Marcin Nagy

Using FEC for Rate Adaptation of Multimedia Streams

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Helsinki 12.06.2012

Thesis supervisor:

Prof. Jörg Ott

Thesis instructor:

M.Sc. Varun Ranjit Singh

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Marcin Nagy

Title: Using FEC for Rate Adaptation of Multimedia Streams

Date: 12.06.2012 Language: English Number of pages:9+122

Department of Communications and Networking

Professorship: Networking Technology Code: S-38

Supervisor: Prof. Jörg Ott

Instructor: M.Sc. Varun Ranjit Singh

The available path capacity in the Internet is variable. Changeable amount of cross traffic generated by other network nodes leads to fluctuations of queue size, which causes additional packet latency, and to congestion, resulting in packet drops from intermediate routers. Therefore multimedia senders need to adapt their sending rates in order to provide the best available media quality perceived by the receiver, and avoid any losses caused by congestion.

In this work, we investigate possibility of using Forward Error Correction (FEC) not only for error recovery, but also for rate adaptation. We create two new RTP rate adaptation algorithms, i.e. FEC Based Rate Adaptation (FBRA), and Non-FEC Based Rate Adaptation (N-FBRA) and evaluate them in the Internet environment. The first part of the work is conducted in the ns-2 simulator, and it concerns comparison of our algorithms against TFRC for RTP in 3 scenarios: a) variable link capacity b) constant link capacity with 1 RTP flow competing against many TCP flows, c) constant link capacity with 2 RTP flows competing against many TCP flows. The second part of the work is conducted in the AMuSys platform, which is also created in this work. It concerns evaluation of the FBRA performance in the Dummynet emulator.

Keywords: Internet, Multimedia, Video Communication, RTP, RTCP, FEC, Rate adaptation, Error resillence

I would like to convey my sincere thanks to:

Professor Jörg Ott

**For motivating, giving invaluable pieces of advice,
always having open doors and appreciating my efforts**

M.Sc. Varun Ranjit Singh

**For being source of great solutions, excellent guide,
and all fruitful scientific discussions**

Professor Raimo Kantola

**For smooth transfer to the Aalto University,
help and understanding**

My Parents, My Sister, and My Family

**For support during whole my studies, understanding,
and all the love**

and Kornelia

**For being the greatest person in my life,
always keeping faith in my abilities, pushing me to go beyond them,
and tolerating all my pitfalls**

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

3GPP	3rd Generation Partnership Project
ABU	Average Bandwidth Utilisation
AQM	Active Queue Management
ARQ	Automatic Repeat reQuest
DCCP	Datagram Congestion Control Protocol
ECN	Explicit Congestion Notification
FEC	Forward Error Correction
FTP	File Transfer Protocol
GOP	Group of Pictures
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
ISP	Internet Service Provider
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector
MTU	Maximum Transmission Unit
NACK	Negative Acknowledgement
NAL	Network Adaptation Layer
NAT	Network Address Translator
PSNR	Peak Signal-to-Noise Ratio
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RAN	Radio Access Network
RLC	Radio Link Control
RPS	Reference Picture Selection
RTT	Round-trip Time
RTCP	Real-time Transport Control Protocol
RTP	Real-time Transport Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SSA	Slice Size Adaptation
SSRC	Synchronisation Source
TCP	Transmission Control Protocol
TFRC	TCP Friendly Rate Control
UDP	User Datagram Protocol
UEP	Unequal/Uneven Error Protection
VCL	Video Coding Layer
VoIP	Voice over Internet Protocol
VRC	Video Redundancy Coding
VVoIP	Video and Voice over IP

1 Introduction

The very quick expansion of the Internet in the last 15 years has changed our world significantly, as most of information is shared and consumed in the network. Even very old traditional media (e.g. newspapers, radios) move some of their services to the Internet to provide subscribers with quick access to information. For example, in 2008 the BBC World Service transmission on short waves was stopped in Europe, and moved to the Internet. This simple example shows that media in the Internet have become very popular recently. Recent report of comScore [9] states that in May 2011 there were more than 147 millions of YouTube [10] users only in the USA. Furthermore Sandvine Global Internet Phenomena Report for Fall of 2011 [1] reveals that real-time entertainment traffic in the Internet has exceeded 53% of all traffic in North America (see fig. 1). Another report issued by Cisco [11] predicts that video traffic will continue to increase significantly and should take about 90% of all consumer Internet traffic by 2015.

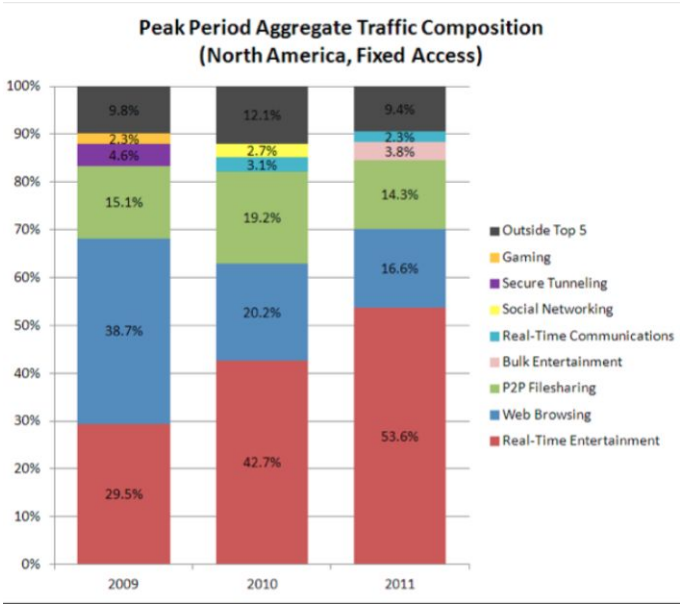


Figure 1: Internet traffic composition in North America in fall 2011. *Picture adopted from* [1]

Emergence of Voice over Internet Protocol (VoIP) applications together with the flat rate pricing pattern made voice calls over the Internet very popular. This trend is also profitable for the operators, as operating and maintenance costs of data networks are lower than similar costs in the PSTN. Many currently available applications (e.g. Skype [12], Gizmo Project [13]) have also support for Video and Voice over IP (VVoIP). Based on all above findings, one can predict grow of popularity of VVoIP services in the Internet.

According to the Nielsen's Law of Internet bandwidth [2] high-end user's connection speed growas about 50% each year or doubles every 21 months (see fig. 2). On the other hand the computing power should increase by 60% every year or double

every 18 months, according to the Moore's Law. It can be concluded that computing power growth occurs quicker than bandwidth grow. Therefore, as the multimedia traffic increases and there are no widely deployed congestion control mechanisms for it (except for HTTP streaming), it seems that such mechanisms must be created to allow so many multimedia applications to compete fairly for limited network resources and guarantee end-users satisfactory experience.

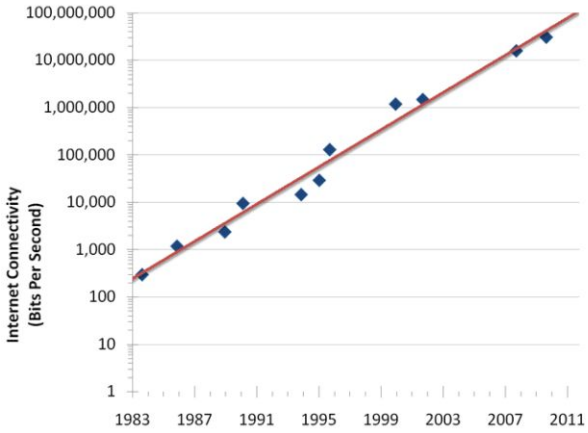


Figure 2: Nielsen's Law of Internet bandwidth. *Picture adopted from* [2]

1.1 Challenges

Video telephony applications must share the bandwidth with other network applications like BitTorrent [14], Web browsers, etc. As network bandwidth is limited, they have to compete for it, which sometimes leads to the congestion situation increasing delays in packet transmission and ultimately their losses.

To combat this problem congestion control mechanisms were introduced. TCP congestion control algorithms are the most popular mechanisms. They are based on the feedback loop in which the receiver sends an Automatic ReQuest (ARQ) containing information about (un-)successful reception of sent packets. This feedback information is then further used by the TCP congestion control algorithms to adapt the sending rate. Although TCP is not optimised for usage with real-time content, it is very often used in HTTP streaming scenarios due to wide deployment of web browsers and HTTP capabilities of NAT and firewall traversals. However, as RTCWeb seems to become a popular solution linking web browsers with RTP protocol, it seems that the is a need to develop congestion control algorithms for use with RTP over UDP.

1.2 Problem Statement

In January 1984 John Nagle predicted problem of congestion collapse in RFC 896 [15]. The actual collapse was not observed until October 1986 when the NFSnet backbone dropped its capacity by three orders of magnitude. Similar problems

started to recur and they were not solved until Van Jacobson's 1988 paper [16] was published.

Concepts presented in [16] led to development of TCP 4.3BSD Tahoe which divided transmission into two different phases: the *slow start* and the *congestion avoidance*. Congestion control in Tahoe is based on altering the congestion window size, which is defined as the total number of packets in transit. Every transmission begins in the slow start phase where the window is incremented for every successfully received acknowledgement. This phase lasts until the window size exceeds *ssthresh* value or a packet loss occurs. This is followed by the congestion avoidance phase in which the congestion window is incremented every round trip time provided that there are no duplicate ACKs received. Furthermore, Tahoe uses *fast retransmit* concept allowing the sender to retransmit potentially lost packet after receiving three duplicate ACKs instead of waiting for the ACK timeout.

In 1990 a new version of TCP called 4.3BSD Reno was introduced. This implementation extended Tahoe's capabilities by adding *fast recovery* functionality. Consequently the sender is allowed to stay in the congestion avoidance phase with its window halved if the fast retransmit procedure succeeds. However, TCP Reno does not resolve the problem of poor performance when burst losses occur. This problem is tackled in the newer version of TCP called New Reno, where the window is maintained full during the fast recovery phase by sending new packets for every received ACK that makes partial progress in the sequence number space. Another important development in TCP congestion control was the introduction of selective acknowledgement options in RFC 2018 [17]. Addition of this functionality makes the sender aware exactly which packets get lost, and therefore it can retransmit just the missing ones, instead of the block of packets beginning with the first missed packet.

The congestion control algorithms in above presented TCP versions are based purely on packet losses. However, there are also TCP algorithms based on other congestion indicators. The oldest one of them is TCP Vegas, which was developed at the University of Arizona in 1994. Unlike previously introduced TCP algorithms, TCP Vegas tries to predict incoming congestion based on increasing round trip time values. Therefore, as very accurate RTT estimation is required TCP timestamps option (RFC 1323) [18] must be used. Finer RTT estimation in comparison to previously mentioned TCP versions, allows also Vegas to calculate retransmission timeout (RTO) more precisely, and therefore quicker respond to encountered packet loss. Other interesting version of TCP taking advantage of precise RTT measurements is Hybla, which is intended to improve transmission performance for long delay networks, as well as for heterogeneous networks containing wireless links.

The very recent research concentrates on creating good TCP congestion control algorithm for high speed networks with high latency between end nodes. Examples belonging to this group are BIC (Binary Increase Congestion control) and CUBIC. Both these algorithms provide unique window growth function. BIC calculates proper window value using the binary search algorithm between the before reduction window size and the after reduction one. CUBIC provides the window growth function which has similar properties to the BIC ones, but it is mainly based on

time elapsed since last window reduction, which makes its performance independent of RTT values, and therefore more friendly to other TCP flows.

All above briefly presented versions of TCP congestion control algorithms are mostly optimised for transmissions where tight delay constraints are not critical. On the other hand, as video conversational communication is delay-sensitive, the end-points must be able to adapt to changing network conditions taking into account also delay constraints. Media packets are sent using the Real-time Transport Protocol (RTP) which is most often run over User Datagram Protocol (UDP) which does not provide any congestion control mechanism. Therefore responsibility for all congestion control is put on applications. Existence of delay constraints makes dependence entirely on packet losses, and RTT estimation insufficient. As a result, applications should be able to process larger set of congestion indicators and try to predict forthcoming congestion in advance to prepare itself for it by reducing its sending rate. This adaptation to network conditions allows making the video communication successful though the video quality is a bit worse. In addition, it also leaves some of the burden from the network, allowing it to quicker combat congestion situation and return to its normal condition.

1.3 Contribution of the Thesis

We investigate possibility of using error recovery mechanism (FEC) [6] also for rate adaptation. Furthermore, we analyse behaviour of available congestion indicators in the Internet multimedia environment. Finally, we link FEC with these indicators to create a new rate adaptation algorithm and test its performance in various scenarios.

1.4 Scope and Goals

The Real-time transport Protocol (RTP) is used to transport real-time content in the Internet, whereas its associated RTP Control Protocol transfers control information between the sender and the receiver. This control information contains important data about reception quality and network conditions. Forward Error Correction (FEC) can be used together with RTP to transfer redundant media content to the receiver which can be used to recover missing data. The thesis objective is to combine FEC together with RTP and existing RTCP extensions standardised by the Internet Engineering Task Force (IETF), and propose possible new rate adaptation algorithm for video communication. Application of this algorithm should not be limited just to video transmission, but it should be used for all types requiring real-time data delivery.

We want to apply possibly designed rate adaptation algorithm in different scenarios inside the *ns-2* simulator [19], where we evaluate performance. Furthermore, we intend to reproduce identical scenarios in the real-world, using existing open-source libraries and network emulators.

1.5 Structure

This is organised in the following way. Chapter 2 gives general overview of the video communication in the Internet environment. In chapter 3 the Internet multimedia traffic transport protocols are described. Chapter 4 describes error resilience mechanisms which are available in the network currently. Chapter 5 introduces concept of the rate adaptation. It focuses on congestion indicators for multimedia flows, and metrics allowing to compare usability of different rate adaptation solutions. Chapter 6 presents two rate adaptation algorithms (FEC Based Rate Adaptation algorithm and Non-FEC Based Rate Adaptation algorithm) which were created specially for the purpose of this thesis. Chapter 7 presents evaluation of designed algorithms. It is divided into two main parts. The first one presents results obtained in ns-2 simulator [19], and the second one presents results from real-world testing environment. The final chapter concludes the thesis and presents observations on possible future continuation work.

2 Multimedia communication and the Internet

This chapter describes multimedia technology in the Internet with specific focus put on the video communication. It also discusses characteristics of the Internet that are important for multimedia communication.

Figure 3 illustrates upstream channel of the video communication system. Any other Internet multimedia system looks similar to the one presented in the picture. For instance, the audio communication system should look the same with the video camera substituted for the microphone and the screen substituted for the loudspeaker. Generally speaking every Internet communication system is divided into three main parts: the sender, the receiver, and the network part. In the video system the sender part comprises the video camera, the encoder, and the packetizer. The video camera is responsible for acquisition of video in the form of raw data (e.g. in RGB or YUV format). Having obtained data, the video camera relays it to the encoder block where raw data are compressed and encoded into video frames. The encoder delivers frames to the packetizer which encodes them into data packets, fragments packets exceeding Maximum Transmission Unit (MTU) size into smaller ones, and encapsulates them in application specific headers. The next sub-part of the system is the sender buffer which schedules packet transmission based on existing network situation. The network part connects communicating users. Packets traverse the network through several routers and versatile physical media (e.g. optical fibre). Packets experience transmission and queuing delays, and may also get corrupted or lost.

The receiver part of the system transfers received packets into raw data that are displayed on the desktop screen. Received packets are delivered to the receiver buffer which reorders them and passes them to the depacketizer. The receiver buffer also drops packets arriving too late to be played. The depacketizer combines fragmented packets, removes application specific headers, and reconstructs compressed frames. They are passed further to the decoder which decompresses them into raw data form that allows for their display on the desktop screen.

Video communication can be divided into two general types: video streaming and conversational video. In the streaming type, packets flow only in one direction (from a server to a client), whereas in the conversational communication packets are transmitted in both directions. The second important difference between both modes of communication concerns packet delay constraints. In conversational video, delivery of a sent packet cannot exceed some fixed value (3GPP recommends 400ms) in order to keep conversation quality on satisfactory level. This constraint does not apply to streaming scenario, in which receiving application may stop playing packets for a moment and restart it after receiving high delayed packets.

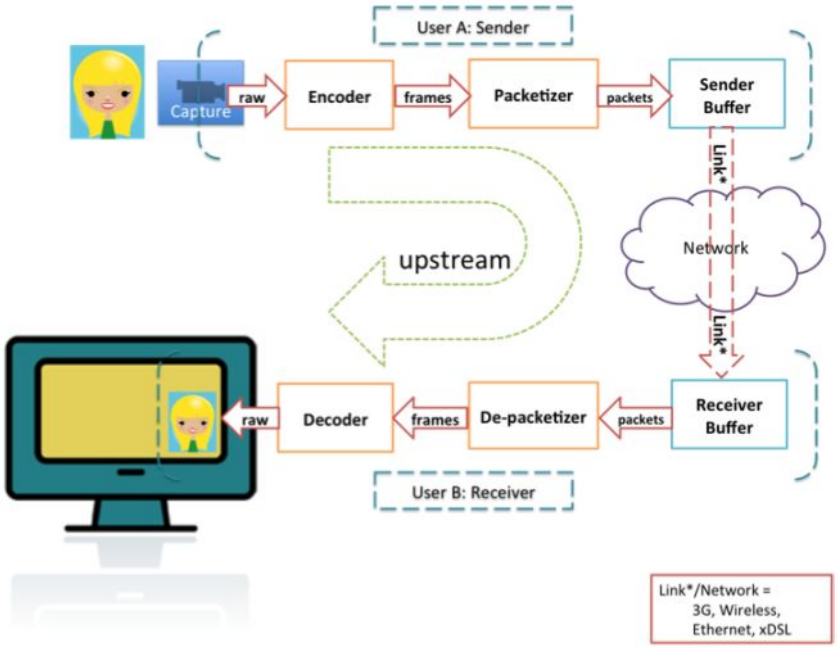


Figure 3: Video communication system. *Picture adopted from* [3]

2.1 Video coding

Video compression is the key element of the communication system, as it allows consuming far less network resources. Video consists of sequences of static pictures (frames) which are displayed on the screen. As neighbouring frames do not differ much from each other, it is pointless to send whole information about every frame. As a result, the encoder divides frames into three different types to utilise network resources better:

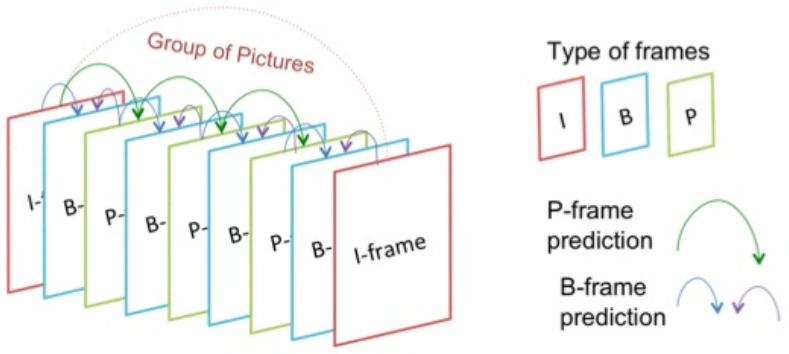
- Intra-coded picture (I-frame)
- Predicted picture (P-frame)
- Bi-predictive picture (B-frame)

An I-frame is a static picture which is compressed spatially. It contains information about whole picture, and can be decoded independently without any other frame. Because of this, its size is significantly larger than size of other frames. Decoding of P- and B-frames depend on other past or future frames. This feature allows conveying only difference between dependent frames, thus saving a lot of network resources.

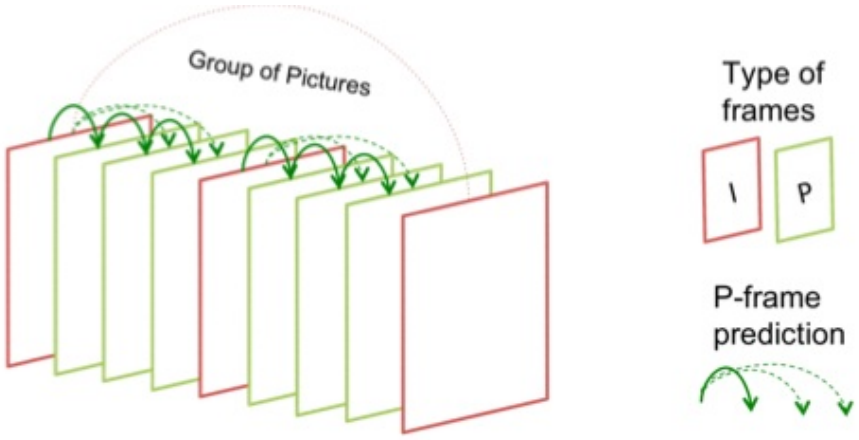
A P-frame holds information only about changes from previous I- and P-frames. For instance, if car motion on a static background is to be encoded, it is sufficient to encode one static I-frame, and a series of P-frames consisting just information with car's movement. A B-frame encodes differences between previous and succeeding I- and P-frames, thus allowing for even higher compression.

As conversational video has very strict requirements concerning packet delivery delay, it is very important to encode and decode data at very high speed. As a result, B-frames, which depend on future frames causing additional coding delay, are not used in this type of video communication. On the other hand when the video is just streamed delay tolerance is much higher and playout delay is also higher, the receiver has enough time to wait for arrival of the B-frame, perform proper packet reordering and display the picture on the screen.

Neighbouring frames form Group of Pictures (GOP). Examples of GOPs are presented in the figure 4. Figure 4a presents GOP for video streaming, whereas 4b shows example of conversational video GOP. In the former picture, the B-frames appear inside GOP, whereas in the latter one, only the I- and B-frames are present.



(a) GOP for streaming

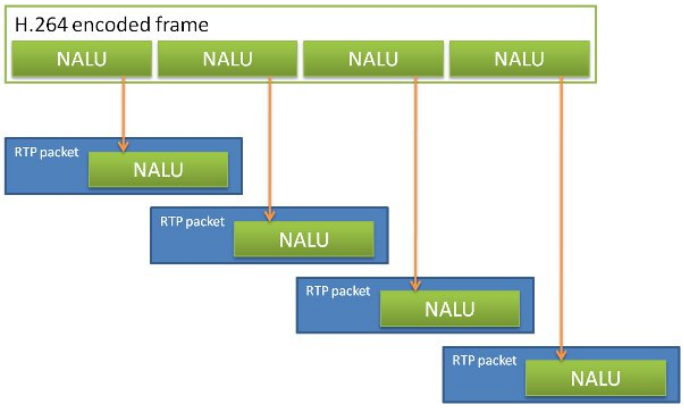


(b) GOP for conversational video

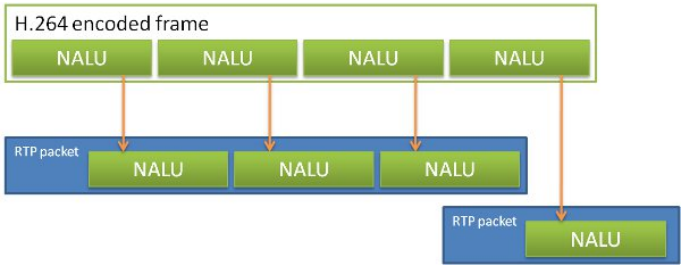
Figure 4: Group of Pictures arrangement (a) for streaming scenario (b) for conversational scenario. *Picture adopted from* [3]

One of the most advanced video coding standard is H.264 [20]. It has been widely adopted as an audio-video coding format by many telecommunications associations like ITU, or 3GPP. H.264 standard divides coding process into two separate layers: the video coding layer (VCL) and the network adaptation layer (NAL). The VCL

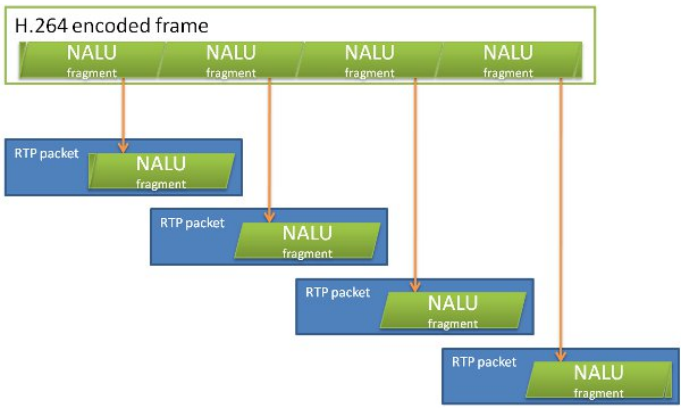
compress video data and also adds redundant information which improves error resilience of the compressed stream. Selected error resilience mechanisms are discussed later. As the VCL is designed to be independent from the underlying network technology, the NAL is responsible for adaptation of generated stream to various network environments. This layer encapsulates stream into Network Adaptation Layer Units (NALUs), which are transported inside RTP payload [21]. Details of H.264 RTP payload format is defined in [22], and presented in the figure 5.



(a) One NALU per packet



(b) Multiple NALUs per packet



(c) Fragmented NALU

Figure 5: Encapsulation of H.264 stream into RTP payload. *Picture adopted from blog.radvision.com/voipsurvivor/*

2.2 Network traffic types

Network traffic can be divided into two fundamental types: *elastic traffic* and *inelastic traffic*. **Elastic traffic** requires absolute transfer reliability, but its completion is not limited by time constraints. It is the traditional traffic in the Internet, for which the network was designed. Typically elastic traffic is used together with the Transmission Control Protocol (TCP), which has its own built-in congestion control mechanisms allowing the sender and the receiver to use highest possible end-to-end capacity between them. Examples of application protocols generating elastic traffic include Hypertext Transfer Protocol (HTTP), Simple Network Management Protocol (SNMP), Telnet, Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP).

On the other hand, **inelastic traffic** does not tolerate high delay, as it has real-time constraints. However, to achieve expected Quality of Service (QoS) level, absolute reliability is not required. Most often inelastic traffic is used together with the User Datagram Protocol (UDP), which does not have any congestion control mechanisms. As a result, capacity usage of UDP-based applications depend on application's data generation rate. However, as such applications may be unaware of availability of network resources, they may require preferential treatment. Examples of such applications include: stock-ticker quotes, live TV, on-line gaming, video conferencing, etc. RTP protocol, which details are covered in the next section, is very often used as the application protocol by this traffic.

2.3 Internet multimedia environment characteristics

Generally there are three main characteristics of the multimedia environment, which are illustrated in the figure 6. These are: *path capacity*, *network latency* and *losses*.

Path capacity characteristic depends on environment conditions and can be static or time-varying. In the wired/fixed Internet capacity is most often constant, and can be:

- used exclusively by one flow
- shared by many flows on one machine
- shared by many flows on many machines by means of Ethernet, WLAN, etc.

When the channel is used by just one flow, its entire capacity can be utilised. However, when multiple flows are present, they must compete for the channel and be fair to other ones, so as to optimise network resources utilisation [23]. In such case congestion may result from unfair capacity utilisation by some applications, or from limited network bandwidth in a bottleneck link.

In mobile and wireless network channel capacity changes dynamically and unpredictably. Capacity fluctuations are caused by user mobility and variable physical conditions over the air interface.

Network latency is the period of time needed by a packet to go from the sender to the receiver. It can be measured also as the round-trip (sum of upstream and downstream latency). However, round-trip network latency measurement may be affected by network's asymmetry which might be caused by variations of queuing, processing and physical propagation delay experienced on both directions of the network.

Networks can be divided into **high latency networks** and **low latency networks**. In former ones, the latency is close to acceptable real-time constraints and leaves very small room for queuing and processing delays. In latter ones, there is much more room for delays caused by queuing and processing.

Losses in the network are caused by routers dropping arriving packets when their queues are full, and by bit-error losses. Losses resulting from intermediate packet router drop are called congestion losses, and are prevailing source of losses in the Internet. Elastic traffic flows using TCP are able to respond properly to such losses. On the other hand, bit-error losses are caused by interference and fading in the wireless environment, or by packet corruption in the wired one. It is very difficult to distinguish type of loss, as most of them in the Internet are congestion losses, TCP response to bit-error losses is very poor, as TCP regards every loss as the congestion loss. Inelastic traffic may additionally suffer from packets arriving too late, and thus becoming useless.

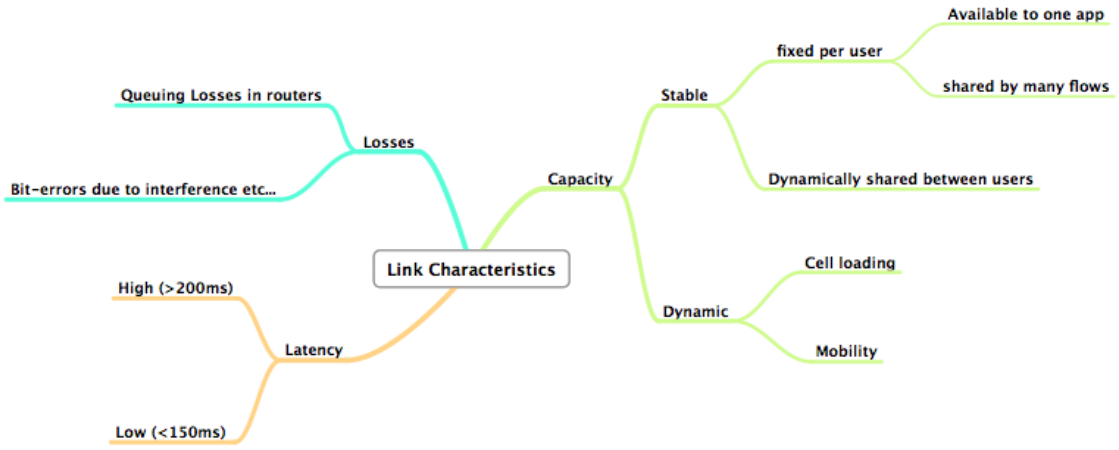


Figure 6: Link characteristics in multimedia environment. *Picture adopted from* [3].

Sections below briefly discuss features of different multimedia environment in terms of these characteristics.

2.3.1 Internet environment

The Internet (see fig.7a) connects globally millions of computers worldwide. Core of the network is built by routers responsible for forwarding packets between end stations. Routers use routing algorithms to choose the most suitable route for the packet. However, as network load varies, and routers may apply traffic engineering policies (e.g. load balancing resulting in forwarding packets using multiple paths), two consecutive packets may experience different end-to-end delay, which accounts for packetisation and serialisation delays. Most of the Internet traffic is transported over optical fibres, which have very low bit-error rates. As a result, almost all packet losses in the Internet are congestion losses. Link capacity is almost constant for the short period of time. Capacity may change in the longer period of time, as Internet Service Providers (ISPs) may differentiate amount of bandwidth available for the user based on his/her subscription.

2.3.2 Mobile environment

The mobile environment (see fig.7b) comprises Radio Access Network (RAN) and the core network, which is very similar in its features to the Internet. In RAN, traffic is controlled on the link layer by Radio Link Control (RLC), which may work in acknowledged mode, unacknowledged one, and transparent one. As multimedia applications send inelastic traffic and link layer delays should be minimised, *unacknowledged mode* is mostly used by them. However, usage of this mode forces applications to implement their own rate adaptation and error resilience mechanisms. The RLC also controls payload size that can be carried over the channel in a particular time. This size varies, as fading, interference, hand-overs appear because of user mobility. Network latencies experienced by packets are potentially higher than in the Internet, because of unpredictability of radio channel conditions, and additional queuing at the base station, and varies from 30-80ms to a few 100ms. Furthermore, the radio interface is much more sensitive to bit-error losses than the wired interface. In this environment, bit-error losses are much more frequent, and it is extremely difficult to distinguish between bit-error losses and congestion ones.

It is also possible to have the heterogeneous environment in which Internet and mobile links are simultaneously present. Such environment characteristics are very similar to these of the mobile environment.

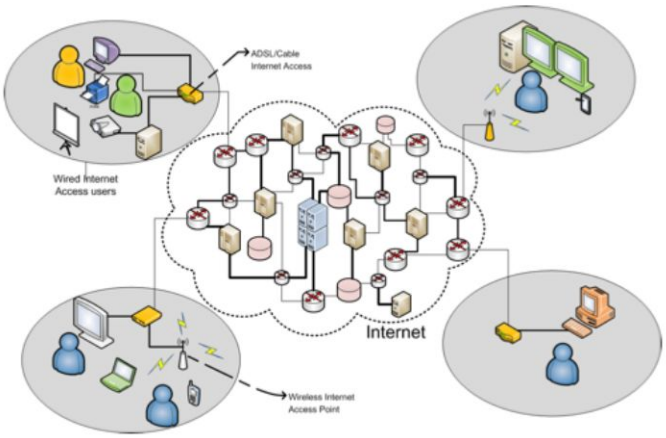
2.4 Summary

The typical video communication system consists of the sender device, the receiver device, and the network between them. The sender side comprises the video camera, the encoder, the packetizer, and the sender buffer. The receiver side is built of the receiver buffer, the de-packetizer, the decoder, and the screen. Video stream is encoded before sending by dividing its raw data into different frames (I-,

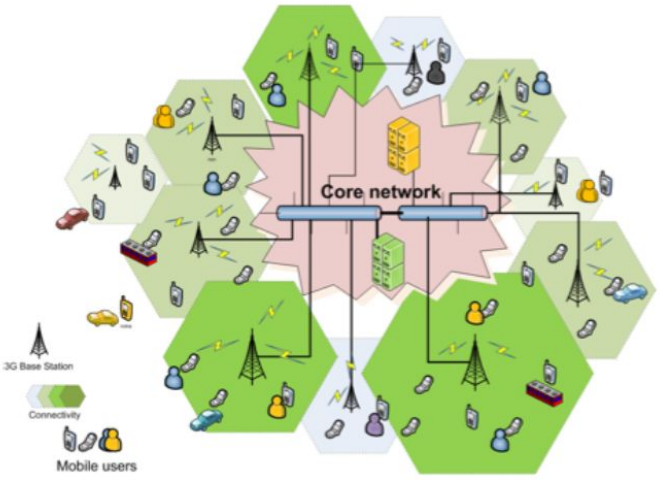
P-, and B-frame). Delay constraints makes usage of B-frames for conversational video impossible.

Traffic in the Internet can be divided into two main groups. Elastic traffic is not time critical, but it demands absolute reliability. On the other hand, inelastic traffic can tolerate small losses, but it must be delivered within required time boundaries.

Path capacity in the wired Internet is rather stable. However, user mobility and various physical conditions over the radio interface cause capacity fluctuations in the wireless networks. Losses in the wired networks result almost always from congestion. In the wireless ones, source of losses is harder to determine, as it may be caused not only by congestion, but also by errors coming from interference and fading effects.



(a) Internet environment



(b) Mobile environment

Figure 7: Multimedia environments. *Picture adopted from* [3]

3 Multimedia transport protocols

Multimedia traffic can be transported by HTTP (so-called **HTTP streaming**), or by RTP. HTTP streaming works over TCP and this makes it sensitive to following potential issues:

- TCP guarantees in order packet delivery, which may cause head of line blocking
- TCP is reliable, but does not take into accounts delays
- TCP has its own congestion control mechanism, which may be unsuitable for codec demands

However, due to presence of web browsers in almost all types of devices, and HTTP ability to traverse Network Address Translators (NATs) and firewalls, HTTP streaming is the most widely used multimedia transport protocol. It is manly used in streaming applications, however, as Skype shows it may also be used for conversational ones.

Generally the conversational communication can be successfully performed on top of TCP if the network conditions are good enough [24]. However, as TCP has its own built-in congestion control algorithm, and there is nothing to create specifically for the multimedia congestion control purpose in HTTP streaming, this work focuses on rate adaptation algorithms for multimedia traffic transported over RTP, and rest of this chapter concentrates on this protocol.

3.1 Real-time Transport Protocol (RTP)

Real-time Transport Protocol (RTP) [4] is the standard application-layer protocol for transmission of data with real-time characteristics (e.g. audio, video) over IP networks. RTP can work with different transport protocols: TCP, UDP and DCCP (Datagram Congestion Control Protocol). Usage of RTP over TCP makes it sensitive to previously mentioned TCP issues. Although DCCP connects advantages of UDP (no reliability and ordering) and TCP (built-in congestion control), experimental results from 2007 [25] show that RTP performs worse over DCCP than UDP, and even TCP. Therefore, RTP is most often used together with UDP.

Every RTP packet header (see fig. 8) carries *sequence number*, *timestamp*, *Synchronisation Source identifier (SSRC)* and *payload type*. *Sequence number* increases by one for every sent packet, and allow the receiver to detect packet losses and reorder them if received in wrong sequence. *Timestamp* represents sampling instant of the first octet in the RTP data packet or video frame. It allows the receiver to know when the packet should be played. If two packets have identical timestamps, but different sequence numbers, they belong to the same frame which was fragmented at the sender side, because of MTU size limitation. *Payload type* uniquely identifies content of the packet, allowing the receiver to understand it, and process it correctly. *SSRC* allows unambiguously distinguish media sources participating in the session. Its value is chosen randomly, in such a way that no two sources in one session can

have identical value. Although it is little possible that two sources choose identical SSRC value, RTP standard defines mechanisms to resolve such conflicts.

Apart from end stations, the RTP standard defines also middle-boxes like mixers and translators. The former ones create a new media stream from one or more incoming streams. This solution is often used in conference bridges, as it reduces bandwidth demands for the receivers. The latter ones operate on the transport level, and may perform e.g. IPv4-IPv6 translation, or multicast-unicast adaptation.

