A **B. Tech Dissertation REPORT**

On

"LIVE VIDEO SURVEILLANCE THROUGH CCTV/MOBILE"

Submitted for the Partial Fulfilment of degree of

Bachelor of Technology

in

Department of Computer Science and Engineering

(2013-2017)

Submitted by

Parth Gupta	2013UCP1341	Pankaj Jangid	2013UCP1424
Akash Garg	2013UCP1190	Nikhil Bajaj	2013UCP1045
Hanuman Ram	2012UCP1580	Aman Sharma	2013UCP1042

UNDER THE GUIDANCE OF

Dr. Namita Mittal

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Malaviya National Institute of Technology

Jaipur - 302017

2013-2017

Department of Computer Science and Engineering

Malaviya National Institute of Technology Jaipur

CERTIFICATE

This is to certify that the project entitled

"Live Video Surveillance through CCTV/Mobile"

Submitted by

Parth Gupta	2013UCP1341	Pankaj Jangid	2013UCP1424
Akash Garg	2013UCP1190	Nikhil Bajaj	2013UCP1045
Hanuman Ram	2012UCP1580	Aman Sharma	2013UCP1042

is a record of bonafide work carried out by them, in the partial fulfilment of the requirement for the award of Degree of Bachelor of Technology at Malaviya National Institute of Technology, Jaipur under my supervision.

Date: / /

Dr. Namita Mittal
Assistant Professor
Department of Computer Science and Engineering
MNIT Jaipur

DECLARATION

We, Parth Gupta (2013UCP1341), Akash Garg (2013UCP1104), Hanuman Ram (2012UCP1584), Pankaj Jangid (2013UCP1424), Nikhil Bajaj (2013UCP1045) and Aman Sharma (2013UCP1042) declare that this report titled, "LIVE VIDEO SURVEILLANCE THROUGH CCTV/MOBILE" and the work presented in it is an authentic record of our own work. This work is done wholly and mainly towards the fulfilment of the dissertation report at MNIT Jaipur. Wherever we have consulted the published work of others, the source of the work is always clearly attributed. We have acknowledged all sources of help and no part of the report is plagiarized and report does not suffer from any act of plagiarism.

Date: / /

Pankaj Jangid	Parth Gupta
…………..............	…………..............
Nikhil Bajaj	Akash Garg
…………..............	…………..............
Aman Sharma	Hanuman Ram
…………..............	…………..............

ACKNOWLEDGEMENT

We are really thankful to the people who have helped us in creating our final year project. We would like to give our sincere thanks and gratitude to our esteemed supervisor Dr. Namita Mittal for providing her valuable guidance and encouragement to learn new things and enhancing our knowledge in the field of live video surveillance. Her kind cooperation and suggestions throughout the course of this research guided us with an impetus to work, and to successfully complete the project.

A special mention to our friends and classmates, who have constantly supported us with their valuable inputs.

ABSTRACT

Nowadays, the Closed-Circuit Television (CCTV) surveillance system is being utilized in order to keep peace and provide security to people. Video surveillance systems are common in commercial, industrial, and residential environments. There are several defects in the video surveillance system, such as: picture is indistinct, anomalies cannot be identified automatically, a lot of storage spaces are needed to save the surveillance information, and prices remain relatively high. This report proposes a better approach by using the DVR (Digital video recorder) and android app which will capture the live video and will send it to the global server and from where it will be send to the Desktop application and android application .The server we have used here is Node JS server and reduced the load on the server. The live video can be viewed from web browser and even from mobile in real-time.

In this project, we have used the ‘libstreaming’ library for the generating RTSP streams through android. The Node JS based server is used to convert the streams to RTMP, so that it can be played at the viewer side. The Server also maintains a database to store all the necessary details about the centres and other users of the system. The android app accesses the database using secure web services using ‘volley’ library. The viewer side displays the RTMP streams using ‘vitamio’ library in android and JW Player in web portal.

Contents

Chapter 1 Introduction 1
1.1 Motivation 2
1.2 Objective 2
1.3 Deliverables 3
1.4 Related Work 3
1.5 Report Structure 4
Chapter 2 Proposed Approach 5
2.1 System Framework 5
2.2 Real Time Streaming Protocol 6
2.3 Real Time Messaging Protocol 7
2.4 Advanced Video Coding (AVC) or H.264 8
2.5 Digital Video Recorder 10
2.6 Peer-to-Peer v/s Multicast 12
Chapter 3 Media Server 15
3.1 Wowza Media Server 15
3.2 Node JS based Server 16
3.2.1 Node JS 16
3.2.2 Using Node JS based RTSP-RTMP server 17
3.3 ffmpeg 19
3.3.1 Saving Streams 19
3.3.2 Accessing DVR Streams 20
3.4 Database 20

3.4.1 Mysql 20
3.4.2 Database Design 21
3.5 Apache HTTP server 22
3.6 JAVAFX 23
3.6.1 Structure of JAVAFX Application: 24
Chapter 4 Web Applications 25
4.1 Admin Control Panel 25
4.2 Viewer’s Panel 26
4.2.1 JW Player 27
Chapter 5 Android Applications 28
5.1 Accessing Database 28
5.1.1 ‘volley’ library 29
5.1.2 JSON format 30
5.2 Streaming App 31
5.2.1 Authentication and allocation of Room 31
5.2.2 Compression and transmission of RTSP stream 32
5.2.3 ‘libstreaming’ library 32
5.3 Invigilation App 35
5.3.1 ‘vitamio’ library 36
Chapter 6 System Initialization 38
Chapter 7 Performance and Testing 41
Chapter 8 Conclusion and Future Scope 43
8.1 Conclusion 43
8.2 Future Scope 43
Chapter 9 References 45

List of figures

Figure 1: Architecture of proposed approach 5

Figure 2: Structure of Database 21

Figure 3: Server Initialization Application 23

Figure 4: Admin Control Panel...25

Figure 5: Displaying Live Streams 26

Figure 6: Streaming App..31

Figure 7: Invigilation App 35

Figure 8: Xampp Control Panel 38

Figure 9: System Initialization Application..38

Figure 10: Viewers Control Panel 39

List of Abbreviations

CCTV	Closed-Circuit Television
DVR	Digital Video Recorder
RTMP	Real Time Messaging Protocol
RTSP	Real Time Streaming Protocol
RTP	Real-time Transport Protocol
RTCP	Real-time Control Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
AVC	Advanced Video Coding
PVR	Personal Video Recorders
P2P	Peer-to-Peer
JSON	JavaScript Object Notation
RPC	Remote Procedure Call
SDP	Session Description Protocol

Chapter 1

Introduction

Surveillance is the monitoring of behavior, activities, or other changing information for the purpose of influencing, managing, directing, or protecting people. This includes observation from a distance by means of electronic equipment such as closed-circuit television (CCTV) cameras. They are often connected to a recording device or IP network, and may be watched by a security guard or law enforcement officer. Cameras and recording equipment used to be relatively expensive and required human personnel to monitor camera footage, but analysis of footage can been made easier by automated software. The cameras of a mobile phone can also be used for this purpose.

One of the main requirement of camera surveillance connected to an IP network is that the latency is kept minimum. [1]In the network video surveillance context, latency is the time between the instant a frame is captured and the instant that frame is displayed. This is also called end-to-end latency or sensor-to-screen latency. This transporting process includes a long pipeline of steps: capture, compress, transmit, decompress and display of the image. Each stage adds its own share of delay, which together produces the total delay. This end-to-end latency can be divided into 3 major stages impacting the total system latency:

1. Latency introduced by the camera (capture, compression, buffer and audio latency),
2. Latency introduced by the network (transmission protocol and bandwidth latency),
3. Latency introduced by the receiver side (client buffer, decompression and display latency).

Each of these latencies needs to be considered when designing the video solution in order to meet the latency goal of the video surveillance system. Our project is to create a Video Surveillance System to meet the requirements of surveillance for Rajasthan Electronic and Instruments Limited (REIL).

1.1 Motivation

Currently companies like REIL use the products and services provided by companies like Vmukti for monitoring remote examination centres using Live Video Surveillance. The main motivation of this project is to develop an open source end-to-end system for live invigilation at the examination centres in the rural areas. Open source refers to any program whose source code is made available for use or modification as users or other developers see fit. Open source software is usually developed as a public collaboration and made freely available.

Secondly, the invigilation at examination centres by individuals cannot be trusted upon, especially in rural areas. An exam invigilator, exam proctor or exam supervisor is someone who is appointed by the examination board and services for maintaining the proper conduct of a particular examination in accordance with the exam regulations. It is the duty of the exam invigilator to watch the examination candidates to prevent cheating during the examination. Exam Invigilators are appointed to a position of trust and they should possess the quality of integrity and vigilance to conduct the examinations in exact accordance with the examination board's instructions. But recently multiple cases are found involving bribes taken by invigilators to allow cheating among students or even worse, assisting students in cheating. Our project provides a solution to this problem as it is based on central invigilation approach, avoiding any contact between the invigilators and students.

1.2 Objective

The objectives of our project reflect the requirements of the system required by REIL in replacement for its current system. These include:

- To have an end-to-end system developed for live video surveillance.
- The tools and libraries used in the system must all be open source or self-developed which are freely available to all.
- The system must be able to send videos using minimum bandwidth by efficiently compressing video without compromising the quality of video being sent.

1.3 Deliverables

The deliverables for this project are:

- Media Server to handle the incoming streams and generate output streams, along with other functionalities like saving streams in the server storage.
- Use of compression algorithm to send streams with minimum bandwidth.
- Android Application to send streams from an android device.
- Android Application to view the streams generated by the server.
- Web based portal to view all the streams.
- Web based portal to manage the system through an admin.
- GUI to start the system.

1.4 Related Work

There are few already existing applications for the same and in this section we are going to look at the pros and cons of few of them.

1. **Purple Streams**

PurpleLive is an on-the-ground Live Streaming and broadcast activation service that provides high speed, high quality streaming over a regular Internet connection. Seamless, and Live, as it happens, when it happens!

2. **Third-Party**

This third-party company stands as one of the pioneer forces involved in the transformation of virtual communication with cloud based video streaming, surveillance and meeting.

With interactive live video streaming and bandwidth aggregation solutions certified by Intel and having bagged numerous awards from reputed international organizations like Nasscom, Red Herring & ISBA, we are a proud service provider to IIM-Ahmedabad, Vibrant Gujarat, Nasscom, Both Central as well as State Govt. Bodies, Large MNCs, Corporates, Private Sector Banks etc.

It is also not very successful with 2G data as it is with other sources also the service not provided everywhere it is not very successful in remote areas. Also, regarding the storage of videos, the cloud to be used is fixed and one cannot use his/her own cloud.

3. atHome

AtHome Video Streamer turns more than 10 million devices worldwide into an IP Camera, acting as baby monitor, pet camera, nanny cameras, elder care etc. All platforms are supported, including computers, smartphones and tablet PC.

This free app supports full of professional functions, like schedule recording, alarm recording, email and push notification, remote capture, two-way talk, video preview, split-screen viewing, video capture, cloud storage of alarm video etc. Video streaming with powerful encryption and P2P transfer technology makes it more secure! AtHome is the best choice for your home video surveillance.

