VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

# SYSTEM FOR RECORDING VIDEO FROM IP VIDEOCAMERAS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. JIŘÍ TRAVĚNEC
AUTHOR

BRNO 2015

# SYSTÉM PRO ZÁZNAM STREAMOVANÉHO VIDEA Z IP KAMER

SYSTEM FOR RECORDING VIDEO FROM IP VIDEOCAMERAS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. JIŘÍ TRAVĚNEC

VEDOUCÍ PRÁCE
SUPERVISOR
Mgr. JANA SKOKANOVÁ

BRNO 2015

## Abstrakt

Tato diplomová práce je zaměřená na přenos multimédií v reálném čase z IP kamer. Jejím hlavním cílem je vysvětlit teoretické základy přenosu v reálném čase přes počítačovou síť a popsat vývoj nahrávacího systému. Tento nahrávací systém je určen převážně k nahrávání přednášek ve školách. Práce obsahuje popis vývoje serverové nahrávací aplikace a webového administračního rozhraní. Teoretická část vysvětluje témata spojená s přenosem médií v reálném čase, počítačovými sítěmi a zpracováním multimédií, jako například real-time streaming protokoly, kódování, komprese, síťová odezva, zahlcení sítě a další.

## Abstract

This diploma thesis focuses on multimedia streaming from IP cameras. Its main goal is to explain theoretical background of real-time streaming via computer networks, and describe development of a recording system. This recording system is meant to be used mainly in schools for lecture recording purposes. The thesis contains description on how a recording server application and web-based management system were developed. The theoretical part explains topics related to multimedia streaming, networking, and multimedia procesing, such as real-time streaming protocols, encoding, compression, network latency, network congestion and many others.

## Klíčová slova

Streamování v reálném čase, nahrávání, real-time streaming protokoly, multimédia, multimediální komprese, kódování, sítě, RTP, RTCP, RTSP, SDP, AngularJS, avconv, libav, live555, Java

## Keywords

Real-time streaming, recording, real-time streaming protocols, multimedia, multimedia compression, encoding, networking, RTP, RTCP, RTSP, SDP, AngularJS, avconv, libav, live555, Java

## Citace

Jiří Travěnec: System for Recording Video from IP Videocameras, diplomová práce, Brno, FIT VUT v Brně, 2015

# System for Recording Video from IP Videocameras

## Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením paní Mgr. Jany Skokanové a že jsem uvedl všechny literární prameny, ze kterých jsem čerpal.

. . . . . . . . . . . . . . . . . . . . . . .
Jiří Travěnec
July 31, 2015

## Poděkování

Tímto bych především rád poděkoval vedoucí mé diplomové práce Mgr. Janě Skokanové za vstřícný a vřelý přístup a odborné vedení mé práce. Rád bych vyjádřil poděkování všem, kteří mě podporovali v mém studijním úsilí, zejména rodině a přátelům. Také bych chtěl poděkovat Markusovi za anglickou záchranu v hodině dvanácté.

By this, I would like to thank primarily to supervisor of my diploma thesis, Mgr. Jana Skokanová, for helpful and pleasant approach, and expert and professional supervision. I would like to express my gratitude to all people, who supported me during my studies. Also, I would like to thank to Markus for an English help in the nick of time.

# Contents

# Chapter 1

# Introduction

In 1893 great engineer Nikola Tesla publicly demonstrates radio and radio connection for the very first time. [1] This event can be seen as a beginning of a new era, which was about to change the world. It has been 122 years now, and the broadcasts went through a lot of revolutions and even more changes. Nowadays, we can transmit multimedia content all over the world, and moreover even communicate with other people over such distances using video-calls.

Six billion hours of video is watched every month on YouTube, the largest video on demand service in the world. [2][3] That means 12 minutes for every user of the site every day. When, in 2013, YouTube suffered a service outage for five minutes, the world internet usage decreased by 40 percent. [4]

Another service called Skype brought world-widely available free video-calls, allowing people from different countries to hear and see each other over the Internet. This was such a change to people's life, that even a new verb was created. „To Skype somebody“, which means to have a video-call with a person. [5]

Just two small statements were pitched to show how important multimedia broadcasts over computer network nowadays are. The streaming, as it is called, is growing area of science and business, and this thesis tries to add some value to this field.

This thesis' main goal is to design and implement Internet Protocol (IP) video-camera recording system to record and store mainly school lectures. I have been studying at two universities, and both my faculties at Brno University of Technology (BUT) and University of Eastern Finland (UEF) were using streaming and/or recording systems in order to improve their studies. The University of Eastern Finland used video-conference units to connect two separated campuses, which were situated in two different towns. The Faculty of Information Technology BUT used their equipment to acquire lecture recordings and provide them to students who could not attend the lectures or wanted to revise parts of the subject. That was priceless help especially when revising algorithms or mathematical procedures, which are hard to be described by text.

The topic of multimedia streaming over computer networks covers many separated fields of study. Especially knowledge of computer networks, multimedia processing and real-time communication is essential. With that in mind, the theoretical part of this thesis brings to mind how computer network works and acts, and most importantly how it can affect the real-time communication quality. These issues are briefly discussed in the chapter 2.

The main topic of this thesis is streaming over computer networks. For such a purpose many network protocols were invented. These protocols, such as the Real-time Transport Protocol, the Real-time Transfer Streaming Protocol, or the Session Description Protocol, are essential for the streaming. And therefore, the chapter 3 focuses on the real-time streaming protocols, and extensively describes and explains its purpose, functionality and how the whole system of protocols works.

Third important component of the streaming is the payload. The content of the stream is the most important part for user, because that is what is delivered to him. From technological point of view, topics about media bit-rate are closely related to the network issues, and therefore it should be discussed. The chapter 4 explains why even constant bit-rate of encoded stream is not precisely constant, and how does the encoding and coding work. It focuses on encoding and decoding process, especially buffers, their tasks, and impact on the overall system performance.

The next three chapters cover the practical part of this thesis, the design and implementation of the recording system. As result of the implementation, a recording system intended to record mainly school lectures from IP cameras will be created. This system will run as a server application and based on a defined schedule it will record the streams from cameras.

The chapter 5 starts with introduction to server applications, explaining how the server applications act and behave, how to design a server application and what such a code has to fulfil. All the necessary topics related to the analysis, such as database draft, use cases, and system decomposition are explained in this chapter, leading to a necessary design of the application.

After the chapter about analysis and design, the first part of the system, the server application, is explained. The chapter 6 expands the design with practical matters, explaining what technologies and tools will be used, and describing implementation-related problems and solutions for them.

The seventh chapter [7] then describes how the management application for the system was created. There is explained what technologies were selected for the development, and how does the final application work and interact with user.

All these chapters together with the conclusion [8], which discusses the results, should very well describe the whole problematic around multimedia streaming and recording the streams. The field of subject is very wide, so some related topics were not described completely as it was not necessary for this thesis.

„The best way to predict the future is to invent it.“

— *Alan Kay*

# Chapter 2

# Networks and servers

Every video broadcast is delivered between at least two points. We can assume that one of these points, a source point, is a camera and the second one, destination, is some consumer - e.g. a user's computer. Even if we have only two-point system, there always is medium, that carries the data between these points. And that is computer network. Network is a crucial part of the broadcast delivery and understanding the problematic is important part of understanding whole world of multimedia broadcasts.

As our demands on the broadcast system increase, we might need to add some other important elements to our system structure: recording devices, servers, balancers, caches, etc.

International Organization for Standardization defines well-known The Open Systems Interconnection model (OSI Model), which is a conceptual model that characterizes and standardizes the internal functions of a computer network by partitioning it into abstraction layers [6]. Every frame of audio or video we want to deliver or receive is affected by what happens on all of these layers. To understand how we can deliver a video broadcast, we must understand what happens with our multimedia data through these layers and how it can affect the quality of final experience.

Moreover, above this application layer, there is another equally important part of the delivery system, which is a video processing part. This segment is described further is chapter 4, as it is not part of the network problematic. However, it is important to take it into account, because it is an relevant part of the processing and delivering chain.

## 2.1 Media layers

Computer networks acts as a communication media in a video-delivery systems. The network connects a source device to a destination consumer using cables, switches, routers, and many protocols are used to provide communication. For multimedia delivery in real-time quality of service (QoS) and network performance is important aspect. It defines limitations and possibilities for such a delivery system.

A throughput, latency, packet delay variation, and bit error rate are the most important parameters of a computer network. [7] Every parameter affects the transmission in different way and for every issue there is a different compensation mechanism.

**Throughput** represents amount of successfully delivered data per period [8]. It is limited by capabilities of devices participating in network transmission such as client network

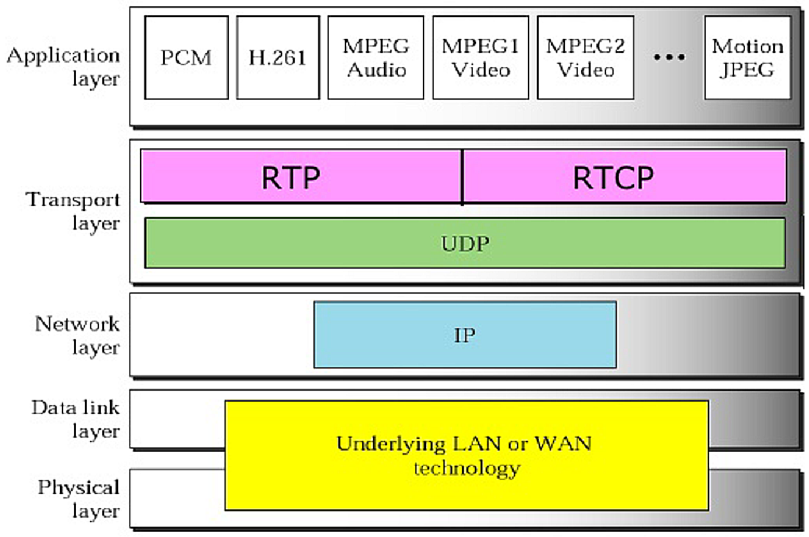


Figure 2.1: Protocol stack for multimedia services

cards, switches, routers etc. Usually it is measured in bits per second (bps) or packets per seconds (pps).

This parameter outlines how voluminous data stream can be delivered through the network. Higher the bit-rate of the stream is, higher throughput we need to provide error free delivery. An effort to transmit more data per time that the throughput is, leads to network congestion.

**Network congestion** occurs when a link or node is carrying so many data that its quality of service deteriorates. Simply explained, it is a moment when a link carries more than the destination node can process. That leads to buffer overflow, queue delaying, packet loss or blocking of new connections. [9] If packets are discarded, a retransmission of packet may come, which leads to even greater congestion.

This situation is very critical and if we take into account the fact, that media streams are usually high bit-rate transmissions, it can easily lead to network congestion. Therefore, the RTP Control protocol (RTCP) was invented to provide feedback on the quality of service. The RTCP is discussed later in chapter 3.2.

Delivering a single real-time audio/video over any modern network will not probably, even closely, lead to network congestion. But consider a device delivering a stream to, for example, one thousand viewers. If we are delivering a stream with bit-rate of 3 Mbps to 1000 viewers, that is a bit-rate of 3 Gpbs. And that is far behind capabilities of standard modern network routers[1]. That is one of the reasons, why the multicast group communication was invented.

[1]Assuming that standard network switch throughput is at maximum 1 Gpbs.

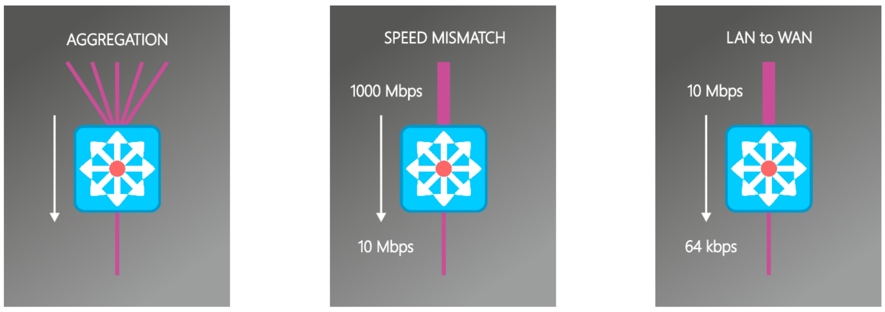


Figure 2.2: Three types of nodes that can cause problems with network congestion.

**Multicast** is a technique to deliver same data stream over an IP network to many clients without need to send them more than once. Therefore, it is one-to-many or even many-to-many group communication mechanism. It is widely used in multimedia streaming and multipoint video-conferencing. Every node which wants to receive a multicast stream sends so called join message which is processed by closest multicast node, e.g. switch. When sender starts sending data to the multicast group, only one copy of data is sent and then the switch provides duplication of packets and delivery to the joined nodes. That lowers required throughput at specific points of network.

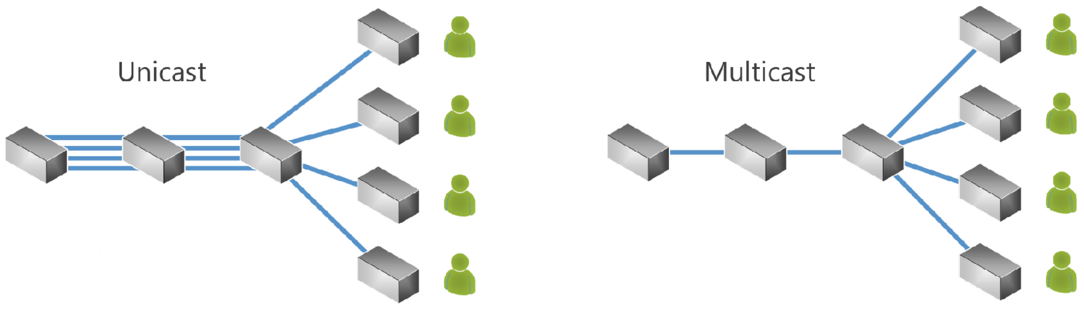


Figure 2.3: Graphical explanation of reduction of bandwidth usage with multicast.

**Latency** is one of the important parameters of computer network. It describes how long it takes to deliver data from source node to the destination node. This parameter is highly important for bidirectional communication (in our case e.g. video-conference), or time-critical deliveries (e.g. Network Time Protocol). For unidirectional communication, such as multimedia stream delivery, it could be ignored or solved by some mechanisms[1]. Usual unit used to measure latency is seconds, respectively milliseconds or microseconds. The time delay between sending and receiving an information is caused by many factors. Big part of latency is produced by software delays in stream processing, which are extensively described in section 4.1, but even lower levels of OSI layer model produces latency while processing every packet. These latencies are created by circuit latencies, processor limits, queueing and other technical and electronic aspects.

[1]If we assume that latency is stable.

**Packet delay variation**, or sometimes commonly called jitter, is a parameter that describes a deviation of packet delivery from basic latency. The cause as well as consequences are the same as mentioned earlier in the paragraphs describing latency. For example, delay variation at hardware level can be caused by utilization of a switch; higher utilization leads to higher buffers usage, which leads to increase packet delivery delay. This issue cannot be easily ignored even in unidirectional communication, otherwise it would cause interruptions in fluency of the data delivery. The solution is to introduce a receive buffer which is big enough to cover the variation. Then the size of buffer (in meaning of time) must be added on to the total latency of the whole network. Setting up a proper buffer can be a challenging task; too big buffer can lead to unnecessary total latency, too low buffer to stream interruptions.

**Bit error rate** (BER) characterizes amount of bits corrupted during transfer. That is caused by transmission channel noise, interference, distortion, bit synchronization problems, attenuation, wireless multipath fading, etc. [10], and so it is a hardware problem which can never be totally solved and it always has to be taken into account. Every packet sent over IP network is signed by Cyclic Redundancy Check (CRC) which is used to detect accidental changes during delivery. If packet is corrupted, it is discarded and therefore not delivered to the destination. This problem can be handled on fourth level of ISO, by transport layer mechanisms, by e.g. resending the packet. These procedures are rather discussed in next subsection.

## 2.2 Host layers

Layers of OSI model numbered four to seven are called host layers. Every layer has its own task including providing transmission mechanisms on transport level, maintaining the communication session, decoding, processing, presenting the delivered data and so on.

Layer four is called transport layer and its main task is to provide mechanisms to transfer data over network reliably. The most common protocols for this task are User Data Protocol (UDP) and Transmission Control Protocol (TCP). Both of these protocols have different key aspect of work.

**Transmission Control Protocol** is a protocol firstly formally defined in the RFC 675 [11] in 1974. Its task is to reliably deliver data over computer network. That means delivering every piece of data in right order regardless the time it takes. For this purpose, it carries many mechanisms. The most important one, when talking about real-time data delivery, is re-send mechanism.