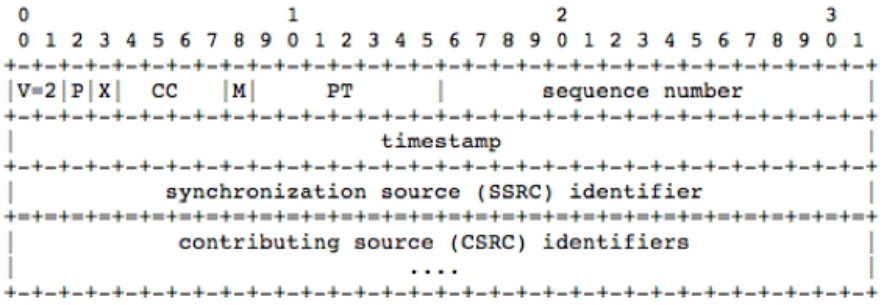


Figure 8: RTP header. *Picture adopted from* [4].

3.2 Real-time Transport Control Protocol (RTCP)

Real-time Transport Control Protocol (RTCP) is the completion of RTP. It provides feedback from the receiver to the sender on quality of received transmission. Figure 9 presents general functionality of these two protocols.

RTCP functionality is based on periodical sending of transmission and reception quality reports (see fig. 10). There are two types of RTCP reports. Sender Reports (SRs) provide information related to media playout, whereas Receiver Reports (RRs) contain long-term reception statistics.

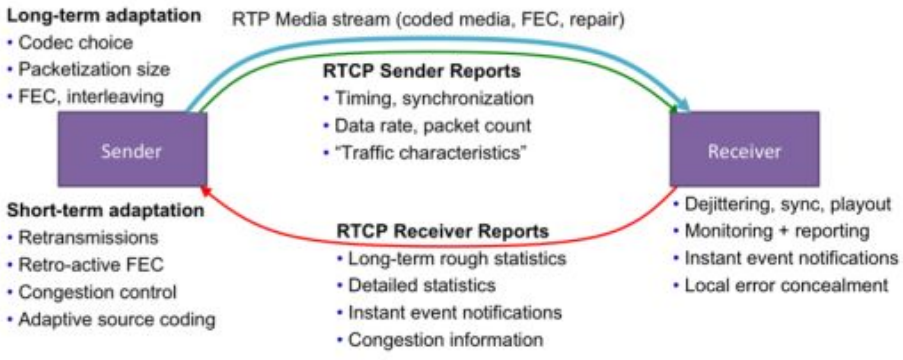


Figure 9: RTP and RTCP overview. *Picture adopted from* [5].

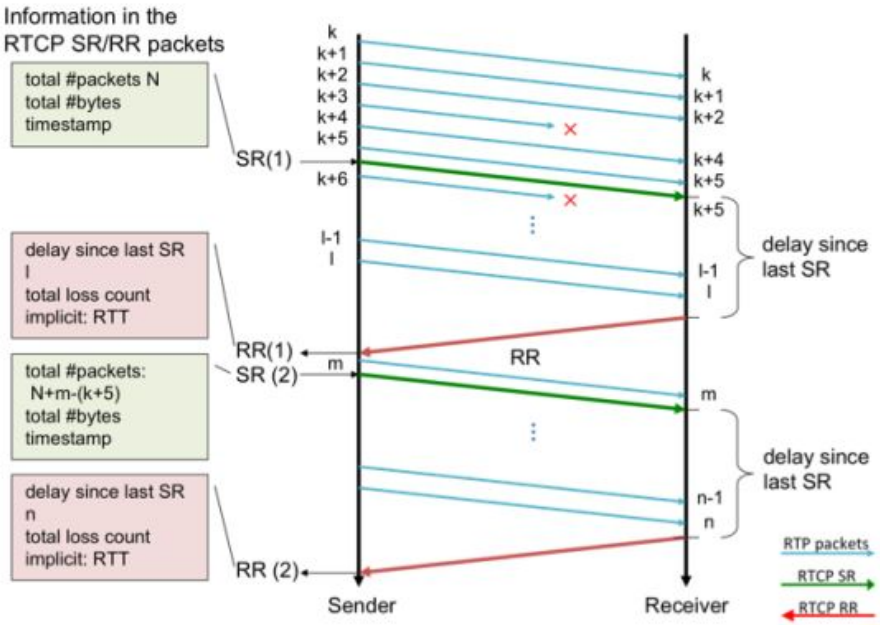


Figure 10: RTP and RTCP flow diagram. *Picture adopted from* [5].

RFC 3550 states that RTCP reports must not exceed 5% of the RTP session bandwidth. Furthermore, as for some high bit-rates session 5% can be a lot, it also states that minimum RTCP report interval should be 5 ± 2.5 seconds. This rule makes congestion control extremely challenging, as the sender only knows long-term statistics, and does not have the most recent data to quickly react to changing situation. Therefore, extended feedback profile (RTP/AVPF) [26] was defined in RFC 4585. This standard allows receivers to send feedbacks more frequent than in the original specification. It releases minimum RTCP report interval rule, but it still maintains 5% bandwidth rule, so as to keep the protocol scalable. Figure 11 presents difference between RTCP report intervals in both standards.

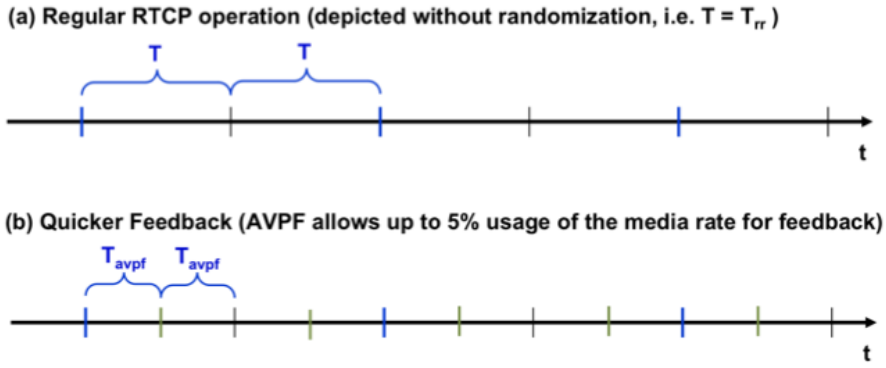


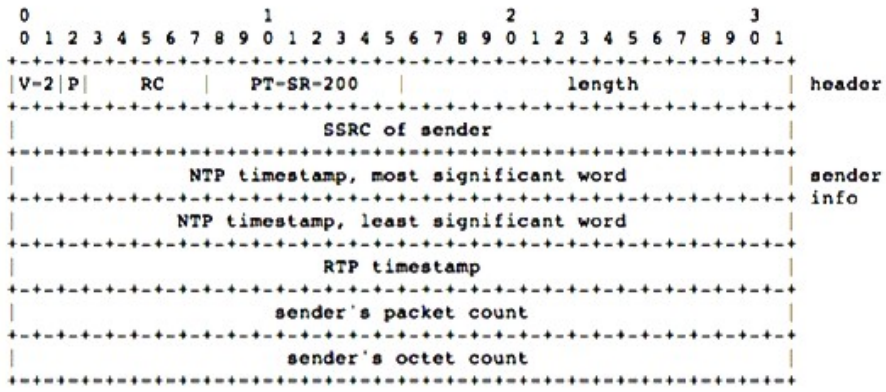
Figure 11: Difference between standard RTCP report interval and extended feedback profile (RTP/AVPF). *Picture adopted from* [5].

RTCP Sender Reports carry information about number of bytes and packets sent. They may be used for sending rate calculations by the receiver. They also contain RTP and NTP timestamps, which allow for synchronisation needed for playback. RTCP Receiver Reports contain following reception statistics: fraction lost informing about losses in the last interval, cumulative lost, highest received sequence number telling the sender if any packet was received in the last interval. They also contain timestamps of last SR reception and delay since its last reception. These timestamps allow sender to calculate Round-trip Time (RTT). RTCP report headers are presented in the figure 12.

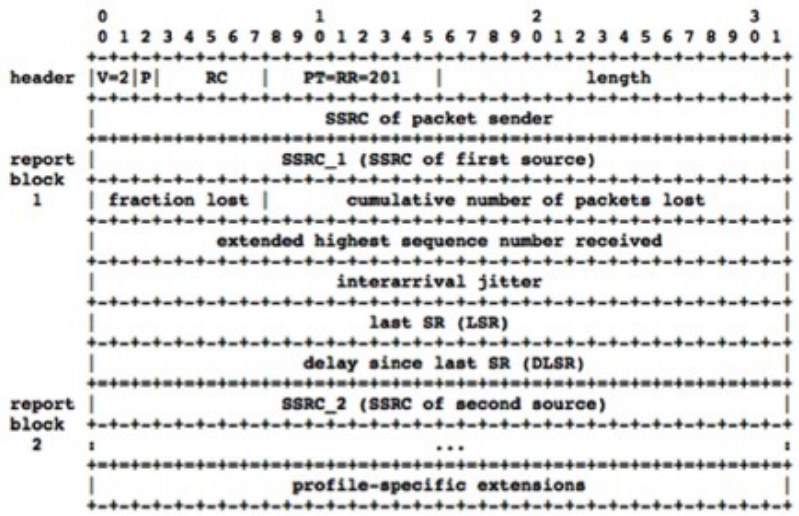
RTP media session may also be supported by RTCP extensions providing more detailed information on reception statistics. Moreover, Forward Error Correction (FEC) may be used together with RTP to provide more protection for RTP packets. RTCP extensions and FEC are described in later chapters.

3.3 Summary

Multimedia traffic is transported in the Internet either by HTTP streaming, or by RTP. HTTP streaming is the less natural method of transport, as it makes the content sensitive to issues resulting from TCP nature. RTCP is associated with RTP and is responsible for conveying information about transmission quality. RTP is run most often over UDP which does not guarantee reliable delivery and have built-in congestion control mechanisms. Thus, to ensure reliable usage of RTP, error resilience and rate adaptation functionality must be added in the application layer. These mechanisms are described thoroughly in the next two chapters.



(a) Sender Report header



(b) Receiver Report header

Figure 12: Basic RTCP report headers. *Picture adopted from* [4].

4 Error resilience mechanisms for conversational video communications

Most of multimedia communications do not have guarantee of reliable delivery. As errors on the receiver side are unavoidable, it is very important to minimise distortions by means of error control mechanisms. Such mechanisms may be able to recover lost or corrupted data, conceal error effects, or mitigate error propagation. Generally error control methods can be divided into three groups: source coding methods, channel coding ones, and end-to-end application/transport ones.

Source coding methods concern all error resilience mechanisms managed by the video codec. They may be applied by error resilient encoding, interactive error control and error concealment. Error resilient encoding concept is based on adding redundant bits to the generated video stream which can be used by the receiver to detect errors, reduce or stop its propagation, and help its concealment. For instance, H.263+ coding standard uses concept of Video Redundancy Coding (VRC). In this standard current P-frame is predicted directly from the previous frame. As a result loss of just one frame breaks whole prediction frame. VRC divides video stream into several substrates, so that current frames are not predicted only from the last frame. Consequently loss of a single frame breaks prediction frame just for one sub-stream, and not for the whole stream, which is much easier to conceal [27]. Interactive error control requires cooperation between the encoder and the decoder. In this group of mechanisms, the decoder sends feedback containing information about correctness of received stream which may be used by the encoder to adapt its policy to current network conditions. Reference Picture Selection (RPS), which is described below in more detail, is the example of such mechanism. In error concealment methods the decoder tries to mitigate effect of loss by interpolating missed frame region from adjacent regions or frames. Furthermore, H.264 introduces new error resilience tools like: Parameter Sets, Flexible Macroblock Ordering (FMO) and Redundant Slices (RS) [21]. Parameter Sets contain information about sequence of pictures, and slices belonging to a single picture. If delivered reliably to the decoder (e.g. by out-of-band methods like SDP in SIP) they greatly improve error resilience. Flexible Macroblock Ordering allows to assign macroblocks to slices in an other order than the scan order. Consequently, if packet containing information about slice group gets lost and FMO is used, other packets containing information about different slices provide much more data about whole frame which facilitates job for error concealment mechanisms. Redundant Slices are sent parallely to other primary slices, but they are encoded with lower quality. If the primary slice is received by the decoder, the RS is discarded. However, if the primary slice is lost, the RS may be used to properly decode picture.

Channel coding methods refer to error control mechanisms implemented on the physical and link layer. They include application of error resilient block or convolutional codes (e.g. Reed-Solomon codes, or Viterbi algorithm) on the physical channel.

Transport layer mechanisms are realised by means of lost packet retransmission, **Forward Error Correction (FEC)** [6], and interleaving. Retransmissions are based on **Negative Acknowledgement (NACK)** feedback, whereas FEC concept uses additional redundant packets to recover lost data. Both of these mechanisms are described in detail below. Interleaving idea makes use of the fact that human perception can easier tolerate individual losses than burst ones. In this mechanism sequence of video frames is reordered using some distribution pattern before being sent, so that neither two adjacent frames are sent one after another. The receiver must know distribution pattern to reorder packets back. As a result, in case of burst losses it is less likely that two consecutive frames are lost, and single losses are easier to conceal. Because it is necessary to reorder received packets, this method is not suitable for conversational services, as their delay budget does not allow for packet reordering.

Examples of application layer mechanisms are **Unequal Error Protection (UEP)** and robust scheduling. UEP closely cooperates with FEC and decides which parts of packets are more important than others, and should receive better error protection by means of FEC, or different video encoding. Similar concept is used for robust scheduling. In this method more important parts of picture data are sent before less important ones. As a result, in case of unexpected throughput change, it is still possible to play video with lower quality.

The rest of this chapter discusses in more detail four chosen mechanisms. Namely, **NACK**, **Slice Size Adaptation**, **Reference Picture Selection** and **Forward Error Correction**.

4.1 NACK

In RTP session the receiver is able to detect packet losses by detecting gaps in sequence numbers of received packets. The receiver periodically sends RTCP reports to the sender informing about general reception statistics. Extended feedback profile (RTP/AVPF) [26] allows the receiver to send the generic NACK message indicating which packets recently sent are regarded by the receiver as lost. When the packet loss is detected, the receiver must check if it is allowed to send the NACK packet, as it is constrained by RTCP session bandwidth limits, and decide based on perceived RTT and playout buffer delay if the retransmitted packet does have a chance to be delivered before it should be played. When the sender receives the NACK packet, it retransmits all lost packets indicated in this message. RFC 4588 [28] defines RTCP retransmission packet format which is used for resending the lost packet. Retransmitted packets are sent in the different stream to the original ones. It is necessary to separate these streams (either by creating different session, or by using different SSRC), so as not to disrupt RTCP statistics and allow for usage of multiple payload types inside one session.

4.2 Slice Size Adaptation (SSA)

Groups of macroblocks belonging to the same video frame form a slice. H.264 standard defines 5 slice types. Namely I-, P-, B-, SP-, and SI- slice. Macroblocks inside slices I, P and B can be decoded in similar way to types of frames equivalent to them (i.e. macroblocks inside I-slice can be decoded independently, and so on). The other two slice types (Switching P - SP, Switching I - SI) are used for encoding of efficient switch between bitstreams coded at various bit rates. Video coding standards allow to adapt slice sizes to existing channel conditions. For lossy channels it is possible to use slices containing just one macroblock, whereas in lossless ones the slice may contain even whole picture. Usage of large slices improves coding efficiency. However, if such slices exceed MTU of underlying link layer, they have to be fragmented and it makes them more vulnerable to frame losses, as it is sufficient just to loss a part of fragmented frame to regard whole frame as lost.

4.3 Reference Picture Selection (RPS)

Modern video codecs (e.g. H.263 [29], H.264) allow not only to use the most recent video frame for predictive decoding, but also older reference ones. Therefore if the encoder learns which pictures were correctly received by the decoder, it can use one of these (previously buffered) pictures as the reference picture for inter-prediction encoding. Usage of this method stops the temporal error propagation caused by a lost packet. On the other hand, if older pictures are used as reference ones, more bits must be used to encode new pictures, as differences between the current picture and the older reference picture is greater than between the current one and the most recent one.

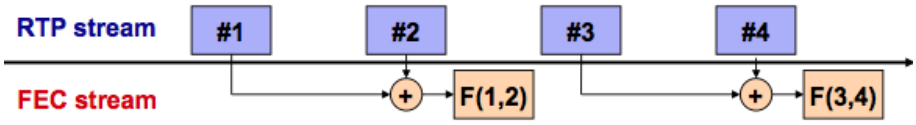
The receiver conveys information about in-/correctly received frames using the extended feedback profile (RTP/AVPF) defined in RFC 4585 [26]. It can operate by reporting missing frames (NACK mode), or correctly received ones (ACK mode). In either mode the encoder knows which frame can be used as the reference one. In order to minimise feedback overhead, the mode of operation can be chosen based on perceived number of frame losses. Feedback messages are carried and encapsulated into RTCP reports using RFC 4585 rules.

4.4 Forward Error Correction with Uneven Error Protection

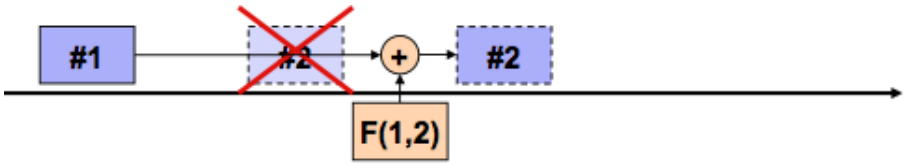
Forward Error Correction (FEC) is the error control method in which the sender adds redundant data to its messages which can be used by the receiver to recover original data that get lost over the channel during transmission. Redundant data are obtained by performing mathematical operations on main data. As a result of the mathematical operation, from **k** initial data packets, **n (n>k)** packets are sent, and the receiver can obtain whole content if up to **n-k** packets were lost. These operations may include parity, Reed-Solomon, or Hamming codes. This method was invented by Richard Hamming in the late 1940's and since then became the standard channel coding mechanism.

Beside physical layer FEC, there is also FEC associated with transport/application layer of multimedia streams [30]. As quality of multimedia conversations suffers from unavoidable packet losses, and strict delay requirements of real-time conversations limit applicability of lost packet retransmissions, it was proposed to additionally protect packets with FEC. This method is very useful in scenarios where retransmissions are impossible and extra bandwidth is available.

The FEC-capable RTP sender takes a set of packets from the media stream and performs mathematical operation (e.g. XOR) across them, generating FEC packets, and sends them along it (see fig. 13). As a result, the receiver is able to recover any packet used to generate the FEC packet. More advanced mathematical functions (e.g. Reed-Solomon codes) give more protection to the original media stream by allowing more packet losses, but they require more computational resources. The smaller the set of packets used for FEC generation is, the better these packets are protected. However, when these sets are too small, too much bandwidth may be used by FEC leading to congestion. Generally packet protection given by FEC is a trade-off between the available bandwidth and the protection strength.



(a) FEC sender



(b) FEC receiver

Figure 13: Concept of FEC sender and receiver operation. *Picture adopted from* [5].

Uneven Level Protection (ULP) was created to provide FEC protection in scenarios where the available bandwidth is relatively small (e.g. wireless channels). ULP concept is based on observation that not all parts of the media packet are of the same importance. Therefore, it is sensible to provide different level of protection to different packet fragments, and use more bandwidth for crucial packet parts. Example of ULP concept is presented in fig. 14. Audio and video packets are divided into parts of decreasing importance. For instance, in Audio Redundancy Coding (ARC) the first payload part carries current audio sample followed by other parts with samples of previous frames, which are less important, as they are only used if primary encodings of these frames were lost. Video coding standards divide frames into reference ones, which are used for future inter-picture prediction, and non-reference ones. Consequently, the former are more important than the latter, and FEC protection may be applied just to them, leaving non-reference frames without protection.

FEC packets are sent in a separate stream, so that receivers which are not FEC-capable can also participate in multimedia sessions. This approach may be very useful for conference scenarios.

The sender constructs the FEC packet by placing it inside RTP payload (see fig. 15a). Meaning of RTP header fields of FEC packets follows general RFC 3550 rules, limiting only payload type value to the dynamic range which is determined by out-of-band signalling. This approach guarantees backward compatibility for receivers without FEC support. The FEC header (see fig.15b) carries information about RTP headers of protected packets. *SN base* field indicates sequence number of lowest packet protected, whereas *TS recovery*, *PT recovery*, *M*, *CC*, *X* and *P* carries values calculated by the protecting function based on values of these fields in original RTP packets. *length recovery* field carries information about lengths of payloads and RTP extensions (if present) of original RTP packets. The FEC payload consists of multiple parts which are associated with every protection level. Every part is divided into FEC level header and FEC level payload which carries protection. The FEC level header contains information about protection length for this level and the mask indicating which RTP packets are protected by this FEC level.

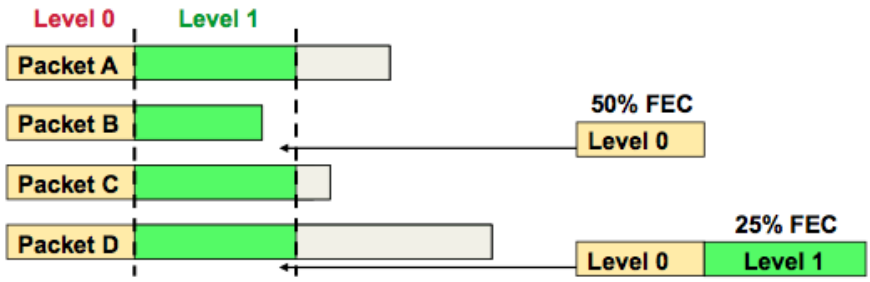
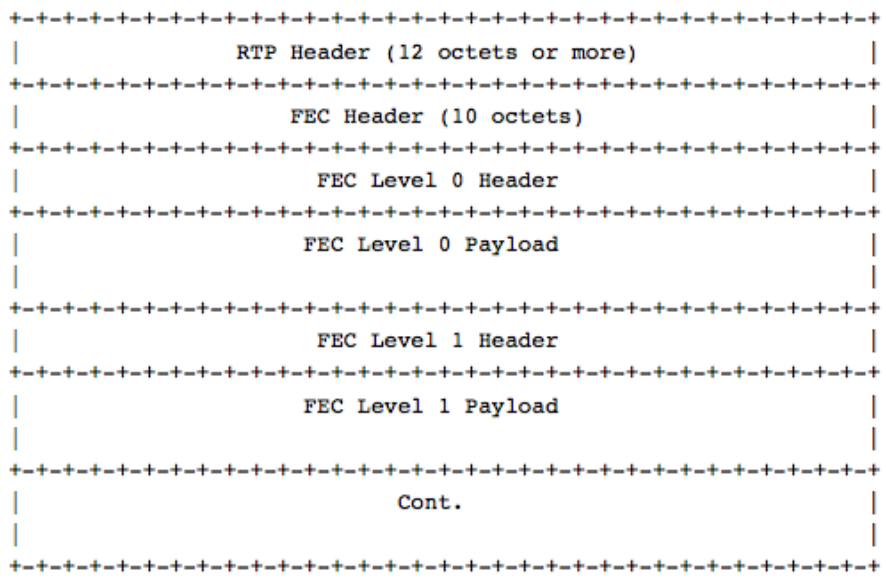
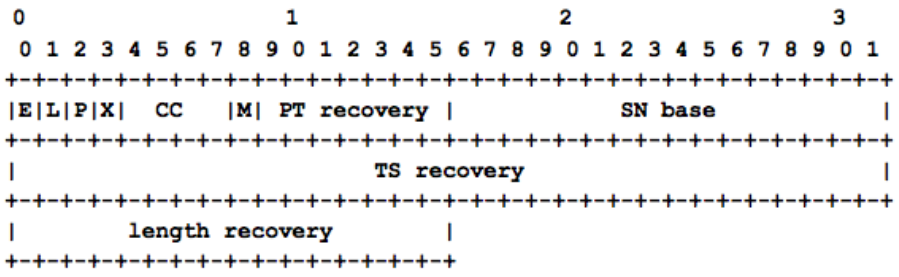


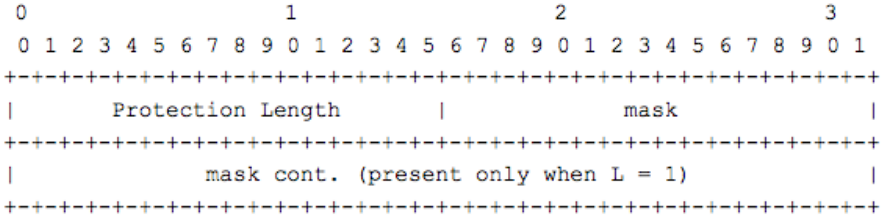
Figure 14: Example of ULP: The first FEC packet has only level 0 protection for packets A and B. Protection operation is performed only on parts of these packets, leaving other parts of them unprotected. The second FEC packet also has level 0 protection for packets C and D, but it also gives level 1 protection for less important parts of packets A, B, C and D. *Picture adopted from* [5].



(a) FEC packet structure



(b) FEC header format



(c) ULP level header format

Figure 15: ULP FEC packets format. *Picture adopted from* [6].

4.5 Summary

Performance of error resilience mechanisms highly depend on end-to-end delay and channel packet loss ratio. Applicability of NACK is limited to low end-to-end delay scenarios where packet losses are also not very frequent in order to keep retransmission overhead in the acceptable level. Performance of RPS is proportional to end-to-end delay and packet loss ratio, and it decreases along its increase. SSA is applicable in most of scenarios. [31] does not define optimum usage scenario for FEC with ULP , but it is expected to perform well in high delay scenarios where repair cost by means of interactive mechanisms is high.

To sum up, there is more than one error resilience mechanism applicable for a particular scenario, and there is a need to create the complex and adaptive error resilience mechanism which would be able to adapt itself to existing conditions by choosing subset of its mechanisms applicable for a particular scenario [31]. Usage of error resilience mechanisms can be further extended if they are joined with addition of congestion control mechanisms. By reducing the sending rate, we are able prevent congestion errors from occurring if the network conditions do not allow for keeping the rate. Furthermore, error resilience mechanisms help minimise effects of losses caused by an inappropriate rate increase.

5 Rate adaptation of multimedia flows

This chapter focuses on rate adaptation of multimedia flows. It discusses problem of congestion in the Internet, and its control methods. Later it describes congestion indicators for multimedia flows. *TCP Friendly Rate Control for RTP* (TFRC for RTP) [8] rate adaptation algorithm is presented in the end as the example.

5.1 Congestion in the Internet

The Internet is a packet switched network that unlike circuit switched networks takes advantage of the statistical multiplexing giving better bandwidth utilisation. Lack of any access control mechanisms in the core Internet concept allows injecting unlimited amount of data into the network. As speed of links and router capabilities are limited, packets buffering in the intermediate routers was introduced, so as to queue incoming packets which cannot be handled instantly. Furthermore, as buffers also have limited space, packets are dropped when the router buffer is filled above some threshold. Figure 16 illustrates possible congestion scenario. In this example the queues of A and P routers are not congested (blue colour in the figure), and both of these routers forward packets to the B router, whose queue is more filled (green colour in the picture). The B router and the X one, whose queue filling is comparable to the B one, forward their packets to the C router, which is unable to handle that many incoming packets. As a result, its queue gets full, and it starts to drop newly arriving packets.

Providing satisfactory service for multiple flows, requires the network end-points to adapt their rates to existing network conditions. TCP traffic has built-in congestion control algorithms allowing rate adaptation and being fair to other TCP flows. As multimedia flows are often sent over UDP, congestion control mechanisms must be implemented for them at the application layer. Multimedia applications may perform its rate adaptation by switching between codecs (e.g. PCM and GSM), using variable bit rate codecs (e.g. SVC H.264), or by adjusting frame rates (e.g. 30 frames/s to 15 frames/s).

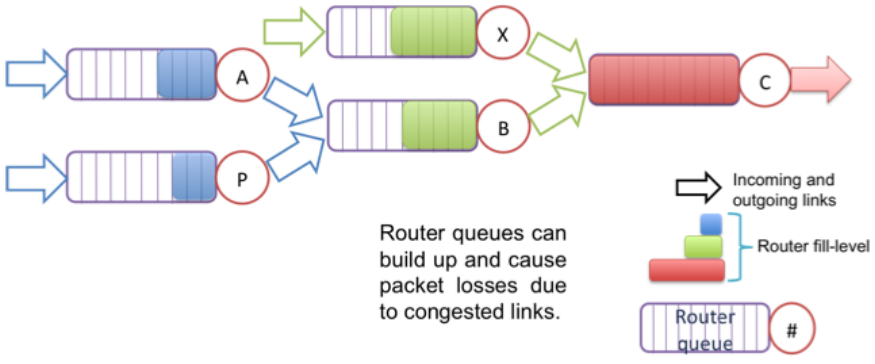


Figure 16: Congested queue in the router. *Picture adopted from* [3].

5.2 Introduction to rate adaptation

In order to implement rate adaptation, the sender must be informed about the receive rate by the receiver in the feedback path. In TCP, receivers send feedback packets acknowledging correct reception of sent segments. Based on these ACK packets, TCP senders change their sending rates to adjust to current network situation. In multimedia traffic feedback is provided by RTCP reports, which are sent far less often than TCP feedback. RFC 4585 defines extended feedback profile (RTP/AVPF) [26] which allows sending more frequent feedback, but even when this profile is used, its frequency may be lower than in the TCP case.

Rate adaptation methods are divided into three different types based on decision maker [32]. In **sender driven rate adaptation** the receiver collects network statistics and conveys them to the sender which uses them to adapt its sending rate (e.g. TFRC [33]). In **receiver driven rate adaptation** the receiver having collected all required data chooses appropriate sending rate, and transfers it to the sender which must adapt to it (e.g. TMMBR/TMMBN [34]). In **network driven rate adaptation** an element in the network (e.g. RTP mixer) monitors network conditions and in case of their change signals this information to the sender or receiver (e.g. TMMBR/TMMBN [34]). This type of rate adaptation is particularly useful for conference scenarios where session participants have no knowledge about network conditions for other participants.

In TCP congestion control algorithms it is possible to distinguish two main phases: **slow-start** and **congestion avoidance**. In the slow-start phase TCP senders aggressively increase their sending rates by expanding the congestion window for every acknowledgement of properly received data segment. In the second phase the congestion window is increased additively with every received ACK, and decreased multiplicatively when segment losses are reported.

In multimedia congestion control algorithms, it is more difficult to distinguish phases of congestion control because of different requirements of the multimedia traffic in comparison to the TCP one. As TCP guarantees end-to-end reliability by retransmission of lost segments, it is not concerned so much with segment losses. However, multimedia packets carry real-time data which should be delivered to the receiver in time constrained by the delay budget. Delivery of multimedia packet in time exceeding the delay budget makes them totally useless for the receiver. Therefore congestion control algorithms of multimedia flows must primarily avoid packet losses and discards, and look for sending rate increase only when the former is ensured. As a result, **congestion avoidance**[1] and **congestion mitigation** phases can be distinguished.

In the congestion avoidance phase, the sender/receiver based on congestion metrics included in RTCP reports tries to detect if congestion may appear soon, or if path capacity is underutilised. If congestion appearance is predicted, the sending rate is decreased, whereas if link underutilisation is perceived, the sending rate can be increased. However, it is also possible that RTCP feedback comes too late to

[1] This term should not be confused with TCP congestion avoidance. We use it because of lack of any other meaningful term.

predict congestion appearance and path has already become congested. In such situation the sender/receiver enters the congestion mitigation phase in which it aggressively cuts its sending rate to minimise congestion effects on the transmission quality.

In multimedia rate adaptation we distinguish three operation modes: **overshooting**, **undershooting**, and **stability**. In TCP rate adaptation similar modes exist and their meaning is also identical. During overshooting the sending rate exceeds current link capacity which leads to increased queuing delays, packet discards, and finally packet losses, therefore it should always be avoided. This mode can be entered either by incorrect perception of available bandwidth, by unexpected change of the path capacity, or by bandwidth probing as in TCP case. During undershooting the sending rate is below available path capacity which leaves bandwidth underutilised. This mode may result from conservative bandwidth estimation by the sender/receiver, algorithm construction (in TCP case), or be used to quicker mitigate congestion effects on the congested link. During stability mode the sender maintains its sending rate around the path capacity. Figure 17 illustrates these rate adaptation operation modes.

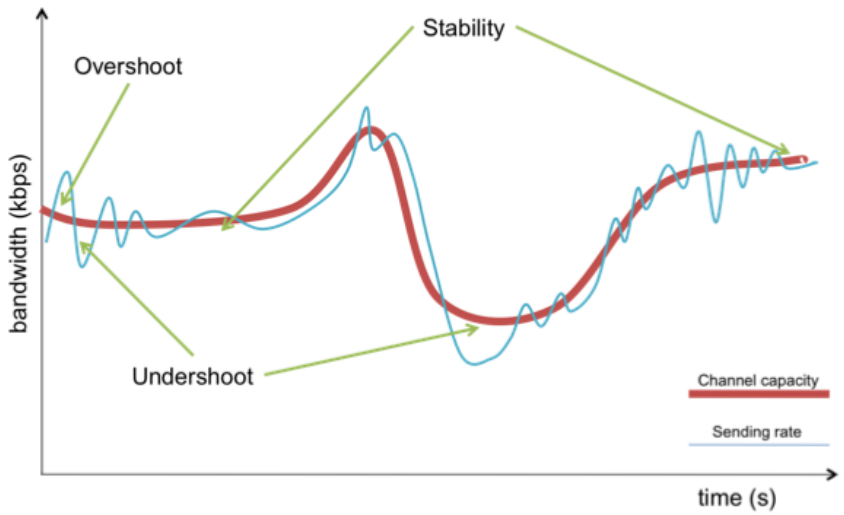


Figure 17: Modes of rate adaptation. *Picture adopted from* [3].

5.3 Congestion indicators in multimedia flows

RTCP feedback packets deliver receiver's reception statistics to the sender. Basic RFC 3550 RTCP profile defines just four metrics for Receiver Reports (RRs) losses, receiver rate, jitter and RTT. RTCP Sender Reports (SRs) carry information about its sending rate. For conversational video, these metrics are insufficient, and consequently IETF defines RTCP extensions allowing to convey additional data to properly assess congestion situation. Many extensions were defined, among which

RTP Control Protocol Extended Reports (RTCP XR), presented in RFC 3611 [7], provides the richest set.

5.3.1 Packet losses and discards

Packet losses in the network are divided into bit-error losses and congestion ones. Bit-error losses are caused mainly by signal fading and interference, and therefore appear randomly which makes them impossible to predict. Conversely congestion losses result from over-utilisation of link capacity, and most often can be predicted. Before packets get lost, a sequence of discarded packets arriving too late at the receiver should appear. Therefore packet losses indicate state of heavy congestion in the network, and packet discards provide early signs of incoming congestion.

Packet losses are reported in standard RTCP specification [4], which does not include discard metric. Despite several proposals of conveying discards statistics [35][36], no standard has been defined.

5.3.2 One-way delay (OWD) and Round-trip time (RTT)

One-way delay (OWD) is the amount of time needed for the packet to traverse the network from the sender to the receiver. This time includes physical signal propagation time, and delays caused by packet serialisation, its queuing in intermediate routers, and processing by them. On the other hand, the Round-trip time (RTT) is the amount of time needed for the packet to go from the sender to receiver and back. As delay caused by propagation, serialisation and processing is almost constant in the wired Internet, measurements of RTT/OWD vary only by queuing delay. Thus, higher RTT may indicate increase of queuing delay, which is an early sign of upcoming congestion. On the contrary the same conclusion cannot be made in the mobile networks, where additional delay may result from retransmission of corrupted frames caused by interference and fading effects.

RTT measurements are included in standard RTCP specification [4], and are performed collaboratively by the sender and the receiver. The sender remembers exact time of generating last SR, and the receiver informs it in the RR of delay between sending its own RR and reception of the last SR. As a result, when the sender gets RR, it is able to calculate RTT.

$$RTT = Time_{recv\_RR} - Time_{snd\_SR} - Time_{delay} \tag{1}$$

where :

$Time_{recv\_RR}$ – *reception time of last RR*

$Time_{snd\_SR}$ – *last SR sending time*

$Time_{delay}$ – *receiver's delay between SR reception and RR sending*

Calculation of OWD is much more complicated, because clock synchronisation between the sender and the receiver cannot be assumed. Very often it is assumed that:

$$OWD = \frac{RTT}{2} \tag{2}$$

However, equation above is only true if the network's upstream and downstream paths are symmetric, which is difficult to accurately determine. [37][38] present algorithms allowing receiver to precisely evaluate packet OWD. The result may be conveyed back to the sender using RTCP APP packet.