But there were some problems associated with this application. Firstly, we can only save the video for 10 secs which was very less in comparison to our need. Secondly, this application requires its own special cameras for use which is athome camera. The biggest problem among all these were that a user was unable to login whenever he/she wanted that reports the bug in the system itself.

1.5 Report Structure

The rest of the report is structured as follows: Chapter 2 gives an overview of the proposed solution for the project. It explains the different components in the project along with the different protocols like RTSP, RTMP and other technologies like H.264 are explained. It also explains the basic functioning of a DVR. Chapter 3, explains the different aspects of the server used in the project like Node JS, ffmpeg, the database, etc. It also gives a detailed explanation of each aspect and its functioning in the project. Chapter 4, describes the implementation of the web applications in the project. Chapter 5, describes the android application used along with the different libraries used for the implementation of the desired functionalities.

Chapter 6 gives a descriptive step-wise approach for starting the system first time. Chapter 7 throws light on the interaction we had with REIL, and the different environments in which the project is tested. Chapter 8 concludes the project, and describes the future course of action for the project. Chapter 9 lists all the references used in the report.

Chapter 2

Proposed Approach

2.1 System Framework

Our proposed approach involves breaking down the system into multiple components based on its functionality.

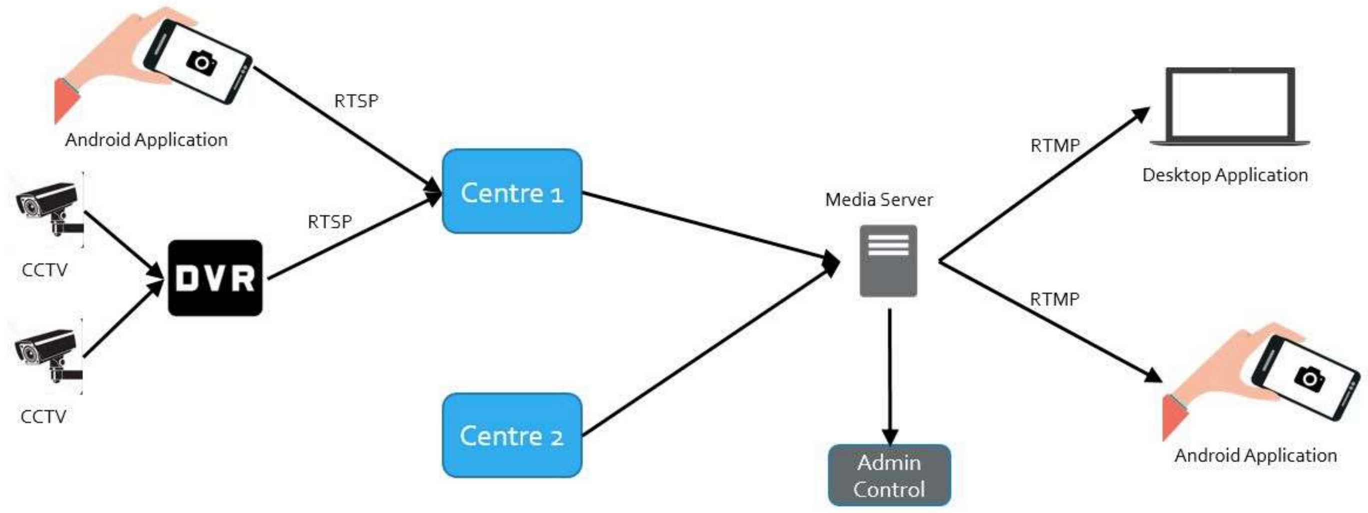


Figure 1: Architecture of proposed approach

The major components of the approach include:

- Media Server
- Desktop Applications
- Android Applications

Our approach involves sending the videos streams using RTSP protocol received from either DVR or the android device. These streams reach the media server where they are forwarded to the viewing end applications using RTMP protocol. The use of RTSP protocol is essential to complete the requirement of live streaming, and the RTMP protocol is essential, so that the stream can be played at the user end. A admin control panel is provided to accomplish the tasks of managing the centres and viewers in the system.

2.2 Real Time Streaming Protocol

The Real Time Streaming Protocol (RTSP) is a network control protocol designed for use in entertainment and communications systems to control streaming media servers. [2]The protocol is used for establishing and controlling media sessions between end points. Clients of media servers issue VCR-style commands, such as play, record and pause, to facilitate real-time control of the media streaming from the server to a client (Video on Demand) or from a client to the server (Voice Recording).

The transmission of streaming data itself is not a task of RTSP. Most RTSP servers use the Real-time Transport Protocol (RTP) in conjunction with Real-time Control Protocol (RTCP) for media stream delivery. While similar in some ways to HTTP, RTSP defines control sequences useful in controlling multimedia playback.

While HTTP is stateless, RTSP has state; an identifier is used when needed to track concurrent sessions. Like HTTP, RTSP uses TCP to maintain an end-to-end connection and, while most RTSP control messages are sent by the client to the server, some commands travel in the other direction (i.e. from server to client).

RTSP Operation

The RTSP protocol is very similar in structure and specifically syntax to HTTP. Both use the same URL structure to describe an object, with RTSP using the rtsp:// scheme rather than the http://. RTSP, however, introduces a number of additional headers (such as DESCRIBE, SETUP, and PLAY) and also allows data transport out-of-band and over a different protocol, such as RTP described earlier. The best way to understand how the components described previously work together to deliver an audio/video stream is to look at an example. The basic steps involved in the process are as follows [2]:

1. The client establishes a TCP connection to the servers, typically on TCP port 554, the well-known port for RTSP.
2. The client will then commence issuing a series of RTSP header commands that have a similar format to HTTP, each of which is acknowledged by the server. Within these RTSP commands, the client will describe to the server details of the session requirements, such as the version of RTSP it supports, the transport to be used for the data flow, and any associated UDP or TCP port information. This information is passed using the DESCRIBE and SETUP headers and is

augmented on the server response with a Session ID that the client, and any transitory proxy devices, can use to identify the stream in further exchanges.

3. Once the negotiation of transport parameters has been completed, the client will issue a PLAY command to instruct the server to commence delivery of the RTP data stream.
4. Once the client decides to close the stream, a TEARDOWN command is issued along with the Session ID instructing the server to cease the RTP delivery associated with that ID.

2.3 Real Time Messaging Protocol

Real-Time Messaging Protocol (RTMP) was initially a proprietary protocol developed by Macromedia for streaming audio, video and data over the Internet, between a Flash player and a server. Macromedia is now owned by Adobe, which has released an incomplete version of the specification of the protocol for public use.

RTMP operation

RTMP is a TCP-based protocol which maintains persistent connections and allows low-latency communication. [3] To deliver streams smoothly and transmit as much information as possible, it splits streams into fragments, and their size is negotiated dynamically between the client and server. Sometimes, it is kept unchanged; the default fragment sizes are 64 bytes for audio data, and 128 bytes for video data and most other data types. Fragments from different streams may then be interleaved, and multiplexed over a single connection. With longer data chunks, the protocol thus carries only a one-byte header per fragment, so incurring very little overhead. However, in practice, individual fragments are not typically interleaved. Instead, the interleaving and multiplexing is done at the packet level, with RTMP packets across several different active channels being interleaved in such a way as to ensure that each channel meets its bandwidth, latency, and other quality-of-service requirements. Packets interleaved in this fashion are treated as indivisible, and are not interleaved on the fragment level.

The RTMP defines several virtual channels on which packets may be sent and received, and which operate independently of each other. For example, there is a channel for handling RPC requests and responses, a channel for video stream data, a channel for audio stream data, a channel for out-of-band control messages (fragment size negotiation, etc.), and so on. During a typical RTMP session, several channels may be active simultaneously at any given time. When RTMP data is encoded, a packet header is generated. The packet header specifies, amongst other matters, the ID of the channel on which it is to be sent, a timestamp of when it was generated (if necessary), and the size of the packet's payload. This header is then followed by the actual payload content of the packet, which is fragmented according to the currently agreed-upon fragment size before it is sent over the connection. The packet header itself is never fragmented, and its size does not count towards the data in the packet's first fragment. In other words, only the actual packet payload (the media data) is subject to fragmentation.

At a higher level, the RTMP encapsulates MP3 or AAC audio and FLV1 video multimedia streams, and can make remote procedure calls (RPCs) using the Action Message Format. Any RPC services required are made asynchronously, using a single client/server request/response model, such that real-time communication is not required.

2.4 Advanced Video Coding (AVC) or H.264

H.264 or MPEG-4 Part 10, Advanced Video Coding (MPEG-4 AVC) is a block-oriented motion-compensation-based video compression standard. [4] As of 2014 it is one of the most commonly used formats for the recording, compression, and distribution of video content.

The intent of the H.264/AVC project was to create a standard capable of providing good video quality at substantially lower bit rates than previous standards (i.e., half or less the bit rate of MPEG-2, H.263, or MPEG-4 Part 2), without increasing the complexity of design so much that it would be impractical or excessively expensive to implement. An additional goal was to provide enough flexibility to allow the standard to be applied to a wide variety of applications on a wide variety of networks and systems, including low and high bit rates, low and high resolution video, broadcast, DVD storage, RTP/IP packet networks, and ITU-T multimedia telephony systems. The H.264 standard can be viewed as a "family of standards" composed of a number of different profiles. A specific decoder decodes at least one, but not necessarily all profiles. The decoder specification describes which profiles can be decoded. H.264 is typically used for lossy compression, although it is also possible to create truly lossless-coded regions within lossy-coded pictures or to support rare use cases for which the entire encoding is lossless.

H.264 was developed by the ITU-T Video Coding Experts Group (VCEG) together with the ISO/IEC JTC1 Moving Picture Experts Group (MPEG). The project partnership effort is known as the Joint Video Team (JVT). [4] The ITU-T H.264 standard and the ISO/IEC MPEG-4 AVC standard (formally, ISO/IEC 14496-10 – MPEG-4 Part 10, Advanced Video Coding) are jointly maintained so that they have identical technical content. The final drafting work on the first version of the standard was completed in May 2003, and various extensions of its capabilities have been added in subsequent editions. High Efficiency Video Coding (HEVC), a.k.a. H.265 and MPEG-H Part 2 is a successor to H.264/MPEG-4 AVC developed by the same Organization, while earlier standards are still in common use.

H.264 is perhaps best known as being one of the video encoding standards for Blu-ray Discs; all Blu-ray Disc players must be able to decode H.264. It is also widely used by streaming internet sources, such as videos from Vimeo, YouTube, and the iTunes Store, web software such as the Adobe Flash Player and Microsoft Silverlight, and also various HDTV broadcasts over terrestrial (Advanced Television Systems Committee standards, ISDB-T, DVB-T or DVB-T2), cable (DVB-C), and satellite (DVB-S and DVB-S2).

