Politecnico di Torino
ICT for Smart Societies

Master's Thesis

Combined ICT Technologies for Supervision of Complex Operations in Resilient Communities

Academic Supervisor:

Prof. Enrico Magli

Supervisor:

Paola Dal Zovo (Santer Reply)

Author:

Seyyed Sadegh Bibak Sareshkeh – s235986

Abstract

Despite the promotion of technology in various fields and its emerging in people's conventional life, natural and manmade disasters and incidents are still among the top external causes of injuries and mortality. Moreover, with the rise of the population and rise of megacities, management of emergency cases is becoming more and more complicated every day. These factors have forced Emergency Management (EM) organizations and related institutions to consider implementing various methods and policies to become more robust and efficient while conducting operations.

Further, in recent years, Information and Communication Technology (ICT) has been identified as one of the most promising success factors to improve emergency management processes. In the last decades, advances in mobile computing technologies have had a fundamental impact on devising ICT systems supporting the daily work of people. Although the domain of EM has evolved consistently over time and several supporting technologies have been developed, there is a lack of suitable methods to realize the benefits of ICT by emergency managers.

The importance of monitoring has led the IP camera market to develop considerably and has brought numerous solutions depending on the scenario. IP cameras are installed in various places in cities (e.g., big squares, stadiums, metros, crowded intersections, etc.) and along the roads for the means of monitoring. Furthermore, in recent years many portable IP camera devices have been developed which reduce the burden of installation considerably and provide service for many diverse locations.

However, although mounting and installation of IP cameras have become very cheap, typically to obtain useful information there is still the need for the presence of a human being to observe their output. Thus, often most of the data is processed after a special event such as an emergency has already occurred, causing losing the benefit of having a real-time monitoring tool and as a result, acting instantaneously.

In this thesis work, we aimed to design a monitoring system to manage the information obtained from the scene of the emergency field in order to help the operators and decision-makers to have a more complete view regarding the emergency field so as act properly and more quickly.

Various solutions were evaluated for each element of the monitoring system including the Media Streaming Server, Media Streaming Protocol, and Video codecs and the best tool is selected for each part. Moreover, an object detection solution is implemented on top of the video elements in order to help the supervisor to identify the required information faster and more accurately.

Table of Contents

List of Figures

List of Tables

Chapter I: Introduction

The concept of “extreme events” encompasses both natural and man-made emergencies, catastrophes, and disasters of exceptional and unthinkable magnitudes. Extreme events hit with little or almost no warning, cause massive injuries, losses of life, and damage to property, displace populations, severely disrupt critical infrastructures including the information and communication infrastructures, pose enormous health risk to survivors, and regularly surpass the capacity of local responders to adequately and effectively respond by a wide margin. When an extreme event unfolds, the operating picture may not be achievable for 24 or 48 hours (or longer) such that the effectiveness of initial responses might be marginal. Typically, it requires extended external support and non-local resources for months or even years to first respond and then help recover from the extreme event [14].

Despite the promotion of technology in various fields and its emerging in people’s conventional life, natural and manmade disasters and incidents are still among the top external causes of injuries and mortality [1]. Moreover, with the rise of the population and rise of megacities, management of emergency cases is becoming more and more complicated every day [2, 3]. These factors have forced emergency management (EM) organizations and related institutions to consider implementing various methods and policies to become more robust and efficient while conducting operations [4,5].

However, decision making in such complex and urgent environments creates extraordinary challenges for EM organizations’ personnel when time and available resources are limited and on the other side, the consequences of failure are severe. Risks are high, both for the personnel taking action and for the population whose lives and properties they seek to protect. As social systems become more interdependent with physical and technical systems, the area of possible interactions among individuals, groups, organizations, and the context in which they function increases, and the number of factors that influence potential actions and outcomes in constructive or destructive ways also increments. Consequently, response to extreme events becomes an emerging, large-scale, sociotechnical system of individuals, groups, organizations, and jurisdictions that necessarily need to coordinate their actions to mobilize coherent, effective operations in an often disrupted and dangerous environment. In some cases, many of the participating individuals and organizations may not know one another, and may not be familiar with the particular context for action, but presumably, they rely on a common base of knowledge and training. Thus, it is of great importance to present as much information as possible earlier and during these operations to help the managers realize the status of the task and evaluate the vital requirements and shortcomings to command, respectively [4, 5].

In recent years, Information and Communication Technology (ICT) has been identified as one of the most promising success factors to improve emergency management processes. In the last decades, advances in mobile computing technologies have had a fundamental impact on devising ICT systems supporting the daily work of people. More recently, the developments in wearable computing and augmented reality technologies allow for conceiving a new generation of systems that can further reshape how people work and collaborate. Although the domain of EM has evolved consistently over time and several supporting technologies have been developed, there is a lack of suitable methods to realize the benefits of ICT by emergency managers [14].

Moreover, the benefits and risks of ICT investments are often unclear to emergency managers and involved organizations. While to many it may seem that more use of indicates better emergency services, there are also issues associated with the introduction of its usage. ICT can be expensive. For example, it may change the traditional balance within organizations. Its introduction can have political consequences. In realizing the use of different information and communication technologies, it is important to explore the variety of use as well as how a particular technology is put into action. Moreover, it is critical to understand how the people who deliver emergency services manage the data and knowledge that ICT provides, especially in terms of potential threats for data security and privacy [15].

One of the systems which are designed to aid in the emergency condition is Urban Integrated Emergency Response System (UIERS). UIERS is a set of information systems integrating with communication, command, dispatch, and position. Without changing the administrative establishment of the emergency department of each section, the UIERS can achieve joint enforcement among city emergency sectors and increase the quality of emergency services. Figures 1-2 show implementations of UIERS [11].

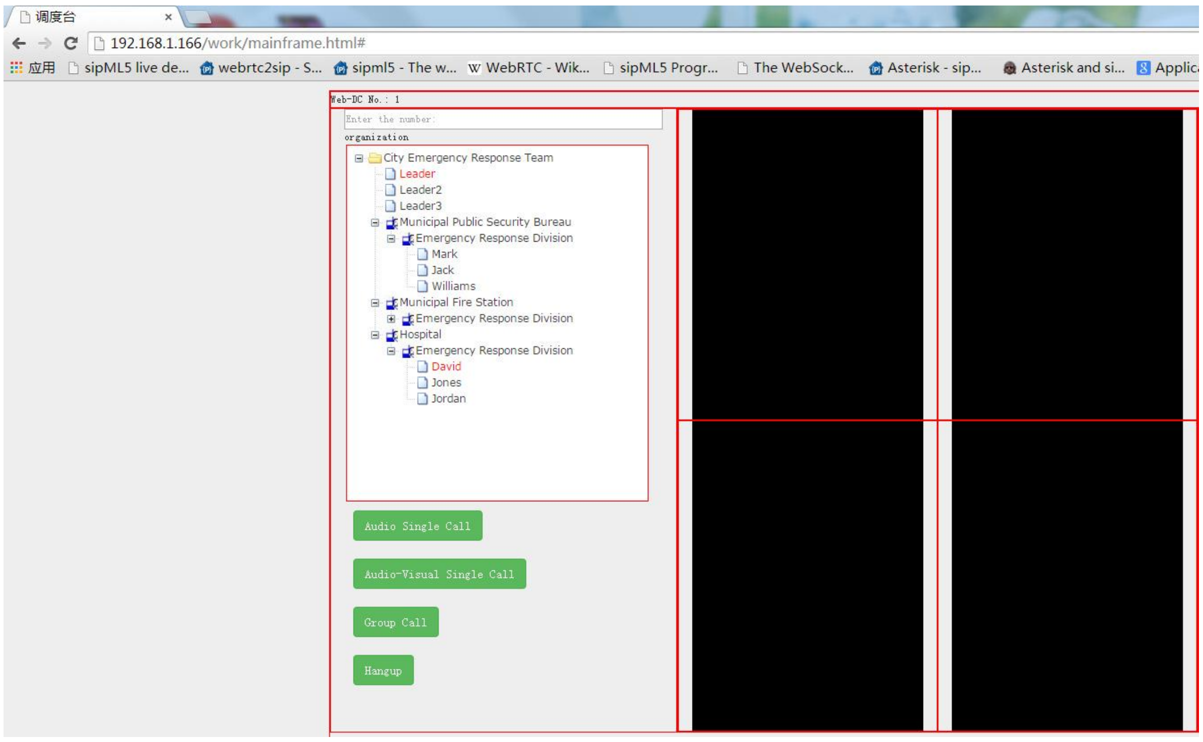


Figure 1: UIERS, Crew’s information presentation [11]

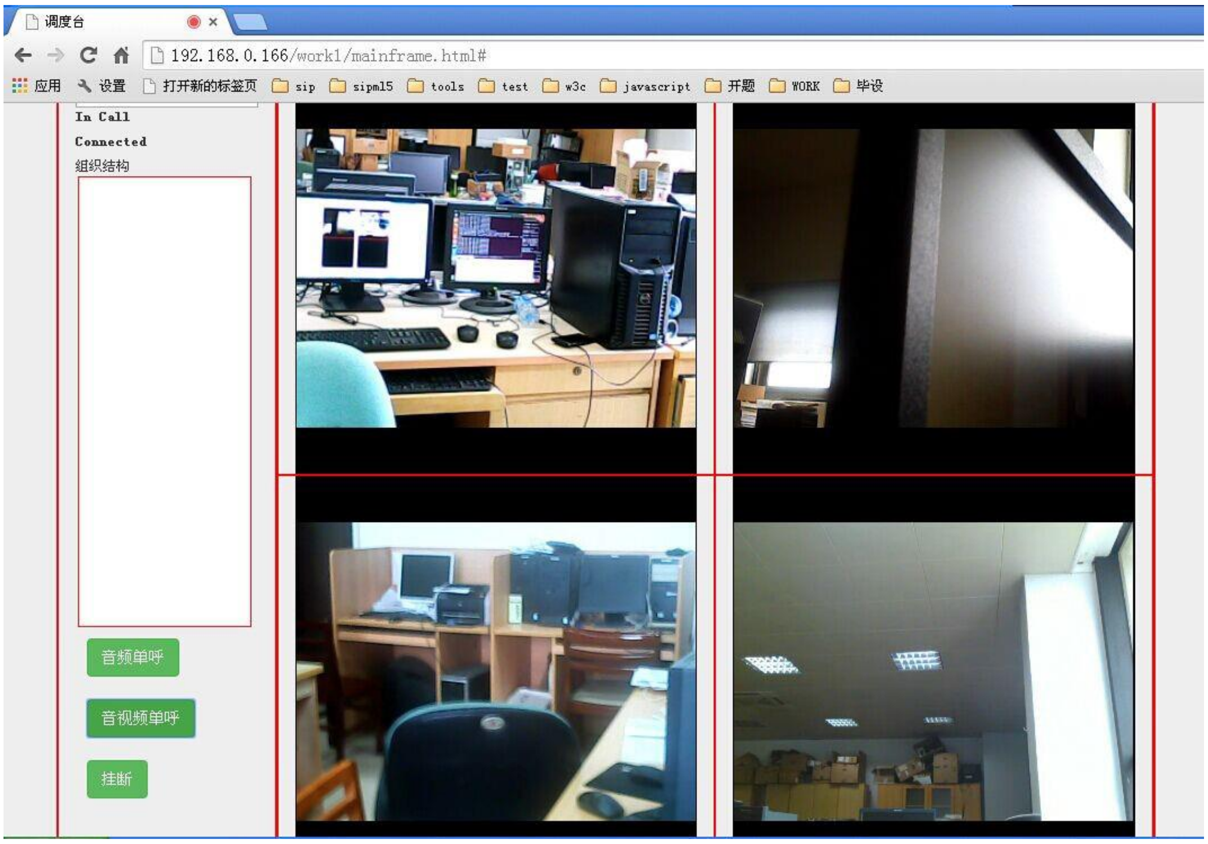


Figure 2: UIERS, Video Monitoring [11]

As mentioned, the processing of public emergency events in a city is a complicated project, as it requires response team crews to have command ability, cooperative ability, and professionalism in their field, as well as the participation of various departments in the city. During processing, the team needs to respond quickly, process effectively, and communicate fully. The team needs to obtain real-time feedback of a situation from the field personnel's equipment in the process of deployment. Emergencies are usually solved by a response team, which is made up of the elites from various departments, and orders can only be given through voice by analogue interphone or cellphone. In this situation, orders cannot be delivered to every field personnel quickly and clearly. When using an analogue interphone, the quality of voice is not very good and it is prone to be taped. Besides, the communication range of an analogue interphone is very limited and cannot cover the whole city or even a district of a city, which cannot satisfy the needs of the management of the whole city. Thus, a new type of communication method seems to be necessary for the management of modern cities [11].

Generally, custom digital terminals cannot be deployed to a large number of field personnel. However, with the popularization of mobile communication technology, everyone has a smartphone or tablet now. Therefore, in utilizing this equipment, everyone can offer live multimedia information to the emergency response team and respond to orders and unexpected events throughout the system [11].

Furthermore, EM is a hot topic in the field of Internet of Things (IoT). Using IoT, an environment is created in which information from the sensors connected to the network is collected to be shared with others in real-time. An example of this scenario is a project called

S.A.F.E, which is dedicated to aiding emergency management groups to have more information regarding the location and the number of victims and trapped people in a building in the occurrence of an earthquake, by inserting sensors inside home furniture and other parts of the house [20]. Besides, robotics research is another field that is already changing emergency management in which robots are being designed for operating in hazardous situations such as radiation, in order to assist the operation of crowd evacuation, and for search and rescue activities. Civilian drones are getting more popularity for emergency and prevention tasks. For example, they can be used in order to see ahead of impassable areas such as jungles, hills or even congested roads and reach the desired location and evaluate the scene before emergency services arrive. In addition, drones can be equipped with technologies such as infrared cameras to help firefighters spotting people trapped in blazes or missing people in rivers or jungles. Finally, there are some studies in the field of human-centered sensing and the impact of social media on EM [12, 13].

Using IP cameras can help the involved staff including mission supervisors to have a better observation by providing additional views to the ongoing task. Moreover, they can be used both in fixed and mobile installations, depending on the scenario requirements; for instance, wearable cameras or cameras mounted on drones can be useful in a certain scenario, while fixed-cameras can be a more reasonable answer and allows long-time monitoring in case of presence of pre-installed cameras. Finally, using IP cameras provides robustness by assuring an alternative monitoring tool (In cases where drone usage is not possible due to the presence of dense smoke caused by fire).

The importance of monitoring has led the IP camera market to develop considerably and has brought numerous solutions depending on the scenario. There exists plenty of customized solutions including the ability to use rechargeable batteries or Power over Ethernet (PoE), being dependent on LAN networks or use wireless technologies to transfer the data, ability to capture clear images in both day and night, etc. IP cameras are installed in various places in cities (e.g., big squares, crowded intersections, etc.) and along the roads for the means of monitoring. Furthermore, in recent years various portable IP camera devices have been developed which using them can reduce the burden of installation considerably and provide service for many diverse locations.

However, although mounting and installation of IP cameras have become very cheap, still there is the need for the presence of a human being to observe their output. Thus, often most of the data is processed after a special event such as an emergency has already occurred, causing losing the benefit of having a real-time monitoring tool and as a result, acting instantaneously. Thus, real-time processing of the visual output from monitoring systems has become a trending topic in the field of computer vision. Various researches have been dedicated to the implementation of computer vision techniques to assist in various stages of emergency management, including prevention, detection, assistance of the response, and understanding of emergencies. Moreover, there are some researches dedicated to identifying a specific scenario in each stage of the emergency case. Figure 3-7 demonstrate some use cases of implementation of video analysis in different stages of EM [12].

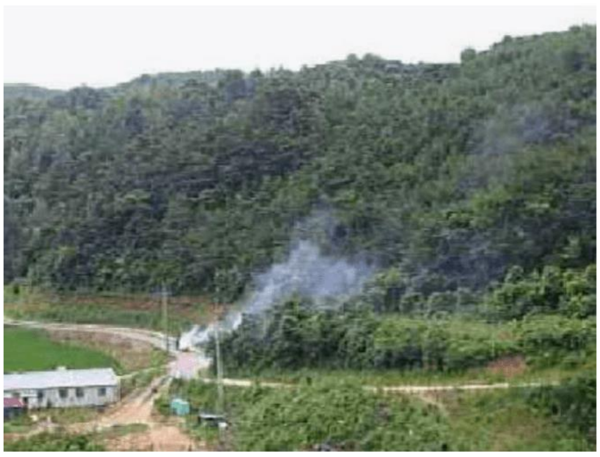


Figure 3: Example of fire prevention by detecting smoke [12].