5.3.3 Sending rate ($\mathbf{BR_S}$), Receive rate ($\mathbf{BR_R}$), Goodput (GP)

Sending rate is the rate at which packets are injected into the network, whereas receive rate indicates rate at which packets arrive at the receiver. Goodput is similar to the receive rate, but it is calculated only based on properly received packets (i.e. not discarded ones). Lower receive rate (or goodput) in comparison to sending rate may indicate network congestion. The following equations presents how these metrics can be calculated using content included in the RTCP reports.

$$Sending\ Rate(BR_S) = \begin{cases} \frac{\sum_{i=last\_SR_{LPS}}^{LPS} S_i}{Time_{now} - Time_{last_SR}} & calculated\ by\ the\ sender \\ \\ \frac{octet_{now} - octet_{last\_SR}}{Time_{now} - Time_{last\_SR}} & calculated\ by\ the\ receiver \end{cases} \tag{3}$$

$$\begin{aligned} &where: \\ &\quad S_i \quad - \ size\ of\ the\ packet\ i \\ &\quad octet \ - \ sender's\ octet\ count\ value\ included\ in\ SR \end{aligned}$$

$$Receive\ Rate(BR_R) = \frac{\sum_{i=last\_RR_{HSN}}^{HSN} S_i}{Time_{now} - Time_{last\_RR}} \tag{4}$$

$$\begin{aligned} &where: \\ &\quad S_i \ - \ size\ of\ the\ packet\ i \end{aligned}$$

$$Goodput(GP) = \frac{\sum_{i=last\_RR_{HSN}}^{HSN} I_i \times S_i}{Time_{now} - Time_{last\_RR}} \tag{5}$$

$where:$

$S_i \quad - \ size\ of\ the\ packet\ i$

$I_i \in \{0,1\},\ where\ I_i = 0\ when\ the\ packet\ i\ lost\ or\ discarded,$
$and\ I_i = 1\ otherwise$

The receiver may calculate sending rate from data included in standard SRs, whereas receive rate can be calculated by the sender from RRs. For goodput calculation RTCP extensions containing discard metrics ([35][36]) must be used.

5.3.4 Jitter

RTP sources usually generate and send packets periodically. However, dynamic network conditions may change packet's OWD, making them appear aperiodically at the receiver. Jitter is the estimate of the statistical variance mean of the RTP packet inter-arrival time [4]. Increase of packet inter-arrival time indicates growing congestion, whereas its decrease suggests possible end of congestion. RFC 3550 [4] defines following method of jitter calculation:

$$J_m = J_{m-1} + \frac{D_{m-1} - J_{m-1}}{16} \tag{6}$$

$where:$

$m-1,\ m\ are\ consecutively\ received\ packets$

However, jitter metric is of limited usefulness, as it covers inter-arrival time information just about last 16 packets, whereas unless extended feedback profile (RTP/AVPF) is used, almost all RRs cover period of time when more packets were received. As a result, jitter metric, conveyed in the RR, usually does not contain information about the whole report interval, but just about a fraction of it.

5.3.5 Other indicators

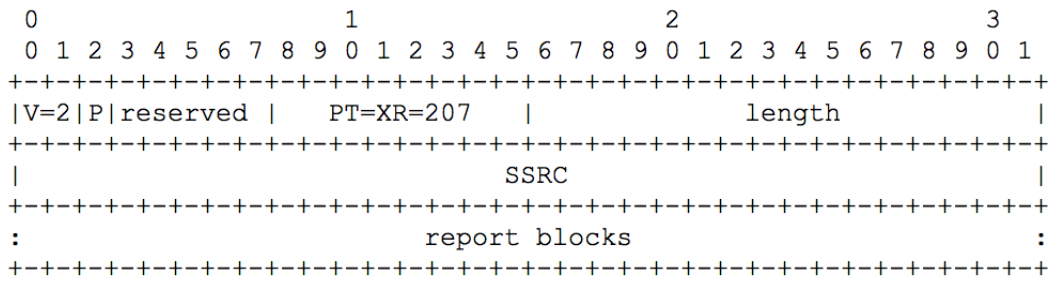
All above presented indicators are accessible in RFC 3550 [4] standard. We focus mainly on them, because they are already included in the basic RTP standard, thus they do not introduce any bandwidth overhead. However, there are many other RTCP extensions allowing to transfer other important data for rate adaptation. Few of them are briefly summarised in the table 1. Furthermore, in the course of our work, we decided to use two additional extensions (Loss RLE Report Block and Discard RLE Report Block) to gather more detailed information about the nature of packet losses and discards. These two extensions are more precisely described in the table 1 and their RTCP headers are presented in the figure 18.

5.3.6 Summary of congestion indicators

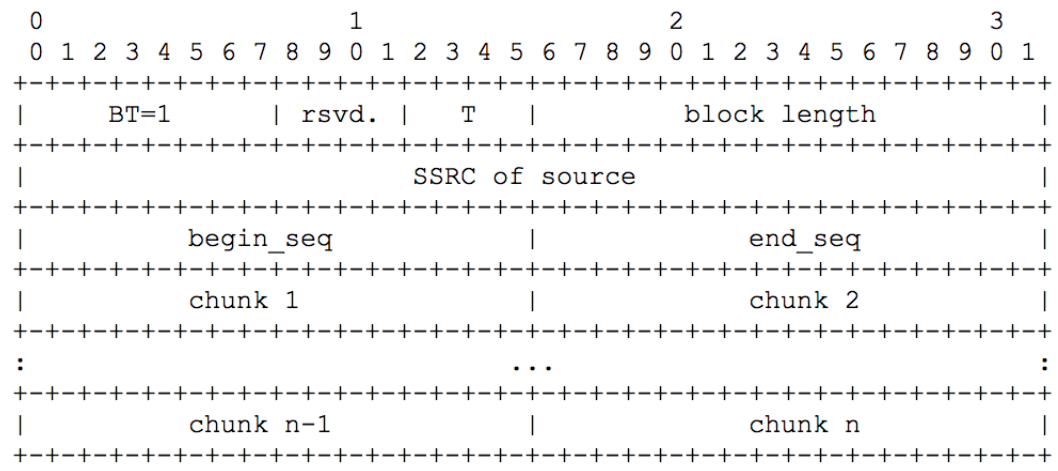
Congestion indicators provide essential information for rate adaptation. They are observed by the sender, or the receiver, or collaboratively. Rate adaptation algorithms often use just a subset of existing indicators. The rate adaptation driving node senses all necessary indicators on its side, and gathers the remaining ones from the RTCP reports. Based on input provided by the indicators, the driving node decides which rate is correct in the current network situation.

Table 1: Summary of important RTCP extensions

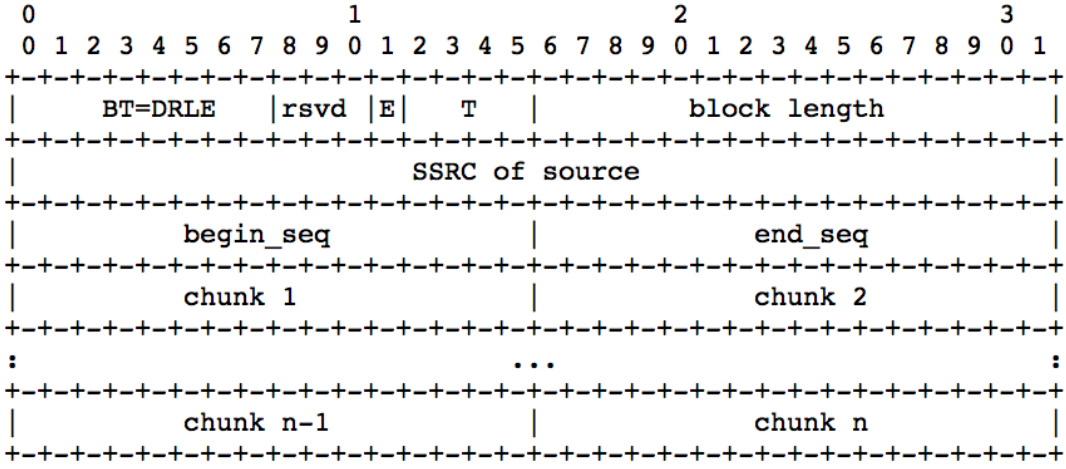
Extension name	**Description**
Detailed packet reporting	Belongs to RTCP XR. Allows the sender to be informed about sequence numbers of lost or duplicated packets, and exact reception times of send packets
Receiver side RTT calculation	Belongs to RTCP XR. Allows non-senders to calculate their RTTs
Loss RLE Report Block	Belongs to RTCP XR. Informs exactly which packets were lost. This extension's header is presented in the figure 18b. *begin_seq* and *end_seq* defines boundaries of the beginning and the end of the reported sequence number space. Chunks can be represented as run length, bit vector, or terminating null (see [7] for more details).
Discard RLE Report Block	Belongs to RTCP XR. Informs exactly which packets were discarded due to too early or too late arrival. This extension's header is presented in the figure 18c. *begin_seq* and *end_seq* defines boundaries of the beginning and the end of the reported sequence number space. Chunks can be represented as run length, bit vector, or terminating null (see [7] for more details).
Duplicate RLE Report Block	Belongs to RTCP XR. Informs exactly which packets were duplicated
Statistics Summary Report Block	Belongs to RTCP XR. Contains general reception statistics not included in standard RR (losses, duplicates, jitter, TTL)
VoIP Metrics	Belongs to RTCP XR. Contains metrics for monitoring VoIP calls (losses, discards, delay, analog metrics, as well as, quality metrics)
Next Application Data Unit	Standard 3GPP extension for Packet switched Streaming Service (PSS). Contains information to recreate receiver buffer (playout delay of the first packet in the queue, free buffer space and sequence number of the first buffered packet)
Feedback Messages	Defined in RFC 4585. Contains transport-, application-layer and payload specific feedback extensions. These headers can convey information about packet receptions (ACK/NACK mode), RPS, Slice Loss Indication, etc.
Codec Control Messages	Contains feedback information facilitating rate adaptation for scenarios with mixers and translators



(a) RTCP XR common header



(b) Loss RLE header in RTCP XR



(c) Discards RLE header in RTCP XR

Figure 18: RTCP XR headers. *Picture adopted from* [7].

5.4 Rate adaptation metrics

Ultimate target of all rate adaptation algorithms is to minimise losses, and simultaneously maximise bandwidth utilisation. An ideal algorithm would be able to perfectly adapt to changeable network conditions by avoiding any congestion losses and use entire bandwidth given to it. In reality such an algorithm does not exist, but there are numerous other solutions which are designed based on some trade-offs, and therefore they perform well in some scenarios, and bad in others.

In order to assess which algorithm is the best suited one for a particular scenario, rate adaptation metrics providing objective tools of evaluation are used. There are many available metrics which can be used for this task. In this work, we present the brief description of three of them: **Peak Signal-to-Noise Ratio (PSNR)**, **Average Bandwidth Utilisation (ABU)**, **Average/Instant Sending Rate/Goodput**. These metrics are later used by us to evaluate performance of algorithms developed for this work.

5.4.1 Peak Signal-to-Noise Ratio (PSNR)

PSNR is commonly used engineering metric to evaluate reconstruction quality of image after its lossy compression. It compares the original signal power with the noisy one. In terms of video communication, the original signal is the one obtained by the video camera in YUV format, whereas the noisy one is the signal which was compressed, sent over the network, and decompressed at the receiver.

PSNR calculation uses **Mean Square Error (MSE)** to evaluate difference between the original video, and the received video. For video frame of size MxN MSE is defined by the following formula:

$$MSE = \frac{1}{M \times N} \sum_{M,N} [I(m,n) - J(m,n)]^2 \tag{7}$$

$$\begin{aligned} &where: \\ &\quad m = 1, 2, 3, \ldots, M \\ &\quad n = 1, 2, 3, \ldots, N \\ &\quad I - original\ video\ frame \\ &\quad J - received\ video\ frame \end{aligned}$$

The above formula is true only for monochrome video frames. Human eye is much more sensible to picture luma (brightness) than to colour information carried in chrominance (see fig. 19). Therefore the luma component (Y) in YCbCr picture format is the weighted average of brightness information carried in RGB channels, and the MSE formula can be modified to the following form:

$$MSE_{colour} = \frac{1}{3} \times MSE_{monochrome} = \frac{1}{3} \frac{1}{M \times N} \sum_{M,N} [I(m,n) - J(m,n)]^2 \tag{8}$$

PSNR is defined by the following formula in which MAX_I is the maximum pixel value:

$$PSNR = 10 \log \frac{MAX_I^2}{\sqrt{MSE}} = 20 \log \frac{MAX_I}{\sqrt{MSE}} \tag{9}$$

Although PSNR is widely used to evaluate perception of video quality, it has some limitations resulting from the nature of human brain. As PSNR calculation is based entirely on MSE of pixels, it is possible that a picture with blurred foreground has higher PSNR than an other picture with blurred background, whereas the human vision percepts quality of both such pictures differently. As PSNR has poor correlation with human vision perception, there is done a lot of research in area of finding alternative method of assessing video ([39] [40]) is done. Another important finding of recent human's vision research is the fact that human's vision is more sensitive to salient objects [41]. All these findings prove that PSNR is not the ideal quality metric. However, as it is still the most widely applied one, we decided to use it.

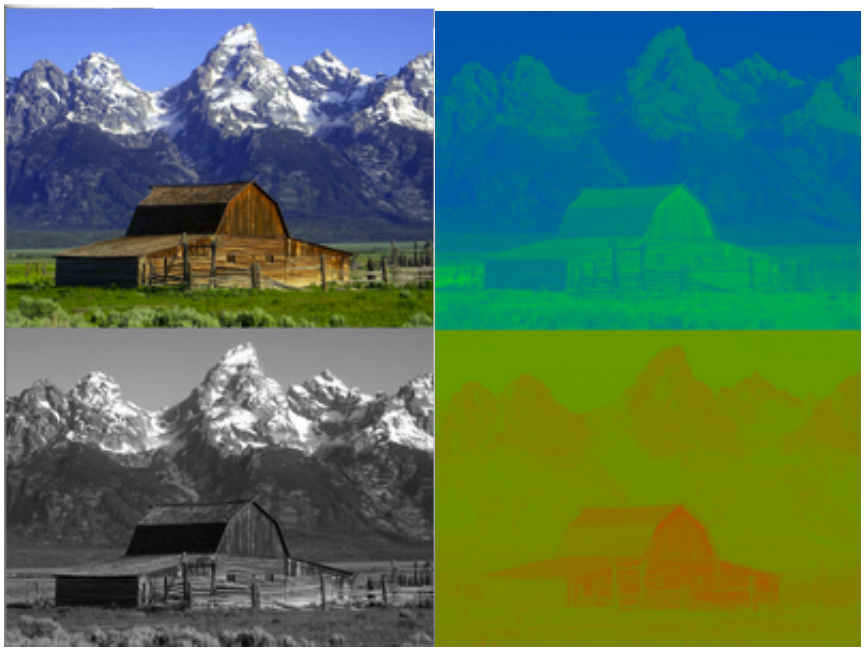


Figure 19: Decomposition of image into luma part (brightness), and chrominance (colour components). *Picture adopted from Wikipedia en.wikipedia.org.*

5.4.2 Average Bandwidth Utilisation (ABU)

Average Bandwidth Utilisation metric, defined in [3], describes relation between usage of network bandwidth by the video stream and the available bandwidth. ABU can be calculated for sending rate, receive rate, and goodput. For values significantly below 1 there is channel underutilisation, whereas values above 1 represent over-utilisation. ABU is calculated from the following formula:

$$ABU = \frac{\int s(t)}{\int c(t)} \tag{10}$$

$$\begin{aligned} &where: \\ &\quad s(t) - sending\ rate,\ or\ goodput,\ or\ receive\ rate \\ &\quad c(t) - channel\ bandwidth \end{aligned}$$

As ABU is the function of time, it is also beneficial to plot Cumulative Distribution Function (CDF) of all available ABU values to notice general trends.

5.4.3 Average/Instant Sending Rate/Goodput and other metrics

Quality of rate adaptation algorithms can also be evaluated with instant or average sending rate/goodput. The higher the value is, the better the algorithm provided that it does not kill other flows with too high sending rate. General statistics can also be used as metrics for evaluation of some algorithm aspects. These metrics are very similar to average/instant sending rate/goodput. They may include cumulative number of lost/discarded packets, average number of packets in the receiver buffer, cumulative amount of time when the receiver buffer is empty, etc.

5.5 Example of rate adaptation algorithm: TFRC for RTP

TCP Friendly Rate Control [33] is the sender driven rate adaptation algorithm for unicast UDP flows, which allows them to fairly compete for network bandwidth with TCP flows. Unlike TCP-based congestion control mechanism, TFRC is designed to ensure smooth sending rate with far smaller throughput variations over time than TCP. Consequently TFRC is useful for multimedia applications where relatively steady bandwidth is important. The algorithm is designed for best performance in applications adapting their rates by varying number of packets sent in the time interval. On the other hand, the multimedia applications require sending packets in regular constant intervals, and they adapt their sending rates by varying packet sizes. Therefore standard version of TFRC [33] is not optimal for these applications, and they should use the modified algorithm version (**TFRC-SP**) defined in RFC 4828 [42].

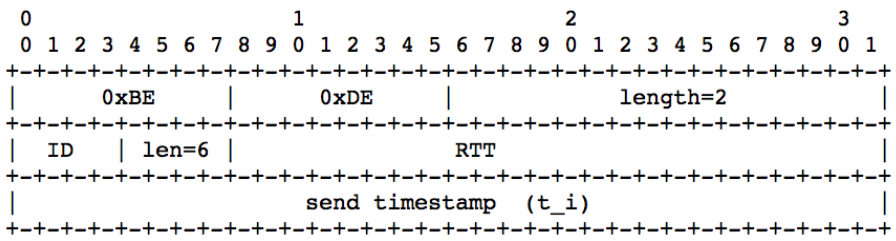
TFRC is based on frequent feedback with 2 congestion indicators **RTT** and **loss event rate**. As RTT is included in feedback packets, and is calculated cooperatively by both sides, TFRC requires the sender to provide the receiver with packet sending

timestamp allowing to calculate RTT. TFRC's operation is divided into the slow-start phase and the congestion avoidance one. In the former the sender is allowed to double its rate per RTT interval until first loss occur which begins the latter. In the congestion avoidance phase the sender uses the **TCP Throughput Equation** (11) to calculate its sending rate.

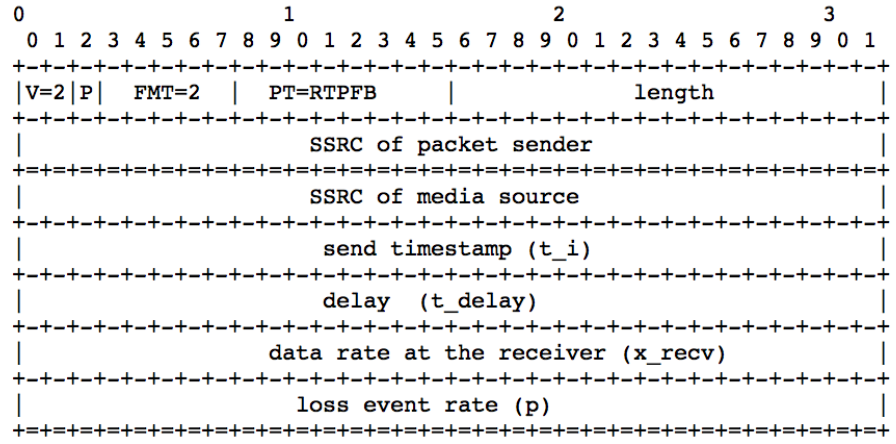
$$X_{kbps} = \frac{seg\_size}{R \times \sqrt{\frac{2 \times b \times p}{3}} + t_{RTO} \times (3 \times \sqrt{\frac{3 \times b \times p}{8}}) \times p \times (1 + 32 \times p^2)} \tag{11}$$

Standard RTP defined in RFC 3550 [4] is not adapted for TFRC usage. Therefore RTP/RTCP extensions were proposed in [8] to permit its usage with TFRC. [8] defines TFRC feedback packet, which belongs to RTCP feedback loop, and should be sent to the sender once per RTT, or for every received packet if RTT estimation is not available. Such frequent feedback significantly increases overhead needed for rate adaptation. Furthermore, it is often necessary to send TFRC feedback encapsulated into compound RTCP reports, as many RTP/RTCP implementations accept just compound reports. As a result, 5% session bandwidth rule defined for RTP/AVPF profile [26] can be violated. [8] redefines feedback timing rules for short delay networks (where $delay < 20ms$) to guarantee complying with 5% rule. The feedback message conveys information about estimated data rate in the last feedback interval, and loss event rate. Additionally it contains also timestamp of the last received RTP packet, and delay between this packet reception and feedback generation. These two elements are used by the sender to calculate RTT. RTCP feedback extension header is presented in the figure 20b. Extension in RTP includes additional 12 bytes informing the receiver of sender's RTT estimation, and exact packet sending timestamp. This extension header is presented in the figure 20a.

The algorithm operates in the following way. The receiver generates feedback packet if it detects a new loss event, or if no feedback packet has been sent upon last feedback timer expiry. The sender adapts the sending rate when the feedback packet is received. If the loss event rate value equals 0, the sender is in the slow-start phase, and it can approximately double its rate each round trip time. However, when the loss occurs (the loss event rate is greater than 0), the sender enters the congestion avoidance phase and calculates the rate using the (11) equation.



(a) RTP extension for TFRC



(b) RTCP extension for TFRC feedback

Figure 20: Extension headers for RTP/RTCP in TFRC algorithm. *Picture adopted from* [8].

5.6 Summary

Internet is a packet switched network, which is vulnerable to congestion. Rate adaptation methods can be divided into three types: the sender driven rate adaptation, the receiver driven one, and the network driven one. Three modes of rate adaptation are distinguished: overshooting, undershooting, and stability. Network conditions can be sensed using various congestion indicators. Performance of various rate adaptation algorithms is evaluated using rate adaptation metrics.

For rate adaptation of multimedia flows it is essential to avoid any losses and look for the rate increase only when the former is guaranteed. This approach is different from the one appropriate for TCP flows, where packet delivery within specified delay budget is not required, leaving more space for retransmissions of lost data.

6 Rate adaptation algorithms

This chapter describes two new RTP rate adaptation algorithms designed for this thesis. They are called FEC Based Rate Adaptation (FBRA), and Non-FEC Based Rate Adaptation (N-FBRA). Both of them use identical subset of RTCP extensions to estimate network conditions and calculate sending rate. Additionally the FEC based algorithm uses also FEC to boost packet recovery probability if the increase of sending rate leads to packet losses.

6.1 FEC Based Rate Adaptation algorithm (FBRA)

The main idea behind this algorithm is usage of FEC for rate adaptation and packet recovery simultaneously. By taking this approach it is possible to apply more aggressive rate adaptation algorithm, as possible losses caused by wrong network conditions judgement during congestion period may be recovered using FEC packets.

The algorithm has following states: *s-*,*s+*,*s++*,*u*,*d*. State names derive from the words: s - *steady*, u - *up*, d - *down*. Additionally the minus sign means that FEC packets are not generated, whereas the plus sign indicates creation of FEC data. When the algorithm is in the "s-family" state, it does not change the sending rate. When it is in the *u* state, it increases the sending rate, whereas in the *d* state, the rate reduction occurs. The sending rate can increase only by the value of the FEC rate. As a result, the FEC stream is switched off, and bandwidth released by it is taken by the media stream. The *s-* state means that current rate is maintained, and generation of FEC packets is switched off. In *s+* and *s++* states the rate is also maintained, but FEC packets are also sent. These states are identical from the application point of view. The only difference between them is that the *s++* state can be entered only from the *s+* state if network conditions are stable for 2 consecutive RTCP report intervals. The figure 21 illustrates the algorithm's state machine, labels are explained in the table 2.

The algorithm specifies following state transition rules. In the *s-* state, the sending rate is kept constant, and FEC is not generated. The algorithm can stay in this state if congestion cues indicate that there is no additional bandwidth to take, or it can go to the *s+* state by probing the path with FEC packets. Usage of FEC stream gives more resistance to losses which may appear as a result of bandwidth increase. Direct transition to the *s+* state is not possible when the current sending rate is higher than 90% of the highest rate recorded in the last 2 seconds. In such case, the algorithm assumes that the current rate is close to the bandwidth limit, and it must make sure that current rate is stable enough by enforcing mandatory state in the current state for one more RTCP report interval. Obviously if the congestion is detected, the algorithm goes from the *s-* state to the *d* one.

In the *s+* state the algorithm maintains the sending rate, but it also sends additional FEC packets. If the next RTCP report content shows congestion signs the algorithm switches off the FEC stream and goes back to the *s-* state. Otherwise it keeps the media and FEC rates constant and goes to the *s++* state. It is also possible that the FEC rate is decreased if the report content indicates that current

amount of FEC overhead may lead to congestion, whereas smaller overhead can be appropriate. Similarly to the *s-* state, if the congestion is detected, transfer to the *d* state occurs.

In the *s++* state the algorithm keeps the rate constant and sends also FEC data. If in this state the RTCP report does not indicate any congestion, transition to the *u* state happens. Otherwise, the algorithm switches to either the *d* state. It is also possible that the report content indicates an in-between situation (the RTT is higher than expected, but there are no losses and discards). In such a situation, the algorithm either goes back to the *s-* state, or it stays in the *s++* state with the FEC rate reduced.

In the *u* state the algorithm increases the sending rate by exchanging the FEC rate for additional RTP rate. From this state, the algorithm may go only to the *s-* state with the new sending rate value, or go back to the *d* state if congestion is detected.

In the *d* state the sending rate is reduced. The rate decrease value is determined from the congestion indicators, and it is described later in more detail. In most cases the algorithm goes from this state to the *s-* one. However, if the congestion is very heavy and the next RTCP report indicates that the rate reduction does not resolve the problem, the algorithm cuts its rate once again and stays in the *d* state.

Finally the algorithm makes also use of early feedback RTCP report and report reception timeout. The receiver sends the report earlier than expected if number of packets in its buffer falls below 80% of its average value. As the sender is able to estimate when it should expect to receive the next report, it determines that every report received sooner than 1.5 of the median RTT value is the early feedback report, and therefore always goes to the *d* state. In addition, the report reception timeout is set to 2 seconds upon receive of every report. If no report is received in 2 seconds, this timeout is expired, and the algorithm enters the *d* state.

Transition from the *s-* to *s+* state requires the algorithm to determine amount of the FEC rate added to the media stream. This rate is calculated as the function of the current sending rate. The FEC rate depends on the FEC interval (i.e. number of media packets protected by a one FEC packet), which increases together with the current sending rate. The FEC interval may take values between 2 and 14. In number of simulations runs, it was concluded that for FEC interval values higher than 14 recovery rate falls almost to 0, and rate adaptation gain from making smaller sending rate increase steps is negligible.

The other very important feature of this algorithm is the network condition estimation. This can be accomplished by usage of extended feedback profile (RFC 4585 [26]), which allows for quick response to changing network conditions. For point-to-point RTP sessions RTCP transmission interval is almost always dependent on the minimum possible transmission interval, as it is virtually impossible to reach 5% of session bandwidth in this scenario. Because in extended feedback profile minimum RTCP transmission interval is not enforced, having conducted numerous experiments, we decided to use value of double RTT as the RTCP interval. The receiver may also send an early RTCP report, but then it must postpone time of next report sending to maintain the two-RTT rule. The algorithm orders sending

the early feedback if number of packets stored in the receiver buffer falls below 80% of its average value.

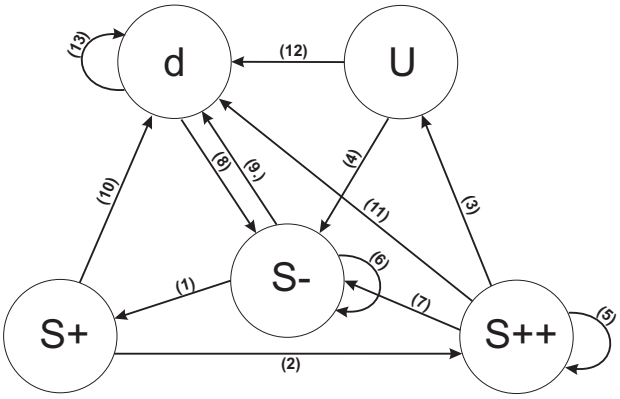


Figure 21: FBRA state machine

Table 2: Conditions of FBRA state transition illustrated in the figure 21.

No.	**Transition conditions**
1	no losses $\wedge$ no recent discards $\wedge$ $CorrOWD_{up} < 1.6 \wedge$ ($CorrOWD_{up} <= 1.1 \vee CorrOWD_{down} <= 1.05$) $\wedge$ maxCorr$<$0.9
2	no losses $\wedge$ no discards $\wedge$ $CorrOWD_{up} <= 1.2$
3	no losses $\wedge$ no discards $\wedge$ $CorrOWD_{up} <= 1.1 \wedge CorrOWD_{down} <= 1.2$
4	no losses $\wedge$ no discards $\wedge$ $CorrOWD_{up} <= 1.4$
5	no losses $\wedge$ no discards $\wedge$ $CorrOWD_{up} <= 1.1 \wedge CorrOWD_{down} > 1.2$
6	not recent losses $\vee$ (no recent discards $\wedge$ ($1.2 < CorrOWD_{up} <= 1.6 \vee$ previous state not s-))
7	not recent losses $\vee$ not recent discards $\vee$ $1.1 < CorrOWD_{up} <= 1.6$
8	no losses $\wedge$ no discards
9	recent losses $\vee$ (not recent losses $\wedge$ previous state s-) $\vee$ recent discards $\vee$ $CorrOWD_{up} > 1.6 \vee$ ($CorrOWD_{up} > 1.2 \wedge CorrOWD_{down} > 1.05 \wedge$ previous state s-)
10	recent losses $\vee$ recent discards $\vee$ $CorrOWD_{up} > 1.2$
11	recent losses $\vee$ recent discards $\vee$ $CorrOWD_{up} > 1.6$
12	recent losses $\vee$ discards $\vee$ $CorrOWD_{up} > 1.4$
13	((recent losses $\vee$ discards) $\wedge$ (previous state not d $\vee$ time elapsed since last report $< 2 \times$ median RTCP interval)) $\vee$ $CorrOWD_{up} > 2.0$

Moreover, the algorithm makes use of RTCP XR Loss RLE [7], and RTCP XR Discard RLE [35]. Based on these extensions, the algorithm knows exactly which packets were correctly received, which were lost, or discarded. Therefore, it may calculate goodput perceived by the receiver in the last second, which is used for the sending rate calculation. Furthermore, the algorithm divides losses into the

recent ones (status of last five packets in the report), and the overall losses in the reported interval. If there are recent losses, the algorithm decides that congestion takes place, and always reduces of the sending rate. On the other hand, presence of the late losses indicates finished congestion, or a random loss, and therefore results in keeping of the current sending rate.

In addition, if there are no losses and discards reported, we also use RTT/one-way delay values to evaluate network conditions. The algorithm takes statistical approach to these measurements by dividing current RTT/one-way delay value by 40th and 80th percentile of all previously collected values. This method is very similar to the one presented in [32]. RTT/one-way delay value is stored only if the report does not contain any losses. If current RTT/one-way delay value is low, the algorithm decides that there is no congestion at the moment. Otherwise, it classifies the situation as an early congestion indication, or on-going congestion, and takes appropriate strategy to combat the problem. RTT/one-way delay takes different values in different network conditions. Therefore to determine if value of RTT/one-way delay is high/low in a particular situation, division result is compared against a coefficient. If it is higher than the threshold, the high RTT state is determined, otherwise the low state is sensed. Threshold values depend on current algorithm state and were derived empirically in the series of experiments.

Finally, we introduce two procedures additional procedures: the **undershooting procedure** and the **bounceback procedure**. The undershooting procedure is always executed when the algorithm reaches the d state. It calculates the reduced sending rate by subtracting double difference between the current sending rate and the perceived goodput and taking 90% of obtained value. After that it checks if following conditions are also fulfilled: the rate reduction cannot exceed 40% and the new rate must be at least 32kbit/s. For details on the sending rate reduction see the **undershooting procedure** pseudocode in algorithm 1f. Furthermore, in some cases the algorithm orders to disable the rate adaptation for some period of time. This action is required to prevent unnecessary rate reduction when the algorithm predicts that signs of the current congestion will be visible in the future reports. Therefore if such action is needed, the procedure stores value of currently perceived goodput and calculates the rate adaptation disability period, which equals time elapsed since the previous RTCP report multiplied by 1.125. When this period is ended, the algorithm executes the bounceback procedure. This procedure examines content of the most recent RTCP report, and if there are no signs of congestion, increases the rate to either the 90% of goodput stored during the undershooting, or the minumum rate of 32kbit/s. However, if the report shows congestion symptoms, it once again enters the undershooting procedure, but for much shorter period (25% of time elapsed since receiving the last RTCP report). If the repeated undershooting does not succeed, the undershooting procedure is called once more, but this time the period is longer (identical to the initial one), and rate adaptation is not disabled. For details on making sending rate bounceback see the **sending rate bounceback procedure** pseudocode in algorithm 1f.

Algorithm 1 FEC based rate adaptation algorithm (FBRA)

Require: $Goodput, SendingRate, CurrentTime(Now), AdaptationDisabilityTimeout$
Require: $MedianRTT, LastRTCP, CurrentState, FractionLost(FL), OWD_{now}$
Require: $Last5Losses(L5L), FractionDiscard(FD), Last5Discards(L5D)$
Require: $ReportLength(RL)$

$CorrOWD_{down} = \frac{40-percentile_{lossless}(OWD)}{OWD_{now}}$
$CorrOWD_{up} = \frac{80-percentile_{lossless}(OWD)}{OWD_{now}}$
$FECRate = \frac{SendingRate}{FECInterval+1}$
$NewFECRate \leftarrow FECRate$
if $Now < AdaptationDisabilityTimeout$ **then**
    $NewState \leftarrow s-$
else if $Now - LastRTCP < 1.5 * MedianRTT$ **then**
    UNDERSHOOT($disable = true, timeout = Now - LastRTCP$)
    $NewFECRate \leftarrow 0$
else if $CurrentState = s++$ **then**
    STATES++
else if $CurrentState = s+$ **then**
    STATES+
else if $CurrentState = u$ **then**
    STATEU
else if $CurrentState = d$ **then**
    STATED
else
    STATES-
end if
return $(NewSendingRate, NewFECRate)$

Subalgorithm 1a Function used when the FBRA is in the s++ state

```
1: function StateS++
2:     if FL>0 then
3:         if L5L>0 then
4:             Undershoot(disable = true, timeout = Now − LastRTCP)
5:         else
6:             NewState ← s−
7:             NewFECRate ← 0
8:         end if
9:     else if FD>0 then
10:        if L5D>0 then
11:            Undershoot(disable = true, timeout = Now − LastRTCP)
12:        else
13:            NewState ← s−
14:        end if
15:        NewFECRate ← 0
16:    else
17:        if CorrOWD_up > 1.6 then
18:            Undershoot(disable = true, timeout = Now − LastRTCP)
19:            NewFECRate ← 0
20:        else if CorrOWD_up > 1.1 then
21:            NewState ← s−
22:            NewFECRate ← 0
23:        else if CorrOWD_down > 1.2 then
24:            IncrementFECinterval
25:            NewState ← s + +
26:        else
27:            NewState ← u
28:            NewSendingRate ← SendingRate + FECRate
29:            NewFECRate ← 0
30:        end if
31:    end if
32: end function
```

Subalgorithm 1b Function used when the FBRA is in the s+ state

```
1: function StateS+
2:     if FL>0 then
3:         if L5L > 0 ∧ RL>5 then
4:             Undershoot(disable = true, timeout = Now − LastRTCP)
5:         else
6:             NewState ← s−
7:         end if
8:         NewFECRate ← 0
9:     else if FD>0 then
10:        if L5D > 0 ∧ RL > 5 then
11:            Undershoot(disable = true, timeout = Now − LastRTCP)
12:        else
13:            NewState ← s−
14:        end if
15:        NewFECRate ← 0
16:    else if L5L > 0 ∨ CorrOWD_up > 1.6 then
17:        Undershoot(disable = true, timeout = Now − LastRTCP)
18:    else if CorrOWD_up > 1.2 then
19:        Undershoot(disable = false, timeout = Now − LastRTCP)
20:    else
21:        NewState ← s + +
22:    end if
23: end function
```

Subalgorithm 1c Function used when the FBRA is in the u state

```
1: function StateU
2:     if (FL > 0 ∧ L5L > 0) ∨ FD > 0 ∨ CorrOWD_up > 1.4 then
3:         Undershoot(disable = true, timeout = Now − LastRTCP)
4:     else
5:         NewState ← s−
6:         NewFECRate ← 0
7:     end if
8: end function
```

Subalgorithm 1d Function used when the FBRA is in the d state

```
1: function StateD
2:     if (FL>0 ∧ L5L>0) ∨ FD>0 then
3:         if PreviousState = d ∧ CurrentTime − LastRTCPReportTime > 2.0 ∗
   MedianRTT then
4:             NewState ← s−
5:         else
6:             if FD>0 ∧ FL=0 then
7:                 Undershoot(disable = false, timeout = Now − LastRTCP)
8:             else
9:                 Undershoot(disable = true, timeout = Now − LastRTCP)
10:            end if
11:        end if
12:    else if CorrOWD_up > 2.0 then
13:        Undershoot(disable = true, timeout = Now − LastRTCP)
14:    else
15:        NewState ← s−
16:    end if
17:    NewFECRate ← 0
18: end function
```

Subalgorithm 1e Function used when the FBRA is in the s- state

```
1: function StateS-
2:     if FL>0 then
3:         if L5L > 0 ∨ CurrentState = s− then
4:             Undershoot(disable = true, timeout = Now − LastRTCP)
5:         else
6:             NewState ← s−
7:         end if
8:         NewFECRate ← 0
9:     else
10:        if L5D>0 then
11:            Undershoot(disable = true, timeout = Now − LastRTCP)
12:        else
13:            if CorrOWD_up > 1.6 then
14:                Undershoot(disable = true, timeout = Now − LastRTCP)
15:                NewFECRate ← 0
16:            else if CorrOWD_up > 1.1 ∧ CorrOWD_down > 1.05 then
17:                if PreviousState = s− then
18:                    Undershoot(disable = true, timeout = Now − LastRTCP)
19:                else
20:                    NewState ← s−
21:                end if
22:                NewFECRate ← 0
23:            else
24:                if maxCorr < 0.9 then
25:                    TurnOnFEC
26:                else
27:                    if PreviousState = s− then
28:                        TurnOnFEC
29:                    else
30:                        NewState ← s−
31:                        NewFECRate ← 0
32:                    end if
33:                end if
34:            end if
35:        end if
36:    end if
37: end function
```

Subalgorithm 1f Rate adaptation help procedures

Require: $LastSecondGoodput(LSGP), ReportGoodput(RGP), SendingRate(SR)$
Require: $CurrentState, Now$

procedure UNDERSHOOT($disable, timeout$)
  if $LSGP < SR$ **then**
    $Delta \leftarrow SR - LSGP$
    $PickedGoodput \leftarrow LSGP$
  else if RGP<GP **then**
    $Delta \leftarrow abs(SR - RGP)$
    $PickedGoodput \leftarrow RGP$
  else
    $NewSR \leftarrow 0.9 * SR$
    $NewState \leftarrow d$ **return**
  end if
  $possibleRate \leftarrow SR - 2 * Delta$
  if $(SR < 2 * Delta) \vee possibleRate < 0.6 * SR$ **then**
    if $CurrentState = d \vee PickedGoodput < 0.6 * SR$ **then**
      $NewSR \leftarrow max(0.6 * SR, MinRate)$ ▷ MinRate=32kbit/s
      if $NewSR = SR$ **then**
        $NewState \leftarrow s-$
      else
        $NewState \leftarrow d$
      end if
      if disable **then**
        $DesiredGoodput \leftarrow PickedGoodput$
        $AdaptationDisabilityTimeout \leftarrow Now + min(1.125 * timeout, RTCPTIMEOUT)$
      end if
    else
      $NewSR \leftarrow SR$
      $NewState \leftarrow s-$
    end if
  else
    $NewSR \leftarrow max(SR - 2 * Delta, 0.6 * SR)$
    $NewState \leftarrow d$
    if disable **then**
      $DesiredGoodput \leftarrow PickedGoodput$
      $AdaptationDisabilityTimeout \leftarrow Now + min(1.125 * timeout, RTCPTIMEOUT)$
    end if
  end if
  $NewFecRate \leftarrow 0$
end procedure

Subalgorithm 1f Rate adaptation help procedures - continued

39: **procedure** SENDINGRATEBOUNCEBACK(Now)
40: **if** $(L5L = 0) \wedge (L5D = 0) \wedge (CorrOWD_{up} < 1.6)$ **then**
41: $NewSR \leftarrow max(0.9 * DesiredGoodput, MinRate)$
42: **else**
43: **if** RepeatedUndershooting **then**
44: UNDERSHOOT($disable = false, timeout = Now - LastRTCP$)
45: $RepeatedUndershooting \leftarrow false$
46: **else**
47: UNDERSHOOT($disable = true, timeout = 0.25 * (Now - LastRTCP)$)
48: $RepeatedUndershooting \leftarrow true$
49: **end if**
50: **end if**
51: **end procedure**

6.2 Non-FEC based rate adaptation algorithm (N-FBRA)

This algorithm is very similar to the FBRA, except that it does not use FEC. The former algorithm has five states and in two of them FEC is used ($s+$, and $s++$). Consequently, as the FEC part is removed, the state machine of the algorithm is limited to just three states: s, u, d (see figure 22). The figure 22 illustrates the algorithm's state machine, labels are explained in the table 3. It uses the same congestion indicators to evaluate network conditions, as the FBRA. Undershooting mechanism is also identical, as well as, bounceback.

