UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL DE CALCULATOARE

PROIECT DE DIPLOMĂ

Aplicatie mobila de video streaming live
Versiunea 2020

Alina-Alexandra Găzdaru

Coordonator științific:

Sl. Dr. Ing. Vlad-Valentin Posea

BUCUREȘTI
2020

UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

DIPLOMA PROJECT

Live video streaming mobile application
2020 version

Alina-Alexandra Găzdaru

Thesis advisor:

Sl. Dr. Ing. Vlad-Valentin Posea

BUCHAREST
2020

SUMMARY

SINOPSIS

Streaming-ul a devenit una dintre caracteristicile definitorii ale oricărei aplicații de social media, atât prin numărul de utilizatori, cât și prin numărul de urmăritori. Realizarea însă a unui live-streaming folosind mai mult de o sursă video presupune costuri enorme, atât din punct de vedere al aplicațiilor software existente, cât și al componentelor hardware ce trebuie achiziționate, care pot ajunge la costuri enorme. LiveToGo este o aplicație mobilă Android ce își propune să realizeze tocmai acest lucru: multi-camera live-streaming cu dispositive telefonice prin implementarea unui server ce va avea rolul de video switcher și redirecționare a stream-ului curent către utilizator. Singurele resurse necesare sunt un dispozitiv Android și conexiune la Internet.

ABSTRACT

Video streaming has become one of the defining features of any social media application, both in terms of the number of users and the number of followers. However, making a live-streaming using more than one video source involves enormous costs, both in terms of existing software applications and hardware components to be purchased, which can reach enormous costs. LiveToGo is an Android mobile application that aims to achieve this goal: multi-camera live-streaming with telephone devices by implementing a server that will act as a video switcher and redirect the current stream to the user. The only necessary resources are an Android device and an Internet connection.

1 INTRODUCTION

1.1 Context

Live streaming has seen an exponential increase in popularity through its unique way of creating a community. People can participate in their favorite events or interact with their idols, from the comfort of their home. The development of smartphones in terms of video capabilities has made it possible for the general public to distribute content at high quality with a minimum of resources. All one needs is a smartphone, an Internet connection, and an exciting topic. According to a survey conducted by Vimeo in 2017[1], more and more viewers prefer to watch a live stream in favor of a pre-recorded video, or even live TV. Thus, it has become mandatory for each event to be streamed live on the Internet, and even news programs broadcast their hourly updates through online platforms such as YouTube, Facebook.

1.2 Problem

The disadvantage of the platforms that have incorporated the live-streaming functionality is that they do not allow broadcasting with several video sources and switching between them as the organizer likes. There are software products on the market that have this option integrated, but they have very high costs. There are also hardware devices in the form of video switchers that allow the connection of several video sources, but their price can easily exceed 2000 RON, plus organizers must provide additional components to make a live-streaming. Consequently, for low-budget events, multi-camera live-streaming is not a priority. Several mobile applications have been developed that allow the use of multiple phones in creating a multi-camera live stream, but they are exclusive for the iOS operating system.

1.3 Solution

With this in mind, we aim to develop a mobile application for Android, which regularly communicates with a server that serves as a video switcher. A video switcher is a device that merges two or more video inputs into a single output feed. All video sources connect to a switcher, and then the user selects which one will appear on the screen. The stream coordinator has an interface, in the form of a web or mobile application, which gives him access to all video sources and options to switch between them. The plus that this application brings

[1] https://livestream.com/blog/live-video-statistics-livestream

is the lack of additional hardware components and the low cost compared to existing products. Our application is a solution brought to those who want to ensure a live-stream at a medium to good quality, with a minimum of necessary resources, physical and financial.

1.4 Thesis roadmap

The first chapter introduces the problem and a brief description of the solution presented in the paper.

The second chapter contains an analysis based on a personal survey about basic functionality and user preferences for this type of mobile application. It also presents the clients that may be interested in our solution.

The third chapter describes the leading technologies used in developing the application. The architecture includes components accepted by the industry, such as Android, RTP/RTSP, H.264, AAC.

The fourth chapter explains how we integrated the technologies presented in the third chapter to accomplish the live video stream. It contains the communication between Android clients and the servers (Wawza Streaming Engine and switcher server).

The fifth chapter presents an experimental evaluation of the system, problems occurred along the way, and whether they are still persistent.

The sixth chapter describes the functionality of similar products already on the market, their advantages, disadvantages, and pricing plans.

Finally, the seventh chapter includes the main conclusions regarding this project and our plans for the future.

2 REQUIREMENTS ANALYSIS

Live streaming is a newly adopted method for self-promoting among social, cultural or political events, among corporations, or individual people. In an article written for the Vimeo website, researcher Caroline Golum describes how corporations can use live streaming to "grow and adapt"[1], with applications in internal meetings, cross-office communications, and external product launches and events. Streaming is an effective way to keep employees active and engaged, and according to the Harvard Business Review, "a happy and engaged workforce [...] raised sales by 37%, productivity by 31%, and accuracy on tasks by 19%."[2] There are many articles through which people who want to create an online brand can learn how to take advantage of the benefits of live streaming, and how to deliver content according to the age groups of viewers. The figure below shows the time spent online by every age group (August 2019).

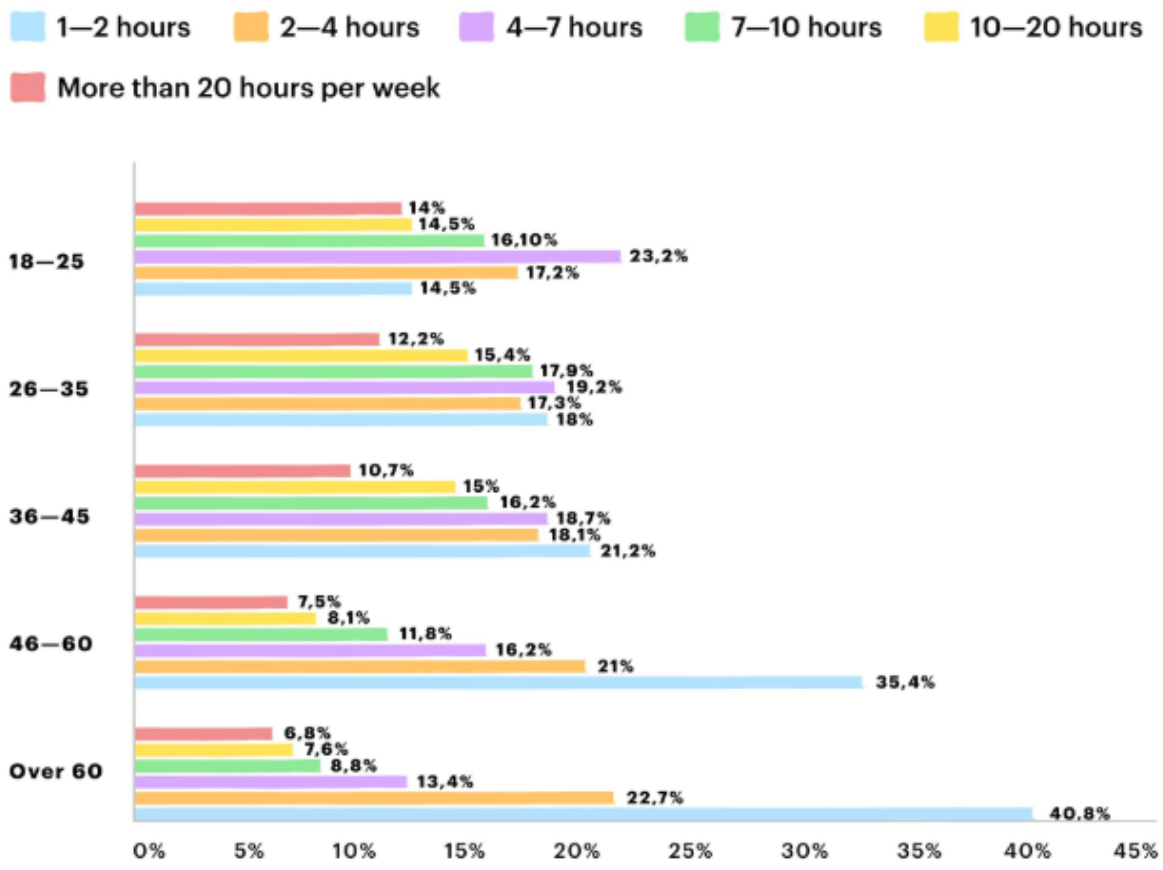


Figure 2.0.1: Weekly time spent with online video worldwide as of August 2019, by age group[3]

[1] https://livestream.com/blog/corporations-using-live-video
[2] https://hbr.org/2011/06/the-happiness-dividend

Given these global statistics, we conducted a survey[4] in order to determine to what extent a live-streaming software product would be used, and what would be the main features that a potential customer is looking for. The software in question is a mobile application for the Android platform, which has both streaming and viewing capabilities, and in addition, allows multi-camera streaming, with an integrated video switcher.

To the question "How likely are you to use a live streaming application during a social or corporate event?" the predominant answer was positive, at 72.6%. In terms of multi-camera functionality, many were neutral (34.2%), with a total neutral to an affirmative response rate of 69.8%. Put in the situation to choose a live-streaming product, 90.4% of the participants chose as the main criterion the video and audio quality, followed by security and cost. When it comes to using such an application, 83.6% said they prefer a user-friendly, easy-to-use interface. Other preferred features are cross-platform integration, the lack of any form of advertising, and multi-bitrate transmission. The percentage of results obtained from this survey can be found in Annex A.

Given the large number of people who have access to the Internet and at least one smartphone, a potential customer of the application we propose could be anyone who wants to make a live broadcast with few resources, deliver content from multiple perspectives, and have control over what is broadcast.

[3]https://restream.io/blog/best-live-streaming-formats-for-business/
[4]https://forms.gle/gzXi9hYGF79ejw4v5

3 RELATED WORK

Live streaming is an online service of video and/or audio recording and broadcasting in real-time that covers a variety of topics, including social media, video games, and even professional sports events. Starting with the first video live streamed in 1993, this service has become the focus of many stand-alone web and mobile applications along the years, and a key feature in social media platforms like Facebook and YouTube.

Some of the significant advantages of live streaming are real-time engagement with the users through real-time chat rooms and comment sections, the costs that can fluctuate from very low to non-existing, considering that users do not need any special software or hardware equipment, and last but not least, very high reachability, users being able to attend various events from the comfort of their homes.

The popularity of live streaming is considerably higher, with an increase of 47% of viewers worldwide only in the last year and 63% of the population aged 18-34 who watch live streamed content. It is expected that by the end of 2020, live streaming will represent 82% of Internet traffic, with the potential of becoming a $124 billion industry by 2025 [12].