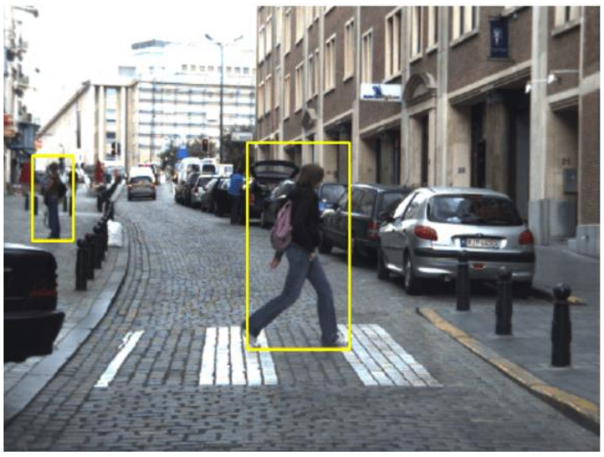


Figure 4: Example of pedestrian detection using a vehicle onboard camera to prevent pedestrians from being run over [12].

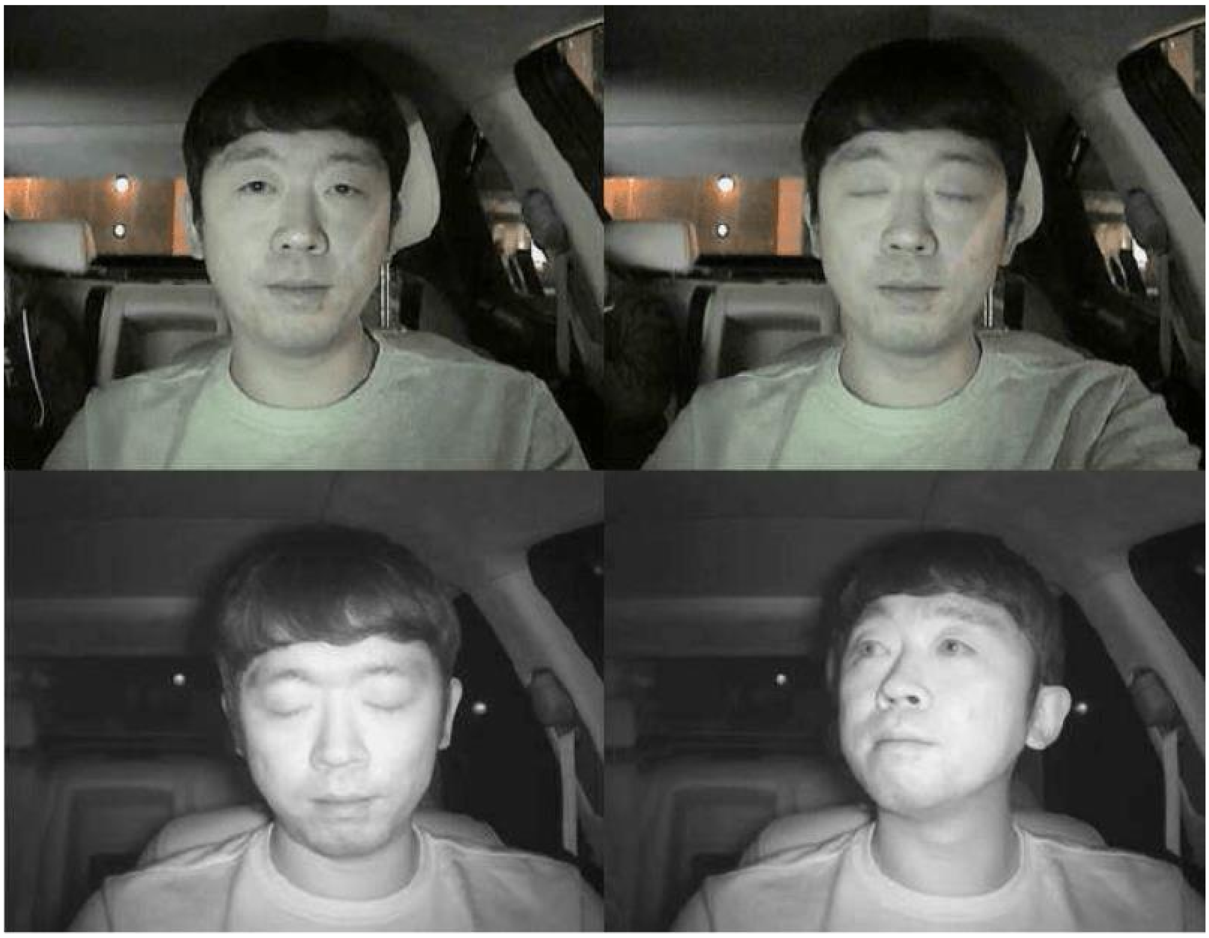
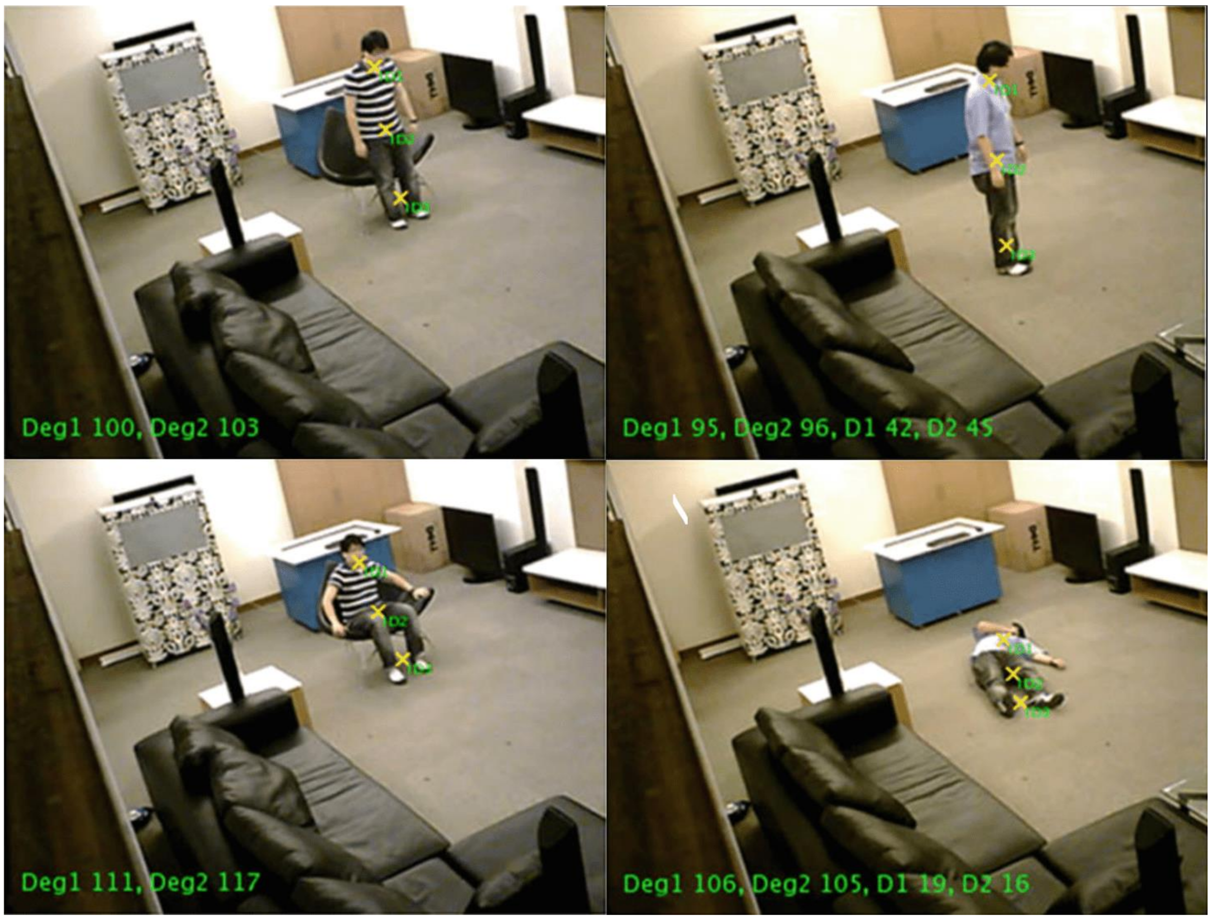


Figure 5: Example of drowsy driver detection [12].


Deg1 100, Deg2 103
Deg1 95, Deg2 96, D1 42, D2 45
Deg1 111, Deg2 117
Deg1 106, Deg2 105, D1 19, D2 16
Figure 6: Detecting two very different scenarios such as sitting and falling [12].

Chapter II: Background Information

Before starting explaining the thesis work, it is necessary to clarify some terms including streaming, protocol, transport or container and the codec.

2-1. HTML5 Video Playing

HTML5 video playing happens when a <video> tag is put in the web page and a certain "src" is set for it. While HTML5 streaming is somehow alike, however, in this case, the "src" points not to a complete video file but rather to an ever-updating video stream, e.g. YouTube usually does HTML5 video playing, while for example Twitch does HTML5 video streaming. That is, <video> tag only cares about "src" pointing to any video stream and does not care about the formation and transmission of the video, or even the browser support.

2-2. Streaming Protocol

The protocol defines how communicating actors exchange data. The stream can flow from the streaming server to the client (video playback) or from the client (e.g. IP Camera) to the streaming server (broadcasting) which in turn, streams to multiple clients. Some examples of protocols are RTSP, RTMP, and HTTP.

2-3. Container

Transport or container defines how compressed video gets packed into bytes for transmission over from one party to another (using some protocol). For example, MPEG-TS and RTMP are containers.

2-4. Codec

The codec is a way to compress raw video before streaming; most streaming services do not deal with codec-level compression and work with protocols and transports only. Some example of codecs are h264, aac, mp3.

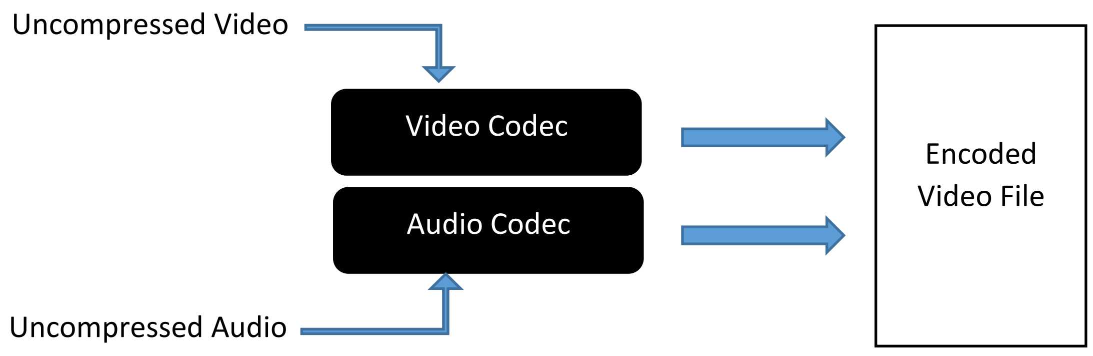


Figure 7: Simplified schema of encoding a video file using video and audio codecs

2-5. Adaptive Streaming

Adaptive bitrate streaming or Adaptive Streaming is a technique used in streaming multimedia in order to deliver the stream data to the receiver in an optimized mode in terms of quality and delay. Due to undeniable dependency on HTTP protocol, it is extremely required to design and implement methods in which the data is delivered efficiently in a broad and distributed network such as the Internet.

It is mainly based on detecting the user devices' properties such as bandwidth and CPU capacity in real-time to transmit the media stream with appropriate quality. Moreover, an encoder is required which is capable of encoding a single source media (video or audio) at multiple qualities and bitrates. E.g., the server maintains multiple profiles of the same video that are encoded in different bitrates and quality levels. Next, the video object is segmented into small multi-second parts to be sent to the client. Just before the start of the streaming procedure, the information regarding the available stream profiles and segments of the streams is sent to the client by a manifest file. Typically, at the starting stage, the client requests the segments from the lowest bit rate stream. Depending on the underlying network conditions, a player can then request different fragments at different encoding bitrates. If it is recognized that the connection condition is good enough, (for example the download speed is greater than the bit rate of the downloaded segment), then the client requests the next highest bit rate segments. However, throughout the streaming, if the client finds the download speed for a segment is lower than the bit rate for the segment (the network's throughput is deteriorated), it requests a lower bit rate segment.

The mentioned method improves server-side scalability. Since the player client switches between streaming different encodings depending on available resources, very little buffering is consumed, the streaming will have fast start time and finally, it leads to a good experience for both high-end and low-end connections [17]. Figure 8 shows a simplified schema of how the adaptive streaming works.

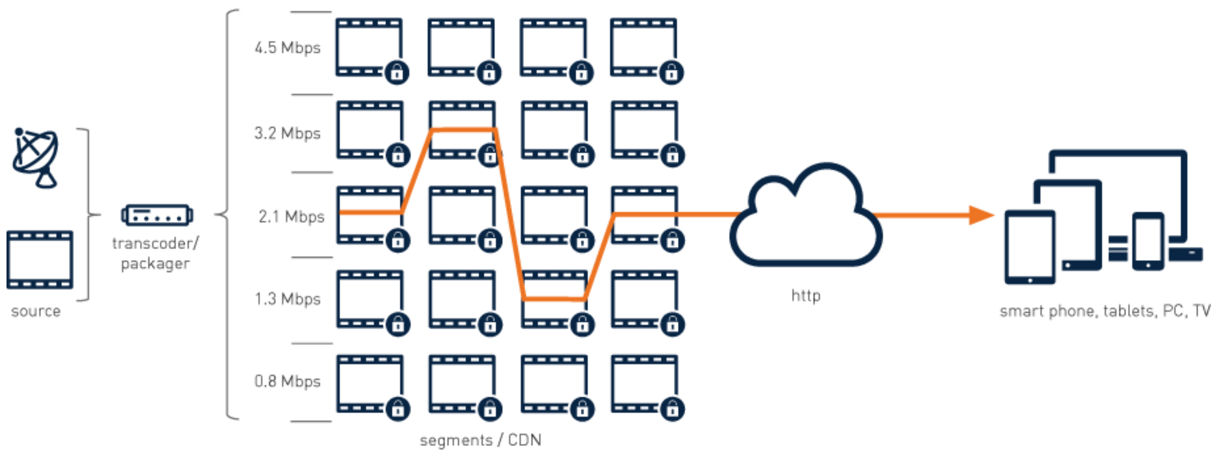


Figure 8: Adaptive Streaming working mechanism [21]

2-6. Media Source Extension (MSE):

MSE is a specification for the front-end that allows JavaScript to send streams to media codecs inside Web browsers. Thus, it gives the developers the possibility to implement client-side prefetching and buffering code for streaming media entirely based on JavaScript code.

Moreover, MSE allows replacing the “src” value present inside the HTML5 media elements with a reference to a MediaSource object, which is a container for information and can refer to multiple SourceBuffer objects representing the different chunks of media that builds the entire stream we are interested in. Besides, MSE provides control over the amount and the frequency of fetching the content, and regulations over memory usage details, such as when buffers should be cleaned [16].

Figure 9 shows the mechanism of MSE.

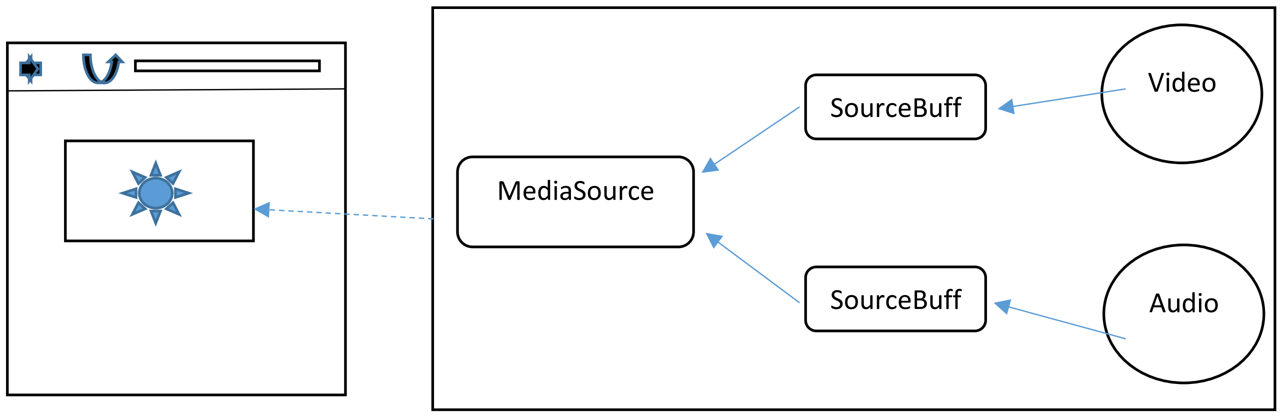


Figure 9: Simple Schema of MSE code elements

Chapter III: Assessment of technologies

3-1. Transcoding

Generally, the process of converting compressed video signals to other video signals is called video transcoding. Usually, the main goal of this procedure is to adapt video features such as bit rate, resolution, or codec, to meet the requirements of communication channels and endpoint devices. It is mainly composed of compressing and decompressing video signals with video coding techniques. The compression is often called encoding and decompressing is denoted as decoding.

Figure 10 shows the implementation of video encoding and decoding.

