Building IPTV streaming service using commodity hardware

***Abstract*—The purpose of this document is to get acquainted with the theory and practice behind IPTV streaming services. We hope that the knowledge obtained during the study will be essential for those who will be maintaining and troubleshooting small scale IPTV infrastructure.**

We start this short document with the discussion of the theory behind digital video broadcasting services, including various modulation schemes, audio and video codecs, error correction algorithms, and other related issues. We then move on to the discussion related to design and implementation of a simple IPTV streaming service using commodity hardware. Finally, to understand the scalability of the proposed architecture, we will perform a set of simple experiments (the experiments will mostly focus on microbanchmarking of various components of the system).

I. Introduction

IPTV today is one of the most convenient way to view live TV channels. Not only it is accessible wherever the Internet access exists, but also it gives a possibility to archive TV shows so that they can be viewed afterwards on demand. IPTV also gives a possibility to build feature reach TV broadcasting services. For example, the TV viewing experience can be augmented with various quiz's, feedback, and even context advertisements.

In this document we will discuss how to build a simple IPTV service using commodity hardware and open source software only. But before we dive into discussion of the architecture, we will first elaborate on theory behind the service. Thus, we will give an overview, on a high level, for mathematics behind digital video broadcasting (DVB) standard, discuss various standards including video encoding, error correction and modulation schemes.

With this respect, this paper is organized as follows. First, we will cover some background material. Here we will discuss the details of three DVB standards, then we will move on to theory related to modulation and error correction techniques. Then, we will overview MPEG and video coding standards. And finally, we will overview HTTP live streaming protocol (HLS) and devote few paragraphs to aspects related to security in HLS and DVB standards.

II. Background

In this section we will review basics of the video broadcasting protocols, modulation algorithms, error correction schemes, video encapsulation formats, audio and video codecs, as well as security mechanisms, which are needed for protection of video streams from eavesdropping by malicious parties.

A. DVB standards

Today there exist three major standards which are used for delivering live video content to customers: digital video broadcast over cable (DVB-C), terrestrial digital video broadcast (DVB-T), and digital video broadcast over satellite (DVB-S). We will review these standards, as well as their modifications, so-called second generation broadcasting protocols, in the given section.

DVB-C

DVB-T

DVB-S

Second generation digital video broadcast protocols

B. Modulations

On high level an information signal, before being transmitted over the medium, such as cable or air, needs to be modulated (combined with carrier frequency) in some way. There exist multiple modulation schemes, examples are frequency modulation, amplitude modulation, phase modulation, and their combinations. In this section, in addition to already mentioned, we will review two additional modulation schemes: quadrature phase shift keying modulation (QPSK) and quadrature amplitude modulation (QAM).

The most simple modulation is *amplitude modulation*. It can be described in the following manner. If the carrier signal is represented as $c(t) = A_c \sin(2\pi f_c t)$, where A_c is the peak value of amplitude of the carrier, f_c is the carrier frequency and t is time; and the modulating signal is represented as $m(t) = A_m \sin(2\pi f_m t)$, where the A_m is the peak value of the amplitude of the modulating signal, f_m is the modulating frequency, then the modulated signal can be described as $y(t) = (A_c + A_m \sin(2\pi f_m t)) \sin(2\pi f_c t) = A_c(1 + M\frac{1}{A_m}m(t))c(t)$, where $M = \frac{A_m}{A_c}$ is the modulation index. Simplifying[1] the equation we get $y(t) = A_c \sin(2\pi f_c t) + \frac{A_c M}{2} cos(2\pi(f_c - f_m)t) - \frac{A_c M}{2} cos(2\pi(f_c + f_m)t)$, in other words the modulated signal (for single tone modulating signal) consists of three harmonics: central frequency, and two side-bands. In Figure 1 we show how amplitude modulated signal will look like when a single tone modulating signal (with frequency equal to 2Hz) is combined with 100Hz carrier signal. The sidebands in this case will be signal with frequency 98Hz and signal with frequency 102Hz. Now coming back to the modulation index: typically, it should be well above the zero, but also less than or equal to one. A modulation index larger than one indicates the over-modulation situation. Such signal cannot be demodulated and therefore no intelligence can be recovered

[1] http://www.tofmal.ru/projects/trigan/page5.html

from it. The ideal condition is when $A_c = A_m$, which gives 100% modulation. This situation results in the greatest output power at the transmitter and the greatest output voltage at the receiver, with no distortion [].