H.264/AVC/MPEG-4 Part 10 contains a number of new features that allow it to compress video much more efficiently than older standards and to provide more flexibility for application to a wide variety of network environments. In particular, some such key features include [4]:

1. Using previously encoded pictures as references in a much more flexible way than in past standards, allowing up to 16 reference frames (or 32 reference fields, in the case of interlaced encoding) to be used in some cases. In profiles that support non-IDR frames, most levels specify that sufficient buffering should be available to allow for at least 4 or 5 reference frames at maximum resolution. This is in contrast to prior standards, where the limit was typically one; or, in the case of conventional "B pictures" (B-frames), two. This particular feature usually allows modest improvements in bit rate and quality in most scenes. But in certain types of scenes, such as those with repetitive motion or back-and-forth scene cuts or uncovered background areas, it allows a significant reduction in bit rate while maintaining clarity.
2. Variable block-size motion compensation (VBSMC) with block sizes as large as 16×16 and as small as 4×4, enabling precise segmentation of moving regions. The supported luma prediction block sizes include 16×16, 16×8,

8×16, 8×8, 8×4, 4×8, and 4×4, many of which can be used together in a single macroblock. Chroma prediction block sizes are correspondingly smaller according to the chroma subsampling in use.

3. The ability to use multiple motion vectors per macroblock (one or two per partition) with a maximum of 32 in the case of a B macroblock constructed of 16 4×4 partitions. The motion vectors for each 8×8 or larger partition region can point to different reference pictures.
4. The ability to use any macroblock type in B-frames, including I-macroblocks, resulting in much more efficient encoding when using B-frames. This feature was notably left out from MPEG-4 ASP.
5. Six-tap filtering for derivation of half-pel luma sample predictions, for sharper subpixel motion-compensation. Quarter-pixel motion is derived by linear interpolation of the halfpel values, to save processing power.
6. Quarter-pixel precision for motion compensation, enabling precise description of the displacements of moving areas. For chroma the resolution is typically halved both vertically and horizontally (see 4:2:0) therefore the motion compensation of chroma uses one-eighth chroma pixel grid units.
7. Weighted prediction, allowing an encoder to specify the use of a scaling and offset when performing motion compensation, and providing a significant benefit in performance in special cases—such as fade-to-black, fade-in, and cross-fade transitions. This includes implicit weighted prediction for B-frames, and explicit weighted prediction for P-frames.

2.5 Digital Video Recorder

A digital video recorder (DVR) is an electronic device that records video in a digital format to a disk drive, USB flash drive, SD memory card, SSD or other local or networked mass storage device. The term includes set-top boxes with direct to disk recording, portable media players and TV gateways with recording capability, and digital camcorders. Personal computers are often connected to video capture devices and used as DVRs; in such cases the application software used to record video is an integral part of the DVR. Many DVRs are classified as consumer electronic devices; such devices are sometimes referred to as personal video recorders (PVRs).

DVR FILE FORMATS

DVRs can usually record and play H.264, MPEG-4 Part 2, MPEG-2 .mpg, MPEG-2 .TS, VOB and ISO images video, with MP3 and AC3 audio tracks. [5] They can also display images (JPEG and PNG) and play music files (MP3 and Ogg).

Some devices can be updated to play and record in new formats. DVRs usually record in proprietary file systems for copy protection, although some can use FAT file systems. Recordings from standard-definition television usually have 480p/576p while HDTV is usually in 720p/1080p.

WORKING OF DVR

In a nutshell [6], a DVR is a glorified hard drive inside a fancy box that looks nice in your entertainment center. The hard drive is connected to the outside world through a variety of jacks on the back of the box, usually the typical RCA connections that you would use to hook up, say, a cable box or a VCR.

The television signal comes into the DVR's built-in tuner through antenna, cable or satellite. If the signal comes from antenna or cable, it goes into an MPEG-2 encoder, which converts the data from analog to digital (MPEG-2, by the way, is the compression standard used to fit information onto a DVD). From the encoder, the signal is shipped off to two different places: first, to the hard drive for storage, and second, to an MPEG-2 decoder, which converts the signal back to analog and sends it to the television for viewing.

Some systems use dual tuners, allowing users to record different programs on different channels at the same time. On a few systems, you can even record two programs while watching a third pre-recorded show.

The device is driven by a customized operating system -- for instance, in the case of TiVo, the machine runs on a highly modified Linux installation. The operating system resides on the hard disk, along with the recording space, a buffer for live broadcasts, and in some cases a space for future expansion.

While the system might seem pretty ho-hum on first analysis, the digital storage of television signals opens up a whole new world of possibilities when it comes to playback and viewing.

First, a DVR is tapeless. With a VCR, the device itself is merely a recording tool; the blank cassette is the media. In a DVR, the media and tool are one and the same. This is obviously a plus if you never seem to be able to find a blank tape when it's time to record something, but it can also be a drawback. Because the media is hard-wired into the machine, adding additional storage space is not possible. There are Web sites that offer instructions on how to open a DVR and add a new hard drive, but beware -- this will definitely void your warranty. Getting more recording time is easy with a VCR -- just buy another box of blank tapes. More recording time on a DVR involves buying a new unit.

You can incorporate some DVRs into your home network, which can allow you to access your system remotely. You could set your DVR to record a specific show from halfway across the world with just a few clicks of your mouse.

Perhaps the most important benefit of DVRs is the unprecedented control over playback. With a VCR, you have to wait for a program to finish recording before you can start watching it. Since there's no tape to rewind, digital recording doesn't have this limitation. A program that started recording 10 minutes ago can be viewed at any time, even while it's still recording.

2.6 Peer-to-Peer v/s Multicast

P2P Networks have appeared almost ten years ago and at that time they were used mostly for file sharing especially copyrighted materials such as music and movie files. The motivation behind P2P networks are very strong which is utilizing unused resources. [7] Today most of us have computers with several gigahertz of speed and several gigabytes of hard drives and several kilobits of bandwidth so why not to use all of these resources instead of wasting them by not using. In P2P networks each host behaves both as a client and server, each computer serves its files to the others and download files from others. The first generation of P2P systems are semi-centralized such as Napster. The central servers just indexes file locations and names and provides search functionality to the peers, the actual files are kept at the peers not in the servers. In terms of functionality of the network there is nothing wrong with these semi-centralized P2P networks, however because users are sharing copyrighted materials these servers became an attacking point for media companies and copyright enforcement institutions such as RIAA.

So, the next generation of peer-to-peer networks are appeared such that there is no central server in the P2P network. In these systems, commonly users have a list of peers and they are forwarding their search queries to them. If these peers have the requested file they can send it to the requesting peer. If they do not have the requested file, they forward the request to their list of pairs. These next level of peers can communicate with the initial peer directly if they have this file, because most of the time IP address of the requester is inside the search query. This searching process can take more time compared to the centralized approach. However due to its decentralized nature it is very hard to suspend the system completely, because if there is two peers who knows each other the network is exist.

Multicast is a proposed solution for efficiently distributing the same packet to more than one host. The packet will not replicated until it is necessary.

However in order to use multicasting the routers on the way have to understand and behave accordingly to multicast packets. And also we need some protocols to manage joining and leavings and these protocols have to be used by all participating users. Today almost all LANs support multicasting and also some of the autonomous systems support multicasting internally, so in these areas applications can benefit from Multicasting.

The reasons why P2P is better than multicasting for Content Distribution Applications [8]:

1. Multicasting requires a common infrastructure

Multicasting requires a common infrastructure in terms of both hardware and software. First of all, the routers must support multicasting and they must be configured by system administrators such that multicasting is enabled. Secondly, all of the Autonomous Systems have to agree on a common multicasting protocol. Supporting multicasting on these routers will decrease their throughput so most of these backbone routers are not supporting multicasting. Even one of the new routers did not check IP checksum in order to achieve high speed (50 GB/s) although it is necessary by the IP Protocol.

So the situation for multicasting is not better than 20 years ago. However, the situation for P2P networks is far better than 10 years ago. The main reason behind these development is P2P networks are open, such that anyone can design and build a P2P network and starts using it immediately. Because everyone can participate to the development of P2P networks new ideas and techniques can appear easily and even competition can occur among them so any of these networks tries to be the best. Actually today we have different P2P networks specialized on different problems, for example BitTorrent is designed to allow fast propagation of a single big file.

2. Achieving Reliable Delivery with Multicast is very hard

Achieving reliable distribution of content is very hard with multicasting. It is true that not all types of content requires reliability, however most of the content such as music or movie files, applications and documents requires reliability.

The main reasons behind the hardness of reliable multicasting are:

The bandwidths of the participants can vary dramatically, so the distributor has to adjust its speed to some level which will permit most of the users to receive it. There has to be a way to retransmit missing packets of each listener, but it is hard to achieve this. There are some proposed solutions to achieve reliability in multicast. One example is NACK based approaches and another example is circulating the same data more than one times. A pure NACK based approach is impractical with huge number of receivers. And also circulating the same data more than on times can be inadequate for low-bandwidth users. In P2P networks reliability is almost default, because all the transmissions between peers are via unicast. So a reliable transmission protocol such as TCP can be used at no cost.

Chapter 3

Media Server

3.1 Wowza Media Server

Wowza Streaming Engine (known as Wowza Media Server prior to version 4) is a unified streaming media server software developed by Wowza Media Systems. The server is used for streaming of live and on-demand video, audio, and rich Internet applications over IP networks to desktop, laptop, and tablet computers, mobile devices, IPTV set-top boxes, internet-connected TV sets, game consoles, and other network-connected devices. The server is a Java application deployable on most operating systems.

Wowza Streaming Engine is compatible with standard streaming protocols. On the playout side [9], these include RTMP (and the variants RTMPS, RTMPT, RTMPE, RTMPTE), HDS, HLS, MPEG DASH, RTSP, Smooth Streaming, and MPEG-TS (unicast and multicast). On the live ingest side the server can ingest video and audio via RTP, RTSP, RTMP, MPEG-TS (unicast and multicast) and ICY (SHOUTcast / Icecast) streams. It also supports incoming streams via the RTSP and WOWZ protocols from mobile Android and iOS devices running the Wowza GoCoder mobile encoding application.

For on-demand streaming, Wowza Streaming Engine can ingest multiple types of audio and video files. Supported file types include MP4 (QuickTime container - .mp4, .f4v, .mov, .m4a, .m4v, .mp4a, .mp4v, .3gp, and .3g2), FLV (Flash Video - .flv), and MP3 content (.mp3).

Advantages

- Ingest Any Live Stream [9] - Use the best encoding solution for your needs, from free software-based RTMP encoders to broadcast-grade MPEG-TS hardware.
- Deliver to Any Device - Deliver video and audio streams to any player and any device, over any protocol, from a single media server.
- Stream Live, Linear, and On-Demand Content - Create live, linear, or on-demand streaming applications for live events, news, surveillance, monitoring, video libraries, and more.

- Record and Archive Live Streams - Record a live stream to a file available for on-demand playback. Record an entire live webcast into a single file, segment it into multiple files for chapter replay, or start and stop recording at predetermined points for partial archiving.
- Transcode Streams for Optimal Viewer Experiences - Transcoding with Wowza technology lets you refine your live encoding and streaming workflow, saving on upload bandwidth while delivering the best-possible-quality streams to each user's device.