3.1 Comercial solutions

Vimeo

Vimeo [10] is one of the most known video hosting services, founded in 2004 by Jakob Lodwick and Zach Klein, with a number of 1272 of thousands of subscribers worldwide in the 1st quarter 2020 [9]. Vimeo operates on an ad-free basis and provided four subscription plans for features, including video creation and video player, collaboration, distribution and marketing, analytics, hosting and management, across a range of devices, permitting video professionals to connect with clients and other professionals.

One subsidiary of Vimeo is Vimeo Livestream; a feature included only in the Premium Plan at the price of $75/mo. It contains services like unlimited events and viewers, video quality up to 1080p, video switcher, and streaming to multiple destinations and live QA, polls, and graphics [10].

Vimeo is compatible with most smart devices from TVs and PCs to mobile/tablets with both Android and iOS operating systems. In order to go live, a user needs a connection to a camera and any RTMP-enabled encoder, which sends the live recording to Vimeo.

ManyCam

ManyCam [6] is a mobile application for Android and iOS that offers live streaming services consisting of mobile RTMP streaming to platforms like Facebook Live, YouTube Live or any other live streaming platform, multiple video sources (mobile camera, videos, images, IP cameras) and streams with picture-in-picture window (smaller stream on top of the screen size stream). ManyCam offers three pricing plans, ranging between $30 and $100. The Premium Plan features consist of up to 24 video sources, 4K video quality, unlimited RTMP streams, H.264 IP camera connection. [6]

vMix

vMix [11] is a software product available for the Windows operating system that allows users to switch inputs and mix audio, and live stream multiple cameras in resolutions up to 4K. Its streaming partners include Facebook Live, Twitch, Vimeo, YouTube Live, WOWZA Media Systems, DaCast. For minimum quality and features, vMix offers a free purchase plan, the PRO plan (highest), reaching a cost of $1200. This comprises of up to 1000 camera and NDI inputs, 4K maximum resolution and broadcast to multiple sources simultaneously through built-in FFMPEG encoder [11].

mimoLive and LIVE:AIR Action

mimoLive [7] and LIVE:AIR Action [5] are two live streaming solutions for iOS-based mobile devices with a multi-camera feature accompanied by custom graphics, transitions, animations. Other features consist of picture-in-picture functionality, fully-featured audio mixer for each video source, custom RTSP/RTMP ingest, NDI input and output support, green screens.

4 ARCHITECTURE

4.1 Android Platform

Android is an operating system developed by Google, based on a modified version of the Linux kernel. Developers use it, especially for touchscreen mobile devices like smartphones and tablets, covering 76.2% of the market share worldwide (Figure 3.1.1). With the launch of the first commercial Android device in September 2008, the OS has had several major version updates, the most recent one being Android 10, released on June 1st, 2020. Android mobile applications can be downloaded from Google Play, hosting a total of 2.8 billion applications (March 2020) [8].

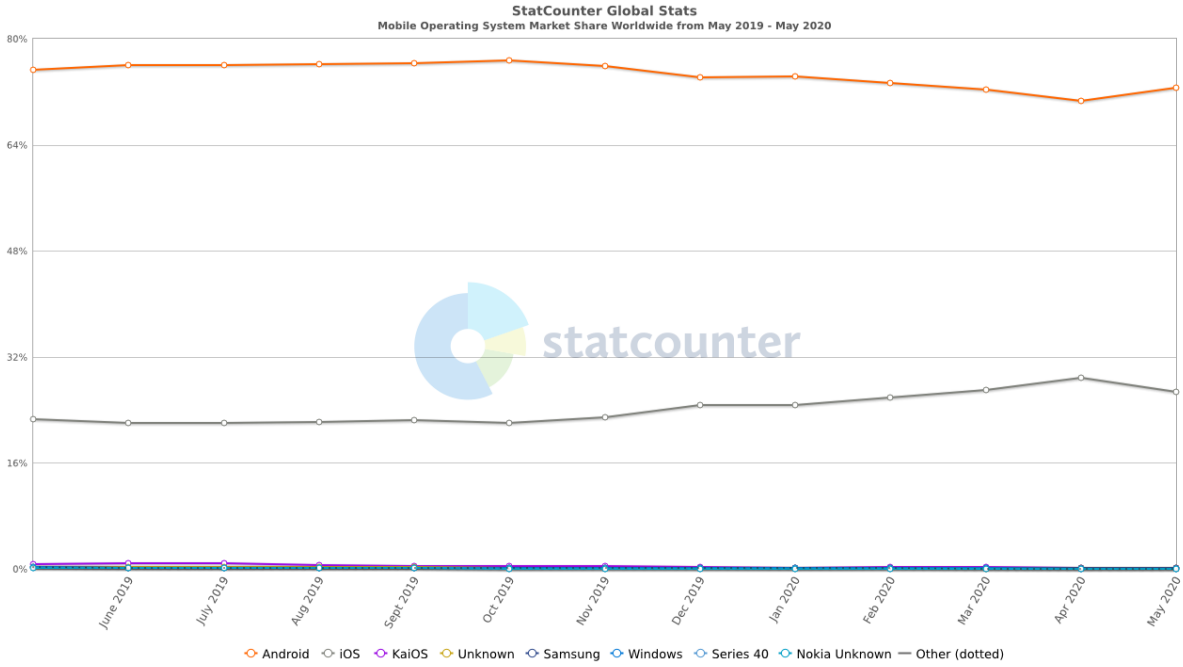


Figure 4.1.1: Mobile opeating systems usage[1]

Android OS is a stack of software components divided into five sections and four main layers, as shown in figure 3.1.2.

- Linux kernel – offers an abstracting level between the device's hardware and contains all the essential hardware drivers (camera, keyboard, display)
- Libraries – a collection of C/C++ libraries representing the foundation of Android components functionality (WebKit with the open-source web browser, libc, SQLite database,

[1]source: https://gs.statcounter.com/os-market-share/mobile/worldwide

audio and video recording libraries, SSL libraries for internet security)

- Android Runtime – Dalvik Virtual Machine – a special Java virtual machine developed and optimized for Android. It uses basic Linux functions like memory management and multi-threading
- Application Framework – has many high-level services in the form of Java classes, the key services being Activity Manager, Content Providers, Resource Manager, Notifications Manager, View System
- Applications – Android comes with a set of basic applications for e-mail, SMS messages, calendars, internet navigation.

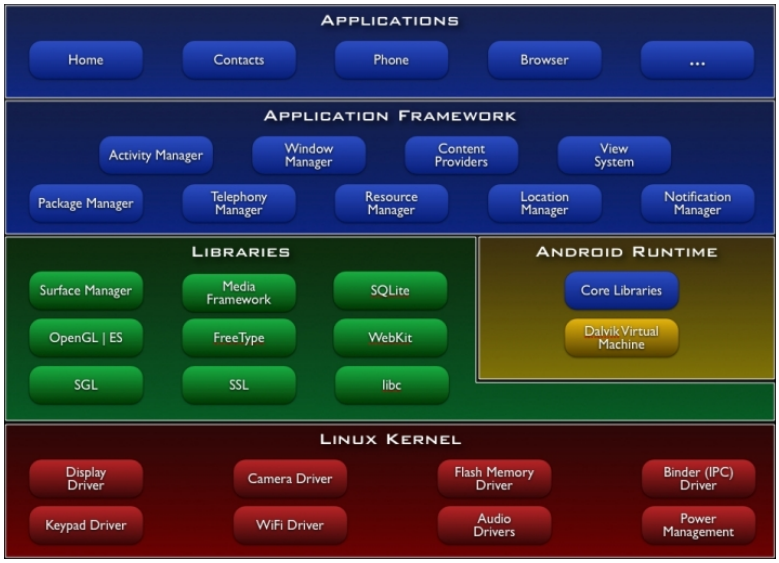


Figure 4.1.2: Android system architecture[2]

The main programming language for Android applications is Java. The SDK instruments compile the code with any given resource/data file in an APK source file (.apk Android packet). An APK file contains the entire application and represents the runnable file used for installation. There are four different Android application components:

- Activities – an activity is the entry point in user interaction. Activity class includes six main methods present in a life cycle: onCreate(), onStrat(), onResume(), onDestroy(), onPause(), onStop()
- Services – general entry points for maintaining a running application in the background to perform long-term operations or long-distance processes
- Broadcast Receivers - a component that allows the system to transmit in-app events outside of the usual user flow, allowing the app to respond to system-wide broadcast announcements
- Content Providers – an account provider manages a standard set of applications data that could be stored in the file system, in an SQLite database, on the web, or any other persistent storage the applications can access.

[2]source: https://developer.android.com/images/system-architecture.jpg

Developers can activate three out of the four components (activities, services, and broadcast receivers) by an asynchronous message called **intention**. Intentions link individual components at run-time and have three fundamental use cases: activity start, service start, broadcast send.

Every application must have a special file called **AndroidManifest.xml** in the root directory. The manifest file offers essential information about the application to the system, information needed to execute any source code. Android Studio utilizes a set of advanced instruments called **Gradle** for source code compilation and application packaging, by automating and managing the build process, while allowing custom flexible configurations.

4.2 Real-time transport protocol

Real-time transport protocol, or for short RTP, is a protocol for real-time media (audio/video) transmission, encoded with various encoder types over an IP network. It was introduced in 1996 as RFC 1889 by the Audio-Video Transport Group of the Internet Engineering Task Force (IETF), which was superseded by RFC 3550 [3] in 2003. Various hardware and software applications support RTP, such as Windows Media Player, VLC, mplayer, and ffmpeg. At transport level, this protocol is transmitted over UDP (User Datagram Protocol) because it has less overhead over headers, being able to carry more data in a single packet and to efficiently utilize the network bandwidth, while simultaneously taking advantage of the lack of TCP (Transmission Control Protocol) acknowledgments or TCP ACK packets. In case of low cellular coverage, hence lower bandwidth, the video stream of a user would be choppy and at a lower quality, but still transmitting.