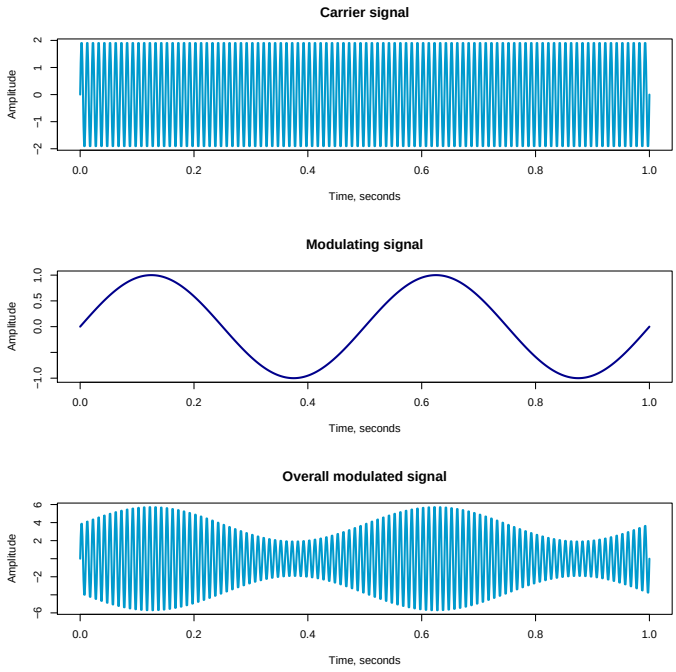


Fig. 1: Amplitude modulation

Other simple examples of modulation are Amplitude Shift Keying (ASK) and On-Off Keying (OOK). In amplitude shift keying only single carrier frequency is present, but the amplitude is changed from time to time. With this modulation it is easy to encode binary message: a signal with high amplitude represents the binary 1, and signal with low amplitude represents binary 0. In on off keying modulation is achieved by simply turning the carrier on and off. Example of such modulation is Morse code: the dot is represented with a short burst of carrier, and dash is modulated with long burst of carrier signal. Code transmission such as this are usually called *continuous-wave (CW) transmissions*.

In frequency modulation the frequency is modified for the purpose of transmission of intelligence signal. Unlike the amplitude modulation, in frequency modulation the amplitude remains constant and only the frequency of the carrier is being changed. In frequency modulation the carrier frequency changes proportional to the changes of the voltage of the intelligence signal: that is, as the modulating signal voltage increases, the carrier frequency also increases; as the modulating signal's voltage decreases, the carrier frequency also decreases.

Another way to modulate signals is to shift the phase of the carrier signal depending on the values of the intelligence signal. These modulations are known as phase shift keying (or PSK). Examples are Binary Phase Shift Keying, Quadrature Phase Shift Keying (QPSK) and Quadrature Amplitude Modulation. It is perhaps easier to understand these modulations with the following example. Lets consider a phase modulated signal $y(t) = A\cos(2\pi f_c t + \varphi)$. Using trigonometric identity[2] we can derive $A\cos(2\pi f_c t + \varphi) = A\cos(2\pi f_c t)\cos(\varphi) - A\sin(2\pi f_c t)\sin(\varphi)$. Now substituting $I(t) = A\cos(\varphi)$ and $Q(t) = A\sin(\varphi)$, we get $A\cos(2\pi f_c t + \varphi) = I(t)\cos(2\pi f_c t) - Q(t)\sin(2\pi f_c t)$. Here $I(t)$ represents the in-phase part of the signal, $Q(t)$ represents the quadrature part of a signal. By carefully selecting $A = \sqrt{2}$ (knowing that $\varphi \in \{\frac{\pi}{4}, \frac{3\pi}{4}, \frac{5\pi}{4}, \frac{7\pi}{4}\}$), $I(t)$ and $Q(t)$ will take values $\{-1, 1\}$. This can be easily seen from IQ diagrams (also called constellation diagram, or phasor diagram), which we show in Figure 2. So, for example, if we need to encode the binary message `00`, the resulting signal can be described with the following equation $y(t) = I(t)\cos(2\pi f_c t) - Q(t)\sin(2\pi f_c t) = \cos(2\pi f_c t) - \sin(2\pi f_c t)$. The signal will have $\frac{\pi}{4}$ phase shift relative to in-phase signal (see Figure 3 for reference).

