9 shows the ISD across time in one session from the East 2 instance. First, the controller leads the screen playback by 436 ms, and Ekho adds 22 frames of 20 ms length to the screen audio to synchronize them. Then, a packet loss event at $t = 57.6s$ at the controller stream leads to the playback missing one frame and jumping ahead by 20 ms, which causes the controller to now lead by 24 ms. Ekho amends this after 6s by skipping one packet at the screen audio. A second packet loss at $t = 98.4s$ at the screen side leads to the screen leading by 40 ms, and Ekho reacts to this after 4 seconds by adding 2 frames to the screen side.

6.2 Marker Audibility

As Ekho is based on adding PN-sequences to the screen-stream audio, it is vital that such markers are imperceptible to the user's ear, and do not distract the player. C in Equation (2) controls the marker volume, and for small enough values, the marker should not be audible. In this section we ask the question: *What is the range of C, in which, marker addition does not affect user experience?*

Test Methodology. As in echo threshold in §3.1, we use the DCR test methodology of the ITU-T P.808 standard [24]. We use the same corpus of 30 game audio clips from 15 popular online games (details in Table 2 in the appendix). To each of the clips we add PN-sequence markers according to Equation (2), but with varied C values, ranging from $C = 0.1$ to $C = 5$.

Human testers first listen to the reference audio clip, and then to the marker infused clip with an unknown C, possibly $C = 0$, *i.e.,* no PN-sequence marker. They are asked to select one of several choices indicating the level of distraction arising from the degradation. We collected a total of 2010 ratings, removed low-quality responses (36% of total data) in accordance with ITU-T P.808 [24]. Each clip was rated ≈ 6 times, and with 30 different clips for each marker volume across all games, this results in a total of ≈ 186 votes per level. The respondent pool consisted of 28% young adults (18-25), 32% adults (26-35), 25% middle-aged individuals (36-50), and 14% seniors (51 and above), and their native language was English. We follow the same ethical guidelines in §3.1, and Microsoft IRB reviewed and approved this survey.

Results. As in §3, we measure audibility for three different categories of game audio: speech, music and game sound effects. Average opinion scores along with 95% confidence intervals for each of these three categories are presented in Figure 10. As seen in Figure 10, across all categories, C up to 1.0 does not significantly change user experience respective to the reference. At $C = 2.5$ the marker is audible and slightly distracting.

6.3 Marker Detection

In this section we ask the question: "What is minimum value of C for which markers are detectable by Ekho and provide accurate ISD estimates?" This will allow us to choose a value of C that is not audible (§6.2), and can be used to estimate ISD reliably by Ekho.

Measuring ISD Groundtruth. In order to measure ISD estimation error by Ekho we need the groundtruth ISD. To this end we introduce a loud 20s long chirp spanning 0Hz to 20KHz at the beginning each audio clip. This chirp serves as a reliable as start of clip marker to help estimate the groundtruth delay.

Experiment Methodology. We record game audio clips with varying marker volumes, *i.e.,* the corpus of samples in the DCR study. We then provide Ekho-Estimator with the recording and timestamps, and log ISD measurements it provides. For timestamps, we set $T_i^{chat} = (i-1) * 20ms$, $T_j^{accessory} = (j-1) * 20ms - x$ where x is the ground truth ISD Ekho-Estimator must measure, and we add markers every second ($j = 50.n, \forall n \in \mathcal{N}$). We vary x from −300 to 300 ms.

Hardware. For recording, we use a $45 Xbox Stereo Headset, representative of the typical headset used in gaming sessions. We use a MacBook Pro 2019 as the screen playing the screen game audio. The screen volume was the same across all recordings, and configured for the typical volume in gaming sessions (60-70 dBA,[4] since game volume varies).

Encoding. We encode the recording with varying compression levels. We use FFMPEG (version N-107374) [34] and specifically libopus to encode and decode clips. Unless otherwise stated, we use an OPUS compression scheme with 32kbps of bitrate budget, super-wide-band (SWB) mode (frequency spectrum kept up to 12KHz), a level 4 search complexity and application set to lowdelay.

[4]A-weighted sound level (dBA), standardized in ISO 226:2003, normalizes frequency amplitudes to be heard at equal levels by a human ear [25]