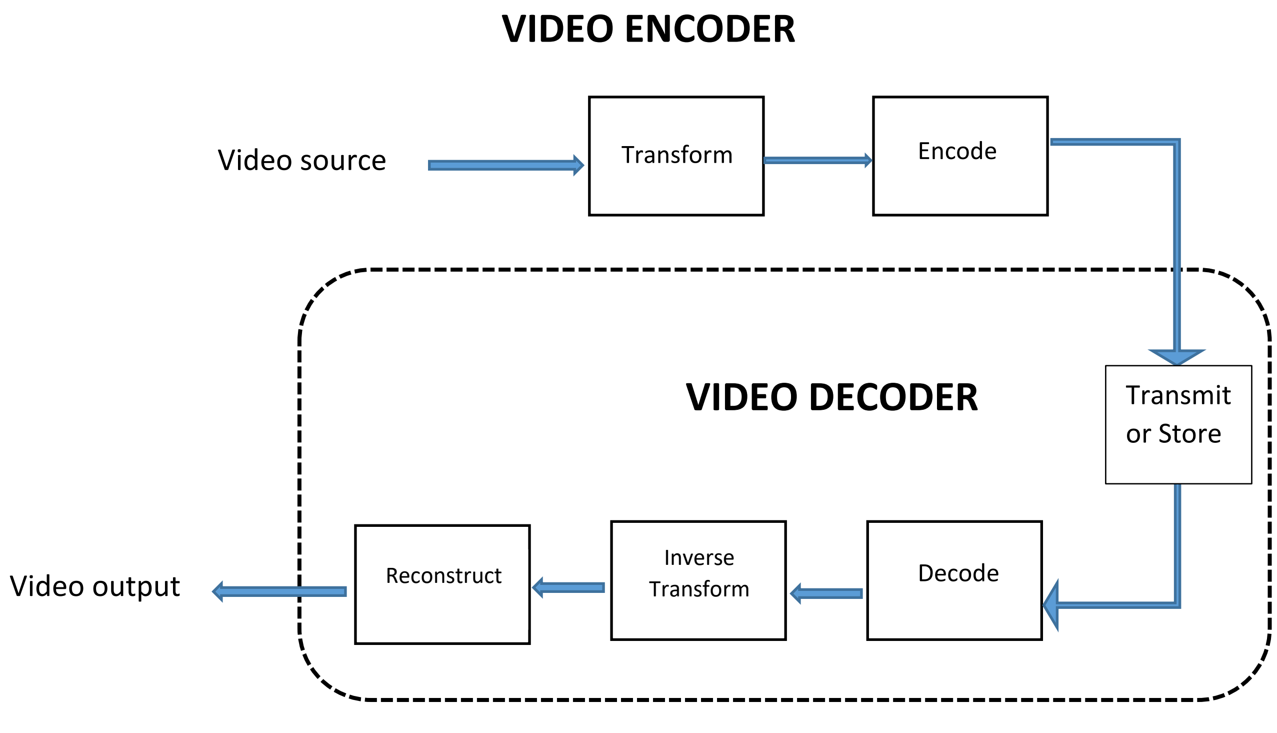


Figure 10: Video encoder and decoder

3-1-1. Motion JPEG

Motion JPEG (M-JPEG or MJPEG) is a popular video compression format. In MJPEG, each video frame (each field of a digital video sequence) is compressed separately as a JPEG image. While being originally developed for multimedia PC applications, M-JPEG is now used by video capturing devices such as digital cameras, IP cameras, and webcams, and also by non-linear video editing systems. Moreover, it is natively supported by the most popular Web Browsers such as Google Chrome, Safari, Mozilla Firefox and Microsoft Edge and by media players such as QuickTime Player.

MJPEG is an intraframe-only compression scheme, and due to lack of interframe prediction, its efficiency is limited to 1:20 or lower, depending on characteristics required in the compressed output. While other modern interframe video formats (such as MPEG1, MPEG2 and H.264/MPEG-4 AVC) obtain compression ratios of 1:50 or better. However, since frames are compressed separately from another, MJPEG requires lower processing power and memory requirements on hardware devices.

The image quality of MJPEG depends on each video frame's static (spatial) complexity. That is, frames with large smooth transitions or monotone surfaces are compressed easier and are more likely to be alike the original video frame. However, few visible compression artifacts may be noted anyway. On the other hand, frames exhibiting complex textures, fine curves and lines are more likely to exhibit Discrete Cosine Transform (DCT) artifacts such as ringing, smudging, and macroblocking. Due to the mentioned points, it can be conceived that the compressed MJPEG video is insensitive to motion complexity (i.e. the variation of the original video over time). That means the video frames are neither deteriorated by highly random motions and nor improved by the absence of motion. The mentioned cases are two opposite extremes, which are commonly used to test interframe video formats.

3-1-2. MPEG4

MPEG-4 is a compression method of audio and visual (AV) digital data. It was designed as a standard for a group of audio and video coding formats. Initially, it was aimed primarily at low bit-rate video communications; however, its scope as a multimedia coding standard was later expanded. MPEG-4 is efficient across a variety of bit-rates ranging from a few kilobits per second to tens of megabits per second. Uses of MPEG-4 includes various fields such as compression of AV data for web (streaming media) and CD distribution, voice (telephone, videophone) and broadcast television applications. It employs many of the features of the older versions of MPEG (MPEG-1 and MPEG-2) and other related standards, and adds new features such as Virtual Reality Modeling Language (VRML) support for 3D rendering, object-oriented composite files (including audio, video and VRML objects), support for externally specified Digital Rights Management and various types of interactivity.

Figure 11 demonstrates a general block-diagram of the MPEG-4 encoder.

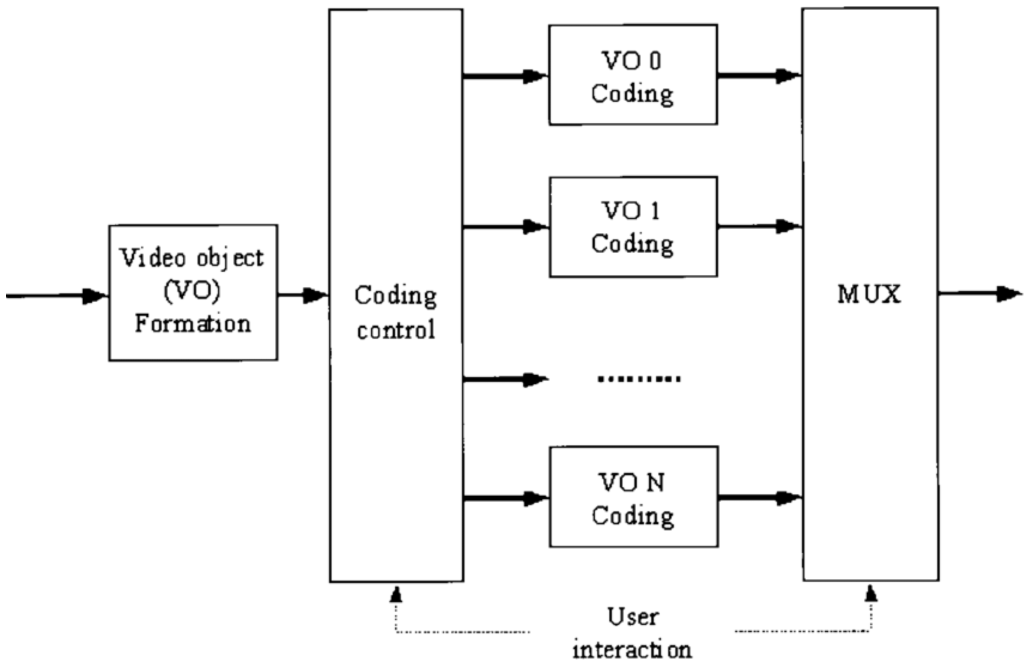


Figure 11: General block diagram of MPEG-4 algorithm (encoder). [18]

3-1-3. H.264

H.264 is a standardized video compression format that was designed to provide high video quality with lower bit rates. It is vastly adopted in the video industry and is used in a range of different applications and contexts especially in video streaming on the internet.

H.264 is composed of three different types of frames I-frames, P-frames and B-frames. The first frame of any video is always an I-frame or intraframe and it contains the entire picture. I-frames are frames that are independent of all other frames and can be decoded without any reference to any other frame. Next, P-frames are predictive intraframes, which means that they reference previous I-frames or P-frames and only contains the difference from the previous frame. Finally, B-frames are bi-predictive interframes. They are quite similar to P-frames in the sense that they contain the difference from the previous frame but can also reference future frames.

H.264 streams' frames are organized in Group Of Pictures (GOP). As mentioned, a GOP starts with an I-frame and contains several P/B-frames. However, as P/B-frames contain the picture expressed as changes in the scene, they are considerably smaller than I-frames; this structure of organizing frames results in a lower bit rate compared to other compression formats.

Figure 12 indicates an example of the structure of a GOP in H.264.

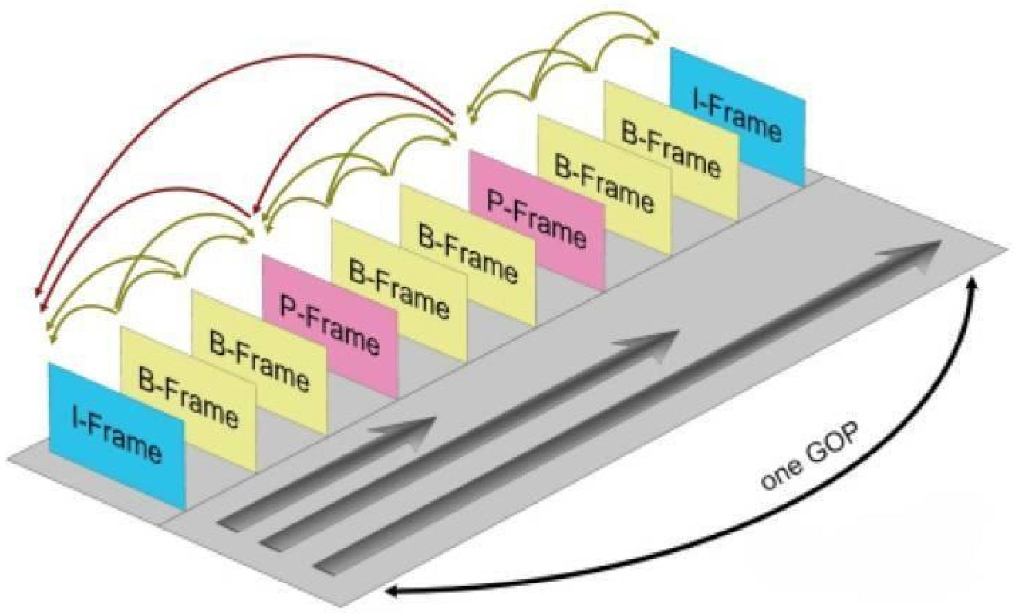


Figure 12: Structure of a GOP used in H.264 [19]

3-1-4. H.265

High-Efficiency Video Coding (HEVC), also known as H.265 is a video compression standard, designed as a successor to H.264. In comparison to H.264, H.265 offers up to 50% better data compression at the same level of video quality. This advantage will lead to reduced required disk space for storing the videos and as well, less bandwidth for broadcasting live video streams. Furthermore, it supports considerably high resolutions including 8K UHD.

H.264 mainly uses macroblocks, whereas H.265 processes information in Coding Tree Units (CTUs). While macroblocks can span 4x4 to 16x16 block sizes, CTUs can process up to 64x64 blocks, which results in the ability to compress information more efficiently. In comparison to H.264, H.265 also has better motion compensation and spatial prediction. However, in order to compress the data, H.265 requires more advanced hardware (such as the BoxCaster Pro) [22].

Table 1 compares the recommended bandwidth for H.264 vs. H.265 encoding.

Resolution	**Minimum Upload Speed**	
	H.264	**H.265**
480p	1.5 mbps	0.75 mbps
720p	3 mbps	1.5 mbps
1080p	6 mbps	3 mbps
4K	32 mbps	15 mbps

Table 1: A comparison of H.264 and H.265 performance for different video resolutions [22].

3-2. Encoders

Two popular and widely used encoders are FFmpeg and GStreamer. The following is a description of their characteristics and advantages.

3-2-1. FFmpeg

FFmpeg is an open-source project consisting of a collection of libraries and programs for managing video, audio, files and streams. It is widely used for format transcoding, basic editings, video scaling, video post-production effects, and standards compliance. FFmpeg libraries are used in numerous software media players such as VLC. It also has been included in core processing for YouTube and the iTunes inventory of files. In addition, codecs for the encoding and/or decoding of most of all known audio and video file formats are included, The FFmpeg program itself provides processing of video and audio files, using the command-line interface.

Besides the core FFmpeg command line program for transcoding multimedia files, FFmpeg includes "libavcodec", which is an audio/video codec library that can be used by both commercial and free software products, and "libavformat" (Lavf), which is an audio/video container mux and demux library.

3-2-2. GStreamer

GStreamer is a pipeline-based multimedia framework. Its principal advantage is linking a wide variety of media processing systems to help to form workflows from very simple ones to complex. GStreamer supports a wide variety of media-handling components and the pipeline design aids to generate various types of multimedia applications such as video editors, transcoders, streaming media broadcasters and media players.

One example of the application of GStreamer is designing a system that reads files in a specific format, processes them, and exports them in another desired format.

3-3. Live Streaming from IP Cameras

Figure 13 shows the general architecture of video streaming:

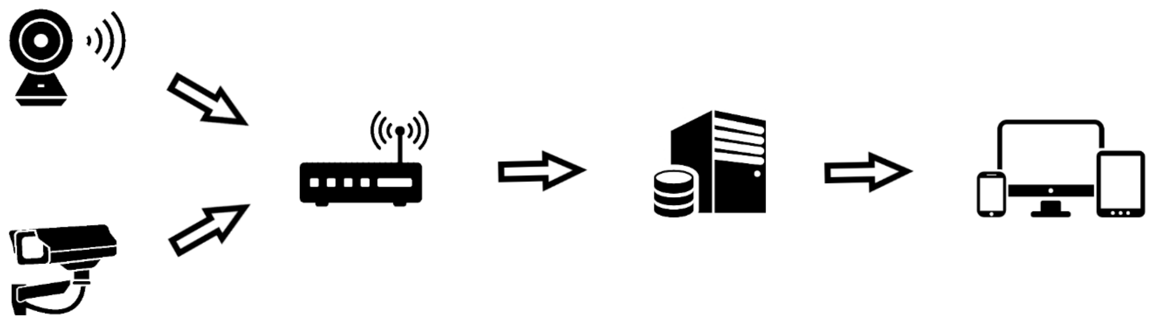


Figure 13: The general architecture of video streaming

IP cameras (both previously installed and the portable ones) transfer the stream through a router with a public IP (which might be integrated with the IP camera itself or separated) to a server where the video is stored and then the server restreams the video to the clients for playback.

While depending on the capabilities of IP cameras, streaming services, etc. some parts of the mentioned architecture can be integrated with other parts and result in a more simplified architecture:

I. Camera with the whole streaming service

Some IP Camera producers provide streaming and recording services on their cloud. The advantage of this solution is that using these services, we will end up having a robust, integrated and supported solution, which makes us needless to worry about the support and maintenance in case of failures or other issues. While on the other hand, besides considerable cost, since these solutions are designed simply for security and monitoring scenarios, they limit us in terms of the total length of recorded videos, availability of the recorded videos (they may be watched only through the manufacturer's user interface), and limited options for choosing transfer protocols and file formats which is correspondent to our specific need.

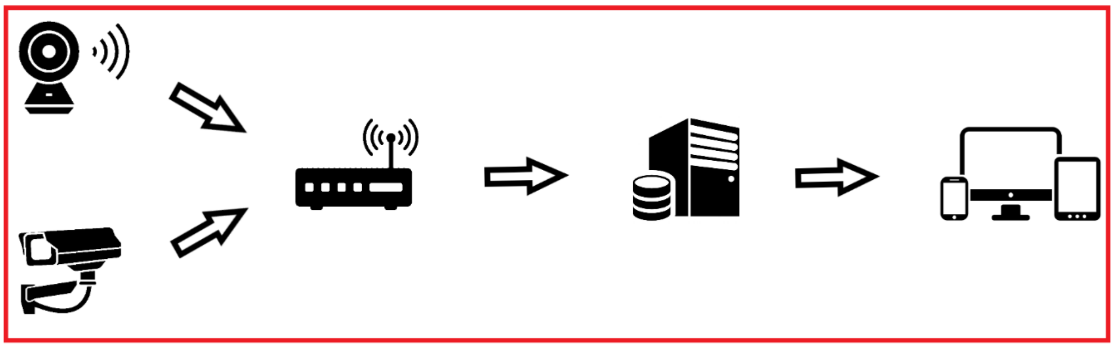


Figure 14**:** Camera with the whole streaming service

It is worthy to mention that several camera models use 4G/LTE to transmit their data to the cloud. However, the problem of this solution is that the video data is only available on their cloud dashboard and it is not possible to cache these streams and store them directly on another server to have enough control on the stored data.