When data are transmitted using TCP, sending site maintains a timer from when the packet was sent. If the packet is not delivered, it is automatically retransmitted after timer has elapsed. This grants the delivery, but one lost packet can lead to suspension of the whole transfer for perceptible time - hundreds of milliseconds. [12] Even if the packet is delivered, but corrupted, the retransmission takes two times round-trip time, which can be unacceptable delay in real-time data transfer.

Last fact that has to be mentioned, is the possibility of network congestion caused my TCP retransmission mechanism, as mentioned in chapter 2.1. If the packet is dropped on a router because of buffer overflow or similar reason, retransmission of the packet can make the situation even worse.

**User Data Protocol** is a protocol formally defined in year 1980 in the RFC 768. [13] It uses minimal and connectionless mechanisms which makes the whole transaction very simple. It sends individual messages, so-called datagrams. We cannot claim this protocol to be reliable, because it does not provide any mechanism guaranteeing the datagram to be delivered, as well as two datagrams being delivered in right order.

In comparison with TCP, it lacks the re-send mechanism, so the data is either delivered on first try or lost. This might seem incorrect, but in real-time delivery, time is key aspect and packet loss can be handled on higher level, e.g. The RTP protocol.

The UDP datagram does not carry any sequence number, so it is impossible to detect packet loss or packet re-order.

In 2000 The IETF Signaling Transport working group defined **Stream Control Transmissin Protocol**, which has later, in 2007, been standardized in the the RFC 4960. [14] This protocol should provide extended mechanisms especially for multimedia and stream data deliveries like paralel independent delivery streams, multihoming, path selection, etc. As of 2015, the protocol is not widely supported. Both Microsoft and Apple lack any support, which makes it rather impractical to deliver media streams. [15]

# Chapter 3

# Real-time streaming protocols

The main task of the real-time protocols is to transfer certain data streams. These data streams has to be related to some time-line, which describes its flow in time. The streams are split into blocks, which are packed into packets and sent over IP network. There has to be a timestamp attributable to every block of data.

The packets can be delivered over standard IP network using standard transport protocols such as UDP, as mentioned in the previous chapter, but some additional attributes must be carried in every packet, to ensure our purpose. One of the very basic information that has to be described, is what kind of data is transmitted, specification of payload, especially codec, bitrate, number of channels, resolution etc. Next, when a block of data in received, it is necessary to identify, where does the block fit to, talking about temporality. For such a purpose, every packet has to be equipped with a timestamp identifying its sampling instant.

Also mechanisms to synchronize two or more data streams mutually (e.g. video and audio stream from the same multimedia broadcast), to report quality of service are needed. Most importantly, a protocol by which we can apply for stream delivery, which can describe what data streams can be transmitted. Because such a functionality is rather complex, it has been split into many separated protocols and RFCs. For data transmission, there is the RTP (Real-time Transport Protocol), which also carries timestamps, payload description and other related information to describe every block of data correctly. The RTCP (Real-time Transport Control Protocol) provides fucntionality to report quality of service, synchronize two or more streams. For establishing and controlling media session purposes, RTSP (Real-time Streaming Protocol), and SIP (Session Initiation Protocol), which are both text-based protocols similar to HTTP (Hypertext Transfer Protocol), were invented.

In case we are transmitting a medium, which carries more than one track (e.g. video and audio), then every track represents different data stream, and these tracks are sent separately. When a client is receiving audio-video file with two tracks using the RTP/the RTCP, two separated sessions will be created with the RTP and the RTCP communication for each one. That means it will establish four separated connections. This allows the client to receive only the streams, it really needs for playback.

| IP Header<br>(20 bytes) | UDP Header<br>(8 bytes) | RTP Header<br>(min. 12 bytes) | Payload<br>(variable) |
|---|---|---|---|

Table 3.1: Structure of the VoIP packet (as in IPv4)

## 3.1 RTP (Real-time Transport Protocol)

The Real-time Transport Protocol (the RTP) is a network protocol for delivering audio and video over IP networks. The RTP is used extensively in communication and entertainment systems that involve streaming media, such as telephony, video teleconference applications, television services and web-based push-to-talk features. [16]

It was firstly published in 1996 as the RFC 1889 [17] and later replaced by the RFC 3550 [18] in 2003. The protocol became de-facto standard for local network based real-time end-to-end video and audio streaming as it is nowadays supported or used by most of the multimedia based devices.

Because the RTP is usually carried by the UDP, it has to provide some additional mechanisms ensuring packet order, packet loss detection, and a way to correctly detect which packet fits what frame of delivered stream. And therefore, the RTP carries two main synchronization mechanisms - packet synchronization (sequence number) and stream synchronization (timestamp, SSRC, CSRC identifiers).

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

| | | | | | | |
|---|---|---|---|---|---|---|
| V | P C | CC | M | PT | Sequence number | |
| Timestamp | | | | | | |
| Synchronization source (SSRC) identifier | | | | | | |
| Contributing source (CSRC) identifiers<br>(list 0 to 15 items; 4 bytes each)<br>... | | | | | | |

Table 3.2: Structure of the RTP header, according to the RFC 3550

The packet synchronization is done in very similar way as in TCP. Packet header holds 16-bit sequence number which is incremented by one for each the RTP data packet sent, and it may be used to detect packet loss [18]. The RFC 3550 does not describe any action in case of packet loss and it is left to the higher level of data processing to resolve such a situation. A loss of one packet, in for example audio stream, can lead to imperceptible lag of a fraction of second, which can be made unnoticeable to user with suitable error concealment algorithms. The start number should be random to make plaintext attacks on encryption more difficult.

The task of the RTP is to carry a single media stream through IP network. For such a purpose, we must split the media stream into packets and send them over network. Size of single IP packet depends on network configuration[1], so depending on size of the packet and kind of media, which is transmitted, one packet can carry more than one frame or only part of the frame.[2] Because one packet very often does not carry one whole frame, the RTP provides additional synchronization mechanism to define what data fits what frame. For such a purpose, every packet carries timestamp.

Timestamp is a 32-bit number, which describes the sampling instant of the first frame in the RTP data packet[3]. The sampling instant must be derived from a clock that increments

[1]MTU, jumbo packets

[2]What affects the size of a frame is described in section 4.2

[3]In case there is more than one frame in data packet, the timestamp describes the first one, and therefore the oldest one. The sampling instant of the other ones can be calculated.

monotonically and linearly in time to allow synchronization and jitter calculations. [18] The clock rate varies depending on payload type, e.g. 8000 Hz, 44100 Hz, 90000 Hz etc. The initial value should be random, as for the sequence number.

Several consecutive data packets will have equal timestamps if they are (logically) generated at once, e.g. belong to one video frame. [18]

As mentioned earlier, every media stream is carried by separated the RTP connection. The RTP timestamp from different media streams usually advance at different rates as well as they have different (random) offset. Therefore, although these timestamps are sufficient to reconstruct the timing of a single stream, directly comparing the RTP timestamps from different media is not effective for synchronization. For such a purpose, the RTCP contains synchronization between the RTP timestamp and so-called wallclock. Wallclock represents real time, usually provided by NTP (Network Time Protocol). This synchronization mechanism is described in chapter 3.2.

The RTP packet header also carries synchronization source identifier, which identifies source of a stream. It is a random 32-bit number which should be globally unique within an the RTP session [18]. This number is called SSRC Identifier and it is also used in the RTCP protocol to refer the RTP stream.

## 3.2 RTCP (Real-time Transport Control Protocol)

The Real-time Transport Control Protocol is a sister of the RTP. Its basic functionality and packet structure is described together with the RTP in the RFC 3550 [18]. Other RFCs then extend this protocol and describes other functionalities.

the RTCP provides out-of-band statistics and control information for an the RTP session. It partners with the RTP in the delivery and packaging of multimedia data, but does not transport any media data itself. The primary function of the RTCP is to provide feedback on the quality of service (QoS) in media distribution by periodically sending statistics information to participants in a streaming multimedia session. [19] The bandwidth usage is generally much lower, and it should not exceed 5%[1] [19] of total session bandwidth.

The RTCP is usually carried over the UDP. The speed of delivery is not that crucial as of the RTP, but since the RTCP includes information about RTCP packet loss (last SR), a packet loss can be compensated. Typically, for the RTP data stream an even-numbered UDP port will be used, and for the RTCP communication will be establish on the next higher odd-numbered port. [20]

The main task of the RTCP is to deliver regular reports of quality of service between sender and receivers. Two basic kinds of reports are distinguished: senders report (SR) and receivers report (RR). Both these reports carries different information and so they have slightly different packet structure. The basic idea behind the packet structure is the same - the RTCP packet carries the RTCP header, which contains SSRC of the packet sender, then sender info (only in case of SR), followed by zero or more reception report blocks, one for each of the synchronization source.

Every statistical information in the report is valid at one precise moment described by timestamp, that is part of the report.

The **sender report** consists of sender info and reception report blocks. The sender info

[1]The RTCP packets are sent regularly and its frequency does not depend on the RTP bandwidth. Number 5% is used only too illustrate the difference.

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

| | | | | | |
|---|---|---|---|---|---|
| V=2 | P | RC | PT=SR=200 | Length | header |
| SSRC of sender | | | | | |
| SSRC #1 (SSRC of first source) | | | | | sender info |
| NTP timestamp, most significant word | | | | | |
| NTP timestamp, least significant word | | | | | |
| RTP timestamp | | | | | |
| sender's packet count | | | | | |
| sender's octet count | | | | | |
| SSRC #1 (SSRC of first source) | | | | | report block 1 |
| fraction lost | | | cumulative number of packets lost | | |
| extended highest sequence number received | | | | | |
| interarrival jitter | | | | | |
| last SR (LSR) | | | | | |
| delay since last SR (DLSR) | | | | | |
| SSRC #2 (SSRC of second source) | | | | | report block 2 |
| ... | | | ... | | |
| ... | | | | | |
| profile-specific extensions | | | | | |

Table 3.3: Sender Report RTCP packet

contains 64-bit NTP timestamp, the RTP timestamp, sender's packet count and sender's octet count. NTP timestamp indicates the wallclock time, which, in combination with timestamps returned in reception reports, can be used to measure round-trip propagation. The RTP timestamp holds 32-bit number, which represents timestamp of the stream at the same time as NTP timestamp defines. These two timestamps can be used together to calculate intra- and inter-media synchronization. [18] Sender's packet count and octet count represents number of packets and octets (bytes) sent by sender.

Reception report blocks consists of source identifier (identifying source to which the information in this reception report block pertains), fraction lost (the fraction of the RTP data packets from such as source lost since previous SR or RR), cumulative number of packet lost (the total number of the RTP data packets lost since the beginning), interarrival jitter (an estimate of the statistical variance of the RTP data packet interarrival time, measured in timestamp units and expressed as an unsigned integer), and other statistical information.

The **receiver report** consists only of header and reception report blocks. The meaning of each statistical information stays the same as for sender report. Last SR timestamp reports and delay since last SR informs when the last sender report was received.

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

| | | | | | |
|---|---|---|---|---|---|
| V=2 | P | RC | PT=RR=201 | Length | header |
| SSRC of packet ender | | | | | |
| SSRC #1 (SSRC of first source) | | | | | report block 1 |
| fraction lost | | | cumulative number of packets lost | | |
| extended highest sequence number received | | | | | |
| interarrival jitter | | | | | |
| last SR (LSR) | | | | | |
| delay since last SR (DLSR) | | | | | |
| SSRC #2 (SSRC of second source) | | | | | report block 2 |
| ... | | | ... | | |
| ... | | | | | |
| profile-specific extensions | | | | | |

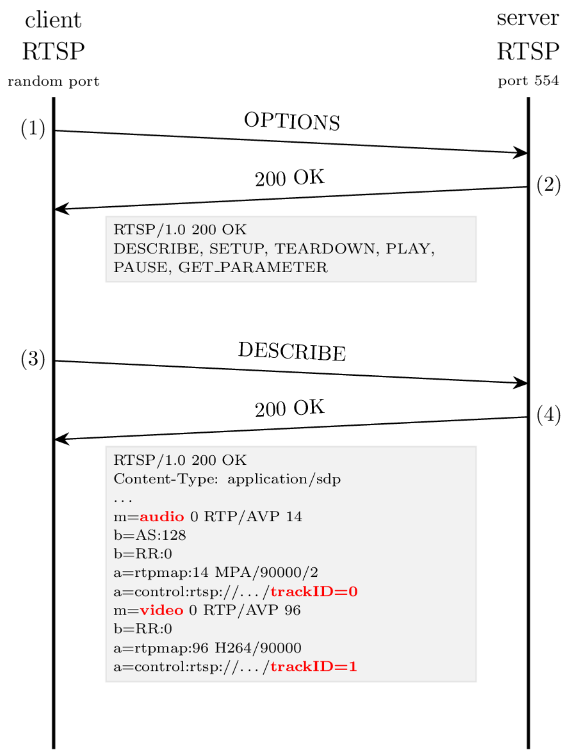
Table 3.4: Receiver Report RTCP packet

## 3.3 RTSP (Real-time Transfer Streaming Protocol)

The Real-time transfer streaming protocol, defined in the RFC 2326 [21], is a session initialization and controlling protocol. Its design is similar to HTTP protocol; but in comparison with stateless HTTP, RTSP has a mechanism to provide state - a session identifier. The default transport layer for the RTSP is both TCP and UDP. Usually, TCP is preferred, and as opposed to HTTP, implementations, where whole communication is performed in one TCP session, can be found.

In the RTSP, every stream is identified by an URL[1]. Two schemes *rtsp*, and *rtspu* are reserved. The *rtsp* requires that commands are issued via a reliable protocol (TCP), and the *rtspu* refers to an unreliable protocol (UDP). [21] Default port is 554 [21], alternatively occasionally 8554 is used. [22]

As another difference from HTTP, not only client is able to initialize the communication, but even server can send commands to client[2] For RTSP, every message is acknowledged in HTTP style, e.g. with 200 OK for success, or 402 Payment Required for some kind of client problem, or even 500 Internal Server Error in case of server problem [21].



```
RTSP/1.0 200 OK
DESCRIBE, SETUP, TEARDOWN, PLAY,
PAUSE, GET_PARAMETER
```

```
RTSP/1.0 200 OK
Content-Type: application/sdp
...
m=audio 0 RTP/AVP 14
b=AS:128
b=RR:0
a=rtpmap:14 MPA/90000/2
a=control:rtsp://.../trackID=0
m=video 0 RTP/AVP 96
b=RR:0
a=rtpmap:96 H264/90000
a=control:rtsp://.../trackID=1
```

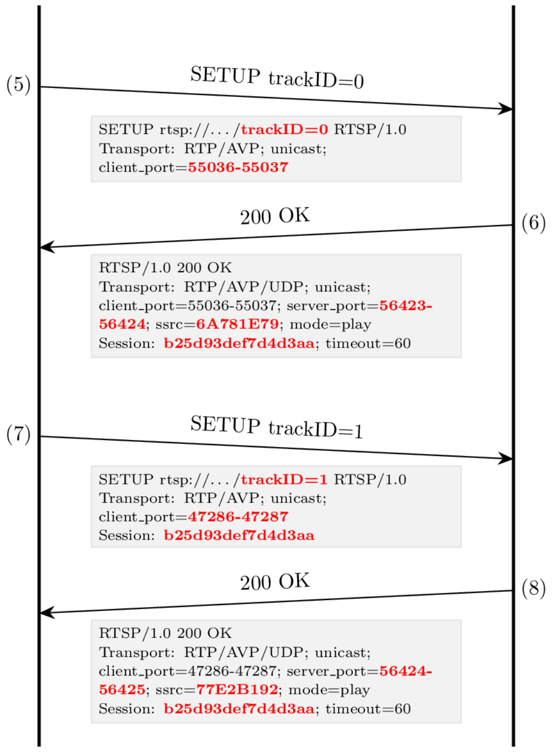
Figure 3.1: RTSP: OPTIONS and DESCRIBE request together with responses.
(The SDP response carries two streams; complete URL addresses were truncated.)

The RTSP controls the session using commands, e.g. *SETUP*, *PLAY*, *PAUSE*, *TEARDOWN* etc. The list of commands supported by server can the client acquire by sending request OPTIONS to a server. This is usually the first message sent in RTSP communica-

[1] e.g. *rtsp://media.example.com:554/twister/audiotrack*
[2] *OPTIONS*, *GET_PARAMETER*, *ANNOUNCE*, *REDIRECT*. [21]

tion and the RFC describes, that it should be used.

