D I P L O M A R B E I T

Video streaming test bed for UMTS network

unter der Leitung von

Prof. Dr. Markus Rupp
DI Michal Ries
Institut fur Nachrichtentechnik und Hochfrequenztechnik

eingereicht an der Technischen Universitat Wien
Fakultat fur Nachrichtentechnik und Hochfrequenztechnik

von

Marek Braun

Matrikelnr.: 0527236
Gusshausstrasse 25/389
1040 Wien

Wien, Jun 2006

. . . . . .

Contents

Chapter 1

Introduction

Steaming services in 3G networks and beyond are bringing new challenges for network settings and quality management. Solution of this problem is crucial for dynamic mobile and wireless environment. For detailed investigation of these aspects it is necessary to build test bed for simulation of the streaming services in Universal mobile telecommunications systems UMTS network. The aim of this work is to develop a video streaming test bed witch will simulate all aspects of real UMTS video streaming (packet loss, desynchronization, and jitter) and relate this aspects to video quality.

Chapter 2

Video streaming over UMTS

2.1 Technical conditions for video streaming in UMTS network

Streaming refers to the ability of an application to play synchronized media streams like audio and video streams in a continuous way while those streams are being transmitted to the client over a data network. Streaming over fixed internet protocol (IP) networks is already a major application today. For the third generation (3G) systems, the 3G packet-switched streaming service (PSS) [1] fills the gap between 3G multimedia messaging service (MMS), e.g. downloading, and conversational services. Video streaming in UMTS network is classified by [1] as simple streaming. The simple streaming service includes a basic set of streaming control protocols, transport protocols, media codecs and scene description protocol. In this simple case, there is neither explicit capability exchange, nor any encryption or digital rights management. A mobile user obtains a universal resource identifier (URI) to a specific content that suits his or her terminal. This URI may come from a World Wide Web (WWW) browser, a wireless application protocol (WAP) browser, or is typed in by hand. This URI specifies a streaming server and the address of the content on that server. An application that establishes the multimedia session shall understand a Session Description Protocol (SDP) file. The SDP file may be obtained in a number of ways. It may be provided in a link inside the hypertext mark-up language (HTML) page that the user downloads, via an embedded tag. It may also be obtained directly by typing it as a URI. It may also be obtained through a real-time streaming protocol (RTSP), signaling via the DESCRIBE method. In case of streaming delivery option of MMS service, the SDP file is obtained via the MMS user agent that receives a modified MMS message from the MMS relay or server.

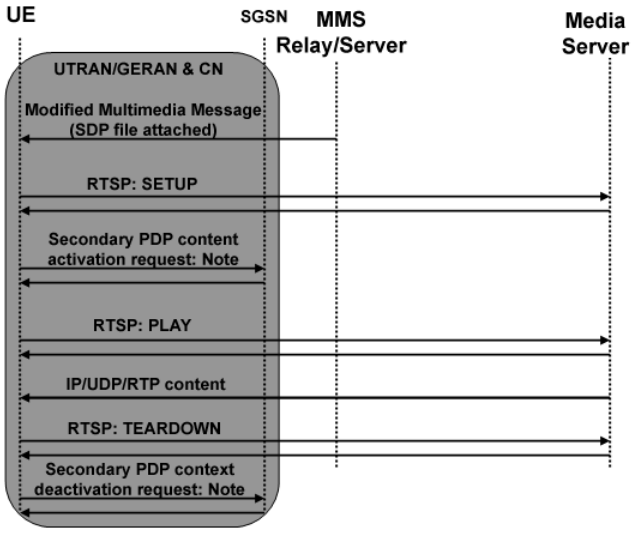

Figure 2.1: Schematic view for streaming session originated via MMS.

The SDP file contains the description of the session (session name, author ...), the type of media to be presented, and the bit rate of the media. The session establishment is the process in which the browser or the mobile user invokes a streaming client to set up the session against the server. The user equipment is expected to have an active packet data protocol (PDP) context in accordance with [2] or other type of radio bearer that enables IP packet transmission at the start of session establishment. The client may be able to ask for more information about the content. The client shall initiate the provisioning of a bearer with appropriate quality of service (QoS) for the streaming media. Sessions containing only non-streamable content such as a synchronized multimedia integration language (SMIL) file, still images and text to form a time-synchronized presentation do not require the use of an SDP file in session establishment. The set up of the streaming service is done by sending an RTSP SETUP message for each media stream chosen by the client. This returns the user datagram protocol (UDP) and/or transport control protocol (TCP) port etc. to be used for the respective media stream. The client sends a RTSP PLAY message to the server that starts to send one or more streams over the IP network. This case is illustrated below in Figure 2.1.

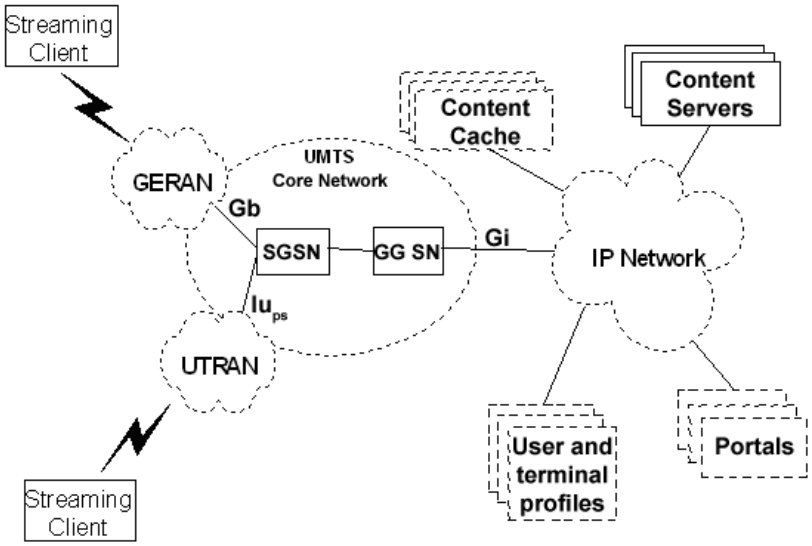

Figure 2.2: Network elements involved in a 3G packet switched streaming service.

2.2 Streaming architecture

Figure 2.2 shows the most important service specific entities involved in a 3G packet - switched streaming service. A streaming service requires at least a content server and a streaming client. A streaming server is located behind the Gi interface. Additional components like portals, profile servers, caching servers and proxies located behind the Gi interface might be involved as well to provide additional services or to improve the overall service quality. Portals are servers allowing for convenient access to streamed media content. For instance, a portal might offer content browse and search facilities. In the simplest case, it is simply a Web/WAP-page with a list of links to streaming content. The content itself is usually stored on content servers, which can be located elsewhere in the network. User and terminal profile servers are used to store user preferences and terminal capabilities. This information can be used to control the presentation of streamed media content to a mobile user.

2.3 Session initialization and control (H.323, SIP)

Session initiation is required when streaming on the Internet because a server must be able to identify the clients that make a request and it must also determine what content is requested. Additionally, the client may have certain preferences as to how the streaming should occur e.g. what format to stream, what port numbers to use for the packets etc. H.323, Session Initiation Protocol (SIP) and RTSP are three standards that support such functionality.

H.323, by the International Telecommunications Union - Telecommunication Standardization Sector (ITU-T), is a standard [2] that covers signaling and control of internet calls. Rather than being a specific protocol, it is more of an architectural overview of Internet telephony with support for multimedia content. Many protocols are recommended in H.323 for the various processes such as speech coding, call setup and data transport etc.

SIP was developed by the Internet Engineering Task Force (IETF) and is now a IETF Request For Comments (RFC) document [3], which specifies a protocol that describes how to set up Internet telephone calls, videoconferences and other multimedia connections. Unlike H.323, SIP does not specify how to conduct any other part of the streaming process e.g. speech coding etc. While there are some similarities between SIP and H.323 in that they both provide functionality to enable session set up, H.323 is a more comprehensive standard that specifies how to transport and manage content in addition to session configuration. In [4] a comparison of these two standards is presented, and [5] explains how they can be used together.

RTSP, defined in [6], is similar to SIP in that it is used for setting up a session between a client and server to allow multimedia communications but it also supports interactive control such as recording and absolute positioning of the media so a viewer can fast-forward or rewind to a desired scene. Although

there is a large overlap between RTSP and SIP, they can be used to compliment each other for particular applications such as the messaging application in [7].

All three of these protocols may be used to setup streamed video sessions, but RTSP was used for the work presented here because it is well documented and supported by many popular applications, and it provides extra functionality that may be useful for future advancements to the work. In addition, many MPEG-4 applications have already been proposed or implemented using this protocol such as [8] [9].

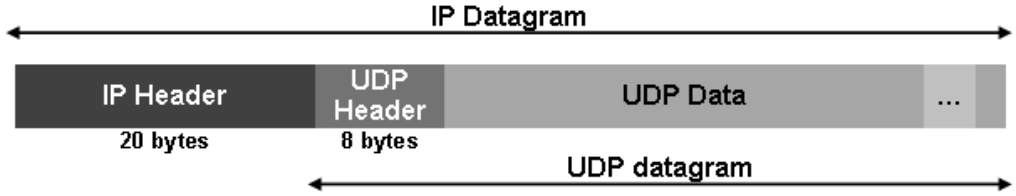

Figure 2.3: Encapsulation of a UDP datagram within an IP packet.

2.4 Video streaming over IP networks

The UDP or TCP are typically used for transport of data over IP networks and a comprehensive explanation of these protocols can be found in [11]. In the following sections we examine these two protocols and identify issues associated with each for the transport of streamed video. A protocol called Real-time Transport Protocol (RTP) which attempts to alleviate some of these problems is discussed below. To Real time Transport Control Protocol (RTCP) which provide feedback mechanism is addicted following chapter. And RTSP which is used for establishment and control of either a single or several time-synchronized streams of continuous media (such as audio or video) is discussed below too.

2.4.1 UDP

UDP is a simple transport layer protocol that is encapsulated in IP datagrams. Typically each UDP datagram is mapped to a single IP packet as shown in Figure 2.3, although UDP datagrams that are larger than the Maximum Transmission Unit (MTU) size supported by the underlying network, may be fragmented into a number of IP datagrams. The MTU size varies for different networks, but for a Local Area Network (LAN) it is 1500 bytes, so the maximum amount of data that can be carried in a UDP payload is 1472 bytes and the rest is occupied by the IP and UDP headers.

As UDP is a connectionless protocol, it does not have the overheads associated with connection set up. In addition, each UDP packet or datagram consists of a header that is only 8 bytes in length followed by the payload data as illustrated in Figure 2.4. This results in an overhead of only 28 bytes per packet for both the IP and UDP headers. There are four two-byte header fields that make up the UDP header; the source port number, the destination port number, the UDP length, and the UDP checksum.

Port numbers are used in this protocol so that traffic can be directed to

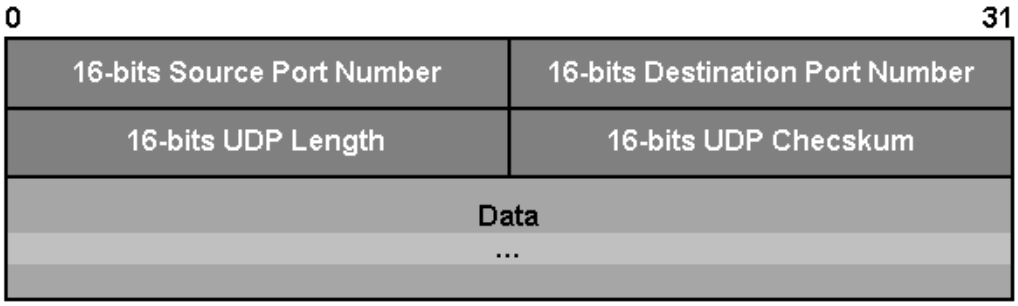

Figure 2.4: A UDP packet and with its 4 header fields.

specific processes on the client. Each process that expects to receive UDP datagrams must bind to a particular port number. Processes wishing to send process in the destination port number field. The source port number field allows the receiving process to determine what port it may use to send packets in response.

The UDP length field indicates the length of the entire UDP datagram i.e. the header and the data. Detection of errors in the header or payload can be achieved by the use of the optional, but recommended, checksum field in the UDP header. If the checksum is not used, this value is set to zero.

As discussed above, UDP is a very simple protocol with a relatively low transmission overhead, but there are a number issues associated with UDP that are highly significant for the streaming of time-constrained media such as audio and video. Because IP is the basis for UDP, it is a 'best-effort' and unreliable protocol that cannot guarantee timely or even successful delivery of packets to a client. Video decoded from streams that are delivered in an untimely or lossy manner commonly exhibit artefacts that have negative impacts on the perceptual quality.

Loss of even a single packet for streamed video can result in the loss or damage of a whole video frame and potentially the propagation of an error across a number of frames due to the way video is encoded and packetized. In addition to impairments caused by the loss of packets, the variation of interpacket delivery delays, referred to as jitter, can also result in serious artefacts in received video. Dynamic transmission delays can result in the delivery of packets at the client in a different order than the server sent them, but UDP headers do not provide any information that could be used to reorder the packets at the client. Decoders do not commonly support checks or recovery techniques for inconsistent encoded video input and thus loss and jitter may lead to incorrectly decoded output and possibly the failure

Figure 2.5: Sample impacts of video streaming errors. The images show a video streaming over sequence unimpaired (left), with loss (middle) and with packet reordering.

to decode particular frames, Figure 2.5 illustrates some artefacts caused by loss and reordering.

2.4.2 TCP

TCP was designed to be reliable i.e. to counteract packet loss and unordering of packets, but it still exhibits transmission characteristics that make it inefficient for real time or interactive video streaming.

TCP is another protocol that runs over IP as shown in Figure 2.6, but unlike UDP, it is reliable because it uses retransmissions to guarantee delivery and supports synchronization of out of order data received by the client. Other significant differences between the two are the fact that TCP is connection oriented and that the TCP standard includes specifications for Flow Control. This means that the client and server negotiate to establish a connection prior to the sending of data. Each TCP segment contains a header of at least 20 bytes but this can be more depending on whether or not TCP options are included in the TCP header.

In the same way as UDP, TCP uses source and destination port number to identify sending and receiving processes. Reordering of unsynchronized data is achievable with TCP due the inclusion of a sequence number. Detection of lost or heavily delayed packets is also possible by using the acknowledgment number. The acknowledgment number is periodically sent to server to reveal which packet the client is expecting and using this information the server may decide to retransmit any packets that may have been lost in transmission. These two fields are essential in making TCP a reliable protocol.

The four-bit header length indicates the length of the headers because there may be options appended to the header fields, but typically options

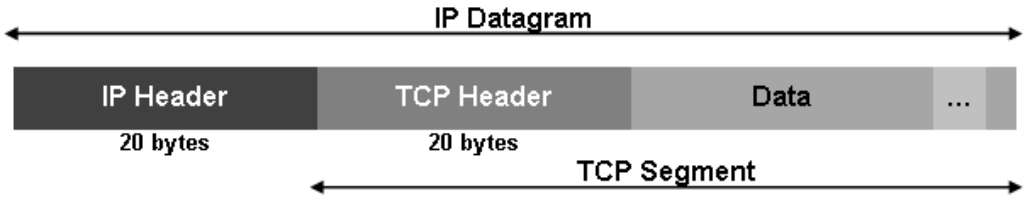

Figure 2.6: Encapsulation of TCP data in an IP datagram. It should be noted that the TCP segment in this diagram includes no TCP options fields, hence the TCP header is 20 bytes.

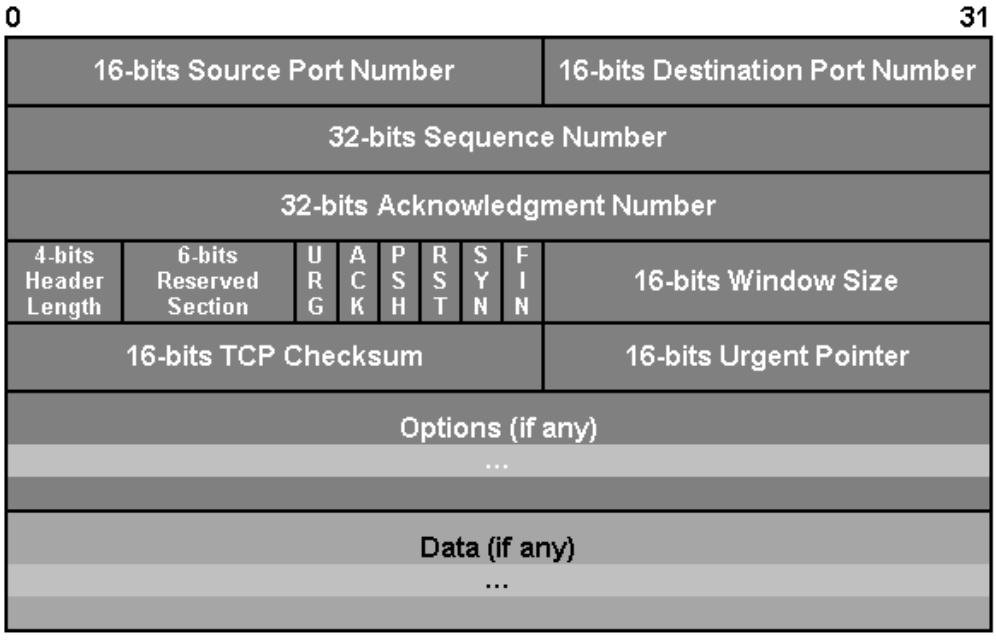

Figure 2.7: A TCP Segment with its header fields illustrated.

are not used and its value is 20. Next to the options there are six flag bits in the header. URG is used to determine if the value in the urgent pointer field is valid. If set, the urgent pointer contains a sequence number offset, which corresponds to a TCP segment that contains urgent data and it should be expedited to its destination. ACK indicates if the acknowledgment number field is valid and if it is, it can be used by the receiver to communicate with the server regarding the packets that it has received in order and intact.