II. IP Camera separated from the streaming system

Another widely implemented option is to purchase a camera which is not or optionally dependent on its cloud infrastructure and use media streaming systems e.g., Wowza, Red5 Pro, Nginx+, Plex, Icecast, etc. as the media server. These technologies provide robust solutions for streaming and handle a considerable portion of the connection management between the cameras and servers, transcoding, storing, rebroadcasting from the servers to the client and SDKs for embedding the video player inside web pages and mobile applications.

Figure 15 shows the architecture of the mentioned solution.

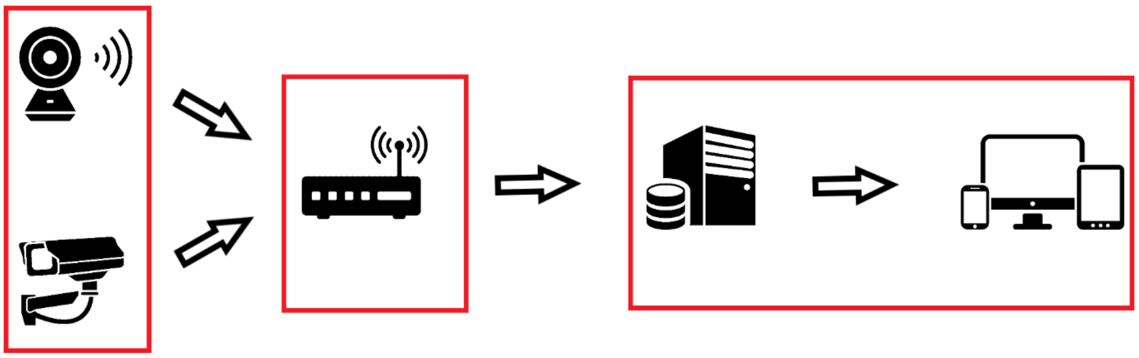


Figure 15: IP Camera separated from the streaming system

These products support different operating systems, protocols, containers, and formats. Moreover, they usually support augmentations like adaptive bitrate streaming and provide customer support. Thus, due to the possible use of different IP cameras based on the scenario, we should consider a system, which provides us with the maximum coverage of the technologies available.

Moreover, these products are usually not free. Further, their free version allows only limited access to the service using trial versions to test the product, or limitations such as restricted live broadcasting, temporary storage of the videos inside the server, limiting the number of IP cameras used by the user, the quality of the streaming, and disabling player SDK. Thus, these integrations do not apply to our case that requires a scalable solution.

III. A separate solution for each section

The last but the most scalable option is to choose the best feasible solution for each main part of the architecture while considering its flexibility and integrability with other sections of the architecture. For the media streaming part, there exist few open-source media streaming systems including Nginx, Janus WebRTC, Kurento, Red5, MistServer, Nimble Streamer, etc. For the front-end side, it is required to deploy an external player, since as mentioned before, usually the open-source version of the streaming servers are not composed of HTML or Android player SDK. For this part, there exists several paid and free libraries which are capable of playing one or few formats. Figure 16 shows the architecture of the mentioned solution.

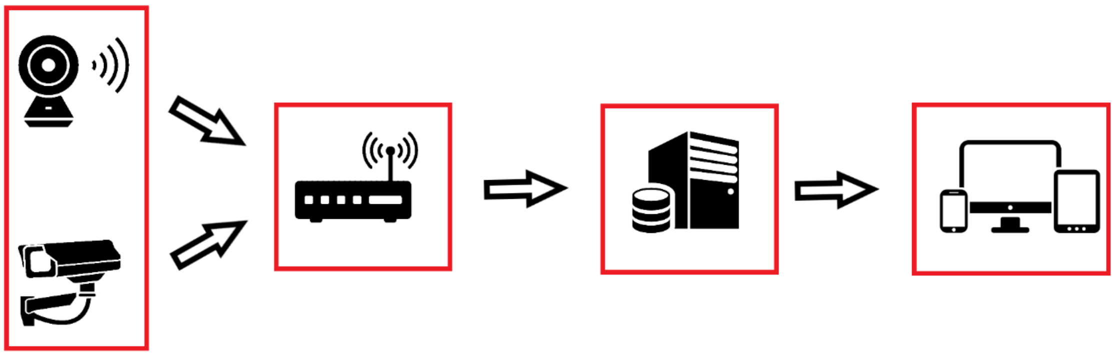


Figure 16: Separate solution for each section

It should be noted that despite having less robustness in comparison to previous architectures, this solution grants a considerable amount of flexibility and hence it would be possible to integrate different types of IP cameras and use various protocols depending on the scenario.

While selecting a proper solution for each section, we should bear in mind that decision on each part effects and limits the number of options for the other parts. I.e. each IP camera supports a specific number of transport protocols; the same is with the streaming media system, which does not support all the broadcasting protocols, while on the other hand, we are limited by delay limitations.

Although by selecting proper multimedia frameworks such as FFmpeg, GStreamer, and UltraGrid, which allow conversion facilities, this dependency can be minimized, though, they introduce some drawbacks including more screen-to-screen delay and complexity.

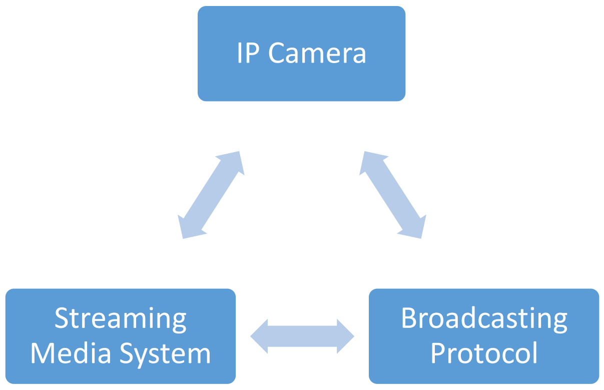


Figure 17: Selection Equilibrium

3-4. Broadcasting Protocol:

3-4-1. Media Streaming Protocols and Standards:

Many protocols have been developed to facilitate the real-time streaming of multimedia content. Communication protocols are rules governing how data is communicated, defining elements like the syntax of the headers and data, authentication and error handling.

- **Real-time Streaming Protocol (RTSP)**

  RTSP was designed to control streaming media servers and mainly acts as a network remote control for one or several synchronized streams of media. Due to the similarities to HTTP, Extension mechanisms of HTTP can also be added to RTSP in most cases. However, HTTP does not have the concept of sessions. Saying that, typically in RTSP a server maintains a session with an identifier for each client but it is agnostic to the underlying transport-level connection and the session is not in any way bound by it. An RTSP session is stateless and both the client and the server can send requests to each other. E.g., it is a request-response protocol that uses a Multipurpose Internet Mail Extensions (MIME) header format, similar to HTTP.

Basically, the RTSP control traffic is sent over TCP, while the media data is often sent over UDP. However, in some cases, the media may be interleaved with the control traffic on the same TCP connection. To match requests and responses, RTSP uses sequence numbers and each media object is identified by an RTSP URL, with the prefix “rtsp:”.

In order to watch the content of the media transmitted by RTSP, we can consider the following scenario: a user downloads a web page that contains a link to a media presentation using the web browser. The web browser then downloads the metafile that contains RTSP URLs for all the multimedia objects in the presentation (e.g., a music clip and streaming text associated with the audio) using the link which points to the metafile hosted by the media server. However, the browser needs to launch the appropriate media player and pass the metafile contents to the player since it is not capable of playing RTSP directly. Finally, the media player parses the metafile and initiates an RTSP connection to the media server.

In RTSP, a session begins when the media player accesses the object and it ends when a TEARDOWN message is sent by the media player. However, there may be several intervening PAUSE and PLAY events. To distinguish among different streams, the protocol relies on session identifiers and as a result, there is not a one-to-one mapping between sessions and RTSP control connections.
Figure 18 describes the diagram of an RTSP session.

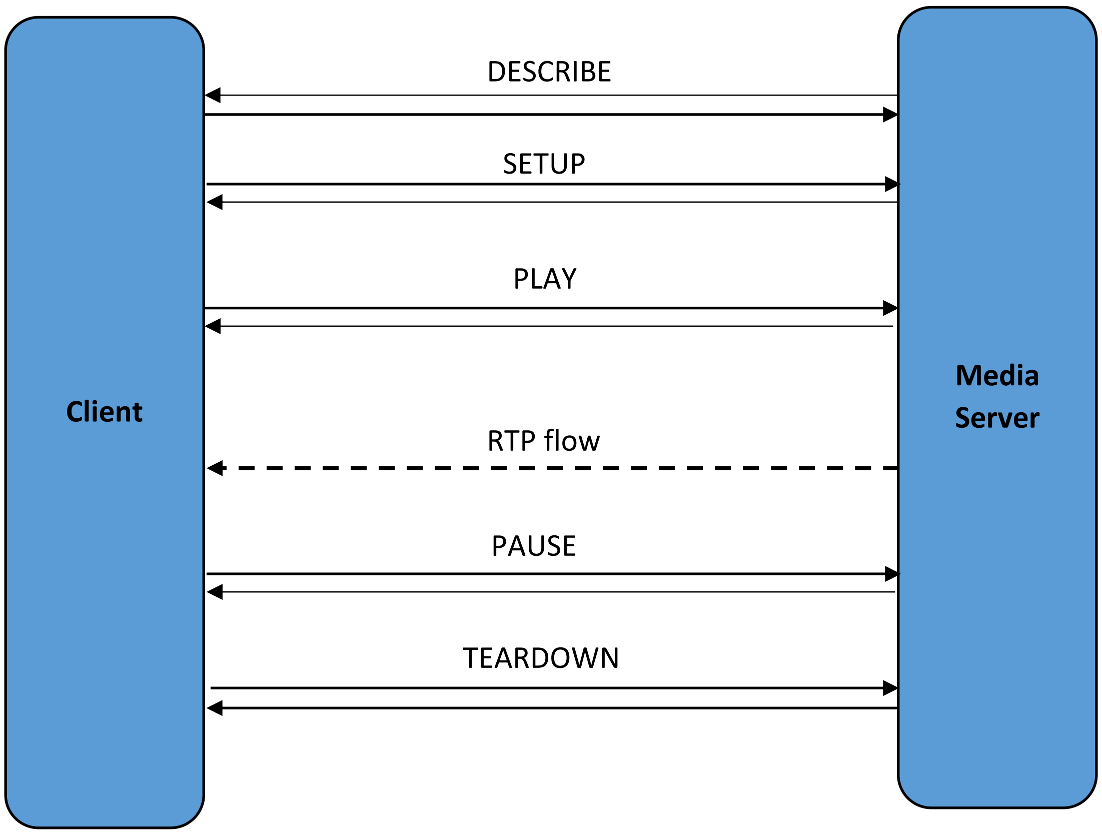


Figure 18: Session establishment and termination in RTSP [9]

- **Real-time Messaging Protocol (RTMP)**

  RTMP provides high-performance transmission of video, audio and data across the internet. It is available to be integrated into developing products and technologies that enable the delivery of video, audio, and data in the open formats compatible with Adobe Flash Player. It is based on TCP protocol and delivers streams by splitting them into chunks.

  The server and the client exchange RTMP messages to communicate with each other. The messages could include audio, video, data, or any other type. The RTMP message has two parts: a message header, which contains message type, length, timestamp and message stream Id, and the message payload, which is the actual data such as audio samples or compressed video data that is contained in the message. Figure 19 describes the diagram of an RTMP session.

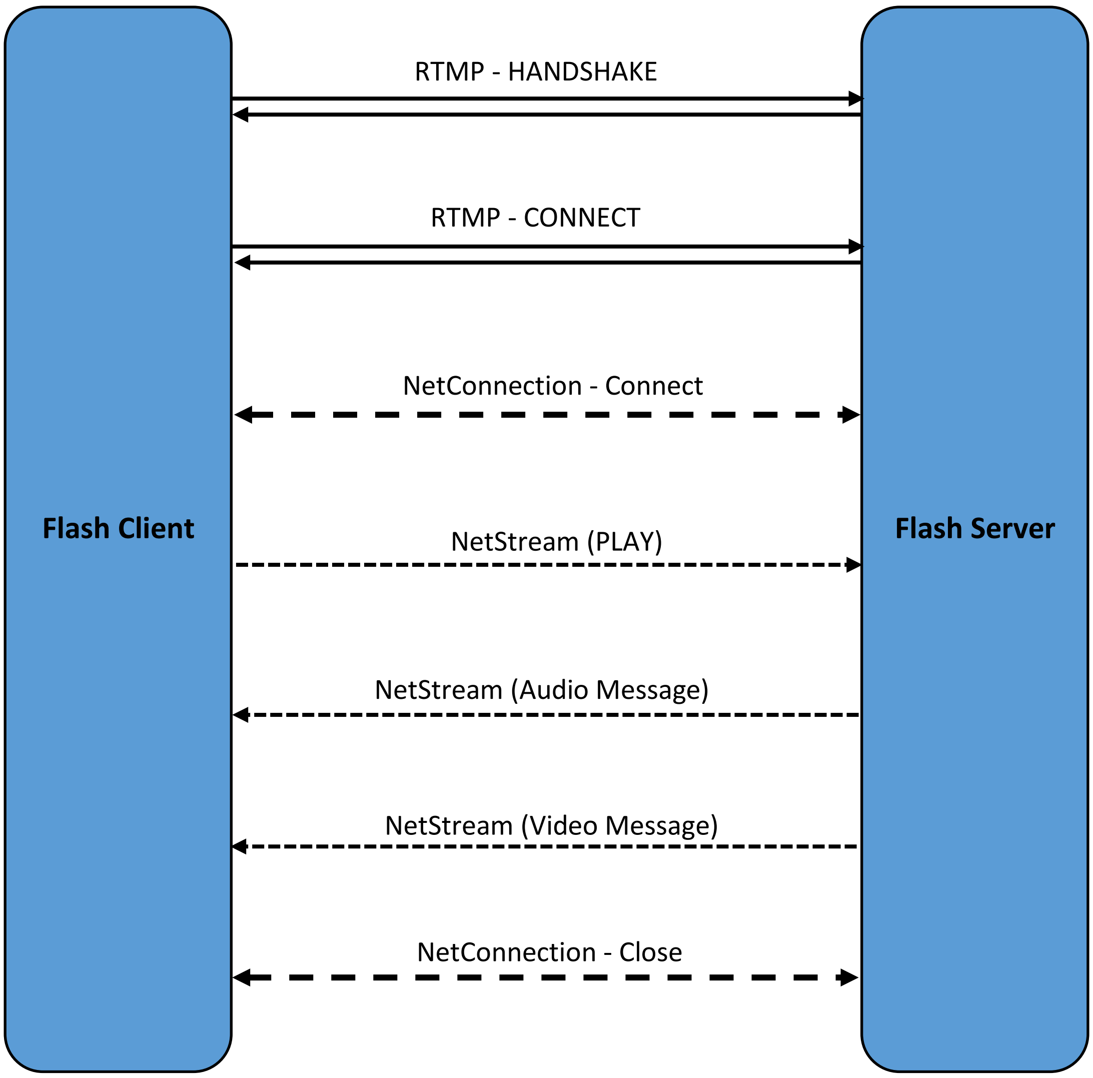


Figure 19: RTMP Streaming Session [10]

- **Dynamic Adaptive Streaming over HTTP (DASH)**

Dynamic adaptive streaming over HTTP is a technology for delivering media content in different bit rates and qualities. DASH is an adaptive bitrate streaming technique in which the control lies with the client. Thus, the client is responsible for requesting the data and changing quality using the HTTP protocol.

DASH is divided into two main parts, the Media Presentation Description (MPD) and the data segments.

The MPD is an XML-file that describes the format and resource identifiers for media segments. A resource identifier is an ordinary URL with or without a byte range, which the client can request with an HTTP GET request. The MPD must supply enough information to provide a streaming service in which the segments can be accessed through the scheme of defined resources.

A DASH client can request different data segments based on the current network conditions. This makes it possible to display a smooth stream with the highest possible quality by always requesting segments with the most suitable bit rate. This means that you can automatically lower the quality of the stream if the network temporarily gets congested.

DASH enables streaming large files by requesting a small duration at a time, which can both be used as a means to stream a recorded video or a live event. A web server is required to provide valid DASH compliant media and an MPD file. The client needs to be able to parse the MPD to request the correct segments.

Figure 20 describes the diagram of DASH communication mechanism.