Disadvantages

- **Delay in operation** – [10] If your server contains low configuration and has low bandwidth internet, then it experience delay in its operation, which means it will not stream in a seamless fashion.
- **High Cost** - It is too costly and offers yearly packages. Price of the Wowza is too high and there is no monthly subscription. Should there be any such monthly subscription, developers or owners of the website can subscribe to check its functioning, and upon being satisfied with the service they can opt for full subscription packages.
- **Not Open Source** - The server is not open source and one cannot edit it as per the need.
- **No Port Mapping** – If we want to send different streams coming from various sources to different sources we can't do this on wowza media server as all of its RTMP traffic is directed toward port 1935.
- **Platform Dependent** – The system of wowza media server is not platform independent which can be an issue as the server needs to be setup on different devices in a different way.
- **No dedicated server** – A dedicated server object cannot be allotted to each individual stream.

3.2 Node JS based Server

3.2.1 Node JS

Node.js is an open-source, cross-platform JavaScript run-time environment for executing JavaScript code server-side [11]. Historically, JavaScript was used primarily for client-side scripting, in which scripts written in JavaScript are embedded in a webpage's HTML, to be run client-side by a JavaScript engine in the user's web browser. Node.js enables JavaScript to be used for server-side scripting, and runs scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js has become one of the foundational elements of the "JavaScript everywhere" paradigm, allowing web application development to unify around a single programming language, rather than rely on a different language for writing server side scripts [13].

Advantages

- **Open Source -** Node.js is open source, so it’s free to use and no need to pay for license. There are also many open source modules supported by Node.js.
- **JavaScript as Programming Language -** It uses JavaScript as a programming language for both front-end and back-end which increase programmer productivity and code reusability.
- **Scalable -** You can scale your Node.js application by using two ways – Horizontal Scaling and Vertical Scaling, which helps you to improve your application performance. In Horizontal scaling you can add more nodes to your existing system. In Vertical scaling you can add more resources to a single node.
- **Better Performance** It provides better performance, since Node.js I/O operations are non-blocking. Also, it uses V8 JavaScript engine to execute JavaScript code. V8 engine compiles the JS code directly into machine code which make it fast.
- **Caching Support -** Node.js supports caching of modules. Hence, when a Node.js modules is requested first time, it is cached into the application memory. So next calls for loading the same module may not cause the module code to be executed again.

3.2.2 Using Node JS based RTSP-RTMP server

An open source node.js based server was edited and used for accessing the live RTSP and RTMP streams. The server consists of various files based on coffee-script and is used to connects clients from various sources. For each client a dedicated server can be launched on a different port. For each client if a RTSP server is launched on port number ‘X’ then the RTMP server is launched on port number ‘X+1’.

A RTSP stream URL looks like:

rtsp://<ip_address_of_server>:<port>/live/id

where **rtsp** is the protocol name **ip_address_of_server** denotes the IPV4 address of the server where the node.js script is running, port denotes the **port** on which RTSP server is accepting requests **live** denotes that the stream is live/real time and **id** is the unique identifier of every stream.

A RTMP stream URL looks like:

rtmp://<ip_address_of_server>:<port+1>/live/id

where **rtmp** is the protocol name **ip_address_of_server** denotes the IPV4 address of the server where the node.js script is running, port denotes the **port** on which RTSP server is accepting requests **live** denotes that the stream is live/real time and **id** is the unique identifier of every stream.

The RTSP server is initialized using a bash script and a port number is given as an input and it denotes the port on which RTSP server is to be launched. When a client makes a request to the RTSP server, its facsimile RTMP stream is generated on a port+1 of RTSP server. This particular helps in load balancing of the streams at different ports as various stream coming to a same port causes a lag in streams.

The server is written using coffee-script [12]. CoffeeScript is a programming language that transcompiles to JavaScript. Almost everything is an expression in CoffeeScript, for example if, switch and for expressions (which have no return value in JavaScript) return a value. As in Perl, these control statements also have postfix versions; for example, it can also be written after the conditional statement. Many unnecessary parentheses and braces can be omitted; for example, blocks of code can be denoted by indentation instead of braces, function calls are implicit, and object literals are often detected automatically.

When the server is initialized using the bash script a config file is created which contains all the necessary information about the server like: port number associated to various protocols, port numbers associated to audio and video transfer, video frame rate, video bitrate/audio bitrate, protocol timeout, rtmp message queue size. The server.coffee reads this config file and uses the various details mentioned to start a streaming server. On the start of streaming server various types of protocols are associated with different ports on the server and the server is now open to accept the requests from the client.

3.3 ffmpeg

ffmpeg is a free software project that produces libraries and programs for handling multimedia data. FFmpeg includes libavcodec, an audio/ video codec library used by several other projects, libavformat (Lavf), an audio/ video container mux and demux library, and the ffmpeg command line program for transcoding multimedia files [14]. FFmpeg is published under the GNU Lesser General Public License 2.1+ or GNU General Public License 2+. Two video coding formats with corresponding codecs and one container format have been created within the FFmpeg project so far. The two video codecs are the lossless FFV1, and the lossless and lossy Snow codec. Development of Snow has stalled, while its bit-stream format has not been finalized yet, making it experimental since 2011. The multimedia container format called NUT is no longer being actively developed, but still maintained. FFmpeg encompasses software implementations of video and audio compressing and decompressing algorithms [15]. These can be compiled and run on diverse instruction sets.

3.3.1 Saving Streams

Our usage of ffmpeg was to record the live streams coming from various centres to our node.js server. The command used for this purpose is:

ffmpeg -i rtsp://<ip_address_of_server>:<port>/live/Stream1 -b 500k -s 640x480 -vcodec copy -r 60 -y path/to/output.mp4

The above command works as follows the **-i** argument denotes the input stream i.e. ***rtsp://<ip_address_of_server>:<port>/live/Stream1*** in this case. The **–b** argument denotes the video bitrate i.e. ***500k*** in this case. Argument **–s** tells us the frame size we require the video in which is ***640x480.*** Argument **–vcodec** specifies to force a particular codec which is ***copy*** in this command, **-r** denotes the frame rate in Hz which is ***60*** here and **–y** denotes to overwrite the output file i.e. ***path/to/output.mp4*** output file will be overwritten every time a new stream is received.

To make sure that the ffmpeg command runs when the record attribute is set true in the system a special mechanism was made for this. Whenever the stream is started before sending any video frames it first tell the server that the stream is started and sets the record attribute true and as soon as this change is made a JAVA application reads this value and runs the command given above and starts recording the stream.

When the client request to stop the stream is again sets the record attribute back to false thus telling the JAVA application to stop recording the stream and saving the recorded stream as the output file. The type and quality of the stream depends on the arguments provided in the command.

3.3.2 Accessing DVR Streams

As previously explained in Chapter 2 that the work of DVR is to record the streams coming from various CCTV cameras and push the stream to a particular URL. The work of ffmpeg in this case was to convert the RTSP streams coming from the DVR to RTMP streams so that they can be played on the player. For this purpose the following command was used:

ffmpeg -i rtsp://<username>:<password>@<ip_address_of_dvr>/MPEG4/ch1 /main/av_stream -f flv -r 25 -s 640x480 -vcodec h264 rtmp://<ip_address_of_server>:<port>/live/ch1

In the above command argument **–i** denotes the input stream, which is ***rtsp://<username>:<password>@<ip_address_of_dvr>/MPEG4/ch1/main/av_stream*** in this case. Here username/password are the username and password of DVR the ***ip_address_of_dvr*** is the static or dynamic ip address given to the DVR and ***ch1*** denotes which camera channel this particular URL is used to access. The **–f** argument in the command denotes the format forced on the stream during conversion from RTSP to RTMP which is ***flv (flash video)*** in this example. The ***–r, -s, -vcodec*** are the same as explained above and the ***rtmp://<ip_address_of_server>:<port> /live/ch1*** URL denotes the address on which the incoming RTSP stream is forced and this new RTMP stream can be accessed by the video players.

3.4 Database

3.4.1 Mysql

MySQL is an open-source relational database management system (RDBMS). MySQL can be built and installed manually from source code, but it is more commonly installed from a binary package unless special customizations are required [16]. On most Linux distributions, the package management system can download and install MySQL with minimal effort, though further configuration is often required to adjust security and optimization settings.

Though MySQL began as a low-end alternative to more powerful proprietary databases, it has gradually evolved to support higher-scale needs as well. It is still most commonly used in small to medium scale single-server deployments, either as a component in a LAMP-based web application or as a standalone database server. Much of MySQL's appeal originates in its relative simplicity and ease of use, which is enabled by an ecosystem of open source tools such as phpMyAdmin. In the medium range, MySQL can be scaled by deploying it on more powerful hardware, such as a multi-processor server with gigabytes of memory.

3.4.2 Database Design

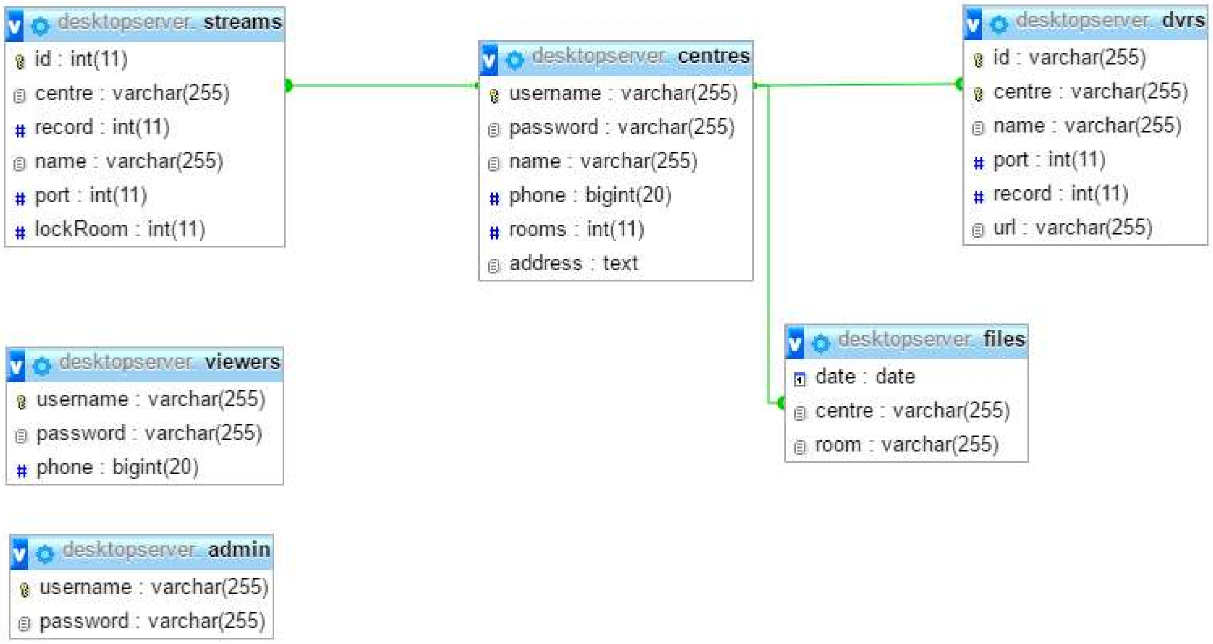


Figure 2: Structure of Database

Our database consists of a total of 6 entities each of which is represented as a table. The meaning of each entity is as follows:

- **Streams:** This entity contains the information about various incoming RTSP streams which are being pushed by CCTV cameras and Tablets/Mobiles. It contains an attribute by the name port number, which gives us information about on which port a particular stream is being send. The attribute centre is a foreign key which denotes the username of the invigilation centre. The attributes record and lockare used to lock the particular stream and to record the particular stream respectively.

