Abstract

Screener is a cross-platform media player which enables geographically disperse participants to enjoy media together as if they were all at a "screening party." Screener enables participants to have a synchronized experience such that if one participant pauses the media then the media is paused for all participants; if one participant seeks to a certain time, all participants seek to that time, et cetera. All participants can also communicate through text using the open Extensible Messaging and Presence Protocol (XMPP). Screener allows for the use of "video overlays", PNG images that can overlayed on top of the playing video. Screener allows participants to use local media or any media accessible through a URI.

Screener supports a variety of features. Screener can play a wide variety of formats, it supports N-way synchronization, text chat, and video overlays. Screener utilizes open source libraries and open standards such as the GStreamer media framework, XMPP through the library xmpppy, and the Python programming language.
1 Related Work

There are a variety of web applications, plugins, and desktop applications that have a subset of features similar to those proposed for Screener. Most, however, are closed source and don’t have many features required for the proposed experience or are too restrictive to be generally useful. Major examples of such projects are listed below.

1.1 NBC Viewing Parties

NBC Viewing Parties are a feature of NBC’s online video site. NBC allows private parties to be created in which viewers can watch a video from NBC in sync with one another. Playback actions such as pause and seek are synchronized between clients and users can communicate by text chat. Viewing parties supports many of the features of Screener but it is limited to only videos found in NBC’s library. It is also not apparent if NBC’s system compensates for playback drift or interruption that may occur between parties.

1.2 CBS Viewing Rooms

CBS Viewing Rooms are similar to NBC Viewing Parties but differ in several key aspects. First, CBS Viewing Rooms do not support private parties, instead users join basically a public chat room with a video already in progress. There is no playback control but users view the content in sync with one another. Unlike NBC but similar to Screener, CBS supports specific overlay images that be superimposed on the playing video.

1.3 Yahoo! Zync

Zync is a plugin for Yahoo’s Messenger application which allows friends to synchronize the playback of videos embedded in popular online video services such as YouTube and Google Video. Zync is a closed source application and requires the use of Yahoo’s Messenger service. It is limited to online videos and synchronization between only two participants. It also doesn’t support advanced features such as video overlays. It is currently in beta. Screener will be open source and utilizes open technologies such as GStreamer and XMPP. Screener is also focused on multi-participant environments and supports longer, higher quality media, whereas Zync is limited to streaming online videos.

1.4 YouTube Streams

YouTube Streams is a beta service from Google’s YouTube which allows participants in a chatroom to create shared playlists of YouTube videos. YouTube Streams only offers shared playlists and chatting. There is no synchronization for the videos, once a video starts playing each participant acts independently of any other. Videos are also limited to only those on YouTube.
1.5 ClipSync

ClipSync is a company which purports to be “pioneering the ‘real-time group interaction’ market for entertainment and media consumption, competition and discussion” \(^1\). ClipSync’s software is used by CBS’s Viewing Rooms.

1.6 syncVUE

syncVUE is an application developed by California based Intelligent Gadgets which offers functionality similar to those proposed in Screener. It offers participants in a Skype voice call the ability to watch fully synchronized video along with various graphical markups \(^3\). syncVUE is a closed, non-free solution, however, and is currently not available either for purchase or trial and in fact appears to be defunct. The target audience also appears to be professionals that review produced videos, instead of Screener’s more peer and entertainment oriented audience.

2 Technical Approach

Screener is developed in Python and utilizes open source software and standards throughout its design. Screener utilizes an object oriented design and is thus divided into major components that perform specific functions.

2.1 Multimedia

Media playback functionality is provided by the GStreamer multimedia framework. GStreamer is an open source, cross-platform, multimedia framework implemented as a C library which provides a pipeline architecture for handling media files and streams \(^6\). The GStreamer project provides bindings for the GStreamer library in Python which eases our development.

Utilizing GStreamer provides us with several advantages. First, we do not have to concern ourselves with the intricate details of media playback such as codecs and graphs. If a codec is installed in GStreamer then we will automatically be able to utilize it, without any more effort on our part. GStreamer also allows us to easily manipulate the rendering pipeline to insert our own functionality which is crucial to Screener’s design, especially in the implementation of overlays.

2.2 Network

Network describes all functionality to find participants for a party, to establish and maintain a party, and to provide textual communication between party members. To implement our network component we used the Extensible Messaging and Presence Protocol (XMPP). XMPP is a XML based push messaging protocol which allows users on federated networks (analogous to email in which
XMPP has several advantages from our perspective. XMPP is a mature and established protocol with a rich community of servers, clients, and developers. XMPP is highly extensible so it easily supports the extensions we required to perform synchronization over the protocol. It is also an open protocol with rich documentation and support. Since it uses XML as its data transport the protocol is human readable which greatly aided in testing and debugging.