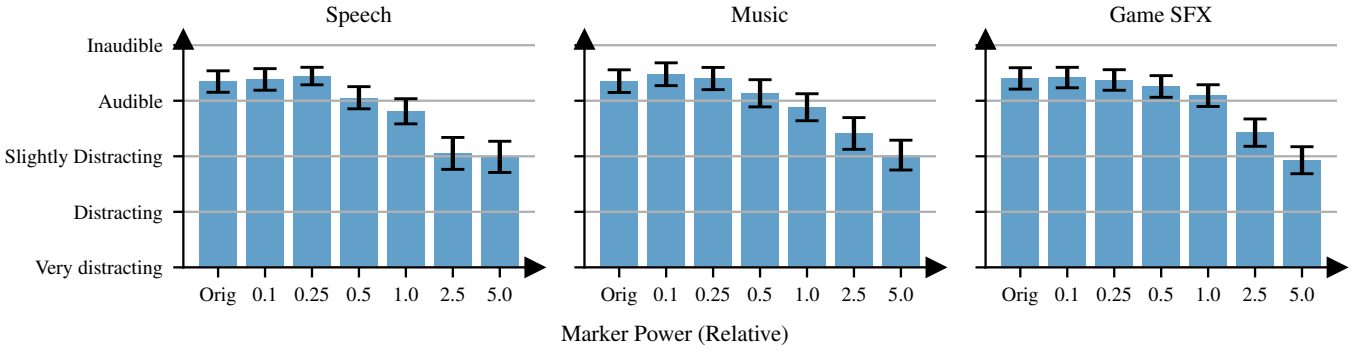


Figure 10: Crowdsourced opinion scores for how the addition of the marker affects user experience. Up to a relative power of 1.0, user experience remains comparable to the reference.

Metric. We assess Ekho-Estimator by looking at **(1)** how accurate ISD measurements are, and **(2)** the ISD measurement rate. For each marker, we could potentially have one ISD measurement, and hence we define this rate as the number of measurements over the number of markers in the sample.

Results. Figure 11 shows histograms (across 30 sample clips) of the ISD estimation error, and ISD measurement rate. We observe that with $C \geq 0.25$, ISD error is less than 1 ms, less than our limit of 10 ms and the lowest echo thresholds from prior work [18]. With the lowest volume $C = 0.1$, Ekho has errors surpassing 10 ms in 1.5% of measurements. All markers (450 total) are detected for $C > 0.5$, and for $C = 0.25$ in worst cases we get one measurement every 4 seconds. With $C = 0.1$ however, there is one sample where no measurement is found. Overall, this experiment suggests that for $C > 0.25$, we should comfortably detect markers and accurately estimate ISD.

Conclusion: Suitable range of C Thus, based on the audibility range for C in §6.2 with the range for reliable detectability, we observe that any $C \in [0.25\ 1.0]$ should be detectable by Ekho-Estimator, and at the same time inaudible to the user and will not effect the gaming experience. Consequently, we pick $C = 0.5$ as our choice of C.

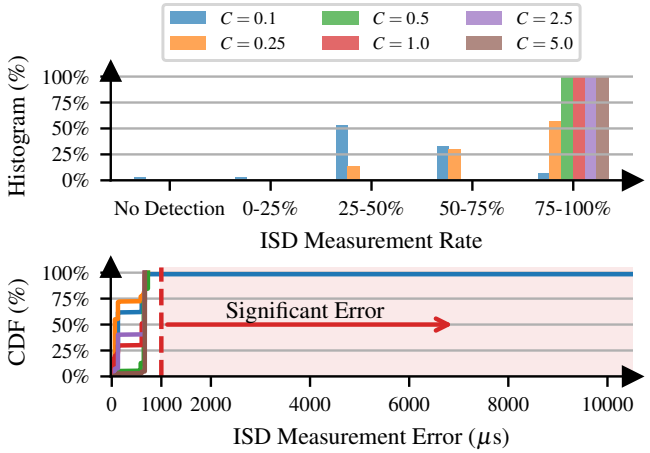


Figure 11: Studying Ekho with six different marker volumes. Based on Figure 10 and these results, ideally $C \in [0.25\ 1.0]$**.**

6.4 Ekho vs. GCC-PHAT

We now compare Ekho to GCC-PHAT [5], a state-of-the-art cross-correlation technique that does not use markers. Since background chatter is common in indoor gaming environments, we repeat the experiment in §6.3, but in addition to the default chatter-free game audio, we conduct additional experiments that include *background voice chatter*. Here, a speech clip is played alongside the game audio, and we vary the speech clip's loudness in three levels, such that the median sound level of the speech clip is: **(Low Chat)** 5 dBA lower than the game audio, **(Med Chat)** as loud as the game audio, **(Loud Chat)** 5 dBA louder than the game audio.

We use GCC-PHAT to measure ISD. At a high level, GCC-PHAT works by computing the Fourier transform of the accessory audio $x[t]$, $X(\omega)$, and the recorded audio $x_{rec}[t]$, $X_{rec}(\omega)$. Then, it computes ISD in the following way:

$$R(\tau) = \int_{-\infty}^{+\infty} \frac{X(\omega)X_{rec}(\omega)}{|X(\omega)X_{rec}(\omega)|} e^{j\omega\tau} d\omega$$
$$ISD = argmax_\tau R(\tau) \quad (8)$$

It is noteworthy that unlike Ekho, GCC-PHAT always produces a measurement, whether confident or not. Thus, there are cases where the error is larger than a second. We believe in practice ISD measurements higher than 300 ms can be flagged as erroneous, and therefore we ignore such values for GCC-PHAT, treating them as missed detections. We attempted to use the same normalization and filtering strategy we used with Ekho, but were not successful in improving GCC-PHAT.