- **Viewers:** This entity denotes the invigilator who will be viewing the stream coming from various centres, each of them will be given with a username and password to login to the system.
- **Admin:** Admins represents a group of users who have the authority to add new viewers, add new centres, edit and delete centres, add the DVR boxes into the system and manage the activities of the system.
- **Centres:** The centres denotes the invigilation centre where the exam is being conducted. Each invigilation centre consists of its basic information like address, phone number, centre code(username), and the number of rooms which are to be invigilated in the centre. The username and password provided is to login inside the invigilators app so that the streaming can be started for a particular room.
- **Files:** The files table consists of the previously recorded streams and their dates.
- **Dvrs:** The dvr table consists of the information about each DVR. It has attributes like unique ID, allotted centre, name. It also contains a pre-specified URL and port number for all CCTV cameras associated with it. It also contains record attribute which shows whether the stream is being recorded or not.

3.5 Apache HTTP server

The Apache HTTP Server, colloquially called Apache, is the world's most used web server software. The Apache HTTP Server Project is a collaborative software development effort aimed at creating a robust, commercial-grade, feature-rich and freely available source code implementation of an HTTP (Web) server [17]. The project is jointly managed by a group of volunteers located around the world, using the Internet and the Web to communicate, plan, and develop the server and its related documentation. This project is part of the Apache Software Foundation. In addition, hundreds of users have contributed ideas, code, and documentation to the project.

In our case the APACHE webserver is used to process the PHP based web application and display it on the browser. The APACHE webserver for our system runs on port 8080. The APACHE webserver renders the data which PHP writes and shows it on the screen. The JWPlayer requires a HTTP server such as APACHE to play the RTMP streams coming from the node.js server.

3.6 JAVAFX

JavaFX is a software platform for creating and delivering desktop applications, as well as rich internet applications (RIAs) that can run across a wide variety of devices. JavaFX is intended to replace Swing as the standard GUI library for Java SE, but both will be included for the foreseeable future. Written as a Java API, JavaFX application code can reference APIs from any Java library. For example, JavaFX applications can use Java API libraries to access native system capabilities and connect to server-based middleware applications.

The look and feel of JavaFX applications can be customized. Cascading Style

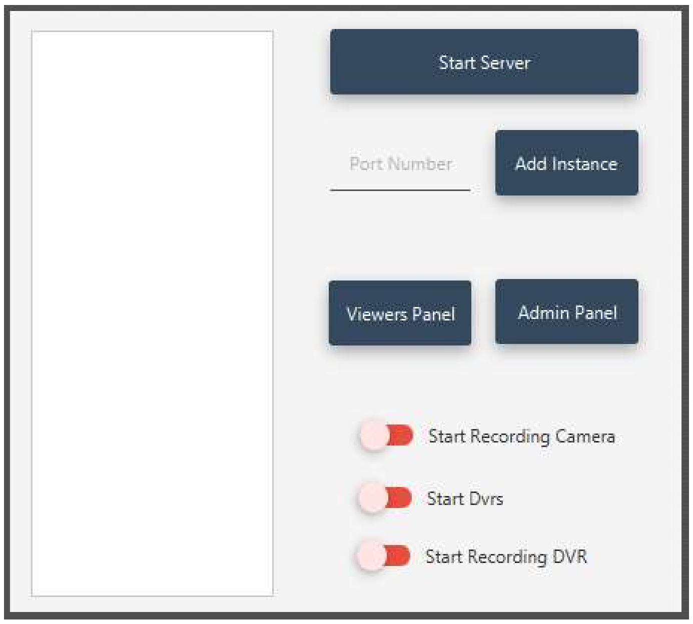


Figure 3: Server Initialization Application

Sheets (CSS) separate appearance and style from implementation so that developers can concentrate on coding. Graphic designers can easily customize the appearance and style of the application through the CSS. If you have a web design background, or if you would like to separate the user interface (UI) and the back-end logic, then you can develop the presentation aspects of the UI in the FXML scripting language and use Java code for the application logic.

In our case, we have made a JAVAFX application, which contains various options of starting the node.js server, starting a new instance of server when a port is given, starting the recording of DVR streams, Camera streams and starting the conversion system of input RTSP DVR streams to output RTMP streams.

The libraries used in the application made are [18]:

- **Jfoenix.jar –** This library is used to make various types of input and output components like buttons, textview, listview etc.
- **Mysql-connector-java.jar –** This library is used to make connection of a java application to the mysql database and after then to create statement and execute various queries required for the application.

3.6.1 Structure of JAVAFX Application:

The javafx application mainly consists of three files which are necessary for the application to run, they are [19]:

- FXMLDocument.fxml – This files consists of a design in fxml language and this is the UI of the application and this is what is displayed to a user.
- FXMLDocumentController.java – This file contains methods and components which are related to the FXMLDocument.fxml, For example, if a design button is made in FXMLDocument.fxml then what events will be performed on clicking the button is written in FXMLDocumentController.java. Basically FXMLDocument.fxml is the front-end and FXMLDocumentController.java is the back-end which handles events and responses.
- Server.java – This is the main class which is called and it creates and sets the scene and stage and runs the application.

When the Start Sever button is pressed, a command is run which runs a bash script and starts the server and all the ports on which server is started are displayed in the list on the left side of the pane. When the Add Instance button is pressed after providing an input port number, a script runs again and starts a server at that particular port. When the Start Recording Camera/DVR button is switched on then the recording of all the streams coming to the server starts with the help of FFMPEG, and when Start DVR button is pressed then the conversion of streams from RTSP to RTMP takes place.

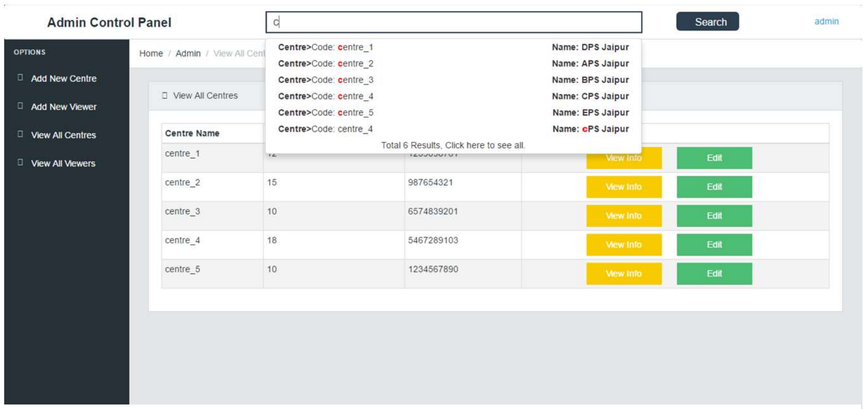
Chapter 4

Web Applications

4.1 Admin Control Panel

The Admin control panel serves as an interface which provides easy access to the database for the admin, while restricting others from misusing the system. The various functions of this panel are:

- Safe access of database only to the admin using a username and password, which is stored in the form of hash. The login page sends the password in the form of hash using the PHP function 'sha1' to generate the hash of the entered password. This is sent using POST method to the server, which matches it with the stored hash value of correct password. It must be noted that the correct password is not stored in the database directly, for security reasons.
- Add new centres and modify or delete details of existing centres. The admin registers any new centre with its name, a username and password, address and contact details along with number of rooms to be monitored at the centre. With this corresponding entries are created in the database, providing necessary information to other components of the system.
- Register details of new viewer with a username, password and contact number. With these details only, a person would be able to login and view the streams from different centres.
- To register the details of a DVR located at a centre. The various details include the username, password, IP address, port number, number of active channels and the centre it belongs to.



Centre>Code: centre_1 Name: DPS Jaipur
Centre>Code: centre_2 Name: APS Jaipur
Centre>Code: centre_3 Name: BPS Jaipur
Centre>Code: centre_4 Name: CPS Jaipur
Centre>Code: centre_5 Name: EPS Jaipur
Centre>Code: centre_4 Name: cPS Jaipur
Total 6 Results, Click here to see all.

Centre Name					
centre_1	12	[illegible]		View Info	Edit
centre_2	15	987654321		View Info	Edit
centre_3	10	6574839201		View Info	Edit
centre_4	18	5467289103		View Info	Edit
centre_5	10	1234567890		View Info	Edit

Figure 4: Admin Control Panel

4.2 Viewer's Panel

The viewer's panel is which the invigilators would be using to monitor the activities in a room. The various functionalities of the panel include the following:

- A safe login page for the registered viewers to login into the portal. This also as mentioned above uses 'sha1' and POST method for security.
- The portal displays a list of all centres to the viewer, along with a mark indicating if any room in the centre is currently active/streaming or not. After selecting a centre, all the active streams, if any, are displayed to the viewer.
- There is also an option to display a single video at a time, in which case the server will not be accessing the other streams generated by the DVR, by concentrating only over one stream. This allows minimum bandwidth to be used at the sender side, as only a single stream is being received and displayed at a time to the server by the DVR.

  As soon as the output of a camera is requested by the viewer, an entry in the database corresponding to that camera is updated to indicate the request. This change leads to the execution of the ffmpeg command for the requested stream and closing such commands for other streams. JAVA is used to perform the task of external trigger. It periodically monitors the database entries for any change, and then responds to it, to execute an external process.