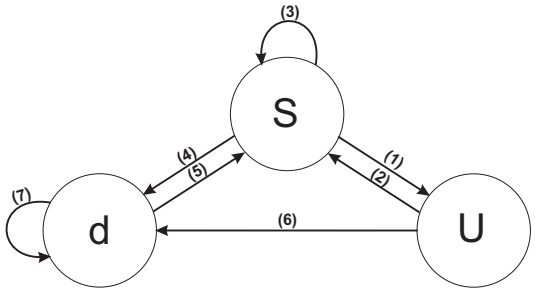


Figure 22: N-FBRA state machine

Table 3: Conditions of N-FBRA state transition illustrated in the figure 22.

No.	Transition conditions
1	no losses $\wedge$ no recent discards $\wedge$ ($CorrOWD_{up} <= 1.1$ $\vee$ $CorrOWD_{down} <= 1.05$) $\wedge$ (maxCorr $< 0.9 \vee$ previous state s)
2	no losses $\wedge$ no discards $\wedge$ $CorrOWD_{up} <= 1.4$
3	not recent losses $\vee$ (no recent discards $\wedge$ $CorrOWD_{up} > 1.1$ $\wedge$ $CorrOWD_{down} > 1.05 \wedge$ previous state not s) $\vee$ (no recent discards $\wedge$ ($CorrOWD_{up} <= 1.1 \vee CorrOWD_{down} <= 1.05$) $\wedge$ previous state not s
4	recent losses $\vee$ recent discards $\vee$ $CorrOWD_{up} > 1.6 \vee$ ($CorrOWD_{up} > 1.4 \wedge CorrOWD_{down} > 1.05 \wedge$ previous state s)
5	(no losses $\wedge$ $CorrOWD_{up} <= 2.0$) $\vee$ ((recent losses $\vee$ discards) $\wedge$ previous state d $\wedge$ time elapsed since last report $>= 2\times$ median RTCP interval
6	recent losses $\vee$ discards $\vee$ $CorrOWD_{up} > 1.4$
7	((recent losses $\vee$ discards) $\wedge$ previous state not d $\wedge$ time elapsed since last report $< 2\times$ median RTCP interval) $\vee$ $CorrOWD_{up} > 2.0$

Similarly to the FBRA version it is possible to go to the d state from every state if congestion is detected. The u state can be entered only from the s state. In this state, the sending rate is increased by amount of hypothetical FEC rate for the current rate (i.e. if the FBRA was used, the new sending rate equals the current sending rate plus the hypothetical FEC rate). The FEC rate is calculated using

exactly the same method, as in the FBRA algorithm (i.e. as the function of current sending rate). In summary, the difference between the N-FBRA and the FBRA is lack of using FEC for bandwidth probing.

N-FBRA pseudo-code is presented in the algorithm 2.

Algorithm 2 Non-FEC based rate adaptation algorithm

Require: $Goodput(GP), SendingRate(SR), Now, AdaptationDisabilityTimeout$
Require: $MedianRTT, LastRTCP, CurrentState, FractionLost(FL), OWD_{now}$
Require: $Last5Losses(L5L), ReportLength(RL), FractionDiscard(FD)$
Require: $Last5Discards(L5D)$
$CorrOWD_{down} = \frac{40-percentile_{lossless}(OWD)}{OWD_{now}}$
$CorrOWD_{up} = \frac{80-percentile_{lossless}(OWD)}{OWD_{now}}$
if $Now < AdaptationDisabilityTimeout$ **then**
  $NewState \leftarrow s-$
else if $Now - LastRTCP < 1.5 * MedianRTT$ **then**
  UNDERSHOOT($disable = true, timeout = Now - LastRTCP$)
else if $CurrentState = u$ **then**
  STATEU
else if $CurrentState = d$ **then**
  STATED
else if $CurrentState = s-$ **then**
  STATES-
end if

Subalgorithm 2a Function used when the N-FBRA is in the u state

1: **function** STATEU
2: **if** $(FL > 0 \wedge L5L > 0) \vee FD > 0 \vee CorrOWD_{up} > 1.4$ **then**
3: UNDERSHOOT($disable = true, timeout = Now - LastRTCP$)
4: **else**
5: $NewState \leftarrow s-$
6: **end if**
7: **end function**

Subalgorithm 2b Function used when the N-FBRA is in the d state

```
1: function StateD
2:     if (FL > 0 ∧ L5L > 0) ∨ FD > 0 then
3:         if PreviousState = d ∧ Now − LastRTCP < 2.0 ∗ MedianRTT then
4:             NewState ← s−
5:         else
6:             if FD > 0 ∧ FL = 0 then
7:                 Undershoot(disable = false, timeout = Now − LastRTCP)
8:             else
9:                 Undershoot(disable = true, timeout = Now − LastRTCP)
10:            end if
11:        end if
12:    else if CorrOWD_up > 2.0 then
13:        Undershoot(disable = true, timeout = Now − LastRTCP)
14:    else
15:        NewState ← s−
16:    end if
17: end function
```

Subalgorithm 2c Function used when the N-FBRA is in the s- state

```
1: function StateS-
2:     if FL > 0 then
3:         if L5L > 0 then
4:             Undershoot(disable = true, timeout = Now − LastRTCP)
5:         else
6:             NewState ← s−
7:         end if
8:     else
9:         if L5D > 0 then
10:            Undershoot(disable = true, timeout = Now − LastRTCP)
11:        else
12:            if CorrOWD_up > 1.6 then
13:                Undershoot(disable = true, timeout = Now − LastRTCP)
14:            else if CorrOWD_up > 1.1 then
15:                if PreviousState == s− then
16:                    Undershoot(disable = true, timeout = Now − LastRTCP)
17:                else
18:                    NewState ← s−
19:                end if
20:            else
21:                if maxCorr < 0.9 then
22:                    FECRate ← GetHypotheticalFECRate
23:                    NewSR ← SR + FECRate
24:                    NewState ← u
25:                else
26:                    if PreviousState = s− then
27:                        FECRate ← GetHypotheticalFECRate
28:                        NewSR ← SR + FECRate
29:                        NewState ← u
30:                    else
31:                        NewState ← s−
32:                    end if
33:                end if
34:            end if
35:        end if
36:    end if
37: end function
```

7 Evaluation

This chapter presents evaluation of algorithms introduced in the previous chapter. Evaluation comprises two parts algorithms simulation in **ns-2** environment [19] and real-world application implementation.

7.1 Simulation Environment

Creation of meaningful Internet simulation is a real challenge due to heterogeneous nature of the network, its immense size, and rapid changes it constantly undergoes. Despite the fact that simulation scenarios are always simplified in comparison to the "real world", they are essential in initial evaluation of new solutions. As simulation takes place in the abstracted model of the world, their results depend on set of input parameters and therefore are repeatable. Results obtained in the "real-world" are always related to a context in which they were conducted, thus they are unrepeatable. As a result, simulation conclusions must be based on multiple runs of simulations with diverse set of input parameters. "Real-world" network represents just the Internet of today, whereas in abstracted world created in the simulator, solutions for the Internet of tomorrow can be tested and evaluated. Moreover in the simulator it is possible to test a solution in a specified set of network conditions evaluating its applicability for a particular environment. In "real-world" such tests are much more expensive, and possibilities of meaningful evaluation in desired scenarios are limited, as control over the Internet is beyond the solution developer.

On the other hand, simulations have many limitations. It is extremely important and also difficult to create simulation scenarios which lead to right conclusions. Past experience shows that it is practically impossible to predict Internet's future evolution (e.g. dominance of HTTP protocol), and thus have guarantee that the simulated protocol/system would work in the real network. This is the problem in designing possible solutions for the Internet of tomorrow which is mentioned in the previous paragraph, as for example we never know what network traffic is applicable for tomorrow. Therefore simulation scenarios must be carefully designed in order to be versatile enough to cope well with heterogeneous network situations [43].

This work was evaluated in the *Network Simulator 2* (ns-2), developed at the University of California, Berkeley in 1997. The whole ns-2 project is open-source and written in OTcl and C++. OTcl is used for writing simulation scripts, and simple task, whereas C++ code performs heavy computations simulating network behaviour. The simulator is the single-threaded and event-driven application. Its heart is the scheduler which orders events execution sequence. The other notable simulator feature is its own internal clock, which is used for distributing events on the axis of time. The scheduler operates by selecting the earliest event, executing it completely, advancing clock until time of new earliest event is reached, and again executing the new event. In case when multiple events are to be executed in the same time, the scheduler serialises them based on first scheduled - first dispatched rule.

7.1.1 Extensions to ns-2 simulator

RTP/RTCP native implementation in *ns-2* is very limited and does not comply with RFC 3550 [4]. Therefore the first step to create the desired simulation environment was proper RTP/RTCP module design. We decided to reuse existing RTP/RTCP code for ns-2 which was developed by Mario Montagud Climent and Fernando Boronat Seguí from Universidad Politécnica de Valencia [44].

We modified the RTP/RTCP code for *ns-2* to be fully compliant with RFC 3550 [4] specification. In addition, we implemented extensions necessary for rate adaptation. Firstly, we updated the code according to RFC 4585 [26] rules. This was mainly concentrated on releasing minimum RTCP report interval *5 seconds rule*, and implementing early feedback module. Secondly, we extended the RTCP module with necessary report headers: the Loss RLE, the Discards RLE, and the RTCP APP header, which carries information about one-way delay. Thirdly, we developed our implementation of the FEC module with ULP support, as it is not part of standard *ns-2* distribution. Finally we implemented three rate adaptation algorithms: *the FEC Based Rate Adaptation (FBRA)*, *the Non-FEC Based Rate Adaptation (N-FBRA)* and *the TFRC for RTP* developed by Ladan Gharai.

7.1.2 Simulation settings

Design of a good simulation scenario is the very challenging task. Thus, we decided to create a fairly simple simulation environment which follows rules suggested by S. Floyd et al. in [43]. This article states that simple scenarios illustrating behaviour of underlying principles are often the best. Furthermore, it emphasise that if conclusions derived in simple scenarios can be also applied in more complex scenarios, it is even better. Finally, it stresses the fact that the simulator must implement functionality which is exactly intended by the researcher. This point of validation can be achieved due to the open-source nature of the *ns-2*, as other researchers can confirm correctness of simulated solutions by re-running them, and modifying some parameters.

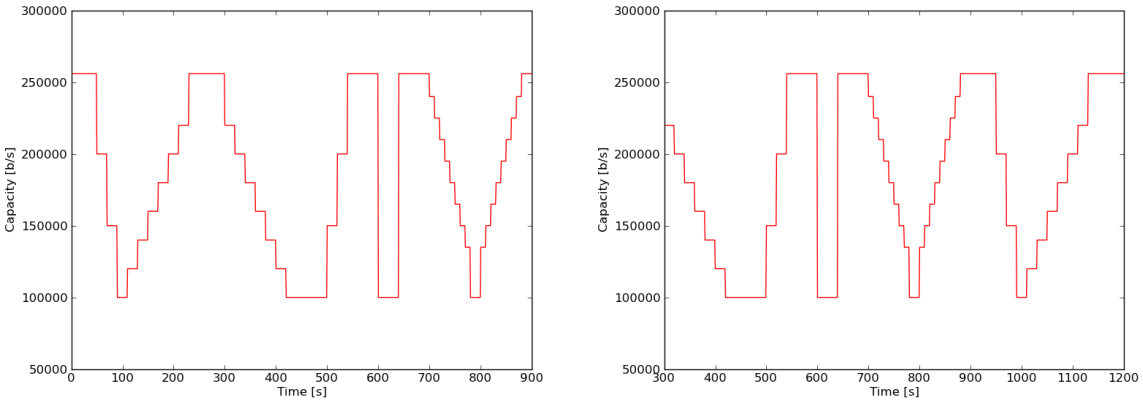
Keeping in mind recommendations of the article [43], and using typical research models presented in S. Floyd et al. in [45], the simple dumbbell topology was chosen as the simulation scenario. As the work concerns unicast congestion control, supporting measurements in forms of variable link characteristics and different router queue types are additionally added [45]. As our research concentrates on rate control in the Internet environment, where bit-error losses are very rare, simple duplex-links not causing such errors are used. For all scenarios capacities of router queues are left default (50 packets), and MTU is set to 1500 bytes. Furthermore, as the simulation results analysis is performed on the packet level, it was decided not to differentiate between different types of video frames, but to represent frames as dummy packets of equal size corresponding to the instantaneous sending rate.

7.1.3 Simulation scenarios

Single RTP flow with variable link capacity

In this simulation scenario the dumbbell topology with two RTP nodes simulating video conversation is used. These nodes are connected to the bottleneck link whose capacity is variable. Access links have 100Mb/s bandwidth available, its delay is 1ms, and Drop Tail queues. The reverse path is symmetrical. The bottleneck link delay is static during the simulation, however, three further scenarios are distinguished, as algorithm behaviour is simulated for link delays of 50ms, 100ms, and 240ms. The link delay values were chosen to evaluate algorithm's behaviour in a wide range of possible delays from intra-continental (50ms) through trans-continental (100ms) to the extreme one (240ms), which was taken from [3]. Furthermore, all experiments are conducted for Drop Tail queues in bottleneck routers as well as for the RED ones. Figure 24 illustrates this network topology.

The bottleneck link capacity varies between 100kb/s and 256kb/s and follow the pattern presented in the figure 23a. It includes many types of capacity changes from very rapid ones (e.g. capacity drops from max to min in the fraction of a second) to smooth ones (e.g. capacity increases slowly and steadily). We believe that such diverse pattern of capacity changes well verifies performance of presented rate control algorithms in different situations. Both video senders begin their session with sending rate of 128kb/s, and delay tolerance is 400ms. Finally, to add some randomisation to obtained results, each simulation is repeated 30 times by making 30 seconds start time shift. As a result, all simulations are run over the channel with identical link characteristics, but they start from different points. Figure 23 illustrates this randomisation.



(a) Variations of bottleneck link capacity

(b) Variations of bottleneck link capacity if flow start is shifted by 300 seconds

Figure 23: Link capacity change pattern

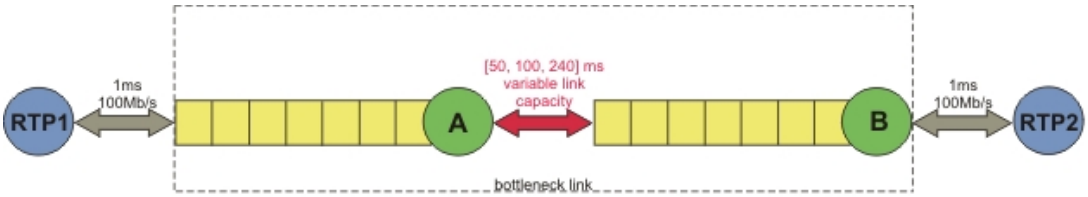


Figure 24: Variable link capacity network settings

Single RTP flow competing for constant link capacity against many TCP flows

The purpose of this simulation experiment is to verify how well performs the rate control algorithms if they have to compete for resources with a number of TCP flows. Similarly to the previous scenario, the dumbbell topology with two RTP nodes simulating video conversation is used. These nodes together with a number of TCP nodes are connected to the bottleneck link. The reverse path is symmetrical. Unlike the previous scenario, the capacity of the bottleneck link is constant and equal to 2Mb/s. Bottleneck link delay is also differentiated for cases of 50ms and 100ms delay, and scenarios are further divided for ones with RED routers and Drop Tail ones. Access links have constant capacities of 100Mb/s and 1ms delay. Similarly to the previous scenario, video senders begin their sessions with initial sending rate of 128kb/s and delay tolerance is set to 400ms. The network topology is illustrated in the figure 25.

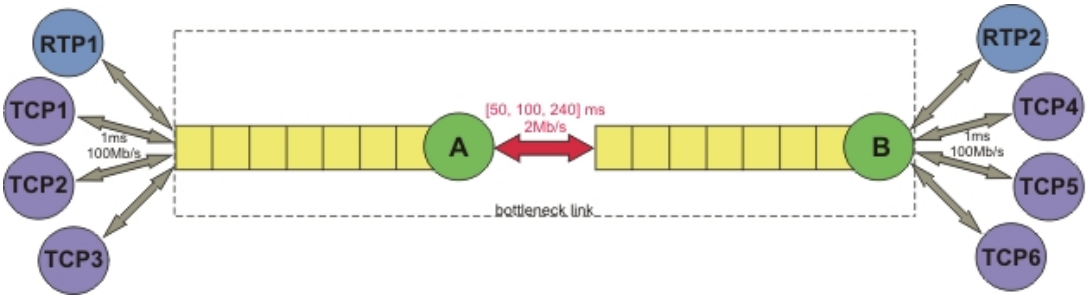


Figure 25: RTP against TCP competition scenario network settings

Two RTP flows competing for constant link capacity against many TCP flows

This simulation scenario is targeted at verification of rate control algorithm performance when 2 video sessions must compete for resources with number of TCP flows, and themselves simultaneously. This scenario is very similar to the previous one, as the only difference between them is existence of two video sessions instead of one. It is illustrated in the figure 26.

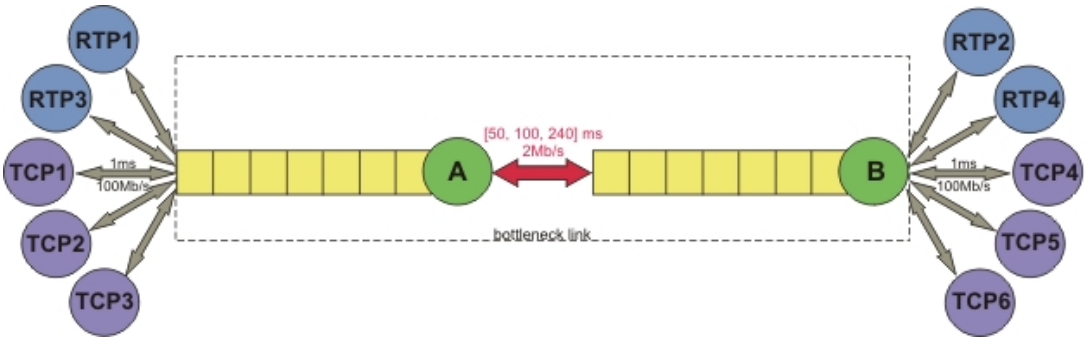


Figure 26: Two RTP sessions against TCP competition scenario network settings

7.2 Simulation results

This part is dedicated to presentation of simulation results. All results are presented as the comparative study evaluating performance of three rate control algorithms. Namely *FEC based rate adaptation algorithm*, its modification not depending on FEC, and *TFRC for RTP*.

7.2.1 Variable link capacity scenario with one RTP flow: Comparative study

Variable link capacity scenario with one RTP flow tests how well rate control algorithms are able to adapt to changes of available bandwidth, which happen according to the pattern illustrated in the figure 23a. Whole scenario analysis is divided into three delay cases (50ms, 100ms, 240ms). Performance of algorithms is rated based on Average Bandwidth Utilisation (ABU), and other general rate adaptation metrics obtained in 30 simulation runs, where each run simulates video session of 900 seconds.

Simulation results for 50ms bottleneck delay:

Simulation results for 50ms delay clearly show that FBRA algorithm outperforms all other ones. Figures 27a, 28a, 29a present time charts for one selected simulation for each algorithm, and table 4 shows general metrics for each one. FBRA algorithm, whose time charts are shown in the figure 27a, has the best bandwidth utilisation of all simulated algorithms. It is able to quickly increase its encoding rate when the link capacity goes up, and respond by reducing rate when there is not enough bandwidth. Furthermore, unlike its version without FEC support (N-FBRA), it avoids TCP-like behaviour when sending rate close to the link capacity limit is approached (in fig. 28a high throughput variation over time is noticeable). FBRA algorithm presents impressive delivery ratio exceeding 99%, as it significantly reduces number of packets discarded at the receiver. Despite this results, usage of this algorithm does not prevent under-flowing, which can be observed in the figure 27b around 90s, 600s, and 800s second of the simulation, from occurrence. Total underflow time for other algorithms is far higher than for the FEC based one. Usage of early feedback

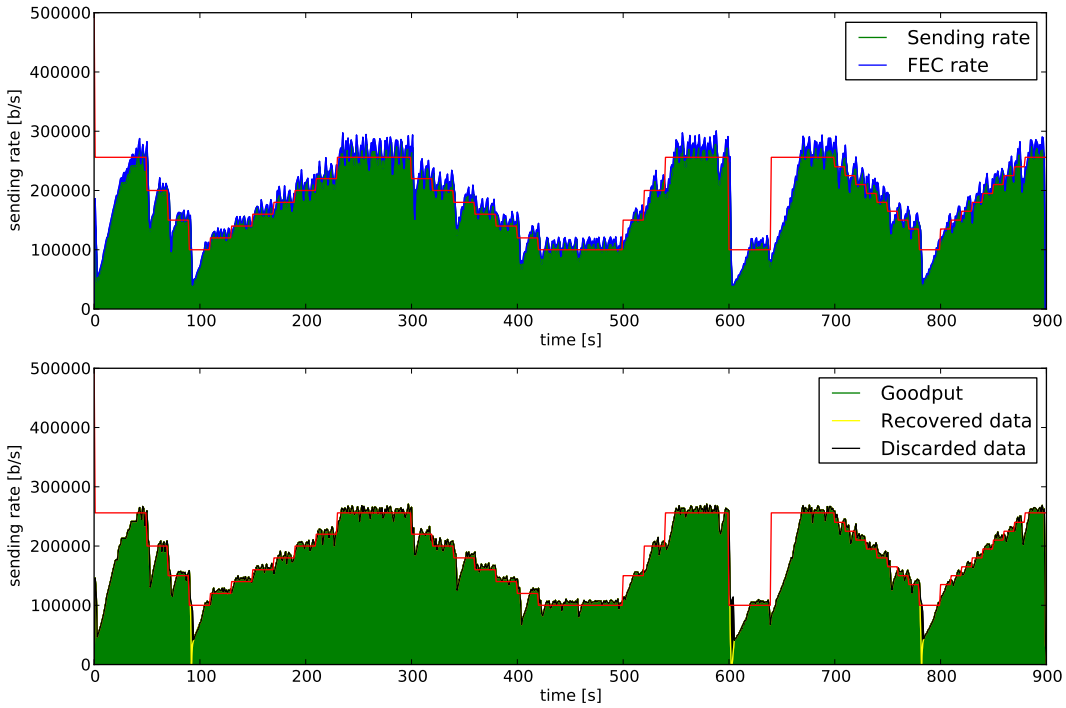
mechanism greatly helps to minimise underflow time, which is far lower for the FEC based algorithm in comparison to its rivals. Figures 27c, 28c, 29c present ABU for all simulated algorithms. Once again the FEC based solution outperforms others by a huge margin with average ABU of 93.92% in comparison to 83.82% achieved by the no FEC based algorithm. TFRC performs really badly in ABU, as its mean is just 24.04%. In all three algorithms average sending rate is almost equal for both sides of conversation, which proves that both flows are fair to each other.

Not much positive can be said about TFRC performance. In the figure 29a TFRC is only able to function satisfactory at the beginning of the simulation run, when the link capacity is constant. If the capacity changes the algorithm is unable to converge to some reasonable value, but oscillates significantly between huge overshooting and undershooting (spikes in the figure). Because of this also average sending rate is much lower than in other considered algorithms. In addition, average underflow time of 220.82 seconds makes this algorithm unsuitable for low delay conversational multimedia sessions. Possible reasons of such a bad performance of TFRC are lack of discarding packets consideration and relying just on RTT values for congestion detection. In this TFRC implementation discards are considered, and treated as losses. But even with this addition to TFRC the algorithm performs very poorly, as more than 6300 packets are discarded by the receiver on average. Network condition evaluation based purely on loss rate and RTT seems to be insufficient for satisfactory rate adaptation.

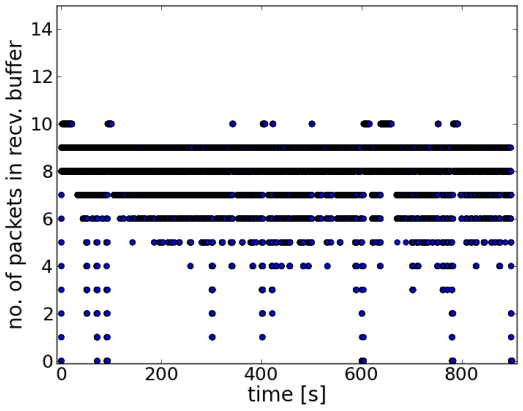
Unfortunately in this scenario we are unable to show that FEC used in the algorithm can be simultaneously applied to rate adaptation and error recovery. As losses are hardly observed (average number of lost packets is just 2.13), it cannot be concluded that FEC packets are unuseful for packet recovery. Such a low number of losses results from the fact that only one flow uses whole link capacity, and router queues are large enough to buffer all incoming packets. However, it can be concluded that the FEC based algorithm is not sensitive to environments where losses are rare, and discards prevail.

Table 4: Overall metrics for 50ms delay in variable link capacity scenario

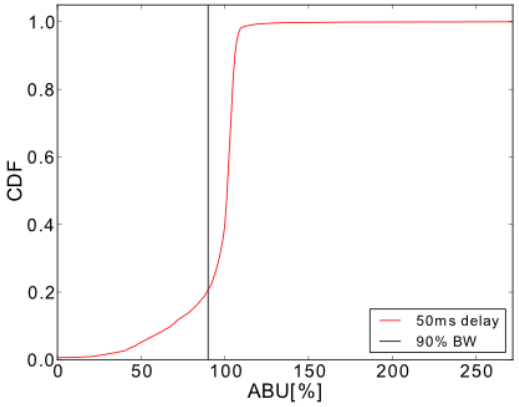
Metric	**FBRA**		**N-FBRA**		**TFRC**	
	avg.	dev	avg.	dev	avg.	dev
Sending rate [kb/s]	178	1.50	161	6.27	92	5.50
Goodput [kb/s]	177	1.59	157	6.98	47	5.22
Delivery ratio [%]	99.39	0.15	97.48	0.58	50.88	2.58
FEC rate [kb/s]	8	0.32	-	-	-	-
No. of lost packets	2.13	2.94	7.53	6.72	22.83	18.75
No. of discarded packets	191.5	40.20	553.83	87.52	6308.3	114.95
No. of recovered packets	0	0	0	0	0	0
No. of packets in recv. buffer	7.69	0.12	7.14	0.13	5.30	0.09
Underflow time [s]	7.90	1.66	23.88	3.61	220.82	4.14



(a) Time chart presenting sending rate, FEC rate, goodput, recovered and discarded data rates

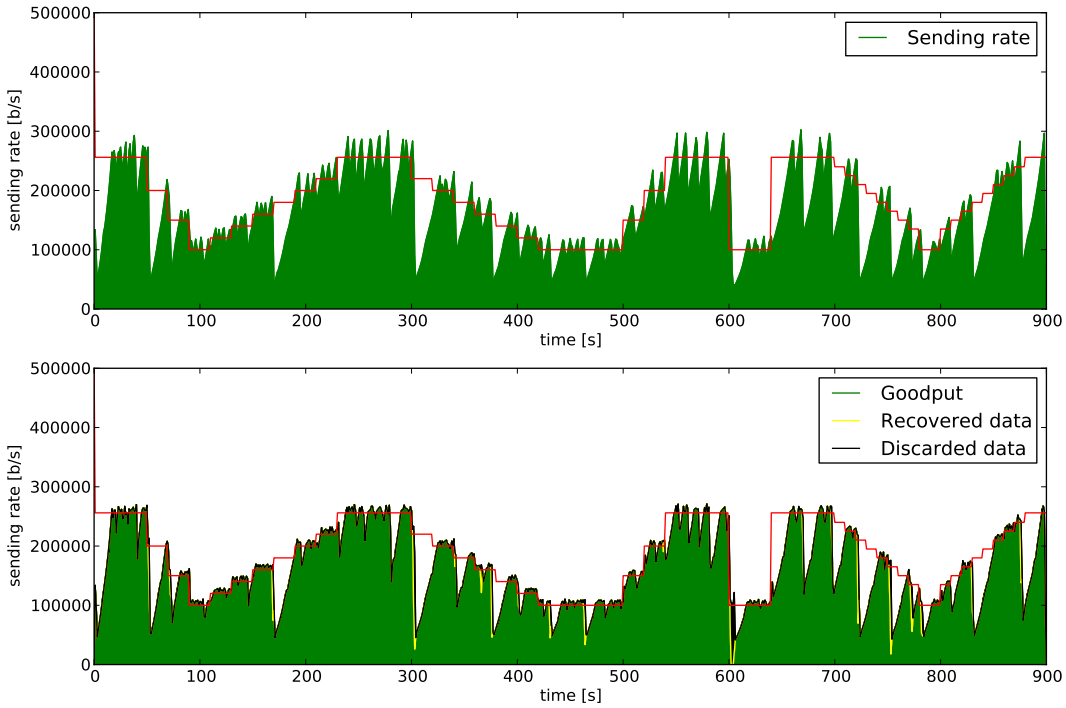


(b) Receiver buffer

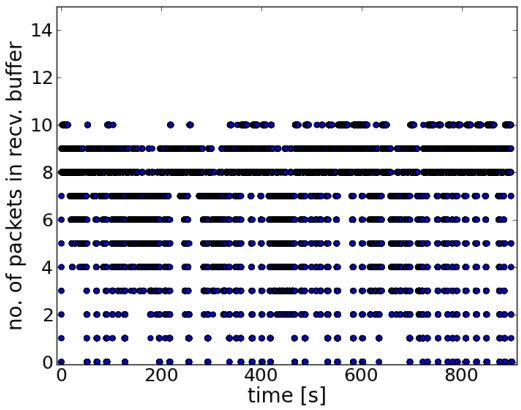


(c) ABU: mean=93.92%

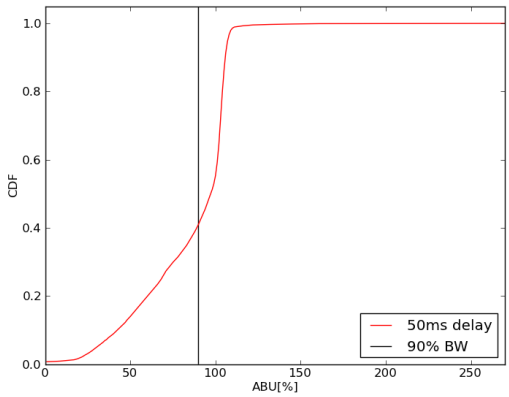
Figure 27: Graphs for the FBRA algorithm for 50ms delay



(a) Time chart presenting sending rate, goodput and discarded data rates

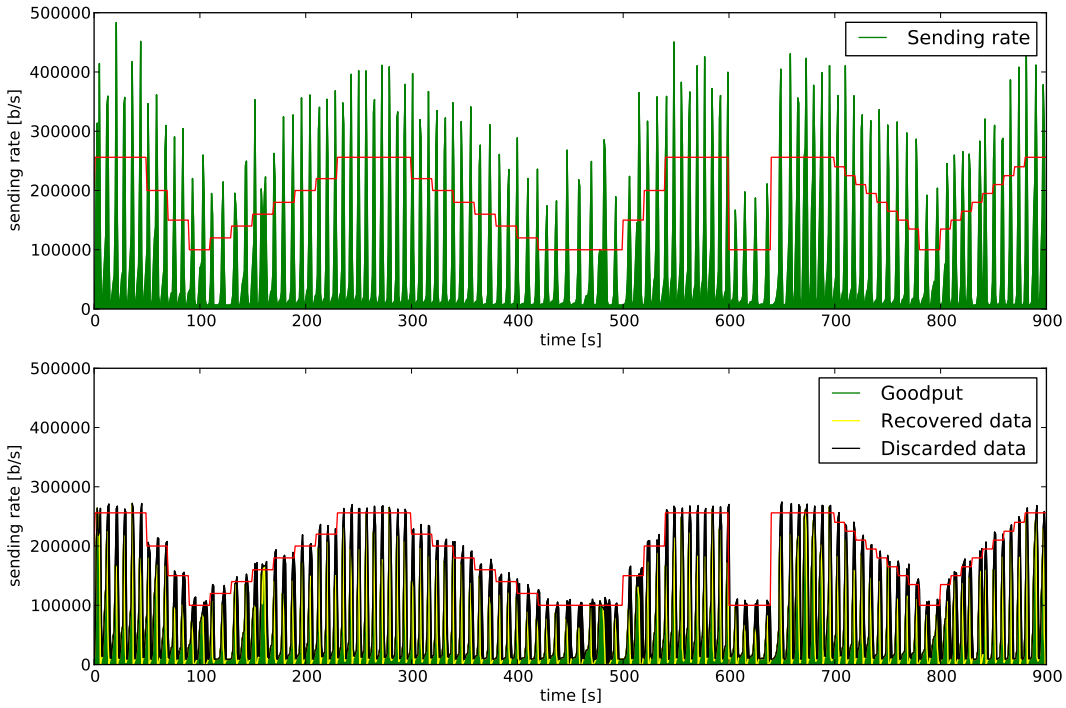


(b) Receiver buffer

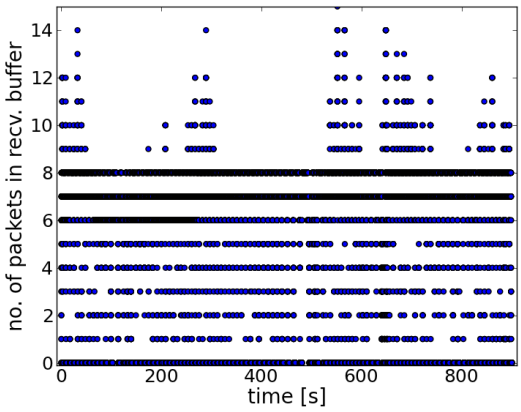


(c) ABU: mean=83.82%

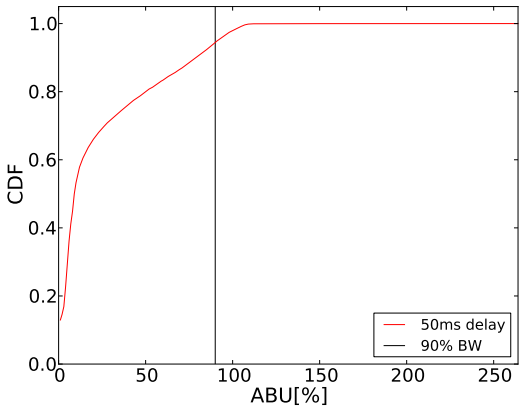
Figure 28: Graphs for the N-FBRA algorithm for 50ms delay



(a) Time chart presenting sending rate, goodput and discarded data rates



(b) Receiver buffer



(c) ABU: mean=24.04%

Figure 29: Graphs for the TFRC algorithm for 50ms delay

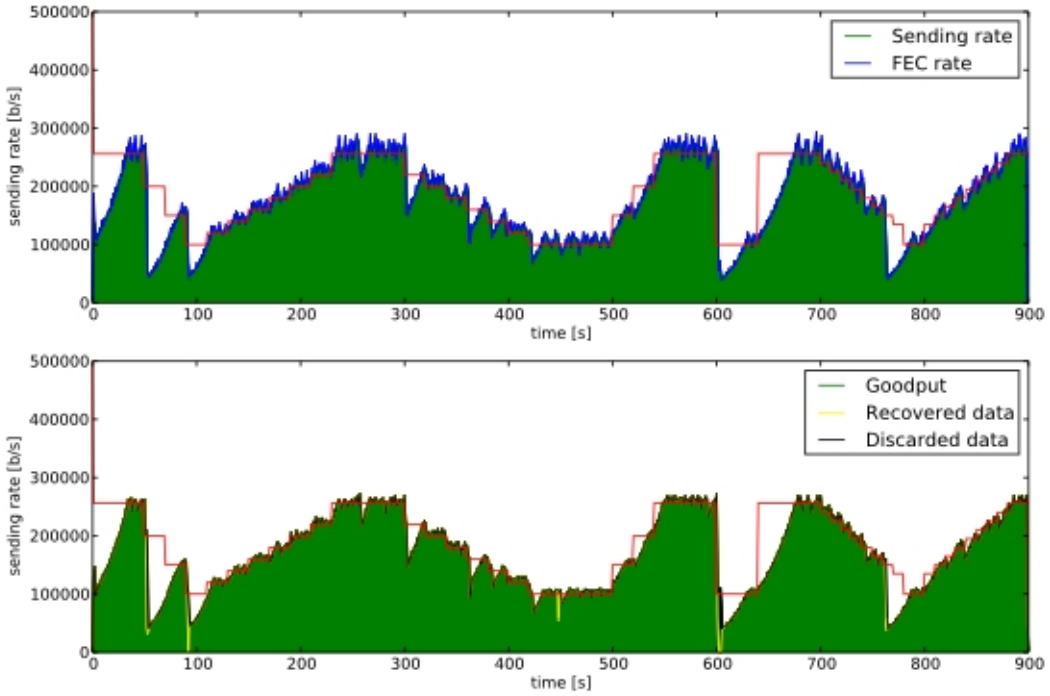
Simulation results for 100ms bottleneck delay:

Simulation results in scenario with 100ms delay are very similar to the previous case. Example of time charts for this scenario are presented in the figures 30a, 31a, 32a, and table 5 shows overall metrics. Again the FBRA algorithm outperforms the other algorithms by huge margin. Its performance is a bit worse than in 50ms delay scenario, but the difference is only marginal, and results from lower delay budget caused by higher physical propagation delay. Delivery ratio is maintained at the impressive level of 99.30%, but underflow metrics are just slightly worse with average underflow time slightly increasing. On the other hand, the N-FBRA algorithm has better results on this scenario than in the previous one. Presumably this performance improvement results from reduced number of states (the algorithm has just 3 states, whereas the FEC based version has 5 ones) leading to quicker reactions which are partially neutralised by the longer delay. Despite a bit better performance, the algorithm still shows signs of TCP-like behaviour, when the sending rate approaches the link capacity limit. The ABU in this scenario is slightly worse with mean value of 89.70% (see figure 30c. The N-FBRA algorithm performs still far worse than its FEC based counterpart, but slightly better than in the 50ms delay scenario. The mean ABU value decreases from 83.82% to 82.15%, but this slump is lower than for the FBRA algorithm. Delivery ratio slightly increases to 97.76% and mean underflow time is reduced to 21.52s, but difference to its FEC supported counterpart is still very significant.