The initialization of communication between a client and a server is shown on figure 3.1 including real, but simplified responses. For demonstration purposes, a communication between VideoLAN VLC player in role of both server and client was captured and illustrated in figures 3.1 to 3.4 showing the whole RTSP communication also with RTP and RTCP messages included.



```
SETUP rtsp://.../trackID=0 RTSP/1.0
Transport: RTP/AVP; unicast;
client_port=55036-55037
```

```
RTSP/1.0 200 OK
Transport: RTP/AVP/UDP; unicast;
client_port=55036-55037; server_port=56423-
56424; ssrc=6A781E79; mode=play
Session: b25d93def7d4d3aa; timeout=60
```

```
SETUP rtsp://.../trackID=1 RTSP/1.0
Transport: RTP/AVP; unicast;
client_port=47286-47287
Session: b25d93def7d4d3aa
```

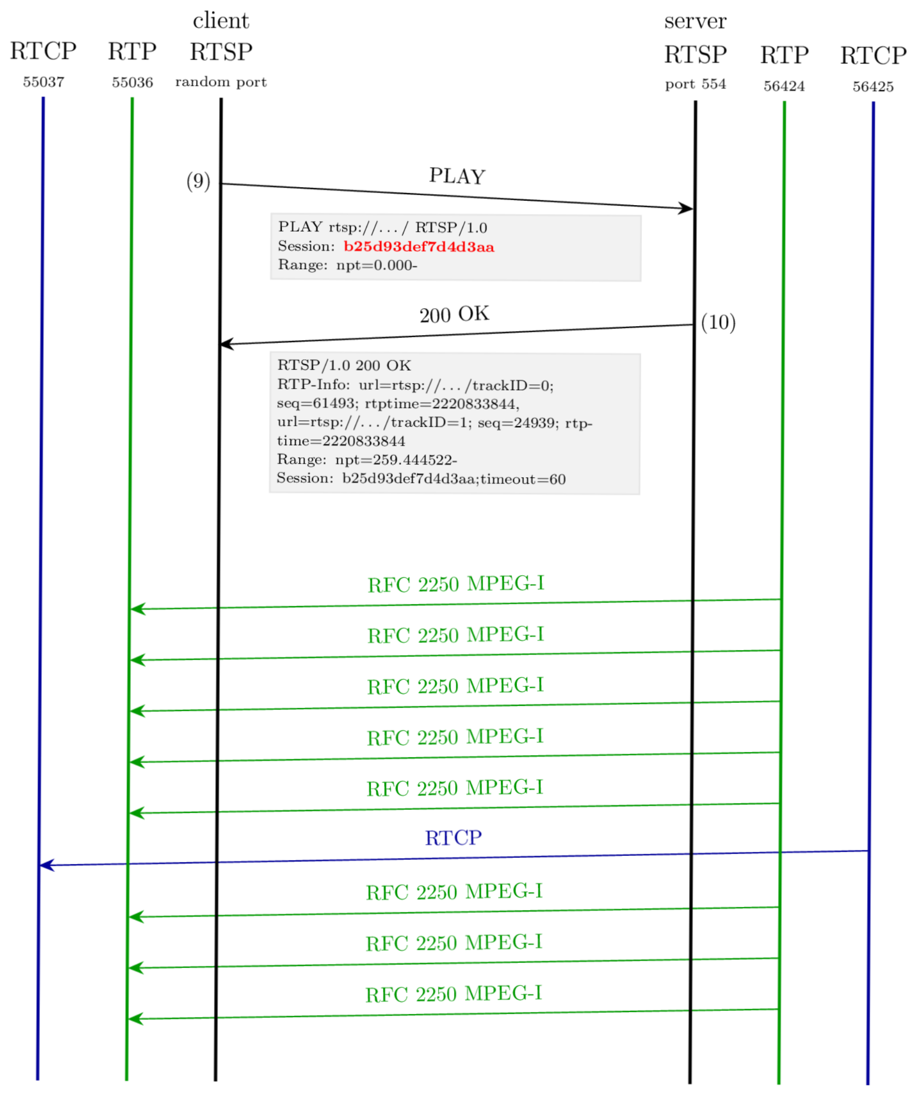
```
RTSP/1.0 200 OK
Transport: RTP/AVP/UDP; unicast;
client_port=47286-47287; server_port=56424-
56425; ssrc=77E2B192; mode=play
Session: b25d93def7d4d3aa; timeout=60
```

Figure 3.2: RTSP: Two SETUP requests together with responses.

After receiving the list of commands server supports, *DESCRIBE* together with URL is asked. As response, the description of a presentation or media object identified by the request URL is returned. For this purpose, the Session Description Protocol is typically used. The server has to describe all media initialization information.

An example of RTSP *DESCRIBE* response is shown on figure. 3.1 Here, two media streams (tracks) are described. The first one is audio payload type 14 (MPA with RTP clock 90000) with bit-rate 128 kbps, identified by *trackID=0*. The second one, identified by *trackID=1*, is a video stream with payload type 96 (dynamic payload, H.264 with clock 90000). More details about the SDP can be found in chapter. 3.4

Upon receiving a list of available media streams, the client can start requesting the streams. That is done by calling SETUP to server, separately for every stream (track) that the client wants to receive (figure 3.2 (5&7)). As part of the negotiation, client issues transport method, broadcast method, destination ports. The range of ports in fact represents couple of RTP and RTCP ports, lower and even for RTP, higher and odd for RTCP. A client can issue a *SETUP* request for a stream that is already playing to change transport parameters.



```
PLAY rtsp://.../ RTSP/1.0
Session: b25d93def7d4d3aa
Range: npt=0.000-
```

```
RTSP/1.0 200 OK
RTP-Info: url=rtsp://.../trackID=0;
seq=61493; rtptime=2220833844,
url=rtsp://.../trackID=1; seq=24939; rtp-
time=2220833844
Range: npt=259.444522-
Session: b25d93def7d4d3aa;timeout=60
```

Figure 3.3: RTSP request PLAY with response and start of transmission of RTP and RTCP packets.

(The RTP connection is established just after RTSP packet number 10.)

As response (figure 3.2 (6)), the server confirms transport method, broadcast method, cliend destination ports, and informs client about server source ports and SSRC of RTP stream. Most importantly, the server issues a new unique session identifier for the client. Starting from response (6), the communication is turned into stateful communication by attaching the session identifier to every request, and response.

When all the tracks are ready, a PLAY request can be sent. The session identifier, attached to the request, clearly identifies the client, and therefore the tracks, which were set up. A timestamp describing a start point as well as end point for the stream broadcast can be attached. Just after receiving the request, the server will start delivering the tracks via RTP on previously agreed ports.

For two streams issued by RTSP, together 5 ports has to be used for both client and

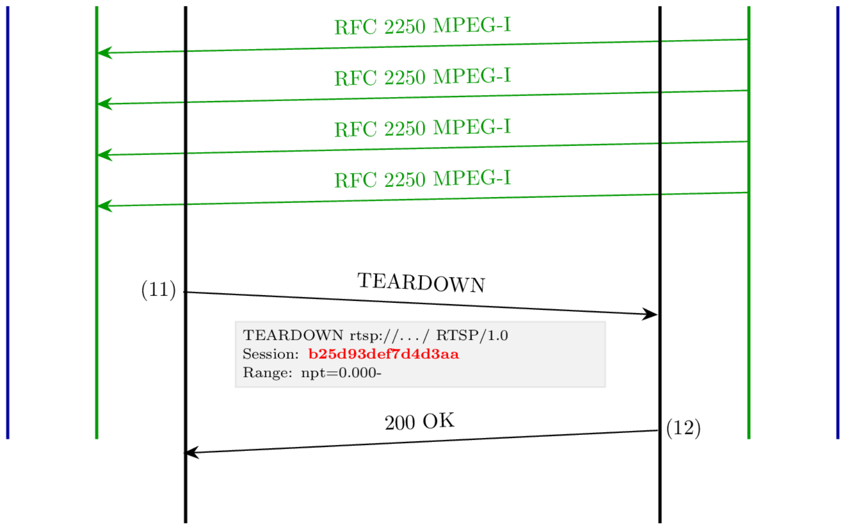


Figure 3.4: Termination of RTP/RTCP transmission by RTSP command TEARDOWN.

server. One for RTSP, two for RTCP and RTP for every stream. Even though the communication is always started by client, the RTP broadcast is established in direction from server to client. This is very unpleasant situation in case when the client is behind firewall or Network Address Translation (NAT). In such a case the RTP connection cannot be established.

On figure 3.3 an example of *PLAY* request together with RTP and RTCP traffic is outlined. Due to simplicity, only on RTP, and RTCP stream are shown.

The last important command to demonstrate simple RTSP session is *TEARDOWN* request. After this request, all stream broadcasts related to this session are terminated and the RTSP communication itself is closed.

## 3.4 SDP (Session Description Protocol)

The Session Description Protocols is a standardized protocol described mainly in the RFC 4566 [23], and its main and only task is to, as the name suggests, describe sessions. It is commonly used to describe multimedia in RTSP session, but it is meant to be generic.

The content is text-base dictionary of keys and values. For every line in the content, there is a key and a value in format $key = value$. The key, so-called type, is a case-significant character. The Session Description Protocol carries multiple information as shows table 3.5. For every SDP, there is one or more time descriptions, and zero or more media descriptions.

The Session Description Protocol is used when describing the media during DESCRIBE method of RTSP communication. It is crutial to identify what tracks are available for broadcast, and kind of media do these tracks carry. The media description informs about payload type (*m=audio 0 RTP/AVP 14*, where 14 means MPEG Audio), media bitrate (*b=AS:128*), information about clock rate (*a=rtpmap:14 MPA/90000/2*, where 90000 represents clock rate of 90 kHz) prior to the stream receiving (see table 3.5).

The RTP can carry many payload types, but only few have their own payload number [24]. The number represents encoding name, media type, clock rate and number of channels. For example payload number 14 represents audio encoded with MPA (MPEG Audio), with clock rate 90000 Hz. Numbers 96 to 127 are dynamic profiles. These profiles do not have any specific information about the media carried, so if RTP with payload type 96 is received, it is almost impossible to resolve, what is inside the packets. Because of that SDP is crucial, because it is the only source which can provide information about media content of the stream.

| **t** | **Example value** | **Description** |
|---|---|---|
| v= | protocol version | 0 |
| o= | originator and session identifier | ... |
| s= | session name | Unnamed |
| i=* | session information | N/A |
| u=* | URI of description | |
| e=* | email address | |
| p=* | phone number | |
| c= | connection information | IN IP4 0.0.0.0 |
| b=* | bandwidth information lines | |
| | | *(one or more time descriptions)* |
| t= | time the session is active | 0 0 |
| r=* | zero or more repeat times | |
| | | |
| z=* | time zone adjustments | |
| k=* | encryption key | |
| a=* | session attribute lines | |
| | | *(zero or more media descriptions)* |
| m= | media name and transport address | audio 0 RTP/AVP 14 |
| i=* | media title | |
| c=* | connection information | |
| b=* | bandwidth information lines | AS:128 |
| b=* | | RR:0 |
| k=* | encryption key | |
| a=* | media attribute lines | rtpmap:14 MPA/90000/2 |
| a=* | | control:rtsp://192.168.19.1/test01/trackID=0 |
| m= | media name and transport address | audio 0 RTP/AVP 32 |
| i=* | media title | |
| c=* | connection information | |
| b=* | RR:0 | |
| k=* | encryption key | |
| a=* | media attribute lines | rtpmap:32 MPA/90000 |
| a=* | | control:rtsp://192.168.19.1/test01/trackID=1 |

Table 3.5: An example of Session Description Protocol structure.
(Types marked with * are optional. If example value is empty, it was not even sent as part of the SDP.)

# Chapter 4

# Encoding, compression, and codecs

A codec is a computer program, whose task is to transform a media signal (e.g. a signal from video camera) to a binary code. For such a purpose, it defines an encoder, and a decoder. The task of encoder is to code the media signal to a binary code, while task of the decoder is reverse.

There is a big variety of codecs [25] and every codec has different main goal as well as advantages and disadvantages. This chapter discusses the most important aspects of audio and video codecs related to real time streaming and explains the impact of such properties on the whole streaming chain.

The source data for codec is an uncompressed stream. This stream has always constant **bit-rate**, which is defined by sampling period or resolution. Tables 4.1 and 4.2 outline bit-rates for different uncompressed video and audio streams. Such a high bit-rates can be unacceptable for transmission over network, so the data are compressed using a selected compression algorithm.

| Resolution | 30 FPS, 8b | 60 FPS, 10b | 120 FPS, 16b |
|---|---|---|---|
| HD *(1920×1080)* | 1.5 Gbps | 3.7 Gbps | 12.0 Gbps |
| 4K *(3840×2160)* | 6.0 Gbps | 15.0 Gbps | 48.0 Gbps |
| 8K *(7680×4320)* | 24.0 Gbps | 60.0 Gbps | 191.0 Gbps |

Table 4.1: Bit-rate of uncompressed video
(with different resolution, frame-rate and colour depth [26])

| Signal | 16 b | 24 b | 32 b |
|---|---|---|---|
| mono, 8000 Hz | 128 kbps | 192 kbps | 256 kbps |
| stereo, 44,1 kHz | 1411 kbps | 2117 kbps | 2822 kbps |
| 5.1 channel, 192 kHz | 18.4 Mbps | 1411 kbps | 36.9 Mbps |

Table 4.2: Bit-rate of uncompressed audio
(with different sample rate, bit depth and number of channels)

## 4.1 Multimedia compression

A **compression** is a process, which can be part of the codec encoder, which, using a special algorithm, encodes the information with fewer bits than the original representation is. [27]. Compressing can be lossy, which means that the quality of original information is degraded, or lossless. In case of lossless compression, the reduction of size is done by identifying statistical redundancies and eliminating them.

Compression of data is not a trivial process and the time needed to compress a block of data is perceptible. ITU G.114 explains that, in case of audio signal, if the latency „mouth-to-ear“ is kept below 150 ms, most of the users would not be significantly affected. [28] Moreover, Sven Ubik and collective adds, that for some special purposes even lower latency can be required. They state that the limit for live music cooperation is 35 ms. [29] In some special cases, like unmanned aerial vehicle video transmission, very low latency can be required. That is a reason why a lot of attention is paid to the process and why in the real-time media broadcast, both compressed and uncompressed data are used.

As an example, let us pick an H.264 codec, which is commonly used codec for real-time streams. As Schreier and Rothermel [30] state as well as Cast, Inc. [31] explains, the latency of H.264 codec can vary from fractions of $T_{frame}$ to units of $T_{frame}$. $T_{frame}$ represents multiples of frame periods and for example the value for 30 FPS video is 33.3 ms, and for 60 FPS, it is 16.6 ms. The total **delay of the processing** (algorithmic encoder latency as called in figure 4.1) can go up to 200 ms, for example when frame-based processing together with two pass encoding is used. That is far from being acceptable for real-time communication, especially bidirectional.

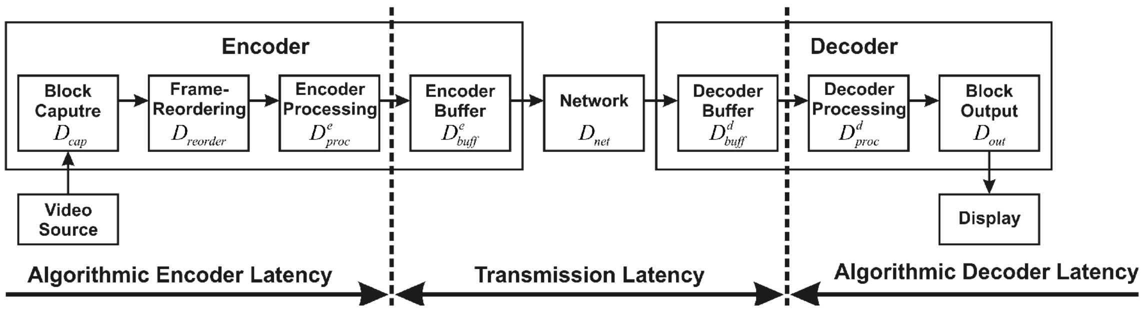

Encoder
Block Caputre $D_{cap}$
Frame-Reordering $D_{reorder}$
Encoder Processing $D^{e}_{proc}$
Encoder Buffer $D^{e}_{buff}$
Network $D_{net}$
Decoder
Decoder Buffer $D^{d}_{buff}$
Decoder Processing $D^{d}_{proc}$
Block Output $D_{out}$
Video Source
Display
Algorithmic Encoder Latency
Transmission Latency
Algorithmic Decoder Latency

Figure 4.1: Compression video transmission system delay sources [30]

For real-time streaming purposes, one pass or even special variants of codecs are used to minimize the compressing algorithm latency. For example for audio codec AAC (Advanced Audio Coded) a low delay variant, called AAC-LD, was derived [32]. To achieve higher frame-rates, quality and lower latencies, different hardware accelerated codecs, such as FPGA/ASIC units [29] or GPU acceleration [26], are being developed.