Results. Whenever Ekho and GCC-PHAT are able to measure ISD (*i.e.,* ignoring outliers for GCC-PHAT), they achieve good accuracy (< 2 ms ISD error). However, Ekho is much more robust in detecting ISD across all chatter levels. In Figure 12 we present the ISD measurement rate for Ekho and GCC-PHAT. Even without background chatter, GCC-PHAT fails to produce any ISD measurements for a third of our corpus, while Ekho has a consistent ISD measurement every second. When we introduce background chatter,

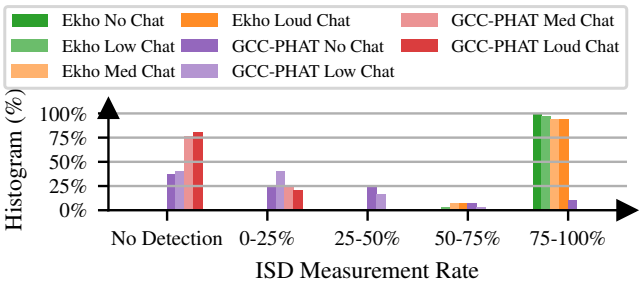


Figure 12: Comparing Ekho with GCC-PHAT [5] against varying background chatter.

GCC-PHAT fails to produce any measurement for more than 75% of the corpus, while Ekho observes a modest drop in detection rate. Overall, Ekho is robust and and maintains a superior detection rate to GCC-PHAT, while its added markers are proven to be inaudible to human subjects.

6.5 Video-to-Audio Sync

When the screen audio is muted (*e.g.,* to avoid disturbing others), the player uses a headphone attached to the controller. In such cases, echo between the screen and accessory audio is not a problem, but we still want the headphone audio and haptic feedback to be in sync with the video being displayed. In these cases, the cloud gaming software can send PN-sequence markers at a constant amplitude (instead of dynamically based on game audio in Equation (2)). We study what a viable amplitude for this scenario is such that **(1)** the marker is detectable for Ekho-Estimator, and **(2)** is not noticeable. While evaluating the first condition is trivial, for the second, we measure the sound levels (dBA) across varying amplitudes for the PN-sequence audio using a commercial sound level meter. For the microphone, we use three types of microphones, described in §B. Figure 13 presents the max ISD error, the detection rate, and the PN-sequence's sound level for varying amplitudes. Where a detection is available, Ekho-Estimator maintains sub-millisecond accuracy. All microphones allow for detection for amplitudes of 6 dB and above. Up to an amplitude of 15 dB, the PN-sequence marker sound level is lower than 40 dBA, a typical sound for a quiet library. This suggests that we can use amplitudes in the range of $[6dB\ 15dB]$ and satisfy both conditions.

7 CONCLUSION

We present Ekho, a robust end-to-end technique for synchronizing media playback over the network. Ekho uses recordings from one device to measure the delay between playbacks, compensates for them directly at the server. Ekho does not require a shared network or cooperation from the screen device, is resilient to aggressive lossy compression in the recording, and remains robust to microphone quality and background noise/chatter. Importantly, Ekho outperforms prior work in such realistic conditions.

We exhaustively evaluate Ekho, and perform standardized tests to measure the impact of Ekho' technique on user experience, as well as the impact of unsynchronized playback. We implement and test Ekho, both from a computational overhead standpoint,

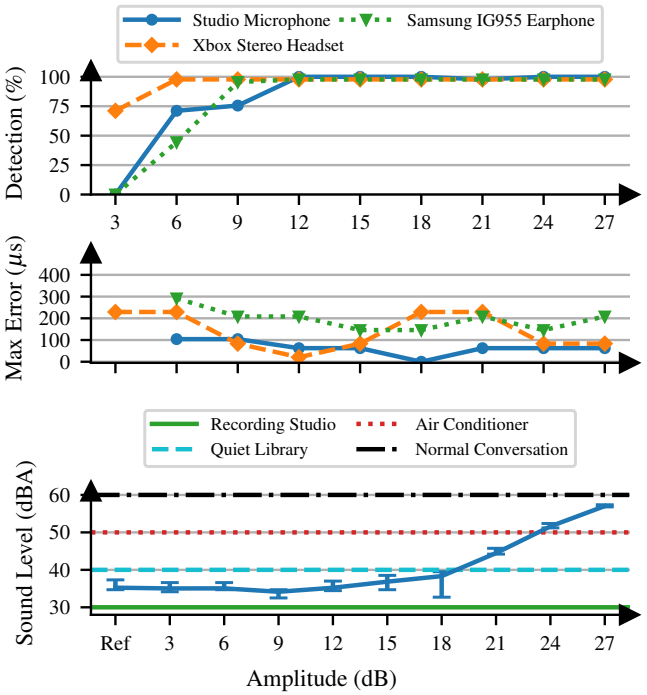


Figure 13: Studying Ekho when screen-stream audio is muted, and the software plays PN-sequences with low amplitude. An Amplitude of 6 dB is enough for marker detection, and up to an amplitude of 15 dB, PN-sequences do not noticeably elevate sound level.

and with a live test with WebRTC sessions over a jittery network path. Ekho continuously recovers synchronization and a smooth playback experience. Ekho's main goal is synchronizing two audio streams, but can also be used to synchronize an audio stream on one device, and a video stream on another.