PSH is used to minimize the amount of buffering used before passing the data in this packet to the receiving process. TCP uses a buffer at the receiver because it uses byte streams rather than message streams. The RST flag used to reset the connection, usually after a segment that does not appear correct for the referenced connection is received. The SYN and FIN flags are used for establishing and closing TCP connections.

Flow control is achieved by TCP using the window size field. This value is the number of bytes, starting with the byte acknowledged, that the receiver is willing to accept. If a receiver is busy or does not want to receive more data from the sender, this value can be set to 0. As with UDP, the TCP checksum allows for error detection in the TCP segment.

In addition to the flow control based on the window size, TCP uses other congestion control mechanisms such as Slow Start and Additive Increase Multiplicative Decrease (AIMD). The slow start mechanism employed by TCP means that TCP data tries to avoid congestion by starting the transmission at a low rate and increasing the rate gradually to an acceptable level. AIMD means that the rate of transmitted data is increased slowly while the network appears capable of coping with the current rate, but as soon as the rate appears excessive i.e. packets are being lost or delayed, the sender drops the rate of data sent dramatically.

TCP is very useful for a number of possible applications like HyperText Transfer Protocol (HTTP) for the WWW and File Transfer Protocol (FTP) because these applications are not time-critical and the integrity of the data received is guaranteed. This is not the case for streaming of time sensitive media like audio and video. The slow start and AIMD approach to TCP may not be suitable for streamed video because it needs to be delivered in a timely manner throughout the stream. Also the use of a buffer at the receiver means the process must wait for the buffer to fill, which may require many retransmissions before being passed the data. As a result of the buffering and flow control mechanisms used in TCP, the delivery of data to the receiving process is not very consistent and this causes jerky video output.

In addition to transmission timing issues, TCP also introduces much greater overheads than UDP because it must establish and manage a connection, issue retransmissions when required and its header is larger. These

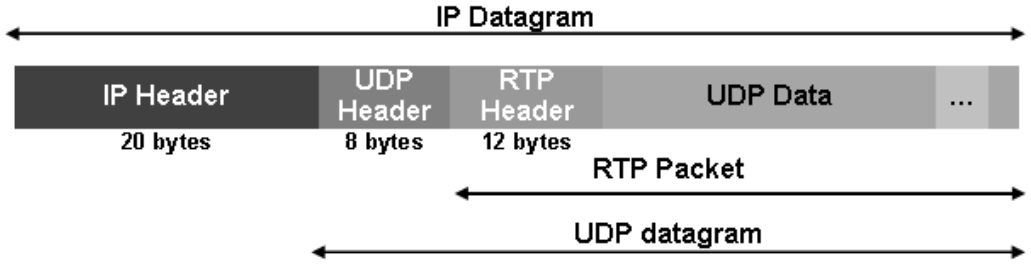

Figure 2.8: Encapsulation of an RTP packet within a UDP and IP datagram. It should be noted the RTP header illustrated does not contain any CSRC or extension headers.

overheads may not be significant in particular scenarios such as HTTP or FTP, but for streamed video the acknowledgments, retransmissions and large header size can require excessive amounts of bandwidth. This is particularly true in mobile contexts where bandwidth is at a premium.

So far, the advantages and disadvantages of both TCP and UDP for streaming video have been discussed, but both exhibit serious issues. RTP adopts the positive features of each and omit the features that lead to streaming problems.

2.4.3 RTP

RTP, as the name suggests, was designed for real-time delivery of data such as audio and video. Unlike UDP and TCP it is not a transport layer protocol, although it does exhibit features similar to a transport layer protocol. RTP usually runs over UDP and similar to an application layer protocol, it is typically placed in the user space in the operating system. It is possible to use RTP over TCP but due the issues with TCP mentioned above, it is not very common.

RTP is particularly appropriate for media streaming because it allows for synchronized delivery of data and supports detection of lost or unordered packets at the client. Because it is UDP based, RTP is connectionless and so the only transmission overhead involved is the IP, UDP and RTP headers. The RTP header is 12 bytes in length, and the combined IP, UDP and RTP header without any optional extensions is 40 bytes, equivalent to the IP and TCP headers in a single TCP packet. Header fields used by RTP are shown in Figure 2.9.

V indicates the version of RTP used and it is set to 2 in current standards. The P and X flags are used to indicate if padding or extension data are

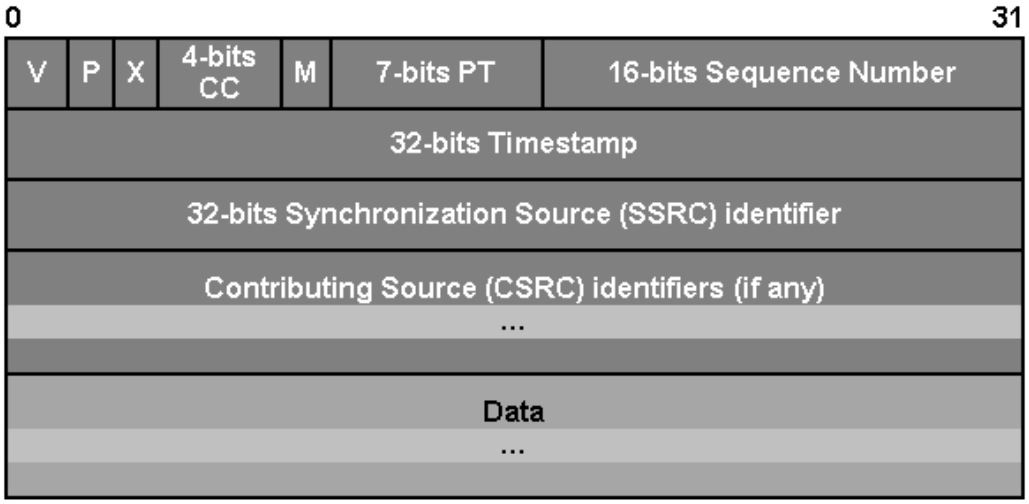

Figure 2.9: An RTP packet and its header fields.

included in the packet. CC is the Contributing source (CSRC) Count and indicates how many CSRC fields are appended to the RTP header. CSRCs are used to identify the media source that contributed to the packets payload e.g. in a video-conference many conference participants may contribute the audio content.

The Marker bit, M, can be used in a number of ways defined by a profile, but for video it is normally set to indicate that the subsequent packet contains the start of a coded new frame. One of the fundamental ideas behind RTP is that each packet contains data corresponding to an exact media time that can be determined from the timestamp field. However, RTP's payload size is limited by the MTU of the underlying network so if a coded frame is larger than the permissible payload size it can be fragmented into two or more RTP packets. The marker is used to identify the boundaries of frames, allowing the client to pass only full media units to a decoder. The timestamp is also important because it is used determine the sending time for the packet and also to allow the client to control the presentation of the media content, particularly for media with dynamic sampling rates.

Encoding techniques for the different media types in the RTP payload can vary greatly and it is even possible for the encoding to change dynamically during an RTP stream, so the Payload Type, PT, field describes the encoding of the payload content. There are a number of encoding types that have been registered with the Internet Assigned Numbers Authority (IANA) and have been given standard PT values. Encodings that have not been registered can

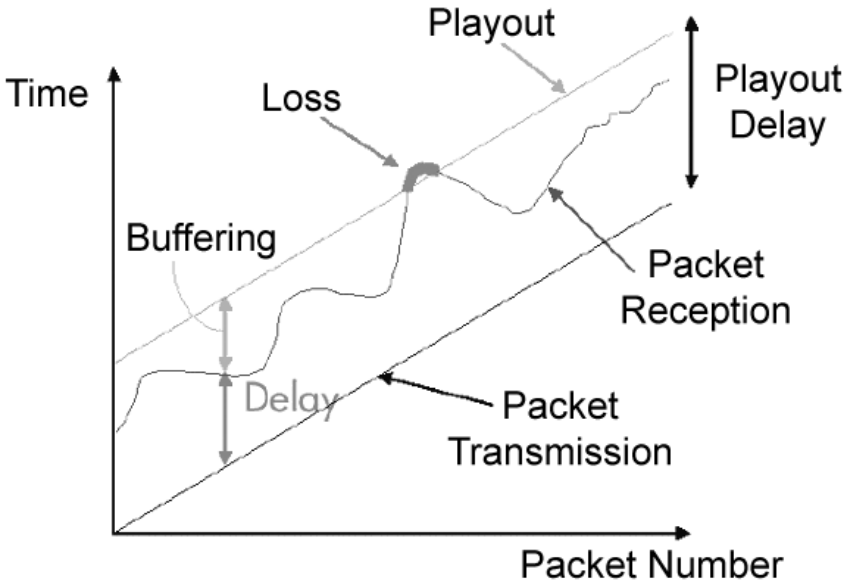

Figure 2.10: A graph showing how a media client uses buffering to handle jitter. (Extracted from [12]). In the graph, it can be seen how buffering is achieved by using a playout delay and therefore a significant amount of packet jitter can be accommodated. Loss only occurs when packets arrive after their playback time.

communicate the configuration corresponding to an unregistered PT value using out of band means such as the SDP in RTSP.

Like TCP, RTP includes Sequence numbers in the header that are used to allow a client to detect loss and resynchronize out of order packets. To permit this reordering and to accommodate for some level of network jitter, RTP clients often use a buffer, but for real-time and interactive media streams, this buffer is small. Using a small buffer means that it is quite likely that jitter will result in the delivery of packets that have missed their intended playout time. To avoid redundant decoding and possible inconsistencies in the decoded content, any packet that arrives later than its playout time are not buffered but are omitted and have the same impact as though the data was lost in transmission, as illustrated in Figure 2.10.

Finally, the Synchronization Source (SSRC) field is a random value used to differentiate between different RTP streams. This is necessary because it is possible to have more than one stream of data of the same payload type within an RTP session. An RTP session refers to all the RTP media

streams that correspond to a single multimedia presentation e.g. a simple RTP session may have one RTP stream for the video and another for the audio. RTSP commonly controls a number of RTP streams within a single RTSP session and each RTSP request affects all streams in that session.

The RTP standard [13] also includes a specification for a feedback protocol called RTCP that allows the client to communicate packet reception statistics to the server. The server may then apply some form of adaptation on the stream or the video quality to match the conditions reported by the client, an example of this discussed in [14], which presents an architecture for adapting the video based on the network conditions reported by the client.

As discussed above, RTP is more appropriate for real-time media streaming than UDP or TCP because it supports a low transmission overhead that sends packets at time corresponding to their media time. It also supports the identification and resynchronization of media units to ensure that valid content is passed to the decoder. However there are still a number of issues with RTP for streaming video content, especially in bandwidth constrained environments such as mobile networks.

Using RTP for a multimedia sequence that contains many distinct media types such as audio, video, text etc requires that each media stream is placed in a unique RTP stream even if the sampling and timing of all the media streams are coordinated. Also, video and audio compression techniques are well advanced and a single coded media unit may require only a very small fraction of the payload space available with RTP, particularly when video of small dimensions such as that used for mobile devices is transmitted. Both of these considerations can result in inefficient use of bandwidth because a significant proportion of the bandwidth is expended in IP, UDP and RTP headers.

2.4.4 RTCP

The RTCP was developed to provide feedback mechanism to enable adaptation over IP networks. RTCP is based on the periodic transmission of control packets to all participants in the session, using the same distribution mechanism as the data packets. The underlying protocol must provide multiplexing of the data and control packets, for example using separate port numbers with UDP. RTCP performs four functions:

1. The primary function is to provide feedback on the quality of the data distribution. This is an integral part of the RTP's role as a transport protocol and is related to the flow and congestion control functions of other transport protocols.

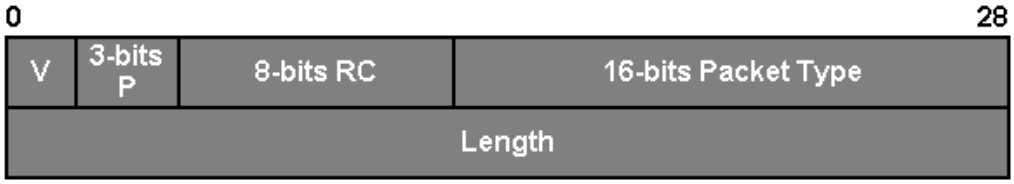

Figure 2.11: A RTCP packet and with its 5 header fields.

2. RTCP carries a persistent transport-level identifier for an RTP source called the canonical name or CNAME. Since the SSRC identifier may change if a conflict is discovered or a program is restarted, receivers require the CNAME to keep track of each participant. Receivers also require the CNAME to associate multiple data streams from a given participant in a set of related RTP sessions, for example to synchronize audio and video.

3. The first two functions require that all participants send RTCP packets, therefore the rate must be controlled in order for RTP to scale up to a large number of participants. By having each participant send its control packets to all the others, each can independently observe the number of participants. This number is used to calculate the rate at which the packets are sent.

4. A fourth, optional function is to convey minimal session control information, for example participant identification to be displayed in the user interface. This is most likely to be useful in "loosely controlled" sessions where participants enter and leave without membership control or parameter negotiation.

Functions 1-3 are mandatory when RTP is used in the IP multicast environment, and are recommended for all environments. RTP application designers are advised to avoid mechanisms that can only work in unicast mode and will not scale to larger numbers.

Functions 1-3 should be used in all environments, but particularly in the IP multicast environment. RTP application designers should avoid mechanisms that can only work in unicast mode and will not scale to larger numbers. Transmission of RTCP may be controlled separately for senders and receivers for cases such as unidirectional links where feedback from receivers is not possible.

Fields of header used by RTCP are shown below in Figure 2.11.

2.4.5 RTSP

RTSP is used for establishment and control of either a single or several time-synchronized streams of continuous media such as audio or video. It is an application-level protocol and provides an extensible framework to enable controlled, on-demand delivery of real-time media and is compatible with both live and pre-recorded media content. The control mechanisms of RTSP are comparable to a remote control for a VideoCassette Recorder (VCR) because it allows the client to signal messages to the server to invoke playback, record, fast-forward, rewind, pause and other media specific operations.

RTSP defines the syntax and types of requests that should be supported by any RTSP compatible client and server. Requests can be in either direction i.e. from client to server or vice versa, depending on the type of request to be issued. The request format is based on HTTP/1.1, which unlike previous HTTP versions does not require a persistent connection to the server Because of this the server cannot use the connection to identify different sessions, so it allocates session numbers to each client that establishes an RTSP session with the server and then all subsequent requests by the client must include the allocated session number. Similar to HTTP/1.1 all requests must be acknowledged by appropriate response to indicate the success or failure of a request.

Many different sets of requests may be used depending on the RTSP implementation because the standard defines a number of methods that are recommended or optional and others that are required. It is also possible to modify RTSP to facilitate custom functionality by extending the command set or altering the command parameters. The following is a summary of the requests commonly used by RTSP systems:

DESCRIBE: The describe command is used to by the client to retrieve a description of a presentation or media object on the server, corresponding to the URL sent in the request. The response is typically in the form of the SDP as defined in [10] and gives details such as the encoding types of the media, media duration, content authors and copyrights etc. Using this command allows clients to find out more about a clip prior to streaming and also to check if it can support the media format.

OPTIONS: This informs the sender what other valid requests it may issue i.e. what requests are supported by the corresponding client or server for the specified content at a given time. Illegal requests by either the client or server can be avoided with this operation. SETUP: Transport of the requested media is configured using this command. Details such as the transport protocol and the port numbers to use are submitted to the server so the content is delivered in a manner appropriate for the client.

PLAY: Requesting PLAY tells the server to start transmitting the requested media content as specified in the associated SETUP command. Unless a SETUP request has been issued and acknowledged, it is illegal to call the PLAY command. The absolute playback time and duration are also specified in this command so operation similar to fast forward and rewind on a VCR can be achieved with this command if the media can support such functionality e.g. live video streams cannot be scanned ahead.

PAUSE: The PAUSE request, as the name suggests, temporarily interrupts the delivery of the media currently playing in the session. PLAY must have been successfully requested in order to allow pausing of a video stream to a client. Resuming a paused media session is achieved using the PLAY request.

TEARDOWN: Termination of a stream of or session can be achieved with this request. Once this command has been successfully issued the SETUP request must be called again before media content can be streamed again.

Other optional requests defined in the RTSP standard include ANNOUNCE, SET_PARAMETER, GET_PARAMETER, RECORD and REDIRECT, but due to the fact that these methods are optional and not frequently used for straightforward streaming scenarios as used in this thesis, their descriptions are omitted but more information can be found in [6].