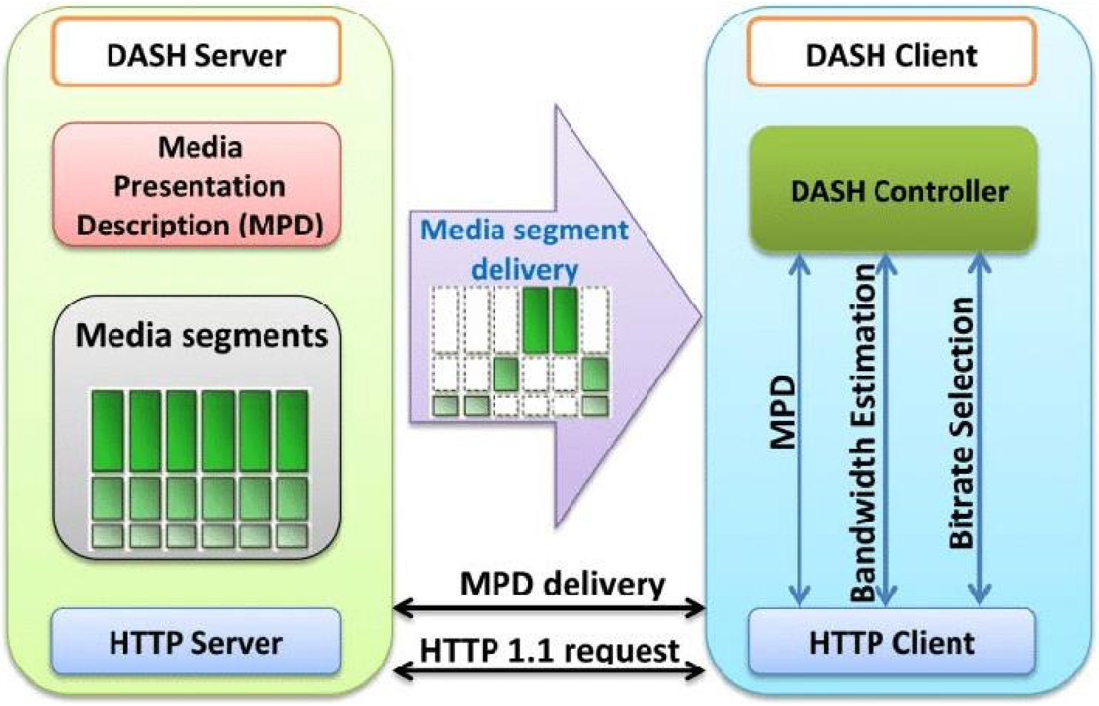


Figure 20: Dash communication mechanism

- **HTTP Live Streaming (HLS)**

  HLS is a technology developed by Apple for streaming audio or video from a web server. It is similar to DASH in the sense that the client is given a manifest file containing URL's to video segments served as small files. The web-server can serve several qualities of the same video and let the client adjust to playing the best-suited quality depending on the current network condition. Currently, only Apple products and Android devices support HLS natively.

  HTTP Live Streaming supports both live broadcasts and prerecorded content (video on demand) and multiple alternate streams at different bit rates and resolutions. HLS allows the client to dynamically switch between streams depending on bandwidth availability. HLS also provides for media encryption and user authentication over HTTPS, allowing publishers to protect their work.

- **Microsoft Smooth Streaming (MSS)**

  Smooth Streaming is a protocol developed by Microsoft in 2008 to satisfy an adaptive bit-rate streaming protocol.

  To deploy Smooth Streaming it is necessary to use Silverlight, which is Microsoft's proprietary developer plugin framework. One strength of Smooth Streaming is enabling adaptive streaming of the media to clients over HTTP.

- **Secure Reliable Transport (SRT)**

  SRT is an open-source streaming protocol that optimizes streaming performance across unpredictable networks using secure streams and easy firewall traversal. The SRT Open Source project, driven by the SRT Alliance, is a collaborative community of industry leaders and developers striving to achieve lower latency internet video transport by continuously improving open-source SRT.

  Before transcoding, SRT decrypts the stream and provides packet loss recovery. Simultaneously, it detects the network performance between the endpoints. These endpoints can be dynamically adjusted for optimal performance and quality of the stream.

- **Web Real-Time Communication (WebRTC)**

  WebRTC is an open-source project that provides real-time communication (RTC) for web browsers and mobile applications using simple APIs. It implements the communication by allowing direct peer-to-peer (P2P) communication and eliminating the requirements to install plugins or downloading other necessary apps.

  Figure 21 shows the working schema of WebRTC. As can be seen in figure 21, the WebRTC architecture consists of web servers and browser clients. With WebRTC, all web application, which supports WebRTC, can communicate with each other through P2P network models. Clients such as PCs, smartphones, and tablets are able to access the Javascript application through browsers. The main advantage of implementing P2P communication paradigm in this concept is that it provides flexibility and scalability to the WebRTC's architecture.

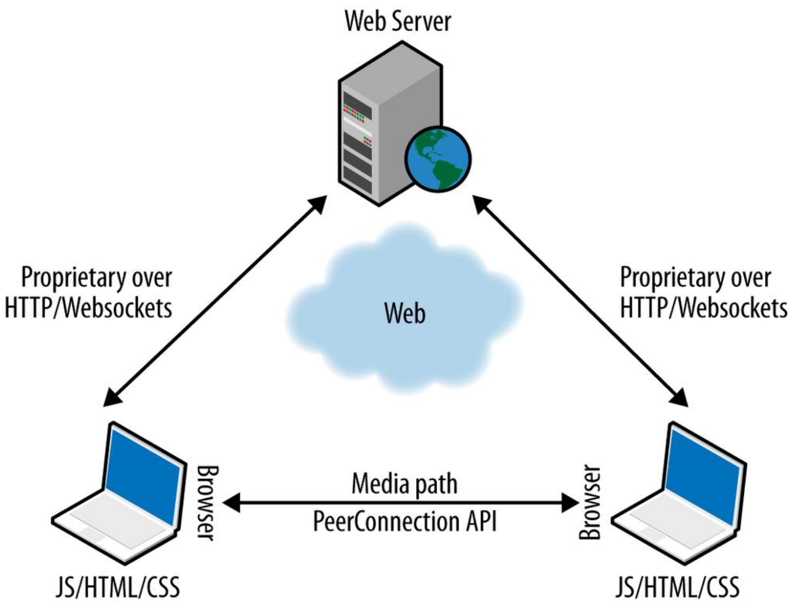


Figure 21: The WebRTC Triangle [7]

The data flow between the web server and the client's browser is controlled by the communication mechanism through signaling messages. Moreover, signaling messages are used to set up and terminate communications.

Figure 22 shows the trapezoid model, which is an extension of the triangle model. The trapezoid model enables servers to connect with one another to finally create a more extended network. Web servers use a signaling protocol such as Session Initiation Protocol (SIP) or other proprietary protocol to manage the communication. However, the media data may not flow directly between the browsers due to the presence of media relays or other elements between them.

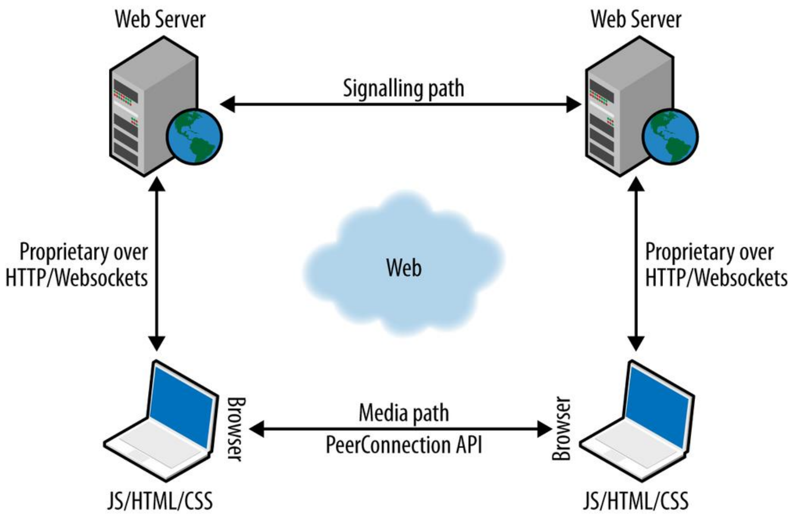


Figure 22: The WebRTC Trapezoid [7]

A WebRTC web application is typically written in HTML and JavaScript as other web applications. The web application interacts with the web browser using the standardized WebRTC API, which allows it to exploit and control the real-time browser function. Thus, WebRTC API must provide a wide set of functions, including connection, encoding/decoding capabilities negotiation, selection and control, media control, firewall and NAT element traversal, etc [7].

Figure 23 is the demonstration of a real-time communication flow of WebRTC in the browser.

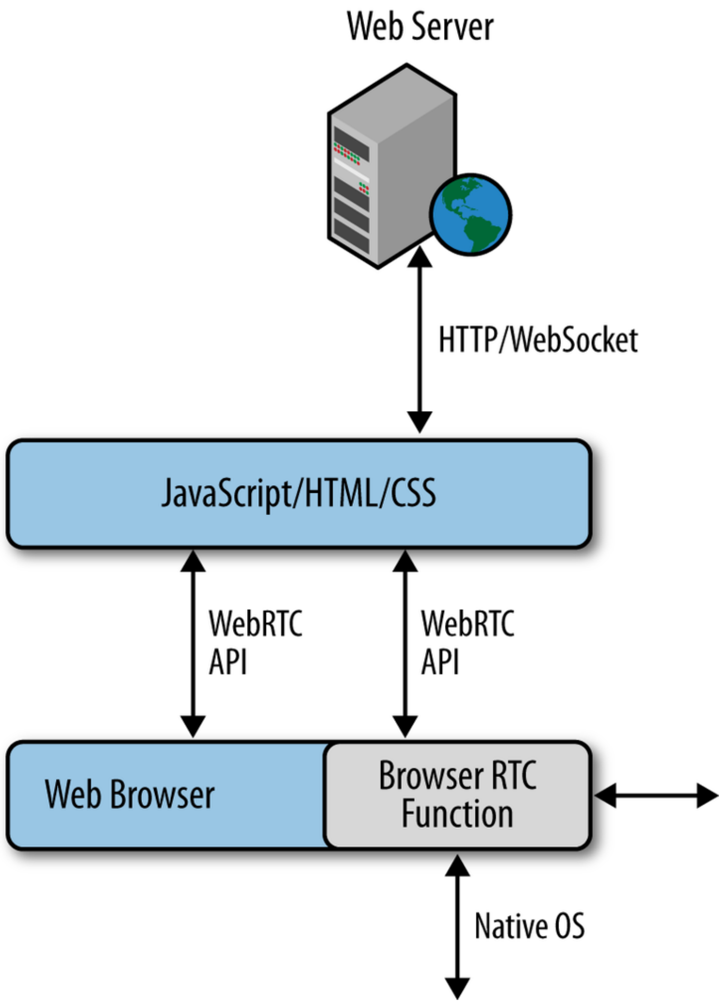


Figure 23: WebRTC Communication Flow in a Browser [7]

Figure 24 provides the general picture of a complete WebRTC call flow composed of channel initiator, channel joiner, and a signaling server, which relays messages between the channel initiator and the channel joiner at channel setup time. The sequence diagram follows the following steps:

1. The initiator connects to the server and the signaling channel is created.
2. After getting the user's consent, the initiator gets access to the user's media.

3. The joiner connects to the server and thus joins the channel;
4. At the moment when the joiner also gets access to the user's local media, a message is sent to the initiator through the server, which triggers the negotiating process.
5. During the negotiation, the initiator and the joiner use the signaling server to exchange network information (For example network reachability, in the form of ICE protocol candidate addresses);
6. When the initiator receives the joiner's response, the negotiation procedure is finished. Next, the two parties switch to peer-to-peer communication is also equipped with a data channel that can be used to exchange messages directly

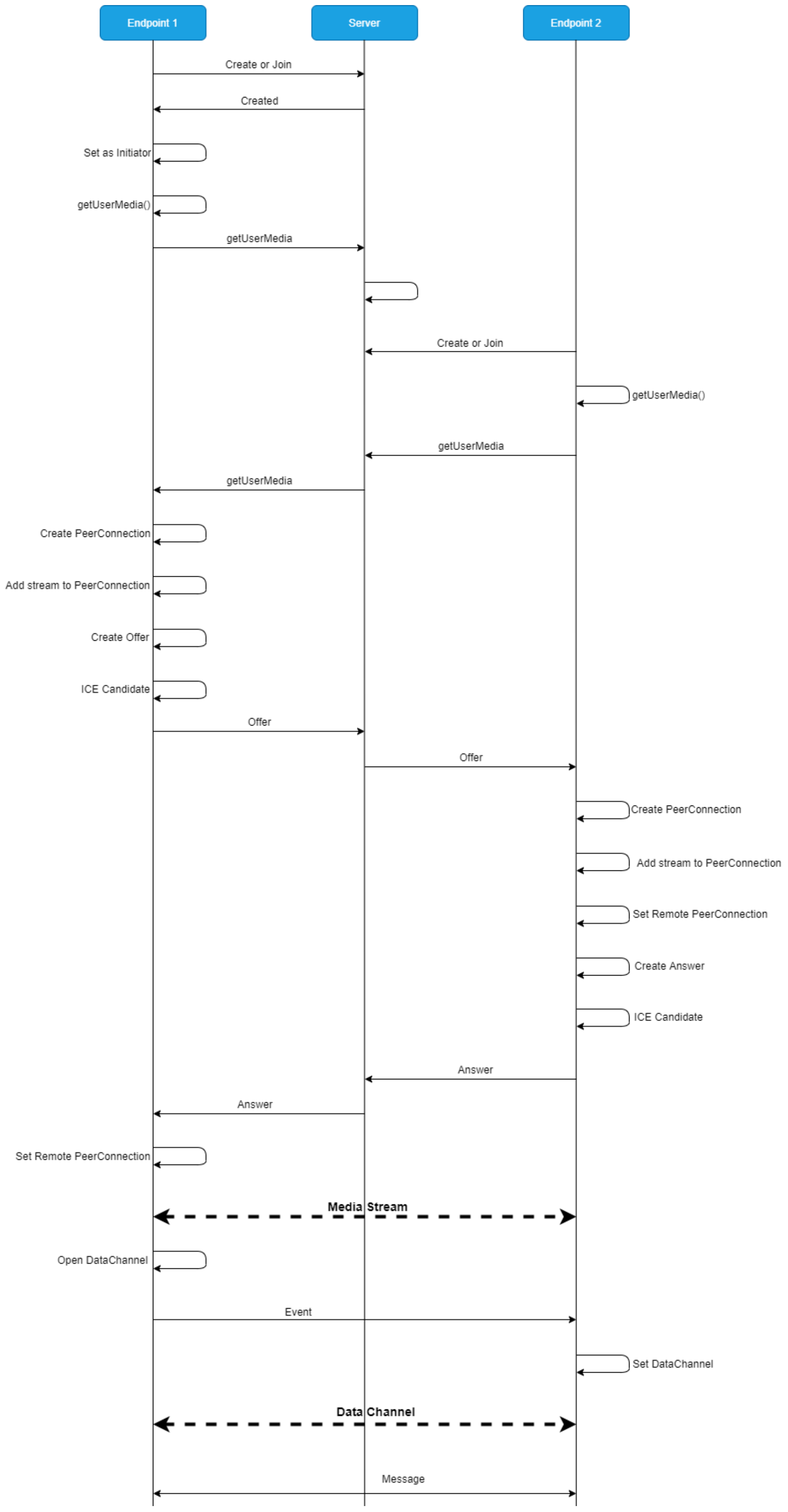


Figure 24: Basic WebRTC call flow [26]

As mentioned, WebRTC consists of several APIs and protocols which work together to accomplish a variety of tasks.

The main three categories of the WebRTC APIs are related to manage the connection, to manage security and telephony. They are listed in detail hereafter.

- **Connection setup and Management:**

  These interfaces are used to set up, initiate, and manage WebRTC connections. This group is composed of APIs representing peer media connections, data channels, and interfaces used when exchanging information on the characteristics and capabilities of each peer in order to select the best possible configuration for a two-way media connection.

API's Name	Description
RTCPeerConnection	It represents a WebRTC connection between the local computer and a remote peer. This interface is mainly used to handle efficient streaming of data between the two peers.
RTCDataChannel	It represents a bi-directional data channel between two peers of a connection.
RTCDataChannelEvent	It represents events that occur while attaching a "RTCDataChannel" to a "RTCPeerConnection".
RTCSessionDescription	It represents the parameters of a session.
RTCSessionDescriptionCallback	The "RTCSessionDescriptionCallback" is passed into the "RTCPeerConnection" object when it requests to create offers or answers.
RTCStatsReport	This interface provides information regarding detailed statistics for a connection; the report can be obtained by calling RTCPeerConnection.getStats(). Details about using WebRTC statistics can be found in WebRTC Statistics API.
RTCIceCandidate	Represents a candidate Internet Connectivity Establishment (ICE) server for establishing an "RTCPeerConnection".
RTCIceTransport	It represents information about an ICE transport.
RTCIceServer	This interface defines how to connect to a single ICE server (such as a STUN or TURN server).
RTCPeerConnectionIceEvent	Represents events that occur in relation to ICE candidates with the target, usually an "RTCPeerConnection". Only one event is of this type: icecandidate.
RTCRtpSender	This interface manages the encoding and transmission of data for a "MediaStreamTrack" on an "RTCPeerConnection".
RTCRtpReceiver	This interface manages the reception and decoding of data for a "MediaStreamTrack" on an "RTCPeerConnection".