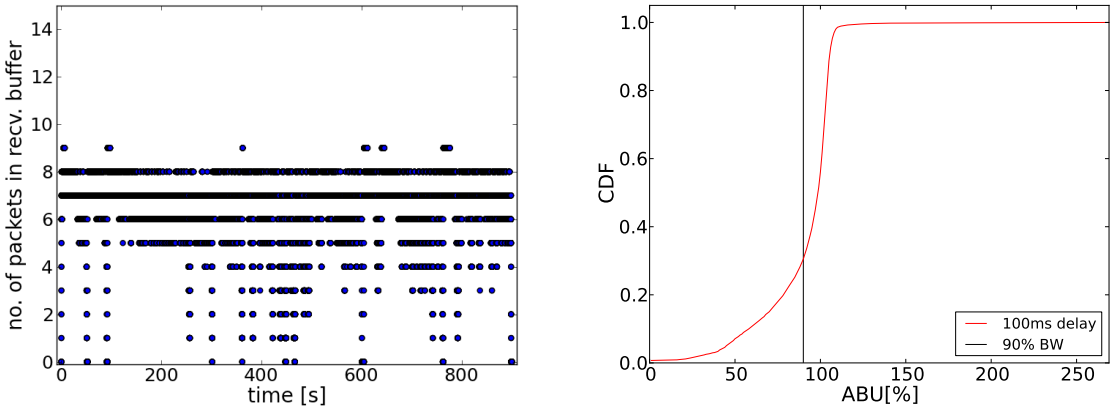
TFRC still performs far worse than FBRA and N-FBRA algorithms. In this scenario it behaves a bit more reliable than in the 50ms case, but delivery ratio of about 50% is unacceptable. Its slight improvement in reliability results from higher physical propagation delay which increases time for the algorithm to see effects of rate adaptation, and therefore reduces a bit drastic changes in the sending rate.

Table 5: Overall metrics for 100ms delay in variable link capacity scenario

Metric	**FBRA**		**N-FBRA**		**TFRC**	
	avg.	dev	avg.	dev	avg.	dev
Sending rate [kb/s]	171	4.20	162	9.35	80	3.25
Goodput [kb/s]	170	4.21	158	10.43	42	2.91
Delivery ratio [%]	99.30	0.18	97.76	0.82	52.64	1.54
FEC rate [kb/s]	5	0.36	-	-	-	-
No. of lost packets	4.73	4.82	4.10	5.37	3.67	7.74
No. of discarded packets	203.07	43.01	501.0	134.77	5361.0	85.06
Recovered packets	0	0	0	0	0	0
No. of packets in recv. buffer	6.88	0.21	6.39	0.12	4.87	0.09
Underflow time [s]	8.54	1.83	21.52	5.82	187.97	3.18



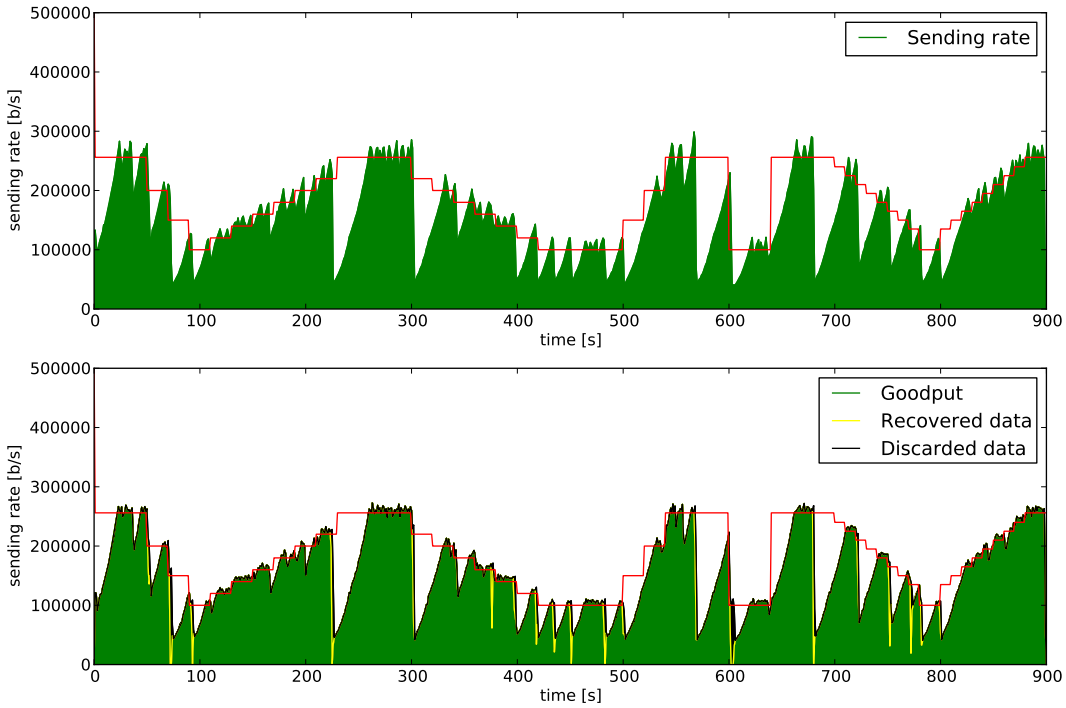
(a) Time chart presenting sending rate, FEC rate, goodput, recovered and discarded data rates



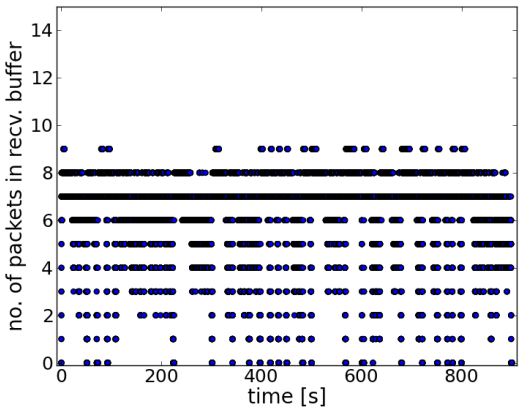
(b) Receiver buffer

(c) ABU: mean=89.70%

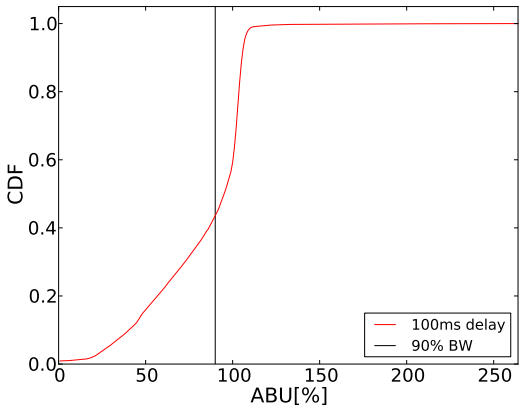
Figure 30: Graphs for the FBRA algorithm for 100ms delay



(a) Time chart presenting sending rate, goodput and discarded data rates

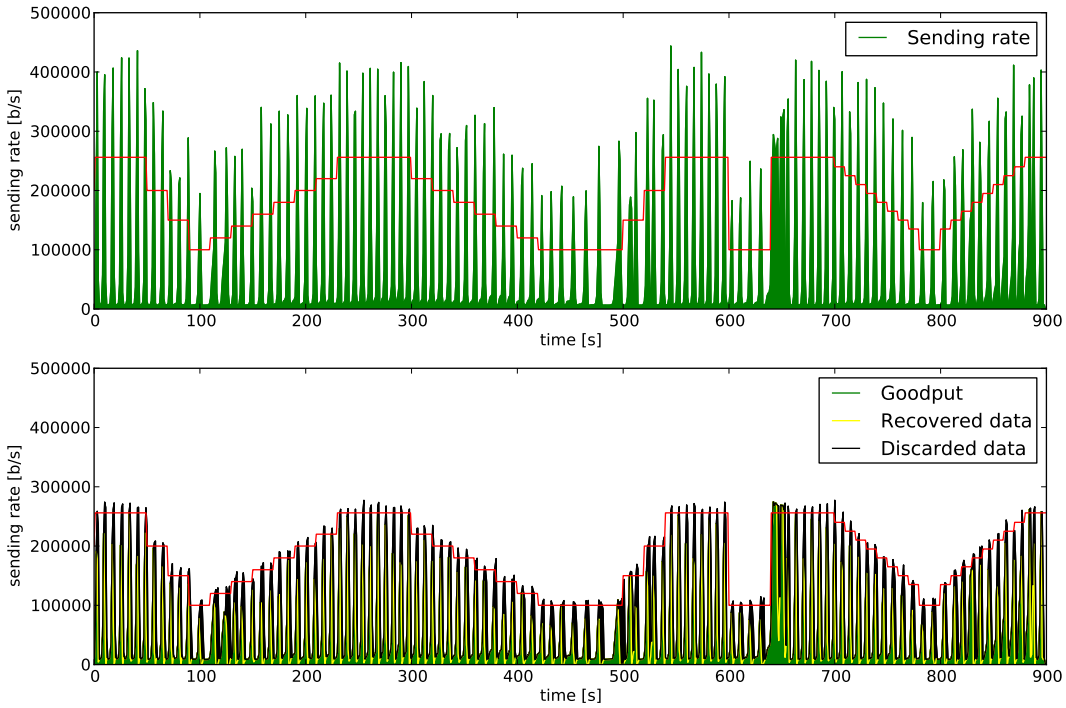


(b) Receiver buffer

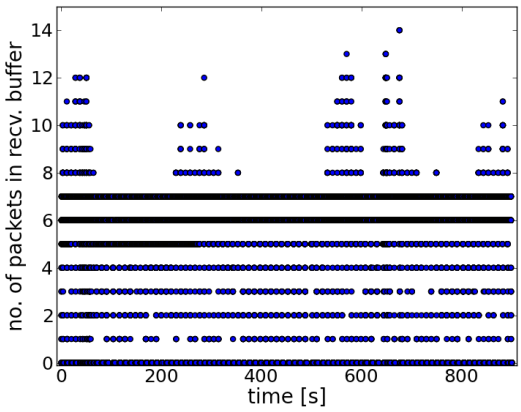


(c) ABU: mean=82.15%

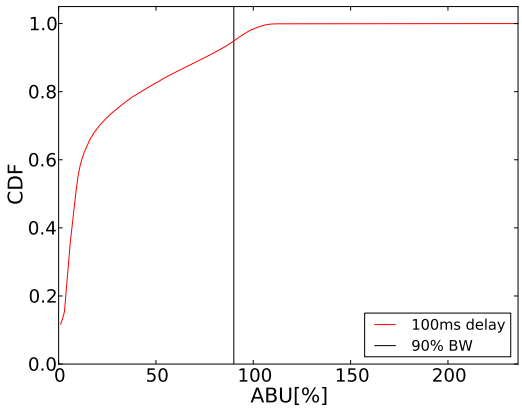
Figure 31: Graphs for the N-FBRA algorithm for 100ms delay



(a) Time chart presenting sending rate, goodput and discarded data rates



(b) Receiver buffer



(c) ABU: mean=22.08%

Figure 32: Graphs for the TFRC algorithm for 100ms delay

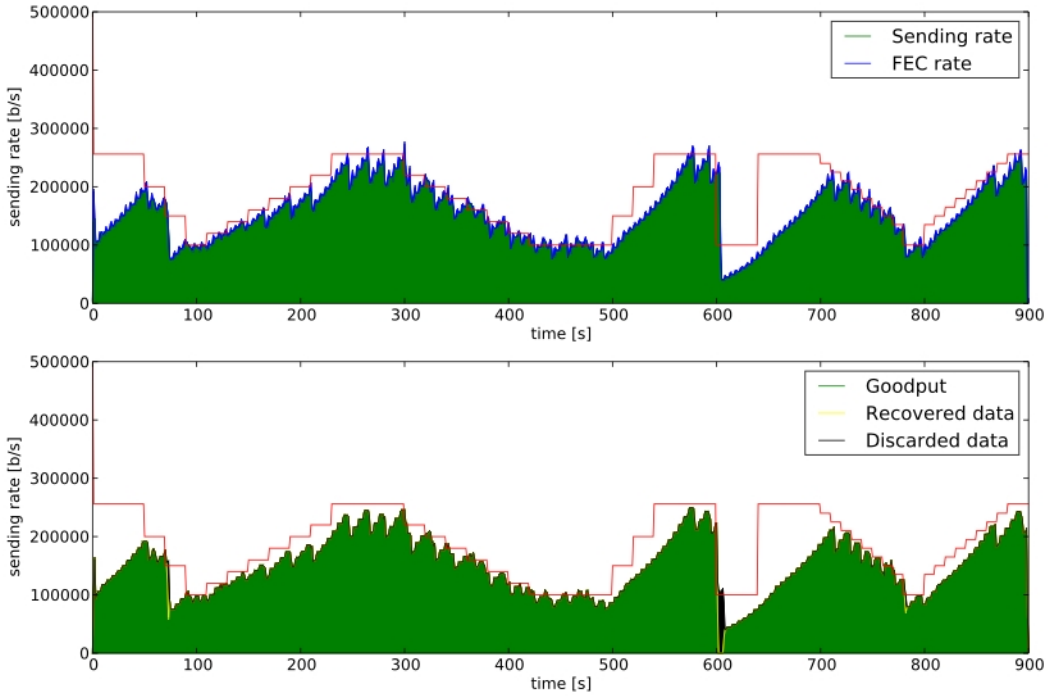
Simulation results for 240ms bottleneck delay:

Contrary to previous scenarios, in the 240ms delay case, there is no clear winner between the FBRA algorithm and its no FEC based counterpart. Example of time charts for this scenario are presented in the figures 33a, 34a, 35a, and table 6 shows overall metrics. The FBRA algorithm suffers a lot from slower feedback response caused by increased delay. Its delivery ratio is still very good, but falls below 99% threshold with 98.9% result. However, the sending rate drop is very significant with the mean value of 149kb/s. This fall is even better illustrated when comparing mean ABUs. In 50ms, and 100ms delay scenarios mean ABUs have values around 90%, whereas in this case it is just 79.01%. On the other hand, the algorithm without FEC support performs significantly better than the in previous cases. Its average sending rate goes up, and surpasses the FBRA algorithm. As a result, ABU metric is also better for this algorithm. The FEC based version still manages to maintain its superiority in terms of reliability. Delivery ratio and underflow metrics stay better in the FBRA algorithm, but their advantage is minimal. Generally it is difficult to unambiguously state which algorithm performs better in this scenario. Possibly PSNR tests, which will be performed in the "real-world" implementation, may answer this question.

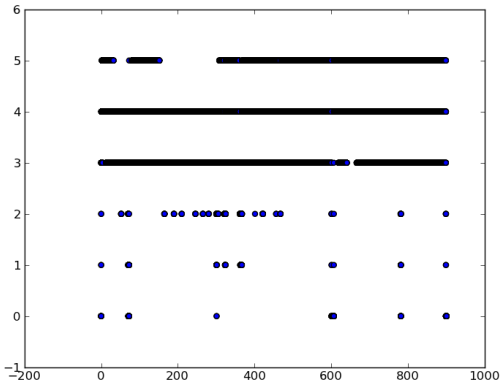
In this scenario TFRC performs performs far better thain the previous ones, but 67.83% of average delivery ratio is still very much below acceptable level. However, it is noticeable that for 240ms delay the algorithm obtains its best results both in terms of the sending rate and the reliability.

Table 6: Overall metrics for 240ms delay in variable link capacity scenario

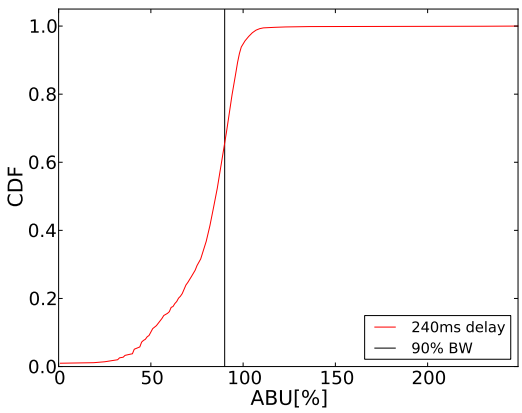
Metric	**FBRA**		**N-FBRA**		**TFRC**	
	avg.	dev	avg.	dev	avg.	dev
Sending rate [kb/s]	149	3.76	160	2.78	97	5.58
Goodput [kb/s]	147.5	3.83	157.75	2.71	66	5.01
Delivery ratio [%]	98.90	0.25	98.62	0.42	67.83	1.52
FEC rate [kb/s]	5.5	0.14	-	-	-	-
No. of lost packets	17.73	8.41	20.5	10.80	9.33	9.42
No. of discarded packets	250.93	52.33	344.67	94.31	4473.1	120.81
Recovered packets	0	0	0	0	0	0
No. of packets in recv. buffer	3.93	0.05	3.85	0.07	2.96	0.08
Underflow time [s]	11.05	2.22	15.09	4.19	162.05	5.24



(a) Time chart presenting sending rate, FEC rate, goodput, recovered and discarded data rates

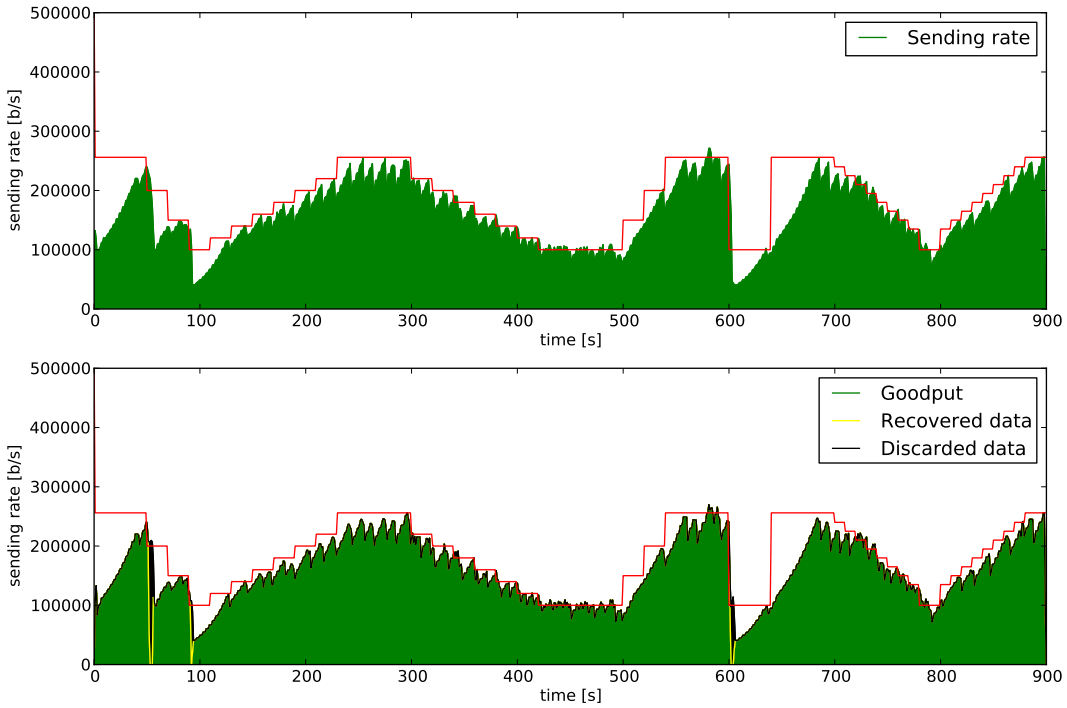


(b) Receiver buffer

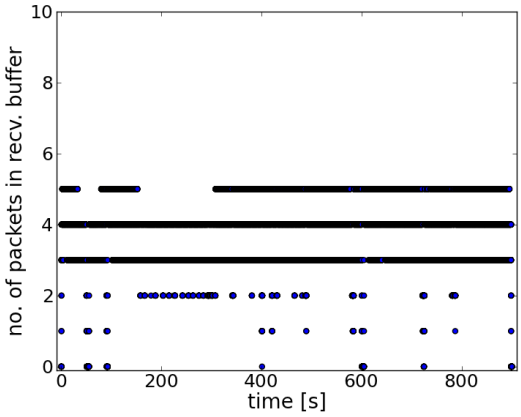


(c) ABU: mean=79.01%

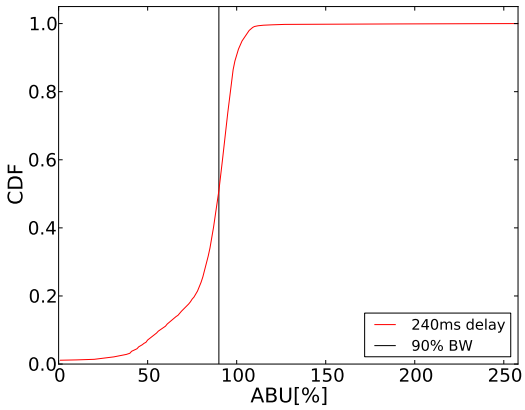
Figure 33: Graphs for the FBRA algorithm for 240ms delay



(a) Time chart presenting sending rate, goodput and discarded data rates

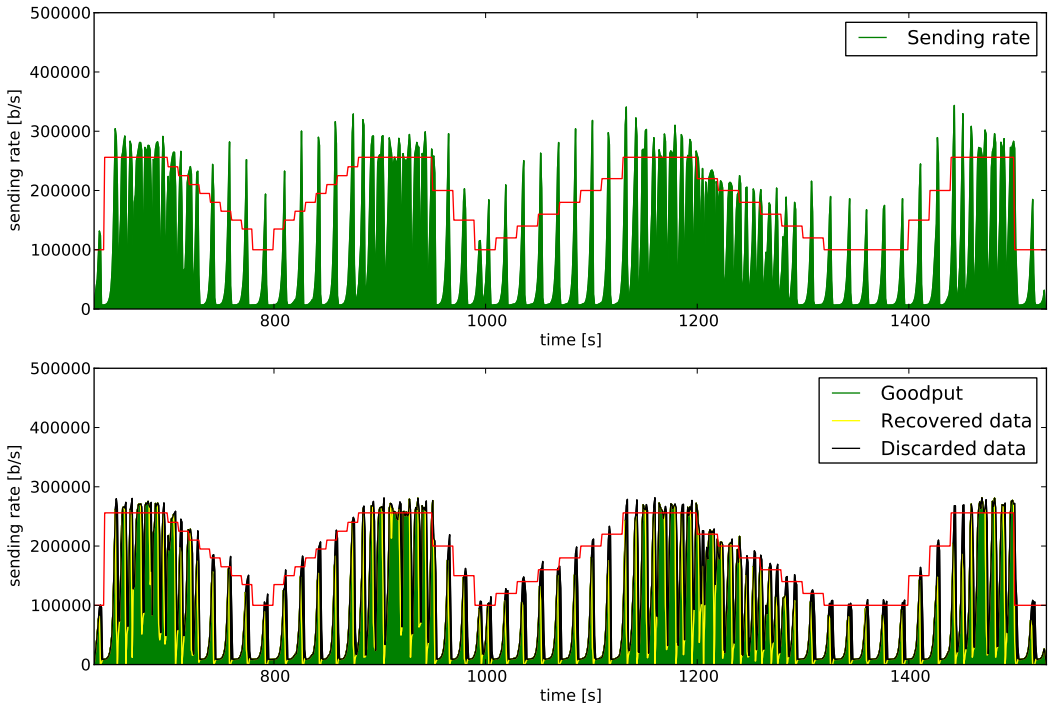


(b) Receiver buffer

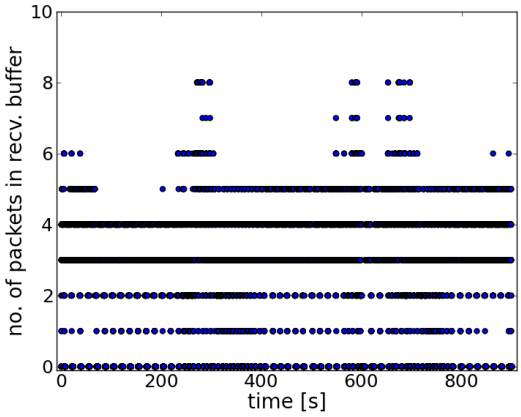


(c) ABU: mean=83.89%

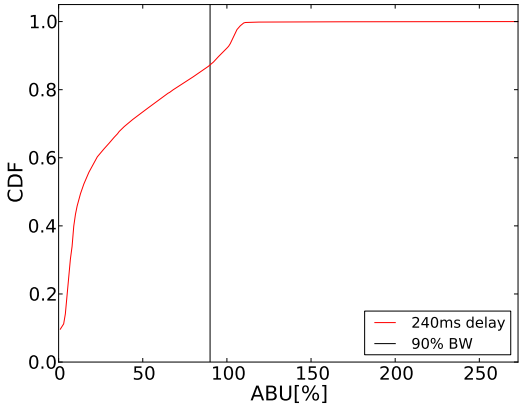
Figure 34: Graphs for the N-FBRA algorithm for 240ms delay



(a) Time chart presenting sending rate, goodput and discarded data rates



(b) Receiver buffer



(c) ABU: mean=31.49%

Figure 35: Graphs for the TFRC algorithm for 240ms delay

Scenario discussion summary:

In three presented above scenarios it was shown that FEC protocol can be effectively used for rate adaptation. The FBRA algorithm outperforms other rate control mechanisms in the two first scenarios, whereas in the third it losses to the N-FBRA algorithm in terms of the sending rate, but still presents better reliability performance. As lack of other flows competing for the link capacity reduces packet losses to the minimum, it is impossible to verify in this scenario hypothesis that FEC can be used for rate adaptation, as well as, for the error recovery. Confirmation of this statement is presented in the next scenario. Furthermore, it is also shown that sending rates obtained by both video session participants are almost equal which proves mutual fairness of the session flows. Finally, it is important to note that the same results were obtained for Drop Tail routers, as well as, for RED ones, which is consistent with RED and Drop Tail behaviour for one flow in the queue.

TFRC performs very badly in all three cases. For higher delay, it is able to maintain reasonable sending rate for a longer period of time when the link capacity is constant. Higher delay causes slower sending rate changes which leads to reduction of number of periods when the rate changes drastically.

7.2.2 Single RTP flow competing for constant link capacity against many TCP flows: Comparative study

Constant link capacity scenario with one RTP flow and multiple TCP flows tests how well rate control algorithms are able to compete for available bandwidth and how fair they are towards other flows. Whole scenario analysis is divided into two delay cases (50ms, 100ms). Furthermore, in each delay case RTP flow competes against one, two, or three TCP sources. TCP traffic for each source is modelled as a sequence of file downloads interleaved with idle periods (on-off traffic). Sizes of files which are downloaded are obtained from the uniform distribution and take values between 100kB and 1.5MB. Lengths of idle periods are drawn from the exponential distribution with the mean value of 10. In all cases simulation time is 900 seconds, bottleneck link capacity is 2Mbit/s and Drop Tail queues in routers are used.

Simulation results for 50ms bottleneck delay:

In this scenario, we can observe packet recoveries in the FBRA simulations. Number of recoveries increases with the number of simultaneous TCP flows, as the more flows are present in the link, the higher is the probability of packet loss. The average number of recovered packets is very low, but number of lost packets is also low. The recovery ratio is between 5-10% of all lost packets. The FBRA algorithm achieves also very high delivery ratio above 99.75%. Analysis of time charts presented in the figure 36 shows that the RTP flow is able to quickly ramp up its sending rate to approach the link capacity limit, and also instantly give up part of its bandwidth when a new TCP flow appears. The question is, if it does not leave too much space for other flows, as its average sending rate is significantly

below TCP flow throughput, and tends to decrease with number of increasing TCP flows. This effect can be explained by the fact that in the FBRA algorithm it is necessary to have an interval of 4 subsequent RTCP reports with good reception statistics to increase the sending rate. In the same time the TCP flow may quicker increase its rate and take the available bandwidth. On the other hand, the RTP flow driven by the N-FBRA algorithm seems to compete much better against TCP flows. In the scenario of competition against just one flow, it achieves average sending rate of around 1.5Mbit/s, whereas TCP gets roughly 540kbit/s. Difference between TCP flows' bandwidth and RTP one decreases, as number of TCP flows increases. When RTP competes against 3 flows, its sending rate is 900kbit/s, while other flows still obtain rates around 540kbit/s. Time charts presenting N-FBRA behaviour are presented in the figure 37. This performance difference between FBRA and N-FBRA results from reduced number of states the N-FBRA algorithm has to go through in order to increase its rate. Similarly to the FBRA, the N-FBRA achieves very impressive packet delivery ratio of around 99%. Flow driven by the N-FBRA algorithm losses far more packets in comparison to the FBRA driven flow. However, number of lost packets is still small for such a high sending rate. Overall metrics for this scenario are presented in tables 7, 8, 9.