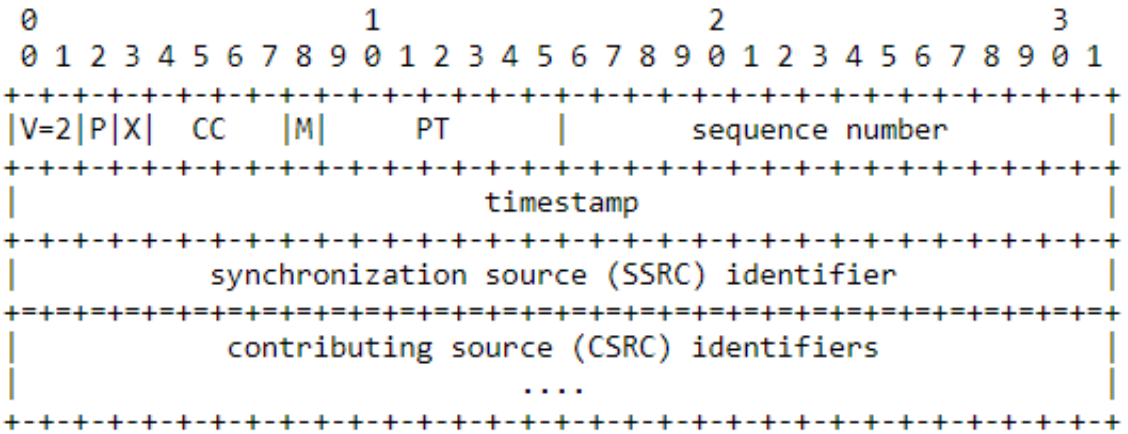


Figure 4.2.1: RTP header [3]

The operational flow of the RTP is shown in figure 3.2.2; for a video, for example, to be streamed, it must go through several steps: encoding, packetizing, transport control, re-assembly, and decoding. RTP works alongside the Real-time Control Protocol (RTCP), which makes sure of Quality of Service (QoS).

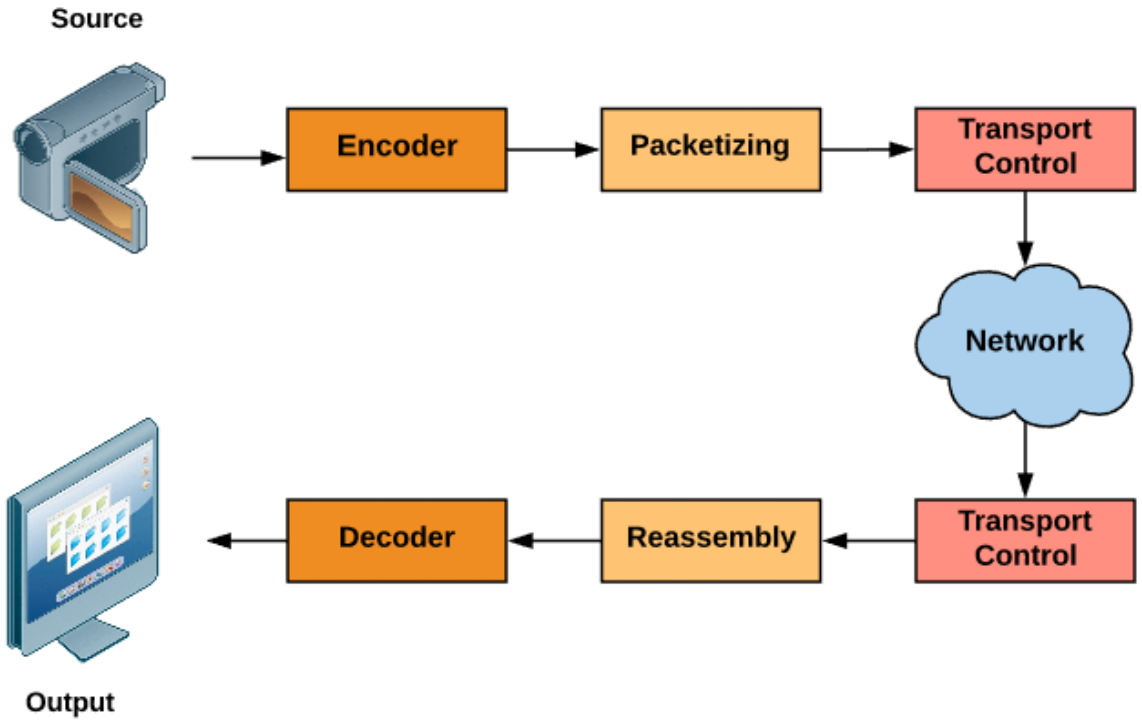


Figure 4.2.2: RTP packet flow[3]

4.3 Real-time streaming protocol

Real-time streaming protocol, commonly known as RTSP, is a widely used protocol for efficient establishment and control of media sessions (video and audio data delivery with real-time properties) between client and server over the internet. It was published in 1998 in RFC2326 [4] and is currently developed by Multiparty Multimedia Session Control Working Group. The transmission of data itself is not a task of the RTSP protocol. Therefore, RTSP uses a combination of protocols: UDP (connectionless protocol), TCP (connection-based protocol), and RTP to achieve numerous functions by maintaining session/state between server and client through an identifier. Because a viewer cannot open an RTP stream directly, an RTSP server needs to offer additional information about the stream and to have control over packet transmission. In order to achieve this, RTSP incorporates the following commands: options, describe, announce, setup, play, pause, and teardown. In the figure below is an example of RTSP protocol in action with the video and audio data being delivered over a separate IDP-based RTP stream.

[3]https://www.oreilly.com/library/view/advanced-infrastructure-penetration/9781788624480/5ce761e5-1024-4556-a0b0-0864a1856de1.xhtml

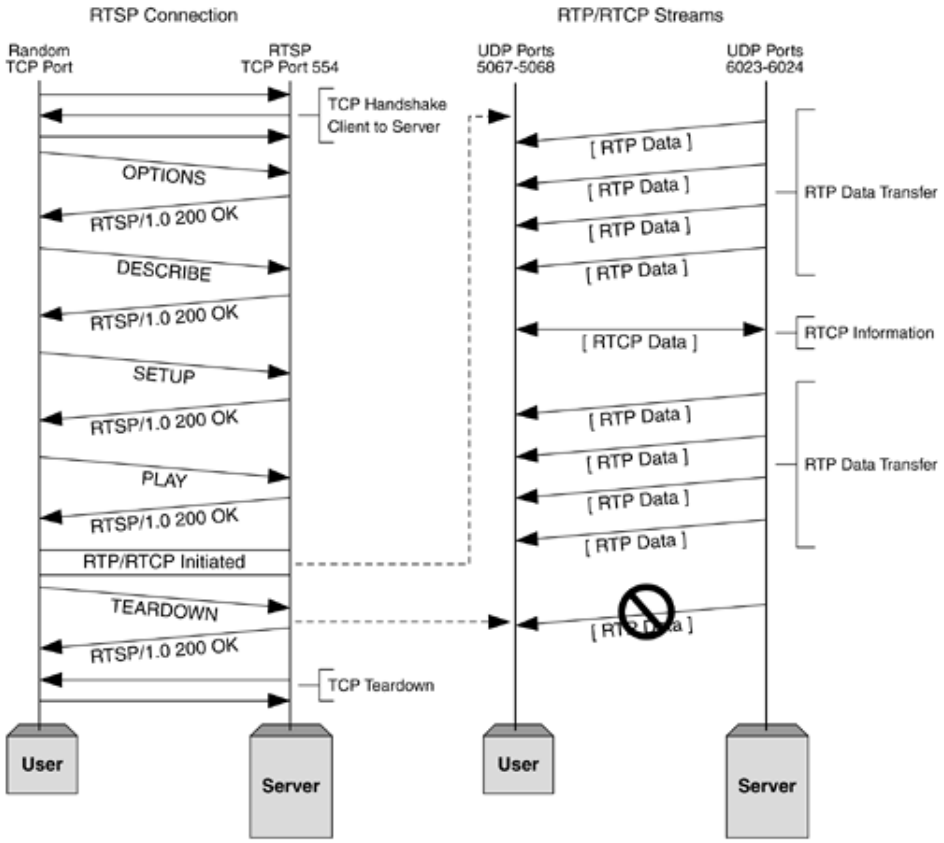


Figure 4.3.1: RTSP transmission flow[4]

4.4 Real-time messaging protocol

Real-time messaging protocol, or RTMP, was the primary protocol used to transport video over the Internet with the advent of streaming. It has its base on the TCP protocol and has been implemented to maintain persistent and low latency connections. Initially developed by Macromedia, it became an Adobe product (after the acquisition of Macromed by Adobe), which states that "Adobe's Real Time Messaging Protocol (RTMP) provides a two-way multiplex service over a reliable stream transport, such as TCP [RFC0793]. intended to carry parallel streams of video, audio, and data messages, with associated timing information, between a pair of communicating peers." [1]

RTMP has support for audio and video codecs such as AAC, AAC-LC, MP3, respectively H.264, VP8, Soreson Spark. The main disadvantage of this protocol is playback compatibility only with Flash Player, Adobe AIR, and any other RTMP compatible player. Despite the benefits it brings in terms of latency and minimal buffering, it is no longer supported by iOS, Android, and most browsers.

RTMP behaves like a tunnel between the client player and the server to transmit rapid video

[4]https://www.informit.com/articles/article.aspx?p=169578seqNum=3

data by maintaining a constant connection. Because it is developed over TCP, it uses a three-way handshake to transport data: the client asks the server to start a connection, the server accepts, the client acknowledges the response, and the session starts.

Although Adobe Flash is no longer a product integrated into many software applications, numerous content creators still use RTMP encoders, as indicated in a report made by Wawza, in which 33.13% of users chose RTMP as the streaming format currently used.