RTCRtpContributingSource	Contains information about a given contributing source (CSRC) including the most recent time a packet that the source contributed was played out.
RTCTrackEvent	The interface used to represent a track event, which indicates that an “RTCRtpReceiver” object was added to the “RTCPeerConnection” object, indicating that a new incoming MediaStreamTrack was created and added to the “RTCPeerConnection”.
RTCConfiguration	It is used to provide configuration options for an “RTCPeerConnection”.
RTCSctpTransport	Provides information which describes a Stream Control Transmission Protocol (SCTP) transport and also provides a way to access the underlying Datagram Transport Layer Security (DTLS) transport over which SCTP packets for all of an “RTCPeerConnection”’s data channels are sent and received.
RTCSctpTransportState	It indicates the state of an “RTCSctpTransport” instance.

Table 2: WebRTC APIs regarding connection setup and management [23]

- **Identity and Security:**

  The WebRTC API includes a number of interfaces, which are used to manage security and identity.

API’s Name	Description
RTCIdentityProvider	This interface enables a user agent capable of sending a request that an identity assertion is generated or validated.
RTCIdentityAssertion	It represents the identity of the remote peer of the current connection. In case that no peer has yet been set and verified this interface returns null. It cannot be changed once set.
RTCIdentityProviderRegistrar	It is used for registering an identity provider (idP).
RTCIdentityEvent	It represents the identity assertion generated by an identity provider (idP), usually for an RTCPeerConnection. The event sent with this type is “identityresult”.
RTCIdentityErrorEvent	It represents an error associated with the identity provider (idP), usually for an “RTCPeerConnection”. Typically Two events are sent with this type: “idpassertionerror” and “idpvalidationerror”.
RTCCertificate	It represents the certificate that an “RTCPeerConnection” uses to authenticate.

Table 3: WebRTC APIs regarding identity and security [23]

- **Telephony:**

  These interfaces are related to interactivity with Public-Switched Telephone Networks (PTSNs).

API's Name	Description
RTCDTMFSender	Handles the encoding and transmission of Dual-Tone Multi-Frequency (DTMF) signaling for "RTCPeerConnection".
RTCDTMFToneChangeEvent	It is used by the "tonechange" event to indicate the condition of DTMF (Whether it is tarted or ended). This event does not bubble and is not cancelable.

Table 4: WebRTC APIs regarding telephony [23]

3-5. Streaming Servers

3-5-1. List of Available Streaming Servers

We started to investigate the candidate servers and sorted them based on different factors such as operating system support, being up-to-date, protocol support, container format support, etc.). After investigating whether there is an open-source or free version of them, we tested them by first installing them on an Ubuntu operating system, which is running on an Oracle VM VirtualBox.

The server software that provides and manages the video media content remotely are called Video Streaming Servers. Not all the available video players are capable of working all with the video streaming servers. Instead, they need a video player that is capable of working with the remote servers to buffer and play the video without downloading. This method also helps the publisher to protect their content from piracy.

The following media/streaming servers have been identified and analyzed.

- **Red5 Open-Source Media Server**

  Red5 is a media server available in both open-source and paid licensed versions. It gives support to various kinds of live streaming. This design of this media server is very flexible and can enhance its capabilities using simple plugins. The plugin architecture also allows for customization of any VOD and live streaming scenario.

- **MistServer Open-Source**

  MistServer is a streaming media server that works well in diverse streaming environments even on a Raspberry Pi. The features of the open-source version are real-time stream decoder, Browser-based management interface, Low latency, Multiplatform, Smart HTML5 meta-player, API, Stream meta-data, and Basic analytic integration.

- **Kurento Media server**

  Kurento is an open-source WebRTC media server. This support both audio and video and offers a set of client APIs which allows the developer to create advanced video applications for web and smartphone platforms. Kurento Media Server features include group communications, transcoding, recording, mixing, broadcasting and routing of multimedia flows.

  The Kurento Audio/Video streaming server provides media processing capabilities involving computer vision, video indexing, augmented reality, and speech analysis. It is capable of integrating with third-party media processing algorithms such as speech recognition, face recognition, etc.

- **Janus WebRTC Server**

  Janus is a WebRTC Server developed by Meetecho designed to function as general-purpose one. Thus, its main goal is to set up a WebRTC media communication with a browser and delivering RTP/RTCP and messages. Many features and applications are provided by server-side plugins that browsers can use by connecting to the Janus server. Some example of such plugins are echo tests, conference bridges, media recorders, SIP gateways and the like.

- **Nginx**

  As an HTTP server and reverse proxy, NGINX is known for its high performance, low resource consumption, stability, and simple configuration. What is more, NGINX does not rely on threads to handle requests. Instead, it uses an asynchronous event-driven architecture.

- **Nimble Streamer**

  Nimble Streamer is a free media server developed by Softvelum, LLC. It provides streaming of live and on-demand media to internet-connected devices.

- **Icecast**

  Icecast is a streaming media project is as free software maintained by the Xiph.org Foundation. It was originally developed as a solution to deal with the university's radio station. Seeing that all of the dorms throughout campus had Ethernet and cost for purchasing a third-party solution, such as RealAudio, was prohibitive. Therefore, Icecast was created. With that not only was SMU's broadcast reach increased throughout campus so was the internet radio and broadcast industry changed.

Out of this shortlist, the experimental setup of Kurento, Janus WebRTC server, Red5 and Nginx have been done. Finally, as it will be better explained in Section 4, Kurento looked as the most promising candidate because of the openness and richness of features, provided thanks to a modular architecture, allowing a pipeline of video processing tasks.

3-6. Object Detection

In many scenarios, the processing of the live stream can be required. A common case is that of video processing for object detection.

One of our main motivations for using Kurento was its integration with OpenCV and an interesting media pipelining architecture that allows the use of multiple filters (media processing algorithms) on real-time streams.

3-6-1. Kurento media API

Kurento Media Server (KMS) is based on pluggable media processing capabilities. These capabilities, which are called “Media Elements”, are exposed to application developers, where each Media Element holds a specific media capability, whose details are fully hidden to application developers. Some of the capabilities of Media Elements are recording streams, mixing streams, applying computer vision to streams, augmenting or blending streams, etc.

The capabilities of creating media pipelines are exposed through a simple network interface based on JSON-RPC. However, an abstract client API created which consumes that interface and directly exposes media elements and pipelines as objects that developers can instantiate and manipulate on client programs. It is called the Media API and currently, Java and JavaScript implementations of it is available.

KMS creates media pipelines through a simple network interface based on JSON-RPC. However, an abstract client API created which consumes that interface and directly exposes media elements and pipelines as objects that developers can instantiate and manipulate on client programs. It is called the Media API and currently, Java and JavaScript implementations of it is available.

3-6-1-1. Kurento media element toolbox

As said previously, creating applications with the Media API is like a Lego game. You instantiate the media elements you want and connect them using the topology you prefer. Thus, the application capabilities can be created based on the available media element implementations. Kurento exposes a rich toolbox of media elements ready to be used, which can be classified into three different types:

- **Endpoints**: These are media elements capable of getting media streams into a pipeline (form the network, disk, etc.) or out of a pipeline (to the network, disk, etc.)

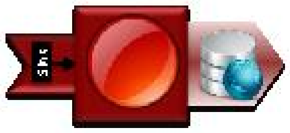
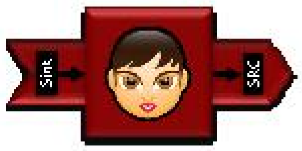
Endpoints		
WebRtcEndpoint		Provides full-duplex capabilities for the exchange of WebRTC media streams. Includes support for ICE, DTLS and SRTP.
RtpEndpoint		Provides full-duplex support for RTP media streams (H.263, H.264 and a bunch of audio codecs are supported). If you connect a WebRtcEnpoint to it, you will have a WebRTC to RTP bridge.
HttpGetEndpoint		Provides support for serving media to browsers supporting WebM in the <video> tag. Connect a WebRtcEndpoint to this and you will have a WebRTC to HTTP bridge.
PlayerEndpoint		Reads media from the file. Connect it with a WebRtcEndpoint and you will have a WebRTC VoD player.
RecorderEndpoint		Stores media into files. Connect a WebRtcEndpoint to it to record your WebRTC session.

Table 5: Kurento Media Server Endpoints

- **Filters**: These media elements are in the path of the media through the pipeline. A filter always receives the stream from another media element, processes the stream somehow, and generates an output stream to be consumed by the next element of the pipeline.

Filters		
FaceOverlayFilter		Detects faces and draws an image overlaid on top of them.

PointerDetectorFilter		Detects movement of a pointer object having a specific colour or shape. Generate events when the object enters into specific regions.
CrowdDetectorFilter		Detects crowds of people on a video stream. Generates events when the crowding is over certain thresholds.
PlateDetectorFilter		Reads (European formatted) car plates using computer vision. Generate events providing the detected plate numbers.
ZBarFilter		Uses computer vision for detecting and reading bar and QR codes. Generates events providing code contents found on the stream.

Table 6: Kurento Media Server Filters

- **Hubs**: These are special media elements designed for group communications. Mixers and video routers are the supported types of hubs.

Hubs		
Composite		Implements a mixer functionality: all input streams getting into this element are mixed into a single output stream with a composite grid layout.
Dispatcher		Implements N to M media router functionality. Each of the M outputs can clone any of the N inputs.

Table 7: Kurento Media Server Hubs

3-6-2. Integration of TensorFlow Object Detection API with Kurento

Though Kurento has many computer vision facilities, the provided tools are not sufficient for us and in our case, object recognition is a more favourable option and this can be achieved by implementing TensorFlow object detection API on top of the WebRTC stream.

3-6-2-1. TensorFlow

Machine learning is a complex discipline. Moreover, although the presence of high requirements and efficiency, implementing acceptable machine learning models less difficult than it used to be, thanks to machine learning frameworks such as Google's TensorFlow.

TensorFlow is an open-source library created by the Google Brain team. It is mostly used for numerical computation and large-scale machine learning. It bundles together an aggregation of neural networking models and algorithms and makes them available to be used by way of a common metaphor.

TensorFlow applications can be run on most targets that are convenient: a local machine, a cluster in the cloud, iOS and Android devices, CPUs or GPUs. If you use Google's own cloud, you can run TensorFlow on Google's custom TensorFlow Processing Unit (TPU) silicon for further acceleration. The resulting models created by TensorFlow, though, can be deployed on most any device where they will be used to serve predictions.

The single biggest benefit TensorFlow provides for machine learning development is an abstraction. Instead of dealing with difficulties of implementing complicated algorithms, or figuring out how to connect the input and the outputs of functions to one another, the developer can focus on the overall logic of the application. [6]

Currently, TensorFlow is probably the most popular Machine Learning frameworks. One of the great things about TensorFlow is that many libraries are actively maintained and updated; TensorFlow Object Detection API is one of them. It is an open-source framework built on top of TensorFlow that facilitates constructing, training and deploying object detection models.

3-7. Microservice-oriented Architecture

In many cases, the user looks for specific information about the camera (the geographical location, installation time, etc.) and the emergency operation. In order to provide this service to the user, it is required to design a backend service.

Microservices are well-known and widely implemented software development technique that is based on structuring an application as a collection of loosely coupled services. The main benefit of implementing an application into various smaller services is that it improves the modularity of the system. E.g. following the microservice topology the application will be easier to understand, develop, test, and as a result, it becomes more resilient. Moreover, it gives us the possibility to integrate our application within other applications. Microservices-based architectures enable continuous delivery and deployment.

However, microservices can be implemented using different programming languages and within different infrastructures. Therefore, the way microservices communicate with each other (synchronous, asynchronous, UI integration) and the protocols used for the communication (REST, messaging, etc.) are the most principal technologies nowadays; since the better performance of these technologies results in significant improvements in the performance of the whole system. In a traditional system, most technology choices like the programming language affect the whole systems. Therefore, the approach to choose technologies is quite different.

In our case, the microservice-oriented architecture is chosen in order to have the flexibility to modify and substitute any part of the architecture depending on the need (in case either their performance is not as expected and does not answer our requirements or a service with better capabilities available).

3-7-1. Node.js

Node.js (also called Node) is a server-side JavaScript environment. It is based on Google's runtime implementation named "V8" engine. Both V8 and Node are implemented using C and C++, so as to have the optimum performance and low memory consumption. However, whereas V8 mainly supports JavaScript in the Web browser, whereas Node aims to support long-running server processes (backend services).

Unlike in most other modern environments, a Node process does not rely on multithreading to support concurrent execution of business logic; it is based on an asynchronous I/O eventing model. Think of the Node server process as a single-threaded daemon that embeds the JavaScript engine to support customization. This is different from most eventing systems for other programming languages, which come in the form of libraries: Node supports the eventing model at the language level.

JavaScript is an excellent fit for this approach because it supports event callbacks. For example, when a browser completely loads a document, a user clicks a button, or an Ajax request is fulfilled, an event triggers a callback. JavaScript's functional nature makes it extremely easy to create anonymous function objects that you can register as event handlers.

3-7-2. RESTful API

A RESTful API is an application program interface (API) that uses HTTP requests to handle and exchange the data using four main methods: GET, PUT, POST and DELETE.
It is mainly preferred to Simple Object Access Protocol (SOAP) technology since REST consumes less bandwidth for exchanging the data, thus making it more suitable for internet usage.

Moreover, REST is a convincing choice for building APIs where users need to connect and interact with cloud services. RESTful APIs are used by such sites as Amazon, Google, LinkedIn and Twitter.

The REST architecture is mainly based on four principles:

- Resource identification through URI
- Uniform interface
- Self-descriptive messages:
- Stateful interactions through hyperlinks

Figure 25 shows a simplified diagram of a RESTful API.

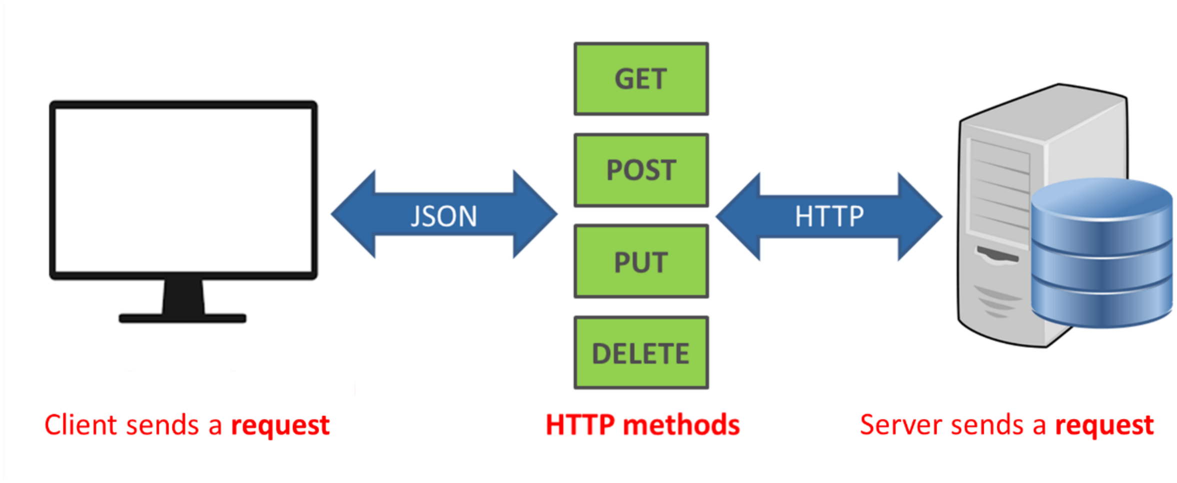


Figure 25: A simplified diagram of a RESTful API

3-7-3. Express.js

Express is a relatively small framework that sits on top of Node.js's web server functionality to simplify its APIs and add helpful new features. Using middleware and routing it facilitates the organization of the application's functionality and adds helpful utilities to Node.js's HTTP objects. Moreover, it facilitates the rendering of dynamic HTML views and it defines an easily implemented extensibility standard.

Generally speaking, Express adds two main features to the Node.js HTTP server:

- Adds several helpful conveniences to Node.js's HTTP server, and simplifies considerably its complexity.
- Provides refactoring one monolithic request handler function into many smaller request handlers that handle only specific bits and pieces.

3-7-4. PostgreSQL Server

PostgreSQL is an open-source general-purpose and object-relational database management system. It was mainly designed to run on UNIX-like platforms, but later it was modified to be portable so that it could run on other popular platforms as well, such as Mac OS X, Solaris, and Windows.

One of the main advantages of PostgreSQL is that it requires low maintenance effort due to its stability. As a result, building applications using PostgreSQL causes low price in comparison with other database management systems. Moreover, in PostgreSQL, you can define your own data types, index types, functional languages, etc. since it is designed to be extensible. Another advantage of PostgreSQL is that it allows the developers to add custom functions developed using different programming languages such as C/C++, Java, etc.

3-7-5. Sequelize.js

Sequelize.js is an Object/Relational Mapper (ORM) which provides easy access to SQL-based databases such as MySQL, MariaDB, SQLite or PostgreSQL. Using its powerful migrations mechanism, it maps database entries to objects and vice versa. Moreover, it is possible to transform an existing database schema into a new version. Moreover, it provides database synchronization mechanisms that can create database structure by specifying the model structure.