The most perceptible latency arises in the **encoder and decoder buffers**. As mentioned earlier in the previous section about codecs 4.2, even constant bit-rate streams does not carry all the frames with exactly the same size. As a consequence of that, the decoder buffer has to balance the incoming packets/frames. The amount of buffering required depends on the bit-rate and the averaging period of the stream. To make sure the decoder does not run out of data during playback, the decoder buffer must store all the data corresponding to one complete averaging period. [31]

The encoder buffer is introduced with similar purpose; it buffers frames or part of the frame during the compression. Depending on compressing methods (e.g. *Intra Coding*, *IP Coding*, *Frame-CBR Intra Refresh Coding*) the size can vary around 0.2 to 0.5 $T_{frame}$. Different situation is when IPB mode is used. Because B-frames are predicted from both previous and forward frames, the buffer has to store more frames for the calculations (2 frames for *IBP-mode*, 3 frames for *IBBP-mode*). This method is definitely not suitable for time critical encoding.

By using these optimization techniques, latencies around 20 ms for 30 FPS video and 10 ms for 60 FPS video can be reached.

Another solution, how to decrease latency of signal processing, is actually not processing it. The RFC 4175 [33] describes how an **uncompressed video** can be delivered using the RTP. The obvious advantage is the omission of the whole compression process, and that means saving time. Most importantly, raw uncompressed media stream will have truly constant bit-rate, and therefore the buffers can be nearly omitted as well. This way, latencies around 3 ms can be achieved using special hardware units. [29]

The down-side of this approach are certainly the bandwidth requirements. As the tables 4.1 and 4.1 show, the bit-rates of standard uncompressed media are high. Despite this fact, this approach can be used in local networks and even for international broadcasts. As Sven Ubik presented in May 2015 at the CESNET conference, they established a connection between Prague and London using uncompressed 4K video and audio with latency lower than 35 ms. [34][29]

All above described problems are important mainly in bidirectional communication. In case of unidirectional communication, most of the problems can be covered by using proper balancing buffers.

## 4.2 Variable vs. Constant Bit-Rate

A compression algorithm produces new data stream with different bit-rate based on input parameters. The streams can vary in the meaning of bit-rate variability. Either the bit-rate is constant within fixed amount of time, or the bit-rate varies from minimum to maximum depending on the media content of the stream and compression algorithm. Sending frames with variable data size over computer network has multiple issues discussed in this section.

Usually, the new stream is coded with **variable bit-rate** (VBR)[1]. The advantages of VBR are that it produces a better quality-to-space ratio compared to a constant bit-rate file of the same data. The bits available are used more flexibly to encode the sound or video data more accurately, with fewer bits used in less demanding passages and more bits used in difficult-to-encode passages.

The disadvantages are that the encoding may take more time, as the process is more complex, and that some hardware might not be compatible with VBR files. Variable bit-rate may also pose problems during streaming when the instantaneous bit-rate exceeds the

[1]VBR is available in Opus, Vorbis, MP3, WMA, AAC, MPEG-2 video, MPEG-4 Part 2 video, H.264 video, and many others

data rate of the communication path. [35]

The figure 4.2 shows a 1400 frames long cut of a movie. The part from frame 0 to 480 is ordinarily dynamic movie scene, then a dark cut (frames 450 - 480) is noticeable. From frame 480 a motion scene with a lot of visual changes between frames starts and continuous until frame 970. In this part, the bit-rate increases rapidly, which is clearly visible on the plot.

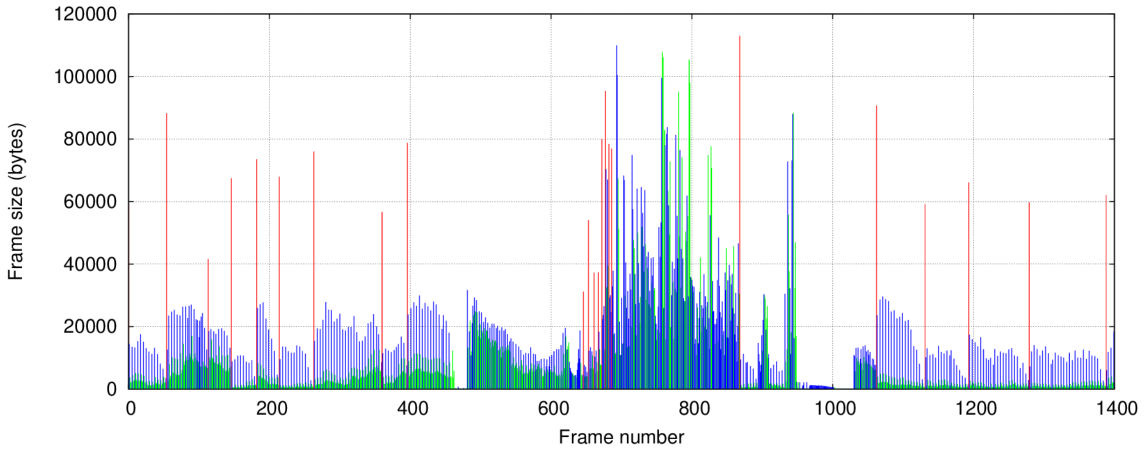


Figure 4.2: Variable bit-rate encoded file
(I-frames coloured red, P-frames blue, B-frames green.)

**Constant bit-rate** (CBR) is the second method how to compress data. As the name states, the target bit-rate is constant. This is rather misinterpretation, because in the real world, even this method has some obstacles. [31] Firstly, most coding schemes, such as Huffman coding or run-length encoding, produce variable-length codes, making perfect CBR difficult to achieve. [36] Without using such a coding, it is hard to achieve a great compression ratio.

Secondly, the CBR is calculated always within fixed number of packets or seconds. [31] Modern compress algorithms use I-frames, P-frames and B-frames. As figure 4.3 shows, the data size difference between I-frame, and P-frame as well as B-frame is enormous. That causes that transferring a I-frame over computer network takes more time, than delivering much smaller B-frame. And therefore the bit-rate is locally inconstant, which means that receiver buffers must be used as well.

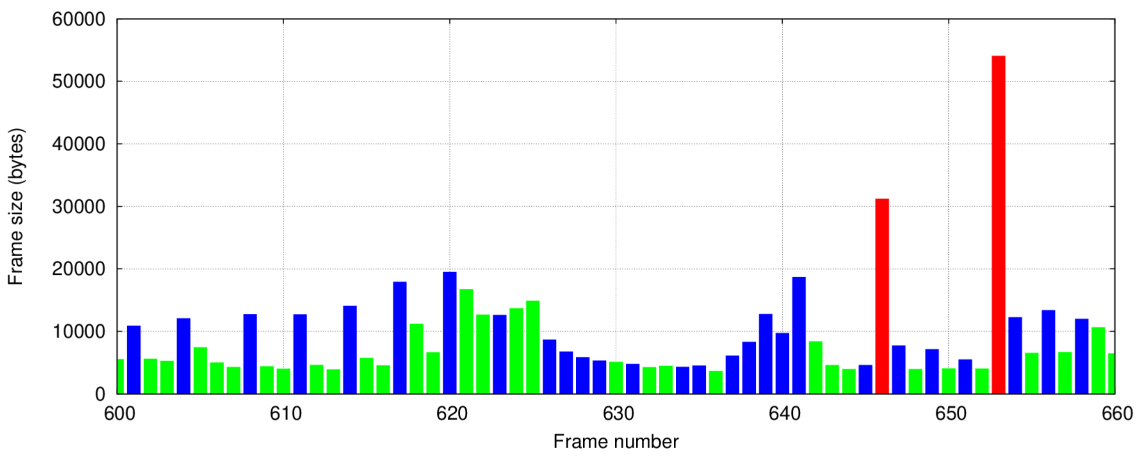
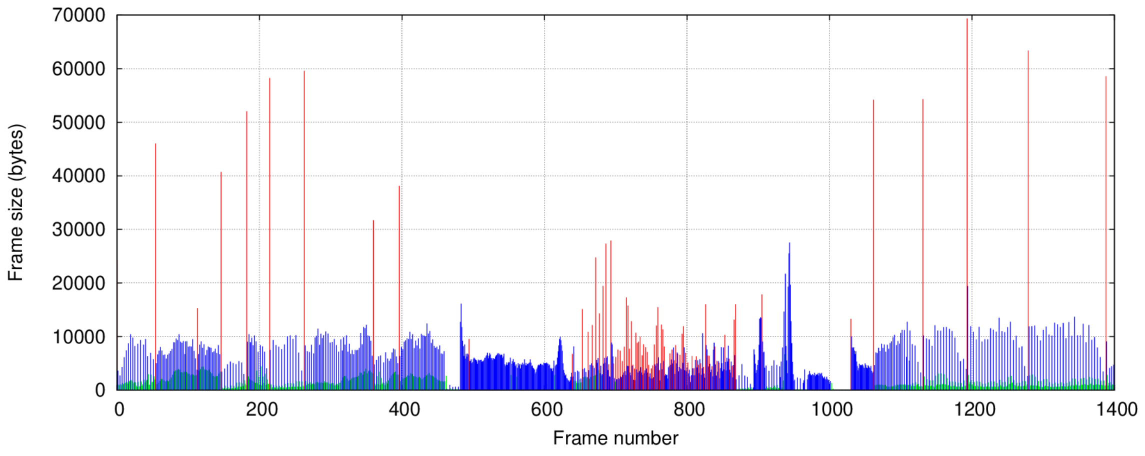
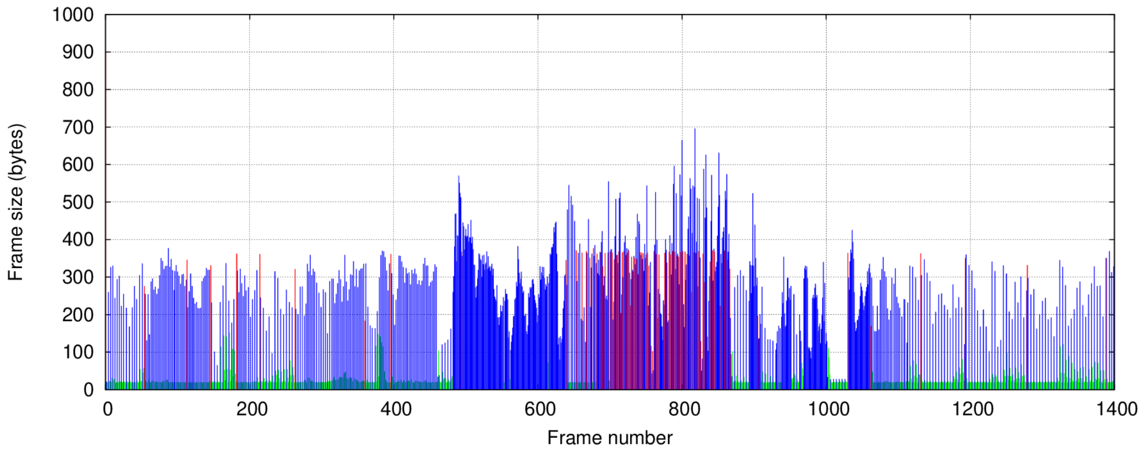


Figure 4.3: Difference in size between I-, P-, and B-frames
(I-frames coloured red, P-frames blue, B-frames green.)



(a) Converted as CBR 1 Mbps with 1000 kB buffer



(b) Converted as CBR 1 Mbps with 5 kB buffer

Figure 4.4: The same stream as on figure 4.2 converted using *avconv*
(Pay attention to the y-axis scale.)

# Chapter 5

# Analysis and design

Server applications are subset of standard applications with some additional principles. One of the key aspects of server applications is that they usually do not interact with users directly, using graphical user interface and the like. Usually they are based on client-server architecture, and for interaction with users and other systems, network communication is used.

Server applications act as services — they are controlled by start/stop requests and upon start they persistently run in background and serve requests. These requests typically represent task and are identified by some unique name, e.g. PHP scripts identified by URI in case of web-server. These tasks are launched according to the requests.

Typical feature of server applications is that they can serve more or many concurrent request and provide multi-user environment. They usually do not provide a control panel itself, so the administration is done using configuration files or separate graphical user interface. To provide ability to observe and examine the application behaviour, log files are typically provided. The log files provide extended summary of all actions taken with time relevant information, all error messages, and other information.

Server application has to be stable and error free program with one important feature: It has to be able to solve every error state, that can appear, itself. That means without asking user what action to take, e.g. by error message popup, and most possibly without shut-downing the whole service.

The application described in this thesis is meant to be a server application, so, with that in mind, it will be designed and developed. On figure 5.1 the very basic idea of how the application works is shown.

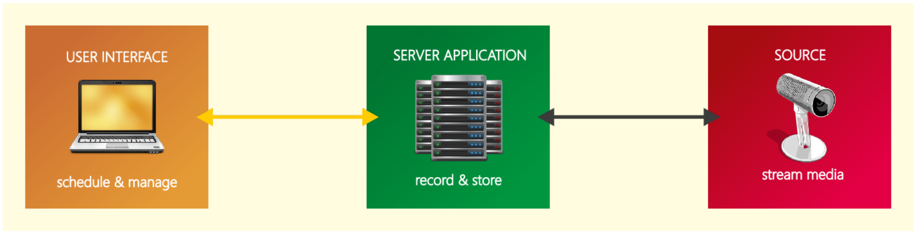


Figure 5.1: The basic idea of the recording system structure

The server application will be able to run as a service as a part of an operating system.

For this purpose it will be started using command *start* and it will provide a way how to stop it using command *stop*. This behaviour is common in Unix-based systems as well as in all Microsoft Windows operating systems. Further more, it can provide shell commands to check the status of the service, and, if needed, restart it. Some services provide configuration check during start-up, which is a great feature.

The application will run independently and perform tasks. These **tasks** will be defined as scheduled items. The items will tell what to record, when to start it, and when to finish the recording, as well as where to store it, and other relevant attributes needed to perform the task. The tasks will be stored in suitable storage, e.g. database or similar. More details about recording tasks can be found in next section [5.1].

For **communication** as well as input/output operations standard network communication will be used. The server application will receive multimedia streams from source cameras using standard protocols such as RTSP [3.3], RTP [3.1], RTCP [3.2]. **Concurrency** has to be provided, the application has to be able to execute more than one task at the time, which means to record two or more streams.

The **management** of the recording tasks and the application will be done using a graphical user interface, which will interact with the application using network protocols. The interface will provide suitable environment for user to perform administration tasks, as enumerated in section 5.2.

Moreover, the application will provide logging and will be able to solve every error state, especially network communication defects and problems.

## 5.1 Entity-based model

The state of the system and all data are stored in a database. These state and data are represented by entities. List of entities used in the system is shown in table 5.1. The whole system is entity-based, which means that every action is connected an entity. For example a *start of recording* is connected to entity *recording*.

| RECORDING SERVER | A server which is able to a record stream. This server represent one instance of running server application. |
|---|---|
| *hostname* | [PK] Unique hostname of the server. |
| *title* | User-friendly title. |
| *address* | Network-valid address used for communication. |

| STREAMER | A source of media stream, which can be recorded. |
|---|---|
| *id* | [PK] Unique id of the streamer. |
| *title* | User-friendly title. |
| *uri* | Valid uri including scheme allowing of the stream *(e.g. rtsp://10.2.0.52/media/video01)*. |

*Continued on next page*

Table 5.1 – *Continued from previous page*

| | |
|---|---|
| EVENT | One category for recordings, e.g. a school subject, room, or time event, and so on. |
| *id* | [PK] Unique id of the recording. |
| *title* | User-friendly title. |
| *folder* | Destination folder where recording files will be stored. |
| *startDate* | Time boundary for recording. No recording will start sooner. |
| *endDate* | Time boundary for recording. No recording will stop later. |
| *isEnabled* | Enabled/disabled flag. |
| RECORDING | One item in the schedule. A recording is a task, that the server application executes. |
| *id* | [PK] Unique id of the recording. |
| *eventId* | [FK] Foreign key to EVENT. |
| *streamerId* | [FK] Foreign key to STREAMER. |
| *serverHostname* | [FK] Foreign key to SERVER. |
| *title* | User-friendly title. |
| *folder* | Destination folder where recording files will be stored. |
| *startDate* | Moment, when the recording will start. |
| *endDate* | Moment, when the recording will stop. |
| *isEnabled* | Enabled/disabled flag. |
| *status* | Enumeration describing status of the recording *(e.g. ok, warning, error)*. |
| RECORDED FILE | Represents result of recording, a multimedia file stored in storage. |
| *id* | [PK] Unique id of the file. |
| *recordingId* | [FK] Id of the recording the file is related to. |
| *relativeFilename* | Filename of the file withing the recording folder. |
| *fileType* | Enumeration describing type of the file *(e.g. video, audio, log file)*. |
| *title* | User-friendly title. |
| *description* | Additional description of the file. |
| *isPublished* | Expresses wherever the file is accessible by users. |

Table 5.1: List of entities present in the system

## 5.2 Use cases

Defining what the application has to be able to do and what kind of request it has to be able to respond on, is crucial for the design. These actions or requests are defined by use cases and represent users' needs.