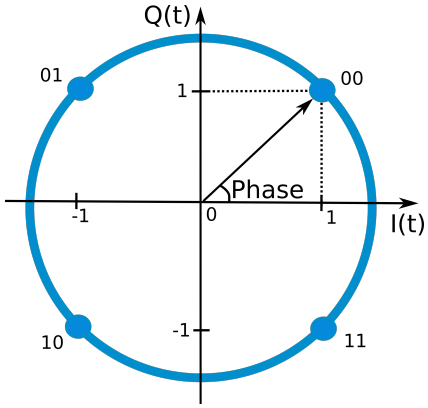


Fig. 2: Constellation diagram

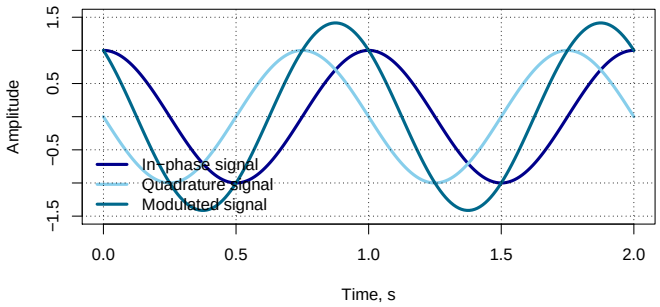


Fig. 3: Example of IQ signal

In Figure 4, we show how the binary message `00 01 11 10` will be modulated using 5Hz carrier signal.

To perform demodulation, modulated signal should be multiplied by $cos(2\pi f_c t)$ to obtain the $I(t)$ component, and by $-sin(2\pi f_c t)$ to obtain the $Q(t)$ component of the signal. Both signals then should be passed through low-pass filter (or exponentially weighted moving average function) so that the original binary message can be recovered (the values of $I(t)$ and $Q(t)$ will be selected based on the sign of a sample at

[2] www.ni.com/tutorial/4805/en/

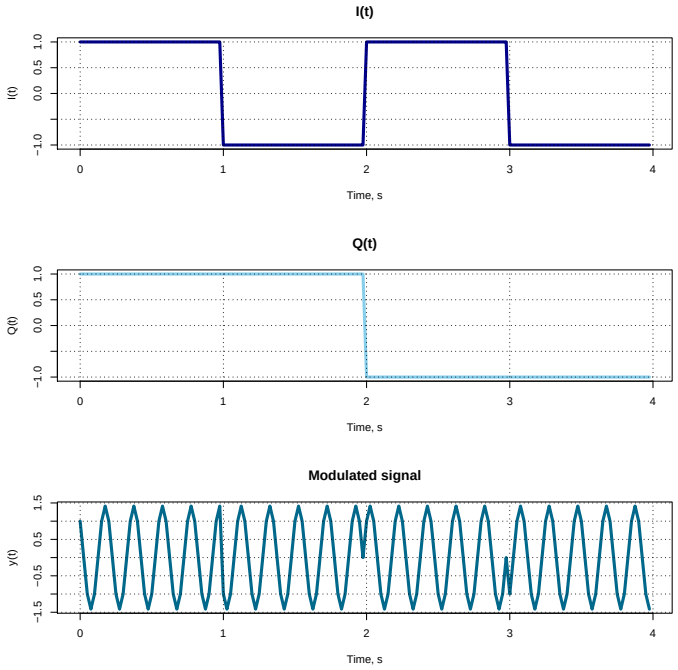


Fig. 4: Modulation of `00 01 11 10` binary message

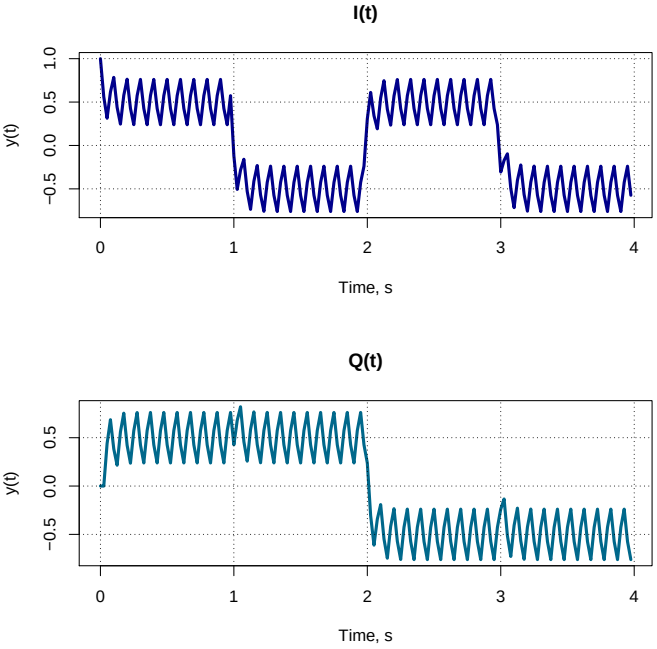


Fig. 5: Decoding binary message

time t: that is, if the value of a sample is greater than 0, $I(t)$ or $Q(t)$ will be 1, otherwise -1 will be assigned to $I(t)$ or $Q(t)$). In Figure 5 we demonstrate the demodulated $I(t)$ and $Q(t)$ values for previously modulated message - `00 01 11 10`.