Chapter IV: Results and Discussion

4-1. Broadcasting Protocol Assessment and Selection

The selection of the broadcasting protocol is quite important in terms of the performance of the whole system since they are quite diverse in terms of quality and end-to-end delay across devices. While HTTP-based protocols (e.g. HLS, MPEG-DASH) deliver a great user experience (e.g. smoothness, quality, etc.), other protocols such as RTSP and RTMP are a better fit for real-time interactivity. There are also some new and evolving technologies like Secure Reliable Transport (SRT), Common Media Application Format (CMAF), and Web Real-Time Communications (WebRTC) that can be used as alternatives to the mentioned traditional protocols.

Traditional streaming protocols such as RTSP and RTMP support low-latency streaming and work best for streaming to a small audience from a single media server. However, they are not supported on all endpoints (e.g., iOS devices) and there is the problem of browser compatibility. For instance, the only way for playing RTMP inside a browser is using the Flash player, which is totally in contrast with the current trend of playing without being required to install any extensions. Moreover, support of Flash player in Google Chrome is going to be stopped in 2020.

Protocols such as HLS, MPEG-DASH, HDS and HSS are best for streaming to abundant audiences. Using adaptive bitrate streaming, these protocols deliver the best video quality and viewer experience possible, regardless of the connection quality, software, or device. That said, HTTP-based protocols deliver the stream with a considerable latency. Although it is possible to tune HLS and DASH for faster video delivery, the latency of at least five seconds still remains which makes these protocols not suitable for cases where near real-time delivery is crucial such as interactivity.

Concerning the selection of an optimal protocol, a considerable challenge is assessing the various, changing and often contrasting information that can be found in the literature and product datasheets, since streaming over the internet does not happen in a stable condition and the quality of network may vary unpredictably.

Let us take another look at the streaming architecture in figure 26.

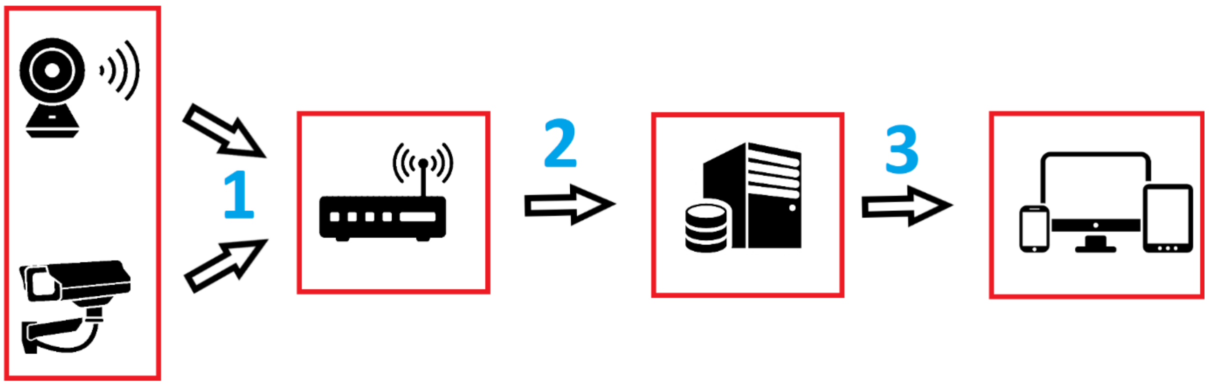


Figure 26: Selected streaming architecture

The uplink transport may be composed of three main connections, where usually the first part is not considered as a separate component, since the router may be integrated with the camera and even not though, the latency is negligible.

At the second phase, we do not have sufficient control over the configuration of used protocols and compressions; as a result, we ought to bind to the specifications of the flow that the camera transmits. Although nowadays IP cameras provide acceptable quality and end-to-end delay using quite fast streaming protocols such as RTSP and RTMP, the streams sent using these protocols cannot be restreamed directly to the HTML5 page due to previously mentioned issues of these protocols.

Accordingly, some alteration must be applied to the flow to make it available and accessible to the final user. Consider a scenario where we are ought to deal with cameras of different types, different characteristics and coping with different network link qualities. As a result, the stream flows reach the server in quite diverse properties. The proposed solution is to fetch the streamflow (usually in RTSP or RTMP) as it is from the camera and then at the server, we evaluate the characteristics of the upcoming flow and act depending on its specifications.

As explained in part 3-3, there are three main cases for the selection of streaming system architecture. In the first case, if the stream's properties including the codec, resolution, and fps are optimal, the server records the stream and at the same time restreams it using the appropriate protocol and quality to experience as least latency as possible.

In the second case, imagine the characteristics of the flow are satisfactory but not perfectly optimized. That is, we are in a condition which conversion may cause more delay than transferring the flow as it is. In this case, the server records the stream in optimal quality and compresses it, before restreaming it with the original quality.

Finally, the last case is when the specification is not satisfactory for streaming at all and we are required to perform conversion before both restreaming and recording.

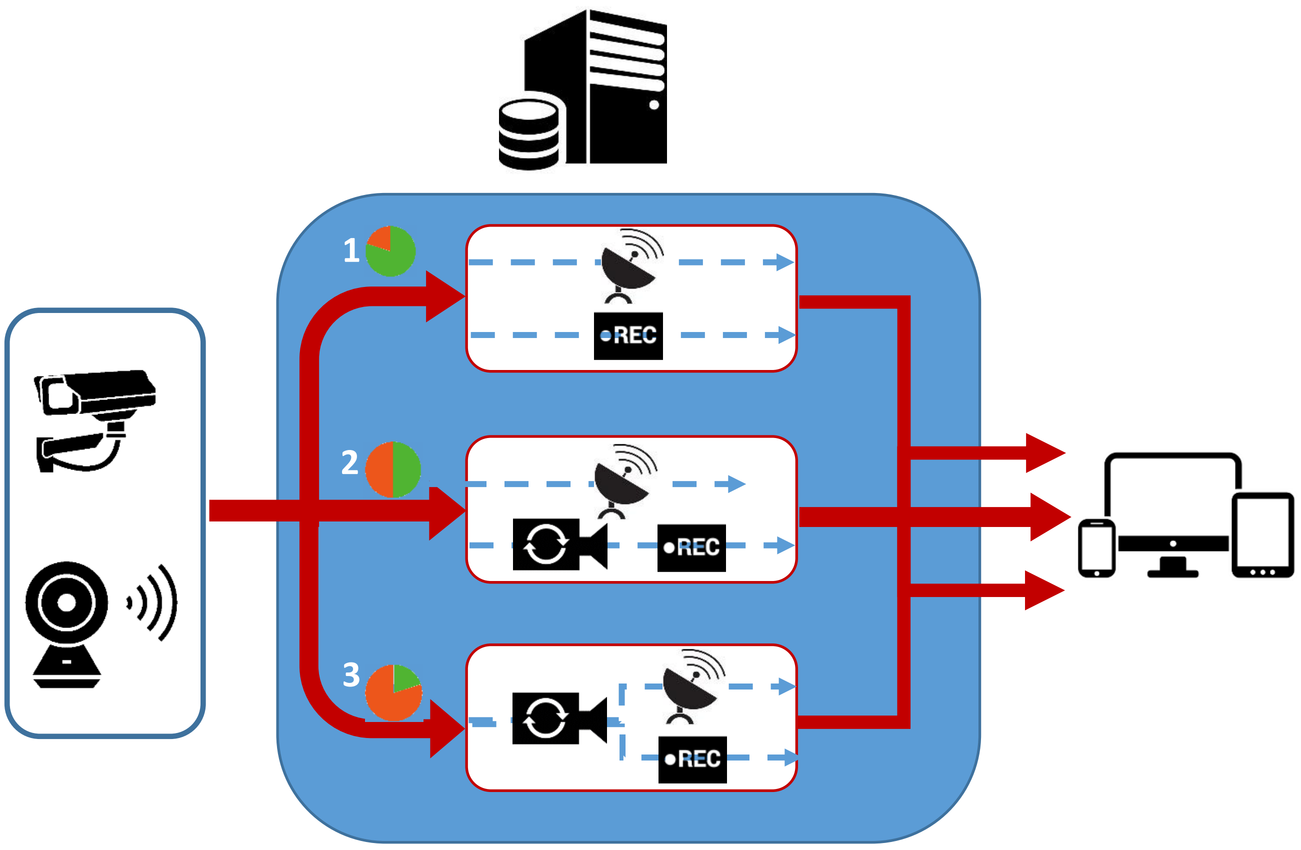


Figure 27: Incoming streaming quality cases

Let us keep in mind that when we talk about undesirable characteristics of the stream, it does not only mean that the quality of the stream is poor or the frame rate is not good enough. Rather, the quality and the frame rate might be so good that restreaming with the original quality will not be possible. Thus, practically, the first and the second case are not likely to happen and usually, we deal with the third case where also adaptive streaming is likely to be used.

As can be seen in figure 26, the only parts of the architecture, which we can adjust, are the server part and the protocol for transmitting from server to the client. However, as mentioned before, due to unavailability of RTMP, RTSP and SRT (where transmitting the flow is feasible but the client has to deal with a lot of complexity to watch the video), only WebRTC, HTTP-based protocols will remain an option.

However, at the end it was decided to select WebRTC due to the fact that it is usage and implementation is being increased every day and its optimized characteristics including being close-to-real time, lightweight, and having adaptive bitrate features implemented inside.

4-2. Streaming Servers Assessment and Selection

For testing the server, flows were taken from an Android app called IP Webcam installed on a smartphone. This app is capable of streaming video on HTTP protocol with different resolutions and frame rates. As explained, the flow is caught by FFmpeg and then is converted to RTMP. The next part is to set up the server. To configure the server, everything should be written to a configuration file.

The following are the description of results followed by installation and testing of Red5 Open-source, Nginx and Kurento Media Server:

- **Red5 Open-source:**

  The first option was the Red5 Open-source version. It supports RTMP directly and RTSP, HLS and WebSocket protocols and via plugins. The main problem of Red5 was weak documentation. Thus, after installing and running some examples, the provided environment did not look quite promising and also, an acceptable tutorial was not found to set up HTML5 streaming using commands and not only the plugins and GUI.

- **Nginx:**

  Secondly, for the means of examining HLS and DASH protocols' quality, I installed Nginx with the RTMP module. Using this combination, it is possible to set up a media server, which takes the streamflow of whatever type they have, converts them with the help of FFmpeg to RTMP if necessary, and then sends them to the end-client using both HLS and DASH. For the playback purpose, Angular-DPlayer is selected due to the capability to play both M38u and MPD files.

  In this case and by using default settings of the server, the delay was about 15 seconds for the HLS protocol that was not acceptable for our scenario.

  After doing some investigation and research on possible tunings, it has been realized that a smaller chunk size serves better for our purpose. However, having too small chunks may cause a considerable amount of overhead for each flow transmission and may affect the smoothness of the playback.

  For the FFmpeg, the latency can be reduced by setting the value for –g option. –g is for Keyframe interval, also known as GOP length. GOP value determines the maximum distance between I-frames. Therefore, very high GOP lengths lead to a more efficient compression but will make following the video slightly more difficult. That said, it is needed to wait for at least one GOP time in order to receive the first feed. Thus, reducing this value can help achieving much less delay value.

  Setting the GOP value equal to 10 has led to having 8 seconds of delay which is quite good for HLS and DASH streaming. However, another problem was the presence of loadings every few seconds (depending on the playlist length) which is not favourable.

  Figure 28 shows the implementation of mentioned cases and it can be seen that the delay is between 8-10 seconds.

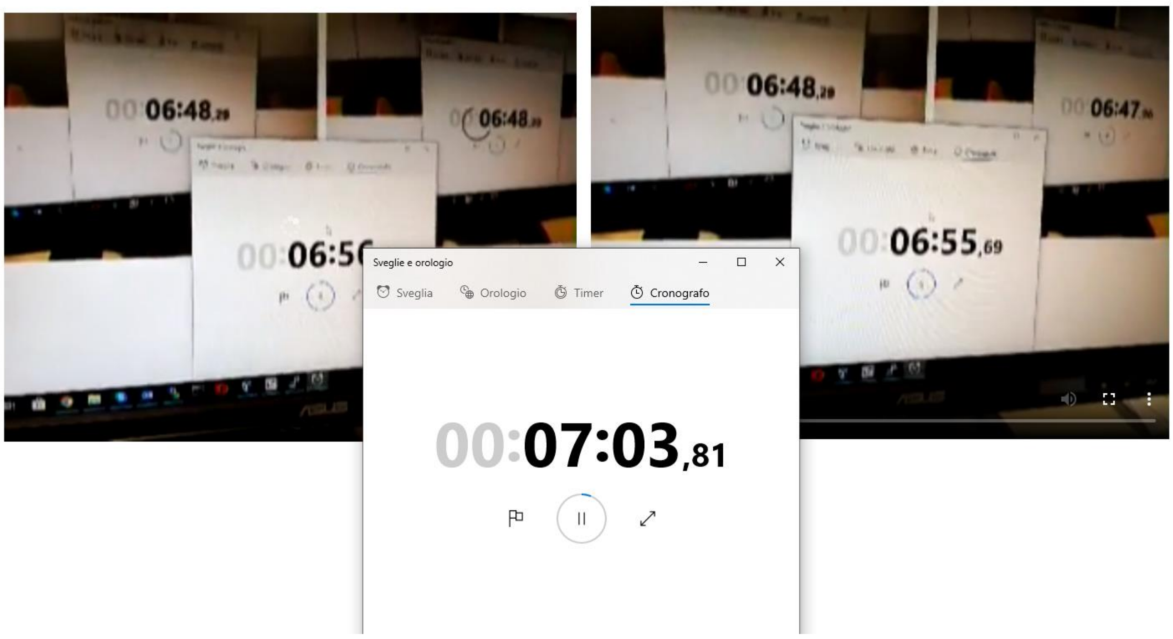


Figure 28: HLS and DASH delay using Nginx RTMP module and FFmpeg

- **Kurento Media Server:**

  WebRTC is a very promising protocol and it was inevitable to examine it as well. Mainly there are two eminent WebRTC-based streaming that are Janus WebRTC Server and Kurento Media Server.

  As mentioned before, Janus WebRTC Server is a general-purpose media server which provides conventional functionalities related to WebRTC. Many features are provided by server-side plugins that browsers can use via Janus. Kurento is quite the same in terms of main functionality and supports both audio and video and provides a set of client APIs, which allows the developer to create video applications. Beside group communications, recording, broadcasting and routing of multimedia flows, its features include transcoding and mixing.

  However, the main advantage of Kurento Media Server over Janus is that it provides media processing capabilities involving computer vision, video indexing, augmented reality, and speech analysis. Kurento is capable of integrating third-party media processing algorithms such as speech recognition, sentiment analysis, face recognition, etc.

As a result, it is decided to use Kurento as our media server.

In order to implement integration with IP camera systems and in general RTSP protocol, a project repository was suggested by an article on Kurento’s website, which facilitates media interoperability.

Hence, for achieving WebRTC interoperability the media gateway requires implementing the media management procedures as shown in Figure 29. Implementing a WebRTC Media Gateway for interoperating with IP cameras in Kurento is possible considering three aspects:

- Kurento Media Server "PlayerEndpoint" supports capturing video streams from different types of sources including RTSP/RTP and HTTP/MJPEG. Thus, it is a favourable solution to capture the media from IP cameras.
- Kurento Media Server "WebRtcEndpoint" supports publishing media streams to WebRTC browsers with adaptive quality.
- Kurento Media Server agnostic media capability performs, transparently for the developer, all the appropriate integrations and adjustments when two incompatible media elements are connected. Thus, only by connecting the PlayerEndpoint source to the WebRtcEndpointsink the H.264/MJPEG to VP8 transcoding shall take place.

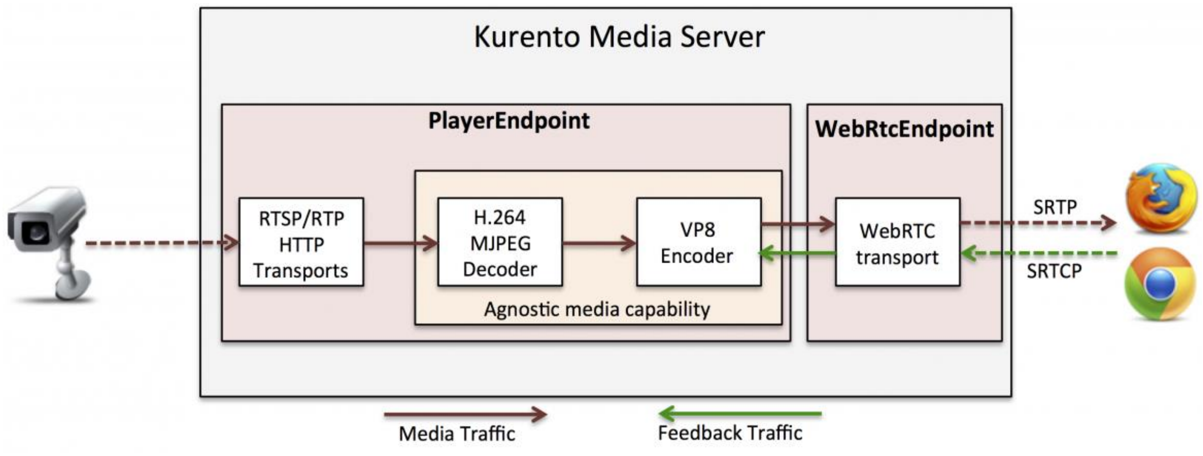


Figure 29: Kurento Media Server implementation of a WebRTC gateway for IP cameras supporting both RTSP/H.264 and HTTP/MJPEG [8]

The delay screen-to-screen delay was less than one second and the quality was quite good, so we have decided to select WebRTC as our main streaming protocol.