Use cases can be written in many ways, e.g. Cockburn style or using UML diagrams and so on. For our purpose, only simple table with list of actions, actors and descriptions will be sufficient to give enough data to start the design of the application. The table contains very basic use cases connected to the administration of the server. The actions are always related to an entity (see [5.1]). For readability reasons, the actions are aggregated in groups with the similar relationship to the entity and same actor.

These use cases describe actions how user or administrator can interact with management (only for the administrator role) or with file browser (the user role). Note that administrator role can be in fact an extension of role user. The actions such as *install server* or *configure server* are omitted, because they are not related to normal operation of the system.

| | |
|---|---|
| **Entity:** | RECORDING SERVER |
| **Actions:** | ADD, EDIT, DELETE |
| **Actor:** | Administrator |
| **Description:** | In case when the administration run separately from recording server application, the administrator must have a possibility to connect to a selected server and manage it. That is done by adding the server into list of managed recording servers. |

| | |
|---|---|
| **Entity:** | STREAMER (Recording source) |
| **Actions:** | ADD, EDIT, DELETE |
| **Actor:** | Administrator |
| **Description:** | Administrator can add a new recording source, e.g. an IP camera or other RTP relevant source, to the system. The ability to edit and delete such a record had to be provided. |

| | |
|---|---|
| **Entity:** | EVENT |
| **Actions:** | CREATE, EDIT, DELETE, ENABLE/DISABLE |
| **Actor:** | Administrator |
| **Description:** | Creating new event described by start and end date, folder to store data, and title. Ability to edit, delete and disable/enable the event. Disabling the event will cause disabling all its recordings. |

| | |
|---|---|
| **Entity:** | RECORDING |
| **Actions:** | CREATE, EDIT, DELETE, ENABLE/DISABLE |
| **Actor:** | Administrator |
| **Description:** | Creating new scheduled recording request defined by start and end date, source streamer, destination server. The server will start recording the source at the start date and will continue until the moment described as end date. Every recording belongs to an event. |

| | |
|---|---|
| **Entity:** | RECORDING SERVER |
| **Actions:** | MONITOR STATUS, VIEW LOGS |
| **Actor:** | Administrator |
| **Description:** | Giving the administrator ability to monitor server's status, see the CPU load of the recordings, used storage space, and view recording logs. |

| | |
|---|---|
| **Entity:** | RECORDED FILES |
| **Actions:** | DELETE, DESCRIBE, PUBLISH |
| **Actor:** | Administrator |
| **Description:** | After a recording, administrator must be able to delete, describe and publish recordings. Action describe means that some additional information can be attached to the recording; action publish means that the recording will be released for user's (public) use. |

| | |
|---|---|
| **Entity:** | RECORDED FILES |
| **Actions:** | VIEW |
| **Actor:** | User |
| **Description:** | Giving the user a possibility to view recorded and published files. |

Table 5.2: List of basic use cases in the management system

## 5.3 The application design

In the previous three sections, the basic design was described. Three separated parts were discussed: the logic of the server applications, the entities in the system, and the user use cases. This analysis leads to detailed scheme of the system:

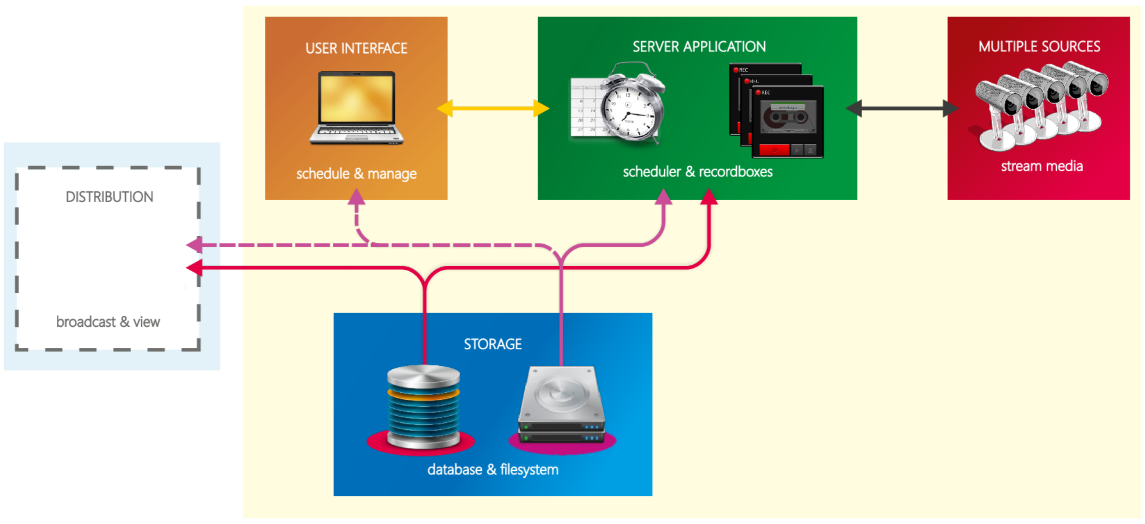


Figure 5.2: The advanced idea of the recording system structure

The figure 5.2 is virtually extension of the scheme shown on the figure 5.1. As mentioned earlier, the application requires file storage and database to store data. So, a new block representing the storage was added to the scheme. The database is required for every part of the system, whilst the file storage is more private. The **database** is used to store entities. As shared storage for all the components of the system, it is highly utilized.

On the other hand, the **file storage** is used to store video recordings and other files. Every server has its own storage, which is not shared between the servers. Only one server at the time writes to its own file storage, so no concurrency issues can appear. The data should be stored securely[1] and accessed only via services, that can provide authorization and authentication, e.g. well-secured static content web server.

Nothing new was introduced about the **user interface**. The management, as it can be called as well, will be an application providing summaries, forms and like to administrate the system. Analysis of its parts and implementation were left for separate chapter [7].

From what we can say now, the server application will consist of two the most important parts: scheduler and so-called recordboxes. The **scheduler** will be the part of the system responsible for timing the recordings and starting as well stopping the recordings. The information about timing will be fetched from the database. The scheduler has to handle all time-related problems to be able to work perfectly. Detailed analysis and implementation description can be found in section 6.2.

A **recordbox**, as it is called in this thesis, is second the most important part of the recording application. Its task is to provide functionality to record the broadcast. It will

[1]Keep on mind, that most recordings are sensitive data strictly protected by copyrights, laws, and so on.

act as a tape recording device, providing *start-stop-like* controls for the scheduler. The recordboxes will wrap all necessary functionality connected with the recording. That especially means running the external recording application, checking status of the application, creating recording folders, and so on. The timing related events will be left for the scheduler.

One big difference in this advanced design is, that it counts with multiple sources and multiple recordings at the same time. Basic concurrency is outlined here by showing, that there will be more than one recordbox per the server application. But one important topic was not discussed so far — **how many instances** of each block are possible, how many physical servers will it require, how many will it support, and in what environment will the whole system run.

As explained many times earlier, one instance of the server application will be able to record more than one stream at the time. Every recording requires (especially) processor time and file system I/O operations. That means the only limitation is the hardware of the server. Running too many recordings at the same time on one server can lead to slowing down of the server, which would lead to loss of the real-time data. And that means corrupting the recording.

With that in mind, the system is designed to be able to run many instances of the server application on different servers[1]. That provides **scaling** and can provide a way how to increase availability by duplicating number of nodes recording one stream.

As mentioned in previous paragraphs, every server will have its own file storage to store data, but only one **database** for whole system will be present. This is the only element in the system, that is critical and has to be run as unique instance. We can assume that the database engines are robust, designed with security in mind, and in many database engines clustering or at least failover clustering[2] is available. Properly set up database engine can provide great data security, so we can leave the most important part of the system running in one unique instance.

There is no limitation for the **management** system instances count. The management will actively write only to the database and send very basic request, such as status check, to the server application using *Representational State Transfer* (REST) communication. However, it is projected that the management system will be done as web application. In such a case we can predict, that only one instance of the web server will run giving only one management node in the system.

To close the topic about servers, it is necessary to say, that the system is split into separated parts to **be able** to provide scaling. It is not required to run all these applications on separate server instances. All software components can be easily, without modifications, installed on one operating system giving the 100% same operability.

It is important to add, that the system is designed to operate within **local area network** (LAN) only. That is indicated by yellowish rectangle around the blocks. That is because the system does not implement any extra security or encryption mechanisms to secure the communication enough to be held over the Internet.

The only part, that is meant to be used over the Internet, is the **distribution part**. This part of the system allows users to download or play the recorded videos as video on demand. This part of the system is not a goal of this thesis, so it will not be implemented.

[1] Although it would be possible to run more than one instance of the server application on one server, it does not really make sense, and therefore it is not supported.

[2] e.g. MySQL Cluster, MariaDB Galera Cluster, MS SQL Failover Cluster, etc.

Additionally, if we assume that the administration will be done as a web application, it can be used over the Internet. In such a case, it is necessary for the operator to provide some additional security mechanisms, especially authentication (e.g. HTTP basic authentication) and encryption (using e.g. SSL).

# Chapter 6

# Recording application

This chapter describes how the application is designed and how it will work. At the beginning of the design there was one more idea. The main question at the beginning of implementation was what technologies and programming languages to chose. Every language has its own benefits and with that in mind it is necessary to design the system.

On the other hand, some questions were already answered. The task of the recording application will be kept as simple as possible, focusing only on recording and support elements. Everything beyond this, especially administration and management, will be done separately. For storing data both filesystem (to store recording files) and relational database (to store other data) will be used.

As the server applications should provide concurrency, the recording application has to be designed which that in mind. For this application, threads will be used to split code and logic from the main thread, and create independent and concurrent workers.

The only supported operating system for the application will be Linux. Benefits of Linux as server operating system are well-known. Linus has great support, manageability, it comes for free and it is worldwide spread. The design of the application is theoretically operating system independent, but probably perceptible change would have to be introduced to the code to make the application running on e.g. Windows Server.

One of the possible options was to use some well-known scripting languages, such as Perl, Python or Bash. These languages are very powerful development tools and can provide all the functionality to create such a server application.

Thanks to my previous work experience, I had good knowledge on building server-side applications using Perl. One of the great benefits of Perl is that creating applications is really fast, and can be done with a few lines of code. By that time we were working on client application in C# and server application in Perl. The comparison of the length of the logic shown that Perl was much shorter.

One of the major disadvantages of Perl had appeared when the server-side application got larger and more complex. By that time, it began to be really hard to maintain the code and implement any larger changes or refactor the code. Perl lacks, as well other scripting languages, first stage compilation, which helps to check the code for the very basic mistakes. Instead of that, the code is checked only when it is executed, and that can lead to an error or an exception while application is running. And that is, of course, unacceptable.

Another approach was to use the scripting languages only to create small scripts handling the recordboxes. These scripts could be run by a scheduling service, e.g. Cron. This approach would be much easier speaking of code complexity and development demands, but also hardly extensible with some additional functionality, such as filesystem watcher [6.4] or system diagnostics [6.5].

With that in mind and after discussions with other programmers, I decided, from the whole spectrum of programming languages, to pick Java. One of the most important reasons was my knowledge of C# programming language, and knowing that development in Java as object-oriented language is in many aspects close development in C#.

Selection of database management system (DBMS) was much more straight forward. After determining that there is no need to use any of NoSQL or document oriented database systems, as well as temporal database or so, the focus came on ordinary relational database systems. In the end, MySQL was picked as one of the most commonly used DBMS. The biggest advantage is the fact, that it is for free.

Later, during the development, an idea how to increase the independence of each part of the system appeared. The question was how to equip every instance of the server application with its local copy of tasks, so it can execute the tasks independently without having the connection to the mutual main database server.

The possible answer was to use SQLite database engine to provide a data storage within the server application. In such a case, there would have to be a synchronization mechanism between the local database and the main mutual database. This principle is commonly used in mobile devices, such as Android. It is necessary to say, that the synchronization mechanism is not trivial and its implementation and testing might take a lot of time. And at the same time, the independence on such a level is not a goal of this work.

Programming language **Java SE 7** is used as the main language for the server application development. It has been selected due to its class-based object-orientation. The Java Runtime Environment, as well as Development Kit is available for free. And therefore, development and usage of the application is not tied with any licence fees.

**License:** GNU GPL *(OpenJDK)*

**MySQL** is a commonly used relational database management system. It uses Structured Querying Language, it is multiplaform, easy to deploy, well-documented, and well-known server. In community version, it comes for free. The version 5.5 was used as main database storage engine.

**License:** GNU GPL *(MySQL Community Edition)*

**Perl** is a high-level, interpreted, dynamic language. It was considered, as well as other interpreted scripting languages, as suitable programming language of the server application. Although Perl has assumptions to create a great stable server application, developing and maintaining the application can be harder than using Java. Eventually, Perl was not used in the project as well as the other scripting languages.

*Continued on next page*

Table 6.1 – *Continued from previous page*

| | |
|---|---|
| | **License:** GNU GPL |
| SQLite | As described in previous paragraphs, **SQLite** would be a great choice in case when the application would keep a local copy of all database data, and then provided synchronization with main database. This principle was not implemented, because it is probably not necessary in LAN environment. Moreover, if high-availability or fail-over mechanisms were required, MySQL cluster version could be deployed in order to achieve duplicity.<br>**License:** Public Domain |
| ubuntu | The whole server application with its dependencies is meant to run on Linux-like operating system. For development, Ubuntu 14.04 LTS (32-bit) was used. Due to chose programming tools, and other related tools, it is expected, that the application is able to run on any *NIX-like platform.<br>**License:** GNU GPL *(mainly)* |
| NGINX | The final goal of this thesis is to record and store streams from IP camera. This thesis does not discuss problematics of forthcoming video-on-demand delivery. For testing purposes **nginx** web server was used. It is great choice for static content delivery with great performance and low demands.<br>**License:** Simplified BSD |

Table 6.1: Tools and products used in the server application development

## 6.1 Threads

As it was described in chapter 5, the server applications should be able to serve more concurrent task at one time. With that in mind, it is necessary to chose a mechanism how to effectively achieve this goal. One of the commonly used methods how to divide tasks is using threads.

In the recording server application, one thread is created for every recording task, as described in section 6.3. Moreover, the application provides some additional logic such as filesystem watcher [6.4], and performance diagnostics [6.5] which run in separate threads to watch resources.

Due to the application design, every thread has it unique and simple task. The main logic of every single thread is as short main logic as possible, and usually at maximum a few programming classes long. That makes the behaviour of every thread readable and maintainable.

The threads self brings advantages to the application design and performance, but it is essential to design the application carefully with concurrency problems in mind. By using threads, the risk of concurrent access to a field or deadlock and others increases rapidly.

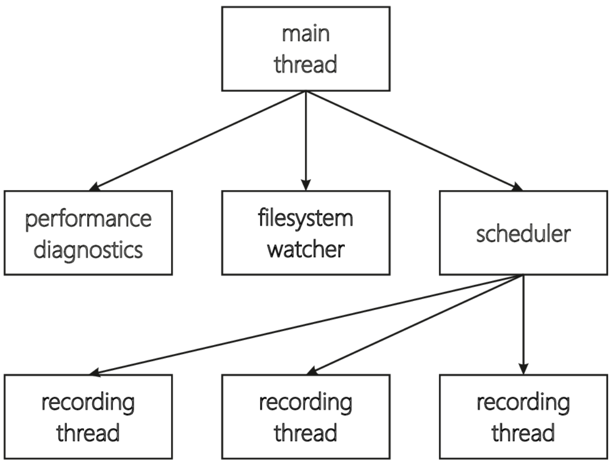


Figure 6.1: Scheme explaining threads usage in the recording application

It has been analyzed which parts of the applications are shared between more than one thread, and these parts were treated with mutual exclusion.

## 6.2 Recording scheduler

The scheduler is one of the crucial parts of the server application. Its task is to plan events inside the application, especially start of the recording and its stop time.

The scheduler is created from the main thread in single instance. It runs independently[1] from the main thread and periodically checks the necessary information from the database. In every cycle, it compares list of recordings that should be running (according to the database records) and recordings that in fact runs. Depending on that, it either creates new recordbox or stops running one.