There are other variants of PSK. To name a few, we can mention Binary Phase Shift Keying (which can encode one bit per symbol) and Quadrature Amplitude Modulation (16-, 64-, 256-QAM). With latter one, 4, 6, and 8 bits per symbol can be transmitted. In QAM the idea is basically the same as in QPSK, whereas the difference lies in the constellation diagram: there are many points per quarter (for example, in 16-QAM there are 4 such points). To decrease the bit error rate (BER) grey code is used to represent the values in constellation diagram. Finally, we should note that DVB-C uses variants of QAM modulation (typically 16-, 64- and 256-QAM is being used).

C. Error correction

A signal, when transmitted over the medium, can be distorted in many ways (for example, due to radio signal attenuation, interference, reflection and other physical phenomena). This will undoubtedly lead to corruption of information. Therefore, it is typically needed to embed some redundant information into the transmitted message, so that the recipient can recover the original message even if some information is corrupted or lost. One such algorithm is Reed-Solomon forward error correction (or FEC). We will review its basics in this section.

Reed Solomon error correction algorithm.

D. MPEG standards

MPEG-2 standard defines how to format the various component parts of a multimedia program. The program may consist of several parts, including compressed MPEG-2 video, compressed audio, control and user data. It also defines how these components are combined into single synchronous transmission bit stream [4]. The process of combining is known as *multiplexing*.

MPEG transport stream. MPEG-TS, which is defined in ISO/IEC 13818-1, is essentially a stream of packets containing encoded multimedia and control data. All packets belong to some elementary stream. MPEG transport stream consists of a sequence of packets of one or more elementary streams multiplexed into one common stream. All packets, as we will discuss later, have an identifier which indicates to which elementary stream the packet belongs.

Each MPEG-TS packet contains a header and a usable payload, such as audio, video information and control data. The structure of a typical MPEG-TS packet is shown in Figure 6. The entire size of a MPEG-TS packet is just 188 bytes. The header size is 4 bytes and carries important information about the payload, leaving only 184 bytes (or less, depending on whether an adaption header is present or not) to the actual payload. The transport stream packet starts with a synchronization byte, whose value should be `0x47` in hexadecimal notation. This byte is recognized by the decoder so that the header and the payload can be deserialized. Next follows the transport error indicator (TEI) bit, which, if set to 1, signalizes that the packet has an error and should be discarded. The next bit is the payload unit start indicator (PUSI). If the bit is set, then it means that the packet is the start of a new packetized elementary stream (PES). Otherwise,

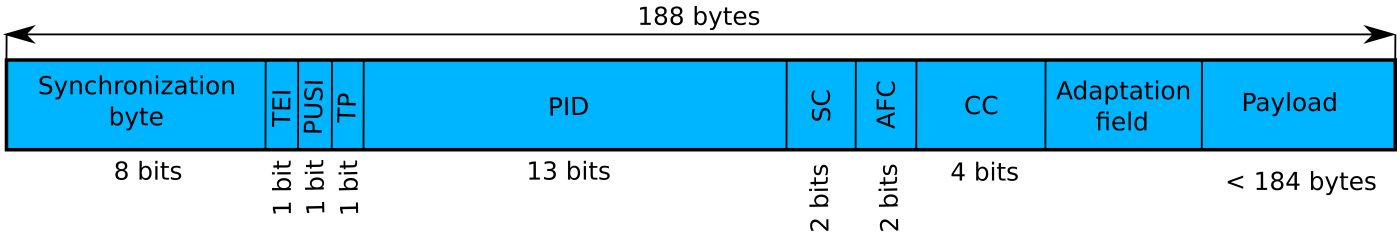


Fig. 6: Transport stream packet structure

TABLE I: Common packet identifiers

PID (hex)	Description
0x00	PAT, Program Association Table
0x10	NIT, Network Information Table
0x11	BAT, Bouquet Association Table
0x11	SDT, Service Descriptor Table
0x12	EIT, Event Information Table
0x13	RST, Running Status Table
0x14	TOT, Time Offset Table
0x20 - 0x1FFA	PMT, Program Map Table, other elementary streams
0x1FFF	Null packet

it is a continuation of an existing PES. As a side note, we should mention that elementary streams can be of two types: (i) packetized elementary stream, which carries video, audio or subtitles and can be up to 65536 bytes long, or (ii) sections, i.e., data structures. The last TS packet for PES packet will contain adaptation header with enough *stuffing* bytes so that end of PES will match the end of TS packet. Next bit, following PUSI bit, is the transport priority (TP) bit, which gives the packet high priority over the packets with the same packet identifier (discussed next). More important (in the course of this work) is the next 13 bits sequence, which encodes unique packet identifier, or PID. A PID can have 8192 unique values, which corresponds to a maximum number of independent elementary streams.