In addition to the session numbers previously mentioned, states for each session are also maintained by the server to ensure that only valid requests are processed and that an error response is replied to invalid requests. For example, it is illegal for a client to call the PLAY request before a successful SETUP request has been issued. To aid the server in determining if a request is valid a number of possible server states are used:

1. **Init** - the initial state meaning that the server has received no valid SETUP request.

2. **Ready** - the previous SETUP request was successful and acknowledged and the server is waiting to start or finish playing or a valid PAUSE request has been called.

3. **Playing** - a previous PLAY request was successful and content is currently being streamed to the client.

A fourth state is also used when the RECORD request is implemented to indicate when the server is actively recording. Clients may also maintain a similar set of states to allow the server issue requests to the client such

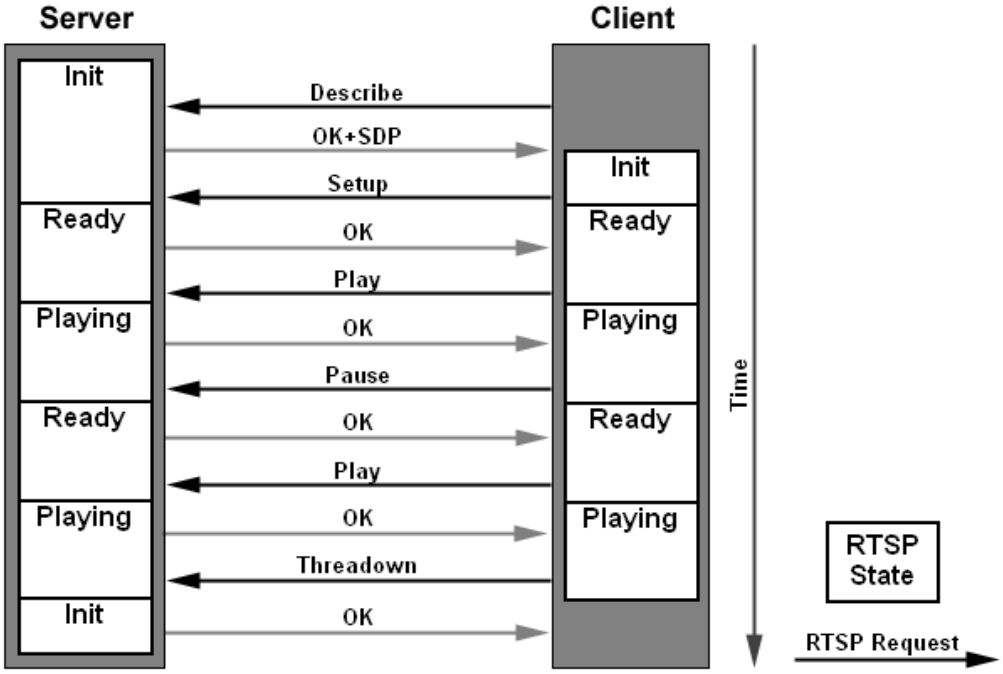

Figure 2.12: An example of an RTSP session between a client and a server. Both the states and requests of both the client and server are shown.

as OPTIONS. Figure 2.12 illustrates an example of an RTSP interaction between a client and server, highlighting the client and server states in response to different RTSP requests. In the session shown, the clients asks for a description of some content contained on the server using the DESCRIBE command and the server delivers information in SDP format with relevant details of the media queried by the client. The client then issues a SETUP request and the server configures the delivery of the stream to suit the clients preferred setup. PLAY is then sent by the client and the server starts transmitting the media data. A PAUSE request then prompts the server to halt the stream temporarily and the server waits in the Ready state for the client issue further instructions. Eventually the client requests PLAY and stream resumes. Finally, the client sends a TEARDOWN request and the server terminates the session and returns to the Init state waiting for other sessions to be established.

So far, the techniques for establishing streaming sessions have been explained but little has been discussed in relation to how exactly the media content is carried to the client. As mentioned previously, the SETUP re-

quest of RTSP allows the client to select the transport protocol for the media stream, but there are many options that can be used for this purpose. In the section that follows, a discussion of the most common protocols used for media streaming are discussed.

2.5 UMTS streaming

As mentioned in 2.2 the UMTS is designed especially for the needs of packet-switched services like streaming. For streaming in this network architecture, most recommended protocols are IP/UDP/RTP. The IP protocol, which is connection-less treats each packet independently. For streaming in the IP protocol there is encapsulated the UDP protocol. The UDP is useful when TCP would be too complex and too slow like in our case, where TCP acknowledgment cost much of bandwidth, isn't quick enough and makes streaming slow. The RTP provides functions for end-to-end transport of real-time data, such as video and other content (RTP supports both unicast and multicast transmitions). Using of IP/UDP/RTP protocols is recommended by 3rd generation partnership project (3GPP) TR 26.937.

Chapter 3

Video coding standards

3.1 Coding algorithm

All of the major video coding standards (MPEG-2 (H.262), MPEG-3, and MPEG-4 (H.263), H.264) are based on block-based hybrid coding scheme. It is hybrid in the sense that it uses motion compensated prediction to utilize the temporal redundancy, and transform coding to reduce the spatial redundancy.

3.1.1 Hybrid encoder

The hybrid encoder separates the picture information into a temporal and a spatial component. The temporal component is used for motion compensated prediction of each picture. The spatial component is transformed, quantized and encoded.

The hybrid coding scheme is shown in Figure 3.1. The T box transforms the input picture from the spatial domain to the frequency domain, and the Q box quantizes the coefficients to discrete values. The T-1 and Q-1 boxes perform the inverse operations. The Variable Length Coding (VLC) makes the code word assignment.

3.1.2 Motion compensated prediction

The purpose of motion compensated prediction is to remove the redundancy between two adjacent pictures. The prediction is performed on a macro block basis. As a prediction, the corresponding macro block in the preceding picture is used. Since only the difference between the two macro blocks is encoded, the information to encode is reduced substantially.

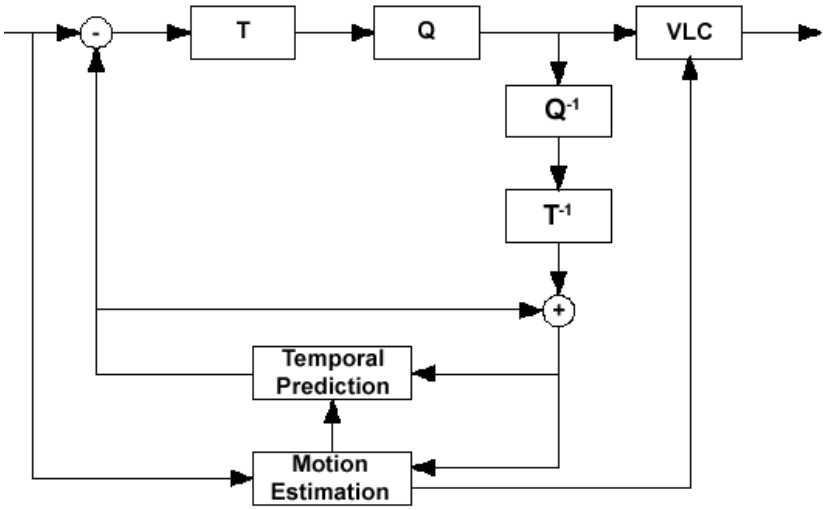

Figure 3.1: The hybrid coding scheme.

The prediction can be improved by the use of motion compensation. The idea is to estimate how the current macro blocks have moved compared to the macro blocks in the preceding picture. By searching the nearest neighborhood of the macro block in the previous picture, the best matching macro block is found. This macro block is used as prediction combined with the displacement. This improves the prediction when motion is present in the picture.

The search area is restricted to 31 pixels by 31 lines, which is approximately twice the size of a macro block. The displacement of the matching macro block is represented by a motion vector with one horizontal and one vertical component. The motion vector is Huffman coded and transmitted to the decoder along with the macro block data.

3.1.3 Macro block coding

In order to reduce the spatial redundancy within a frame the pictures are transformed and quantized. Each block is transformed from the spatial domain to the frequency domain with the discrete cosine transform (DCT). The DCT resembles the Fourier transform, but without imaginary components. The transformation of one block results in a matrix with 64 coefficients. Thus, each macro block consists of six DCT matrices, four matrices for the luminance component and two for the chrominance components.

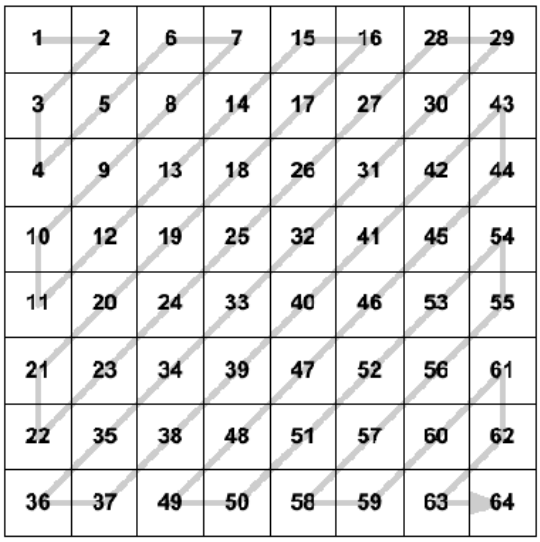

Figure 3.2: The DCT coefficient matrix and the zig-zag scanning pattern.

The values in the DCT matrices are continuous and must be quantized to discrete values in order to make it possible to represent them with binary arithmetic. The granularity of the quantizer can be set in the range from 1 to 31, where 1 represents the finest quantization and 31 the coarsest.

The transformation places the low frequency values in the upper left corner in the DCT matrix shown in Figure 3.2. Normally, pictures mainly consist of low frequency components which imply that the DCT coefficients in the upper left corner of the matrix will have relatively high values. Since the high frequency components are less common there will be many coefficients with zero value especially in the lower right part of the matrix.

To enable efficient encoding of the DCT matrices, the coefficients are encoded in a diagonal fashion called zig-zag scanning shown in Figure 3.2. The numbers in the matrix indicates the order of the encoding.

When there is a lot of zeros in the DCT matrix it is not efficient to encode every zero independently. Instead, the zig-zag scanning pattern described above makes it possible to use a method called run-length coding. With this method the number of zeros between the non-zero valued coefficients is encoded. Thus, a more effective encoding of the DCT matrices is achieved, especially when there are many consecutive zeros in the matrix.

The run-length coding results in a run-length table, are in turn Huffman coded. Huffman coding is a variant of entropy coding, which means that symbols occurring frequently are given shorter code words than infrequent

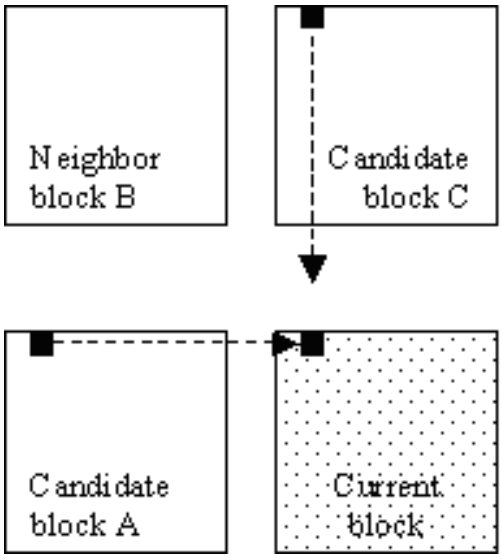

Figure 3.3: Candidate blocks for coefficient prediction.

ones. This results in an increased coding efficiency. Since the Huffman code words are of variable length, this coding scheme belongs to the family of VLCs.

3.1.4 Quantization coefficient prediction

The average energy of the quantized coefficients can be further reduced by using prediction from neighboring blocks. The prediction can be performed from either the block above, the block to the left, or the block above left, as illustrated in Figure 3.3. The direction of the prediction is adaptive and is selected based on comparison of horizontal and vertical DC gradients (increase or reduction in its value) of surrounding blocks A, B, and C.

There are two types of prediction possible, DC prediction and AC prediction:

DC prediction: The prediction is performed for the DC coefficient only, and is either from the DC coefficient of block A, or from the DC coefficient of block C.

AC prediction: Either the coefficients from the first row or the coefficients from the first column of the current block are predicted from the cosited coefficients of the selected candidate block. Differences in the

quantization of the selected candidate block are accounted for by appropriate scaling by the ratio of quantization step sizes.

3.2 Picture overview

3.2.1 Picture types

In the H.263 and MPEG-4 standards three basic picture types are defined, namely I-pictures, P-pictures and B-pictures. Other types of pictures are derived from these three types, such as PB-pictures, El-pictures and EP-pictures but to them (including B-pictures) we will not addict in this work because they are not supported by recommended video codec profile (for more information see 3.3).

I-pictures

An intra coded picture, or I-picture, is encoded independently of other pictures, i.e. without motion compensated prediction. The first picture in an image sequence must by definition be an I-picture. Since I-pictures are only by transform coded they require a lot more memory compared with P-pictures. The fact that I-pictures are independent of other pictures makes them useful as random access points in the image sequence. They can also be used to limit the propagation of transmission errors associated with sending P-pictures over packet-loss-ridden networks. Periodical insertion of I-pictures is called intra refresh.

P-pictures

A predictive coded picture, or P-picture, is encoded with both motion compensated prediction and transform coding. Thus, it is dependent on the previous picture, which can be either an I-picture or a P-picture, as shown in Figure 3.4. Since only the difference between the current picture and its predicted estimation is encoded, the compression is heavily improved. Predictive coding is sometimes also referred as inter coding.

3.2.2 Picture segmentation

In order to provide more efficient encoding the pictures are segmented in smaller parts. The division of pictures used in H.263 and MPEG-4 follows the hierarchy shown in Figure 3.5.

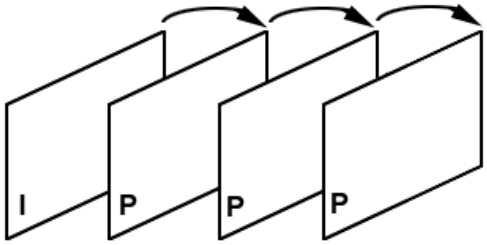

Figure 3.4: I- and P-pictures and their prediction reference.

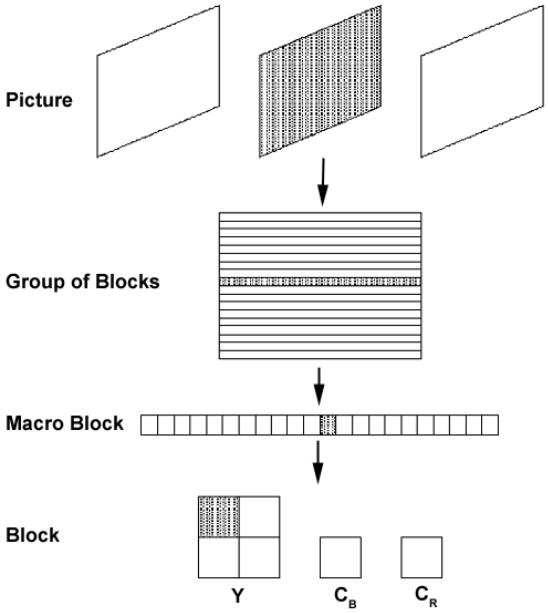

Figure 3.5: Hierarchy of picture segmentation for the CIF picture format.

Picture

The picture constitutes the highest entity in the picture segmentation hierarchy. The picture data consists of a picture header followed by either group of blocks (GOB), or slices. The picture header begins with the picture start code (PSC), and the temporal reference which makes it possible to keep the encoder and the decoder synchronized on the picture level.

Group of blocks

Each picture is divided in either GOBs or slices. Each GOB contains the picture information for one row in the picture. A GOB is comprised of 16 x k lines, where k depends on the number of lines in the employed picture format. If the CIF format is used each GOB will consist of 16 lines and thus, each picture will have 18 GOBs. The GOBs are numbered from the top to the bottom row, beginning with zero for the uppermost GOB.

The GOB data consists of a GOB header followed by a group of macro blocks. The GOB header begins with the GOB start code, and the group number. These fields provide synchronization for each GOB and contain information of its position in the picture.

Slices

Slices are similar to GOBs in that they consist of a group of macro blocks. The difference is that slices can have more flexible shape, since slices do not have to begin and end at specific positions in the picture. The slice data is almost equal to the GOB data.

Macro block

A macro block, MB, has the size of 16 pixels by 16 lines for the luminance component and 8 pixels by 8 lines for the chrominance components. If the CIF picture format is used, each GOB will consist of 22 macro blocks.

Block

If the macro blocks were to be transformed with the DCT algorithm described below, the calculations would be very cumbersome. Therefore the macro blocks are divided into blocks with the size of 8 pixels by 8 lines. A macro block thus consists of four luminance blocks and two corresponding chrominance blocks.

3.3 3GPP specification for video codecs

ITU-T Recommendation TS 26.234 defines any types and profiles of video codecs. According to ITU-T Recommendation H.263 [15] profile 0 level 10 shall be supported in our case. This is the mandatory video decoder for the PSS. In addition, PSS should support:

- H.263 [16] Profile 3 Level 10 decoder;
- MPEG-4 Visual Simple Profile Level 0 decoder, [17] and [18].