The period of the scheduler, which is defined in a server configuration file, defines the minimum distinguishable interval for start and stop actions.

The recording scheduler has to be designer with all the time-relevant problems in mind. One of the ideas was to set up a timer that will elapse in precise moment to start (or stop) the recording. That moment is described by milliseconds of time. But unfortunately there is a lot of factors, that can lead to inaccurate timing, especially when inappropriate time representation is used (text representation instead of standardized UNIX timestamp).

The issues to be mentioned are:

- **Leap year** (nearly every four years February has 29 days);
- **Summer time** (once in a year there can be twice the same hour on clock; 25 hour long day);
- **Leap second** [37] (one-second adjustment occasionally applied, the most recent on June 30, 2015 at 23:59:60 UTC; called also sixtieth second);

[1]The scheduler is created in the main thread, but the periodical tasks are called in timer, which means in separate threads.

- **Time change** (in case of administration changes in system);
- **Time skew** (time correction mechanism [38]);
- **Shutdown or hybernation** (missing the proper moment due to system downtime);
- **Time-zone issues** (when the management is in different time-zone than the recording server, or not well set up);
- . . . and of course **real-time changes** in the database due to user interaction.

Eventually, it turned out that the best idea is simply periodically check the database with the most recent timestamp and take proper actions regarding the database results, rather then implementing some smart scheduling and timing mechanism.

As mentioned above, all the operations are related to the wall time. In such a situation, it is good practice to ensure correct time settings among all the servers involved in the system. For this purpose, correctly set up Network Time Protocol daemon (NTPd) is the solution.

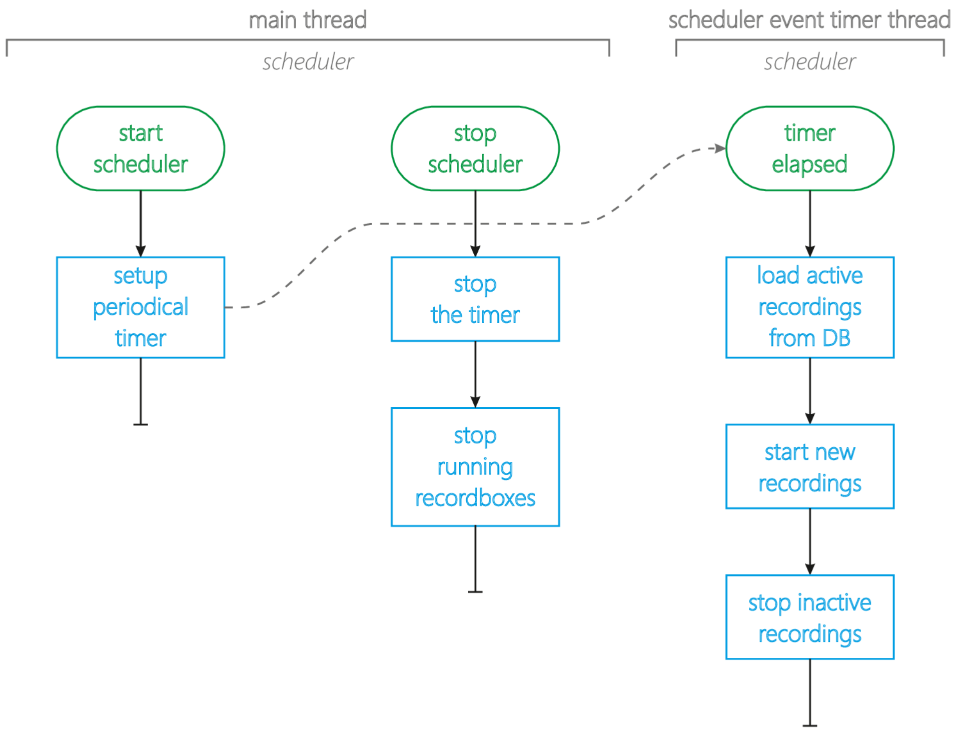


Figure 6.2: Flow chart showing the most significant methods of the scheduler

There was one more attitude to the development of the scheduler - to use something already created and proved. I concentrated my focus on Cron, the Unix system scheduler. Using this scheduler together with some start/stop scripts (e.g. written in Perl) could have been a solution.

Afterwards during examination of this idea, specific problems or restrictions of Cron usage appeared. One of the restriction is time granularity. The minimum interval for the Cron is one minute. That can be sufficient or not. But definitely cannot be changed.

Other problem, that might appear, could be problems with user rights in Linux. It has to be handled well in case of the Cron. But the most significant problem is how to feed the Cron with information about scheduled task, how to synchronize it with external database.

Moreover, the Cron is designed only to start task, not to be able to stop them at some moment. That means there would have to be an extra mechanism providing that. And providing synchronization mechanisms and other relevant things to achieve the goal.

At that point, I decided, that writing an own scheduler is easier than trying to use the Cron for such a purpose.

## 6.3 Recordbox

The name „recordbox“ is used for a part of the program that encapsulates the external recording process and creates standardized interface to control it. The recordbox itself is represented by set of classes. The main class (*recordbox*) is instanced from the scheduler and contains start and stop methods to control the behaviour.

The recordbox creates the external recording process and waits during its execution. For this purpose, a new recording thread (class *recordingthread*) is created to avoid suspending the scheduler thread during the recording. It is responsible for controlling and checking the behaviour of the external program.

If the program ends preliminary, the thread reports the situation to its parent recordbox and it has to take proper action. Depending on the occasions, it can restart the recording, which means to create a new recording thread, immediately or with a delay or even report the problem and stop recording itself (e.g. when the external program cannot be run).

The scheduler keeps list of all the recordboxes created, so it can stop them when necessary. The list of running recorboxes is compared to the list of scheduled recordings from database and if the recordbox' recording schedule is not in the list from database anymore, the recording is stopped. The reason to stop a recording does not necessarily have to be only due to its duration, but also if it has been disabled and so. This shows that checking the database periodically is better than timing the stop event by some timer or similar.

Most of the implementation is placed in a base class. This class carries common behavioural traits regardless used external program. For every different external program, there is a derived class overriding the original behaviour. This construction allows extending the application possibilities and range of used external programs. For example, a re-streaming application can be used giving the server application entirely new possibilities.

The figure 6.3 show flow chart illustrating basic callings and methods between *scheduler*, *recordbox* and *recordthread* classes. It also tries to point out where new threads are created.

The described part of the application uses timers in two locations. First timer is an interval timer, which invokes the *timer elapsed* method of class *scheduler* periodically. Every time, the timer creates new thread. This thread load data from database, listing active recordings. According to the list, as described earlier, starts or stops recording.

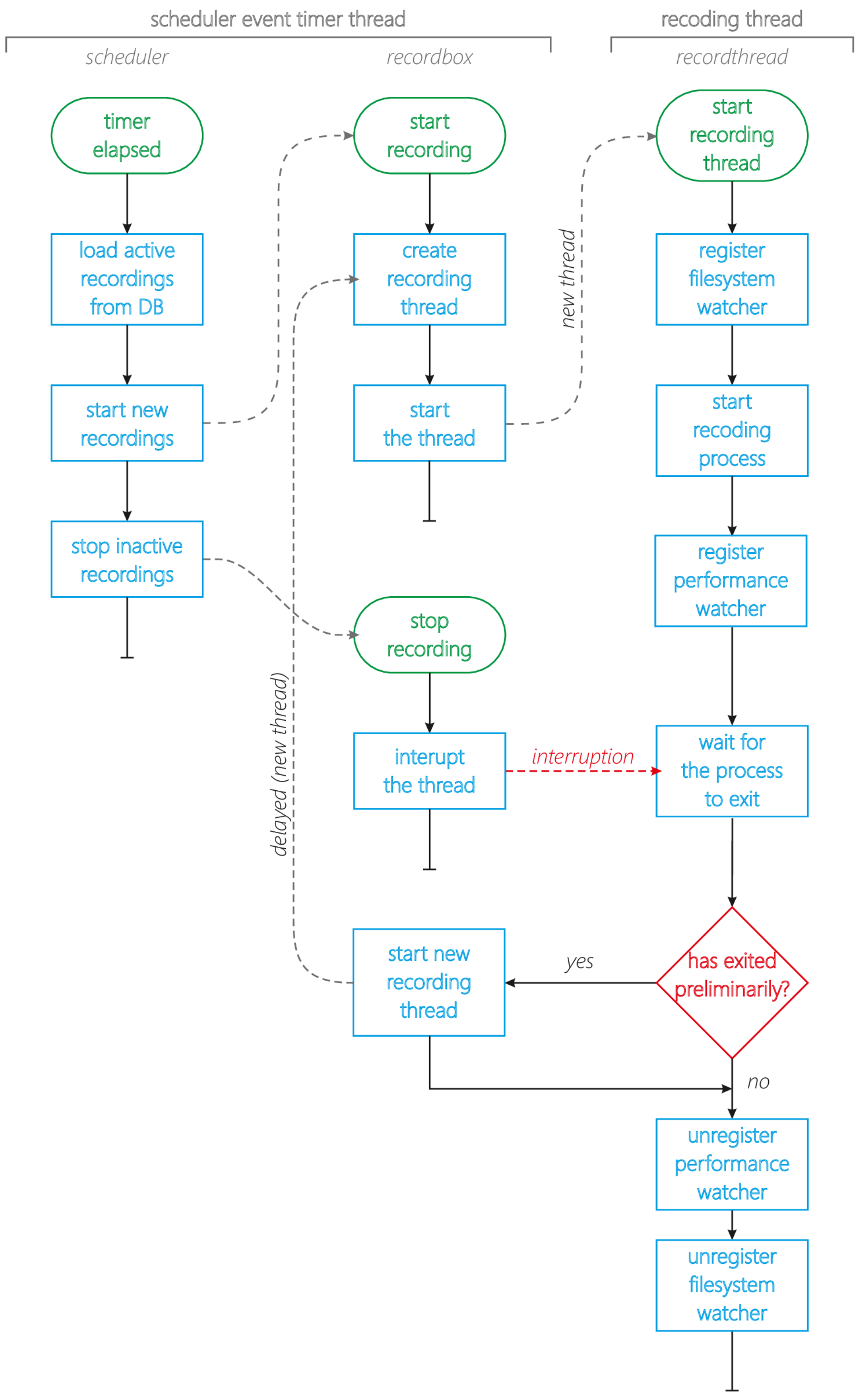


Figure 6.3: Flow chart showing recording process control

Second timer is used when external recording process end preliminary. The recordbox then defers the re-start using timer. When the recording is restarted, it is done in completely new thread.

The code also contains one explicit thread creating and that is the recording thread. The recording thread sets everything before the recording and disposes used sources after the recording. The thread especially takes care of recording folder creation, writing permissions check, registering the filesystem watcher and the performance watcher, and starting the recording process.

While the recording process is running, the recorded thread is suspended. If the recording process ends itself, it is evaluated as preliminary exit and it is handled by the recordbox. When the recording is supposed to be stopped by the scheduler, the scheduler invokes *stop* method of the recordbox and the recordbox interrupts the thread. The thread reacts on that signal by termination of the recording application, disposing sources and exiting.

**Libav** is a project consisting of libraries, and applications for handling multimedia data. It is multiplaform, officially supporting Linux, Mac OS X, Microsoft Winddows, and many others. Libav is able to handle many containers, and codecs. Most of them can read, decoded, encoded, and stored. Libav implements support for streaming protocols, such as RTP, RTSP, SDP, RTMP[1], and many others. Together with supported formats, and armament of processing filters, processors, muxers, demuxers, it is a powerful tool.

**License:** GNU LGPL, and GNU GPL

**Live555** is a set open-source project, which are focused on real-time broadcast. The core part, Live555 Streaming Media is a set of C++ libraries for streaming using RTP/RTCP, RTSP, and SIP. Live555 is used in many projects including VideoLan VLC player, and can be compiled under Linux, Windows, and any POSIX-complaint systems. The libraries contain only streaming functionalities, so it can be used for any compression, or similar modifications to the stream. At the moment, it natively supports MPEG Transport Stream, MPEG-1 or 2 Program Stream, WebM, MPEG-4 Video Elementary Stream, H.264 Video Elementary Stream, VOB (audio + video), DV video, MPEG-1 or 2 audio, WAV, AMR, AC-3, and AAC audio.

**License:** GNU LGPL

Table 6.2: List of studied recording applications

[1]Adobe related Real-time Messaging Protocol

## 6.4 Filesystem watcher

The filesystem watcher is a support logic of the application. It runs in separate thread, and it is initialized from the main thread. Its only task is to observe selected folders and report every change within these folders to event listeners. This technique is probably the only perfect way, how to detect what files were created as result of the recording in real-time. The other solution would be list the directory periodically, but since Linux operating system provides subsystem to get noticed on filesystem changes, it is the best idea to use that.

The subsystem is called Inotify and it is part of the Linux kernel since version 2.6.13 (August 2005). Its task is to notify listeners when an inode is changed. That practically means when a file is opened, closed, modified, moved, deleted, created and so on.

For development purposes, a library called *jnotify* was used. This library provides common interface to watch filesystem changes on Windows, Linux, and Mac OS X. Due tu its interoperability limitations, the part sustainable only for Linux was used. This library allows the filesystem watcher to announce when a file is created, opened for writing, and closed after changes.

There is only one instance of filesystem watcher running in the recording server application. It is created during application start-up, and it runs as separate thread. Other threads (especially recordboxes) can invoke method *addDirectory* to start directory observation. This is done just before the external recording program is executed, and the watched directory is the output directory for the program. After the recording ends, *removeDirectory* is called to stop observation. These methods are some thread-safe methods in the application.

The complete sequence of actions is following:

1. ***Recordthread*** creates *filesystem events listener* and registers folder for watching in *filesystem watcher.*
2. ***Recordthread*** starts *recording process.*
3. ***Recording process*** creates a file.
4. ***System kernel*** invokes *inotify* subsystem.
5. ***Inotify*** invokes *filesystem watcher* callbacks.
6. ***Filesystem watcher*** fires *recordthread listener.*
7. ***Recordthread listener*** identifies the file and saves information about file to the database.

## 6.5 Performance diagnostics

One of the extra functionalities of the system is performance diagnostics. It is an additional subsystem implemented in the recording server application, proving that running the system as one complex is better than running in as set of scripts (as discussed at the beginning of this chapter and in section 6.2).

At the moment, it has two main tasks. Firstly, it periodically checks the available amount of space of the recording partition. The motivation to implement this functionality came during first tests of the recording application. By that time, I left the application recording one IP camera for a week on provided testing server. After a week of recording, the application consumed all free space on the server, and forced the server to freeze due to lack of usable disc space. Fixing that problem took a lot of time.

Now, the application observers the filesystem, and shutdowns itself in case of overusing the free space. At the same time, the values of used space are shown to the administrator via web management.

Secondly, the diagnostics checks registered recording processes using internal Linux command *ps* and stores CPU and memory usage to the database. This can be handy to track what recording settings lead to what utilization of the server, and how many recordings can run on the server at one time.

Both these statistics are checked periodically. Periods are defined in configuration file, and can be different for every task. Results of the checks are stored in the database in ARCHIVE storage engine. If desirable, the values are plotted as a graph and shown to the user in the management [7.2].

# Chapter 7

# Web-based management

As it was previously mentioned in chapter 5, server applications usually do not provide any user interface to manage the application. For such a purpose, separated application is developed. This application connects to the server and manages it using administration protocol or changing its configuration files. These applications can implement another management functionalities, such as log viewers, server application diagnostics, monitoring, and like.

The applications can be console application, gui-based desktop application, or web-based application. To pick the right kind of management application, several questions have to be answered. How will be the management done; what network protocols if any will be used; what technologies does the management application require; is the graphical user interface necessary; if yes, how should it be done; and many others.

For our purpose to manage the recording server application, a web-based management system was chosen. One of the key aspects of selecting the web, was the fact it is multiplaform by design. And by saying multiplaform, not only various desktop operating systems are meant. Web-based applications can run on different devices such as desktop, tablets, mobile phones and others, making the management available practically everywhere. And that can be seen as on of the key usability goals of a good management application.

Another reason to use web environment is the fact, that it does not have to be installed on user's computer, and, more importantly, it does not have to be maintained on their computer. Upgrading the management system for whole company means upgrading it only at one point - the web server.

A downside of using web technologies can be its limitations. Some tools or features might not available for web, and that can mean the management cannot be done using web technologies. The recording server application was designed with that in mind. All the management tasks are stored in MySQL database, which is very common database storage in web environment. If it was necessary, the design also counted with usage of the Representational State Transfer (REST) communication to provide straight communication between the client web application and the server recording application.