Several libraries implement XMPP to varying degrees such as libpurple, a C library which forms the core of Pidgin and several other instant messaging clients. There are also several implementations in Python. For our development we chose xmpppy, a Python library which implements the XMPP protocol. We chose xmpppy for the robustness of its API and its extensibility.

2.3 User Interface

Screener uses a command prompt as an interface to the application. Playback controls, library manipulation and message relaying are all done through a command line interface. Other interfaces could be easily added to Screener, however.

2.4 Synchronization

The synchronization component provides the mechanism to synchronize playback and other events such as graphical overlays between the participants in a Screener party. This component is written from scratch for this project in Python.

Synchronization describes the goal of having each participant watching the video experience it as if everyone were sitting around the same computer or television. For example, if one participant pauses their playback then the playback is paused at the same moment for all other participants. Other actions like seeking and video overlays are also synchronized between all participants. The host of the party acts as an arbitrator of synchronization actions; relaying synchronization messages, making sure all participants remain in sync, and preventing participants from performing actions that they are not permitted to perform like seeking the video if that is not a privilege granted to them.

Synchronization forms an ad hoc centralized network, with the host acting as a “server” and all other participants as “clients”. Other topologies were considered but were dismissed as likely to result in scaling issues and undue complexity.

Synchronization performs two major tasks: first, keeping participants’ frame rates in sync so they don’t drift out of sync and second, relaying events from one participant to all others.
2.4.1 Frame Rate

Video is composed of a series of still images (frames) that are displayed in sequence at a fixed rate, thus creating the illusion of motion. The rate at which the frames are displayed is appropriately called the frame rate. The frame rates of participants in a party must be kept in sync, otherwise over time they will drift out of sync and ruin the shared experience. Luckily, this problem has been addressed, in part, by media frameworks that insure that the media is played back as closely to its intended frame rate as possible. In situations where the media cannot be displayed at its full frame rate due to processing constraints, then less frames will be displayed for longer periods of time in a process referred to as “dropping frames.” Thus, in most cases, if participants begin in sync they should remain in sync for the entirety of the playback.

However, situations can arise where participants can become out of sync. There could still be drift if some participants’ frame rates are slightly lower than the mean and some are slightly faster than the mean. Some participants could also experience spikes in processor usage that cause their media processing to be delayed and result in them suddenly jumping out of sync.

To handle this issue, all participants need to be made aware of where all other participants are in their playback. This can be performed by many elaborate schemes, but the most straightforward is to have the host of the party act as an authority on playback. Therefore all other participants synchronize with the host and the host synchronizes with no one. This is accomplished by the host periodically sending out an XMPP message to all participants with its current playback time. The playback time is a simple integer that is transmitted as the value of an XML element inside an XMPP iq (info/query) message:

\[
<\text{iq} \ldots> \ldots <\text{playtime}>36573</\text{playtime}></\text{iq}>
\]

Figure 1: Example playtime message

Participants can then calculate how far ahead or behind of the host they are and throttle down or up their playback speed to match the host. Throttling is important as it prevents the viewer from experiencing unpleasant “jumping” in their video.

Since Screener is intended to be used over the Internet, adverse network conditions could cause messages to be received much later than they were sent, causing the data received to become stale. To resist such effects, participants would have to know exactly when the host sent the messages. This is done by attaching a timestamp to the host’s playtime message:

\[
<\text{iq} \ldots> \ldots <\text{playtime} \text{ time}="1222619053">36573</\text{playtime}></\text{iq}>
\]

Figure 2: Example playtime message with timestamp

In order for the timestamp to be usable, the host and the participants’ clocks must be in sync. This is accomplished through the use of the Network Time
Protocol (NTP). At startup each client communicates with an NTP server to receive an accurate time measurement. This time is time compared with the local system clock to generate an offset. This offset is then used to send and compare message timestamps. With the use of an accurate timestamp, this means even messages that are delayed for extended periods of time are usable.