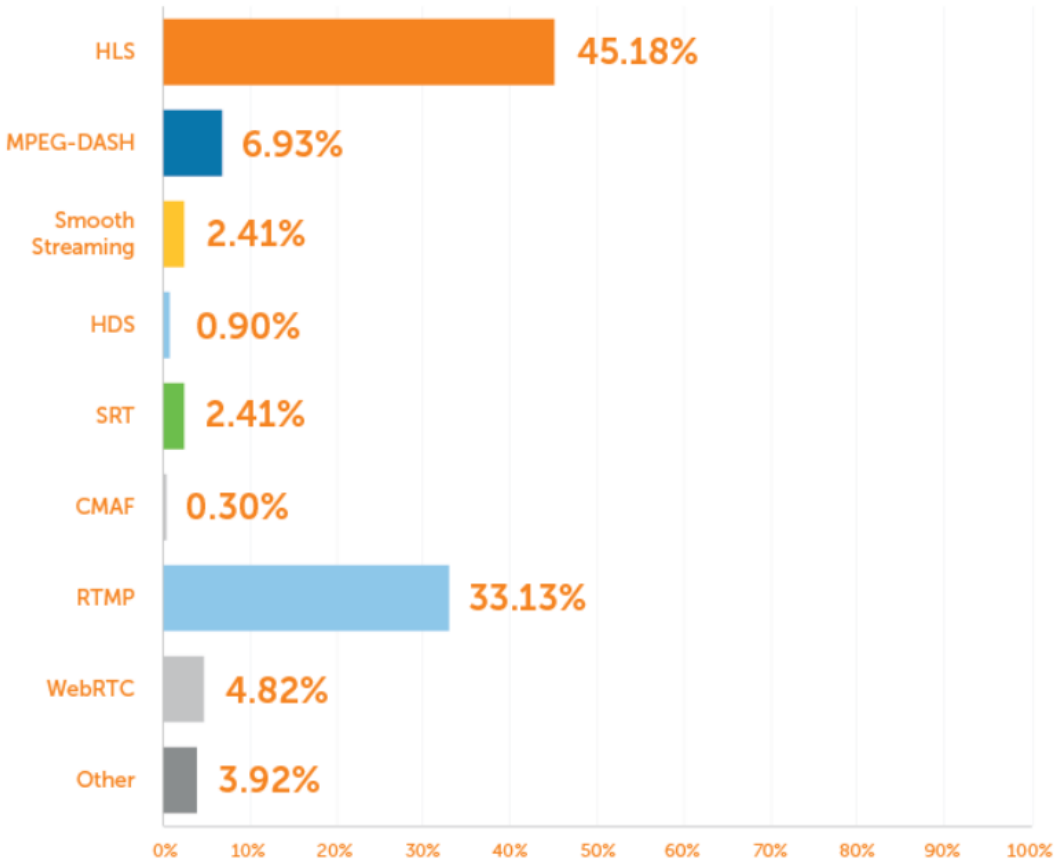


Figure 4.4.1: Streaming encoders usage - Wawza report[5]

4.5 H.264

H.264 is a Video Compression Standard specified in Part 10 of MPEG-4, and is the most widely used codec for compression, recording and distributing HD videos. It has a high compression rate as it can reduce the size of a digital video file by more than 80% compared to the JPEG format and was developed to improve video quality while transmitting them at a lower bitrate than its predecessors, MPEG-2 H.262 and H.263. With these improvements, H.264 is expected to become the leading standard in video compression.

H.264 is used in the development of many applications that involve video data transmission

[5]https://www.wowza.com/blog/2019-video-streaming-latency-report

due to the transmission efficiency that is double the MPEG-2 standard, the ability to store long videos at high quality, and to work with a wide range of networks and systems.

H.264 includes seven sets of capabilities, called profiles, the best-known being baseline, high and extended. The **baseline profile** is mainly used by standard services such as videoconferencing and mobile applications. The **high profile** is the main profile for disk transmission and storage, especially for high definition television applications. Lastly, the **extended profile** is used for video streaming purposes.

4.6 Advanced Audio Coding (AAC)

Advanced Audio Coding is a technique for compressing and encoding digital audio files, meant to become the logical successor of MP3 that offers better sound quality at the same bit rate. AAC can be used in applications that involve the storage and transmission of mono, stereo, or multi-channel audio signals, where the quality of the decoded sound must be high.

AAC relies on three strategies to achieve maximum optimization. First, it uses a high-resolution transform to eliminate redundancy, utilizing the statistical properties of sound. Second, it uses a signal-adaptive model of the human auditory system to establish a threshold for human sound perception, thus eliminating any information that is not perceived. Moreover, entropy coding is used to match the entropy of quantized values with their bitstream representations' entropy.

AAC can identify frequencies between 8Hz and 96 kHz and up to 48 full-bandwidth audio channels and 16 channels with low-frequency effects. This is an international standard used by companies such as Sony Corp., Nokia Corp., Dolby Laboratories Inc., but also as the default audio codec for the .m4v format.

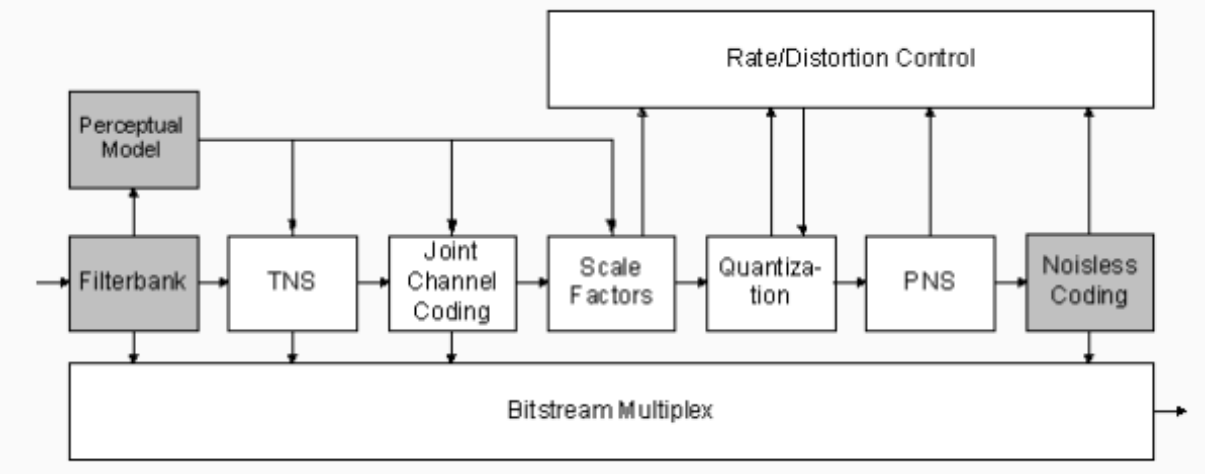


Figure 4.6.1: Block diagram of the AAC encoder[6]

[6]https://mpeg.chiariglione.org/standards/mpeg-2/advanced-audio-coding

5 IMPLEMENTATION

The solution has two main components: Android mobile application and server. The application serves both as video source and client, as seen in the diagram below.

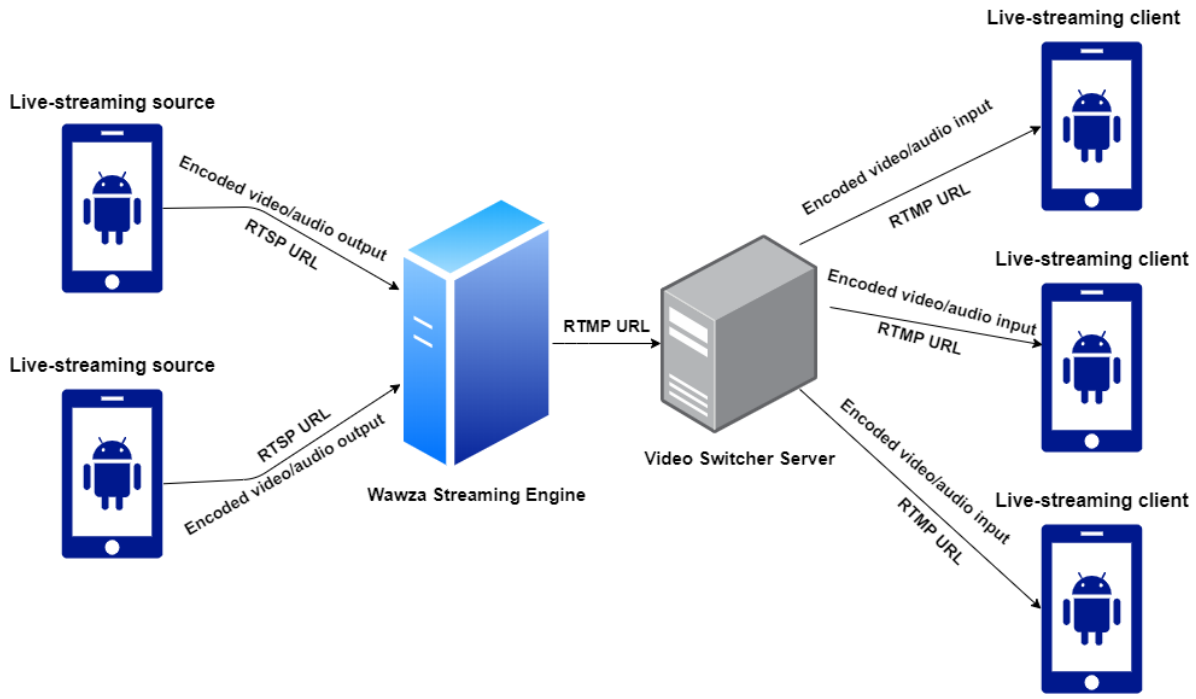


Figure 5.0.1: General functionality

5.1 Android mobile application

5.1.1 General application flow

The general functionality of the application is simple; the interface is user friendly and very intuitive.

As a start, if the user accesses the application for the first time since the installation, he will be asked to create a new account or authenticate with existing credentials. To not have to login each time, the user's credentials will be stored locally through an AccountManager. This class provides a secure and centralized method of access to a user's online accounts. The user can enter credentials only once per account, giving the application access to online resources with a single click.

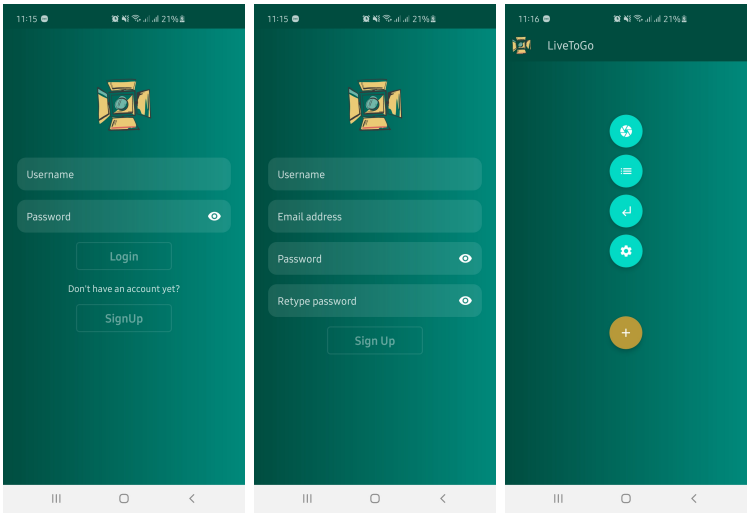


Figure 5.1.1: Login (a), Sign Up (b), Main Menu (c)

Once in the main menu, the user has several options available. First, he can log out of the application and return to the login prompt, or access the application settings page, where changes such as changing the username are possible. Finally, the user can list all the live-streamed events, start any of them, or create his event, becoming the one that broadcasts live video and audio content.