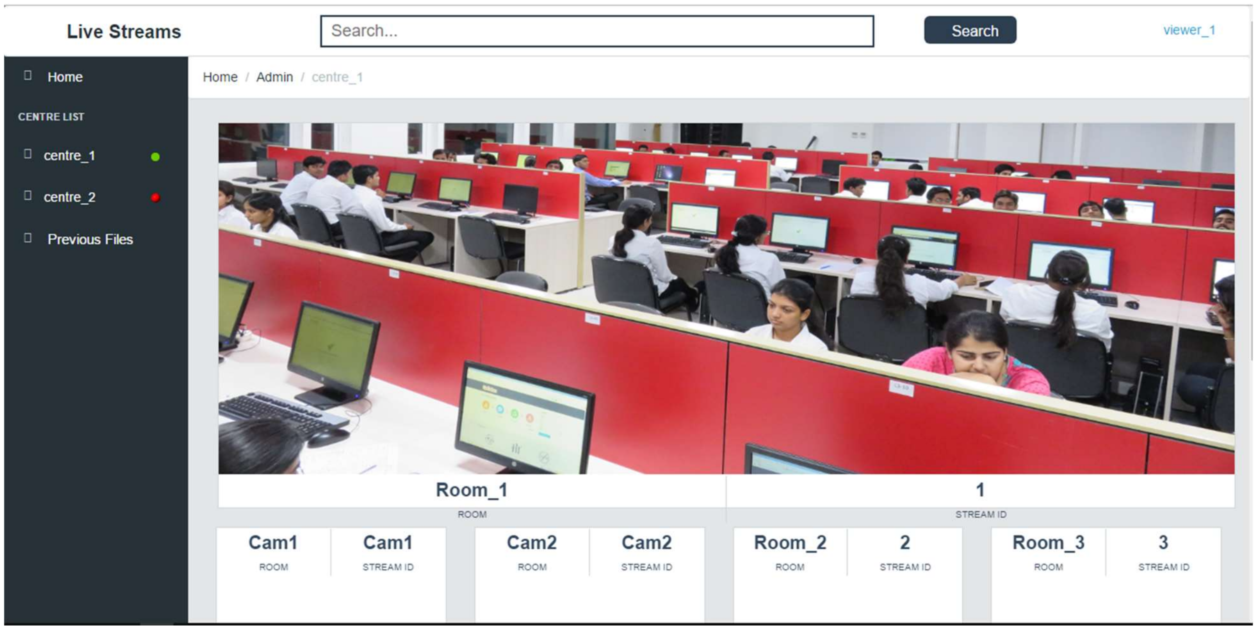


Figure 5: Displaying Live Streams

- The viewer can also view the previously saved videos stored at the server.
- The viewer can use the search functionality, in which he can search on the basis of date, centre name, room number, etc. The search functionality is implemented using AJAX.
- The RTMP stream sent by the server is played on the webpage using JW Player. The focus on a player can be swapped with that of the other by clicking the other player.

4.2.1 JW Player

JW Player is a New York based company which has developed a video player software of the same name. The player, for embedding videos into web pages, is used by news, video-hosting companies and for self-hosted web videos.

During the early development, before it was purchased by Google, YouTube videos were streamed by JW Player [20]. In 2015 JW Player was rewritten to reduce size and load time. Version 7 was licensed under the proprietary Creative Commons Attribution-Noncommercial-Share Alike 3.0 license. It had integrated support for HTML5 Video and Flash Video, allowing video to be watched on phones, tablets and computers. That year the company's paying customer base grew by more than 40 percent to 15,000, 60% from the USA. 2.5 million Websites used the free edition, playing about a billion videos per month.

Chapter 5

Android Applications

5.1 Accessing Database

Both the android applications explained below access the database for various purposes like to validate information like username and password, or to access information like details about a centre, active rooms, etc., or to update the information to reflect changes of the state of video stream in the database.

Connection to a database can be made directly by using JDBC [21]. But this is never to be used in android applications because of the following reasons:

- Never use a database driver across an Internet connection, for any database, for any platform, for any client, anywhere. That goes double for mobile. Database drivers are designed for LAN operations and are not designed for flaky/intermittent connections or high latency.
- An android apk file can easily be reverse engineered to generate code. This would lead to leaking of important information related to the database like the name of database and its password.

Due to the above reasons, web services, designed for use over the internet, have been used to provide access to database to the android applications. For Android, REST Web services are probably the easiest to consume, since there is no built-in support for SOAP or XML-RPC. But whether the Web service is implemented in Java, or PHP, or Perl, or SNOBOL, is a personal choice.

In this project we have used PHP to access provide access to the database. With the help of PHP, we create an interface to be used by android applications, which allows it to check if the entered username and password are correct or not, to access the lists of centres and its rooms, along with details like if it is streaming or not, etc.

To access the various server side scripts to access database content, we use ‘volley’ library that makes the use of network requests easier. That data is exchanged in JSON format, i.e.; the database content is send in JSON format by the server side scripts.

5.1.1 'volley' library

Volley is an HTTP library that makes networking for Android apps easier and most importantly, faster [22]. It manages the processing and caching of network requests and it saves developers valuable time from writing the same network call/cache code again and again. And one more benefit of having less code is less number of bugs and that's all developers want and aim for.

My mean of writing the same network call code is AsyncTask and the logic/code you write for fetching response from Web API and displaying it in particular View. We have to take care of displaying ProgressBar/ProgressDialog inside onPreExecute() and onPostExecute().

Advantages of using Volley are:

- Automatic scheduling of network requests.
- Multiple concurrent network connections.
- Transparent disk and memory response caching with standard HTTP cache coherence.
- Support for request prioritization.
- Cancellation request API. You can cancel a single request, or you can set blocks or scopes of requests to cancel.
- Ease of customization, for example, for retry and backoff.
- Strong ordering that makes it easy to correctly populate your UI with data fetched asynchronously from the network.
- Debugging and tracing tools.

Volley excels at RPC-type operations used to populate a UI, such as fetching a page of search results as structured data. It integrates easily with any protocol and comes out of the box with support for raw strings, images, and JSON [23]. By providing built-in support for the features you need, Volley frees you from writing boilerplate code and allows you to concentrate on the logic that is specific to your app.

Volley is not suitable for large download or streaming operations, since Volley holds all responses in memory during parsing. For large download operations, consider using an alternative like DownloadManager.

5.1.2 JSON format

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language [24].

JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

In JSON, they take on these forms:

- An *object* is an unordered set of name/value pairs. An object begins with {(left brace) and ends with} (right brace). Each name is followed by: (colon) and the name/value pairs are separated by, (comma).
- An array is an ordered collection of values. An array begins with [(left bracket) and ends with] (right bracket). Values are separated by, (comma).
- A value can be a string in double quotes, or a number, or true or false or null, or an object or an array. These structures can be nested.
- A string is a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes. A character is represented as a single character string. A string is very much like a C or Java string.
- A number is very much like a C or Java number, except that the octal and hexadecimal formats are not used.
- Whitespace can be inserted between any pair of tokens.

5.2 Streaming App

The Streaming Application is responsible for sending the video streams, captured from the camera of the device, to the media server. The various tasks performed by the app for accomplishing this are:

- Authentication and allocation of a room to the device.
- Compression followed by transmission of RTSP stream to the server.

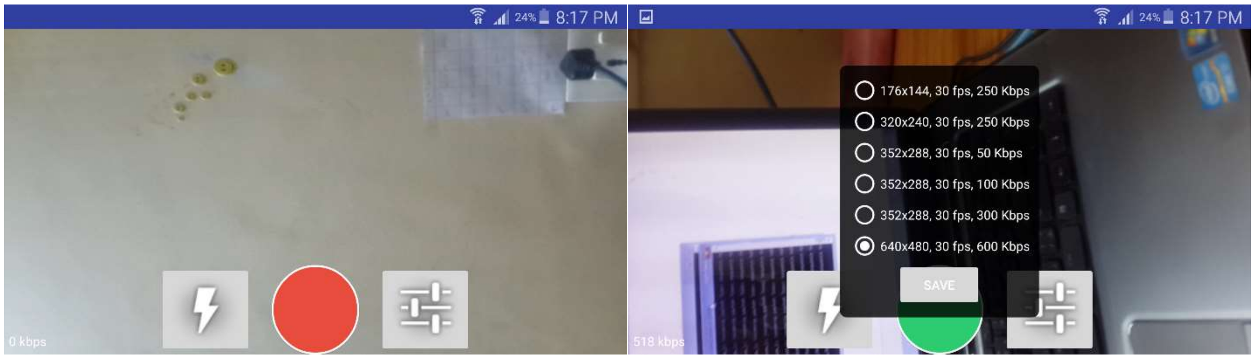

8:17 PM
0 kbps
176x144, 30 fps, 250 Kbps
320x240, 30 fps, 250 Kbps
352x288, 30 fps, 50 Kbps
352x288, 30 fps, 100 Kbps
352x288, 30 fps, 300 Kbps
[✓ 640x480, 30 fps, 600 Kbps]
SAVE
518 kbps

Figure 6: Streaming App

5.2.1 Authentication and allocation of Room

The app begins with the authentication process in which each device at a centre logs in with a username and password. These credentials are common for all the devices in a particular centre. After the authentication process, all the rooms allocated to centre are listed which are yet to be allocated to any device. Those rooms which are already selected by other devices are not displayed in the list.

From the list, the device selects a room, which is allocated to device. The device then receives a unique URL of the RTSP stream that it is supposed to stream. With this the streams from different rooms can differentiated from one another at the server side. But this process has certain complications involving multiple devices being allocated to a single room as explained below.

Locking Mechanism

Say two devices, A and B, login one after the other to the app with the credentials of the same centre. Note that both A and B have not selected the room after logging in. Say a room, Room1, is available at the time when both the devices logged in. This room would be displayed in the list of both the devices A, B. Now say that A has locked Room1 and proceeded further. But Room1 is still being displayed in the list of device B. Now if device B also selects Room1, then both the devices would be sending the same stream over the network, of which only one would be accepted by the server, leading to loss of data from the second stream.

Now that the problem is clear from the above illustration, we have used the property of atomicity of database transactions so that the above problem is resolved. We have a column in the database indicating that a room has been selected a device. In this case, when the device A selects the room, he sets the value accordingly in the database. Now when device B tries to select the room, the server side script tries to set the value in the database. It responds with an error indicating that the value is already set, i.e. the room is already selected and allocated to a device.

Using the property of Atomicity of database transactions, now each device would be allocated to a unique room.

Note that, allocation of room to a device does not mean that the device has started streaming video to the server. When the device starts the stream, a different entry in the database is set corresponding to it, which allows the server to know that the stream has started from the specific room. In this case, if the recording option is set at the server by the admin, then this database entry triggers execution of an external program to save the stream at the server. This external trigger is implemented using JAVA at the server side.

5.2.2 Compression and transmission of RTSP stream

Once the room is allocated, and stream is started, then the video from the camera of the device is captured, compressed and transmitted using the RTSP protocol. The compression used in the app is H.264. Both the compression and transmission is accomplished using the ‘libstreaming’ library explained below.

5.2.3 ‘libstreaming’ library

libstreaming is an API that allows you, with only a few lines of code, to stream the camera and/or microphone of an android powered device using RTP over UDP [25].

- Android 4.0 or more recent is required.
- Supported encoders include H.264, H.263, AAC and AMR.

The first step you will need to achieve to start a streaming session to some peer is called 'signaling'. During this step you will contact the receiver and send a description of the incoming streams. You have three ways to do that with libstreaming.

- With the RTSP client: if you want to stream to a Wowza Media Server, it's the way to go.
- With the RTSP server: in that case the phone will act as a RTSP server and wait for a RTSP client to request a stream
- Or you use libstreaming without using the RTSP protocol at all, and signal the session using SDP over a protocol you like.

How does it work?

There are three ways on Android to get encoded data from the peripherals:

- With the MediaRecorder API and a simple hack.
- With the MediaCodec API and the buffer-to-buffer method which requires Android 4.1.
- With the MediaCodec API and the surface-to-buffer method which requires Android 4.3.

1. Encoding with the MediaRecorder API

The MediaRecorder API was not intended for streaming applications but can be used to retrieve encoded data from the peripherals of the phone. The trick is to configure a MediaRecorder instance to write to a LocalSocket instead of a regular file.