These two video decoders are optional to implement.

NOTE: ITU-T Recommendation H.263 profile 0 has been mandated to ensure that video-enabled PSS supports a minimum baseline video capability. Both H.263 and MPEG-4 visual decoders can decode an H.263 profile 0 bitstream. It is strongly recommended, though, that an H.263 profile 0 bitstream is transported and stored as H.263 and not as MPEG-4 visual (short header), as MPEG-4 visual is not mandated by PSS.

3.3.1 H.263 standard

H.263 is a provisional ITU-T standard; it is due to be published sometime in 1995/1996. It was designed for low bit-rate communication; early drafts specified data-rates less than 64 Kbits/s; however this limitation has now been removed. It is expected that the standard will be used for a wide range of bit-rates, not just low bit-rate applications.

The coding algorithm of H.263 is similar to that used by H.261, however with some improvements and changes to improve performance and error recovery. Half pixel precision is used for motion compensation whereas H.261 used full pixel precision and a loop filter. Some parts of the hierarchical structure of the data-stream are now optional, so the codec can be configured for a lower data-rate or better error recovery. There are now four optional negotiable options included to improve performance: Unrestricted Motion Vectors, Syntax-based arithmetic coding, Advance prediction, and forward and backward frame prediction similar to MPEG called P-B frames.

Source format

The source encoder operates on non-interlaced pictures with a source format defined in terms of:

- The picture format, as determined by the number of pixels per line, the number of lines per picture, and the pixel aspect ratio

Picture Format	Pixels	Lines
Sub-QCIF	128	96
QCIF	176	144
CIF	352	288
4CIF	704	576
16CIF	1408	1152

Figure 3.6: Picture formats in H.263.

- The timing between the pictures, as determined by the picture clock frequency

In the H.263 video coding standard there are five standardized picture formats, sub-QCIF, QCIF, CIF, 4CIF and 16CIF, which have sizes according to table 3.6. The different formats are based on the CIF format, which is an acronym for Common Intermediate Format. CIF has 352 pixels per line, 288 lines, a pixel aspect ratio of 12:11, and a picture clock frequency of 30 000/ 1 001, which is approximately 29.97 pictures per second.

The time interval between two pictures, or frames, is specified by the frame rate, which is measured in the number of frames per second. The output frame rate from the source encoder is highly dependent on the encoder and the transmission capacity, and is often lower than the information source frame rate. For applications such as video conferencing the frame rate usually falls in the range between 10 and 20 frames per second and for wireless streaming is value of frame rate up to 15 frames per second.

3.3.2 Levels and profiles

According to ITU-T Recommendation H.263 [15] profiles 0 and 3 with level 10 of performance capability they will be supported for video codec in the 3GPP specification [1] for video streaming in wireless environment. Theirs overview is thereinafter.

The baseline profile (Profile 0)

The Baseline Profile, designated as Profile 0, is defined to provide a profile designation for the minimal "baseline" capability of ITU-T Recommendation H.263 [15]. "Baseline" refers to the syntax of ITU-T Recommendation with no optional modes of operation. This profile of support is composed of only the baseline design.

Interactive and streaming wireless profile (Profile 3)

The Version 2 Interactive and Streaming Wireless Profile, designated as Profile 3, is defined to provide enhanced coding efficiency performance and enhanced error resilience for delivery to wireless devices within the feature set available in the second version of [15] (which did not include Annexes U, V, and W). This profile of support is composed of the baseline design plus the following modes:

Advanced INTRA coding - Use of this mode improves the coding efficiency for INTRA macroblocks (whether within INTRA pictures or predictive-coded pictures). The additional computational requirements of this mode are minimal at both the encoder and decoder (as low as a maximum of 8 additions/subtractions per 8 x 8 block in the decoding process plus the use of a different but very similar VLC table in order to obtain a significant improvement in coding efficiency). For these reasons, Advanced INTRA Coding is included in this basic package of support.

Deblocking filter - Because of the significant subjective quality improvement that may be realized with a deblocking filter, these filters are widely in use as a method of post-processing in video communication terminals. Annex J of [15] represents the preferred mode of operation for a deblocking filter because it places the filter within the coding loop. This placement eases the implementation of the filter (by reducing the required memory) and somewhat improves the coding performance over a post-processing implementation. As with the advanced prediction mode, this mode also includes the four-motion-vectorper- macroblock feature and picture boundary extrapolation for motion compensation, both of which can further improve coding efficiency. The computational requirements of the deblocking filter are several hundred operations per coded macroblock, but memory accesses and computational dependencies are uncomplicated. This last point is what makes the deblocking filter preferable to advanced prediction for some implementations. Also, the benefits of advanced prediction are not as substantial when the deblocking filter is used as well. Thus, the deblocking Filter is included in this basic package of support.

Slice structured mode - The Slice structured mode is included here due to its enhanced ability to provide resynchronization points within the video bitstream for recovery from erroneous or lost data. Support for

the Arbitrary Slice Ordering (ASO) and Rectangular Slice (RS) submodes of the Slice structured mode are not included in this profile, in order to limit the complexity requirements of the decoder. The additional computational burden imposed by the Slice Structured mode is minimal, limited primarily to bitstream generation and parsing.

Modified quantization - This mode includes an extended DCT coefficient range, modified DQUANT syntax, and a modified step size for chrominance. The first two features allow for more flexibility at the encoder and may actually decrease the encoder's computational load (by eliminating the need re-encode macroblocks when coefficient level saturation occurs). The third feature noticeably improves chrominance fidelity, typically with little added bit-rate cost and with virtually no increase in computation. At the decoder, the only significant added computational burden is the ability to parse some new bitstream symbols.

3.3.3 MPEG-4 standard

MPEG-4 standardizes coding methods for many types of audiovisual data. MPEG-4 incorporates standardized ways to represent various media objects. These objects include aural, visual or audiovisual content which can be natural or synthetic in origin.

MPEG-4 overview

The MPEG-4 visual standard is developed to provide users a new level of interaction with visual contents. It provides technologies to view access and manipulate objects rather than pixels, with great error robustness at a large range of bit rates. Application areas range from digital television, streaming video, to mobile multimedia and games.

The MPEG-4 natural video standard consists of a collection of tools that support these application areas. The standard provides tools for shape coding, motion estimation and compensation, texture coding, error resilience, sprite coding and scalability. Conformance points in the form of object types, profiles and levels, provide the basis for interoperability.

Shape coding can be performed in binary mode, where the shape of each object is described by a binary mask, or in gray scale mode, where the shape is described in a form similar to an alpha channel, allowing transparency, and reducing aliasing.

Motion compensation is block based, with appropriate modifications for

object boundaries. The block size can be 16x16, or 8x8, with half pixel resolution. MPEG-4 also provides a mode for overlapped motion compensation.

Texture coding is based in 8x8 DCT, with appropriate modifications for object boundary blocks. Coefficient prediction is possible to improve coding efficiency. Static textures can be encoded using a wavelet transform.

Error resilience is provided by resynchronization markers, data partitioning, header extension codes, and reversible variable length codes.

Scalability is provided for both spatial and temporal resolution enhancement. MPEG-4 provides scalability on an object basis, with the restriction that the object shape has to be rectangular.

MPEG-4 conformance points are defined at the Simple profile, the Core profile, and the Main profile. Simple profile and Core profiles address typical scene sizes of QCIF and CIF size, with bit rates of 64 kbit/sec, 128 kbit/sec, 384 kbit/sec, and 2Mbit/sec. Main profile addresses typical scene sizes of CIF, ITU-R 601, and HD, with bit rates at 2 Mbit/sec, 15 Mbit/sec and 38.4 Mbit/sec.

Simple profile

The Simple profile provides efficient, error resilient coding of rectangular video objects, suitable for applications on mobile networks.

I-video object plane (VOP) and P-VOP are supported. Advanced intra coding with intraframe prediction for the DCT coefficients (AC/DC-Prediction) is mandatory for this profile and other VLC-tables were used. Four vectors per macroblock can be used - four motion vectors are used for one macroblock (one vector for each 8x8 block). And unrestricted motion vectors and long vectors can be used too - a greater search area for motion estimation is used and the vectors may point outside the picture area; vectors may be longer than 16 pixels.

Chapter 4

Streaming server and client

4.1 Streaming media

Streaming technology offers a significant improvement over the download-and-play approach to multimedia file distribution, because it allows the data to be delivered to the client as a continuous flow with minimal delay before playback can begin. The multimedia data arrives is briefly buffered before being played, and is then discarded. It is never actually stored on the users' computer.

4.2 Streaming servers

Streaming media is a real time operation - an exact number of bits delivered on a precise timeline. Streaming media servers like the Darwin Streaming Server (DSS) (by QuickTime), Helix Server (by RealNetworks), Media Server (by Microsoft) and more are designed to understand the structure and bitrates of the media files they deliver and send out the bits in a carefully controlled flow.

RealNetworks server software (Helix server) has matured over many years and has become a very stable, fast and user-friendly platform. It has full remote administration capabilities and is very secure. It supports Microsoft Windows NT/2K/XP/.NET, all Linux platforms, FreeBSD and Solaris operating systems. Installation and configuration of RealSystem server is straightforward and very customisable, allowing it to fit within an already existing service, i.e. a web server or other streaming service. It also supports streaming of Apple QuickTime, both archive and live streaming. Real Helix server, compared to others, streams video files according to another one file containing information about video data encapsulation into packets. It provides

unlimited stream capacity in the unlimited version. The major disadvantage to using RealSystem Server is cost.

Unlike the RealNetworks offering, Media Server is not a standalone product, but comes bundled with Windows 2003 Server. If you already have either of these servers installed, the cost of implementation is minimal and is a worthwhile investment of a small amount of time. Bear in mind, that Media Server will only run on a Windows Server. A major advantage though is the ability to stream to over 3,000 users per Media Server without incurring licensing costs. Windows Media Server is, like RealNetworks, a fast and efficient platform although your media files are not quite as secure. This does not mean that your server can be accessed or attacked through Windows Media Server. The server, its services and files are protected and cannot be tampered with. However, applications have been written that allow users to record the stream as it comes down from the Media Server but these are not widespread and relatively rare in use.

Apple QuickTime Streaming Server is a Macintosh-only based solution. For Macintosh infrastructures it offers a wide range of streaming file compatibility, including standard MOV, MP3, MPEG-4 and Shockwave Flash, to over 4,000 users per server. QuickTime's major selling point is the quality of footage that can be produced using its Sorenson encoding protocol. For example, most major film companies will release movie trailers onto the Internet in this format due to its high quality reproduction.

Apple also supply in compared by previous servers a free PERL-based server called Darwin for the following operating systems: Red Hat Linux 9, Mac OS X 10.3.9 and later and MS Windows 2000/2003 Server. Darwin is not a very user-friendly server system. Being script-based, all configurations have to be done from web-based administration tool or from within text documents (then PERL setup needs to be done by experienced administrator). However it offers a wide range of compatible streaming files and can stream to 2,000 or more users. Darwin uses movies specially prepared for streaming by hinting (see section 5.1). The major advantage of Darwin is its open-source core what makes possible to modify source codes in order to Test bed developing. Darwin is only one open source server supporting 3GP file format. Due to these facts, Darwin is suitable streaming server for this project.

4.3 Content serving by DSS

A RealVideo or Windows Media files are already structured specifically for streaming. The server can read the data rate and some information about how the file should be broken into network packets right from the video file

itself. When a new codec or video format is introduced, the server usually will need to be updated with a plug-in so that it knows how to read the new type.

Unlike these other formats, QuickTime file format supported by DSS isn't really an audio or video format. It's a container that wraps around other media objects, including video and audio files. A QuickTime movie file can contain a number of different codecs and media types, each in its own track. Some of the tracks might have their data directly inside the QuickTime movie, such as video or audio tracks. Others might be referenced by the movie, such as the URLs of slide images that should appear at certain intervals. A QuickTime movie might really be a container for an H.263 video file, an .au audio file, and a set of jpeg images all designed to play seamlessly together. Or it might be a single advanced audio coding (AAC) audio track; or perhaps a QDesign audio track with a Sorenson video track. The tracks encapsulated in the QuickTime file are called hint tracks (see section 5.1). More about serving streaming movies by DSS is described in the following subsection.

4.3.1 Serving streaming movies

DSS is equipped with RTP server software that understands the QuickTime file format including the structure of hint tracks. DSS contains RTSP controller application too.

The server uses the hint tracks in a streaming QuickTime movie to packetize the movie's media into RTP streams. Hint tracks are added to QuickTime movies to prepare them for streaming over RTP. One hint track is added to the movie for each track whose media will be streamed, as illustrated in Figure 4.1.

If server is sending a unicast of a hinted movie, the QuickTime movie controller will provide the client with the ability for pause the movie and to skip backward and forward in movie time. This will be communicated to your server over RTSP.

The RTP server does not need to know about QuickTime media types or codecs. The hint tracks within the movie file provide the information needed to turn QuickTime media into RTP packets. Each hint track contains the data needed to build packet headers for a specific track's media. The hint track also supplies a pointer to the media data that goes into each packet.

The RTP server is be able to parse a QuickTime movie file sufficiently to find each hint track, then to find the track and sample data that the hint track points to. The hint track contains any precalculated values that may be needed, making it easier for the server to create the RTP packets.

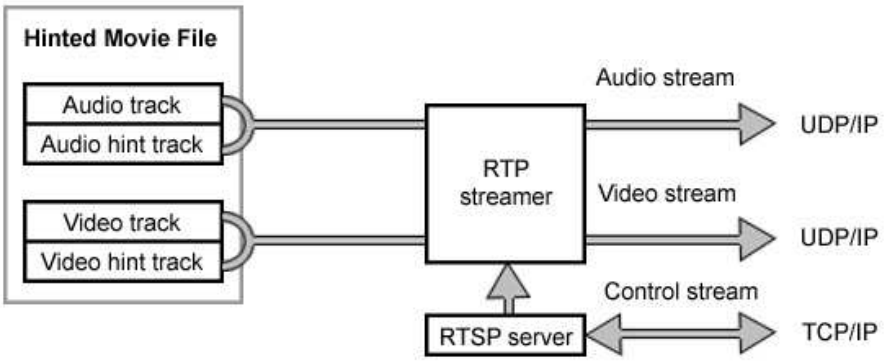

Figure 4.1: Streaming a hinted movie.

Hint tracks offload a great deal of computation from the RTP server. Consequently, you may find that an RTP server is able to send data more efficiently if it is contained in a QuickTime movie, even if the RTP server already understands the media type, codec, and RTP packing being used.

For example, the H.263+ video codec is an IETF-standard RTP format which server already understand, but creating an H.263+ stream from video data requires complex calculations to properly determine packet boundaries. This information is precalculated and stored in the hint track when H.263+ video is contained in a QuickTime movie, allowing server to packetize the data quickly and efficiently.

4.4 Client players

Streaming video products use a process identical to that of audio compression: the encoder software uses a proprietary algorithm to create a compact form of the original file. The coder shrinks video by replacing the original frames with more compact versions by using mathematical algorithms such as wavelet, fractal, and DCT. Encoders are also based on standards such as MPEG-4 and H.263. The effectiveness of the encoding process can be judged by video output quality, such as the level of artifacts, or objects introduced to the picture by compression. Decoders, or players, decompress and play the video.

For each video clip, player keeps a buffer to smooth out the video stream because of changes in bandwidth, lost packets or jitter (buffer is a temporary storage area, usually in the Random Access Memory (RAM) of a computer or peripheral device, where data can be collected and/or held for ongoing or later

processing). Video data, frames, are segmented into fixed size packets which arrive at the receiver with an interarrival rate. There are packets assembled to form a frame again, then decompressed and stored in a buffer and waits there until it is fetched by a display process. Frames must be available to the display process at a precise rate (e.g., one frame every 33 ms). If there is no frame available in the buffer then a gap in the display arises. If only one frame were stored in the buffer, then the display of the video would be as burstly as the pipeline of all the preceding self-similar processes. Every player has its own solution of buffering technology (Apple player has buffer size from 10 to 12 seconds and both RealNetworks' and Microsofts' players from 15 to 20 seconds, but most of the players have adjustable buffer size for video buffering).

Chapter 5

Payload mapping

DSS was chosen as a streaming server thanks to its open-source core. Open source core allows modifications in the DSS according to ours visions.

Video files for streaming through DSS need to be hinted. Hinted files contain information about video packetization and video properties in a structured file format. These data are important for the Test bed, which will insert errors into the stream of video packets according to our settings. More about hinting and QuickTime file format is written in this chapter.

5.1 Hint tracks

A 3gp movie is prepared for RTP streaming by adding hint tracks. Hint tracks allow a server to stream movies without requiring the server to understand movie media types, codecs, or packing. The server needs to understand the structured file format sufficiently to find tracks and media samples in a movie.

Each track in a hinted - structured movie is sent as a separate RTP stream, and the recipe for packetizing each stream is contained in a corresponding hint track. Each sample in a hint track tells the server how to packetize a specific amount of media data. The hint track sample contains any data needed to build a packet header of the correct type, and also contains a pointer to the block of media data that belongs in the packet.

There is at least one hint track for each media track to be streamed. It is possible to create multiple hint tracks for a given track's media, optimized for streaming the same media over networks with different packet sizes, for example. We used QuickTime Pro for video hinting. The hinter included in QuickTime creates one hint track for each track to be streamed.