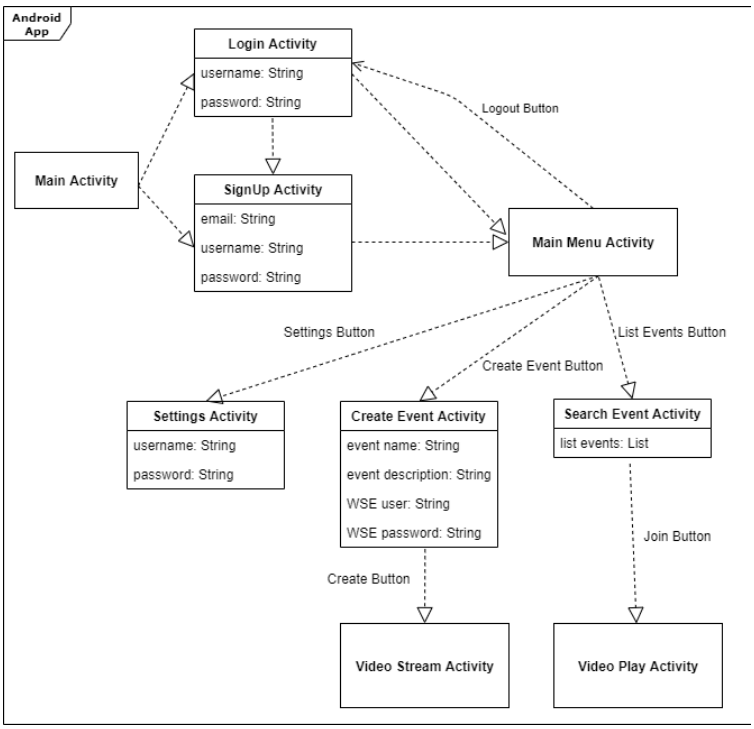


Figure 5.1.2: Application UML Diagram

5.1.2 Video Streaming

A first step in distributing media over a network is to choose the method the data is transmitted, and more explicitly, the transport protocol. Thus, we initially tried direct communication between the Android application and a local server via sockets over UDP (User Datagram Protocol). All we needed for the socket connection was the server's IP address and an available port number, and the choice of UDP protocol was due to the lack of acknowledgment when receiving a packet. In live video transmission, the most crucial aspect is that it is continuous, even if the video has missing pieces. To view the packages (which consist of video frames), we used the OpenCV library available in Python. The most significant problem with this method was the latency and bandwidth, which were very high.

This situation led us to the RTP protocol (real-time transport protocol) and its streaming variant, RSTP. The RTP protocol was designed precisely for this purpose, namely media data distribution over the internet. Its advantage is data compression, which reduces latency and saving bandwidth.

To implement an RTSP client (the Android device that streams), we used the API provided by the GitHub user fyhertz [2], namely libstreaming, which contains a set of methods for streaming the camera and microphone of a device using RTP over UDP. This API was specially designed for communication with the streaming engine developed by Wowza.

To access the device peripherals, it was necessary to define the appropriate permissions in the AndroidManifest.xml file, which defines the main aspects of the application. These permissions consisted of access to the camera, for video recording, access to the microphone, for audio recording, and additionally, permission to access the Internet and WiFi state, for IP / port connection and data transmission over the network. Starting with Android version 6.0, the user is no longer explicitly notified what permissions the application needs at the time of installation, so we required permission to access the device's peripherals.

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
```

Listing 5.1: Permissions list

```
if(ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA)
    != PackageManager.PERMISSION_GRANTED)
     ActivityCompat.requestPermissions(this, new String[]{Manifest.
    permission.CAMERA}, requestCode);
if(ContextCompat.checkSelfPermission(this, Manifest.permission.
    RECORD_AUDIO) != PackageManager.PERMISSION_GRANTED)
     ActivityCompat.requestPermissions(this, new String[]{Manifest.
    permission.RECORD_AUDIO}, requestCode);
```

Listing 5.2: Permission requests

This library provides several video and audio encoding methods: H.264 and H.263 for video encoding; AAC and AMRN for audio encoding. We chose to use H.264 and AAC due to maintaining high-quality videos.

First, we instantiate a session (Session class), which establishes the audio and video encoding methods, which camera will be used (back or front), video, and audio quality. The session represents a message transmitted between the client and the server through SCP (Session Control Protocol) over TCP protocol, through which the client informs the server about the stream format to broadcast later.

```
mSession = SessionBuilder.getInstance()
    .setContext(getApplicationContext())
    .setAudioEncoder(SessionBuilder.AUDIO_AAC)
    .setAudioQuality(new AudioQuality(8000,16000))
    .setVideoEncoder(SessionBuilder.VIDEO_H264)
    .setSurfaceView(mSurfaceView)
    .setPreviewOrientation(0)
    .setCallback(this)
    .build();
```

Listing 5.3: Session instantiation

Second, we instantiated a clientRtsp object, a class provided by the libstreaming API. When creating an event, the user is required to enter a URL of the form rtsp://IP-server:1935/eventName /streamName, a valid username, and password that will be used to connect to the Wowza server. This clientRtsp object makes the connection between the device and the server, having defined four types of requests: ANNOUNCE (announces that a connection is being tried), RECORD (starts the transmission), TEARDOWN (interrupts the transmission), OPTIONS. The VideoStream window allows the user to select the preferred video format from a list of formats available to the device, which will also be passed to the clientRtsp object.

The code snippet below presents the RRSP client instantiation:

```
protected void onCreate(Bundle savedInstanceState) {
  [...]

  mClient = new RtspClient();
  mClient.setSession(mSession);
  mClient.setCallback(this);

  [...]
}
```

Listing 5.4: RTSP client instantiation

The method toggleStream() parses the streaming URL, extracting the server IP address, port number, and full path of the stream sets the credentials for the RTSP client and starts the stream.

```
public void toggleStream() {
```

```
2       mProgressBar.setVisibility(View.VISIBLE);
3       if (!mClient.isStreaming()) {
4           String ip,port,path;
5           SharedPreferences mPrefs = PreferenceManager.
    getDefaultSharedPreferences(TestStream.this);
6           Editor editor = mPrefs.edit();
7
8           editor.putString("uri", mEditTextURI.getText().toString());
9           editor.putString("password", mEditTextPassword.getText().
    toString());
10          editor.putString("username", mEditTextUsername.getText().
    toString());
11          editor.commit();
12
13          Pattern uri = Pattern.compile("rtsp://(.+):(\\d*)/(.+)");
14          Matcher m = uri.matcher(mEditTextURI.getText()); m.find();
15          ip = m.group(1);
16          port = m.group(2);
17          path = m.group(3);
18
19          mClient.setCredentials(mEditTextUsername.getText().toString(),
    mEditTextPassword.getText().toString());
20          mClient.setServerAddress(ip, Integer.parseInt(port));
21          mClient.setStreamPath("/"+path);
22          mClient.startStream();
23
24      } else {
25          mClient.stopStream();
26      }
27  }
```

Listing 5.5: Start stream method

The first snapshot (snapshot a) shows the camera preview, where the user can select video quality at which the live-stream will run (snapshot b). Snapshot c shows the started stream. The down left corner of the screen displays the amount of data transmitted in kbps.

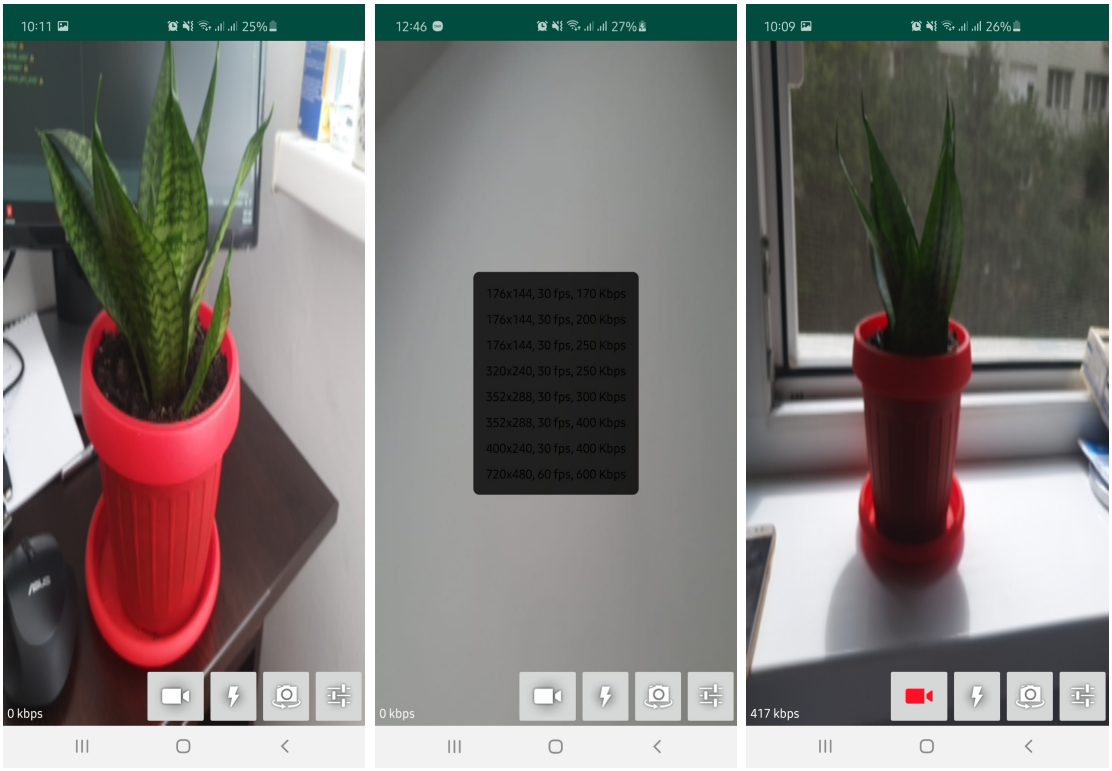


Figure 5.1.3: Preview (a), Resolutions list (b), Live-streaming (c)

5.1.3 Video Play

For a client to view a live-stream, it was necessary to incorporate a media player. Since Android does not support RTMP streams, the solution was to use a player API that offers this functionality. Google developed an alternative to Android's MediaPlayer API that can play audio and video locally or online, called ExoPlayer. ExoPlayer[1] is an application-level media player for Android, easy to customize, with support for RTMP playbacks, H.264, and ACC codecs. The source code for this player is opensource on Google's GitHub page[2].

ExoPlayer's integration with Android was quite simple. We had to add the necessary dependencies in the Gradle file of the application, and in a new activity, to instantiate a player and a media source that we instantiate with the stream URL. When a user wants to connect to an event stream, the application requests the video switcher server for the streaming link with the prefix "rtmp://". The next code snippet shows the instantiation of a media player.