Some predefined values for PIDs are listed in Table I. An elementary stream is a sequence (concatenation) of transport stream packets with the same PID value in the header. Thus, for example, the video elementary stream for a certain program will always have the same PID. This can be used to efficiently demultiplex the program. Next 2 bits carry information about scrambling (encryption) of payload. This two-bit sequence is denoted as scrambling control (SC). Adaption Field Control (AFC) is a two bits sequence. A thorough description of this field and the actual adaptation field can be found in [1]. The last field in the header is the continuation counter (CC). The counter can have 16 values and is used to detect loss, reordering, and duplicates.

The two most important PIDs (not taking into consideration the PIDs for actual audio and video packets) are 0x00 for the program association table (PAT) and the PID for program map table (PMT). The PID for the program map table can be found in the program association table. In turn, the PIDs for audio and video data can be found program map table. As indicated in [9], PAT and PMT are transmitted at a maximum interval of 500ms.

Audio and video codecs. Most common video codec found in MPEG-2 standard is H.264 []. For audio AAC audio codec is being tipically used [] in MPEG-TS streams. In this section we will briefly review these codecs, while a thorough description can be found in [].

E. HTTP Live Streaming (HLS)

The details of HLS protocol are described in RFC 8216 [7]. Here we will briefly outline its key concepts.

F. Security

Digital restrictions management (DRM) is essential in digital era. Briefly, it can be described as a set of practices and techniques used to protect the content from unauthorized viewing and distribution. There are perhaps many solutions to protect the content from being eavesdropped, but in this work, we will focus only on encryption of streams with block ciphers. HLS and DVB use different approaches (in terms of algorithms) to protect the video streams from unauthorized usage. Thus, in HLS the entire MPEG-TS stream is encrypted with the widely used Advanced Encryption Standard (AES) algorithm [10], using a key of size 128 bits. DVB, in turn, uses its own, or shall we say domain specific, block cipher called the Common Scrambling Algorithm (CSA) [11], with a relatively small key of just 48 bits in length (in reality, the key is equal to block size, but the 2 bytes in the key are computed in a deterministic manner, leaving only 48 bits of entropy).

DVB Common Scrambling Algorithm (DVB-CSA) is a proprietary algorithm, which was designed for DVB standards. In essence, CSA is a block cipher, which operates in two phases. In the first phase, the payload is split into equal size blocks, each of size 64 bits. All blocks are then encrypted in reverse order: the first block in order (the last block in the original sequence) is encrypted using all zero initialization vector, all other blocks prior to encryption are also xored with the previously encrypted block (here block cipher is also initialized with all zero initialization vector). The first block in the original sequence is also tweaked with an encryption key. In the second phase, each block, starting from the second block (in original order), is *xored* with the key: the key for the second pass is produced with a stream cipher, which is initialized with the value of the first block and the encryption key. In Figure 7 we schematically demonstrate the operation of CSA algorithm.

HLS security. As described in [7], the media segments can be delivered as plaintext (with no encryption), encrypted with AES-128 in Cipher Block Chaining (CBC) mode [10], or as ciphertext produced with SAMPLE-AES encryption method.

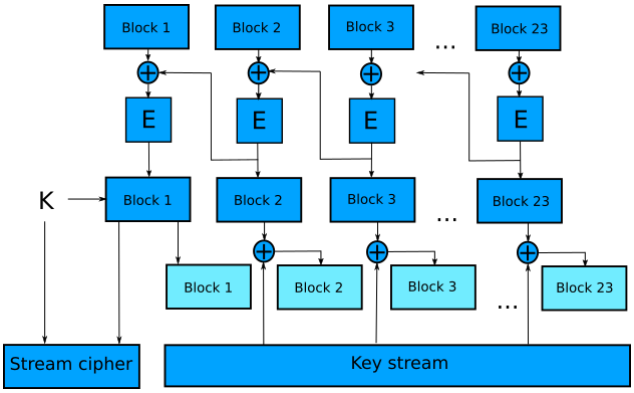


Fig. 7: DVB-CSA structure

As the name implies, in no encryption mode nothing is being encrypted and segments are delivered to clients as a plaintext. Of course, HTTP supports secure mode of operation, which is known as HTTPS. Secure version is built with help of SSL [6] or TLS [5] protocol. This means that eavesdropper will not be able to get access to contents of segments, even if it will sniff packets on wire or air. However, additional mechanisms for authentication and authorization are still required to protect segments from unauthorized access.

