Kappa: A Synchronized Video Viewer

Zhan Xiong Chin
chinz@seas.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

David Liao
liaod@seas.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

Yoo Jin Kim
yoojkim@seas.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

Abstract—**E-sports is a market that is rapidly becoming popular and widespread, yet often does not see high quality software solutions catering to it. Professional e-sports teams, whose members typically play remotely, do not have good ways to analyze or review their games together. Instead, they rely on ad-hoc solutions using a combination of existing software, leaving considerable room for improvement.**

To address this need, we propose Kappa, a synchronized all-in-one video viewing platform that allows e-sports teams to view, annotate, and analyze video replays of games remotely. The main technical contribution is designing both a scalable backend that supports our custom syncing protocol as well as pioneering a frontend UI that solves existing complications that arise when integrating with other third party systems. We achieved the creation of a fluid and responsive UI that is intuitive and integrates well with Youtube. With scalability and cost in mind, the UI is designed to complement the backend by helping minimize server-side upstream communication. The backend architecture emphasizes flexibility and reliability, with communication latencies of less than 100ms to all clients under all loads using techniques that exploit differential payloads and minimize overhead communication costs.

I. Introduction and Problem definition

E-sports is a rapidly growing market, with a burgeoning number of professional teams such as Evil Geniuses or Fnatic entering the field in games such as DOTA 2 or League of Legends to compete for multi-million dollar prize pools in online and offline competitions. With so much at stake, these professional teams spend a considerable amount of their time practicing and attempting to improve their skills. For instance, a typical day for a professional player would be to spend the morning playing a number of matches, then reviewing their video replays as a team in the afternoon. Similar to athletes in other sports, any improvements to the training process can yield a significant impact on their success rates.

However, despite the importance of such training and analysis time, the processes that teams use for these are often woefully inadequate. Unlike a conventional professional sporting team, e-sports teams are often decentralized in nature, with their members potentially being in completely different cities or countries for most of the year. Thus, any shared analysis time tends to rely on ad-hoc solutions. For example, a team might communicate via multiple screenshots edited with Microsoft Paint, or by manually communicating to each other the relevant video timings in a video replay. Such ad-hoc solutions slow down the analysis process and significantly increase the chance of a miscommuncation, while making it difficult to remember the details of a previous analysis seession.

II. Solution outline

A. Overview

Our proposed solution to this need is Kappa, an all-in-one web platform that aims to make collaborative video analysis sessions easy to run. Our platform allows videos to be loaded from Youtube, synchronizing their playback across multiple browsers. A shared canvas overlaid across the videos allows for annotations and other details to be easily communicated to the other users of the platform.

B. Architecture

The main web server of the platform is in charge of handling login and account registration processes. It is hosted using a custom server written in Go. It also allows the user to create rooms (each room is a single shared video instance) and invite friends and teammates to join the relevant room. All data is stored in an SQL database.

The synchronized video viewer is in charge of tracking changes to the video's state (play/pause, volume, current playback time) and sending them to the sync server (described below). It is built in JavaScript and React on top of the YouTube IFrame Player.

The shared canvas provides a layer on top of the video which allows for annotations and drawings to be overlaid over the video. Any drawing or annotation made on the canvas is immediately synchronized with other clients via the sync server as well. It is also built in JavaScript and React.

The sync server is in charge of passing messages back and forth between the various clients to synchronize their video and canvas state. It is also written in Go and relies on WebSockets to provide a reliable and high-performance communication channel between itself and its clients.

III. Design decisions

A. YouTube Player

Initially, we implemented our own video player and was considering extending our platform to include video hosting. However, after conversations with potential users, we decided that a better choice would be to make use of YouTube's embedded player API instead, and extend our own functionality over that.

Making use of the YouTube player rather than writing our own player enables users to more easily make use of existing video resources such as previously hosted YouTube videos. This not only leverages YouTube's high performance content delivery network across all users viewing the same video, but also makes it easy for teams to analyze video replays from other competitions or tournaments, which are frequently posted to YouTube.