As of Android Lollipop using a LocalSocket is not possible anymore for security reasons. But using a ParcelFileDescriptor does the trick.

This hack has some limitations:

- Lip sync can be approximative.
- The MediaRecorder internal buffers can lead to some important jitter. libstreaming tries to compensate that jitter.

It's hard to tell how well this hack is going to work on a phone. It does work well on many devices though.

2. Encoding with the MediaCodec API and the buffer-to-buffer method

The MediaCodec API do not present the limitations mentioned above, but has its own issues. There are actually two ways to use the MediaCodec API: with buffers or with a surface [25].

The buffer-to-buffer method uses calls to dequeueInputBuffer and [queueInputBuffer] to feed the encoder with raw data. Well it's not easy, because video encoders that you get access to with this API are using different color formats and you need to support all of them. Moreover, many encoders claim support for color formats they don't actually support properly or can present little glitches. All the hw package is dedicated to solving those issues.

If streaming with that API fails, libstreaming fallbacks on streaming with the MediaRecorder API.

3. Encoding with the MediaCodec API and the buffer-to-buffer method

The surface-to-buffer method uses the createInputSurface() method. This method is probably the best way to encode raw video from the camera but it requires android 4.3 and up.

The gl package is dedicated to using the MediaCodec API with a surface.

It is not yet enabled by default in libstreaming but you can force it with the setStreamingMethod(byte) method.

Packetization process

Once raw data from the peripherals has been encoded, it is encapsulated in a proper RTP stream. The packetization algorithm that must be used depends on the format of the data (H.264, H.263, AMR and AAC) and are all specified in their respective RFC [25]:

RFC 3984 for H.264: H264Packetizer.java

RFC 4629 for H.263: H263Packetizer.java

RFC 3267 for AMR: AMRNBPacketizer.java

RFC 3640 for AAC: AACADTSPacketizer.java

RTCP packets are also sent to the receiver since version 2.0 of libstreaming. Only Sender Reports are implemented. They are actually needed for lip sync. The rtp package handles packetization of encoded data in RTP packets.

5.3 Invigilation App

The Invigilation App is responsible for displaying the live stream send by the media server. The basic tasks of the app are:

- To authenticate the viewer credentials.
- To display a list of all centres and corresponding rooms with the detail that if the room is currently streaming or not.
- To display the RTMP stream for the room.

The app initially gets the credentials like username and password for the viewer. These credentials are sent for authentication with the database entries using the server side scripts and 'volley' library.

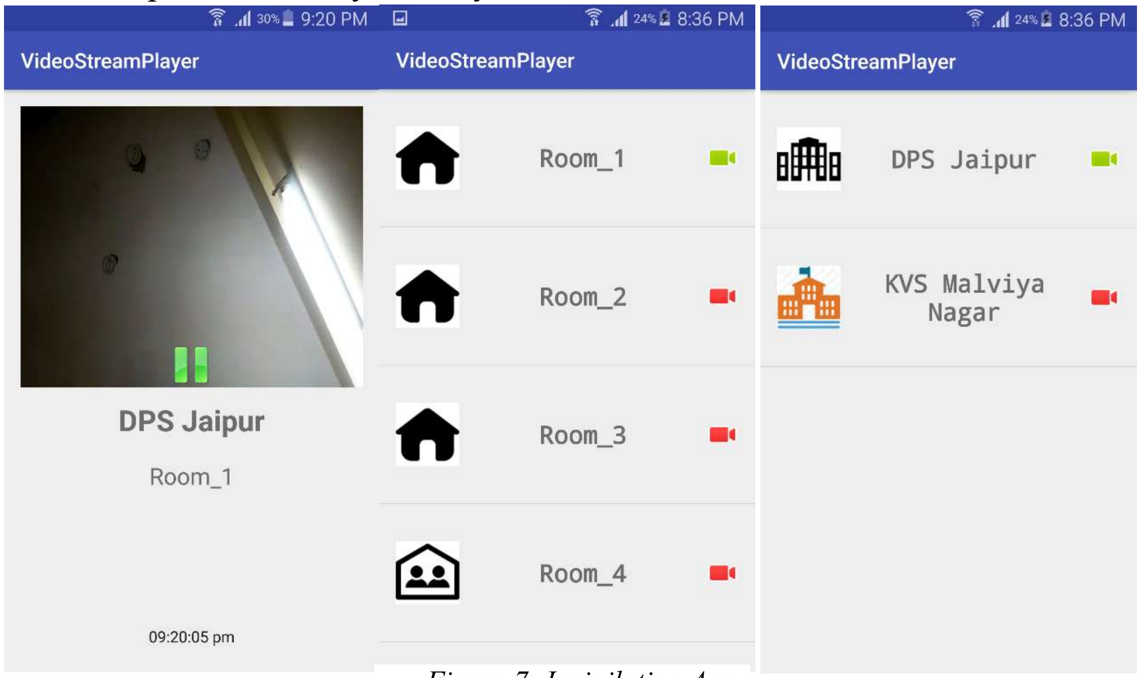


Figure 7: Invigilation App

Once a legitimate viewer logs in the app, a list of all centres is displayed to him. Along with this an indication of whether any room in the centre is streaming currently or not. When the viewer selects a centre with active rooms, the list of active rooms in it are displayed to the viewer. From this he selects one room, to display the RTMP stream corresponding to the room.

The RTMP stream can be displayed using any of the following three ways:

- Using VideoView
- Using flash plugins
- Using external libraries like 'vitamio'

Displaying multiple videos is not always possible using VideoView as it depends on the hardware specifications of the devices.

Adobe may have cut support for Flash in Android Jelly Bean and beyond. Hence this option is also not preferred.

The 'vitamio' library efficiently uses VideoView internally to play RTMP streams. We display a single stream at a time on the device. The library and its functions are explained below.

5.3.1 'vitamio' library

Vitamio is an open multimedia framework or library for Android and iOS, with full and real hardware accelerated decoder and renderer. It's the simple, clean and powerful API of Vitamio that makes it famous and popular in multimedia apps development for Android and iOS.

According to the developers' feedback, Vitamio has been used by more than 10,000+ apps and 500 million users around the world.

Vitamio can play 720p/1080p HD mp4, mkv, m4v, mov, flv, avi, rmvb, rm, ts, tp and many other video formats in Android and iOS. Almost all popular streaming protocols are supported by Vitamio, including HLS (m3u8), MMS, RTSP, RTMP, and HTTP [26].

Network Protocols:

The following streaming protocols are supported for audio and video playback [26]:

- MMS
- RTSP (RTP, SDP), RTMP
- HTTP progressive streaming
- HLS - HTTP live streaming (M3U8)

And yes, Vitamio can handle on demand and live videos in all above protocols.

Media formats

Vitamio used FFmpeg as the demuxers and main decoders, many audio and video codecs are packed into Vitamio beside the default media format built in Android platform, some of them are listed below.

- RMVB
- MKV
- MOV
- M4V
- AVI
- MP4
- 3GP, etc.

Vitamio also supports the display of many external and embedded subtitle formats.

There are more wonderful features available in Vitamio [26].

- Support wide range screens from small phone to large tablet
- Multiple audio tracks support
- Mutitiple subtitles support, including external and embedded ones
- Processor optimization for many platforms
- Buffering when streaming
- Adjustable aspect ratio
- Automatically text encoding detection

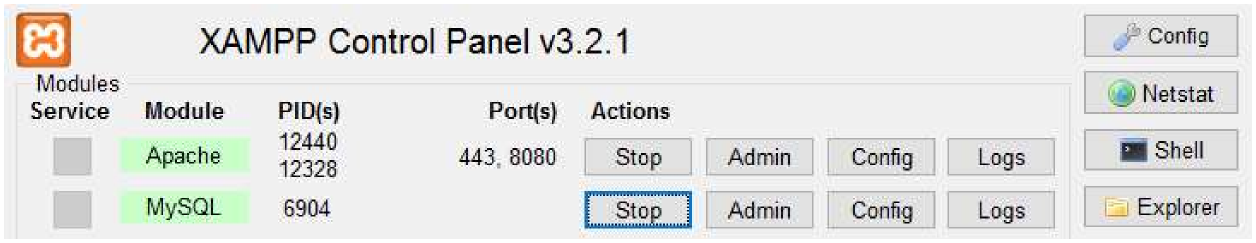
Chapter 6

System Initialization

To run the complete system following are the steps to be taken care of in sequence:

STEP 1

- Run the Xampp and start the Apache Web Server and the MYSQL server.



XAMPP Control Panel v3.2.1

Modules

Service	Module	PID(s)	Port(s)	Actions			
	Apache	12440 12328	443, 8080	Stop	Admin	Config	Logs
	MySQL	6904		Stop	Admin	Config	Logs

Figure 8: Xampp Control Panel

STEP 2

- Launch the JAVAFX based Server Initialization Application by starting Server.jar.