ACKNOWLEDGEMENTS

We are grateful to our reviewers for insightful comments. We thank Babak Naderi and Ross Cutler for their help with the ITU-T P.808 standard, and Thomas Puget Abadie for helping us understand the Xbox cloud streaming setup.

REFERENCES

[1] Bernard D. Adelstein, Durand R. Begault, Mark R. Anderson, and Elizabeth M. Wenzel. 2003. Sensitivity to Haptic-Audio Asynchrony. In *Proceedings of the 5th International Conference on Multimodal Interfaces (ICMI '03)*. Association for Computing Machinery, New York, NY, USA, 73–76. https://doi.org/10.1145/958432.958448

[2] WiFi Alliance. 2017. Wi-Fi CERTIFIED TimeSync™ brings precise synchronization to Wi-Fi® devices. https://www.wi-fi.org/news-events/newsroom/wi-fi-certified-timesync-brings-precise-synchronization-to-wi-fi-devices. (2017). [Accessed 20-Jul-2023].

[3] Ross Bencina and Phil Burk. 2001. PortAudio-an open source cross platform audio API. In *ICMC*.

[4] Fernando Boronat, Dani Marfil, Mario Montagud, and Javier Pastor. 2018. HbbTV-Compliant Platform for Hybrid Media Delivery and Synchronization on Single- and Multi-Device Scenarios. *IEEE Transactions on Broadcasting* 64, 3 (2018), 721–746. https://doi.org/10.1109/TBC.2017.2781124

[5] Maximo Cobos, Fabio Antonacci, Luca Comanducci, and Augusto Sarti. 2019. Frequency-Sliding Generalized Cross-Correlation: A Sub-band Time Delay Estimation Approach. (2019). https://doi.org/10.48550/ARXIV.1910.08838

[6] Advanced Television Systems Committee et al. 2003. ATSC implementation subcommittee finding: Relative timing of sound and vision for broadcast operations. *IS-191* 26 (2003).

[7] Matthias Dick, Oliver Wellnitz, and Lars Wolf. 2005. Analysis of Factors Affecting Players' Performance and Perception in Multiplayer Games. In *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames '05)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/1103599.1103624

[8] Ngoc Q. K. Duong, Christopher Howson, and Yvon Legallais. 2012. Fast second screen TV synchronization combining audio fingerprint technique and generalized cross correlation. In *2012 IEEE Second International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*. IEEE, Berlin, Germany, 241–244. https://doi.org/10.1109/ICCE-Berlin.2012.6336458

[9] Matteo Frigo and Steven G. Johnson. 2005. The Design and Implementation of FFTW3. *Proc. IEEE* 93, 2 (2005), 216–231. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[10] Mojgan Ghasemi, Partha Kanuparthy, Ahmed Mansy, Theophilus Benson, and Jennifer Rexford. 2016. Performance Characterization of a Commercial Video Streaming Service. In *Proceedings of the 2016 Internet Measurement Conference (IMC '16)*. Association for Computing Machinery, New York, NY, USA, 499–511. https://doi.org/10.1145/2987443.2987481

[11] Helmut Haas. 1972. The influence of a single echo on the audibility of speech. *Journal of The Audio Engineering Society* 20, 2 (march 1972), 146–159.

[12] Christopher Howson, Eric Gautier, Philippe Gilberton, Anthony Laurent, and Yvon Legallais. 2011. Second screen TV synchronization. In *2011 IEEE International Conference on Consumer Electronics -Berlin (ICCE-Berlin)*. IEEE, Berlin, Germany, 361–365. https://doi.org/10.1109/ICCE-Berlin.2011.6031815

[13] Fortune Business Insights. 2023. Cloud Gaming Market Size, Share, Growth &; Forecast [2030]. https://www.fortunebusinessinsights.com/cloud-gaming-market-102495. (2023). [Accessed 20-Jul-2023].

[14] Cullen Jennings, Florent Castelli, Henrik Boström, Jan-Ivar Bruaroey, Adam Bergkvist, Daniel C. Burnett, Anant Narayanan, Bernard Aboba, and Taylor Brandstetter. 2023. WebRTC: Real-Time Communication in Browsers. https://www.w3.org/TR/webrtc/. (2023). [Accessed 20-Jul-2023].

[15] Jeremy Lainé. 2023. aiortc. https://github.com/aiortc/aiortc. (2023). [Accessed 20-Jul-2023].