If HLS implements encryption with AES method, then entire segments are encrypted using AES encryption in CBC mode using 128 bit key. When this approach is implemented, EXT-X-KEY attribute in the playlist file must specify appropriate encryption method (see for reference [7]), URL of the key file, block cipher initialization vector (IV), and some other parameters. It is then responsibility of the client to decrypt the segments using appropriate parameters. We should note that key can be protected from unauthorized access with cookie files [3]. This means that the client first needs to use methods advertised by the content provider to authenticate itself, and only then try to access encrypted segments and key file. It is rather arguable whether this mechanism is more advantageous over mechanism with no encryption when used together with HTTPS protocol. On one hand, one can build a similar access mechanism for plaintext segments, i.e., the delivery of segments can be organized over HTTPS protocol, while cookies can be used for authentication. On the other hand, when AES mode of encryption is being used together with HLS protocol, the need for HTTPS to deliver the media segments vanishes (segments can be securely delivered over insecure HTTP protocol, and only key file needs to be transmitted over encrypted tunnel). This means that extra CPU cycles, which are needed to perform key exchange, key derivation and encryption of the payload, can be avoided. Perhaps, real-life measurements can shed a light on this matter.

Finally, there is SAMPLE-AES method. In this method, only media samples such as audio or video, contained in segments, are encrypted using AES algorithm. How this media segments are encrypted and encapsulated depends largely on media encoding [7].

III. Hardware and software

To build the prototype, we have used two components. Basically, the computing board and DVB stream demodulation board. For the first one we have used cheap Raspberry PI board, which has ARM CPU with four cores, one gigabyte of random access memory, several universal serial buses and is capable of running Linux operating system. For the DVB signal demodulation, we have used Geniatech T230C2 card [2]. The card is capable of demodulating the DVB-C and DVB-T signals. Both pieces of hardware are inexpensive.

Fig. 8: Prototype setup

We should note, that to build system, which is capable of delivering multiple TV channels (which are not multiplexed in the same stream), an array of devices is needed: One computing board and one demodulation device is needed to deliver one MPEG stream. Of course, several channels can be parsed from the stream using single device, since many channels use the same carrier frequency and we were able to demultiplex such channels (more precisely we were able to demultiplex four different channels (although there are many more channels exist in the same stream)).

In our demo setup (see for clarity Figure 8), Raspberry PI not only was capturing the live stream from DVB-C card, but also was serving the MPEG2 streams and frontend requests using `nginx` server and a custom server written in `python` language. The python server (together with nginx) was responsible for such tasks as: (i) user authentication and authorization; (ii) serving static web pages; (iii) serving the m3u8 playlists and MPEG-TS segments. Finally, to implement the frontend (web UI) we have used Angular.js library. For the demonstration in Figure 9 we show how the web user interface looks like.

IV. Experimental evaluation

In this section, we present some rudimentary experimental evaluation of the IPTV streaming service. Basically, we will present the results for the following key performance indicators: (i) time required to execute separate blocks of code, such as stream encryption, stream processing, etc; (ii) overall CPU

Fig. 9: Web UI of the prototype

load as a function of time (on both client and server); (iii) memory usage as a function of time (on the server only); and finally, (iv) utilized network bandwidth.

The first experiment we have conducted was microbanchmarking of AES encryption on Raspberry PI. For each file of size 2^i megabytes, where $i \in [0, 7]$, we have performed 100 rounds of encryption and derived mean encryption time. We report the running times in seconds in Figure 10. Basically, the running time is negligible for small streams and grows linearly with the size of the plaintext. In our prototype, the size of the segment was set to just 4MB. The mean encryption time for such segment size is about 600ms, which constitutes about $5-6\%$ from segment duration, and thus should not cause delays.