Table 7: Overall metrics for 50ms delay in single RTP flow against one TCP flow scenario

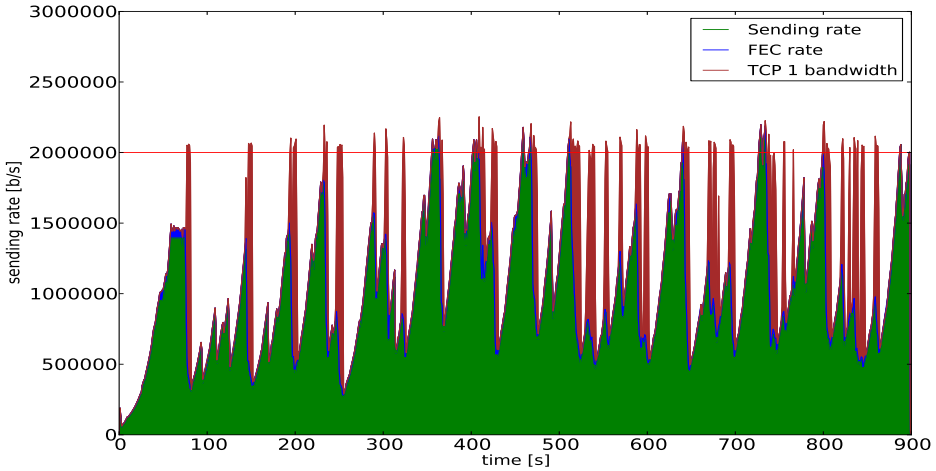
	Metric	**FBRA**		**N-FBRA**	
		avg.	dev	avg.	dev
RTP	**Sending rate [kb/s]**	831.86	99.18	1503.97	114.46
	Goodput [kb/s]	831.16	99.11	1487.22	106.68
	Delivery ratio [%]	99.92	0.02	98.92	0.72
	FEC rate [kb/s]	30.98	3.08	-	-
	No. of lost packets	65.23	16.74	1869.5	1436.0
	No. of discarded packets	0	0	0	0
	Recovered packets	2.9	1.19	0	0
	No. of packets in recv. buffer	27.63	2.65	37.51	1.69
	Underflow time [s]	0	0	0.0006	0.004
TCP	**Throughput in "on" state [kb/s]**	952.61	134.42	541.34	88.81
	"on" time [s]	258.75	27.56	345.42	38.75
	Percentage of fair share	95.26	13.44	54.13	8.88

Table 8: Overall metrics for 50ms delay in single RTP flow against two TCP flows scenario

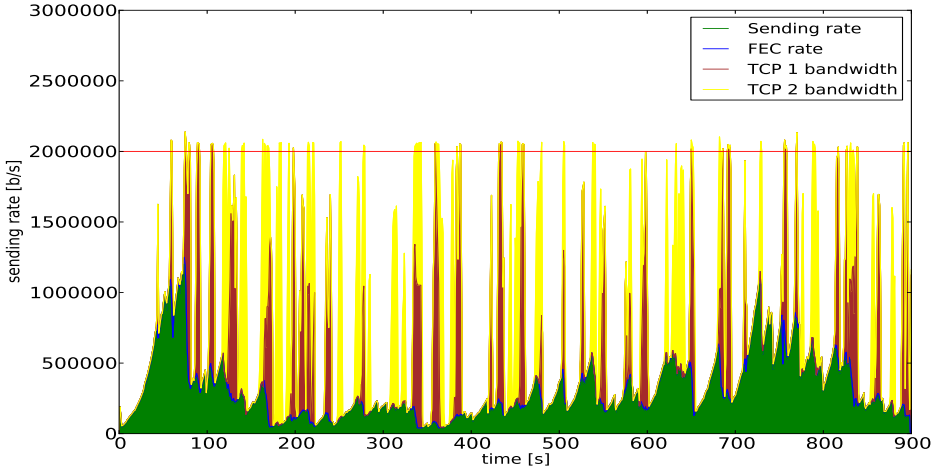
	Metric	FBRA		N-FBRA	
		avg.	dev	avg.	dev
RTP	**Sending rate [kb/s]**	284.35	80.57	1149.93	139.55
	Goodput [kb/s]	283.92	80.39	1140.44	134.14
	Delivery ratio [%]	99.85	0.06	99.21	0.41
	FEC rate [kb/s]	12.33	3.18	-	-
	No. of lost packets	51.67	25.83	941.03	643.27
	No. of discarded packets	0	0	0	0
	Recovered packets	4.27	2.11	0	0
	No. of packets in recv. buffer	12.67	2.07	31.11	2.01
	Underflow time [s]	0.048	0.06	0.006	0.01
TCP	**Throughput in "on" state [kb/s]**	955.95	163.42	581.16	99.12
	"on" time [s]	257.31	38.29	363.69	47.09
	Percentage of fair share	143.39	24.51	87.17	14.87
TCP	**Throughput in "on" state [kb/s]**	957.75	117.04	560.51	71.22
	"on" time [s]	248.10	26.14	324.72	46.48
	Percentage of fair share	143.66	17.56	84.08	10.68

Table 9: Overall metrics for 50ms delay in single RTP flow against three TCP flows scenario

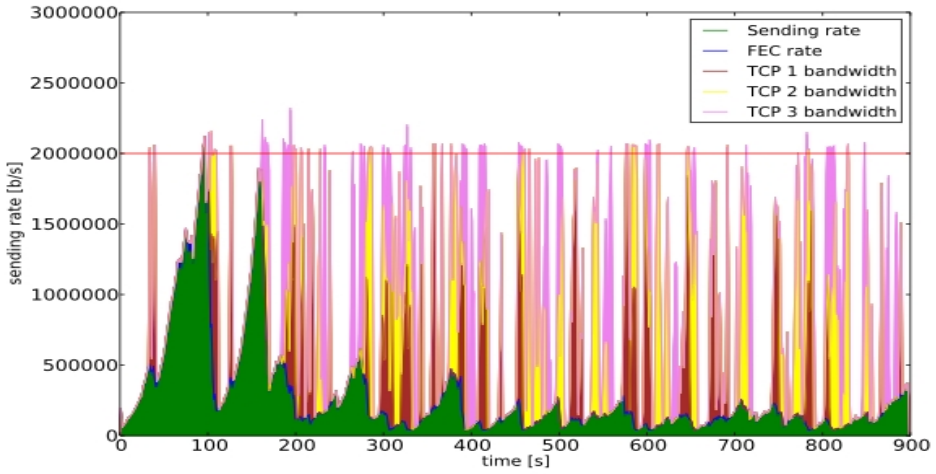
	Metric	FBRA		N-FBRA	
		avg.	dev	avg.	dev
RTP	**Sending rate [kb/s]**	184.89	30.91	898.55	138.54
	Goodput [kb/s]	184.44	30.88	890.19	134.93
	Delivery ratio [%]	99.75	0.06	99.11	0.29
	FEC rate [kb/s]	8.32	1.17	-	-
	No. of lost packets	79.0	15.27	816.03	396.53
	No. of discarded packets	0	0	0	0
	Recovered packets	8.27	2.50	0	0
	No. of packets in recv. buffer	10.52	0.75	25.30	2.49
	Underflow time [s]	0.088	0.07	0.034	0.02
TCP	**Throughput in "on" state [kb/s]**	839.94	118.71	560.75	82.32
	"on" time [s]	275.97	32.78	355.0	40.91
	Percentage of fair share	167.99	23.74	112.15	16.46
TCP	**Throughput in "on" state [kb/s]**	823.80	90.94	547.60	71.22
	"on" time [s]	260.57	26.93	348.60	44.18
	Percentage of fair share	164.76	18.19	109.52	14.24
TCP	**Throughput in "on" state [kb/s]**	819.97	114.75	527.82	77.63
	"on" time [s]	235.10	31.60	317.73	39.16
	Percentage of fair share	163.99	22.95	105.56	15.53



(a) Time chart presenting competition between RTP and TCP flow

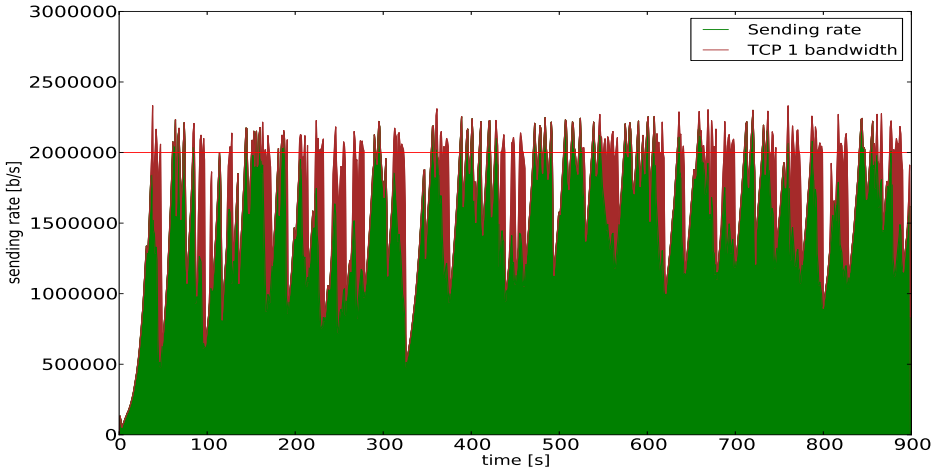


(b) Time chart presenting competition between RTP and 2 TCP flows

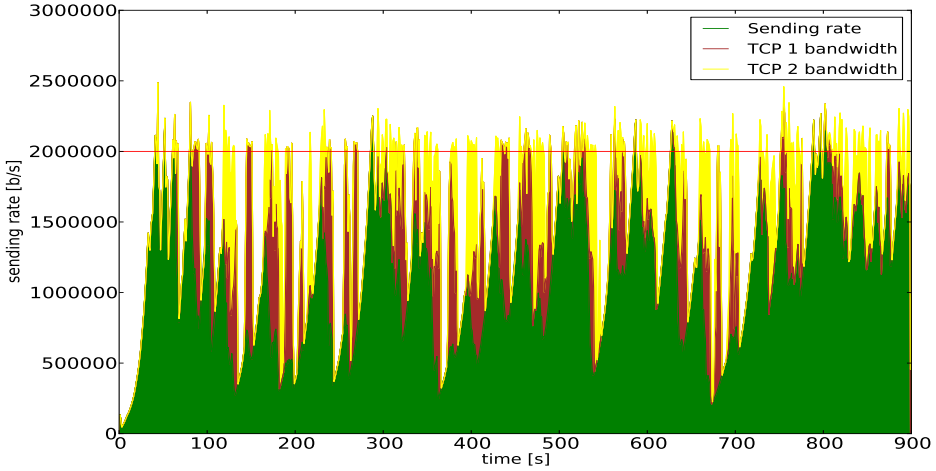


(c) Time chart presenting competition between RTP and 3 TCP flows

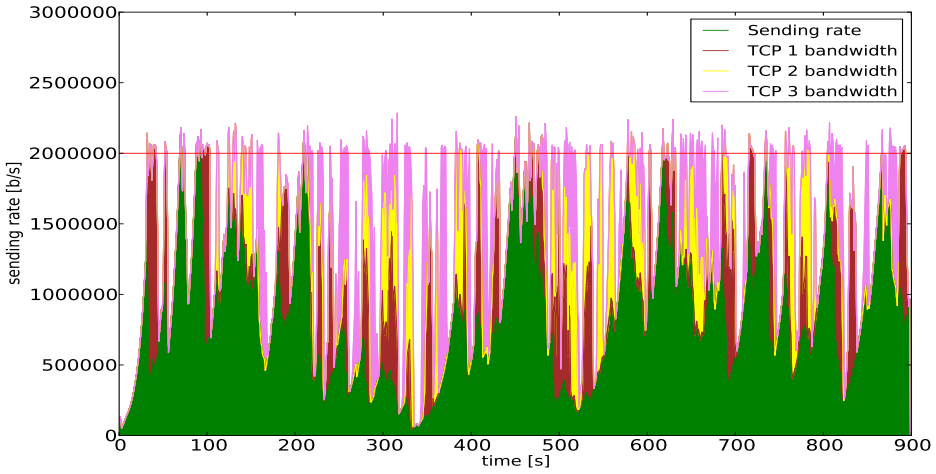
Figure 36: RTP vs. TCP competition in 50ms scenario with FBRA used



(a) Time chart presenting competition between RTP and TCP flow



(b) Time chart presenting competition between RTP and 2 TCP flows



(c) Time chart presenting competition between RTP and 3 TCP flows

Figure 37: RTP vs. TCP competition in 50ms scenario with N-FBRA used

Simulation results for 100ms bottleneck delay:

For 100ms delay scenario, it can be observed that the average sending rate increases when the FBRA and N-FBRA algorithms are used for the RTP flow, whereas TCP flows achieve lower throughput than in 50ms delay scenario. Worse TCP performance results from higher network latency limiting rate at which TCP segment acknowledgements arrive at the sender. To sum up, the higher network latency is, the lower is the total number of bytes sent by the TCP flow in the time unit. On the other hand, the RTP flow is not limited by any window, and therefore it can easier expand its sending rate by pushing back TCP flows. Simultaneously increase of the RTP sending rate does not kill TCP transfers, as they are still able to achieve average throughput of around 500kbit/s regardless 1, 2, or 3 TCP flows compete in the same time. Packet recoveries are still observed when the FBRA algorithm is used, but their average number is lower, as higher latency reduces delay budget leaving less time for possible recovery of the lost packet. Average packet delivery ratio maintains above 99% for FBRA and N-FBRA and underflow time is in all cases below 2 seconds. In this scenario packet discards begin to appear, but they don't affect results very much, as they are very low in all cases. Again number of lost packets for the N-FBRA driven flow is much higher than for the FBRA one, but this difference results from much higher sending rate achieved by the former algorithm. Example of time charts for this scenario are presented in the figures 38, 39, and tables 10, 11, 12 show overall metrics.

Table 10: Overall metrics for 100ms delay in single RTP flow against one TCP flow scenario

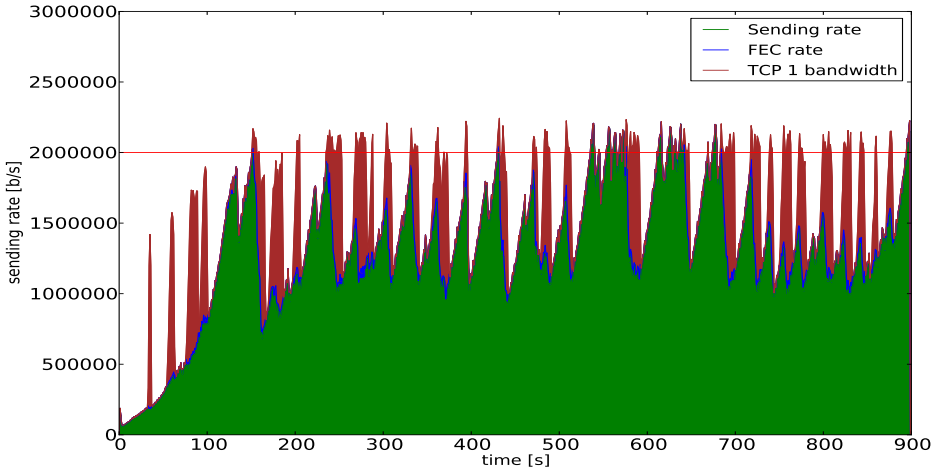
	Metric	**FBRA**		**N-FBRA**	
		avg.	dev	avg.	dev
RTP	**Sending rate [kb/s]**	1308.48	62.58	1528.87	54.86
	Goodput [kb/s]	1306.28	62.50	1519.06	50.78
	Delivery ratio [%]	99.83	0.04	99.37	0.35
	FEC rate [kb/s]	37.76	1.39	-	-
	No. of lost packets	170.43	45.70	976.8	579.61
	No. of discarded packets	32.70	6.56	57.73	30.06
	Recovered packets	4.7	2.44	0	0
	No. of packets in recv. buffer	33.40	1.26	33.14	1.12
	Underflow time [s]	0.45	0.11	0.47	0.19
TCP	**Throughput in "on" state [kb/s]**	582.70	76.08	445.89	47.71
	"on" time [s]	344.37	37.94	391.80	37.33
	Percentage of fair share	58.27	7.61	44.59	4.77

Table 11: Overall metrics for 100ms delay in single RTP flow against two TCP flows scenario

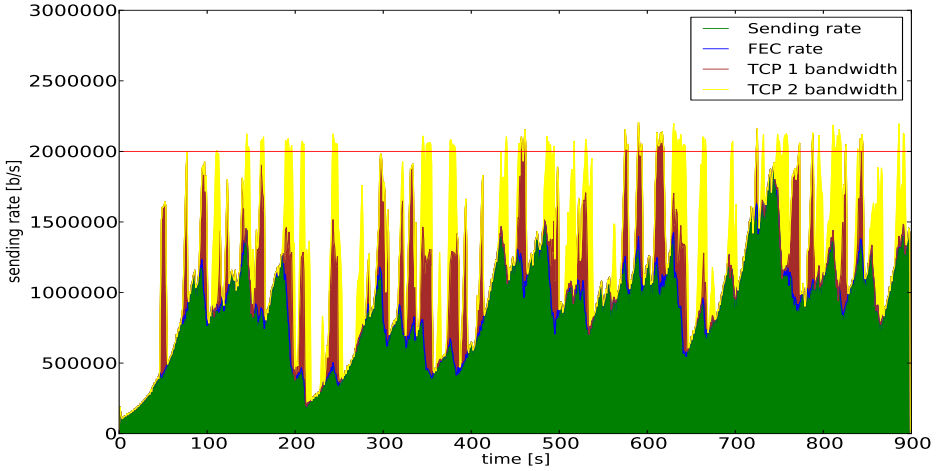
	Metric	FBRA		N-FBRA	
		avg.	dev	avg.	dev
RTP	**Sending rate [kb/s]**	840.01	73.24	1312.46	63.10
	Goodput [kb/s]	837.42	72.99	1301.99	60.91
	Delivery ratio [%]	99.69	0.06	99.21	0.25
	FEC rate [kb/s]	26.52	1.80	-	-
	No. of lost packets	210.67	53.25	963.10	373.47
	No. of discarded packets	34.20	5.58	44.7	15.01
	Recovered packets	5.67	2.89	0	0
	No. of packets in recv. buffer	23.17	1.62	28.75	0.66
	Underflow time [s]	1.01	0.20	0.84	0.21
TCP	**Throughput in "on" state [kb/s]**	581.28	79.63	416.78	49.05
	"on" time [s]	355.40	38.00	416.03	35.20
	Percentage of fair share	87.19	11.94	62.52	7.36
TCP	**Throughput in "on" state [kb/s]**	564.09	74.11	405.00	48.33
	"on" time [s]	332.60	35.23	393.53	44.42
	Percentage of fair share	84.61	11.11	60.75	7.25

Table 12: Overall metrics for 100ms delay in single RTP flow against three TCP flows scenario

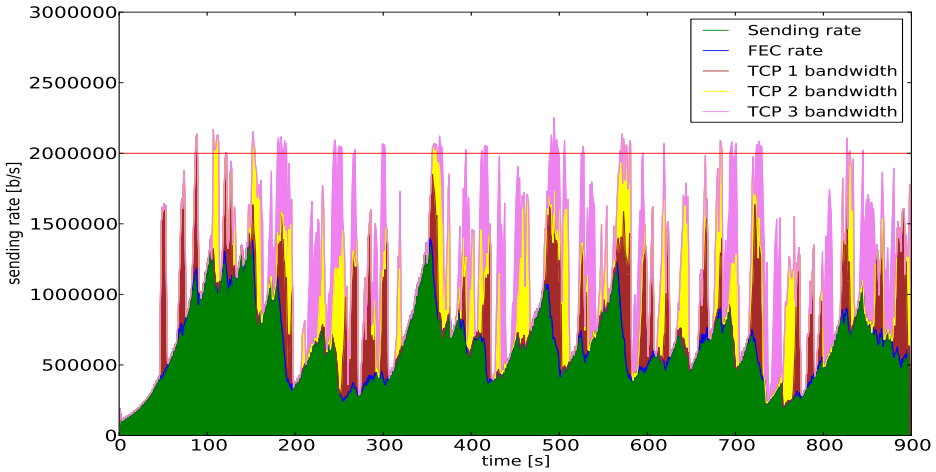
	Metric	FBRA		N-FBRA	
		avg.	dev	avg.	dev
RTP	**Sending rate [kb/s]**	485.25	96.71	1074.14	108.07
	Goodput [kb/s]	483.14	96.09	1064.36	105.10
	Delivery ratio [%]	99.57	0.10	99.11	0.27
	FEC rate [kb/s]	17.19	2.95	-	-
	No. of lost packets	193.40	68.89	884.83	336.06
	No. of discarded packets	27.40	6.75	40.27	11.61
	Recovered packets	6.80	3.07	0	0
	No. of packets in recv. buffer	14.95	2.28	24.73	1.60
	Underflow time [s]	1.64	0.27	1.26	0.20
TCP	**Throughput in "on" state [kb/s]**	567.09	75.92	408.34	60.61
	"on" time [s]	349.13	44.92	411.97	53.01
	Percentage of fair share	113.42	15.18	81.67	12.12
TCP	**Throughput in "on" state [kb/s]**	550.85	82.30	401.03	46.54
	"on" time [s]	334.17	45.57	401.13	39.62
	Percentage of fair share	110.17	16.46	80.21	9.31
TCP	**Throughput in "on" state [kb/s]**	546.22	71.00	396.99	48.61
	"on" time [s]	308.07	34.92	376.67	41.40
	Percentage of fair share	109.24	14.20	79.40	9.72



(a) Time chart presenting competition between RTP and TCP flow

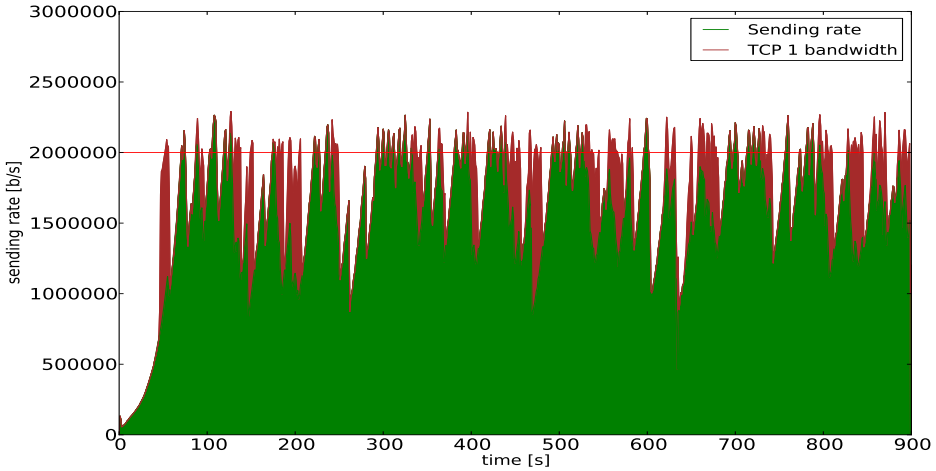


(b) Time chart presenting competition between RTP and 2 TCP flows

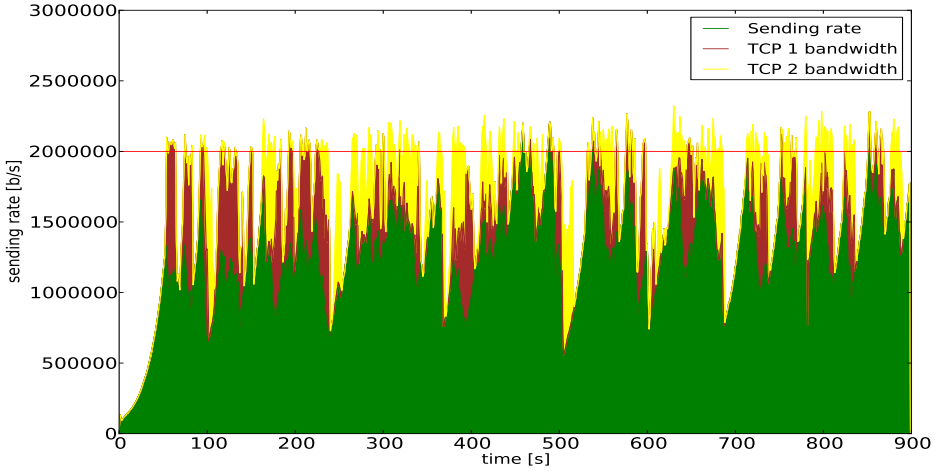


(c) Time chart presenting competition between RTP and 3 TCP flows

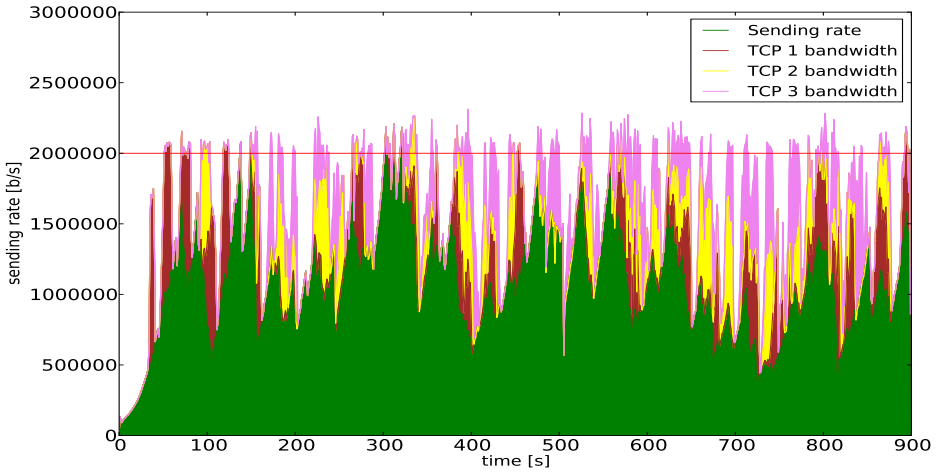
Figure 38: RTP vs. TCP competition in 100ms scenario with FBRA used



(a) Time chart presenting competition between RTP and TCP flow



(b) Time chart presenting competition between RTP and 2 TCP flows



(c) Time chart presenting competition between RTP and 3 TCP flows

Figure 39: RTP vs. TCP competition in 100ms scenario with N-FBRA used

Scenario discussion summary:

Two presented above scenarios show that the FBRA and N-FBRA rate adaptation algorithms can compete successfully for network bandwidth against multiple TCP flows. Furthermore, it was also shown that FEC in the FBRA algorithm is not only able to be used for rate adaptation, but that it can also recover some number of lost packets. Comparison of these two algorithms performance indicates that the N-FBRA algorithm competes much better against TCP flows than the FBRA one. The reliability of the N-FBRA algorithm is slightly lower than its counterpart, but its excellent sending rate performance can compensate for this loss.

Having analysed results, it is important to evaluate contribution of FEC to rate adaptation. It is undeniable that the FBRA algorithm achieves excellent results in terms of reliability. However, its sending rate performance differs widely from its N-FBRA counterpart, which also achieves very good reliability results. There is no simple answer to the question if usage of FEC for rate adaptation can be justified. Probably there are scenarios (e.g. lossy network links) where the N-FBRA driven flows achieve far worse reliability results. Furthermore, as it was shown in the variable link capacity scenario results, the FBRA algorithm can outperform its counterpart. Therefore it is impossible to definitely state that using FEC for rate adaptation is pointless. Perhaps it could be possible to create a hybrid solution in which the algorithm can switch between FBRA and N-FBRA algorithms depending on the network situation.

7.2.3 Two RTP flows competing for constant link capacity against many TCP flows: Comparative study

Constant link capacity scenario with two RTP flows and multiple TCP flows tests how well rate adaptation algorithms are able to compete for available network bandwidth against elastic flows and another multimedia flow. Whole scenario analysis is divided into two delay cases (50ms and 100ms). Additionally in each delay case two RTP flows compete against one, two, or three TCP flows. TCP traffic is modelled as a sequence of file downloads interleaved with idle periods (on-off traffic type). File sizes are drawn from the uniform distribution within the range of 100kB and 1.5MB. Idle period lengths are obtained from the exponential distribution with the mean value of 10. In all delay cases simulation time is set to 900 seconds, bottleneck capacity is 2Mbit/s and Drop Tail queues in routers are used.

Simulation results for 50ms bottleneck delay:

In the 50ms delay scenario, again N-FBRA driven RTP flows obtain much higher average goodput than RTP flows driven by the FBRA algorithm. Sending rates of RTP flow 1 and 2 differ significantly when the FBRA algorithm is used, but the rates for both participants of the session in the same flow are comparable, which proves that the FBRA algorithm assures fairness for both sides of the session. Difference between flow sending rates decreases, as number of background TCP flows increases.

In comparison to the previous scenario, it can be stated that TCP throughput is not affected by presence of another RTP flow, and multimedia flows tend to share available bandwidth between them. However, it is also noticeable that cumulative sending rate of 2 RTP flows is higher than the sending rate of the RTP flow in the previous scenario. This means that overall link capacity utilisation increases with the number of competing flows. In this scenario packet recoveries are also present and they are a bit more frequent than in the previous scenario. This is logical as number of losses also increase, as there is always one more flow in competition for the network bandwidth. Observations concerning the N-FBRA performance are very similar. Sending rate difference between RTP flows when it is used, is not so significant, and for 3 TCP flows it becomes negligible. In this scenario also the TCP throughputs do not seem to be in any way affected by the constant presence of another RTP flow. Both algorithms achieve very high delivery ratio. The FBRA algorithm exceeds 99% on average in every case, whereas the N-FBRA only in a single case falls below 99%. Example of time charts for this scenario are presented in the figures 40, 41, and tables 13, 14, 15 show overall metrics.

Table 13: Overall metrics for 50ms delay in two RTP flows against one TCP flow scenario

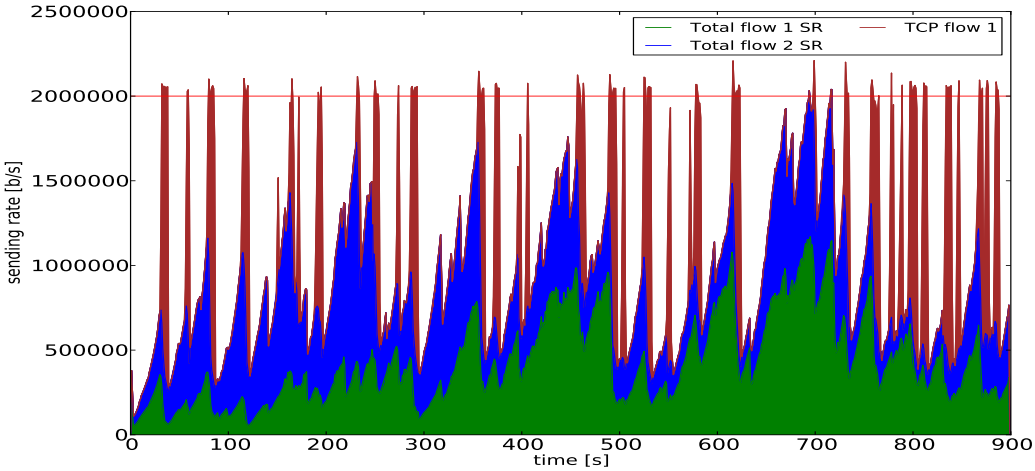
	Metric	**FBRA**		**N-FBRA**	
		avg.	dev	avg.	dev
RTP flow 1	**Sending rate [kb/s]**	348.90	127.84	718.04	161.46
	Goodput [kb/s]	348.58	127.49	714.71	159.99
	Delivery ratio [%]	99.93	0.06	99.56	0.19
	FEC rate [kb/s]	14.37	4.65	-	-
	No. of lost packets	34.83	35.44	381.53	201.29
	No. of discarded packets	0	0	0	0
	Recovered packets	1.93	1.75	0	0
	No. of packets in recv. buffer	14.18	3.41	20.68	3.60
	Underflow time [s]	0.001	0.007	0	0
RTP flow 2	**Sending rate [kb/s]**	584.23	154.39	871.15	168.53
	Goodput [kb/s]	583.33	154.04	871.15	168.53
	Delivery ratio [%]	99.85	0.05	99.34	0.27
	FEC rate [kb/s]	22.89	5.23	-	-
	No. of lost packets	89.63	40.35	650.0	400.37
	No. of discarded packets	0	0	0	0
	Recovered packets	3.07	1.79	0	0
	No. of packets in recv. buffer	20.28	4.20	23.61	3.70
	Underflow time [s]	0.01	0.01	0.01	0.01
TCP	**Throughput in "on" state [kb/s]**	924.86	158.38	506.27	80.50
	"on" time [s]	263.30	35.32	379.03	49.04
	Percentage of fair share	138.73	23.76	75.94	12.08

Table 14: Overall metrics for 50ms delay in two RTP flows against two TCP flows scenario

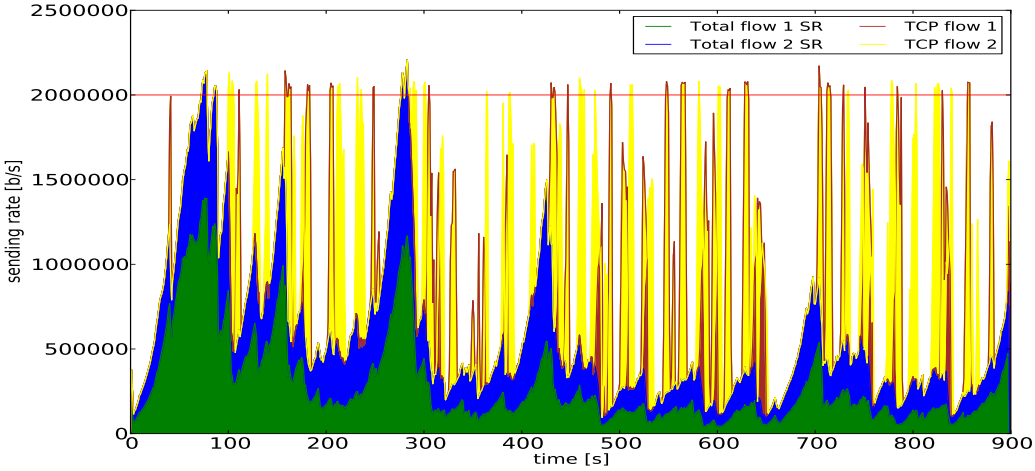
	Metric	FBRA		N-FBRA	
		avg.	dev	avg.	dev
RTP flow 1	**Sending rate [kb/s]**	235.51	44.75	641.96	118.03
	Goodput [kb/s]	234.80	44.62	637.33	116.18
	Delivery ratio [%]	99.70	0.06	99.31	0.22
	FEC rate [kb/s]	10.05	1.77	-	-
	No. of lost packets	88.20	20.62	526.60	225.92
	No. of discarded packets	0	0	0	0
	Recovered packets	5.50	2.96	0	0
	No. of packets in recv. buffer	11.21	1.07	18.84	2.59
	Underflow time [s]	0.14	0.09	0.01	0.02
RTP flow 2	**Sending rate [kb/s]**	178.43	38.54	690.40	108.67
	Goodput [kb/s]	178.01	38.40	684.07	106.63
	Delivery ratio [%]	99.77	0.06	99.10	0.23
	FEC rate [kb/s]	7.99	1.70	-	-
	No. of lost packets	57.67	18.06	664.97	230.94
	No. of discarded packets	0	0	0	0
	Recovered packets	3.97	2.12	0	0
	No. of packets in recv. buffer	9.93	0.79	19.64	2.40
	Underflow time [s]	0.07	0.05	0.03	0.04
TCP	**Throughput in "on" state [kb/s]**	914.53	131.22	489.59	60.34
	"on" time [s]	264.60	32.98	385.70	42.16
	Percentage of fair share	182.91	26.24	97.92	12.07
TCP	**Throughput in "on" state [kb/s]**	918.08	125.55	478.48	56.47
	"on" time [s]	247.57	29.89	361.33	35.40
	Percentage of fair share	183.61	25.11	95.70	11.29

Table 15: Overall metrics for 50ms delay in two RTP flows against three TCP flows scenario

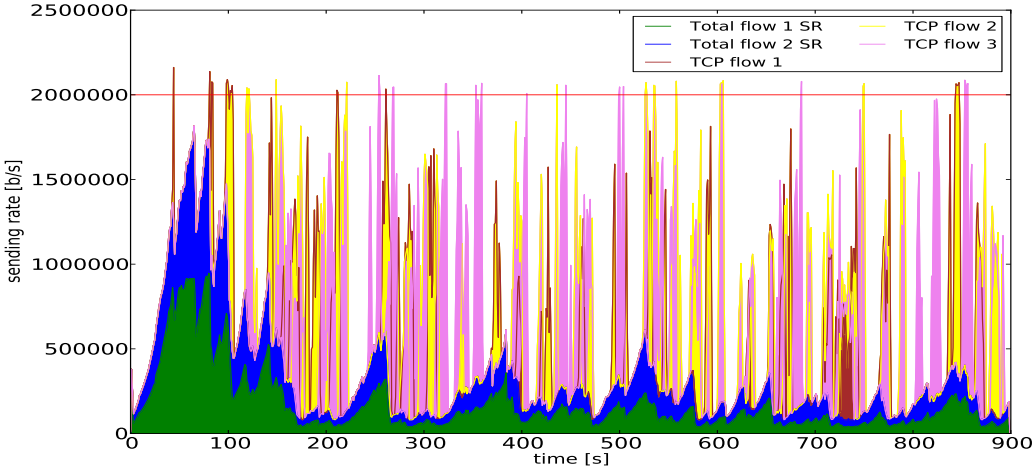
	Metric	FBRA		N-FBRA	
		avg.	dev	avg.	dev
RTP flow 1	**Sending rate [kb/s]**	159.05	26.58	557.25	95.43
	Goodput [kb/s]	158.28	26.42	551.83	93.56
	Delivery ratio [%]	99.52	0.10	99.06	0.27
	FEC rate [kb/s]	7.19	1.05	-	-
	No. of lost packets	140.0	24.08	609.27	219.73
	No. of discarded packets	0	0	0	0
	Recovered packets	9.47	2.54	0	0
	No. of packets in recv. buffer	9.80	0.64	16.89	2.06
	Underflow time [s]	0.21	0.10	0.03	0.03
RTP flow 2	**Sending rate [kb/s]**	125.08	16.69	556.28	83.00
	Goodput [kb/s]	124.51	16.57	548.97	81.22
	Delivery ratio [%]	99.54	0.11	98.71	0.26
	FEC rate [kb/s]	5.99	0.73	-	-
	No. of lost packets	125.0	26.38	759.73	209.80
	No. of discarded packets	0	0	0	0
	Recovered packets	8.73	3.01	0	0
	No. of packets in recv. buffer	9.08	0.32	16.60	1.80
	Underflow time [s]	0.18	0.14	0.06	0.05
TCP	**Throughput in "on" state [kb/s]**	809.90	136.05	484.53	56.39
	"on" time [s]	289.50	44.50	383.50	36.82
	Percentage of fair share	202.48	34.01	121.13	14.10
TCP	**Throughput in "on" state [kb/s]**	796.88	91.04	471.42	45.36
	"on" time [s]	269.0	29.13	354.03	36.75
	Percentage of fair share	199.22	22.76	117.85	11.34
TCP	**Throughput in "on" state [kb/s]**	798.29	123.98	465.43	39.74
	"on" time [s]	258.23	38.75	328.13	30.30
	Percentage of fair share	199.57	31.00	116.36	9.94



(a) Time chart presenting competition between 2 RTP and TCP flow

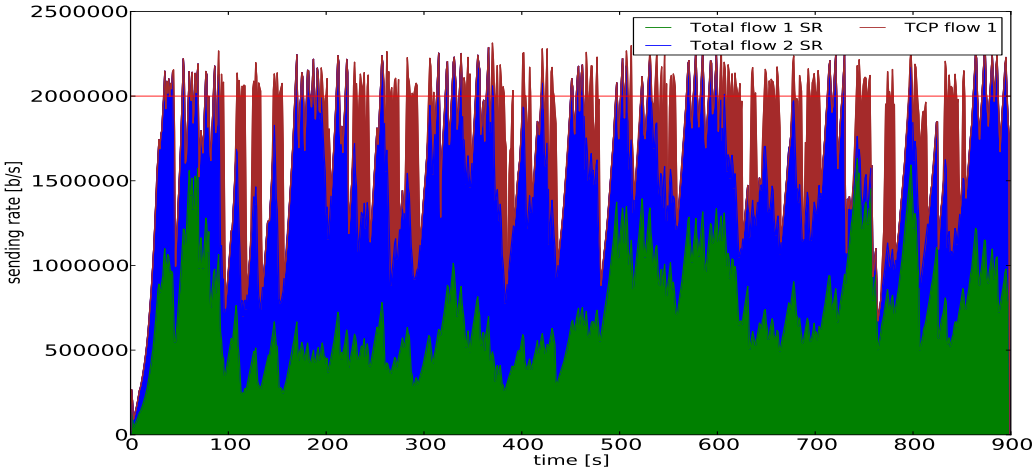


(b) Time chart presenting competition between 2 RTP and 2 TCP flows

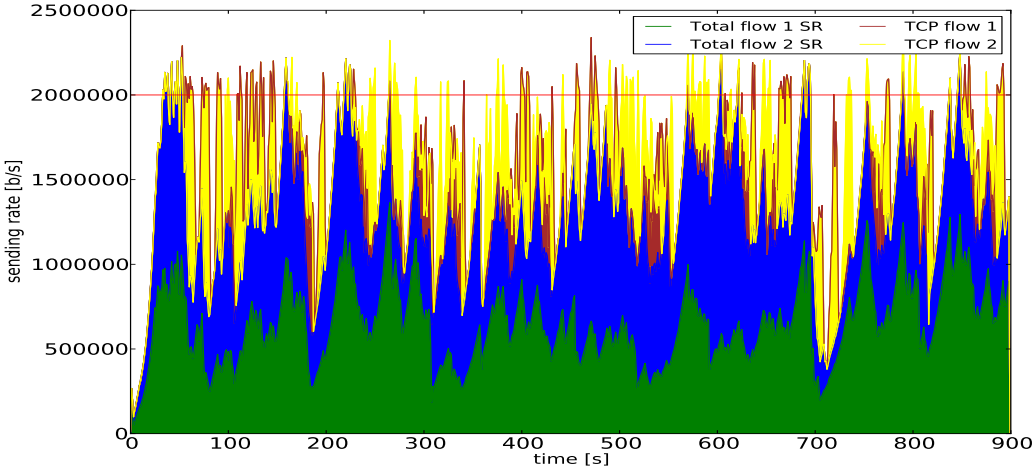


(c) Time chart presenting competition between 2 RTP and 3 TCP flows

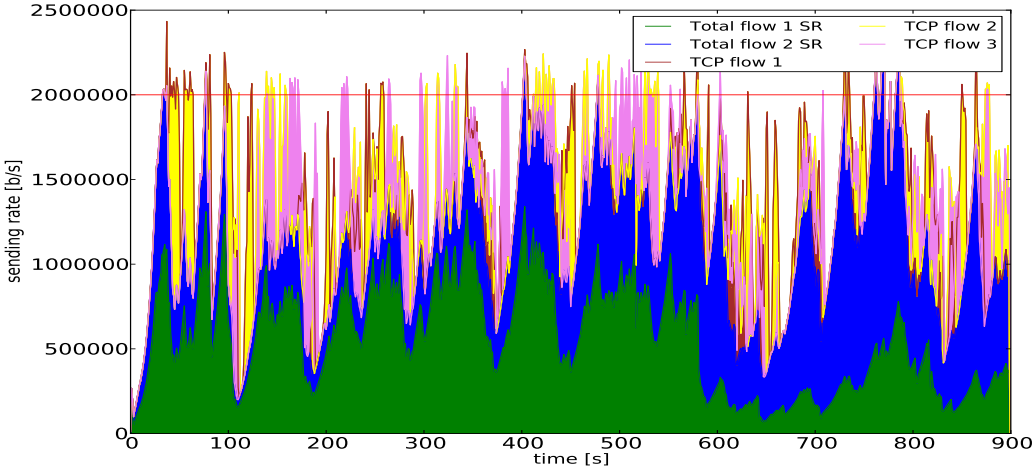
Figure 40: 2 RTP vs. TCP competition in 50ms scenario with FBRA used



(a) Time chart presenting competition between 2 RTP and TCP flow



(b) Time chart presenting competition between 2 RTP and 2 TCP flows



(c) Time chart presenting competition between 2 RTP and 3 TCP flows

Figure 41: 2 RTP vs. TCP competition in 50ms scenario with N-FBRA used

Simulation results for 100ms bottleneck delay:

In the 100ms delay scenario, results are very similar to the previous case. Again the N-FBRA algorithm seems to be superior towards the FBRA one in terms of average sending rate/goodput performance. Differences between sending rates of particular flows when the FBRA algorithm is used are smaller than in the previous case, and tends to become negligible with the increasing number of flows. Overall results of the FBRA algorithm performance are better than in the 50ms delay case. Sending rate and goodput are higher, as similarly to the one RTP flow competition scenario TCP flows cannot accelerate as quickly as in the 50ms case due to increased delivery time of packet acknowledgements. Number of lost and discarded packets is low, as delivery ratio exceeds 99.4% in all simulated cases. Packet recoveries are also observed, but they don't exceed 5% of lost packets on average. Sending rate differences between N-FBRA driven flows are enclosed in rather small interval regardless of number of other flows. Average number of lost and discarded packets is far higher than in the FBRA case, but it results from higher number of packets sent in overall. Packet delivery ratio is also very impressive with all but one cases significantly exceeding 99%. Example of time charts for this scenario are presented in the figures 42, 43, and tables 16, 17, 18 show overall metrics.