[16] Bong-Ki Lee and Joon-Hyuk Chang. 2016. Packet Loss Concealment Based on Deep Neural Networks for Digital Speech Transmission. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 24, 2 (2016), 378–387. https://doi.org/10.1109/TASLP.2015.2509780

[17] Narimene Lezzoum, Ghyslain Gagnon, and Jérémie Voix. 2016. Echo threshold between passive and electro-acoustic transmission paths in digital hearing protection devices. *International Journal of Industrial Ergonomics* 53 (2016), 372–379. https://doi.org/10.1016/j.ergon.2016.04.004

[18] Ruth Y. Litovsky, H. Steven Colburn, William A. Yost, and Sandra J. Guzman. 1999. The precedence effect. *The Journal of the Acoustical Society of America* 106, 4 (1999), 1633–1654. https://doi.org/10.1121/1.427914 arXiv:https://doi.org/10.1121/1.427914

[19] Reza Lotfidereshgi and Philippe Gournay. 2018. Speech Prediction Using an Adaptive Recurrent Neural Network with Application to Packet Loss Concealment. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, Calgary, AB, Canada, 5394–5398. https://doi.org/10.1109/icassp.2018.8462185

[20] Team Luna. 2021. Amazon Luna – Quick Start Guide. https://amazonluna.blog/getting-started-a473f603c2d4. (2021). [Accessed 20-Jul-2023].

[21] Dani Marfil, Fernando Boronat, Almanzor Sapena, and Anna Vidal. 2019. Synchronization Mechanisms for Multi-User and Multi-Device Hybrid Broadcast and Broadband Distributed Scenarios. *IEEE Access* 7 (2019), 605–624. https://doi.org/10.1109/ACCESS.2018.2885580

[22] David L. Mills. 2012. Executive Summary: Computer Network Time Synchronization. https://www.eecis.udel.edu/~mills/exec.html. (2012). [Accessed 20-Jul-2023].

[23] Mario Montagud, Pablo Cesar, Fernando Boronat, and Jack Jansen. 2018. *Introduction to Media Synchronization (MediaSync)*. Springer International Publishing, Cham, 3–31. https://doi.org/10.1007/978-3-319-65840-7_1

[24] Babak Naderi and Ross Cutler. 2020. An Open Source Implementation of ITU-T Recommendation P.808 with Validation. In *Interspeech 2020*. ISCA, Shanghai, China, 2862–2866. https://doi.org/10.21437/interspeech.2020-2665

[25] International Standardization Organization. 2003. ISO 226: 2003(E): Acoustics—Normal Equal-Loudness-Level Contours. (2003).

[26] Abhinav Pathak, Himabindu Pucha, Ying Zhang, Y. Charlie Hu, and Z. Morley Mao. 2008. A Measurement Study of Internet Delay Asymmetry. In *Proceedings of the 9th International Conference on Passive and Active Network Measurement (PAM'08)*. Springer-Verlag, Berlin, Heidelberg, 182–191.

[27] Oswaldo Sebastian Peñaherrera-Pulla, Carlos Baena, Sergio Fortes, Eduardo Baena, and Raquel Barco. 2021. Measuring key quality indicators in cloud gaming: Framework and assessment over wireless networks. *Sensors* 21, 4 (2021), 1387.

[28] Hubert Pham. 2023. PyAudio: Cross-platform audio I/O for Python, with PortAudio. https://people.csail.mit.edu/hubert/pyaudio/. (2023). [Accessed 20-Jul-2023].

[29] Byron Reeves and David Voelker. 1993. Effects of audio-video asynchrony on viewer's memory, evaluation of content and detection ability. *Stanford Univ., Stanford, CA, Tech. Rep.* (1993).

[30] Earl D. Schubert and Joel Wernick. 1969. Envelope versus Microstructure in the Fusion of Dichotic Signals. *The Journal of the Acoustical Society of America* 45, 6 (1969), 1525–1531. https://doi.org/10.1121/1.1911633 arXiv:https://doi.org/10.1121/1.1911633

[31] Juan M. Silva, Mauricio Orozco, Jongeun Cha, Abdulmotaleb El Saddik, and Emil M. Petriu. 2013. Human Perception of Haptic-to-Video and Haptic-to-Audio Skew in Multimedia Applications. 9, 2, Article 9 (may 2013), 16 pages. https://doi.org/10.1145/2457450.2457451

[32] Charles Spence, Roland Baddeley, Massimiliano Zampini, Robert James, and David I Shore. 2003. Multisensory temporal order judgments: When two locations are better than one. *Perception & psychophysics* 65 (2003), 318–328.

[33] Xbox Support. 2023. Xbox Adaptive Controller Troubleshooting. https://support.xbox.com/en-US/help/account-profile/accessibility/adaptive-controller-wont-connect-to-xbox-or-pc. (2023). [Accessed 20-Jul-2023].

[34] Suramya Tomar. 2006. Converting video formats with FFmpeg. *Linux Journal* 2006, 146 (2006), 10.