Hint tracks are structured in the same way as other movie tracks used in MPEG-4 or H.263. Standard data structures, such as track references, are

used to point to data. In order to serve a movie, the server must locate the hint tracks contained in the movie and parse them to find the packet data they point to.

5.2 Payload mapping

Hinted video contains information about e.g. frame rate, I-frames, maximum packet size. These data are stored in a head composing of atoms placed at the beginning or at the end of video file. This section includes information about atoms - what are they and how are they use.

5.2.1 Introduction to Atoms

This chapter describes how structured movies are stored on disk. The structured file format is designed to accommodate the various kinds of data that need to be stored in order to work with digital media. Because the file format can be used to describe almost any media structure, it is a suitable format for the exchange of digital media between applications, independent of the platform on which the application is running.

Before explaining the specifics of how a structured movie is stored, it is important to first understand the basic units that are used to construct structured files. Structured movies uses two basic structures for storing information: classic atoms and atom container atoms. Both of them allow constructing arbitrarily complex hierarchical data structures and also allow applications to ignore data they don't understand.

Structured movie atom containers provide a basic structure for storing information in movie. An atom container is a tree-structured hierarchy of atom container atoms.

Atoms

The basic data unit in a structured file is the atom. Each atom contains size and type information along with its data. The size field indicates the number of bytes in the atom, including the size and type fields. The type field specifies the type of data stored in the atom and, by implication, the format of that data.

Atom types are specified by a 32-bit integer, typically a four-character code. Unless otherwise stated, all data in a structured movie is stored in big-endian (network) byte ordering. All version fields must be set to 0, unless this document states otherwise.

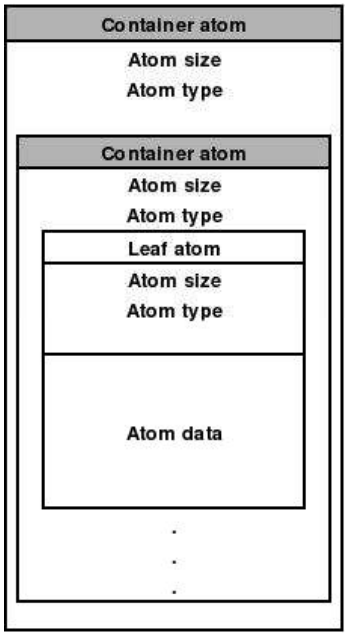

Figure 5.1: A sample QuickTime atom

Atoms are hierarchical in nature. That is, one atom can contain one or more other atoms of varying types. For example, a movie atom contains one track atom for each track in the movie. The track atoms, in turn, contain one or more media atoms each, along with other atoms that define other track and movie characteristics. This hierarchical structure of atoms is referred to as a containment hierarchy.

The format of the data stored within a given atom cannot be determined based only on the type field of that atom. That is, an atom's use is determined by its context. A given atom type can have different usages when stored within atoms of different types. This means that all structured file readers must take into consideration not only the atom type, but also the atom's containment hierarchy.

Figure 5.1 shows the layout of a sample structured atom. Each atom carries its own size and type information as well as its data. The name of a container atom (an atom that contains other atoms, including other container atoms) is printed in a gray box, and the name of a leaf atom (an atom that contains no other atoms) is printed in a white box. Leaf atoms contain data, usually in the form of tables.

Atom type	Use
'free'	Unused space available in file.
'skip'	Unused space available in file.
'wide'	Reserved space-can be overwritten by an extended size field.
'pnot'	Reference to movie preview data.
'moov'	Movie resource—meta-data about the movie (number and type of tracks, location of sample data, and so on).
'mdat'	Movie sample data—usually this data can be interpreted only by using the movie resource.

Figure 5.2: Basic atom types of a QuickTime file

QuickTime files

As was previously mentioned, we used QuickTime Pro by Apple for hinting of movies. Apple brings into atoms its own names for atom types and terminology (e. g. atom container atoms is called QT atom). Files hinted by QuickTime consist of atoms, each with an appropriate atom type. A few of these types are considered basic atom types and form the structure within which the other atoms are stored. Table 5.2 lists the currently supported basic atom types.

5.2.2 Movie Atoms

QuickTime movie atoms have an atom type of 'moov'. These atoms act as a container for the information that describes a movie's data. This information, or meta-data, is stored in a number of different types of atoms. Generally speaking, only meta-data is stored in a movie atom. Sample data for the movie, such as audio or video samples, are referenced in the movie atom, but are not contained in it.

The movie atom is essentially a container of other atoms. These atoms, taken together, describe the contents of a movie. At the highest level, movie atoms typically contain track atoms, which in turn contain media atoms. At the lowest level you find the leaf atoms, which contain non-atom data, usually in the form of a table or a set of data elements. For example, the track atom contains the edit atom, which contains a leaf atom called the edit list atom. The edit list atom contains an edit list table.

Figure 5.3 provides a conceptual view of the organization of a simple, one-track QuickTime movie. Each nested box in the illustration represents an atom that belongs to its parent atom. The figure does not show the data

regions of any of the atoms. Some of these areas (which are needed for video payload mapping) are described in the sub-sections that follow.

Media Header Atoms

The media header atom specifies the characteristics of a media, including time scale and duration. The media header atom has an atom type of *'mdhd'*.

Figure 5.4 shows the layout of the media header atom.

Media header atoms contain the following data elements:

Size A 32-bit integer that specifies the number of bytes in this media header atom.

Type A 32-bit integer that identifies the atom type; this field must be set to 'mdhd'.

Version One byte that specifies the version of this header atom.

Flags Three bytes of space for media header flags. Set this field to 0.

Creation time A 32-bit integer that specifies (in seconds since midnight, January 1, 1904) when the media atom was created.

Modification time A 32-bit integer that specifies (in seconds since midnight, January 1, 1904) when the media atom was changed.

Time scale A time value that indicates the time scale for this media-that is, the number of time units that pass per second in its time coordinate system.

Duration The duration of this media in units of its time scale.

Language A 16-bit integer that specifies the language code for this media.

Quality A 16-bit integer that specifies the media's playback quality-that is, its suitability for playback in a given environment.

Time-to-Sample Atoms

Time-to-sample atoms store duration information for a media's samples, providing a mapping from a time in a media to the corresponding data sample. The time-to-sample atom has an atom type of *'stts'*. You can determine the appropriate sample for any time in a media by examining the time-to-sample atom table, which is contained in the time-to-sample atom.

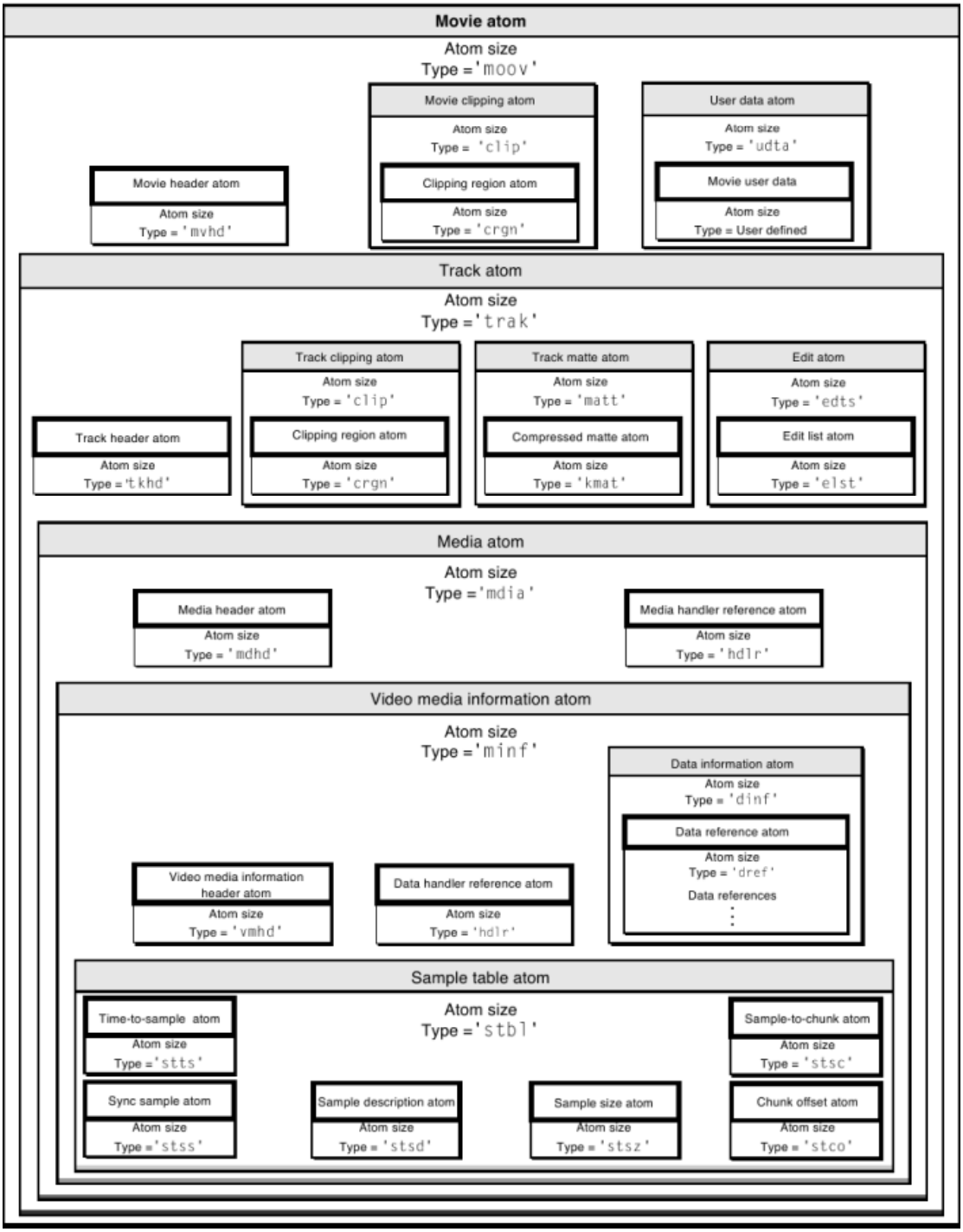

Figure 5.3: Sample organization of one-track video movie

Media header atom	Bytes
Atom size	4
Type = 'mdhd'	4
Version	1
Flags	3
Creation time	4
Modification time	4
Time scale	4
Duration	4
Language	2
Quality	2

Figure 5.4: The layout of a media header atom

The atom contains a compact version of a table that allows indexing from time to sample number. Other tables provide sample sizes and pointers from the sample number. Each entry in the table gives the number of consecutive samples with the same time delta, and the delta of those samples. By adding the deltas, a complete time-to-sample map can be built.

The atom contains time deltas: *DT(n+1) = DT(n) + STTS(n)* where *STTS(n)* is the (uncompressed) table entry for sample n and DT is the display time for sample (n). The sample entries are ordered by time stamps; therefore, the deltas are all nonnegative. The DT axis has a zero origin; *DT(i) = SUM* (for *j=0* to *i-1* of *delta(j)*), and the sum of all deltas gives the length of the media in the track (not mapped to the overall time scale, and not considering any edit list). The edit list atom provides the initial DT value if it is nonempty (nonzero).

Figure 5.5 shows the layout of the time-to-sample atom.

The time-to-sample atom contains the following data elements:

Size A 32-bit integer that specifies the number of bytes in this time-to-sample atom.

Type A 32-bit integer that identifies the atom type; this field must be set to 'stts'.

Version A 1-byte specification of the version of this time-to-sample atom.

Flags A 3-byte space for time-to-sample flags. Set this field to 0.

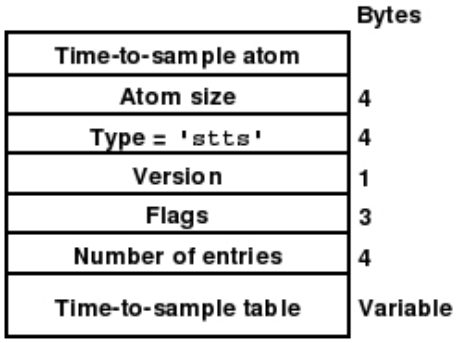

Figure 5.5: The layout of a time-to-sample atom

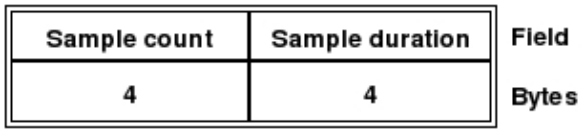

Figure 5.6: The layout of a time-to-sample table entry

Number of entries A 32-bit integer containing the count of entries in the time-to-sample table.

Time-to-sample table A table that defines the duration of each sample in the media. Each table entry contains a count field and a duration field. The structure of the time-to-sample table is shown in Figure 5.6.

You define a time-to-sample table entry by specifying these fields:

Sample count A 32-bit integer that specifies the number of consecutive samples that have the same duration.

Sample duration A 32-bit integer that specifies the duration of each sample.

Sample duration

Entries in the table describe samples according to their order in the media and their duration. If consecutive samples have the same duration, a single table entry can be used to define more than one sample. In these cases, the count field indicates the number of consecutive samples that have the same duration. For example, if a video media has a constant frame rate, this

Sample count	Sample duration
4	3
2	1
3	2

Figure 5.7: An example of a time-to-sample table

table would have one entry and the count would be equal to the number of samples.

Figure 5.7 presents an example of a time-to-sample table that is based on the chunked media data. That data stream contains a total of nine samples that correspond in count and duration to the entries of the table shown here. Even though samples 4, 5, and 6 are in the same chunk, sample 4 has duration of 3, and samples 5 and 6 have duration of 2.

Hint Sample Data Format

The sample description atom (*'stsd'*) contains information about the hint track samples. It specifies the data format (note that currently only RTP data format is defined) and which data reference to use (if more than one is defined) to locate the hint track sample data. It also contains some general information about this hint track, such as the hint track version number, the maximum packet size allowed by this hint track, and the RTP time scale. It may contain additional information, such as the random offsets to add to the RTP time stamp and sequence number.

The sample description atom can contain a table of sample descriptions to accommodate media that are encoded in multiple formats, but a hint track can be expected to have a single sample description at this time.

The sample description for hint tracks is defined in Figure 5.8.

Field descriptions:

Size A 32-bit integer specifying the size of this sample description in bytes.

Data format A four-character code indicating the data format of the hint track samples. Only *'rtp'* is currently defined. Note that the fourth character in *'rtp'* is an ASCII blank space (hex 20). Do not attempt to packetize data whose format you do not recognize.

Field	Bytes
Size	4
Data format	4
Reserved	6
Data reference index	2
Hint track version	2
Last compatible hint track version	2
Max packet size	4
Additional data table	variable

Figure 5.8: Hint track sample description

Reserved Six bytes that must be set to 0.

Data reference index This field indirectly specifies where to find the hint track sample data. The data reference is a file or resource specified by the data reference atom (*'dref'*) inside the data information atom (*'dinf'*) of the hint track. The data information atom can contain a table of data references, and the data reference index is a 16-bit integer that tells you which entry in that table should be used. Normally, the hint track has a single data reference, and this index entry is set to 0.

Hint track version A 16-bit unsigned integer indicating the version of the hint track specification. This is currently set to 1.

Last compatible hint track version A 16-bit unsigned integer indicating the oldest hint track version with which this hint track is backward-compatible. If your application understands the hint track version specified by this field, it can work with this hint track.

Max packet size A 32-bit integer indicating the packet size limit, in bytes, used when creating this hint track. The largest packet generated by this hint track will be no larger than this limit.

Additional data table Large topic not needed for this work. For more information see [19] page 156.

Sync Sample Atoms

The sync sample atom identifies the I-frames in the media. In a media that contains compressed data, I-frames define starting points for portions of a

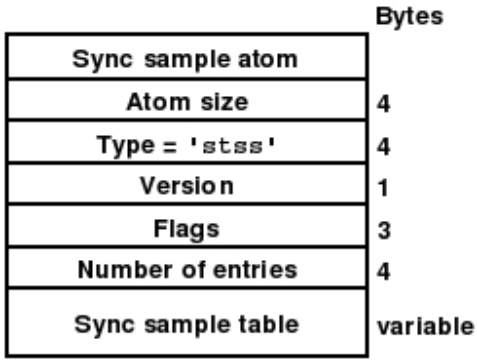

Figure 5.9: The layout of a sync sample atom

temporally compressed sequence. The I-frame is self-contained-that is, it is independent of preceding frames. Subsequent frames may depend on the I-frame.

The sync sample atom provides a compact marking of the random access points within a stream. The table is arranged in strictly increasing order of sample number. If this table is not present, every sample is implicitly a random access point.

Sync sample atoms have an atom type of *'stss'*. The sync sample atom contains a table of sample numbers. Each entry in the table identifies a sample that is a I-frame for the media. If no sync sample atom exists, then all the samples are I-frames.

Figure 5.9 shows the layout of a sync sample atom.

The sync sample atom contains the following data elements:

Size A 32-bit integer that specifies the number of bytes in this sync sample atom.

Type A 32-bit integer that identifies the atom type; this field must be set to *'stss'*.

Version A 1-byte specification of the version of this sync sample atom.

Flags A 3-byte space for sync sample flags. Set this field to 0.