Table 16: Overall metrics for 100ms delay in two RTP flows against one TCP flow scenario

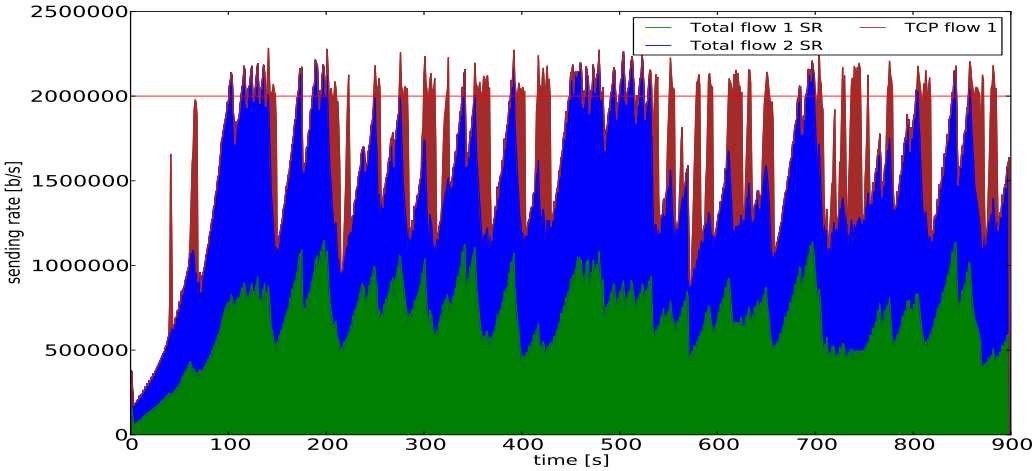
	Metric	FBRA		N-FBRA	
		avg.	dev	avg.	dev
RTP flow 1	**Sending rate [kb/s]**	538.65	116.52	799.41	167.06
	Goodput [kb/s]	538.00	116.19	796.02	165.69
	Delivery ratio [%]	99.88	0.05	99.59	0.14
	FEC rate [kb/s]	18.74	3.47	-	-
	No. of lost packets	61.30	40.36	362.43	160.05
	No. of discarded packets	9.93	3.58	10.67	3.84
	Recovered packets	2.17	1.46	0	0
	No. of packets in recv. buffer	15.95	2.67	19.01	3.36
	Underflow time [s]	0.39	0.12	0.34	0.11
RTP flow 2	**Sending rate [kb/s]**	855.84	121.21	809.52	169.61
	Goodput [kb/s]	853.87	120.81	804.86	167.30
	Delivery ratio [%]	99.77	0.05	99.46	0.21
	FEC rate [kb/s]	26.58	2.60	-	-
	No. of lost packets	186.20	50.41	497.80	272.58
	No. of discarded packets	12.40	3.32	11.73	4.57
	Recovered packets	4.43	2.25	0	0
	No. of packets in recv. buffer	22.73	2.53	18.86	2.88
	Underflow time [s]	0.54	0.14	0.49	0.15
TCP	**Throughput in "on" state [kb/s]**	566.81	70.57	432.85	60.66
	"on" time [s]	355.03	35.96	401.50	47.13
	Percentage of fair share	85.02	10.59	64.93	9.10

Table 17: Overall metrics for 100ms delay in two RTP flows against two TCP flows scenario

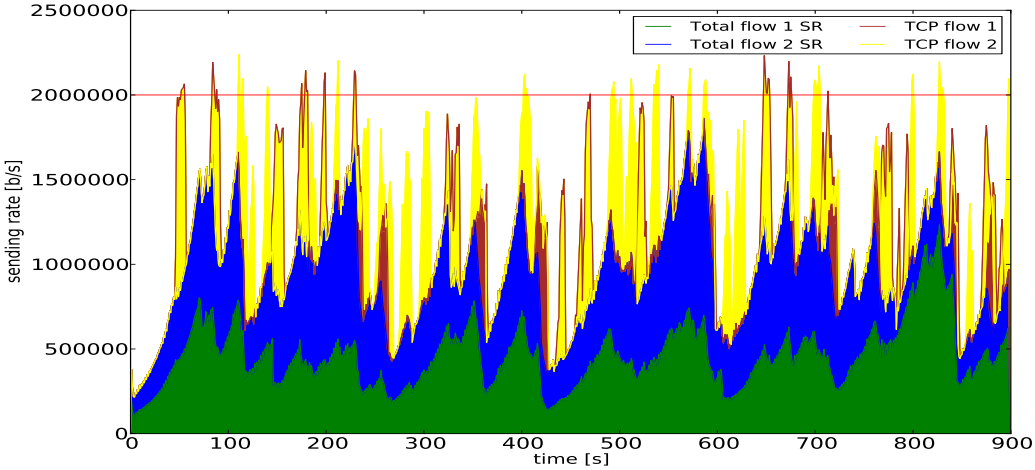
	Metric	FBRA		N-FBRA	
		avg.	dev	avg.	dev
RTP flow 1	**Sending rate [kb/s]**	432.69	97.19	706.61	140.71
	Goodput [kb/s]	431.47	96.61	701.23	138.52
	Delivery ratio [%]	99.73	0.03	99.27	0.24
	FEC rate [kb/s]	15.87	2.99	-	-
	No. of lost packets	123.60	63.21	567.40	247.41
	No. of discarded packets	12.63	3.57	13.43	3.60
	Recovered packets	4.70	2.35	0	0
	No. of packets in recv. buffer	13.50	2.21	16.90	2.59
	Underflow time [s]	0.88	0.22	0.66	0.17
RTP flow 2	**Sending rate [kb/s]**	518.48	102.88	690.56	142.75
	Goodput [kb/s]	516.52	102.23	684.08	140.31
	Delivery ratio [%]	99.63	0.08	99.09	0.23
	FEC rate [kb/s]	18.33	2.82	-	-
	No. of lost packets	189.57	66.64	652.73	267.87
	No. of discarded packets	12.57	3.45	12.53	5.09
	Recovered packets	3.63	2.04	0	0
	No. of packets in recv. buffer	15.42	2.35	16.45	2.49
	Underflow time [s]	1.13	0.22	0.89	0.19
TCP	**Throughput in "on" state [kb/s]**	553.75	82.98	400.05	60.29
	"on" time [s]	356.97	43.47	433.67	51.86
	Percentage of fair share	110.75	16.60	80.00	12.06
TCP	**Throughput in "on" state [kb/s]**	551.52	71.50	390.66	50.08
	"on" time [s]	335.97	36.17	394.37	42.72
	Percentage of fair share	110.30	14.30	78.13	10.02

Table 18: Overall metrics for 100ms delay in two RTP flows against three TCP flows scenario

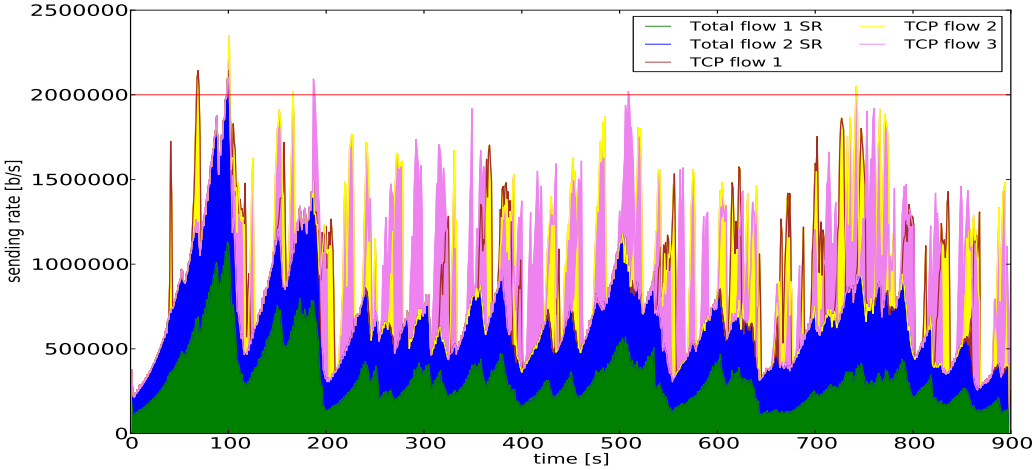
	Metric	FBRA		N-FBRA	
		avg.	dev	avg.	dev
RTP flow 1	**Sending rate [kb/s]**	304.54	45.75	581.02	93.10
	Goodput [kb/s]	303.15	45.26	576.03	91.42
	Delivery ratio [%]	99.55	0.13	99.17	0.23
	FEC rate [kb/s]	11.52	1.80	-	-
	No. of lost packets	156.77	61.83	544.97	189.79
	No. of discarded packets	10.97	3.11	12.77	4.31
	Recovered packets	6.30	2.05	0	0
	No. of packets in recv. buffer	10.57	1.04	14.67	1.90
	Underflow time [s]	1.50	0.30	0.91	0.24
RTP flow 2	**Sending rate [kb/s]**	319.55	51.81	622.88	123.16
	Goodput [kb/s]	317.71	51.38	614.87	120.13
	Delivery ratio [%]	99.43	0.11	98.76	0.36
	FEC rate [kb/s]	12.11	1.80	-	-
	No. of lost packets	200.57	59.58	805.73	324.55
	No. of discarded packets	10.67	3.26	10.43	4.76
	Recovered packets	5.80	2.15	0	0
	No. of packets in recv. buffer	10.93	1.15	15.23	2.11
	Underflow time [s]	1.71	0.28	1.20	0.23
TCP	**Throughput in "on" state [kb/s]**	542.62	70.92	383.67	57.65
	"on" time [s]	362.40	40.79	418.83	51.60
	Percentage of fair share	135.66	17.73	95.92	14.41
TCP	**Throughput in "on" state [kb/s]**	527.49	55.43	365.18	42.33
	"on" time [s]	342.63	30.10	420.20	34.79
	Percentage of fair share	131.87	13.86	91.30	10.58
TCP	**Throughput in "on" state [kb/s]**	529.88	91.09	363.05	45.25
	"on" time [s]	306.03	42.66	376.33	42.04
	Percentage of fair share	132.47	22.77	90.76	11.31



(a) Time chart presenting competition between 2 RTP and TCP flow

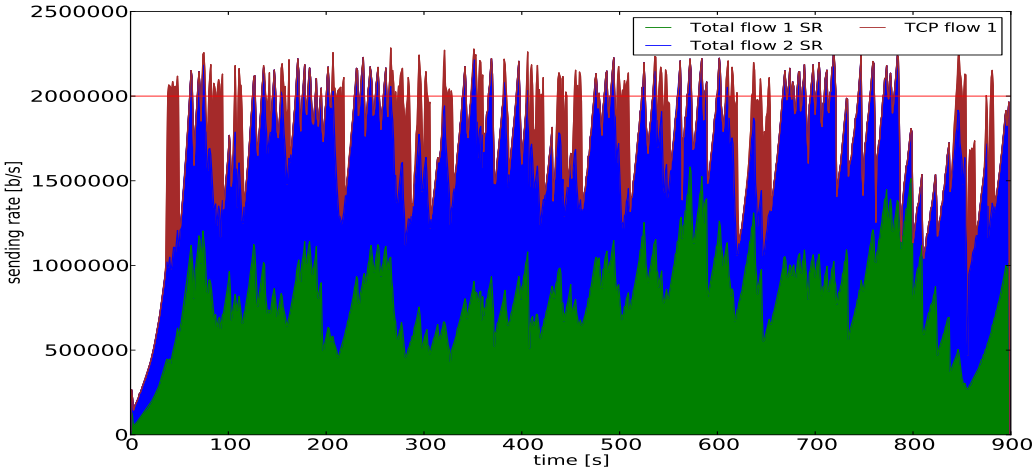


(b) Time chart presenting competition between 2 RTP and 2 TCP flows

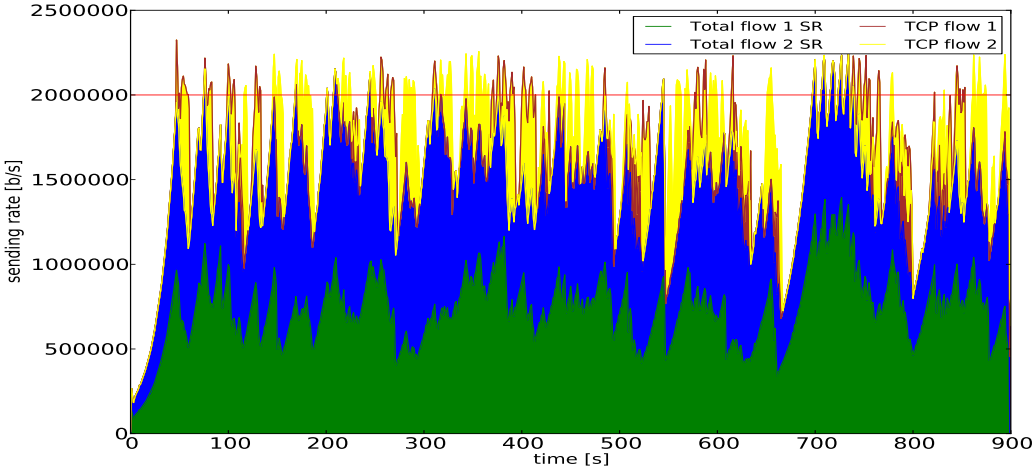


(c) Time chart presenting competition between 2 RTP and 3 TCP flows

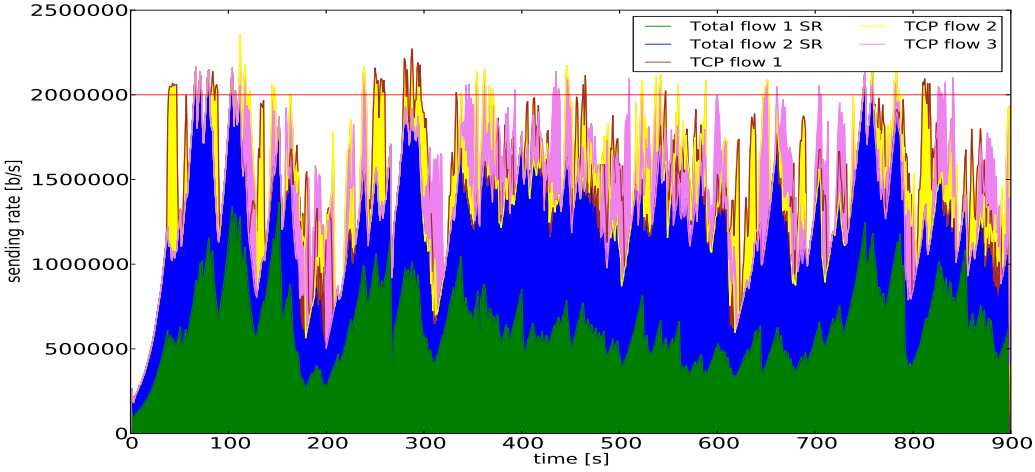
Figure 42: 2 RTP vs. TCP competition in 100ms scenario with FBRA used



(a) Time chart presenting competition between 2 RTP and TCP flow



(b) Time chart presenting competition between 2 RTP and 2 TCP flows



(c) Time chart presenting competition between 2 RTP and 3 TCP flows

Figure 43: 2 RTP vs. TCP competition in 100ms scenario with N-FBRA used

Scenario discussion summary:

In two above presented cases, it was proven that 2 RTP flows can successfully compete for network bandwidth against many TCP flows and against each other. Furthermore, this scenario shows that RTP flows within the same session are fair to each other, as sending rates of participating parties are comparable. In terms of competition between different session, when the FBRA algorithm is used differences between flows are higher than in the N-FBRA case. However, it is very important to state that no situation in which one flow kills another one was observed.

Similarly to results from the one RTP flow competition scenario, question about FEC contribution is hard to answer. Undeniably FBRA algorithm's delivery ratio never falls below 99%, which makes this algorithm superior in terms of reliability. On the other hand FEC usage introduces additional delay in the bandwidth ramp up process which makes this algorithm worse the N-FBRA one in the competition performance. In comparison to the one RTP flow competition scenario, the sending rate results are better for the FBRA algorithm which leads to the conclusion that its usability increases with number of flows competing for the bandwidth.

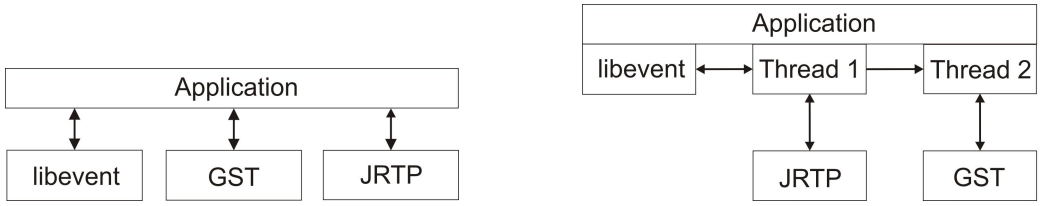
7.3 Real-world implementation

Ns-2 simulator is a very convenient tool to evaluate how newly created solution can perform in the current and future Internet. However, it has also some limitations making *ns-2* obtained results a bit different to the real world. First of all, *ns-2* works as the standalone application modelling only the network part of the system. On the other hand, in the reality network application behaviour is more complicated and depends not only on the core application and the network, but also on system kernel and device drivers. These components may have significant effect on the system performance and they are impossible to simulate in the "idealised world", which *ns-2* is. Furthermore, in our simulation scenarios video packets are represented only as dummy packets. As a result, there is no division for I- and P-frames resulting in impossibility of calculating PSNR metric. Therefore, having successfully tested performance in the simulator, it is vital to check if the system can also function well in the real world.

When testing system performance in the real Internet, it is very difficult to draw any conclusions, as the actual traffic situation in the network is always unknown. As a result, to obtain any meaningful results, it is necessary to collect very large set of data over a long period of time covering wide range of traffic situations. To avoid performing these time consuming experiments, **Adaptive Multimedia System Toolset** was designed.

The **Adaptive Multimedia System Toolset (AMuSys)** uses available open-source library to provide a generic framework for testing video applications. It can be divided into two main parts: multimedia tools and networking ones. The multimedia tools are implemented using open-source GStreamer library [46]. At the moment it has only bindings available for H.264 video usage, but it can be easily extended in the future to provide functionality for other multimedia standards. This part was implemented as the standard GStreamer pipeline. Data can be acquired from live camera, or be injected into the pipeline from a file. Because the GStreamer file source component does not have any clocking properties, process of reading data from file is implemented separately and data are injected into the pipe using the application source component. The GStreamer library includes built-in standard RTP library, but its functionality is insufficient to perform rate adaptation. The networking component was created on top of the open-source JRTP library [47] developed at the Hasselt and Maastricht University. The core library is fully compliant with RFC 3550 [4]. As rate adaptation requires additional functionality, the basic library was extended with few additional modules. The most important addition is the FEC module which is fully compliant with RFC 5109 [6]. By default the library performs protection operation as the XOR function. However, there are bindings created to allow developers apply their own protection method. Furthermore, the library also provides extended feedback profile RFC 4585 [26], but only for point-to-point sessions. In addition, 4 extensions to RTCP headers from RFC 3611 [7] were added (loss RLE, discard RLE, LRRT,DLRR). To allow communication between all these components and avoid multithreading issues standard Linux **libevent** library was used. The sender application is single threaded, and is driven by events generated by

the libevent, and callbacks of received RTCP packets invoked by the JRTP library. Inside these callbacks actual rate adaptation is performed. The receiver application architecture is more complicated, as it consists of two threads. The first thread is responsible for interaction with the network part of the system. It periodically polls JRTP library for received packets, and with the help of libevent generates events of pushing these packets to the decoder, which entrirely operates in the second thread. The general illustration of the RTP sender and the receiver architecture is presented in the figure 44.



(a) Sender architecture (b) Receiver architecture

Figure 44: Video platform sender and receiver architecture

Networking part of the **AMuSys** is built using the Dummynet emulator [48]. This tool allows imposing additional packet delay (emulating physical data propagation), changing link capacity in runtime, adjusting network interface queue, and imposing probabilistic random packet losses. As this work is concentrated on congestion control in the Internet environment, only link capacity, delay and queue size settings are used. Sender and receiver applications use loopback interface to emulate one-way video session between them. The RTP packets and RTCP SRs flow from the sender to the receiver, whereas RRs are sent in the opposite direction.

The purpose of these experiments was to reproduce **ns-2** scenarios presented in the previous chapter, and compare rate the FBRA algorithm performance with results obtained from the simulations. However, this cannot be achieved, because of following limitations of the Dummynet:

- link delay set in the Dummynet varies in an unpredictable manner. Consequently, it is impossible to draw certain conclusions from RTT measurements, as an RTT sample result calculated by JRTP can be higher than it should be for existing network conditions. This leads to worse bandwidth utilisation in comparison to ns-2 results, as the FBRA algorithm may decide to reduce the sending rate based on unjustified high RTT value. Furthermore it is also impossible to reproduce the RTP vs. TCP competition scenarios, as TCP flows are unable to use more than 20% of available link capacity. Unpredictable RTT issue is illustrated in the figure 45. In the presented experiment ICMP packet of 8008 bytes size was being sent every 66ms (15 packets per second) simulating constant bit rate traffic of 1 Mbit/s. Dummynet was set to emulate 2 Mbit/s constant link with one-way delay of 50ms. The ping statistics show that average RTT value was 142ms, whereas it should be around 100ms. Very bandwidth utilisation of TCP flows also results from these variable RTT values, as TCP Cubic version which highly relies on RTT is used by most Linux machines.

- Capacities of network router queues are measured in bytes, and according to [49] they should equal link capacity times RTT. Although it is possible to define queue size in the Dummynet in bytes, random packet losses occur when the link capacity is changed at runtime. Therefore, queue size is set for 50 packets.

```
marcin@Kubuntu-LTS:~/ipfw3$ sudo ipfw add pipe 2 ip from 127.0.0.1 to 127.0.0.1 out
00100 pipe 2 ip from 127.0.0.1 to 127.0.0.1 out
marcin@Kubuntu-LTS:~/ipfw3$ sudo ipfw pipe 2 config bw 2Mbit/s delay 50ms queue 50
marcin@Kubuntu-LTS:~/ipfw3$ sudo ping -s 8000 -i 0.066 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 8000(8028) bytes of data.
8008 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=102 ms
8008 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=149 ms
8008 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=114 ms
8008 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=151 ms
8008 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=122 ms
8008 bytes from 127.0.0.1: icmp_seq=6 ttl=64 time=155 ms
8008 bytes from 127.0.0.1: icmp_seq=7 ttl=64 time=125 ms
8008 bytes from 127.0.0.1: icmp_seq=8 ttl=64 time=163 ms
8008 bytes from 127.0.0.1: icmp_seq=9 ttl=64 time=129 ms
8008 bytes from 127.0.0.1: icmp_seq=10 ttl=64 time=159 ms
8008 bytes from 127.0.0.1: icmp_seq=11 ttl=64 time=121 ms
8008 bytes from 127.0.0.1: icmp_seq=12 ttl=64 time=160 ms
8008 bytes from 127.0.0.1: icmp_seq=13 ttl=64 time=129 ms
8008 bytes from 127.0.0.1: icmp_seq=14 ttl=64 time=166 ms
8008 bytes from 127.0.0.1: icmp_seq=15 ttl=64 time=129 ms
8008 bytes from 127.0.0.1: icmp_seq=16 ttl=64 time=163 ms
8008 bytes from 127.0.0.1: icmp_seq=17 ttl=64 time=131 ms
8008 bytes from 127.0.0.1: icmp_seq=18 ttl=64 time=158 ms
8008 bytes from 127.0.0.1: icmp_seq=19 ttl=64 time=129 ms
8008 bytes from 127.0.0.1: icmp_seq=20 ttl=64 time=159 ms
8008 bytes from 127.0.0.1: icmp_seq=21 ttl=64 time=125 ms
8008 bytes from 127.0.0.1: icmp_seq=22 ttl=64 time=159 ms
8008 bytes from 127.0.0.1: icmp_seq=23 ttl=64 time=125 ms
8008 bytes from 127.0.0.1: icmp_seq=24 ttl=64 time=159 ms
8008 bytes from 127.0.0.1: icmp_seq=25 ttl=64 time=125 ms
8008 bytes from 127.0.0.1: icmp_seq=26 ttl=64 time=159 ms
8008 bytes from 127.0.0.1: icmp_seq=27 ttl=64 time=125 ms
8008 bytes from 127.0.0.1: icmp_seq=28 ttl=64 time=163 ms
8008 bytes from 127.0.0.1: icmp_seq=29 ttl=64 time=127 ms
8008 bytes from 127.0.0.1: icmp_seq=30 ttl=64 time=164 ms
8008 bytes from 127.0.0.1: icmp_seq=31 ttl=64 time=123 ms
8008 bytes from 127.0.0.1: icmp_seq=32 ttl=64 time=157 ms
8008 bytes from 127.0.0.1: icmp_seq=33 ttl=64 time=129 ms
8008 bytes from 127.0.0.1: icmp_seq=34 ttl=64 time=161 ms
8008 bytes from 127.0.0.1: icmp_seq=35 ttl=64 time=132 ms
8008 bytes from 127.0.0.1: icmp_seq=36 ttl=64 time=157 ms
8008 bytes from 127.0.0.1: icmp_seq=37 ttl=64 time=126 ms
8008 bytes from 127.0.0.1: icmp_seq=38 ttl=64 time=163 ms
8008 bytes from 127.0.0.1: icmp_seq=39 ttl=64 time=128 ms
8008 bytes from 127.0.0.1: icmp_seq=40 ttl=64 time=152 ms
8008 bytes from 127.0.0.1: icmp_seq=41 ttl=64 time=124 ms
8008 bytes from 127.0.0.1: icmp_seq=42 ttl=64 time=154 ms
8008 bytes from 127.0.0.1: icmp_seq=43 ttl=64 time=124 ms
8008 bytes from 127.0.0.1: icmp_seq=44 ttl=64 time=160 ms
8008 bytes from 127.0.0.1: icmp_seq=45 ttl=64 time=129 ms
8008 bytes from 127.0.0.1: icmp_seq=46 ttl=64 time=158 ms
8008 bytes from 127.0.0.1: icmp_seq=47 ttl=64 time=127 ms
8008 bytes from 127.0.0.1: icmp_seq=48 ttl=64 time=160 ms
8008 bytes from 127.0.0.1: icmp_seq=49 ttl=64 time=125 ms
8008 bytes from 127.0.0.1: icmp_seq=50 ttl=64 time=159 ms
8008 bytes from 127.0.0.1: icmp_seq=51 ttl=64 time=125 ms
8008 bytes from 127.0.0.1: icmp_seq=52 ttl=64 time=163 ms
8008 bytes from 127.0.0.1: icmp_seq=53 ttl=64 time=129 ms
8008 bytes from 127.0.0.1: icmp_seq=54 ttl=64 time=159 ms
8008 bytes from 127.0.0.1: icmp_seq=55 ttl=64 time=125 ms
8008 bytes from 127.0.0.1: icmp_seq=56 ttl=64 time=163 ms
8008 bytes from 127.0.0.1: icmp_seq=57 ttl=64 time=130 ms
8008 bytes from 127.0.0.1: icmp_seq=58 ttl=64 time=160 ms
8008 bytes from 127.0.0.1: icmp_seq=59 ttl=64 time=130 ms
8008 bytes from 127.0.0.1: icmp_seq=60 ttl=64 time=160 ms
8008 bytes from 127.0.0.1: icmp_seq=61 ttl=64 time=125 ms
^C
--- 127.0.0.1 ping statistics ---
63 packets transmitted, 61 received, 3% packet loss, time 4245ms
rtt min/avg/max/mdev = 102.869/142.705/166.795/17.497 ms, pipe 3
```

Figure 45: Ping experiment illustrating Dummynet RTT issues

7.4 Evaluation of real-world implementation

The primary purpose of this work is to verify the hypothesis if the FEC can be used for the rate adaptation and error recovery simultaneously. Therefore we decided to limit evaluation of the real-world implementation only to the FBRA algorithm, as it joins ideas of rate adaptation and error recovery. Furthermore, taking into account above Dummynet limitations following testing scenarios were conducted:

- Constant link capacity with single RTP flow
- Variable link capacity with single RTP flow
- Constant link capacity with two RTP flows

The purpose of the first scenario is to verify how stable the FBRA algorithm can perform when the link capacity is constant, and it does not have to compete with other flows. The second scenario is targeted at simulating competition between RTP flow and TCP flow. Change of the link capacity is similar to the effect when a TCP flow takes/releases network bandwidth. Although this scenario does not ideally reproduce RTP vs. TCP competition, there is no better solution to verify this feature of the algorithm performance. The final experiment aims at verifying fairness between RTP flows when they have to compete for the network bandwidth.

In all scenarios video session just in the upstream direction is emulated. It is assumed that in the downstream direction results would be very similar. To make PSNR measurements one well-known file presenting news broadcast is read and sent over the network. On the receiver side the stream is decoded and stored in file. PSNR is calculated based on comparison of the original file with the decoded one. In order to evaluate value of PSNR the rate adaptation module of the sender logs all its decisions, so that another session mirroring behaviour of the rate adaptation module can be run on the lossless link. Assessment of PSNR is conducted by comparison of the obtained PSNR value with the value achieved on the lossless link. However, it is important to note that due to interference of various factors (e.g. task scheduling) it is impossible to exactly reproduce the video session. Therefore the recreated experiment is quasi-identical, and its deviation from the original status does not influence results. Furthermore, all scenarios are divided into three separate cases in which bottleneck delays of 50ms, 100ms, and 240ms are set.

Experiment setup is illustrated in the figure 46. It consists of two machines connected with the Ethernet cable. To emulate desired network characteristics outgoing traffic is sent to the Dummynet driver, whereas the incoming traffic is left untouched.

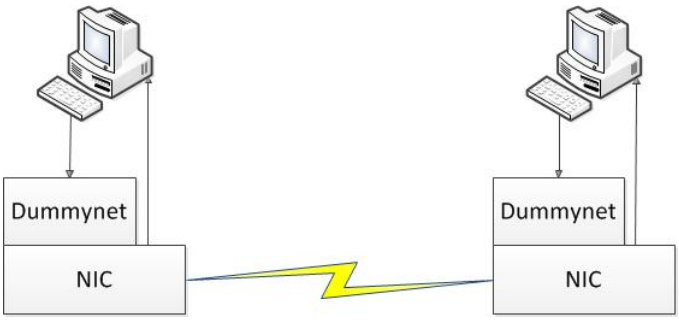


Figure 46: Network setup

7.4.1 Constant link capacity with single RTP flow scenario

Emulation results for 50ms bottleneck delay:

Generally speaking results obtained in the 50ms delay scenario are very stable. The average sending rate is about 1160 kb/s, and the delivery ratio is 99.49%. It is possible to observe Dummynet related issues, as despite never approaching the link capacity limit, on average 2.3 packets are discarded in every run. Exemplary graphs illustrating this problem in one run are presented in the figure 47. Graph 47c visualises RTT measurements as the function of time. RTT variations are similar to the effect which was observed in the ping experiment. Such RTT value oscillations prevent the rate adaptation algorithm from obtaining higher throughput. The other noticeable difference in comparison to ns-2 results is much higher FEC overhead, as it exceeds 10% now, whereas in the simulator it was oscillating around 5%. This unexpected increase also results from RTT instabilities, as transitions between encoding rate changes do not happen as smoothly as in the simulation scenario. Here due to high RTT variations the rate adaptation algorithm very often does not go through easily through s-,s+,s++,u sequence of states, but period of time spent in s+ and s++ states is longer.

In the series of 10 runs, we were able to obtain average PSNR value of 45.20dB. On the lossless channel average PSNR reached 45.32dB. Slight difference between the lossy PSNR result and the lossless one indicates that the rate adaptation algorithm does not cause deterioration of quality. All results of this scenario are presented in the table 19.

Table 19: Overall metrics for 50ms delay in constant link capacity scenario

Metric	FBRA	
	avg	dev
Sending rate [kb/s]	1164.39	94.76
Goodput [kb/s]	1158.43	92.06
Delivery ratio [%]	99.51	0.54
FEC rate [kb/s]	148.35	7.31
No. of lost packets	0	0
No. of discarded packets	2.3	2.79
Recovered packets	0	0
PSNR [dB]	45.20	0.52
Lossless PSNR [dB]	45.32	0.64

Emulation results for 100ms bottleneck delay:

Results of the 100ms delay scenario are again very stable. The sending rate is higher than for 50ms delay case and reaches almost 1300kbit/s on average. On the other hand the reliability worsens, as the delivery ratio falls to 98.47%. There are no observed packet losses, and all deteriorations result from about 50 packets discarded on average. In this scenario it is easier to exceed the delay budget in comparison to the previous case, as it is at the beginning smaller by 50ms. The smaller budget together with unpredictable physical propagation delay is mainly responsible for such a significant increase of discarded packets. Figure 48 presents results of one scenario run. Looking at the figure 48b one can notice that most of discards appear when the sending rate approaches the link capacity limit (for example between 350th and 400th second of the run). However, all these discards cannot be logically explained, as the capacity limit is exceeded only a few times, and for a very short period of time, and it is unlikely that such a small overshooting cause packet queuing exceeding the delay budget. Similarly to the previous case the FEC overhead is also much higher than in the identical scenario conducted in the *ns-2* simulator, and the reason for this behaviour is identical to the one in the 50ms delay case.