[35] Jean-Marc Valin, Koen Vos, and Tim Terriberry. 2012. *Definition of the Opus Audio Codec*. RFC 6716. RFC Editor. 1–325 pages. https://www.rfc-editor.org/rfc/rfc6716.txt

[36] Ray van Brandenburg and Arjen Veenhuizen. 2013. Immersive second-screen experiences using hybrid media synchronization. In *MediaSync Workshop*. Nantes, France, 1–7.

[37] Di Xie, Sanjeev Mehrotra, Jin Li, and Y. Charlie Hu. 2013. *URCP: Universal Rate Control Protocol for Real-Time Communication Applications*. Technical Report MSR-TR-2013-64. Microsoft. https://www.microsoft.com/en-us/research/publication/urcp-universal-rate-control-protocol-for-real-time-communication-applications/

Appendices

Appendices are supporting material that has not been peer-reviewed.

A EKHO RELIABILITY

Here, we provide an analytical model for Ekho's peak detection method, and shall analyze the false-positive rate in this approach. First, let us consider the false positive rate for the outlier strategy. Recall that we define the cross-correlation metric in the following way:

$$Z[t] = \sum_{i=1}^{H} x_{rec}[t+i].w[i] \tag{9}$$

Concretely, let us separate the recording $x_{rec}[t]$ to a game audio component $\hat{x}[t]$ and a marker component $\hat{w}[t]$, and for simplicity, let us assume that the PN-sequence is not spectrum limited and thus the cross-correlation of $w[t]$ and $\hat{w}[t]$ with itself is 0 when not aligned properly. If we assess the cross-correlation at such times, $Z[\tau]$, we have:

$$\begin{aligned} Z[\tau] &= \sum_{i=1}^{H} (\hat{x}[\tau+i] + \hat{w}[\tau+i]).w[i] \\ &= \sum_{i=1}^{H} \hat{x}[\tau+i].w[i] \\ w[i] &\sim N(0, \sigma^2) \\ \Rightarrow Z[\tau] &\sim N(0, \sigma^2.\sum_{i=1}^{H} \hat{x}[\tau+i]^2) \\ Z[\tau] &\sim N(0, \hat{\sigma}[\tau]^2) \end{aligned} \tag{10}$$

In other words, for all non-aligning indexes τ, $Z[\tau]$ is a Gaussian random variable with an unknown standard deviation. Thus if $\bar{Z}[\tau]$ is a normalized cross-correlation we have:

$$\begin{aligned} \bar{Z}[\tau] &= \frac{Z[\tau]}{\sqrt{\frac{1}{S}\sum_{i=1}^{S} Z[\tau+i]^2}} \\ \Rightarrow \bar{Z}[\tau] &\sim N(0,1) \end{aligned} \tag{11}$$

Where is S is the length of the normalization window. Thus, if we set a threshold θ, the false positive rate is $FP = 1-\Phi(-\theta)+\Phi(\theta)$. For the choice of $\theta = 5$, the false positive rate is $2E-4\%$, which despite sound like very little, is actually one false positive sample every 10 seconds.

Now, let us consider the back-to-back filter strategy. If the chance of an outlier in the past analysis is $p << 1$, and we naïvely assume that false positives pass the envelope, the chance of a sample having two back to back outliers is $p.(1-(1-p)^{2\delta+1}) \approx (2\delta+1).p^2$. For $\theta = 5$ and $p = 2E-4\%$, this amounts two a false peak rate of $8E-8\%$. This means that there will be one false peak every 7 hours on average, which should be robust enough for a commercial setting.

B ABLATION: MICROPHONE QUALITY

To assess the effect of microphone quality on marker detection, we use three microphones, visualized in Figure 16, with frequency responses in Figure 17: **(1)** a $25 Studio microphone, representative of the highest recording quality, **(2)** a $45 Xbox Stereo Headset, representative of the typical headset used in gaming sessions, **(3)** a $4 Samsung IG955 Earphone, representing the lowest quality microphones usable.

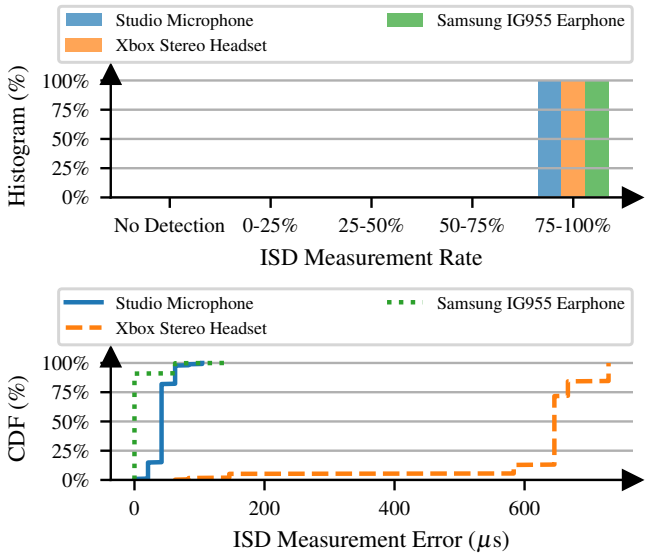


Figure 14: Studying Ekho with three different microphones. Ekho maintains sub-millisecond accuracy and high marker detection throughout different hardware.