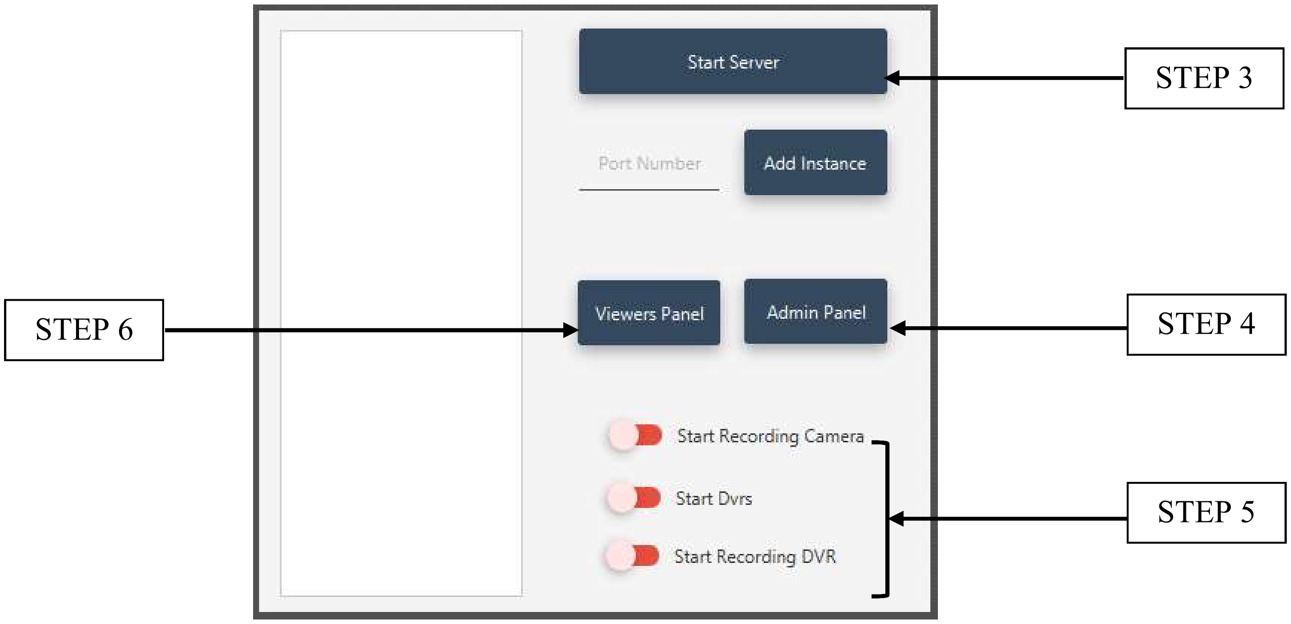


Figure 9: System Initialization Application

Step 3

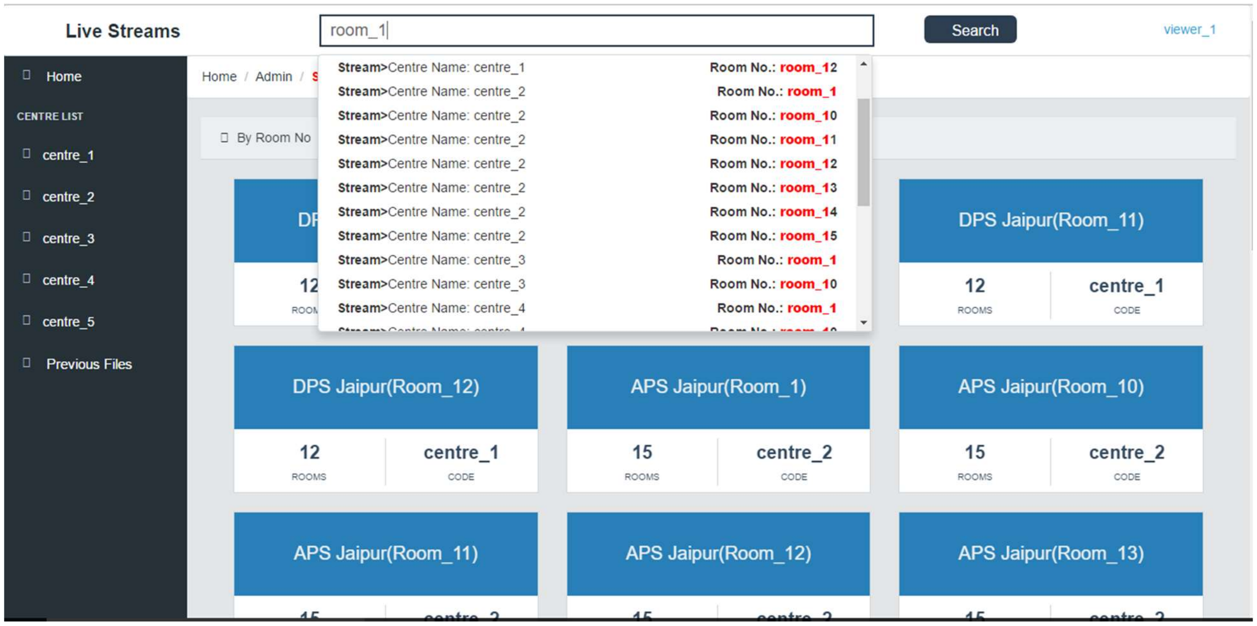
- Click on the button Start Server in same application to start the node.js server.

Step 4

- Launch the Streaming App from mobile device and login. Select the room from the list available. Select the video quality from the quality list available and press the red button to start the stream. To stop the stream press the green button. In addition, start the DVR and enter its IP address using the admin panel to the database using Admin Panel button on the JAVA FX application.

Step 5

- Press the Start DVR button on the JAVA FX application to start the streaming of DVR.
- If also want to record both the camera and the DVR stream then toggle the Start Recording Camera and Start Recording DVR button.



Stream>Centre Name: centre_1	Room No.: room_12
Stream>Centre Name: centre_2	Room No.: room_1
Stream>Centre Name: centre_2	Room No.: room_10
Stream>Centre Name: centre_2	Room No.: room_11
Stream>Centre Name: centre_2	Room No.: room_12
Stream>Centre Name: centre_2	Room No.: room_13
Stream>Centre Name: centre_2	Room No.: room_14
Stream>Centre Name: centre_2	Room No.: room_15
Stream>Centre Name: centre_3	Room No.: room_1
Stream>Centre Name: centre_3	Room No.: room_10
Stream>Centre Name: centre_4	Room No.: room_1

Figure 10: Viewers Control Panel

Step6

- Press the Button Viewers Panel on the JAVAFX application, a browser window will open, enter credentials to login select the center and watch the streams coming from different rooms of that centre.
- If also want to view Streams coming from different room open the viewers android application, login and select centre, room and view the stream.

Chapter 7

Performance and Testing

Our system was analyzed by a few representatives from REIL and the first set of suggestions in the form of comparison of our project with their existing project, given by them is shown below, along with our comments to each.

No	Description	Third Party Solution	Our Approach	Reasons
1.	Bandwidth	At present, 60kbps uploading speed is require to uploading video stream from anywhere in tablet (01 Nos cameras) and 100 kbps for the DVR encoder (16 Nos Camera) with the help of Indigenously developed Bandwidth Bonding Algorithm/Aggregator.	At present, 200-300 kbps uploading speed is require to uploading video stream in mobile (01 Nos cameras).	The tablet used by the third party has a camera of 1MP only, while the mobile used for testing by us had 13MP camera. So our stream required around 300kbps bandwidth and theirs required 60kbps. You require to display only one camera at a time from a DVR connected to 16 cameras, then we can also do this at a bandwidth of 100kbps.
2.	Encoder	Self-developed patent video compression technology.	Depend upon online third party software for video compression.	Developing a new Compression engine is itself a research project. So we have used open source H264 compression engine.
3.	Media Server	Dynamic Adaptive Bit rate. Geo Delivery of Content. Compatible with existing technology and applications on client side. Secured Streaming with dual encryption and Geo IP Locking. Intelligent Live Reporting and Log Generation. Cloud based technology enabling	At present, MNIT's server is used for above which is not fully designated for the webcasting purpose. Some features/applicati on may be absent.	MNIT's VM is used to host the open-source, free Node-JS based server. Otherwise many sophisticated paid servers like Wowza, Red5, etc. which provide all the mentioned services and CDN.

		Scaling upto unlimited attendees.		
4.	Decoder	Minimum Jittery and Latency, SSL implementation along with authenticated access. Proxy and Firewall Complaint. No plug-in required. Independent of any operating system or browser. Token Based Streaming and Restricted Download.	Third party solution is implemented for this purpose which is yet to be tested in field application.	For our system, no plugin is required, it is independent of any operating system and implements token based streaming and recording. Video is decoded using JWPlayer, which is an open source software.

Table 1 : Comparison of our Project with Vmukti's as given by REIL

After this we took our project to REIL office and demonstrated about it to them. We installed our media server on their local intranet and accessed 5 streams from two different DVR's located at different locations in the factory. One inside the office and the other was located to monitor the gates of the factory. All 5 streams were shown simultaneously to them, with minimum latency and almost no jitter. The users over there were impressed to see that there are no frozen frame in our system, which they encountered in their current system.

But the existing system used by REIL had a different approach for handling the streams from DVR. As per their existing system, the DVR is connected to a Wifi router, which sends the streams to a tablet, which is also connected to the same router. So now in this case, all the DVR's are configured to the same IP address and accessed by the app installed on the tablet. While in our case, each DVR is to be given a unique IP address, such that it is visible to the remote media server. Our project, in the case of DVR, relies on the compression provided internally by DVR, to compress the streams. These streams are then directly accessed by the server. While in the case of Vmukti's system, these streams are accessed by the app installed on the tablet, which further compresses them, and then forwards it to the server.

Chapter 8

Conclusion and Future Scope

8.1 Conclusion

This report explains the functioning and implementation of the Live Video Surveillance system, along with directions to use it. This system efficiently transmits multiple video streams from both cameras of android devices, and from CCTV captured videos through DVR. It implements Live streaming by using protocols like RTSP for sending the video from cameras, and RTMP for receiving the videos at the viewing end.

A major aspect of the project was the media server to be used. The media server converts the RTSP stream received to RTMP stream to be played at the viewing end. The network load on the server is distributed by using different ports. Additional functionalities such as saving video on the server is implemented using ffmpeg. As the DVR does not send a stream to the server, we have to access it using ffmpeg at the server.

At the viewing end we have the option of displaying multiple streams together or a single stream at a time, along with an option to view all the previous streams. Almost all the requirements of REIL are fulfilled, but to use it at the field, certain more improvements are required followed by a thorough testing.

8.2 Future Scope

This project can be further improved by adding more functionalities and making it more efficient and easy to use. These can be:

- Implementing the DVR part by using an android application as an intermediator between server and DVR to further compress the video streams.
- To make the system adaptive, such that the quality of video being sent is controlled on the basis of the load on the server, or network bandwidth available at the sender side.
- To add additional functionalities such as sending of Voice messages from the invigilator to the exam hall, where the recording is taking place, to better monitor and control classrooms.

- To improve the compression algorithm by using H.265. High Efficiency Video Coding (HEVC), also known as H.265 and MPEG-H Part 2, is a video compression standard, one of several potential successors to the widely used AVC. In comparison to AVC, HEVC offers about double the data compression ratio at the same level of video quality, or substantially improved video quality at the same bit rate. It supports resolutions up to 8192×4320, including 8K UHD.

Chapter 9
References

[1] https://www.axis.com/files/whitepaper/wp_latency_live_netvid_63380_external_en_1504_lo.pdf

[2] https://tools.ietf.org/html/rfc2326

[3] http://www.adobe.com/devnet/rtmp.html

[4] https://en.wikipedia.org/wiki/H.264/MPEG-4_AVC

[5] https://en.wikipedia.org/wiki/Digital_video_recorder

[6] http://electronics.howstuffworks.com/dvr.htm

[7] https://www.cs.rutgers.edu/~rmartin/teaching/fall08/cs552/position-papers/004-01.pdf

[8] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.126.3249&rep=rep1&type=pdf

[9] https://www.wowza.com/

[10] http://www.spaculus.org/blog/why-wowza-server-is-a-nightmare-for-developers-here-is-the-reason-behind-it/

[11] https://nodejs.org/en/

[12] https://en.wikipedia.org/wiki/CoffeeScript

[13] http://www.dotnettricks.com/learn/nodejs/advantages-and-limitations-of-nodejs

[14] https://ffmpeg.org/

[15] https://linux.die.net/man/1/ffmpeg

[16] https://en.wikipedia.org/wiki/MySQL

[17] https://en.wikipedia.org/wiki/Apache_HTTP_Server

[18] http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm

[19] http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784

[20] https://support.jwplayer.com/customer/portal/articles/1403727

[21] http://stackoverflow.com/questions/2256082/best-way-to-implement-client-server-database-architecture-in-an-android

[22] https://developer.android.com/training/volley/index.html

[23] http://www.technotalkative.com/android-volley-library-example/

[24] http://www.json.org/

[25] https://github.com/fyhertz/libstreaming

[26] https://www.vitamio.org/en/docs/Basic/2013/0509/4.html