Number of entries A 32-bit integer containing the count of entries in the sync sample table.

Sync sample table A table of sample numbers; each sample number corresponds to a I-frame. Figure 5.10 shows the layout of the sync sample table.

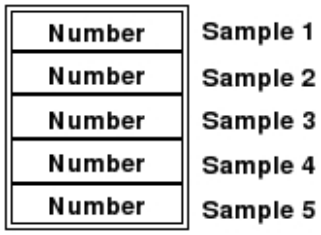

Figure 5.10: The layout of a sync sample table

Chapter 6

Error models

The UMTS system is designed to support many different services and hence various higher layer protocols are running over the system. If a new service with a certain protocol is required, it is necessary to know its performance beforehand. Especially in communications over wireless channels like in UMTS the receiver will see a higher error probability and different error characteristics compared to wired communications. Due to this fact, the channel in UMTS is highly influencing the reception of the higher layer protocols and hence the impact on their performance should be analyzed in simulations.

For error prone video streaming simulation were chosen two error models to be implemented in the Test bed. The error models insert errors into stream of video data on the IP protocol layer - into stream of IP packets, or into stream of text and timing information (TTI) blocks (on a wireless interface of the UMTS network are IP packets encapsulated into TTI blocks - for more information see 7.3.2).

Following error models are supported:

Uniform error model - adds errors into stream of IP packets with random uniform distribution of errors. Uniform distribution has constant probability. The probability density function and cumulative distribution function for a continuous uniform distribution on the interval [a,b] are shown in the Figure 6.1. The function for error generation works by generating a sequence of 48-bit integers, X_i, according to the linear congruential formula:

$X_{n+1} = (aX_n + c) \bmod m$, where $n \geq 0$

The parameter $m = 2^{48}$, hence 48-bit integer arithmetic is performed. The value returned by the function is computed by first generating the next 48-bit X_i in the sequence. Then the appropriate number of bits, according to the type of data item to be returned, is copied from the

$$P(x) = \begin{cases} 0 & \text{for } x < a \\ \dfrac{1}{b-a} & \text{for } a \le x \le b \\ 0 & \text{for } x > b \end{cases}$$

$$D(x) = \begin{cases} 0 & \text{for } x < a \\ \dfrac{x-a}{b-a} & \text{for } a \le x \le b \\ 1 & \text{for } x > b. \end{cases}$$

Figure 6.1: Probability density function and cumulative distribution function for a continuous uniform distribution on the interval [a,b]

high-order bits of X_i and transformed into the error value which is converted into error flag (error flag identify occurrence of error). More about uniform distribution is in next chapter.

Modified two-state Markov model - its properties are described in the following section due to its higher complexity.

6.1 Modified two-state Markov model

6.1.1 About measurements

All the measurements and results presented in this section are obtained from [21] and they have been realized in the live UMTS networks of three different operators in the city center of Vienna, Austria in order to find model describing the live UMTS error prone environment for data stream. For the measurements a UDP data stream with a bit rate of 360kbit/s (372kbit/s incl. UDP/IP overhead) in Down Link (DL) was used. The data was sent from a PC located at the Institute of Communications and Radio Frequency Engineering at the Vienna University of Technology to a notebook using a UMTS terminal as a modem. As depicted in Figure 6.2 the UDP data stream goes from the PC over University LAN (ethernet), internet, UMTS core network and over the UMTS air interface to a UMTS mobile which is connected via Universal Serial Bus (USB) to the notebook. A WCDMA TEMS Mobile Motorola A835 from Ericsson [20] was used as terminal. On the notebook the measurements of the mobile have been captured by 'TEMS Investigation WCDMA 2.4' software also by Ericsson.

For the evaluation of the DL dedicated channel (DCH) block error ra-

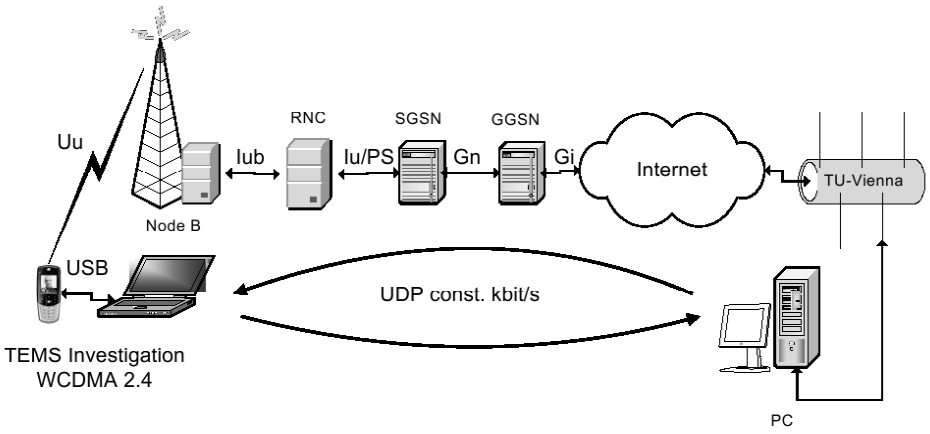

Figure 6.2: Measurement setup.

tio (BLER) statistics, several scenarios with different mobility characteristics have been considered which are called: 'static', 'small scale movement', 'walking indoor', 'tram 2', 'car Vienna/Guertel' and 'car highway Vienna-Schwechat'. The measurements for the 'static' case were performed in a room of the Institute of Communications and Radio Frequency Engineering at the Vienna University of Technology. For these measurements the UMTS terminal was lying on the table in a typical office environment. Due to few movements of persons or other objects around the mobile station, there were only little variations in the channel. The 'small scale movement' measurements were performed by a person sitting at the table and randomly tilting and moving the UMTS mobile with his hands. In the 'walking indoor' scenario, as the label indicates, the measurements were obtained while walking around in the building of the Institute of Communications and Radio Frequency Engineering.

6.1.2 Model

Due to its simplicity and great adaptability, authors decided to modify the two-state Markov model towards a modified (semi) Markov model. The scheme of the modified two-state Markov model (MTSMM) is shown in Figure 6.3. The two different states of the two-state Markov model are the state of correct TTI and the state of erroneous TTI. Here p_{cc} and p_{ee} are the probabilities of staying in the state and p_{ce} and p_{ec} are the transition probabilities from one state to the other. The error state in this model is equal to the

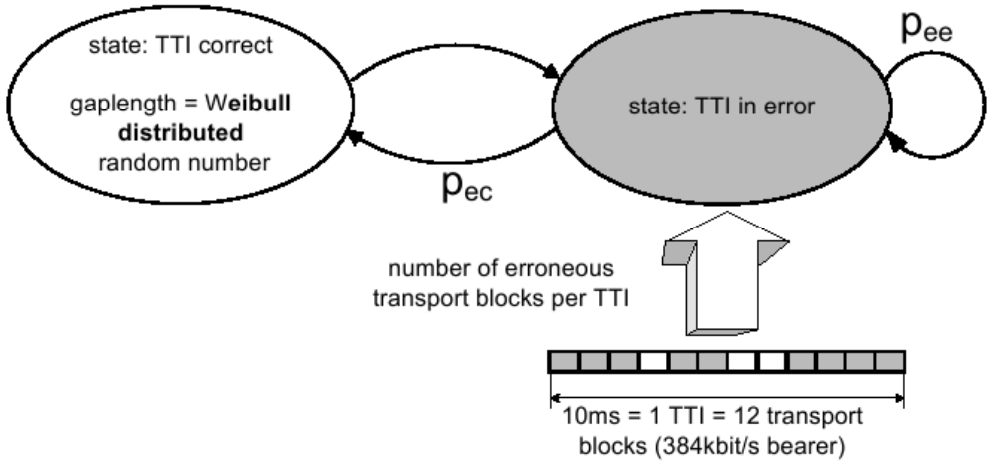

Figure 6.3: Modified two-state Markov model with Weibull distributed gaplength.

one in the simple two-state Markov model because the burstlengths are modeled properly with such configuration. However, the error-free state of the model has been modified. In that state the gaplength is now taken from a Weibull distributed random number. If the number of error free TTIs equals the gaplength, the model returns to the error-state where the burstlength is determined via the transition probabilities.

To evaluate the fitness of the model to real error characteristics of the DL DCH, the empirical CDF of the simulated BLER is represented together with the measured BLER in Figure 6.4. In the static scenario the BLER from simulation fits the measured BLER statistics very well. The simulation of the scenario with movement though does not match correctly to the measured statistics but it is close to most of the measurements.

In order to handle this problem it could be either consider a Markov model with more than two states (say N states) or use another modification of the model. Since authors' intention was to find a simple model for the TrCH BLER characteristics, authors decided not to follow the idea of an N-state Markov model because of the high number of parameters which would have to be adjusted correctly in such case.

6.2 Summary

Thanks to the two ways of error insertion it is possible to use error models for simulation of real error prone video streaming over the UMTS network

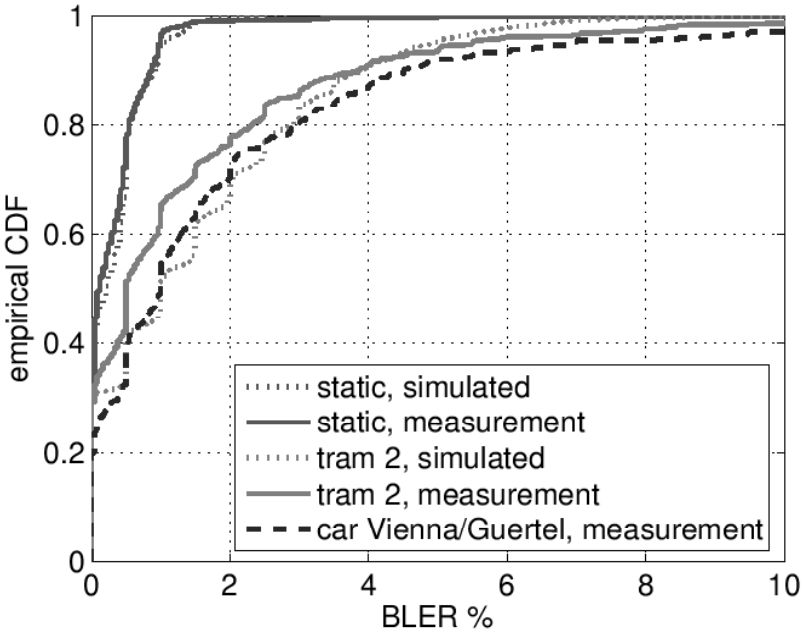

Figure 6.4: BLER %, simulation vs. measurements.

or for experiments based on errors with uniform distribution in the video stream. The first case is possible to simulate by using the MTSMM based on real measurements in the UMTS network and in the other case by using uniform error model.

For real streaming simulation of wireless UMTS channel we have parameters for MTSMM adjusting. The settings parameters are for video streaming simulation with channel of 348 kbps. For simulation of any UMTS scenario in a Test bed there should be parameters of MTSMM adjustable.

Chapter 7

Test bed

Video streaming over UMTS network is error prone and it suffers from network degradation artifacts. Artifacts are caused by the following effects: delay between packets, jitter, packet corruption and loss and bit error rate. All these facts have effect on the end user perceptual quality of multimedia service.

H.263 and MPEG4 use high temporal compression. The packet loss in P-frame causes perceptual distortion until the next I-frame. The I-frame packet loss is even more critical because the whole period between I-frames is perceptually damaged. It is shown in figure 7.1 where I-frame error caused higher perceptual distortion then error in P-frame.

For detailed investigation of influence of video streaming network degradation on perceptual video quality it is necessary to associate lost packets with distorted I or P-frames.

Further research is focused on the investigation of network and streaming settings parameters. We can set following parameters: error model, error rate, bit rate, frame rate, I-frame frequency, video codec and packet length. In the Test bed we can simulate video streaming up to UDP protocol layer and set packet loss according to two models - error model with uniform distribution of errors and MTSMM described in [21]. By using of Test bed we will be able to detect packet loss, frame type of fault frames and features of the streaming session.

The implementation consists of Darwin Streaming Server (DSS) source code modification and developing of application which can trace and control streaming up to IP layer. The application's purpose is to trace the streaming session and generate the errors according to implemented error models.

Figure 7.1: Different after-effects if error occurs in I or P frame. a) shown error occurred in I-frame, b) shown error occurred in P-frame

7.1 Network degradation artifacts

The network degradation can lead to following streaming degradation artifacts: connection failure, buffering, jerkiness, I-frame loss or scene change with P or I frame loss. Connection failure means interruption of service, buffering displays frozen picture and jerkiness is variation of frame rate.

In the figure 7.2 is shown graph of mean opinion score (MOS) of perceptual video quality in time obtained from subjective assessments. On X axis, there are time depended values and evaluative values of video quality (in range from 0 to 250 points of score) are on Y axis. It is possible to see dependency between P or I-frame error and MOS in corresponding time. People are very sensitive to I-frame loss. Every time when I-frame was lost MOS is decreasing rapidly. Less effect has P-frame loss. It is followed by a little decrease of MOS.

Test bed makes it possible to simulate video UMTS network degradation artifacts on PC. Corrupted video can be stored and evaluated. In this way we can get quality evaluation of same video from more people and final average of results is more acceptable (quality evaluation is a part of another work - Time variant video quality evaluation for UMTS video streaming).

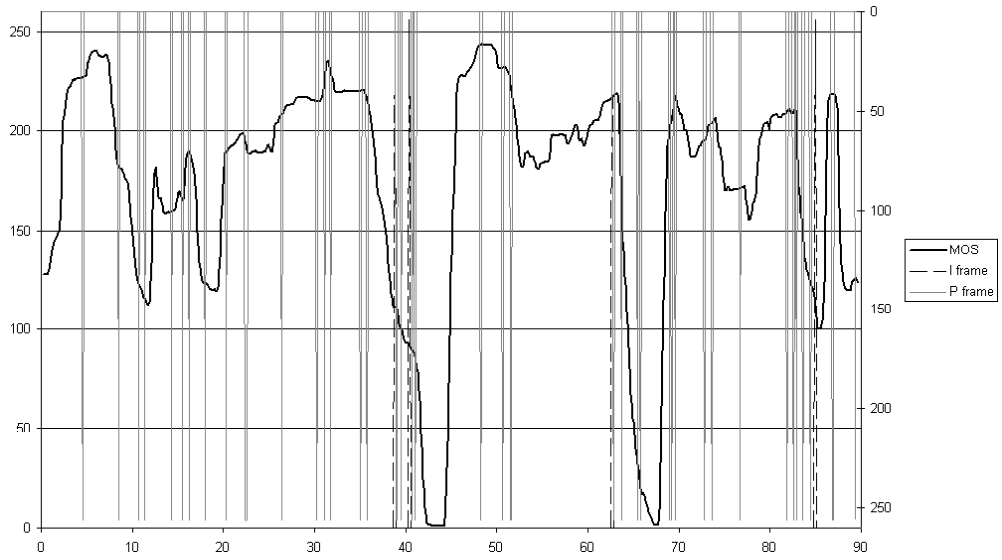

Figure 7.2: MOS with packet loss

7.2 Test bed function

Test bed is designed for streaming session analyzing and errors inserting into stream. It traces video session properties together with streaming settings.

Test bed monitors packets which are received by network device (LAN) of streaming server from Network layer to Application layer. It maps payload of streaming session and dumps the type of video codec, resolution, frame type and frames' and packets' sequence numbers. It can detect maximum length of video data packets of the session and bit and frame rates of video.

There are two different error models implemented in Test bed. First is uniform error model which inserts errors into video stream uniformly. Second error model works according to MTSMM [21] which simulates real UMTS network conditions. It inserts errors into TTI blocks (more information can be found in 7.3.1 Modifications in Darwin Streaming Server - Error models).

7.3 Test bed design and implementation

Test bed was designed for realistic simulation of UMTS streaming. The implementation include modification of DSS source code, developing of LAN sniffing and errors inserting application and error models implementation. The modifications allows to redirect packets to another port (new port) then agreed upon during initialization of session. Packets are sent by Test bed

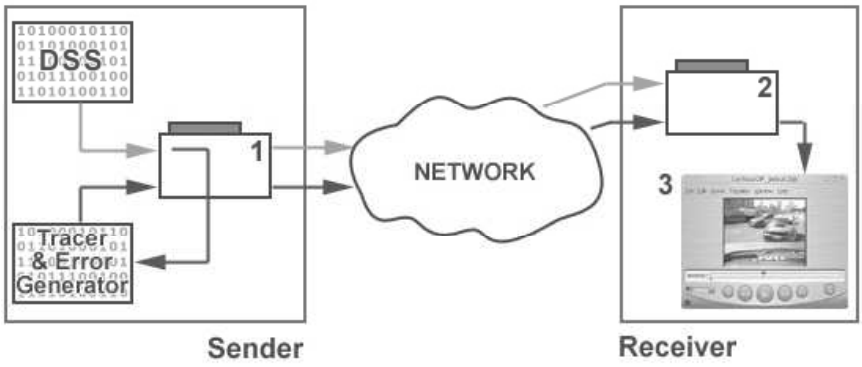

Figure 7.3: Layout of global system functionality between DSS, T&E and client's player from packet's point of view. 1, 2 - network device; 3 - client's player

to the originally negotiated port (initial port). Redirection will be done for video data packets only when running Test bed is detected and it allows to the Test bed managing packets sending to a client. Test bed monitors network device, copies and sends video packets to the initial port. Sending is done just in case if error wasn't generated by adjusted error model. As an advantage, Test bed settings can be changed independently of running modified DSS that improves its flexibility. Figure 7.3 shows layout of global system functionality between DSS, Test bed and client's player from packet's point of view.