We repeat the experiment in §6.3, and plot ISD error and measurement rate in Figure 14. As observable, despite extreme differences in frequency responses across these three microphones, Ekho maintains 100% marker detection and sub-millisecond accuracy throughout.

C ABLATION: ENCODING

Figure 15 demonstrates the effect of encoding level on measurements, when using the Xbox Stereo Headset. We consider no compression (lossless), a 32kbps bitrate budget, a 24kbps bitrate budget and 24kbps bitrate budget with a level 4 search complexity. While encoding does affect perceptible quality, it does not noticeably affect Ekho-Estimator's performance.

D DATASET

We use 30 audio samples from 15 popular online games, as listed in Table 2. These games span a variety of genres, from platform to racing and role-playing games, and samples consist of different stimuli such as speech, music and game sound effects.

E HARDWARE

We use three microphones in our evaluation (§6), visualized in Figure 16. In Figure 17, we present the frequency response of the recording channel, with different microphones. This channel consists of the speaker hardware, the physical environment surrounding the speaker to microphone path, and the microphone itself. We only change the microphone, and keep the speaker and physical

Game Title	Genre	Audio Category
CrossFireX	First Person Shooter	#1: Game Sound Effects
		#2: Game Sound Effects, Speech
GRID Legends	Racing Simulator	#1: Game Sound Effects, Speech
		#2: Game Sound Effects
Resident Evil Village	Survival Horror	#1: Speech
		#2: Game Sound Effects
Death's Door	Isometric Action-Adventure	#1: Music
		#2: Music, Game Sound Effects
Halo Infinite	First Person Shooter	#1: Game Sound Effects
		#2: Speech, Game Sound Effects
Sable	Adventure & Exploration	#1: Music, Game Sound Effects
		#2: Music
Dying Light 2 (RWDI)	Action Role Playing Game	#1: Speech
		#2: Speech
OlliOlli World	Sports Action Platformer	#1: Music, Game Sound Effects
		#2: Music, Game Sound Effects
Tales of Arise	Action Role Playing Game	#1: Speech, Music
		#2: Speech, Music
Elden Ring	Soulsborne Role Playing Game	#1: Game Sound Effects
		#2: Game Sound Effects
Ori and the Will of the Wisps	Metroidvania Platformer	#1: Game Sound Effects, Music
		#2: Game Sound Effects, Music
The Artful Escape	Adventure Platformer	#1: Speech, Music
		#2: Speech, Music
Forza Horizon 5	Racing Simulator	#1: Music, Speech
		#2: Game Sound Effects, Music, Speech
Psychonauts 2	Adventure Platformer	#1: Speech
		#2: Speech
Tormented Souls	Psychological Horror Shooter	#1: Speech, Music
		#2: Game Sound Effects, Music

Table 2: Description of the game audio clips used in the evaluation. There are two clips per game, and all clips are 15 seconds long.

environment the same throughout the setup. As observed, the high quality studio microphone has a relatively flat frequency response (Figure 17a), especially in our region of interest of 6-12KHz, while the Xbox Stereo Headset has several peaks and troughs (Figure 17b). The Samsung earphone displays the worst response and deterioration, where the difference from the lowest to highest amplitude in the response exceeds 30dB (Figure 17c).