```
protected void onCreate(Bundle savedInstanceState){
  [...]

    BandwidthMeter bandwidthMeter = new DefaultBandwidthMeter();
    TrackSelection.Factory videoTrackSelectionFactory = new
   AdaptiveTrackSelection.Factory(bandwidthMeter);
```

[1]https://exoplayer.dev/
[2]https://github.com/google/ExoPlayer

```
6       TrackSelector trackSelector = new DefaultTrackSelector(
        videoTrackSelectionFactory);
7       SimpleExoPlayer player = ExoPlayerFactory.newSimpleInstance(this,
        trackSelector);
8
9       PlayerView playerView = findViewById(R.id.simple_player);
10    playerView.setPlayer(player);
11
12      RtmpDataSourceFactory rtmpDataSourceFactory = new
        RtmpDataSourceFactory();
13
14      MediaSource videoSource = new ExtractorMediaSource
15              .Factory(rtmpDataSourceFactory)
16              .createMediaSource(Uri.parse(rtmpUrl));
17
18      player.prepare(videoSource);
19      player.setPlayWhenReady(true);
20
21    [...]
22  }
```

Listing 5.6: RTMP video player

The snapshots below show playback of the streaming video using the integrated video player.

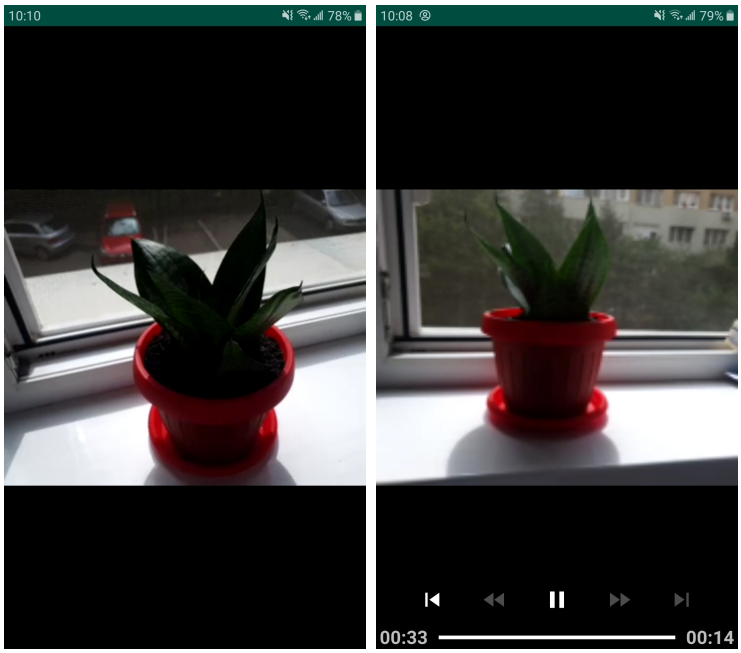


Figure 5.1.4: Live-stream playback

5.2 Server

Wowza Streaming Engine is a streaming media server software product developed by Wowza Media Systems, used for live streaming and on-demand video over IP networks to desktops, tablets, mobile phones. It is compatible with most operating systems and provides a variety of transmission options. The most important feature is that it can interpret RTP packets and provide statistics related to bandwidth consumption, number of connections, and memory. Its primary disadvantage is that the cost that can reach $125/month.

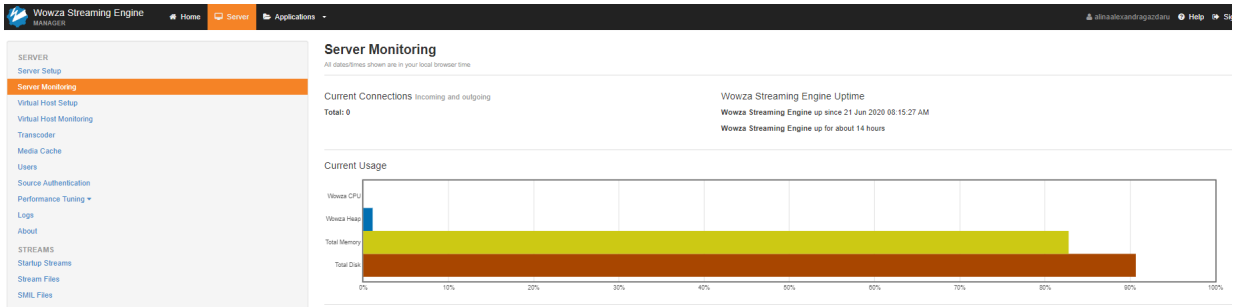


Figure 5.2.1: WSE server monitoring tab

Wowza Streaming Engine is installed locally, it receives the IP address of the device on which it runs, and other devices can connect to it via the Internet, on port number 1935, the RTP port, through a URL. It works based on applications: a user creates an application, whose name must be added to the login URL, and a stream name of user choice. In turn, Wawza Streaming Engine creates a stream access link, in the form of rtmp://IP-server:1935/appName/streamName.

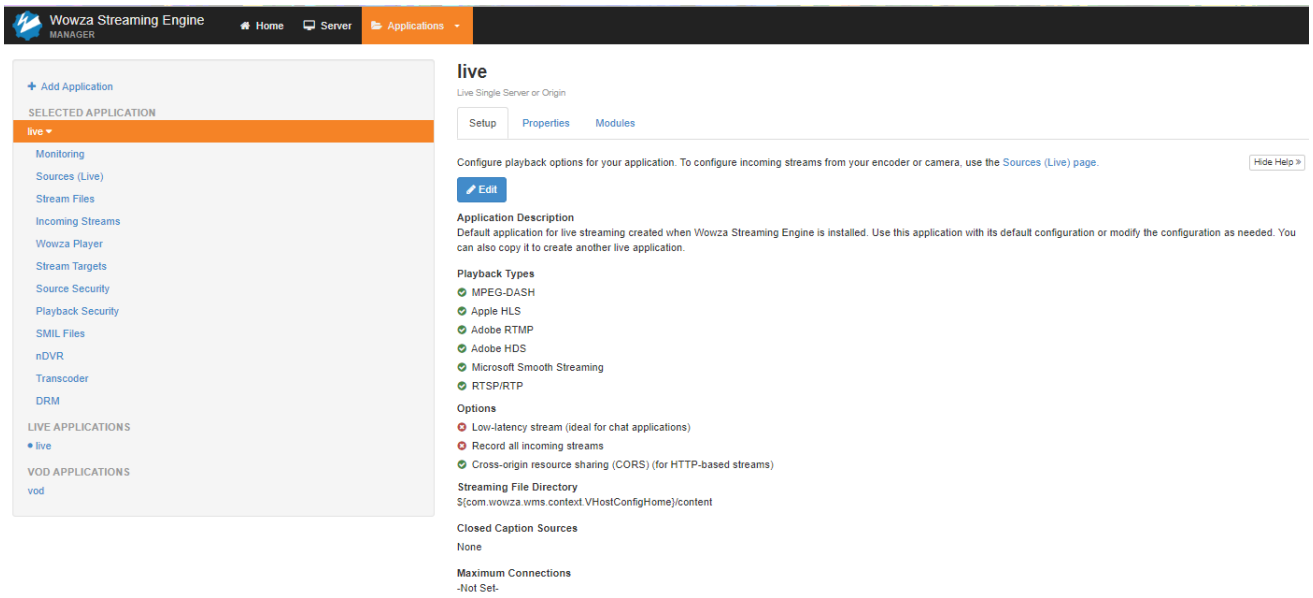


Figure 5.2.2: Application example WSE

Since the Wowza Streaming Engine does not have a video switcher option, we had to implement a separate server that captures all the streaming links of a specific application and

further, it chooses which will be sent to the client to be broadcast. This is a multi-threaded server, and the communication with the clients is done through sockets, as the transmitted data are small. We chose to implement this server in Python due to its speed and auxiliary libraries.

There are two types of clients: source client and destination client. For each client, the switcher server waits for a connection and creates a new thread.

The source client is the one that starts a live stream. At the press of the "start stream" button, a new socket connection is created with the switcher server on port 8000. This connection is brief because the client sends only three messages. First, it informs the server that it is a source client, then sends the streaming event name, and finally, the RTMP access link. The server keeps a dictionary with (key, value) pairs in the form of (event name, list of streaming links).

The destination client is the one that requests access to a streaming event. When selecting an event, a new socket connection to the switcher server is made on port 8000, where the client informs the server that it is a destination client, and sends the name of the event it wants access to. The switcher server maintains the connection alive and sends back to the destination client the RTMP access link selected by the coordinator. We fixed a maximum number of streaming sources per event to four.

6 EVALUATION

In this chapter, we will present the experimental results of the project in its current version and what future implementation plans we propose. The implementation and testing of the functionalities were performed on the Google Pixel 3a emulator from Android Studio, and on Samsung smartphones that run Android 10.