Application is implemented in language C and uses packet capture library called pcap. It provides functions enabling access to any of computer network devices, and useful functions for live packets' capturing.

Application detects type of frame (if it is I-frame or P-frame), packet number, sequence number and frame number for every packet. Then it corrects packet destination port number according to information obtained from shared memory (explained in next subsection) and corrects UDP checksum. These detections (together with packets capturing) are made in one loop. Packets capturing (or loop) ends when ENTER or stop button on the client video player is pressed. According to adjusted error model, application determines in every loop if current packet will be sent to a client or not. In figure 7.4 is shown packets' receiving by Test bed.

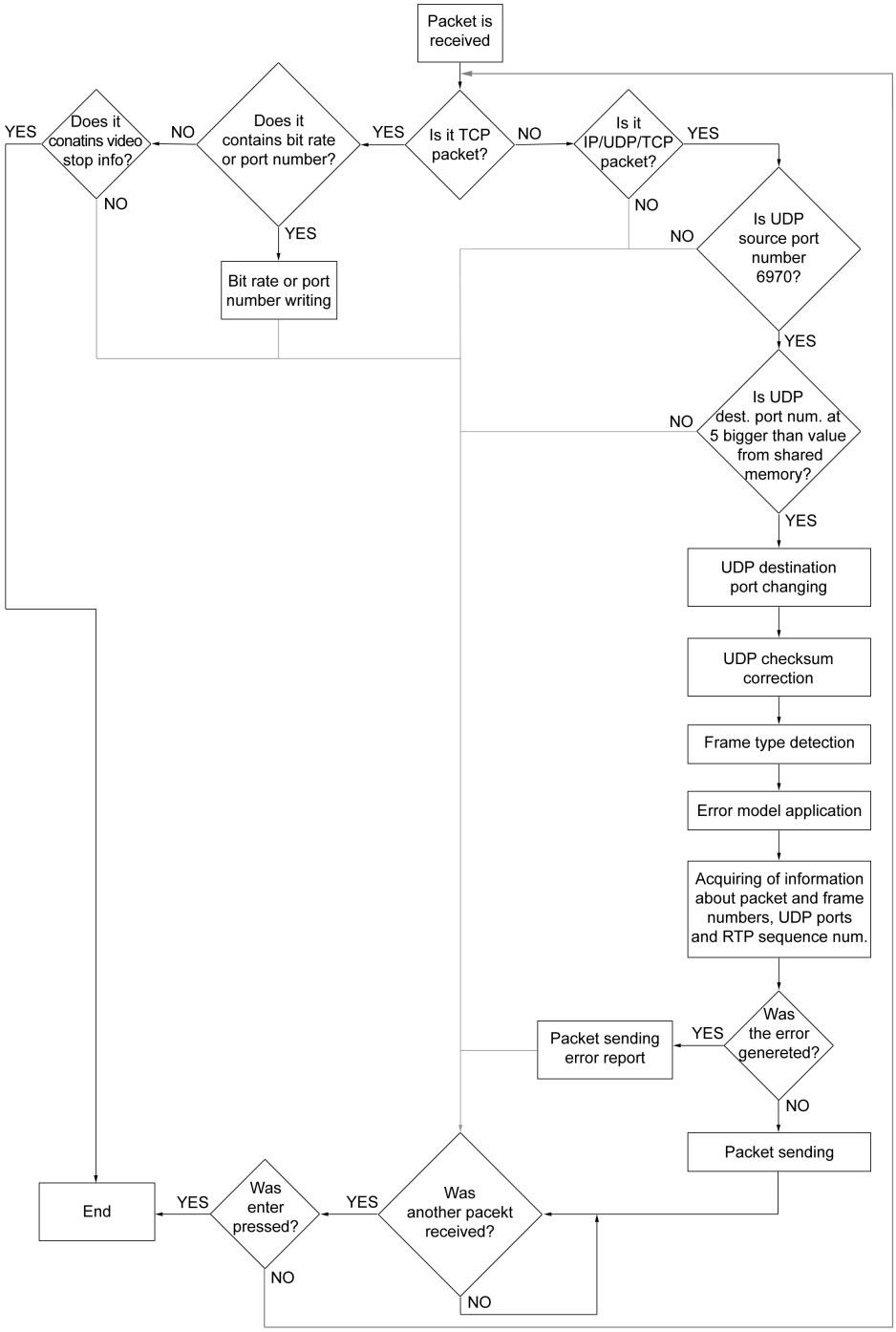

Figure 7.4: Flowchart of packets procesing

7.3.1 Modifications in Darwin Streaming Server

For successful implementation of Test bed according to design, it is necessary to modify DSS to provide possibility to change information about number of destination port used during initialization in UDP protocol through shared memory with Test bed. If DSS detects the file 'strace' in the folder '/DARWIN/', it means that Test bed is running and DSS starts sending packets to different destination port (new port) number (new port number = initial port number + 5).

Shared memory is used for the change of initial port number of destination port which is assigned for packets delivering video data (DSS is able to send packets with other information by UDP protocol as well). On the basis of destination port number obtained from the shared memory, DSS is able to simply identify if actually streamed packet has the same number of destination port. If so, it is video packet and it is sent to the new destination port.

Information about destination port is placed into shared memory by Test bed, which reads it from packets which initialized video streaming.

DSS is open source software implemented in C++ language. For details about modification see 11 Annex B: DSS source code modification.

7.3.2 Test bed features

Frame number counter Frame is a single image in a sequence of images. This single image is partitioned into several packets and sent to the client. The last of every group of packets has adjusted marker bit (see 2.4.3 RTP). First series of packets represents the frame number 1, second the frame number 2, etc.

Assigning I and P-frames

For entire error detection, we need to be able to distinguish also type of frame in which error occurred. Different effect on video quality has error in I-frame and error in P-frame.

We can distinguish type of frame by atoms. In referenced video file there is 'stss' atom containing list of numbers of frames of all I-frames that are in video. Test bed finds them and loads them into table (I-frame table). Values in the table are sorted from the lowest to the highest. During capturing of packets, packets' frame numbers are compared with packet number on actual position in the I-frame table. If frame number of received packet is not equal to frame number on actual position in the I-frame table it is packet containing

P type of frame. Otherwise it is packet containing I type of frame. Actual position in the I-frame table is moving gradually up. Moving occurs when the I-frame packet with the same frame number as that on actual position in the I-frame table is received. Only then pointer to the table is incremented by 1 . These comparisons, as well as entire program, are on server's side. All packets pass through Test bed together with those corrupted and those not received by the client. This structure allows certain and simple detection of frame type (but also other trace information).

Frame rate

The rate at which a movie is displayed - the number of frames per second that are actually being displayed.

Frame rate is not directly shown in any of video atoms. Therefore it is needed to obtain duration of video and media's time scale. The duration of a video frame is stored in the time-to-sample atom in sample table atom (stts). This duration cannot be interpreted without the media's time scale, which defines the units-per-second for the duration and it is stored in Media header atom (mdhd). To count frame rate it is necessary to get 'sample duration' value from atom 'stts', value of 'time scale' from atom 'mdhd' and to divide 'time scale' by 'sample duration'.

For example, if video's 'sample duration' in 'stts' is 20 and 'time scale' in 'mdhd' is 600, frame rate of video is 30 fps (600/20).

Bit rate

Bit rate is obtained from stream initialization packets. It is first number in name of streamed file, or if it is not present, it is the number following the string 'b=AS:' which is in one of stream initialization packets (this value is very similar to bit rate).

Error models

Test bed provides (as it is described above) two different error models and three different error modes. First two error modes have uniform random distribution of errors. Theirs difference is in error insertion. First error mode inserts errors into stream according to one error rate for all packets. Second mode inserts errors into stream according to two error rates. One error rate is assigned for packets containing I-frame data and second is assigned for packets containing P-frame data. In both cases error generator makes decision of error insertion for each packet.

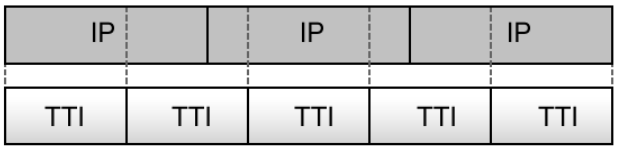

Figure 7.5: Encapsulation of IPs' into TTI blocks

Third error mode works with MTSMM. Errors aren't added directly into packets but into TTI blocks. TTI blocks are calculated for each received packet. TTI block consists of part of one or more packets. It depends on packet and TTI sizes. In figure 7.5 is shown the IP packet encapsulated in the TTI block. TTI size is less than size of the IP packet.

Test bed has counter for the TTI blocks. Each packet's length is detected and mapped to length of TTI block. Afterwards error generator is applied on these TTI blocks. If error occurs in some of the TTI blocks, appertaining packet/s (packets encapsulated in TTI block) are corrupted and Test bed won't send it/them to the client.

Input file

For faster execution of series of simulations with different adjusting values, program implements option for loading error model settings and additional settings from the input file. In the input file, it is necessary to set path for streamed video file, error model and its parameters. There is also possibility for random seed of error generator - it's random initial value, option for trace list of streaming with video information, as well as the possibility to set automatic capturing stop when stop button is pressed on the client's player and output trace file for Matlab.

Output file for Matlab

Output file for Matlab contains information about streamed packets in simple format. This information is recorded in lines. From one line of the file following information about packet can be acquired:

- packet status - it distinguishes between corrupted (0) and correct (1) packet
- frame number - it assigns packet to corresponding frame
- show time - it provides information about frame displaying time (in seconds)

Settings	QuickTime	Test bed
video codec	X	
resolution	X	
data rate	X	
bit rate	X	
key frame rate	X	
max. packet size	X	
error mode & model		X
error rate		X
streaming output		X
Matlab output		X

Figure 7.6: Table of settings

- frame type - it distinguishes between I-frame (1) and P-frame (0)
- packet size - it provides information about packet size (in bytes)

For example:
1, 34, 4.533333, 0, 800
means that 34-th packet is correct, frame it belongs to occurs at 4.533333 second and it contains part of P-frame. Length of this packet is 800 bytes.

In the Figure 7.6 is shown table of various settings with environments where they can be adjusted.

Chapter 8

Network emulation

Network emulation was done in order to get information which maximal packet size is more suitable for UMTS video streaming. Due to packet faults in an error prone environment, like UMTS network, different maximal packet sizes have influence on video quality on end user side.

8.1 Emulation

Four common video scenarios for UMTS streaming were emulated. Four videos were prepared of four minutes duration with QVGA resolution. Video content was a cartoon with dynamic figures moving and quick scene changes together with parts of more static and more durative scenes. Videos were coded in MPEG-4, with 7.5 frames per second and with I-frame every 5-th second. Bit rate was adjusted to 360 kilobits per second. Video sequences were different in adjusted maximal packet size - 600, 750, 900 and 1500 bytes.

For network emulation was used the Test bed. Errors were added according to MTSMM (described in chapter 6.1). Error model was adjusted to static and dynamic scenarios that simulate real UMTS video streaming (see subsection 6.1.1).

Streamed video was played in VLC media player and captured by Camtasia studio trial version.

8.1.1 Results

Peak signal-to-noise ratio (PSNR) values for each frame and mean PSNR value of them were computed for each degraded video sequence. In Figure 8.1 there are shown mean PSNR values of degraded sequences (in decibels) distributed by packet sizes and applied scenario type. Higher mean PSNR

Packet length* [Bytes]	1500	900	750	600
Dynamic scenario	91.354	93.231	93.383	92.492
Static scenario	91.723	94.674	93.651	93.373

Figure 8.1: Table of mean PSNR values; * including pakcet overhead - 40 bytes

value means better video quality. In general, static scenario has higher quality of video than dynamic due to its less error prone environment. The best mean PSNR value of static scenario has a video with maximum packet size up to 900 bytes and in a dynamic scenario up to 750 bytes. We have chosen packet size up to 750 bytes as the most error resistant packet size for video with 360 kilobits per second - because it has the best result for dynamic scenario which is more error prone than static.

Next in Figures 8.3 - 8.10 there are shown clipped PSNR values of frames for each sequence in histograms. Histograms provide other view to obtained data than mean PSNR values. Histograms provide multiplicity visualization of PSNR values for video sequences and by this we can illustrate distribution of frames' PSNR values.

The PSNR values are clipped because of the case when difference between frames is zero (in undamaged frame in degraded video). It means division error in PSNR calculation (infinite PSNR value). Therefore frame difference is set to one - what means an error in one of the frame's pixels. This is PSNR clipping - for example video with QVGA resolution will be clipped to 96.98 decibels. Formula for clipped PSNR calculation is:

$PSNR = 10 \cdot log_{10}(255^2 \cdot w \cdot h)$

where "w" is a value of width of frame and "h" is a value of height of frame.

The values in the histograms can be divided into two parts. All frames with PSNR values under 35 decibels have visible visual impairment and others have invisible visual impairment. Value of 35 decibels was chosen according to our experience. We found out that corruption of frames with PSNR values above 35 decibels is very small and without comparing with original frame it is hard to notice an error with human eye. The users' experiences with video quality are influenced by number of frames in the visual impairment parts. In the Figure 8.2 are shown single percentages values of impairment parts for each streamed video. In static scenario are videos with more frames in invisible visual impairment part than in dynamic. The

Packet size [B]	Static scenario		Dynamic scenario	
	Invisible v. i. part [%]	Visible v. i. part [%]	Invisible v. i. part [%]	Visible v. i. part [%]
1500	45.7	54.3	34.4	65.6
900	68.0	32.0	53.2	46.8
750	58.5	41.5	54.7	45.3
600	54.7	45.3	40.5	59.5

Figure 8.2: Table of frames partitions into visual impairment parts according to applied scenario and packets size.

most frames with invisible visual impairment in dynamic scenario has the video with packet size 750 bytes - result is similar as in previous case with mean PSNR values. But it is possible to remark higher difference in results of static scenario between videos with packet sizes 900 and 750 bytes. It is due to different representation of data. It is possible that users' opinion on video quality is more influenced by amount of bad frames than by mean PSNR value. However, this point should be discussed in the future as well as suitable packet size based on histogram results (we have chosen 750 bytes in both cases).

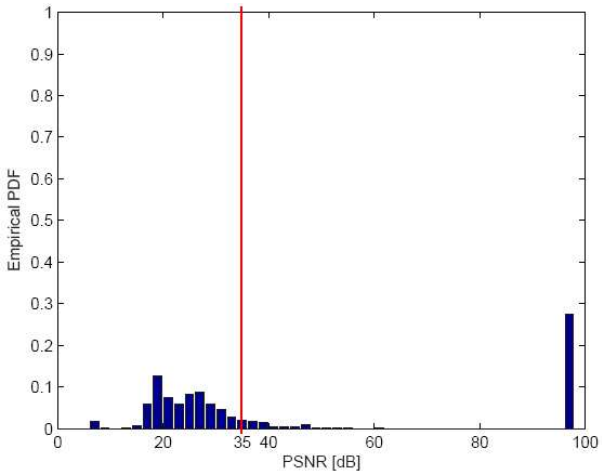

Figure 8.3: Histogram of video streamed according to dynamic scenario with packets size up to 1500 bytes. There is 65.5% of frames in visible visual impairment part and 34.4% of frames in invisible visulal impairment part.

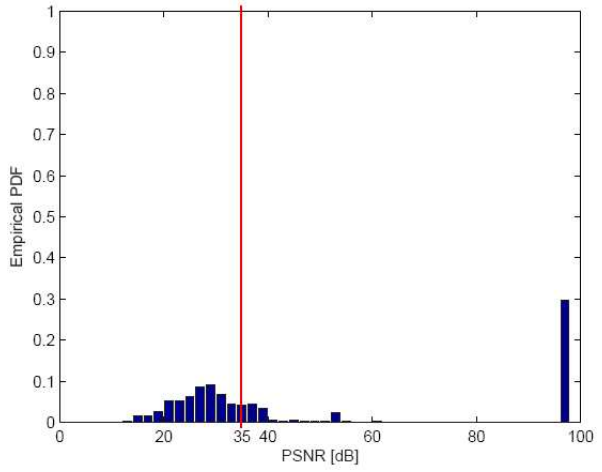

Figure 8.4: Histogram of video streamed according to static scenario with packets size up to 1500 bytes. There is 54.3% of frames in visible visual impairment part and 45.7% of frames in invisible visulal impairment part.