The tradeoff for this decision is that we do not have the same granularity when tracking the video state, leading to minor delays in responsiveness of the synchronization. Unlike most of e-sports and gaming, video viewing and analysis are not heavily time-sensitive, thus we deemed this delay in response tolerable, as long as throughput and reliability were maintained.

Another minor tradeoff was the development time needed to adapt our code to the new player. This was significantly mitigated by the modular design of our code. The video player was implemented in its own React component, allowing the change to be made quickly and efficiently with almost no impact on the rest of the codebase.

B. *Canvas drawing behavior*

The interaction between our two main front-end components, the video player and the canvas, offered multiple option s, of which the best choice was not initially clear. For instance, a small sample of the multiple possible behaviors we brainstormed are listed below:

1) The canvas and the video could be completely separate. Drawings on the canvas would be unaffected by the time listed in the video, only disappearing when erased.
2) Drawings on the canvas could be temporary, disappearing a fixed number of seconds (e.g. five seconds) after being drawn. They would be unaffected by video playback.
3) The ability to draw on the canvas would only be enabled when the video was paused, disappearing once the video is played again.
4) Drawings on the canvas would be permanent until erased, but would be "tagged" to a specific video timing, reappearing if the video was played through the same time.
5) Drawings on the canvas would be tagged to a specific video timing, only appearing when the video is playing at that time.

To resolve this issue, we approached the problem in three ways. Firstly, we made a list of use cases for the drawing feature, then established properties that the drawing behavior should display based on each use case. Secondly, we implemented the most compelling behaviors we thought of, then used it ourselves for a brief period to see how intuitive the interactions would be. Thirdly, we examined the technical features and tradeoffs (e.g. performance, memory required) needed for each behavior, using this information to supplement our design decision.

As an example of the first approach, one use case we came up with was for a user to circle a moving character on screen to draw attention to the character's position in the game. Analyzing this use case suggests that any permanent drawing not tied to a specific time in the video is intuitively unsatisfactory: the circle would be redundant at any other point in time. However, if the video is paused, the drawing should remain for as long as desired. Additionally, it would be preferred for a drawing to reappear if the video is rewinded to the same point in time, as a reminder to the idea communicated. Furthermore, if a user decides that the annotation or drawing is no longer relevant to that point of the video, there should be an easy way to delete it from the playback.

Based on this and the other two approaches listed, we eventually settled on the last option listed above: having drawings on the canvas tagged to a specific video timing. More concretely, we store drawings as a freeform collection of pixels, with each pixel recording the starting and ending time that it appears on the video. Initially, when a pixel is created, the pixel's starting time is tagged to the video time where it was drawn, with the ending time tagged to a fixed number of seconds after its starting time. This exhibits both the desired play and pause behavior we want, as well as the ability for drawings to reappear when a video is rewinded back to the same time. This also makes erases easy to handle: if a pixel is erased, we just set its ending time to the video timestamp where it was erased.

Looking at the technical aspect of this choice, it does not increase bandwidth significantly, since we need to send either per-pixel or per-stroke information regardless of which method was chosen. On the other hand, it does have the potential to impact performance, since now fast-forwarding, rewinding, etc. need to make look-ups against the entirety of the canvas pixel data across the entire video. Despite this concern, testing revealed that it was not a significant factor. We note that in the future, if need be, further optimizations can be made either by improving the data structures which store the canvas data, or by adding GPU acceleration to the canvas (each pixel can be treated individually from any other pixel, which is ideal for a GPU-based speedup).

IV. Technical depth

V. Evaluation of solution

VI. Technical depth

VII. Evaluation of solution

VIII. Implications of Project

IX. Future work

X. Conclusion

In conclusion, we have built an easy-to-use and scalable solution for a target demographic.

Furthermore, despite our focus on the gaming and e-sports market, our platform is fairly general in nature. We can foresee numerous use cases outside of e-sports, even with minimal tweaks to the system. For example, the platform could be easily adapted to education, by providing a shared virtual classroom setting with a teacher teaching via video stream while making use of the annotation feature to easily communicate information to students.

Acknowledgment

We would like to thank our senior design advisor Professor Boon Thau Loo for the invaluable support and advice that he has provided us throughout this project.

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.