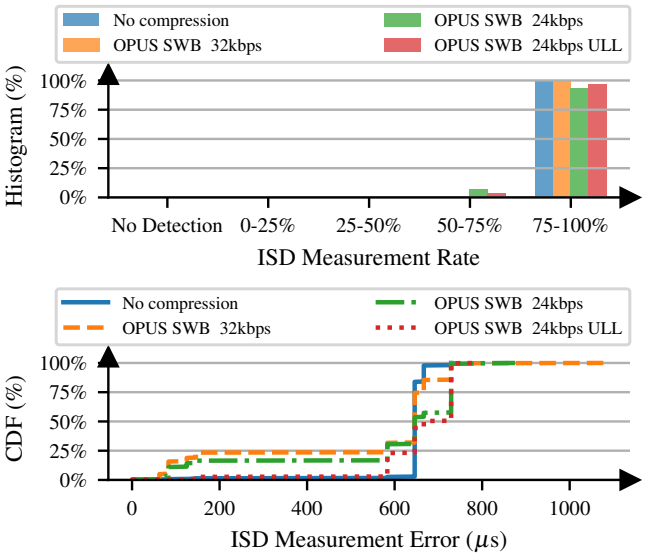
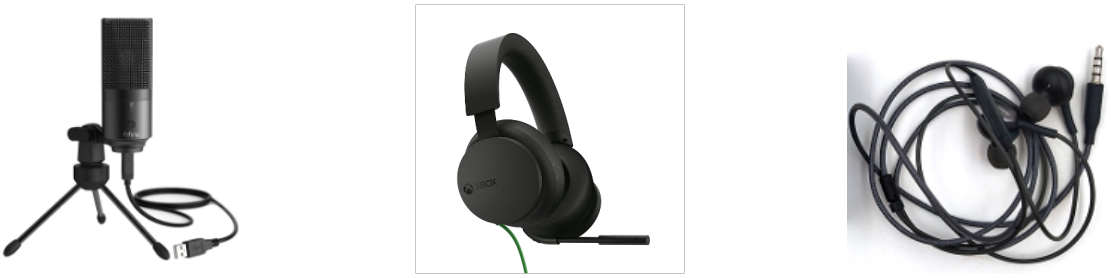
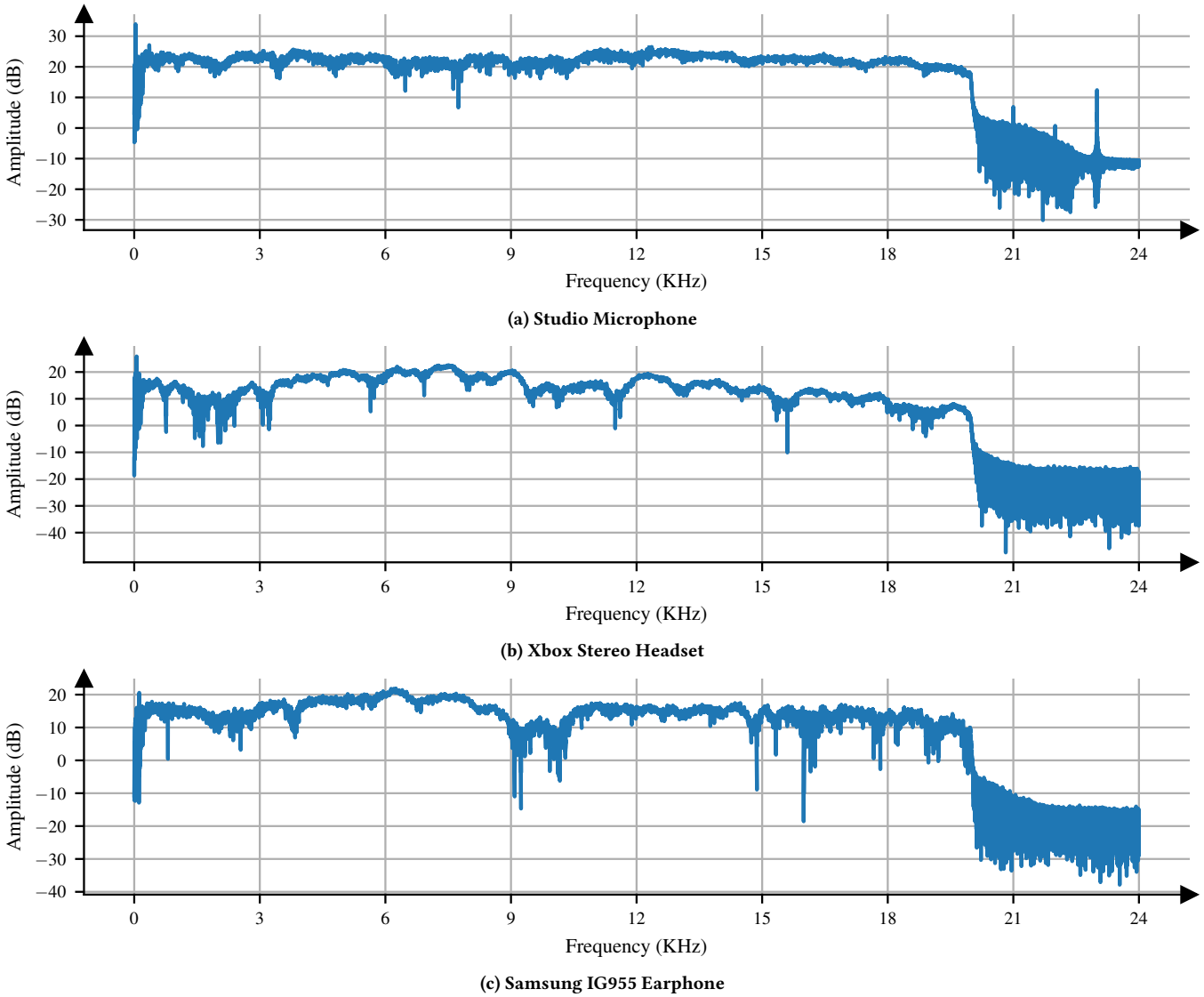


Figure 15: Studying Ekho with four different levels of compression. Aggressive encoding makes marker detection harder, but Ekho maintains a satisfiable level of detection throughout.



(a) Studio Microphone (b) Xbox Stereo Headset (c) Samsung IG955 Earphone

Figure 16: Microphones used in our experiments.



(a) Studio Microphone

(b) Xbox Stereo Headset

(c) Samsung IG955 Earphone

Figure 17: Frequency Responses for tested microphones. High quality microphones have flat frequency responses.