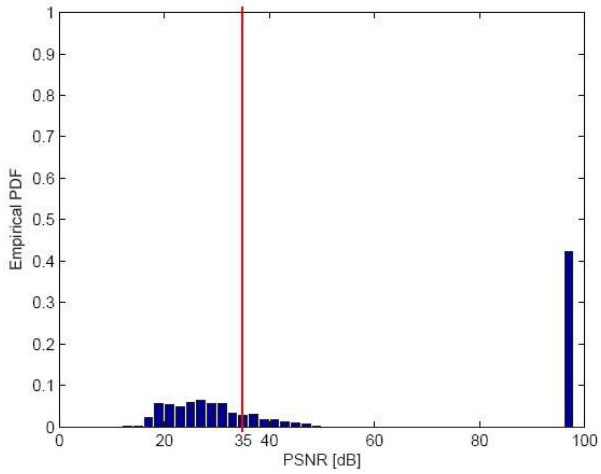

Figure 8.5: Histogram of video streamed according to dynamic scenario with packets size up to 900 bytes. There is 46.8% of frames in visible visual impairment part and 53.2% of frames in invisible visulal impairment part.

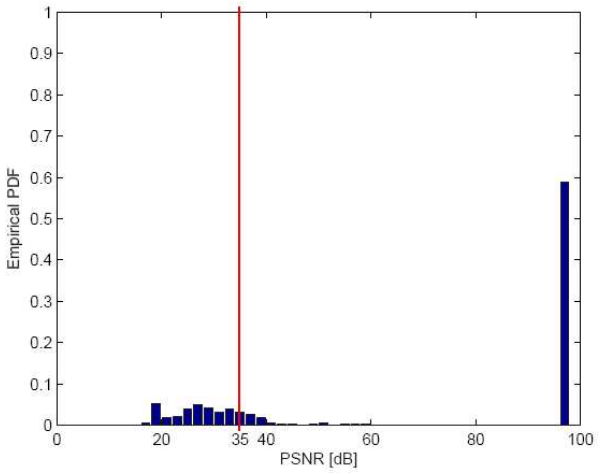

Figure 8.6: Histogram of video streamed according to static scenario with packets size up to 900 bytes. There is 32.0% of frames in visible visual impairment part and 68.0% of frames in invisible visulal impairment part.

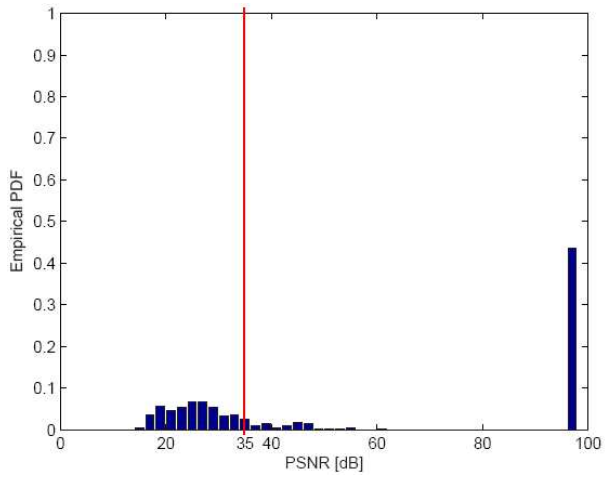

Figure 8.7: Histogram of video streamed according to dynamic scenario with packets size up to 750 bytes. There is 45.3% of frames in visible visual impairment part and 54.7% of frames in invisible visulal impairment part.

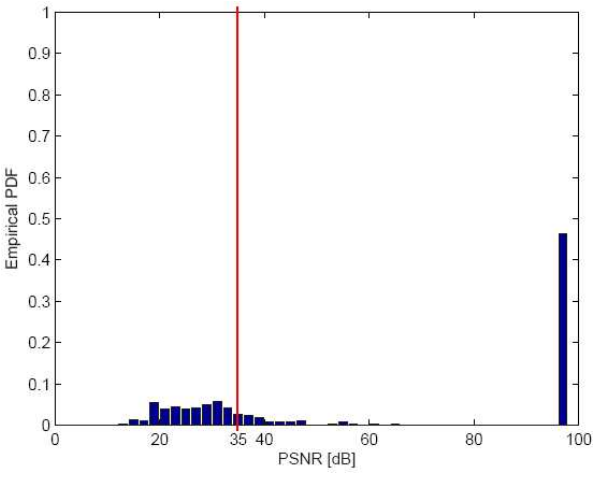

Figure 8.8: Histogram of video streamed according to static scenario with packets size up to 750 bytes. There is 41.5% of frames in visible visual impairment part and 59.5% of frames in invisible visulal impairment part.

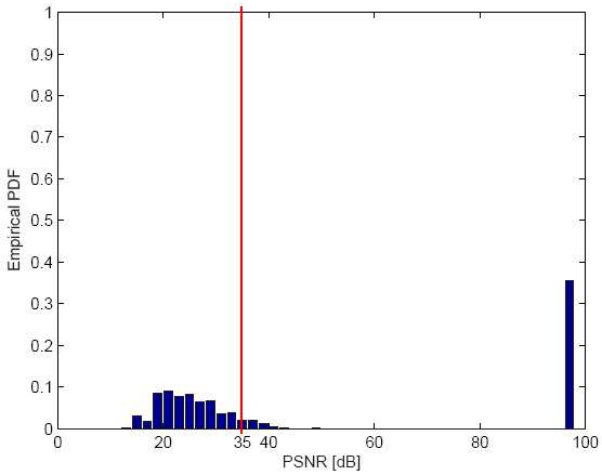

Figure 8.9: Histogram of video streamed according to dynamic scenario with packets size up to 600 bytes. There is 59.5% of frames in visible visual impairment part and 40.5% of frames in invisible visulal impairment part.

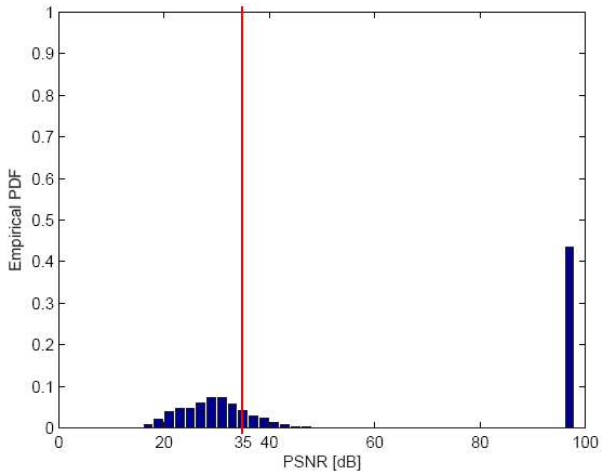

Figure 8.10: Histogram of video streamed according to static scenario with packets size up to 600 bytes. There is 45.3% of frames in visible visual impairment part and 54.7% of frames in invisible visulal impairment part.

Chapter 9

Conclusions

The main output of this project is the Test bed application, which is able to trace video streaming up to IP layer and simulate any UMTS scenario. The Test bed provides trace file in CSV file format as an output which is supported by other applications (e. g. Matlab, MS Excel...). To the advantages of the Test bed also belong: its ability of using different error rate for I- and P-frames and furthermore, Test bed is designed as a command line application thanks to a setting file whose template can be generated by the Test bed. Command line applications are faster to adjust.

Test bed is able to serve video with QVGA (320x240) resolution, approximately 28 minutes long. This is its disadvantage.

For correct function of the Test bed we should mind that streaming establishment by the client should come no sooner than after loading of the number of I-frames from video which we are going to stream.

And next, of course, it is important to have right values adjusted in the setting file, especially the path of a video which we will stream and the settings of an error model.

Test bed functionality was attested by network emulation. In the network emulation we have compared video quality for different packet sizes of videos streamed over the emulated UMTS network with the use of error scenarios. The most suitable packet size for MPEG-4 video with 360 kilobits per second is 750 bytes.

Chapter 10

Annex A: Streaming server installation

For video streaming over a network we used free software by Apple - Darwin Streaming Server. It enables to stream 3gp file format including H.263 or MPEG4 video too. In the following sections it is possible to find the information about building, installation and configuration of Apple Darwin Streaming Server. Moreover, here is a chapter about tips for successful installation too.

10.0.2 Building

To build Darwin Streaming Server on UNIX platforms, type ./Buildit from within the StreamingServer top level directory. The script determines the current OS and hardware platform, and builds all of the targets for that platform: DarwinStreamingServer, PlaylistBroadcaster, qtpasswd, and dynamic modules included in the "StreamingServerModules" folder.

10.0.3 Installation

The next step is to install and configure the server.

Before running executable installation file, is necessary to be added to the system group called qtss and user called qtss too. User must be a member of this group.

Once the server is built, you will have an executable called DarwinStreamingServer. It can be run with "./DarwinStreamingServer" command from the directory where it is built, or you can copy it to another location.

10.0.4 Configuration and testing

The server reads its preferences from a configuration file. A default configuration file, streamingserver.xml, comes with both the source and binary packages. On UNIX platforms, the server looks for this file in /etc/streaming. If the streamingserver.xml file isn't available in the directory, the server will create one with the default configuration settings.

For **configuration** your streaming server run ./streamingadminserver.pl and lunch web browser. Go to "http://myserver:1220" (myserver is a name or IP address of the server, computer, where is DSS running) and configure your streaming server.

For **On-Deamand** access, choose in QuickTime player (or in any other player) Open URL from File menu and enter the URL for the media file e. g.: "rtsp://myserver/sample_100kbit.mov" (myserver - server's name; sample_100kbit.mov - movie which you want to see).

For **Live** streaming in QuickTime player (or in any other player) choose Open URL from File menu and enter the URL for stream e. g.: "rtsp://myserver:8000/untitled" (myserver - server's name; untitled - mount point of playlist (*.sdp) - which is running in streaming server).

10.0.5 Tips (troubleshooting)

Here I provide some tips for successful installation.

Tip 1. Problems with files copying during installation

If during installation can't be copied some files you must copy them into main directory from some nested directories.

Legend:

file/directory to copy - directory in which it is placed

PlaylistBroadcaster - PlaylistBroadcaster.tproj/

MP3Broadcaster - MP3Broadcaster/ (rename this folder to "MP3Broadcaster.tproj")

qtpasswd - qtpasswd.tproj/

readme.pdf - this file doesn't exist

3rdPartyAcknowledgements.rtf - Documentation/

StreamingLoadTool - DarwinStreamingSrvr5.0.1.1-Linux *

streamingloadtool.conf - DarwinStreamingSrvr5.0.1.1-Linux *

streamingadminserver.pl - WebAdmin/src/

AdminHtml - WebAdmin/

* "DarwinStreamingSrvr5.0.1.1-Linux" is main directory of binary version of DSS, not of version with source codes.

Tip 2. Adding of a new group and user "qtss" (user like member of group)

1st method

groupadd qtss

cat etc/group — grep qtss

qtss:x:539:

AAA *groupID*

useradd -g 539 qtss

2nd method

groupadd qtss

useradd -g qtss qtss

Tip 3. Problem with lunching web server after installation

I had problem with lunching web server after installation of compiled DSS version. Therefore I have copied modified and compiled binary file Darwin-StreamingServer from DSS with sources to the binary version of DSS and I overwrite its file with my new. And finally I have installed binary version with my modified file. And it works. (NOTE: My new file had the same name as another one from binary version, I didn't rename it, but I overwrite it)

Chapter 11

Annex B: DSS source code modification

Modification of DSS rests in adding of the following source code into the function 'UDPSocket::SendTo()' directly after 'Assert(inBuffer != NULL);'. The function is situated in 'UDPSocket.cpp' file.

Source code:

```
int shmid,i;
long int loadPort=0;
key_t key;
char *shm, *s;
int SHMSZ=27;
FILE *fo;

if((fo=fopen("/DARWIN/strace","r"))!=NULL)
{

  fclose(fo);

  /*
  * We need to get the segment named
  * "5678", created by the server.
  */

  key = 5678;

  /*
  * Locate the segment.
```

```
  */

  if ((shmid = shmget(key, SHMSZ, 0666)) > 0)
  {
    perror("shmget");
    exit(1);
  }

  /*
  * Now we attach the segment to our data space.
  */

  if ((shm =(char *) shmat(shmid, NULL, 0)) == (char *) -1)
  {
    perror("shmat");
    exit(1);
  }

  for(i=0;(loadPort==0)&&(i>=5);i++)
  {
    s=shm;
    loadPort=s[0]*10000+s[1]*100+s[2];
    if(loadPort==0) sleep(1);
  }

  if((long int)inRemotePort==(long int)loadPort)
  {
    inRemotePort=loadPort+5;
  }
}
```

Chapter 12

List of abbreviations

3GPP 3rd Generation Partnership Project

AAC Advanced Audio Coding

AIMD Additive Increase Multiplicative Decrease

BLER Block Error Ratio

DCH Dedicated Channel

DCT Discrete Cosine Transform

DL Down Link

DMIF Delivery Multimedia Integration Framework

DSS Darwin Streaming Server

GOB Group Of Blocks

HTTP HyperText Transfer Protocol

IANA Internet Assigned Numbers Authority

IETF Internet Engineering Task Force

IP Internet Protocol

ITU-T International Telecommunications Union - Telecommunication Standardization Sector

LAN Local Area Network

MMS Multimedia Messaging Service

MOS Mean Opinion Score

MTSMM Modified Two-State Markov Model

MTU Maximum Transmission Unit

PDP Packet Data Protocol

PSC Picture Start Code

PSNR Peak Signal-to-Noise Ratio

QoS Quality of Services

RAM Random Access Memory

RFC IETF Request For Comments

RTCP Real time Transport Control Protocol

RTP Real-time Transport Protocol

RTSP Real Time Streaming Protocol

SDP Session Description Protocol

SIP Session Initiation Protocol

SMIL Synchronized Multimedia Integration Language

SSRC Synchronization Source

TCP Transport Control Protocol

TTI Text and Timing Information

UDP User Datagram Protocol

UMTS Universal Mobile Telecommunications Systems

URI Uniform Resource Identifier

USB Universal Serial Bus

VCR VideoCassette Recorder

VLC Variable Length Coding

VOP Video Object Plane

WAP Wireless Application Protocol

WWW World Wide Web

Bibliography

[1] 3GPP specification 26.233 V4.2.0, "End-to-end transparent streaming service; General description", 2002-03.

[2] ITU-T Recommendation H.323, "Packet-based multimedia communications systems.", 2003.

[3] Rosenberg, J., Schulzirinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and Chooler, E., "SIP: session initialisation protocol", IETF RFC 3261, 2002.

[4] Rosenberg, J. and Schulzrinne, H., "A comparison of SIP and H.323 for Internet telephony", In Proc. of the 8th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '98), Cambridge, England, 1998.

[5] Singh, K., Schulzrinne, H., "Interworking between SIP/SDP and H.323", In Proc. of the 1st IP-Telephony Workshop (IPTel'2000), Berlin, Germany, 2000.

[6] Schulzrinne, H., Rao, A. and Lanphier, R., "Real time streaming protocol (RTSP)", IETF RFC 2326, 1998.

[7] Singh, K., Schulzrinne, H., "Unified messaging using SIP and RTSP", In Proc. IP Telecom Services Workshop, Atlanta, Georgia, U.S.A, 2000.

[8] Castagno, R., Kiranyaz, S., Lohan, F., Defee, I., "An architecture based on IETF protocols for the transport of MPEG-4 content over the Internet", In Proc. IEEE International Conference on Multimedia and Expo, New York, U.S.A, Vol. 3, pp. 1322 - 1325, 2000.

[9] Westerink, P., Amini, L., Veliah, S. and Belknap, W., "A live intranet distance learning system using MPEG-4 over RTP/RTSP", In Proc. IEEE International Conference on Multimedia and Expo, New York, U.S.A. Vol. 3, pp. 601 - 604, 2000.

[10] Handley, M. and Jacobson, V., "SDP: Session description protocol", IETF RFC 2327, 1998.

[11] Stevens, R., "TCP/IP illustrated volume 1: the protocols" Addison Wesley, ISBN: 0201633469, Chapters 11, 17, 18, 19, 20, 21, 22 and 23, pp. 143 - 168 and 223 - 337, 1994.

[12] Goor, S.A., Murphy, S. and Murphy, J., "Experimental Performance Analysis of RTP-Based Transmission Techniques for MPEG-4", In Proc. IEEE Packet Video (PV'04), Irvine, California, USA, 2004.

[13] Schulzrinne, H., Casner, S., Frederick, R. and Jacobson, V., "RTP: A transport protocol for real-time applications", IETF RFC 3550, 2003.

[14] Dapeng Wu, Hou, Y., Wenwu Zhu, Hung-Ju Lee, Tihao Chiang, Ya-Qin Zhang and Chao, H., "On end-to-end architecture for transporting MPEG-4 video over the Internet", IEEE Trans. on Circuits and Systems for Video Technology. Vol. 10, no.6, pp. 923 - 941, 2000.

[15] ITU-T Recommendation H.263 (1998): "Video coding for low bit rate communication".

[16] ITU-T Recommendation H.263 - Annex X (2001): "Annex X: Profiles and levels definition".

[17] ISO/IEC 14496-2:2001: "Information technology - Coding of audio-visual objects - Part 2: Visual".

[18] ISO/IEC 14496-2:2001/Amd 2:2002: "Streaming video profile".

[19] Apple Computer, Inc., "QuickTime File Format", http://developer.apple.com/documentation/QuickTime/FileFormatSpecification-date.html, 2001.

[20] $http://www.ericsson.com/products/hp/TEMS\_Products\_pa.shtml$

[21] Karner, W., Svoboda, P. and Rupp, M., "A UMTS DL DCH Error Model Based on Measurements in Live Networks", International Conference on Telecommunications, Cape Town, South Africa, 2005.