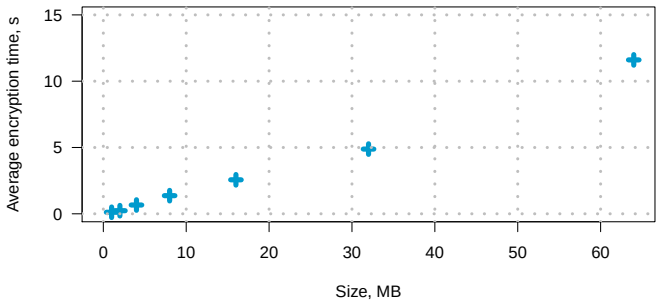


Fig. 10: Microbanchmarking of AES encryption operation

Our next experiment was related to re-encoding the audio signal. It turned out that *Flash* player - a common add-on to browsers, which is used for playing videos - does not support audio signal, encoded with MPEG2 audio codec. Since our content provider was broadcasting audio encoded with this codec, we required to transcode each segment using AAC audio codec. It was in our interest, thus, to benchmark the performance of this operation. In Figure 11 we plot the mean time for this operation as a function of time, during which the signal was recorded. Again the mean time to re-encode the segment of size 4MB was about 1500ms, or $13-15\%$ from the segment duration. Together with encryption, both operations require just 2.1 seconds, and should not cause any delays in video playback on the client - our experience with working prototype confirmed this.

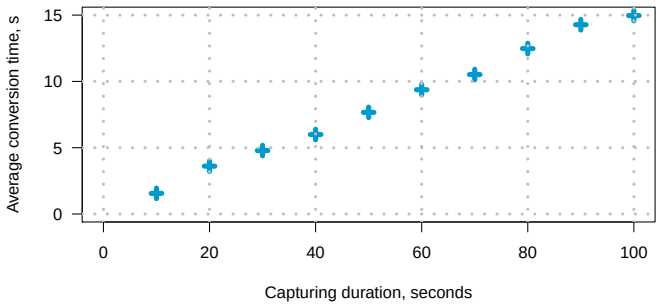


Fig. 11: Microbanchmarking of audio transcoding operation

In Figure 12 we demonstrate the running prototype (we tested stream decoding in both browser (using Flash player) and *ffplay* - desktop application). The server was configured to encrypt the segments with the AES-128 algorithm in CBC mode and was also transcoding the audio stream from MPEG-2 audio into AAC audio format. In order to understand the burden of capturing, re-encoding and encryption processes, we set down and measured the CPU utilization. For that matter, in Figure 13 we show histogram and timeseries plot for the typical CPU utilization. From the data, we derived the mean CPU utilization, which was somewhere around 11% (whereas maximum value was reaching 55%). Basically, this means that the board is capable of processing several MPEG-TS streams without a problem. In this pace, we have also measured the CPU utilization, when audio and video streams for four channels were captured using a single board. In Figure 15 we show the histogram for this data. From the data, we derived the mean CPU utilization, which was slightly more than 48%. To exclude IO bottleneck we have mounted RAM file system, and the capturing process was saving MPEG-TS streams to this file system. We have noticed that the best quality and viewing experience was achieved for the setup with one channel per computing and capturing device.

To support more channels, a production setup should have at least one processing and signal demodulation devices per two channels. To serve the video a load balancer (an HTTP router) should be placed in front servers. The main role of the load balancer is to route HTTP requests to proper server based on the client's IP address.

We have also measured the CPU utilization on the client. In Figure 14 we show histogram and timeseries plot for this data. Here, the mean CPU utilization was roughly 44%, whereas maximum value was reaching 93%.

Our next step was to measure the average network utilization. It turned out, that roughly 3.2 Mb/s channel was needed to serve a single client. Back of the envelope calculation suggests that a 1 gigabit link can provide enough capacity for 300 concurrent clients. To improve the performance (increase the number of concurrent users) a multicast protocol (or IGMP) should be used. In this case, of course, to our best knowledge

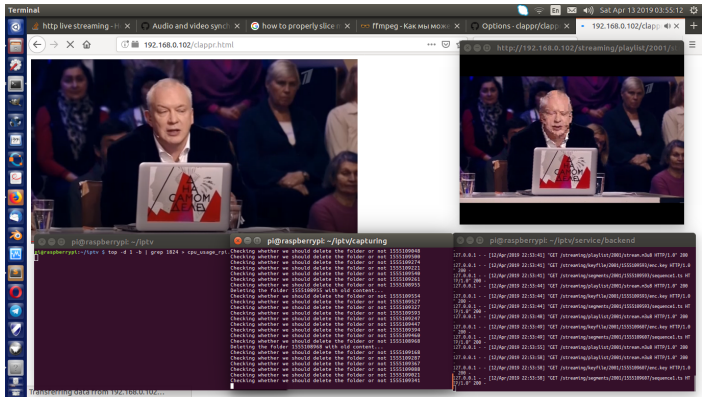
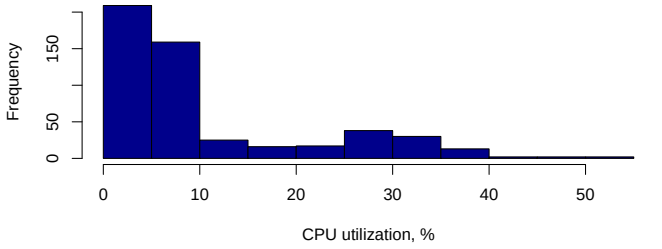