For this reason, a small stand-alone prototype of Java REST server was created. It was base on library called Jersey providing the REST communication functionality. The purpose of the prototype was check up if the recording application can use the REST protocol to communicate with web management. As result, it was decided that it is well-usable communication mechanism. However, it was not implemented in the final recording application as it was not needed for any purpose. The whole communication is done passively[1] through the database.

Creating web-based applications has great benefit in how easily it can be delivered to the user. On the other hand it is necessary to master many programming languages, development technologies, tools, frameworks and other. For my development I spent quite a lot of time testing and choosing what technology to use. The tools used are described in the next table.

**HTML5** is new standard of HTML markup language. It has been released in October 2014, seven years after previous version HTML 4. It brings a lot of important improvements, which allows creating modern web-based applications. HTML5 was supported by major web browsers even during its development, which means it can be practically used these days. HTML5, for example, comes with native video player or support for custom tags, which are important for development of web-based applications.

**AngularJS** is a popular JavaScript framework maintained by Google and by its community. It is Model-View-Whatever[2] framework focused on developing single-page applications. In July 2015 it was used by 8.400 websites out of 1 million including NBC, Intel, or ABC News. [**wikiAngularJs**]

**License:** MIT License

Twitter **Bootstrap** is a front end framework for creating websites and web applications. It contains HTML and CSS based design templates for typography, UI elements, forms, dialogs and many others. For purpose of the management application a free template based on Boostrap called **AdminLTE2** was used.

**License:** MIT License *(both Boostrap and AdminLTE2)*

**Symfony** is a PHP web application framework. As difference from AngularJS, it is back end framework running the application code on server side. Symfony is widely used framework with great community. Originally the management application was written using Symphony, but later discarding all the work and switching to AngularJS, which is more convenient for client application development.

**License:** MIT License

*Continued on next page*

[1]Note that MySQL does not provide any notification mechanism when modification is done.

[2]Model-View-Whatever (works for you), see https://plus.google.com/+AngularJS/posts/aZNVhj355G2

Table 7.1 – *Continued from previous page*

| | |
|---|---|
| php | **PHP** is well-known interpreted dynamic programming language mostly used for web pages development. When the decision not to use Symfony was made, most of the application logic moved from PHP scripts to JavaScript front-end scripts. Still, PHP was used in the project for small server-side scripts handling model data for client application.<br>**License:** PHP License |
| Apache HTTP SERVER | **Apache HTTP Server** is one of the most commonly used web servers providing great support for scripting languages and CGI. The server management application was build and tested on Apache HTTP Server, although it is not anyhow limited to this product.<br>**License:** Apache License 2.0 |

Table 7.1: List of technologies, and software used during web development

## 7.1 JavaScript-based client application

Developing JavaScript-based client application is an individual programming category. For many years, the web was created by static pages generated by server side scripts, such as PHP. Of course these web sites could have carried JavaScript snippets providing additional client side scripting, but the possibilities were rather limited.

With introduction of AJAX, HTML5, and CSS3, possibilities how to create independent client-side web application become wider, and wider. Nowadays, several web framework provide abilities to create great Model-View-Controller (MVC[1]) client applications.

In the development of the management applicating, AngularJS was used. This framework focuses on creating single-page client application. That means, that the client loads only one HTML page, which contains the application logic. The application is then able to load other sources if needed, but the main page stays the same only changing its content.

In July 2012 the development team categorized the framework as MVW framework[2]. This was result of indecisiveness of wherever the framework is Model-View-Controller, or Model-View-ModelView. Humorously, it end up with label Model-View-Whatever (works for you) showing the reality behind development in AngularJS.

The model for AngularJS applications is usually strictly separated. The data are usually loaded as live data from web-server using AJAX requests, and held as objects in the client application. For the communication JSON is typically used to carry the data objects. AngularJS support two-way binding, which means the data are automatically propagated to the view once they are loaded.

The view is represented by HTML code with special tags representing AngularJS directives. These directives can describe model or controller for specific part of the page, declare basic behaviour (such as loops and conditions), but most importantly are used to print out

[1]MVC, or MVVM, or MVP, or MVW; depending on framework.

[2]https://plus.google.com/+AngularJS/posts/aZNVhj355G2

model values to the web page.

The last part of AngularJS trio is the application logic. It is written using JavaScript. Here AngularJS provides enormous functionality to create a responsible client-side application. In the fact many approaches to link the view, model and application logic can be used, leaving the decision to the programmer.

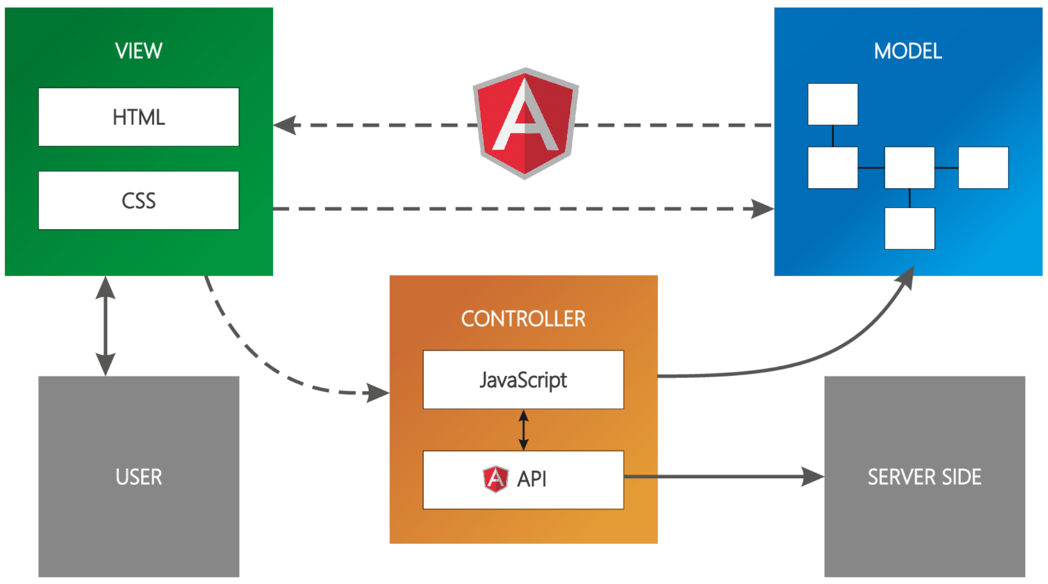


Figure 7.1: AngularJS Model-View-Whatever architecture

## 7.2 The user interface

The administration utilizes AdminLTE2 template created by Almsaeed Studio[1]. This template extends standard Bootstrap 3, and it is meant to be used to create various administration interfaces.

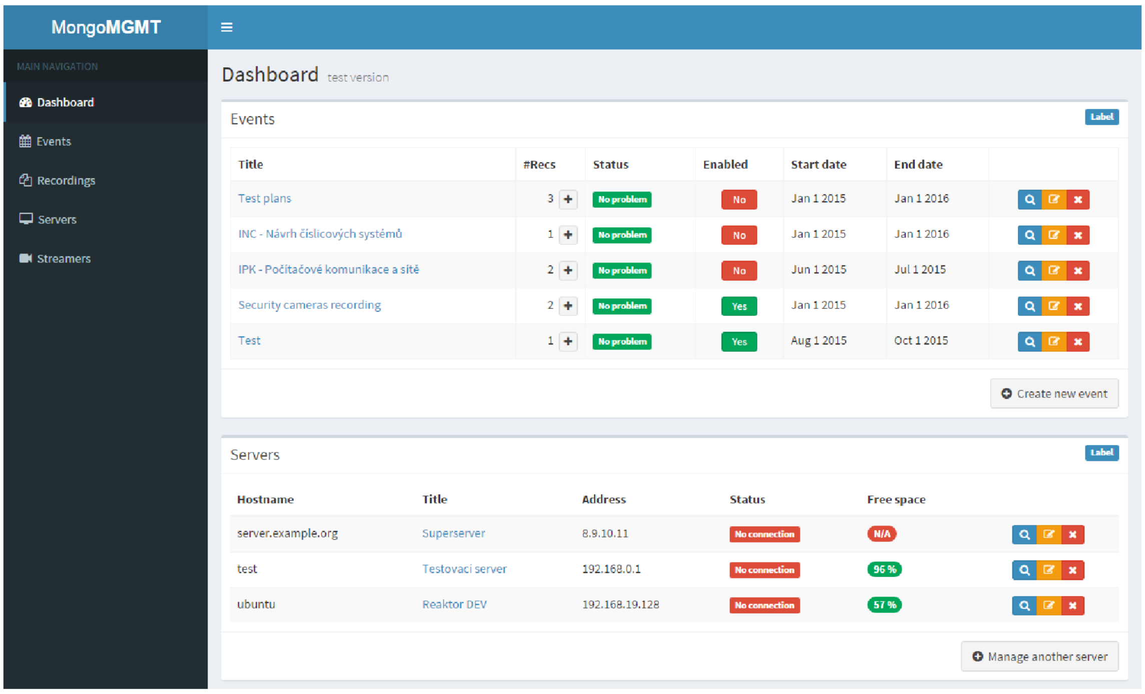
The template provides a lot of UI elements, widgets, skins and others. For example boxes, buttons, alerts, callouts, modal dialogs, tabs, progress-bars, calendar, charts, tables, form elements and so on. Everything is bundled with the template, and ready to use.

The site is divided into three main areas: side-bar menu, top-bar menu, and main content area. While top-bar menu is not used in this project, side-bar menu contains link to quickly access the most important actions of the web management.

The most important area is the content area, which changes with every page. This area is divided by boxes. Every box encloses one unit, and functionality, e.g. overview, list, calendar, form. Usually every box has loading overlay, which shows when the model data of the box are loading, or changing.

Most commonly, the lists of data from database, such as list of servers, events, recordings, and so on, are represented by tables. Every row represents one listed item. Typically,

[1]https://almsaeedstudio.com/, MIT License



| Title | #Recs | Status | Enabled | Start date | End date |
|---|---|---|---|---|---|
| Test plans | 3 | No problem | No | Jan 1 2015 | Jan 1 2016 |
| INC - Návrh číslicových systémů | 1 | No problem | No | Jan 1 2015 | Jan 1 2016 |
| IPK - Počítačové komunikace a sítě | 2 | No problem | No | Jun 1 2015 | Jul 1 2015 |
| Security cameras recording | 2 | No problem | Yes | Jan 1 2015 | Jan 1 2016 |
| Test | 1 | No problem | Yes | Aug 1 2015 | Oct 1 2015 |

| Hostname | Title | Address | Status | Free space |
|---|---|---|---|---|
| server.example.org | Superserver | 8.9.10.11 | No connection | N/A |
| test | Testovací server | 192.168.0.1 | No connection | 96 % |
| ubuntu | Reaktor DEV | 192.168.19.128 | No connection | 57 % |

Figure 7.2: The final management: The dashboard overview

when possible, for every item buttons for detail view, edit, delete and enable/disable are provided. The management is equipped with an in-place enable/disable feature allowing the user to enable, or disable events and scheduled plans by a single click.

One of the widgets used is a calendar plugin from Adam Shaw[1]. The calendar gives a view on the recordings as their are scheduled in time, as well as quick access to the events.

Another used widget is so-called timeline. It shows sequence of events in time sorted from newest to the oldest. This timeline is used to present newest events in the recording, especially inform about recently recorded files, or logs.

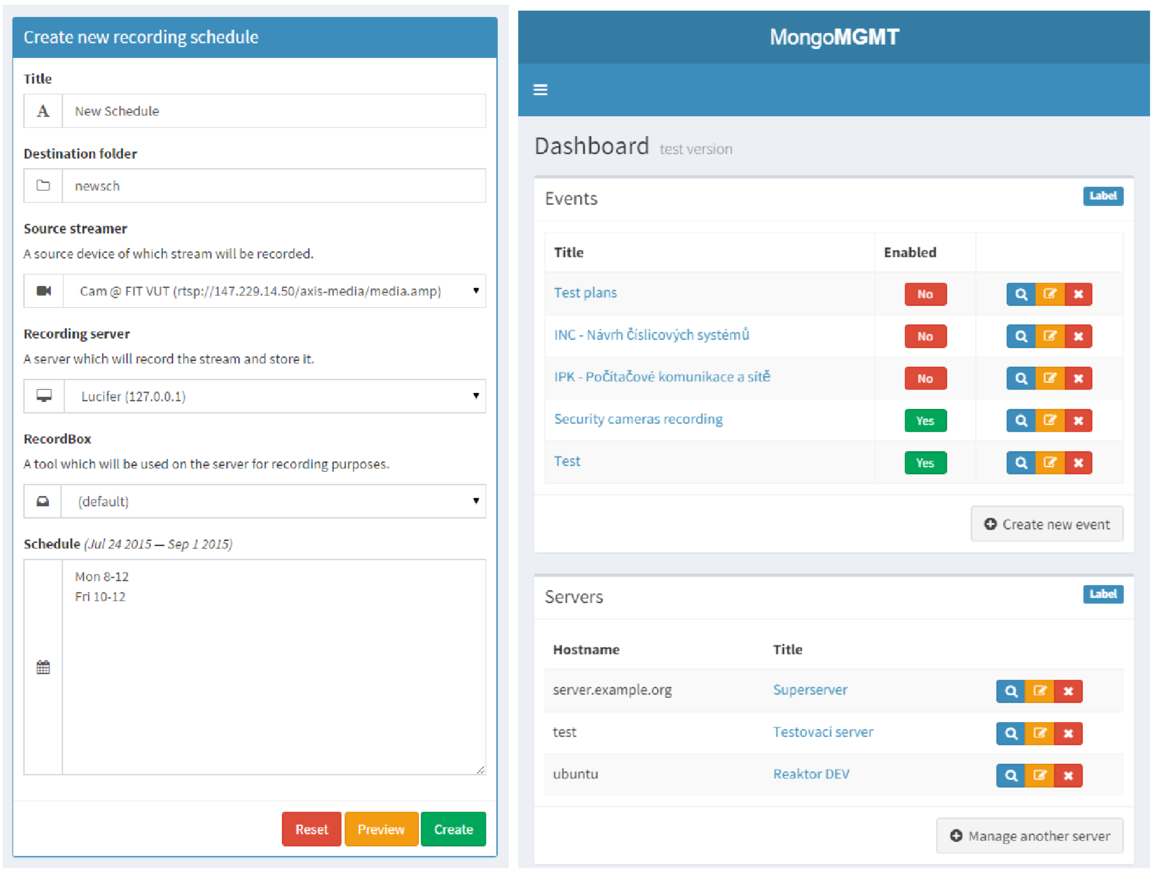
The management also utilizes Flot[2] graphs to show servers utilization, and free storage space (see section 6.5).

The forms in the management application are changed from original HTML forms in both design, and functional way. The design of elements is changed by bootstrap and AdminLTE2 templates, providing smooth, and clean graphics with the same result on all devices. Some additional graphical features are used to distinguish element's function: input fields are marked with icons, reset and save buttons are divided by colour, and so on.

The important part is the functional, how the forms work. Every form has a defined model, which binds value from the model to the inputs (if the data are edited), and after modifications back to the model. The submit button only invokes method *save*, which handles the model data. That means checks the validity, and, if they are valid, sends them using asynchronous request to the web server to store the changes in database.

During the transmission to the database, a loading overlay is shown to the user to indicate progress. After the data are stored, small box appears on the form telling the user wherever the action was, or was not successful.

[1]http://fullcalendar.io/, MIT License
[2]http://www.flotcharts.org/, MIT License


Create new recording schedule

Title

New Schedule

Destination folder

newsch

Source streamer

A source device of which stream will be recorded.

Cam @ FIT VUT (rtsp://147.229.14.50/axis-media/media.amp)

Recording server

A server which will record the stream and store it.

Lucifer (127.0.0.1)

RecordBox

A tool which will be used on the server for recording purposes.