We started by creating a design as simple and pleasant as possible in order to create a better experience. The login and sign-up pages have the classic two- and four-field design, respectively, and do not require much personal information other than a valid email address. In the main menu, we kept a minimum of functionalities, namely logout from the application, a settings button, and the two main functionalities: starting a live video stream and connecting to a stream. At this point, the settings page offers the possibility to change the username and password. In the camera preview prompt, the user can use the "switch camera" button to choose the front or back camera and set the video transmission quality from an available list (resolution, number of frames per second, and data transfer speed). If the live stream is available, the video player will automatically connect and start playing.

The maximum number of devices with which the application was tested was four: two transmission devices and two reception devices. We chose two sets of parameters (resolution, frames per second, transfer rate) to observe the qualitative differences between a low-quality and a higher-quality transmission. The two test sets were:

- Resolution: 176 X 144, 30 FPS, 250 Kbps
- Resolution: 352 X 288, 60 FPS, 400 Kbps

In the case of low quality, the delay between stream and playback is, on average, three seconds, extending to five, sometimes six seconds, in higher video quality. This result was to be expected as the amount of data transmitted is higher in the second situation. Image differences can be observed in the video captures below. At a lower video quality, the image is more pixelated, the details are not as well seen, compared to a stream at a higher quality, where the shapes are better defined and the colors more prominent. This can be observed on the source device, which makes the transmission, and on the destination one.

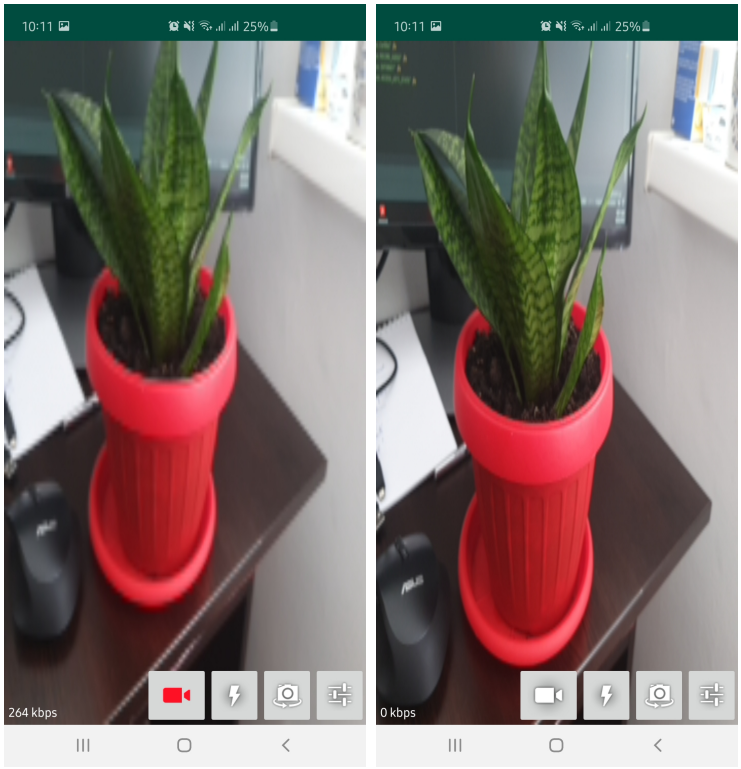


Figure 6.0.1: Comparison high and low video quality on the source device

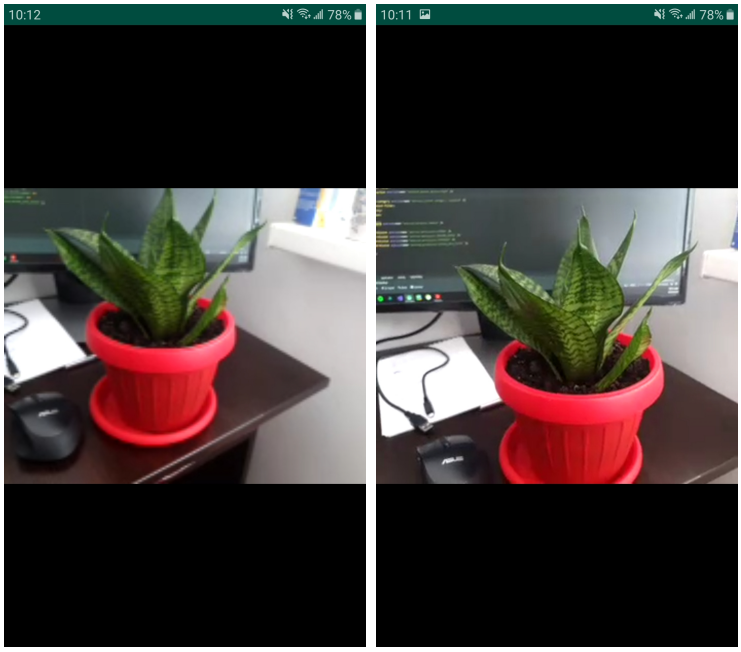


Figure 6.0.2: Comparison high and low video quality on the destination device

Next, we analyzed the application in terms of resources consumed. The results were obtained after an hour-long live transmission, with a source and a destination device. First, we wanted to document the amount of resources consumed locally on each phone. Energetically, on the source phone, the application consumed a total of 18% battery and 45.39 MB of storage, while on the destination phone, the application consumed only 10% battery and used a total

of 37.55 MB of storage.

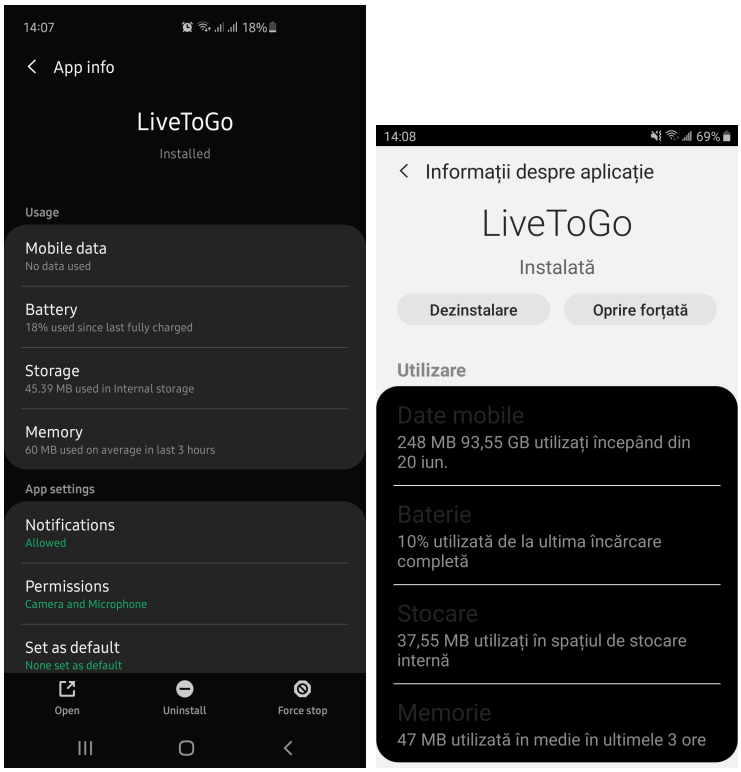


Figure 6.0.3: Resource usage on source (a) and destination (b) devices

The image below shows a statistic about the encoding formats received and sent by the Wawza engine. In our case, they are only RTSP and RTMP.

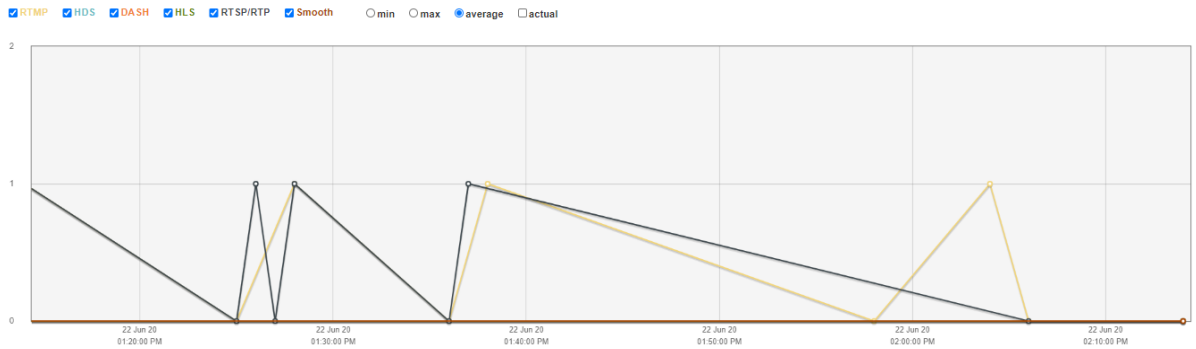


Figure 6.0.4: WSE video encoders

The Wawza Streaming Engine server also provides administrators with graphical information about the bandwidth consumed at any given time, Wawza CPU usage percentages, connections number, and memory consumed. The graph below shows the bandwidth of both input and output was largely constant (400 Kbps). The moments when the value reached 0, are because the stream is interrupted both on the source and the destination when the application runs in the background on the source phone. On the destination, it is necessary to restart the stream.

Figure 6.0.5: In and Out bandwidth

The maximum percentage of CPU used on the Wawza Stream Engine reached a value of 7%, for two connections, with an average value of only 2%.

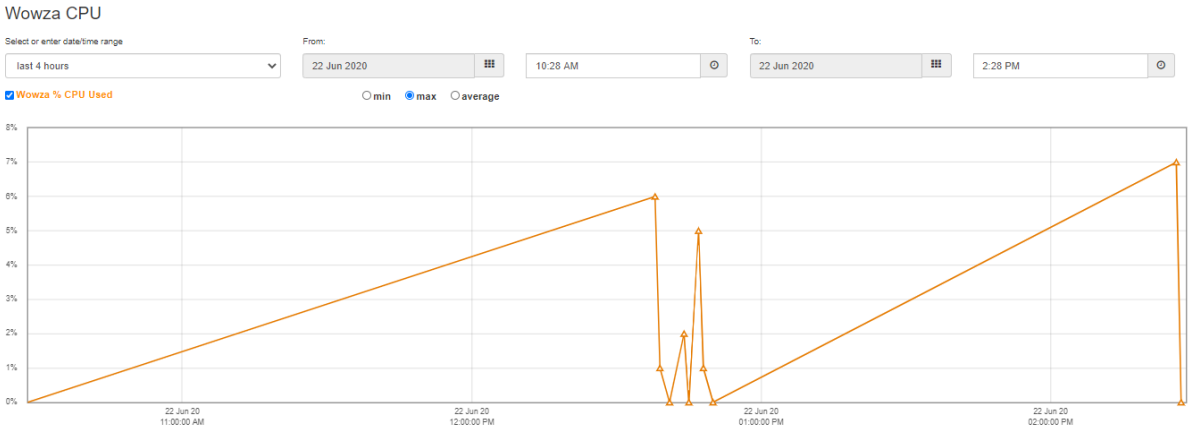


Figure 6.0.6: WSE CPU

This version of the application is not 100% functional. One of the main problems is identifying the default display of each device as landscape. Therefore, the image on the source phone is rotated 90 degrees, both for the front and back camera. The images below show the difference between portrait and landscape positioning.

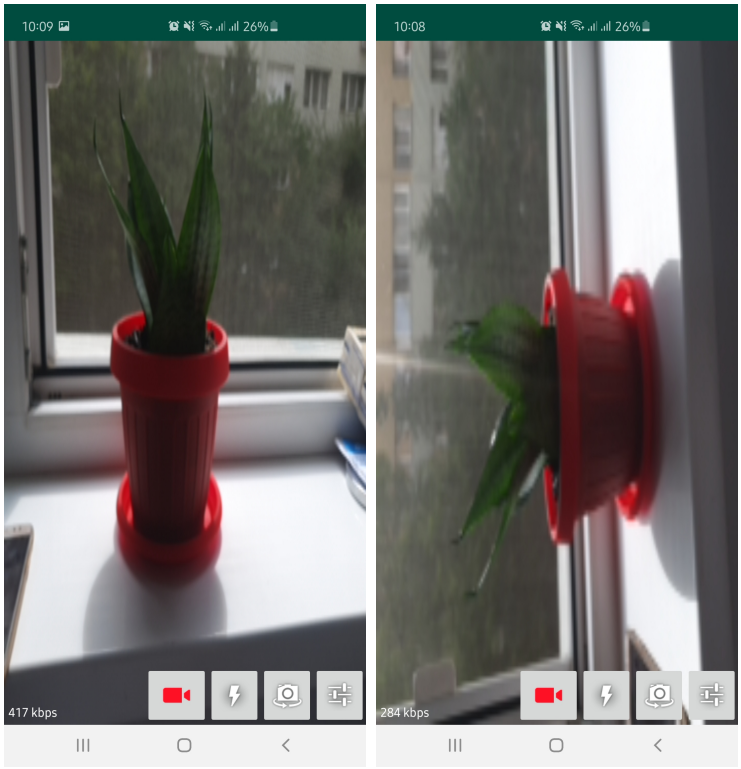


Figure 6.0.7: Landscape orientation

As a result, the maximum resolution that can be set is also affected. The maximum value at which we could record was 352 X 288. When setting a resolution of 720x480, we received an error: "The following settings are not supported on this phone. (start failed.)", as can be seen in the image below.