### 2.4.2 Events

Events describes all actions that occur at a specific time in the playback and must occur in sync with all participants. An event could be a simple playback action such as pausing or it could be a more advanced action such as drawing a video overlay. When a participant generates an event, it is sent as a timestamped message to the host which decides whether or not to relay the message to all other participants. The event message contains the type of event, the frame at which it occurs, and an optional payload describing the event more fully.

```
<iq ...><event action="pause" frame="36573"></event></iq>
```

Figure 3: Example event message

An immediate problem with synchronized events is that if we are doing our job of keeping everyone in sync, then once an event occurs and is sent to every participant, each participant will have already passed the frame on which the event occurred. Since events cannot be sent and processed instantaneously, then this is an unavoidable problem. Therefore events are processed as soon as possible, and not in perfect sync with the frames. However, for some events, such as pause events, each participant uses the opportunity to “backtrack” to the appropriate frame.

### 3 Conclusion

Significant progress has been made in overlays, pipeline control and multi-user chat and synchronization. Our original goal for overlays was to allow users to draw arbitrary shapes on the video window in a similar manner to telestraters found in sports broadcasting. However, no existing, widely supported GStreamer element allows for the drawing of arbitrary shapes. Instead we opted to implement overlays as displays of fixed PNG images. PNG images were chosen because of their support for a transparent alpha layer which allows them to be displayed nicely on top of video. To implement image overlays we dynamically generate new GStreamer elements while the pipeline is running. These elements read an image file from disk, decode it and output it as a video stream, and converts the video stream’s colorspace so that it can be mixed in with the proper video stream. Because of GStreamer’s strong use of threading, implementing such dynamacism properly proved to be difficult. Attaching the new elements to the playing pipeline would cause the pipeline to stall, apparently the result of race conditions that existed between GStreamer
threads. Workarounds were used such as flushing and restarting the pipeline after the addition of new overlays. However, while these workarounds proved successful there are still instances where the pipeline stalls.

In comparison, implementing on screen display of text messages was trivially easy as there is an existing GStreamer element to render text on top of a video stream.

In terms of multi-user chat and synchronization, there were a few options that could be implemented. One option was to create an ad-hoc group of users maintained by the host that are party of the viewing party. The advantage of this would be the flexibility in design, and the lack of need to have a separate room for multi-user chat. The disadvantage would be the inability to use presence messages that are afforded to XMPP chat rooms. The adhoc group would have to keep track of which users are entering and which are leaving and this would significantly add to the overhead. The other option would be to use XMPP chat rooms. However, this option is limited by the number of chat rooms the service provides for the users.

The compromise that was made was to treat the Jabber ID of the host itself as a chat room. Guests wishing to enter the room would ping the host, who would in turn provide the guest an alias to enter. Guests are still afforded presence messages and the system itself is very scalable because additional rooms are not required to be made if the system grows very large.

Therefore, the progress that has been made in these fields represents an open-source, standards-based approach to group video chat rooms. The use of XMPP gives great flexibility and low overhead and the synchronization performs well when all offsets are considered. Gstreamer and the Python wrappers around it give an easily extensible and accessible way to build a video pipeline and in the future, to build a graphical user interface. While some parts of our original proposal had to be neglected, such as a 3D accelerated user interface, based on the Clutter API, we have accomplished all of our core goals. We hope to continue working on this project and once properly hardened to release it openly.

References

    Key: clipsync1
    Annotation: Basic introductory information about clipsync. No actual product or in depth information is given. Copyright notice is recent, but information on website has not changed in months.

    Key: xmpp1
    Annotation: Summary of the XMPP protocol which describes its basic purpose and architecture. Cites published RFC documents which standardize the protocol. Authoritative since published by
the XMPP Standards Foundation, which officially manages the XMPP standard.


**KEY:** syncevue1

**ANNOTATION:** Corporate website expousing the benefits of the synceVUE product. While the website offers some information on the product including a feature list and screenshots, the product is currently not attainable for either purchase or trial. This has been true for several months.


**KEY:** zync1

**ANNOTATION:** Zync was chosen to be developed as a plugin for Yahoo! Messenger since instant messaging (IM) conversations are a natural medium in which the participants are likely to want to view media in sync. A scholarly paper recently submitted to a conference. Authors are employees of Yahoo! working in conjunction with the University of California at Berkeley.


**KEY:** libpurple1

**ANNOTATION:** Brief description of libpurple, a C library which implements several instant messaging (IM) protocols. Published in a wiki for Pidgin development, Pidgin being the name of the project which developed libpurple.


**KEY:** gstreamer1

**ANNOTATION:** Part of the GStreamer Application Development Manual published by the GStreamer project as a document that developers utilizing GStreamer can use to aide in projects using GStreamer. It describes the basic pipeline architecture of GStreamer and its API. Highly authoritative in this regard as it comes directly from the developers of the software.


**KEY:** youtube1

**ANNOTATION:** Summarization of YouTube experimental features including the YouTube Streams service. Part of corporate website designed to entice users to try service.