In order to implement object detection on top of WebRTC, a sample project was introduced in WebRTCHacks website, which is used in order to be developed for our case [27]

The proposed architecture can be seen in figure 30. Flask, which is a lightweight WSGI web application framework, serves the HTML and JavaScript files for the browser to render. The local video stream is sent to the server and is grabbed by the TensorFlow Object Detection API. Then the TensorFlow Object Detection API sends the objects it sees and their locations in the image to *objDetect.js*. This information is wrapped up in a JSON object so that it can be properly shown with bounding boxes around detected objects and labels of what we see.

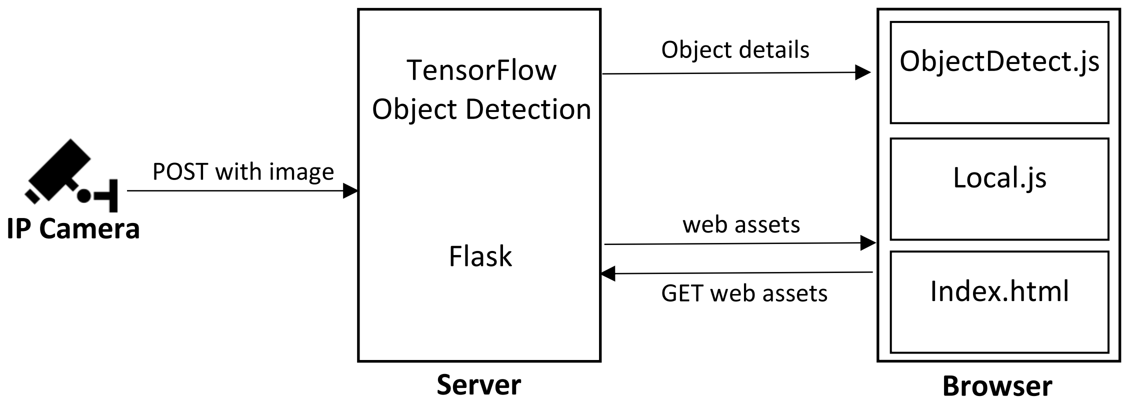


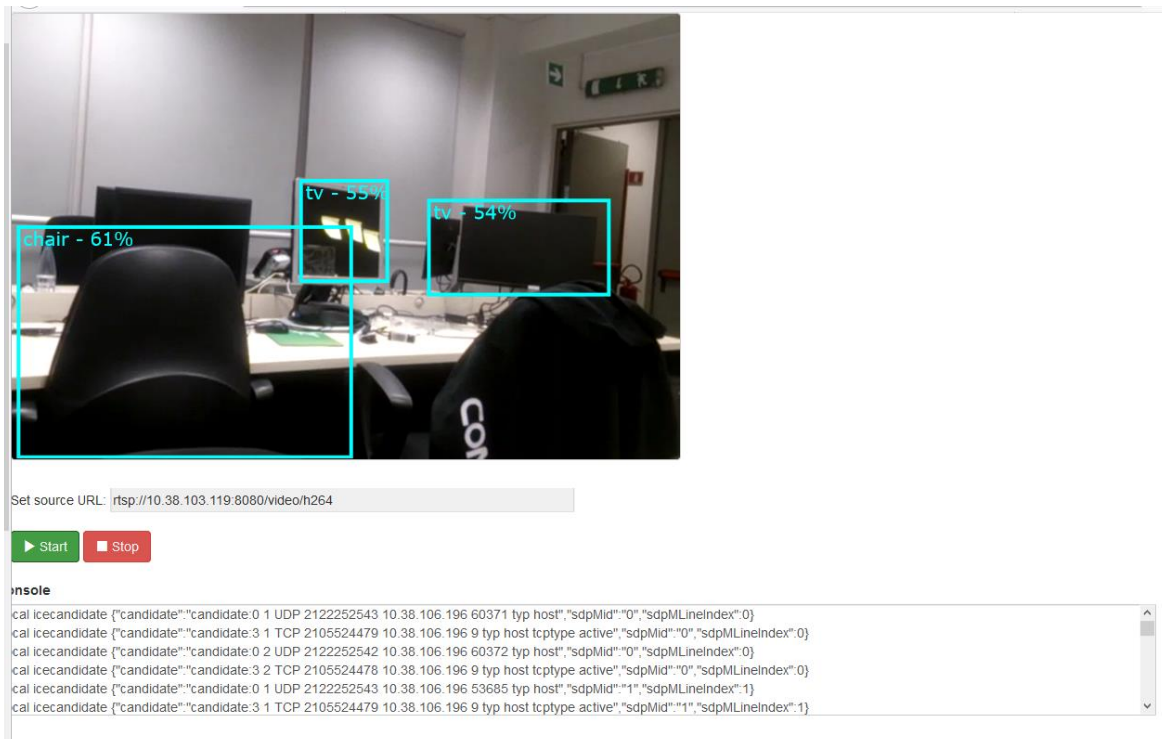
Figure 30: IP Camera and TensorFlow Object Detection API integration proposed architecture

Furthermore, we have changed the used pre-trained object detection model which was “ssd_mobilenet_v1_coco” at the beginning with “ssd_mobilenet_v2_coco” which is a newer version with higher accuracy.

Due to the complexity of the object detection project, it is preferred to move the “Kurento RTSP to WebRTC” project into the mentioned architecture.

Then, the part of the code used for detecting objects and specifying them was imported inside the page, considering the need to perform some customization to the code. The main issue was the fact that the canvas tag’s size must have been equalized with the size of the video element inside the page (in default it is set to the size of the streamed video which is not applicable for the case of this page). Otherwise, there were boxes drawn in places where there is no connection with the labelled object.

Figure 31 shows the implementation of the WebRTC live streaming and object detection using Tensorflow. On the top left, there is the media player where we can see the Reply’s office. It can be seen that the detected objects are indicated using a box, with the label which indicates the name of the object detected, and the certainty of the detection. In the bottom, there is a console located which shows logs regarding the WebRTC communication.



Set source URL: rtsp://10.38.103.119:8080/video/h264

nsole

```
cal icecandidate {"candidate":"candidate:0 1 UDP 2122252543 10.38.106.196 60371 typ host","sdpMid":"0","sdpMLineIndex":0}
cal icecandidate {"candidate":"candidate:3 1 TCP 2105524479 10.38.106.196 9 typ host tcptype active","sdpMid":"0","sdpMLineIndex":0}
cal icecandidate {"candidate":"candidate:0 2 UDP 2122252542 10.38.106.196 60372 typ host","sdpMid":"0","sdpMLineIndex":0}
cal icecandidate {"candidate":"candidate:3 2 TCP 2105524478 10.38.106.196 9 typ host tcptype active","sdpMid":"0","sdpMLineIndex":0}
cal icecandidate {"candidate":"candidate:0 1 UDP 2122252543 10.38.106.196 53685 typ host","sdpMid":"1","sdpMLineIndex":1}
cal icecandidate {"candidate":"candidate:3 1 TCP 2105524479 10.38.106.196 9 typ host tcptype active","sdpMid":"1","sdpMLineIndex":1}
```

Figure 31: Implementation of Object Detection over WebRTC stream provided by Kurento Media Server

4-3. Selected Services Architecture

The selected architecture for the management of the metadata can be seen in figure 32.

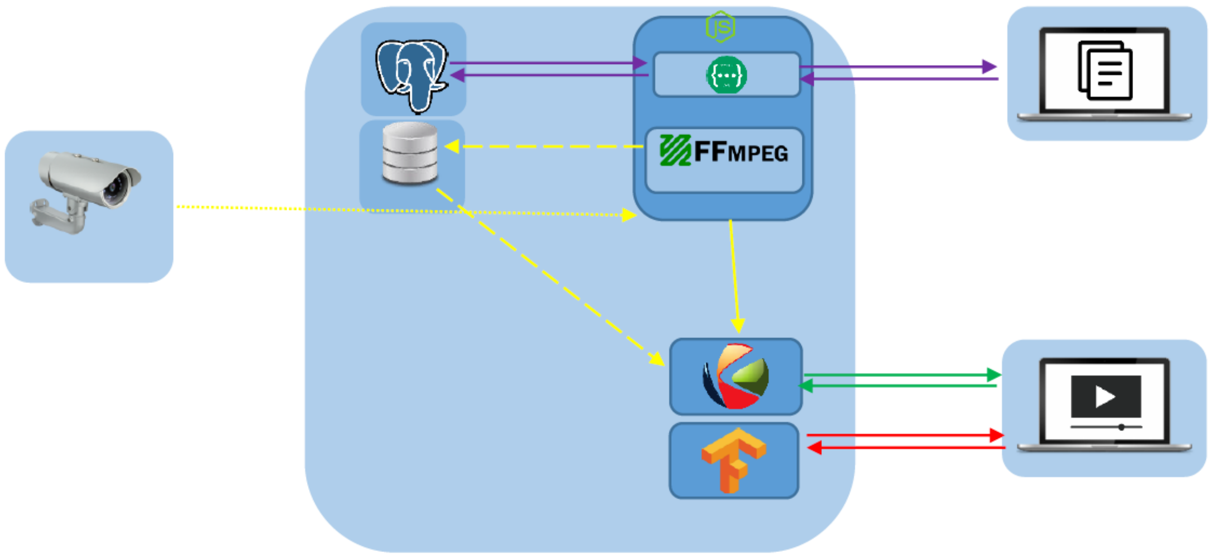


Figure 32: Selected Architecture

First, the IP camera broadcasts the video stream to the server, the stream is grabbed and its characteristics are controlled in order to be matched with the desired requirements both for restreaming and storage. If any of these requirements are not satisfied, it is converted to be adapted with our desired requirements.

Further, the user may ask for a live stream or an already stored stream. This request is handled by Kurento, which broadcasts to the user the video data asked.

Simultaneously TensorFlow grabs the video frames sent by the HTML5 player element and implements object detection on each frame and sends back the result to the HTML5 page, so as to the implemented JavaScript code can draw the box around the detected object.

In case the user is interested in acquiring data, a request is sent from the browser to the server which will be handled by the NodeJS service. The NodeJS service access the PostgreSQL server and send back the requested data to the user through the RESTful API.

Chapter V: Conclusion

In this thesis work, we aimed to design an IP Camera monitoring system to observe the streams coming from IP Cameras in order to have a broader view toward the ongoing emergency management operation and decide accordingly to order the crew or provide the desired resources more quickly.

After evaluating and testing various streaming protocols, it was decided to use WebRTC due to its unique characteristics including being close-to-real time, lightweight, and having adaptive bitrate features implemented inside.

Among the media servers available, Janus WebRTC and Kurento Media Server were the ones that give us the best freedom to use WebRTC. Finally, Kurento Media Server was selected due to its excellent characteristics including the endpoints and filters provided for better handling of video streams.

However, due to lack of integration with IP Cameras since usually they use RTSP, RTMP and HTTP-based streaming protocols, it is required to use FFmpeg in order to encode streams both to prepare them to be used by the Kurento media server and also to adapt them to be in coincidence with our storage requirements such as storage and quality.

Moreover, TensorFlow was adapted in order to be implemented inside the HTML5 webpage and on top of the media player to aid the users with object detection.

Furthermore, a metadata management system is implemented by mainly using Node.js and PostgreSQL in order to provide data management service for the users to store and access data, including the information and location of IP Cameras, information regarding the emergency operations (Both ongoing and previously conducted).

The final task was to design an architecture to provide these services to the client. A micro-service oriented architecture was designed with the aim to provide Real-time and on-demand media streaming, enabling object detection feature, and availability of the metadata management.

References

[1] Heron, Melanie P., National Center for Health Statistics (U.S.). Division of Vital Statistics. (2018). Deaths: leading causes for 2016. *National vital statistics reports; v. 67, no. 6; DHHS publication; no. (PHS) 2018–1120.*

[2] Sim, T.; Wang, D.; Han, Z. (2018). Assessing the Disaster Resilience of Megacities: The Case of Hong Kong. *Sustainability*, *10*, 1137.

[3] Pojani, D.; Stead, D. (2015). Sustainable Urban Transport in the Developing World: Beyond Megacities. *Sustainability*, *7*, 7784-7805.

[4] Panagiotopoulos, P., Barnett, J., Bigdeli, A. and Sams, S. (2016). Social media in emergency management: Twitter as a tool for communicating risks to the public. *Technological Forecasting and Social Change, Volume 111, October 2016, Pages 86-96*

[5] Surakitbanharn C., Ebert D.S. (2018). Improving the Communication of Emergency and Disaster Information Using Visual Analytics. *Nunes I. (eds) Advances in Human Factors and Systems Interaction. AHFE 2017. Advances in Intelligent Systems and Computing, vol 592. Springer, Cham*

[6] What is TensorFlow? The machine learning library explained https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html

[7] Real-Time Communication with WebRTC by Simon Pietro Romano, Salvatore Loreto https://www.oreilly.com/library/view/real-time-communication-with/9781449371869/ch01.html

[8] Kurento.org: Interoperating WebRTC and IP cameras https://www.kurento.org/blog/interoperating-webrtc-and-ip-cameras

[9] Boutabia, Mohammed & CARDENAS, Luis & Afifi, Hossam. (2011). SESSAMO: session mobility for video streaming applications. *International Journal of UbiComp (IJU). 2. 10.5121/iju.2011.2204.*

[10] Juluri, P., Tamarapalli V., Medhi, D. (2016). Measurement of Quality of Experience of Video-on-Demand Services: A Survey. *IEEE Communications Surveys & Tutorials*, *vol. 18, no. 1, pp. 401-418, Firstquarter 2016.*

[11] Tang, Y. & Lin, Z. & Xie, P. & Sung, C.. (2015). Design of urban emergency response system based on webRTC and IMS. 26. 76-86.

[12] Lopez-Fuentes, L., van de Weijer, J., González-Hidalgo, M. et al. (2018). Review on computer vision techniques in emergency situations. *Multimed Tools Appl (2018) 77: 17069. https://doi.org/10.1007/s11042-017-5276-7*

[13] Scholl, Hans & Patin, Beth & Chatfield, Akemi Takeoka. (2012). ICT-Enabled City Government Field Operations: Resiliency During Extreme Events. *Proceedings of the Annual Hawaii International Conference on System Sciences. 2346-2356. 10.1109/HICSS.2012.307.*

[14] Brunetti, P., Croatti, A., Ricci, A. and Viroli, M. (2015). Smart Augmented Fields for Emergency Operations. *Procedia Computer Science Volume 63, 2015, Pages 392-399*

[15] Rahm, Dianne & Reddick, Christopher. (2013). Information and Communication Technology ICT for Emergency Services: A Survey of Texas Emergency Services Districts. *International Journal of E-Politics Volume 4 Issue 3, July 2013 Pages 30-43.*

[16] Media Source Extensions | Revolvy
https://www.revolvy.com/page/Media-Source-Extensions

[17] Akhshabi, Saamer & Begen, Ali & Dovrolis, Constantine. (2011). An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. Proc. ACM Conf. on Multimedia Systems. 157-168. 10.1145/1943552.1943574.

[18] Brunetti, P., Croatti, A., Ricci, A. and Viroli, M. (1998). Smart Augmented Fields for Emergency Operations. Proceedings of the IEEE*, vol. 86, no. 6, pp. 1109-1125, June 1998.*

[19] What is GOP in HDMI SDI H.264 IPTV Streaming Video Encoder
http://bbs.fmuser.com/t/what-is-gop-in-hdmi-sdi-h-264-iptv-streaming-video-encoder/139

[20] S.A.F.E: DESIGN SOSTENIBILE DI SISTEMI DI ARREDO CON FUNZIONE SALVAVITA DURANTE EVENTI SISMICI
http://www.safeproject.it/

[21] Adaptive streaming
https://opentv.nagra.com/player/adaptive-streaming

[22] HEVC (H.265) Vs. AVC (H.264) - What's The Difference?
https://www.boxcast.com/blog/hevc-h.265-vs.-h.264-avc-whats-the-difference

[23] WebRTC API
https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API

[24] Presta, Roberta & Romano, Simon Pietro. (2014). Setting up CLUE telepresence sessions via the WebRTC data channel. *IPTComm '14 Proceedings of the Conference on Principles, Systems and Applications of IP Telecommunications.*

[25] Computer Vision on the Web with WebRTC and TensorFlow
https://webrtchacks.com/webrtc-cv-tensorflow/