Fig. 12: Running prototype



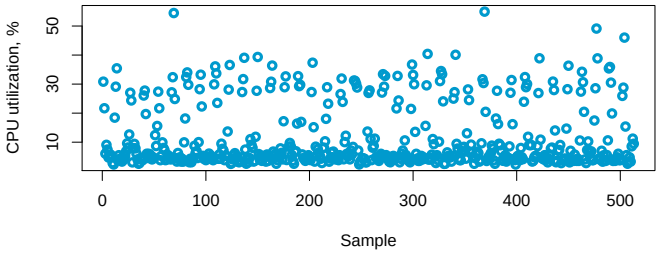
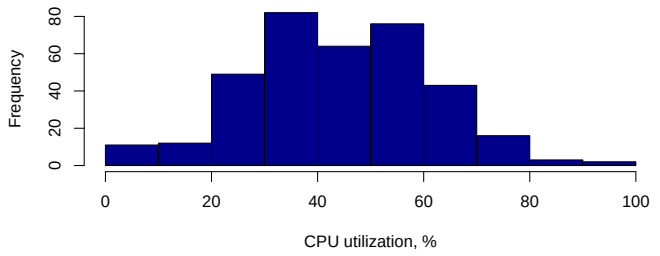


Fig. 13: CPU utilization on the server (single channel)



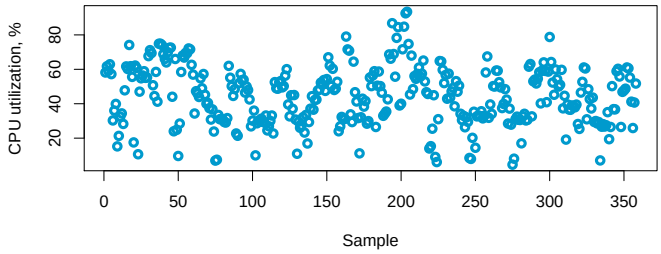


Fig. 14: CPU utilization on client (only for Flash player)

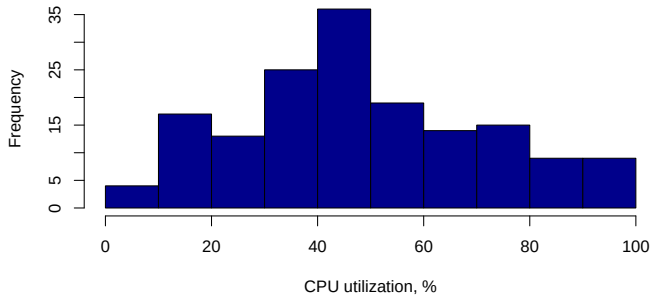


Fig. 15: CPU utilization on server (demultiplexing four channels)

HLS protocol is not usable, and Real-time Transport Protocol (RTP) should be considered instead. This configuration is outside of the scope of this paper, but on a high level, in this setup the server simply sends RTP packets (which contain MPEG-TS data) to a multicast IP address and the clients, which are subscribed to the group will receive and decode the MPEG-TS payloads. A further discussion on how to encapsulate MPEG-TS stream into RTP packets can be found in [8].

V. Conclusions

References

[1] A Guide to MPEG Fundamentals and Protocol Analysis. http://www.img.lx.it.pt/ fp/cav/Additional_material/MPEG2_overview.pdf.

[2] Geniatech T230. Online https://www.geniatech.com/product/t230/.

[3] HTTP cookie. https://en.wikipedia.org/wiki/HTTP_cookie.

[4] MPEG-2 Transmission. https://erg.abdn.ac.uk/future-net/digital-video/mpeg2-trans.html.

[5] RFC 5246. https://tools.ietf.org/html/rfc5246.

[6] RFC 6101. https://tools.ietf.org/html/rfc6101.

[7] RFC 8216. https://tools.ietf.org/html/rfc8216.

[8] Rfc2250: Rtp payload format for mpeg1/mpeg2 video. https://tools.ietf.org/html/rfc2250.

[9] W. Fischer. Digital television : a practical guide for engineers / walter fischer. *SERBIULA (sistema Librum 2.0)*, 04 2019.

[10] D. Stinson. *Cryptography: Theory and Practice, Second Edition*. CRC/C&H, 2nd edition, 2002.

[11] E. Tews, J. Wälde, and M. Weiner. Breaking DVB-CSA. In *Research in Cryptology*, pages 45–61, 2011.