(default)

Schedule *(Jul 24 2015 — Sep 1 2015)*

```
Mon 8-12
Fri 10-12
```

Reset Preview Create

MongoMGMT

Dashboard test version

Events

| Title | Enabled |
|---|---|
| Test plans | No |
| INC - Návrh číslicových systémů | No |
| IPK - Počítačové komunikace a sítě | No |
| Security cameras recording | Yes |
| Test | Yes |

Create new event

Servers

| Hostname | Title |
|---|---|
| server.example.org | Superserver |
| test | Testovaci server |
| ubuntu | Reaktor DEV |

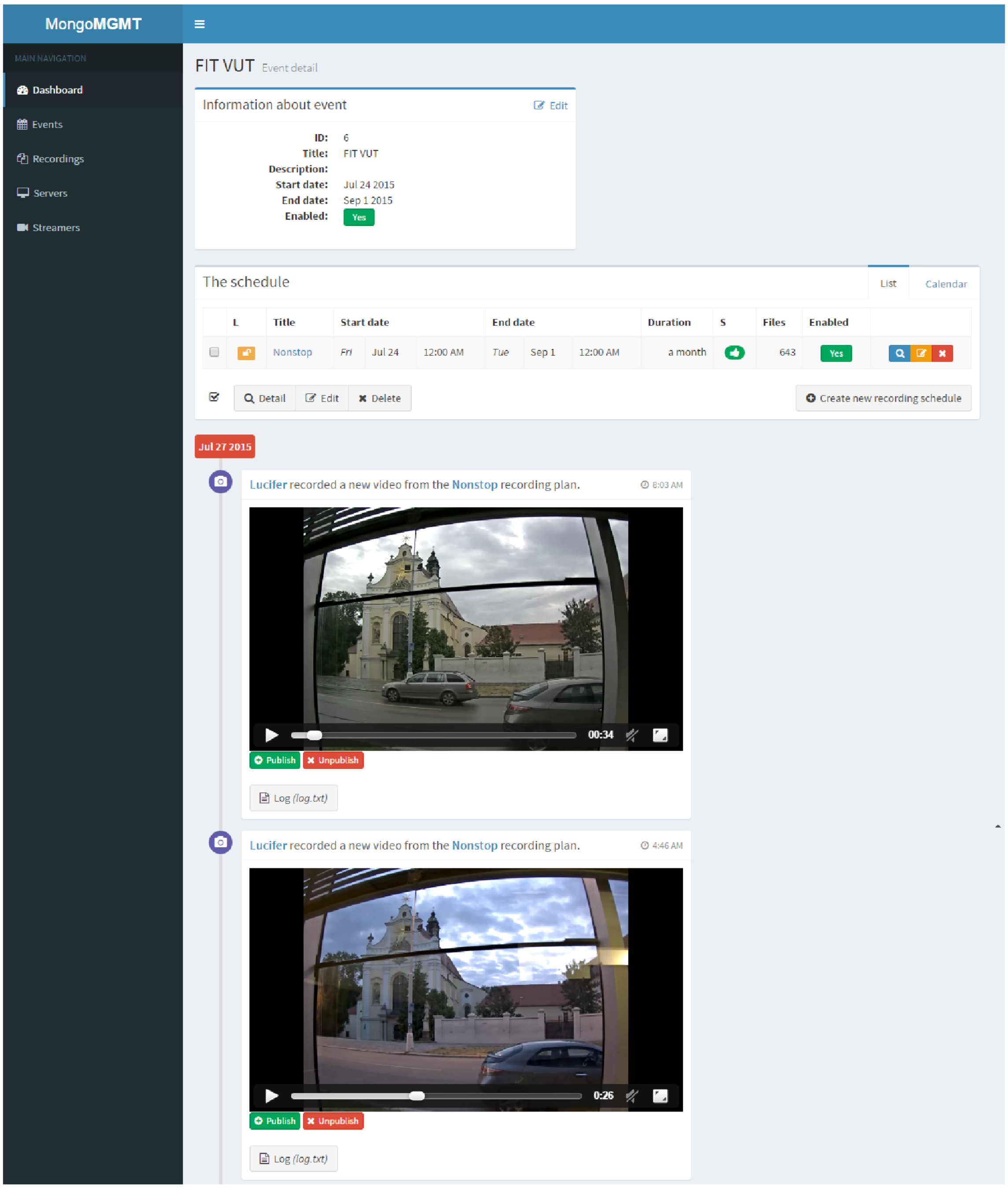
Manage another server

(a) The schedule edit form (b) The responsive mobile design

Figure 7.3: The final management screenshots

The web management is created as responsive web-page. That is mostly done by Bootstrap and AdminLTE2 templates, and their features, but, for example, in case of wide tables, it was necessary to apply another responsive design principles. The webpage was repeatedly tested on many devices, such as mobile phones, tables, laptops, and desktops.

Although it seems as part of the system, no user browser or video viewer was implemented with intend to distribute the video over the Internet. This problematic is much complex than this thesis can cover. Also, authentication system was omitted, leaving this to higher design principles, such as HTTP basic authentication.



FIT VUT Event detail

Information about event

| | |
|---|---|
| ID: | 6 |
| Title: | FIT VUT |
| Description: | |
| Start date: | Jul 24 2015 |
| End date: | Sep 1 2015 |
| Enabled: | Yes |

The schedule

| | L | Title | Start date | | | End date | | | Duration | S | Files | Enabled | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Nonstop | Fri | Jul 24 | 12:00 AM | Tue | Sep 1 | 12:00 AM | a month | | 643 | Yes | |

Jul 27 2015

Lucifer recorded a new video from the Nonstop recording plan. 8:03 AM

Log (log.txt)

Lucifer recorded a new video from the Nonstop recording plan. 4:46 AM

Log (log.txt)

Figure 7.4: The final management: The event overview

# Chapter 8

# Conclusion

This thesis was focused on multimedia streaming. It was split into two separate parts — the theoretical part and the practical part, both counting three chapters. The selected topic is very wide and it is connected to many other relevant field of studies.

Explaining the multimedia streaming is matter of point of view. If this thesis was written in computer networking related field of studies, most probably it would describe the protocols and surroundings much more deeply. My goal was to take the streaming as a tool, explain how does it work and how it can be utilized from practical point of view of stream delivery to end user.

Nevertheless, even from practical point of view, it is important to understand the mechanics behind the transmissions, and how these things affect the quality of delivery. That was the motivation to write chapter 2, and bring there to mind basic networking principles, and explain how these principles alter the quality.

Terms such as *throughput*, *network congestion*, and *multicast* were explained to point out, that delivering multimedia streams over computer network requires high bandwidth, and these three issues are related to it. It was explained how drastically can the network congestion decrease quality of service, and how multicast can decrease the required bandwidth for multiple instances of one broadcast.

Also, terms *latency*, *packet delay variation*, and *bit error rate* were described to elucidate how these networking-related circumstances affect stream delivery delay, which is very important especially in bidirectional communication, e.g. video-calls.

The section 2.2 then focused on two main transport protocols, explaining the main difference between *Transmission Control Protocol* (TCP) and *User Datagram Protocol* (UDP), and illustrating why UDP is used to carry media data packets over computer network.

The chapter 3 focused on the real-time streaming protocols. It extensively described the four most important protocols for real-time streaming, which are the *Real-time Transport Protocol* (RTP), the *Real-time Transport Control Protocol* (RTCP), the *Real-time Transfer Streaming Protocol* (RTSP), and the *Session Description Protocol* (SDP). It was shown how these protocols collaborate in communication, what is the task of every single protocol, what do the packets carry, what information can be acquired from the packets, as well as drawbacks of the design, such as problems with delivering the multimedia stream over Network Address Translation (NAT) device.

The main goal of chapter 4 was to pitch three important topics about multimedia. At the beginning of the chapter, terms like *encoding*, *decoding*, *encoder*, *decoder*, and *codec* were explained. Also, it was noted how a multimedia stream is coded before it is transmitted over computer network, and what are the bit-rates for uncompressed streams.

The section 4.1 extensively discussed the problem of frame processing delay in case of stream compression. As result, explanation on why low latency streams are important, and how the end-to-end latency can be decreased by using special codecs, hardware devices, GPU processing, and like, was introduced.

One of the delicate problems about media delivery is the size variation between frames. MPEG-based encoders use *I-frames*, *B-frames*, and *P-frames*. As the section 4.2 pointed out, the size difference between an *I-frame* and the other two is enormous. That means every compressed stream has locally inconstant bit-rate. The whole section discussed about this problematic, explaining what terms *variable bit-rate* and *constant bit-rate* mean, and showing a few graphs plotted from real streams.

The next three chapters spoke about the practical part of this thesis — analysis, design, and implementation of the recording system. Starting with chapter 5, very simplified idea of how the system should work was shown. It was described what is necessary for a server application to fulfil in order to work properly. Then the design was extended by explaining entities in this system, and user use-cases. The chapter is concluded with advanced design, which explained the system more in detail, and suggested realization draft.

Chapters 6 and 7 then described development of the recording application and web-base management system respectively. These chapters listed used technologies and software, and explained how the applications worked, how they were implemented and what are the principles behind the development. Details on how the used frameworks or programming languages work, were omitted on purpose. These can be found in programming articles, and I personally did not find meaningful to re-write such an information.

As it was mentioned, two applications are a result of this thesis. The applications were developed with intend to create stable and functional server application for recording. This purpose was achieved. The final application is able to run independently on a server and record streams described by schedule. A management application was created as well to provide manageability to the system.

Both application work, and the system as whole is functional. It was tested during development, as well as deployed at the end of the development on a server to see how well it works. The system worked as expected.

In the future it would be interesting to see the system in real production environment, with even the smallest flaws cleared out. The recording system could be extended by adding new *recordboxes* or improving the current one. The administration could go through user experience evaluation and based on that it would be nice to improve the interface and available tools in the system.

# Bibliography

[11] V. Cerf, Y. Dalal, and C. Sunshine. Specification of Internet Transmission Control Program. RFC 675. Internet Engineering Task Force, December 1974. URL: http://www.ietf.org/rfc/rfc675.txt.

[12] V. Paxson, M. Allman, J. Chu, and M. Sargent. Computing TCP's Retransmission Timer. RFC 6298 (Proposed Standard). Internet Engineering Task Force, June 2011. URL: http://www.ietf.org/rfc/rfc6298.txt.

[13] J. Postel. User Datagram Protocol. RFC 768 (INTERNET STANDARD). Internet Engineering Task Force, August 1980. URL: http://www.ietf.org/rfc/rfc768.txt.

[14] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard). Updated by RFCs 6096, 6335, 7053. Internet Engineering Task Force, September 2007. URL: http://www.ietf.org/rfc/rfc4960.txt.

[17] Audio-Video Transport Working Group, H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 1889 (Proposed Standard). Obsoleted by RFC 3550. Internet Engineering Task Force, January 1996. URL: http://www.ietf.org/rfc/rfc1889.txt.

[18] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (INTERNET STANDARD). Updated by RFCs 5506, 5761, 6051, 6222, 7022, 7160, 7164. Internet Engineering Task Force, July 2003. URL: http://www.ietf.org/rfc/rfc3550.txt.

[20] C. Huitema. Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP). RFC 3605 (Proposed Standard). Internet Engineering Task Force, October 2003. URL: http://www.ietf.org/rfc/rfc3605.txt.

[21] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). RFC 2326 (Proposed Standard). Internet Engineering Task Force, April 1998. URL: http://www.ietf.org/rfc/rfc2326.txt.

[23] M. Handley, V. Jacobson, and C. Perkins. SDP: Session Description Protocol. RFC 4566 (Proposed Standard). Internet Engineering Task Force, July 2006. URL: http://www.ietf.org/rfc/rfc4566.txt.

[24] H. Schulzrinne and S. Casner. RTP Profile for Audio and Video Conferences with Minimal Control. RFC 3551 (INTERNET STANDARD). Updated by RFCs 5761, 7007. Internet Engineering Task Force, July 2003. URL: http://www.ietf.org/rfc/rfc3551.txt.

[26] M. Liška. Speciální přenosy - technologie a jejich uplatnění [online]. Cesnet, z. s. p. o. May 2015. URL: http://www.cesnet.cz/wp-content/uploads/2015/04/Liska-CESNET-prenosy.pdf (visited on 07/08/2015).

[28] International Telecommunication Union. G.114. Telecommunication Standardization Sector of ITU, March 2003. URL: https://www.itu.int/rec/T-REC-G.114/en (visited on 07/08/2015).

[29] S. Ubik et al. Minimalizace zpoždění - přenosy pro spolupráci v kultuře [online]. Cesnet, z. s. p. o. May 2015. URL: http://www.cesnet.cz/wp-content/uploads/2015/04/Ubik-CESNET.pdf (visited on 07/08/2015).

[30] A. Rothermel R. M. Schreier. A latency analysis on h.264 video transmission systems [online], 2008. URL: http://publik.tuwien.ac.at/files/pub-inf_4978.pdf (visited on 07/08/2015).

[33] L. Gharai and C. Perkins. RTP Payload Format for Uncompressed Video. RFC 4175 (Proposed Standard). Updated by RFC 4421. Internet Engineering Task Force, September 2005. URL: http://www.ietf.org/rfc/rfc4175.txt.

[34] Cesnet. Hudebníky dělilo 1000 km, přesto spolu zahráli [online]. Cesnet, z. s. p. o. May 2015. URL: http://www.cesnet.cz/sdruzeni/zpravy/tiskove-zpravy/hudebniky-delilo-1000-km-presto-spolu-zahrali/ (visited on 07/09/2015).

# Other relevant articles

[1] S. Deffree. Tesla gives 1st public demonstration of radio, march 1, 1893 [online]. EDN Network. March 2015. URL: http://www.edn.com/electronics-blogs/nikola-tesla/4408090/Tesla-gives-1st-public-demonstration-of-radio--March-1--1893 (visited on 07/30/2015).

[2] C. Smith. By the numbers: 100+ amazing youtube statistics [online]. July 2015. URL: http://expandedramblings.com/index.php/youtube-statistics/ (visited on 07/30/2015).

[3] Google Inc. San bruno (youtube) [online]. Google Inc. 2015. URL: https://www.google.com/about/careers/locations/san-bruno/ (visited on 07/30/2015).

[4] J. Kužník R. Všetečka. Google vypadl jen na pět minut, ale provoz internetu klesl o 40 procent [online]. URL: http://technet.idnes.cz/vypadek-googlu-01t-/sw_internet.aspx?c=A130819_102205_sw_internet_vse (visited on 07/30/2015).

[5] OxfordDictionaries.com contributors. Skype [online]. URL: http://www.oxforddictionaries.com/definition/english/Skype (visited on 07/30/2015).

[6] Wikipedia contributors. Osi model [online]. URL: https://en.wikipedia.org/wiki/OSI_model (visited on 06/20/2015).

[7] Wikipedia contributors. Computer network [online]. URL: https://en.wikipedia.org/wiki/Computer_network (visited on 06/20/2015).

[8] Wikipedia contributors. Throughput [online]. URL: https://en.wikipedia.org/wiki/Throughput (visited on 06/20/2015).

[9] Wikipedia contributors. Network congestion [online]. URL: https://en.wikipedia.org/wiki/Network_congestion (visited on 06/20/2015).

[10] Wikipedia contributors. Bit error rate [online]. URL: https://en.wikipedia.org/wiki/Bit_error_rate (visited on 06/20/2015).

[15] Wikipedia contributors. Stream control transmission protocol [online]. URL: https://en.wikipedia.org/wiki/Stream_Control_Transmission_Protocol (visited on 06/20/2015).

[16] Wikipedia contributors. Real-time transport protocol [online]. URL: http://en.wikipedia.org/wiki/Real-time_Transport_Protocol (visited on 06/25/2015).

[19] Wikipedia contributors. Rtp control protocol [online]. URL: https://en.wikipedia.org/wiki/RTP_Control_Protocol (visited on 06/30/2015).

[22] Wireshark contributors. Rtsp [online]. Wireshark. April 2008. URL: https://wiki.wireshark.org/RTSP (visited on 07/09/2015).

[25] Wikipedia contributors. List of codecs [online]. URL: https://en.wikipedia.org/wiki/List_of_codecs (visited on 07/08/2015).

[27] Wikipedia contributors. Data compression [online]. URL: https://en.wikipedia.org/wiki/Data_compression (visited on 07/08/2015).

[31] Inc. Cast. White paper: understanding and reducing latency in video compression systems [online]. Cast, Inc. October 2013. URL: http://www.cast-inc.com/blog/white-paper-understanding-and-reducing-latency-in-video-compression-systems (visited on 07/08/2015).

[32] N. Fazlija. Improved mpeg low-delay audio coding on davinci and ti c64 series dsps [online]. Fraunhofer-Gesellschaft. March 2007. URL: http://www.ti.com/lit/ml/sprp526/sprp526.pdf (visited on 07/08/2015).

[35] Wikipedia contributors. Variable bitrate [online]. URL: https://en.wikipedia.org/wiki/Variable_bitrate (visited on 07/08/2015).

[36] Wikipedia contributors. Constant bitrate [online]. URL: https://en.wikipedia.org/wiki/Constant_bitrate (visited on 07/08/2015).

[37] Wikipedia contributors. Leap second [online]. URL: https://en.wikipedia.org/wiki/Leap_second (visited on 07/20/2015).

[38] D. L. Mills et al. Ntpd(8) - linux man page [online]. December 2009. URL: http://linux.die.net/man/8/ntpd (visited on 07/08/2015).

# Appendix A

# List of Figures