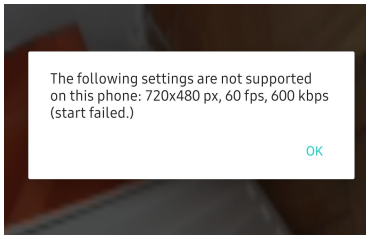


Figure 6.0.8: Resolution error

7 CONCLUSIONS AND FUTURE WORK

The purpose of this paper was to present a live video streaming application for the Android platform, with an external video switcher. Taking into consideration the implemented functionalities, the application works according to the indications, but with small bugs. Compared to other products already on the market, our application is inferior in performance, but has enormous potential, considering that it is the first product of its kind exclusively for the Android platform, which also holds the title of the most used mobile operating system.

The use of current technologies, defining in the live-streaming industry (H.264, AAC, RTP), places the application among the top software products, with high development potential. By implementing our own RTP packet interpretation server, we aim to reduce the cost of the application to small amounts, or even free.

The current version of the LiveToGo application is not final. We plan to develop it further in order to obtain a product with the best possible performance. Our development plans involve fixing all existing bugs and adding new features to the mobile application, such as live video editing elements. We also aim to implement our own RTP packet processing server, so that it no longer depends on third-party software, such as Wawza, and to implement a web application that facilitates video switching functionality for administrators.

BIBLIOGRAPHY

[1] H. Parmar M. Thornburgh. *Adobe's Real Time Messaging Protocol*. Last accessed: 20 June 2020. December 21, 2012. URL: https://wwwimages2.adobe.com/content/dam/acom/en/devnet/rtmp/pdf/rtmp_specification_1.0.pdf.

[2] fyhertz. *libstreaming API*. Last accessed: 18 June 2020. URL: https://github.com/fyhertz.

[3] R. Frederick V. Jacobson H. Schulzrinne S. Casner. *RTP: A Transport Protocol for Real-Time Applications*. Last accessed: 16 June 2020. URL: https://tools.ietf.org/html/rfc3550.

[4] H. Schulzrinne A. Rao R. Lanphier. *Real Time Streaming Protocol (RTSP)*. Last accessed: 16 June 2020. URL: https://tools.ietf.org/html/rfc2326.

[5] *LIVE:AIR Action*. Last accessed: 12 June 2020. URL: https://teradek.com/collections/live-air-family.

[6] *manyCam*. Last accessed: 12 June 2020. URL: https://manycam.com/mobile/.

[7] *mimoLive*. Last accessed: 12 June 2020. URL: https://boinx.com/mimolive/.

[8] *Number of available applications in the Google Play Store from December 2009 to June 2020*. Last accessed: 10 June 2020. URL: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/.

[9] *Number of Vimeo subscribers worldwide from 1st quarter 2015 to 1st quarter 2020*. Last accessed: 10 June 2020. URL: https://www.statista.com/statistics/705598/vimeo-subscribers-worldwide/.

[10] *Vimeo*. Last accessed: 12 June 2020. URL: https://vimeo.com/.

[11] *vMix*. Last accessed: 12 June 2020. URL: https://www.vmix.com/software/.

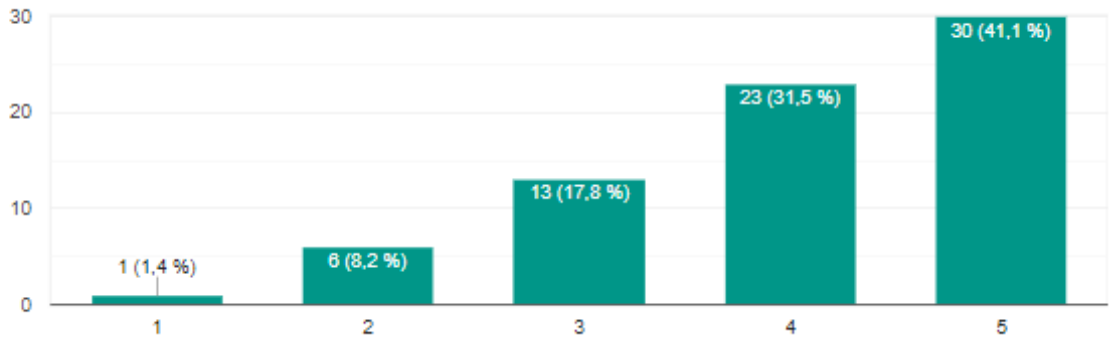
[12] Victor Yanev. *Live Streaming Statistics Every Marketer Should Keep In Mind in 2020*. Last accessed: 10 June 2020. URL: https://techjury.net/blog/live-streaming-statistics/.

A SURVEY RESULTS

Below are the results of a personal survey conducted to determine the main functionalities a live-streaming Android application should have, and the most used software applications in this domain. The total number of answers was 73.

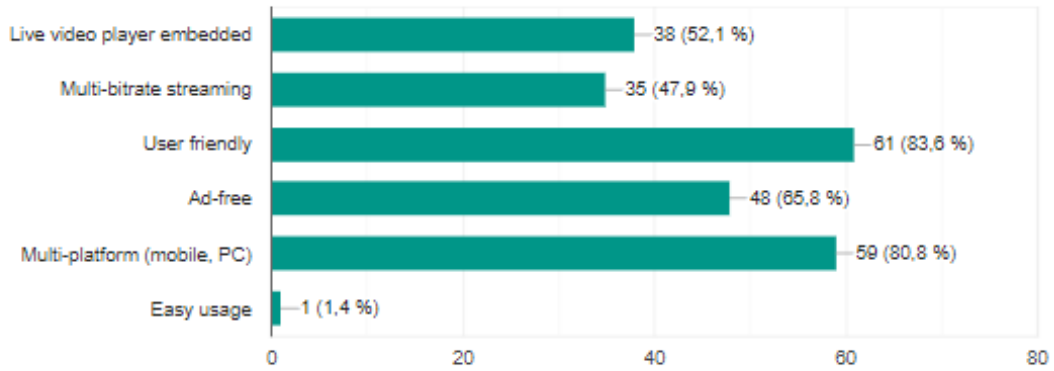
How likely are you to use a live streaming application during a social or corporate event?

73 de răspunsuri



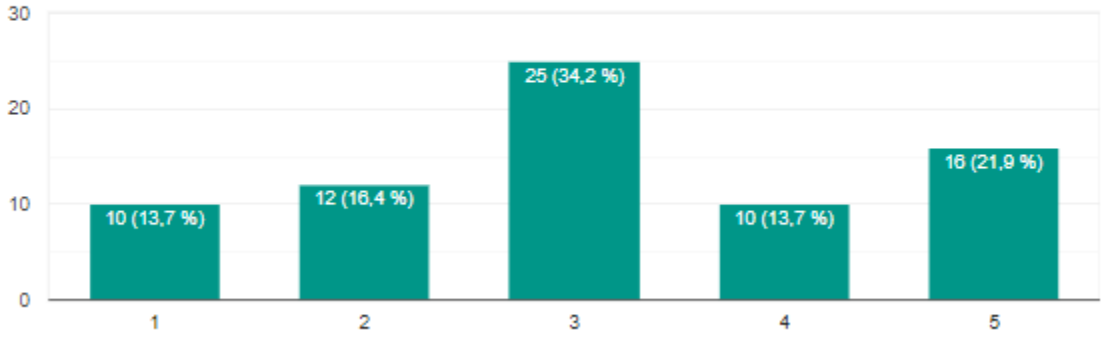
What are the key features such an application should provide?

73 de răspunsuri



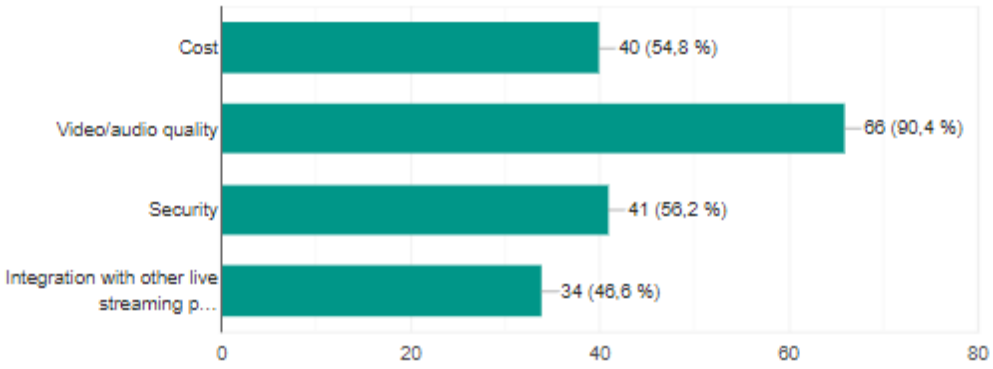
How important is multi-camera streaming (multiple angles) for you?

73 de răspunsuri



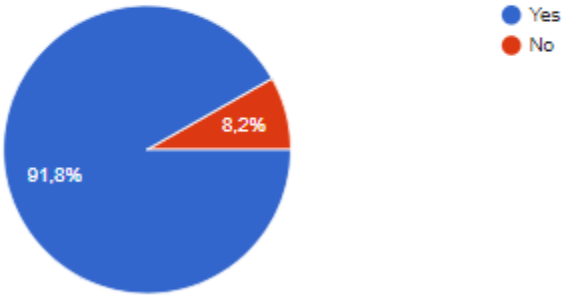
On what criteria do you base your choice of live streaming provider?

73 de răspunsuri



In case of a multi-camera application, would you prefer having control over which feed is streaming?

73 de răspunsuri



Which of these video streaming software have you used?

73 de răspunsuri