Average PSNR achieved on the lossy channel equals 43.80dB, whereas on the lossless link 45.98dB was achieved. The difference between these values is much higher than in the previous case, and it mainly results from significantly higher number of discarded packets. All statistical results of this scenario are presented in the table 20

Table 20: Overall metrics for 100ms delay in constant link capacity scenario

Metric	FBRA	
	avg	dev
Sending rate [kb/s]	1299.26	41.30
Goodput [kb/s]	1279.33	41.43
Delivery ratio [%]	98.47	0.48
FEC rate [kb/s]	149.76	4.13
No. of lost packets	0	0
No. of discarded packets	49.80	13.16
Recovered packets	0	0
PSNR [dB]	43.80	0.71
Lossless PSNR [dB]	45.98	0.30

Emulation results for 240ms bottleneck delay:

In the 240ms delay scenario, the results are significantly worse than in both previous cases. The goodput is poor in comparison to the previous scenarios, as well as, the delivery ratio which barely exceeds 97% border. Furthermore, the average number of discarded packets also increases because of identical reasons, as in the 100ms delay scenario. Because the rate adaptation algorithm orders sending rate reduction based on appearance of discarded packets, much lower goodput can be explained by relatively high RTT values, which increase unpredictably, leading to discards, which finally makes the rate adaptation algorithm reduce the sending rate. Figure 49 presents results of one scenario run, whereas the overall metrics are presented in the table 21

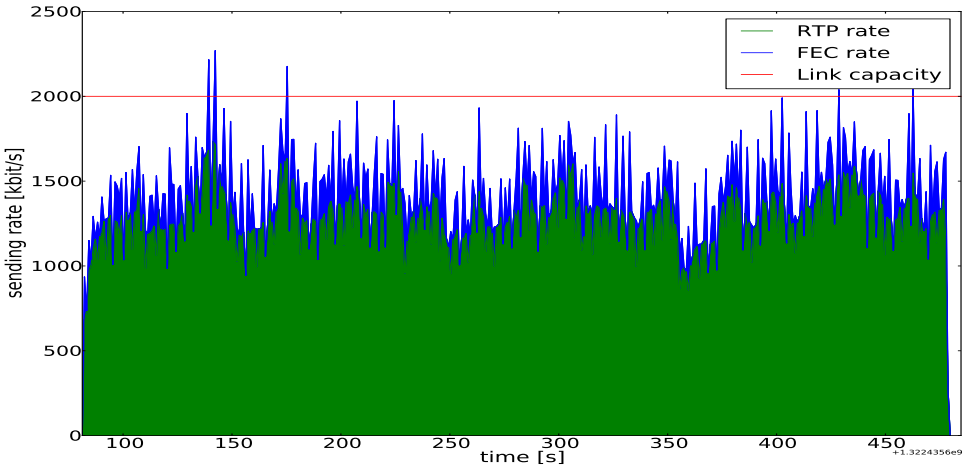
The PSNR results are very bad for this scenario. The average value on the lossy channel is 35.51dB, whereas on the lossless channel 42.60dB is achieved. Difference of more than 7dB is hard to accept, in the case where only around 70 packets were discarded on average. Such a huge difference implies that among these discarded packets must be a lot of I-frames whose losses contribute more significant damages. It is even more probable that I-frames must be among discarded packets, in our observations it was found out that the Dummynet poorly handles larger packets (I-frames).

Table 21: Overall metrics for 240ms delay in constant link capacity scenario

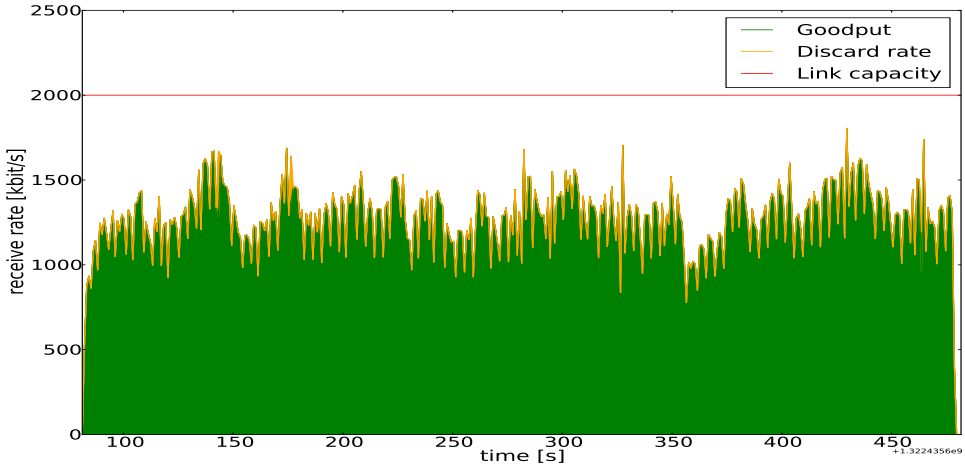
Metric	FBRA	
	avg	dev
Sending rate [kb/s]	787.24	27.67
Goodput [kb/s]	765.48	26.33
Delivery ratio [%]	97.24	0.53
FEC rate [kb/s]	93.78	3.75
No. of lost packets	0	0
No. of discarded packets	70.20	10.41
Recovered packets	0	0
PSNR [dB]	35.51	1.28
Lossless PSNR [dB]	42.60	0.37

Scenario discussion summary:

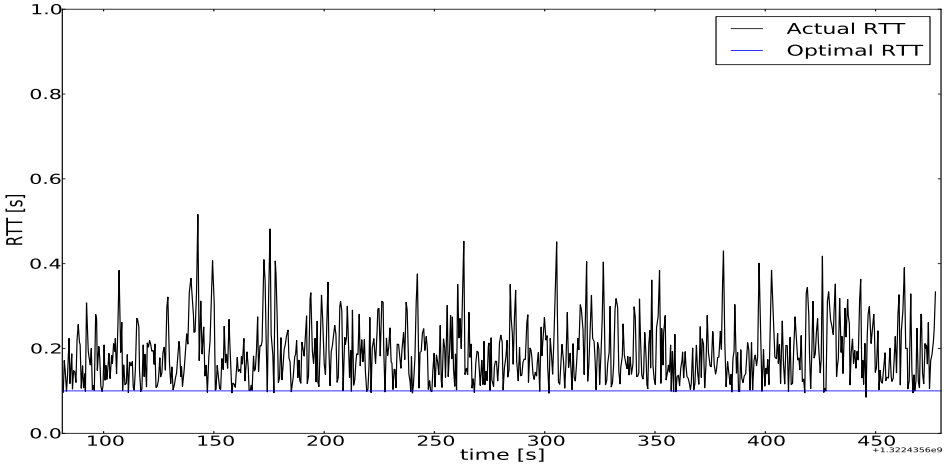
The purpose of this scenario is to evaluate how stable the FBRA algorithm can perform in the real Internet. In terms of stability results are very good for all subscenarios. Analysing time charts presented in the figures 47a, 48a, 49a, it can be stated that the sending rate value oscillates around some average number and changes "periodically" in the "sine wave" style. Results of this experiment are unfortunately not completely accurate due to already explained RTT issues. If the Dummynet emulator did not have these limitations, discards would not appear for these sending rates values, and FEC overhead would be significantly lower. It is possible that the FBRA algorithm would achieve far better sending rate, and be able to almost keep it constant without making "sine wave" like fluctuations.



(a) Sending rate

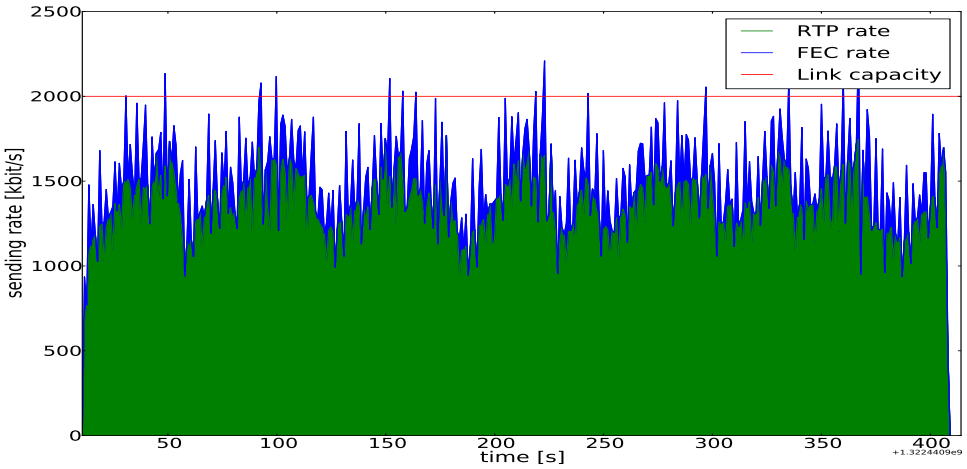


(b) Goodput

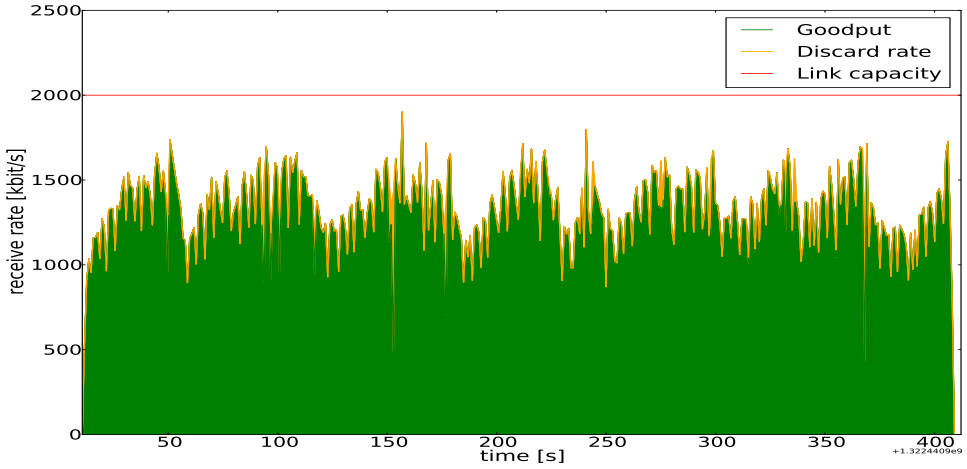


(c) RTT

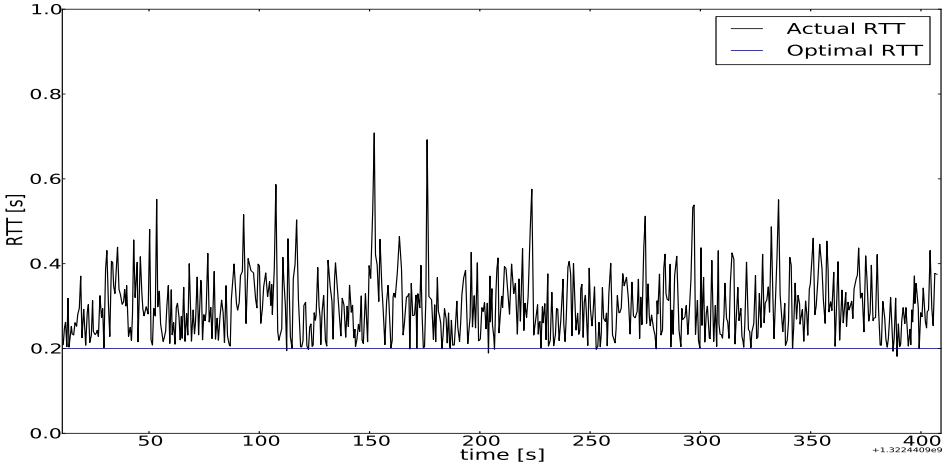
Figure 47: Time charts for 50ms delay constant link capacity scenario



(a) Sending rate

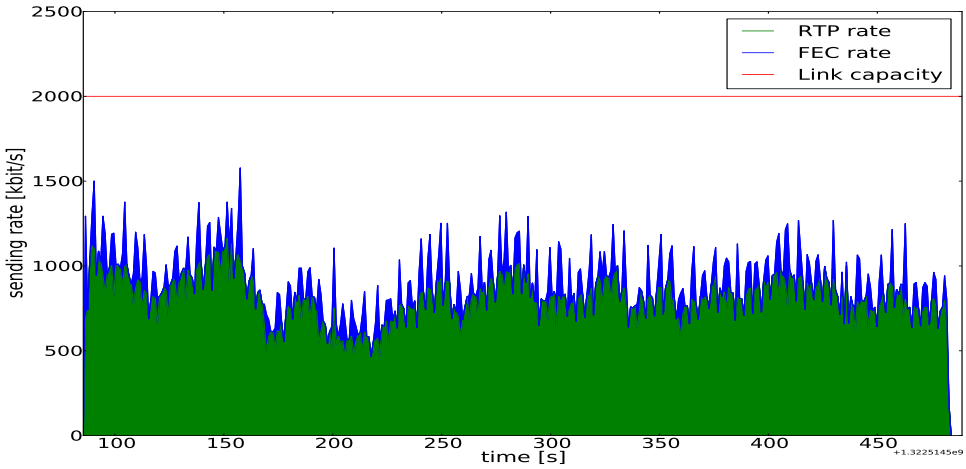


(b) Goodput

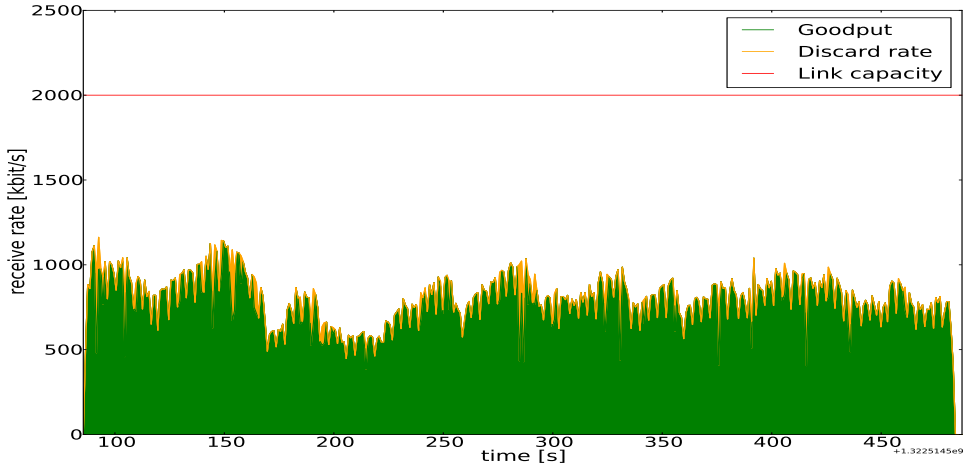


(c) RTT

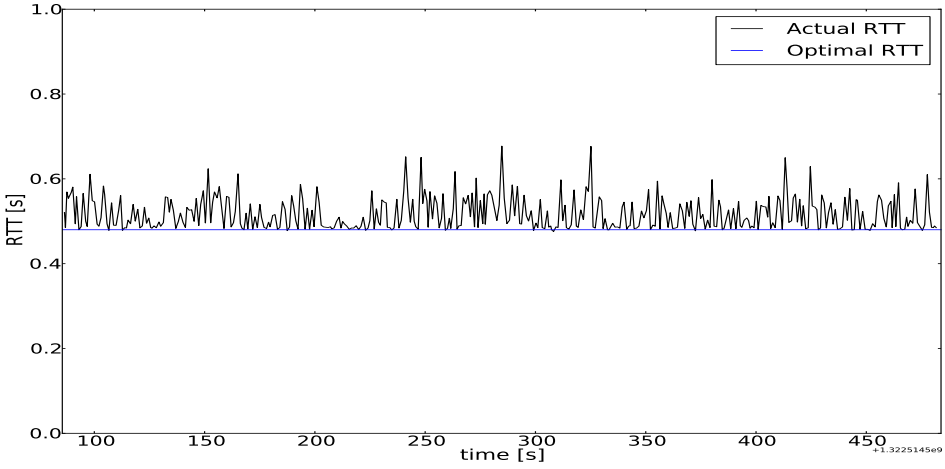
Figure 48: Time charts for 100ms delay constant link capacity scenario



(a) Sending rate



(b) Goodput



(c) RTT

Figure 49: Time charts for 240ms delay constant link capacity scenario

7.4.2 Variable link capacity with single RTP flow

Emulation results for 50ms bottleneck delay:

In the 50ms delay variable link capacity scenario, the main differences between the *ns-2* results, and the results obtained in the real Internet are much lower throughput, and worse delivery ratio. Similarly to all previous real-world scenarios, obtained results are far worse than in the identical experiment conducted in the *ns-2* and unpredictable variation of RTT is the root of differences. Time plots illustrating a single experiment run are presented in the figure 50. RTT variations are far higher in this scenario than in the previous one. It can be noticed in the figure 50c that at one point it exceeds even 1 second, whereas its optimum value is 100ms. Such a high amplitude of RTT variations leads to dramatic increase of the number of packets discarded. Furthermore packet losses are also 10 times higher than in the *ns-2* simulations. This is also the effect of RTT variations, as RTCP reports are not delivered to the sender in the appropriate time, therefore limiting its capabilities of making a correct response to the network situation.

Concerning PSNR results, average values obtained on the lossy channel and the lossless one differ only by 0.6dB. If we compare these results with the fact that more than 300 packets are discarded on average, it can be concluded that most of lost or discarded packets must contain P-frames. Results of this scenario are presented in the table 22.

Table 22: Overall metrics for 50ms delay in variable link capacity with single RTP flow scenario

Metric	**FBRA**	
	avg	dev
Sending rate [kb/s]	107.85	5.84
Goodput [kb/s]	104.60	4.85
Delivery ratio [%]	97.03	0.98
FEC rate [kb/s]	17.72	0.58
No. of lost packets	22.80	26.51
No. of discarded packets	288.0	78.90
Recovered packets	0	0
PSNR [dB]	27.22	0.84
Lossless PSNR [dB]	27.84	1.79

Emulation results for 100ms bottleneck delay:

In the 100ms delay scenario, results are similar to the 50ms delay case. The average sending rate is almost equal, but the goodput is a bit worse. There is a significant increase in the number of discarded packets in comparison to the previous case. This boost is the result of the smaller delay budget available in this scenario. To sum up, again the real-world results are much different from the ones obtained in the *ns-2*, and the RTT issue is the main reason for this. Time plots of one experiment run are presented in the figure 51. It can be observed in many places on

these plots that despite almost constant bandwidth underutilisation, packet RTTs increase significantly above the optimum value, and packet discards appear.

Average PSNR value obtained on the lossy channel is lower by 3dB to the lossless result. This significant difference in comparison to the previous case indicates that some I-frames must be lost/discarded. This scenario results are presented in the table 23.

Table 23: Overall metrics for 100ms delay in variable link capacity with single RTP flow scenario

Metric	**FBRA**	
	avg	dev
Sending rate [kb/s]	107.54	1.65
Goodput [kb/s]	102.24	1.26
Delivery ratio [%]	95.08	0.90
FEC rate [kb/s]	16.65	0.40
No. of lost packets	15.30	13.89
No. of discarded packets	450.0	116.82
Recovered packets	0	0
PSNR [dB]	25.45	0.69
Lossless PSNR [dB]	28.46	0.15

Emulation results for 240ms bottleneck delay:

Results of the 240ms delay scenario show constant bandwidth utilisation. The average sending rate of 69kbit/s is far below the rate obtained in the *ns-2*. Delivery ratio is almost equal to the previous case, as well as, the number of discarded packets is. Such poor results are again difficult to reasonably explain, as the bandwidth utilisation is during whole experiment way below the link capacity. There are almost no losses, but unexpectedly recoveries appear. This is surprising, as in the long delay scenarios probability of delivering needed FEC packet in the delay budget time is small, and despite presence of losses in the previous cases recoveries did not appear before. No serious conclusions should be drawn from these recoveries, as they are really negligible, and do not contribute a lot to the overall results. Time plots of a single experiment run are presented in the figure 52.

Average PSNR results on the lossy channel and the lossless one differ by more than 3dB which again indicates that I-frames must reach the receiver too late to be sent to the decoder. All results are presented in the table 24.

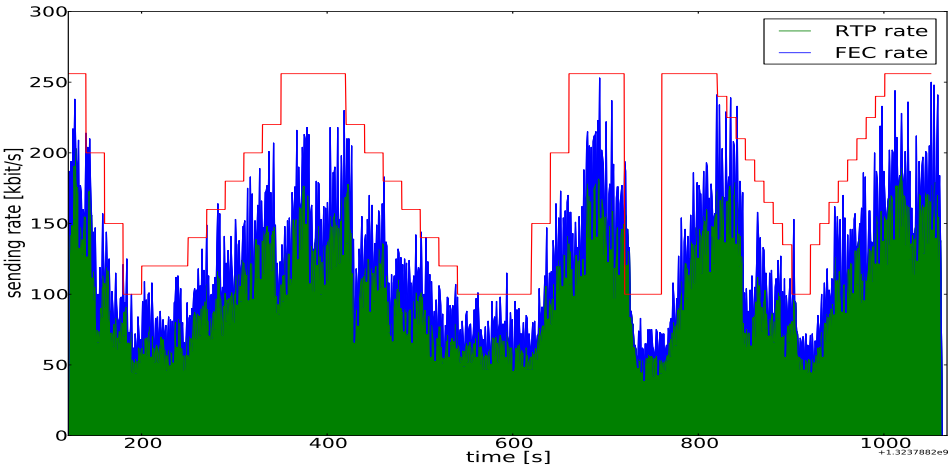
Table 24: Overall metrics for 240ms delay in variable link capacity with single RTP flow scenario

Metric	FBRA	
	avg	dev
Sending rate [kb/s]	68.60	2.83
Goodput [kb/s]	64.35	3.37
Delivery ratio [%]	93.79	0.02
FEC rate [kb/s]	11.00	0.44
No. of lost packets	0.20	0.75
No. of discarded packets	464.40	388.69
Recovered packets	0.40	0.49
PSNR [dB]	21.81	0.57
Lossless PSNR [dB]	25.64	0.24

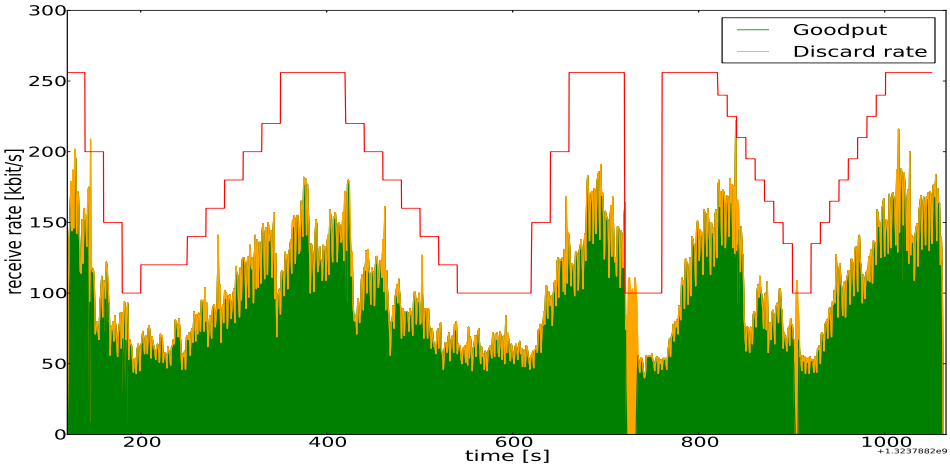
Scenario discussion summary:

The purpose of this scenario is to evaluate how well the FBRA algorithm performs on the variable link capacity channel and compare the real-world results with the ones obtained in the *ns-2*. Unfortunately it is impossible to obtain even similar outcome in the real-world, as in the *ns-2*. The main factor responsible for this blow is the unpredictable packet propagation time, when the Dummynet is used to simulate particular network conditions. These issues leads to unreasonably high RTT values preventing the FBRA algorithm from utilising more link capacity. Furthermore the increased packet propagation time sometimes leads to packet discards, although the bandwidth limits are not exceeded. As a result, far worse delivery ratio and sending rate values are observed, as well as, PSNR values differ significantly from PSNR results obtained on the lossless channel.

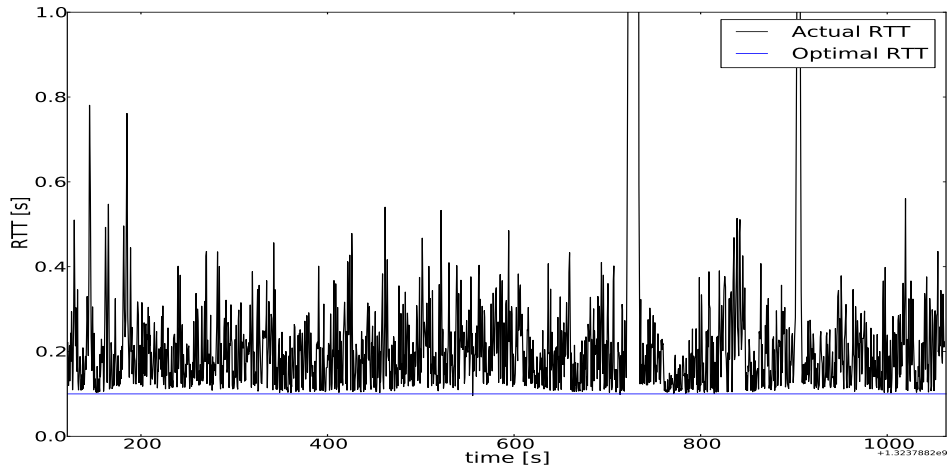
On the other hand, in the "real" Internet the network situation is completely unpredictable. It is impossible to say that RTT values recorded during real network measurements are not impacted by similar effects either. As we can observe that the sender reacts correctly to the RTCP cues, we can state that in overall the FBRA algorithm works correctly in the real-world.



(a) Sending rate

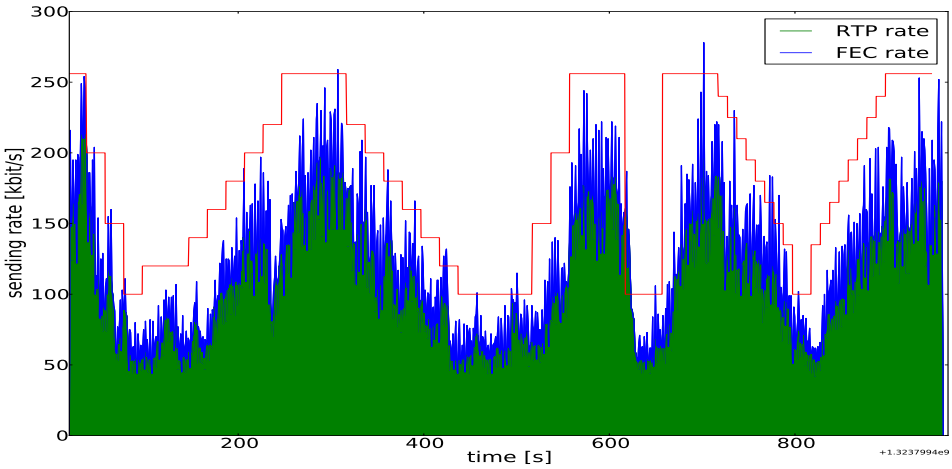


(b) Goodput

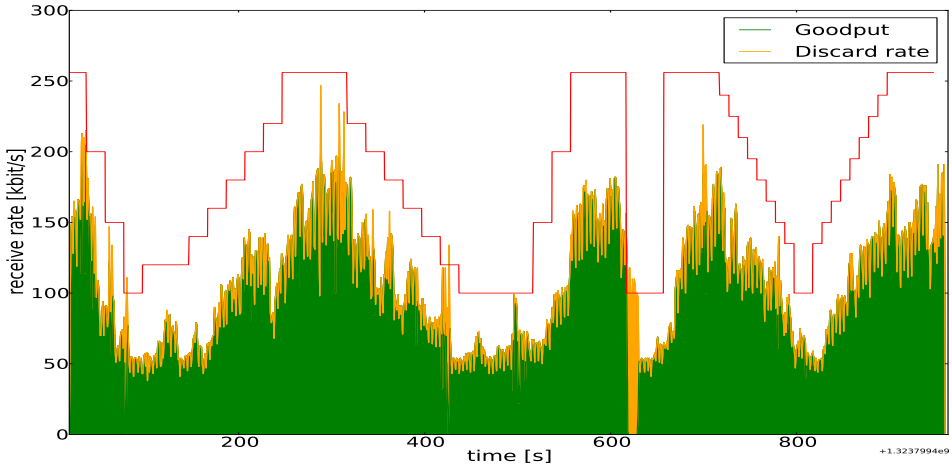


(c) RTT

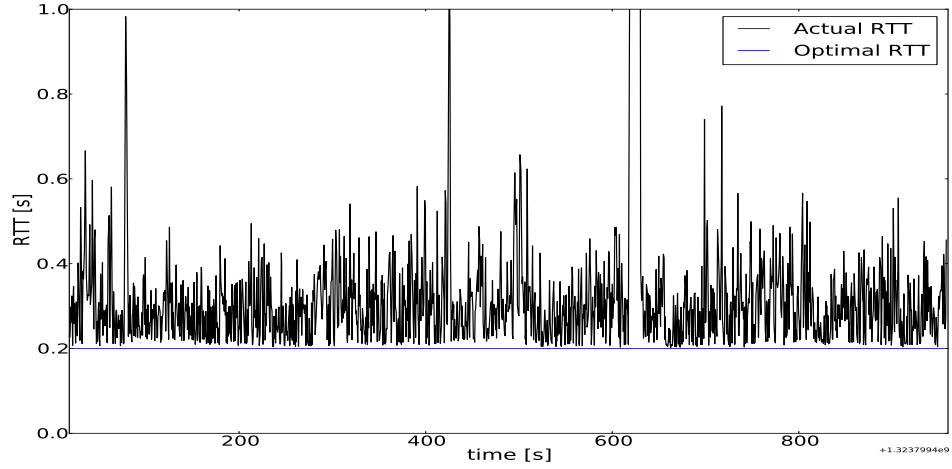
Figure 50: Time charts for 50ms delay variable link capacity scenario



(a) Sending rate

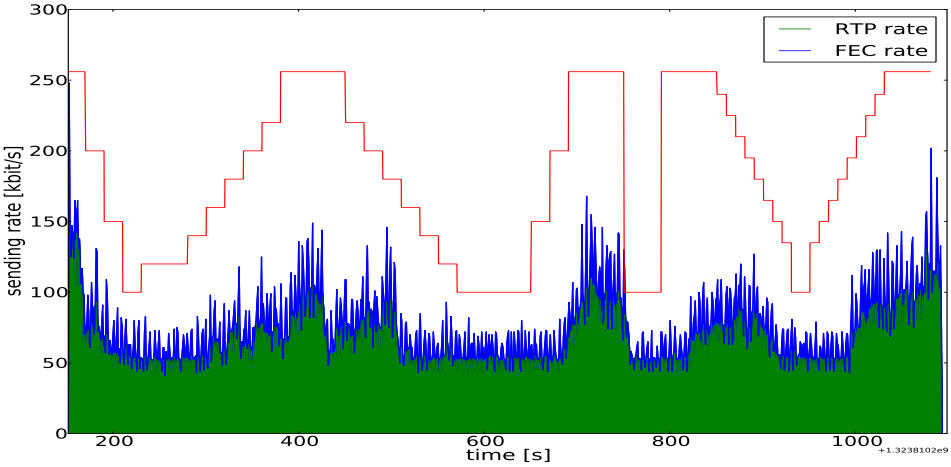


(b) Goodput

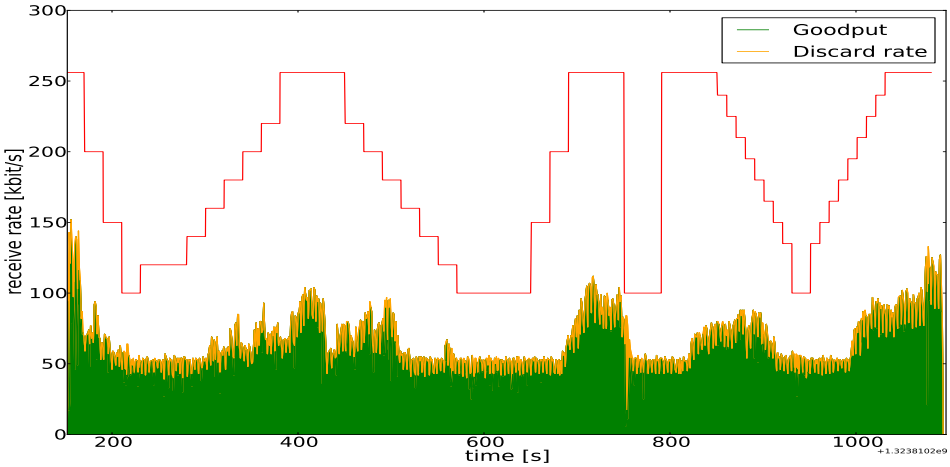


(c) RTT

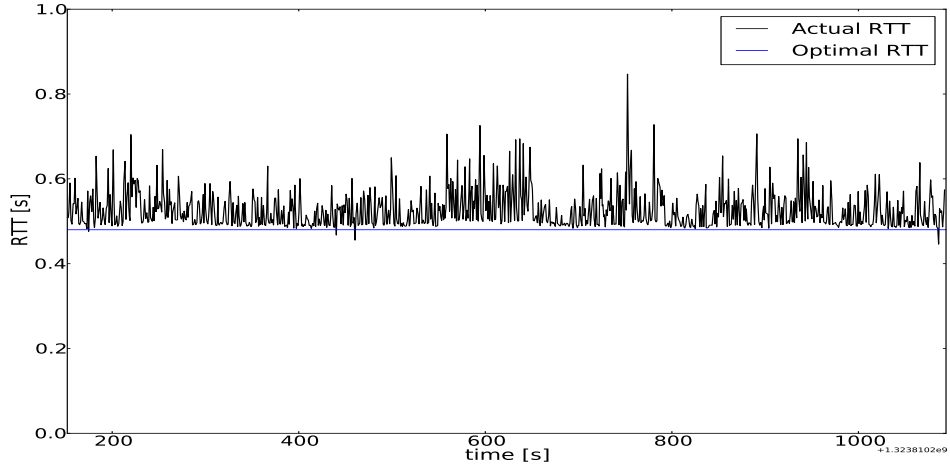
Figure 51: Time charts for 100ms delay variable link capacity scenario



(a) Sending rate



(b) Goodput



(c) RTT

Figure 52: Time charts for 240ms delay variable link capacity scenario

7.4.3 Constant link capacity with two RTP flows

Emulation results for 50ms bottleneck delay:

In the 50ms delay scenario, it can be stated that both flows are fair to each other. Both flows take about 1300kbit/s in total, with flow 1 using a bit more than 700kbit/s and flow 2 taking about 620kbit/s. FEC overhead takes about 14% of the sending rate of each flow. Delivery ratio exceeds 99.40%, there no losses, and consequently no recoveries. Discards appear, but they are very rare. PSNR recorded by both flows is similar and equals about 40dB. Overall metrics for this scenario are presented in the table 25.

Similarly to all previous real-world scenarios, results of this experiment are also impacted by the RTT issue in the Dummynet. This problem is visualised in the figures 53c and 53d. It is noticeable in these graphs that actual RTT is much higher than its optimal value, and there is no reason for this behaviour. Figures 53a and 53b presents time charts of a single experiment run.

Table 25: Overall metrics for 50ms delay in two RTP flows competition scenario

	Metric	FBRA	
		avg	dev
RTP flow 1	**Sending rate [kb/s]**	709.86	206.80
	Goodput [kb/s]	703.91	205.21
	Delivery ratio [%]	99.16	0.95
	FEC rate [kb/s]	95.90	22.26
	No. of lost packets	0	0
	No. of discarded packets	31.70	44.85
	Recovered packets	0	0
	PSNR [dB]	40.59	3.01
	Lossless PSNR [dB]	41.92	2.97
RTP flow 2	**Sending rate [kb/s]**	622.17	206.75
	Goodput [kb/s]	618.98	205.77
	Delivery ratio [%]	99.49	0.49
	FEC rate [kb/s]	89.76	20.78
	No. of lost packets	0	0
	No. of discarded packets	11.40	9.29
	Recovered packets	0	0
	PSNR [dB]	40.26	2.09
	Lossless PSNR [dB]	41.58	2.11

Emulation results for 100ms bottleneck delay:

In the 100ms delay scenario, overall throughput of both flows is higher than in the previous case. Flows are fair to each other, as the first flow takes about 750kbit/s on average, and the second one occupies about 670kbit/s. No packet losses are reported, and discards are low for both flows. Average delivery ratio is slightly below 99%. FEC overhead decreases in this scenario by 1% to 13%. PSNR

results are similar for both flows and is about 38dB. All these results are presented in the table 26.

RTT issues are visible in the time charts presented in the figures 54.

Table 26: Overall metrics for 100ms delay in two RTP flows competition scenario

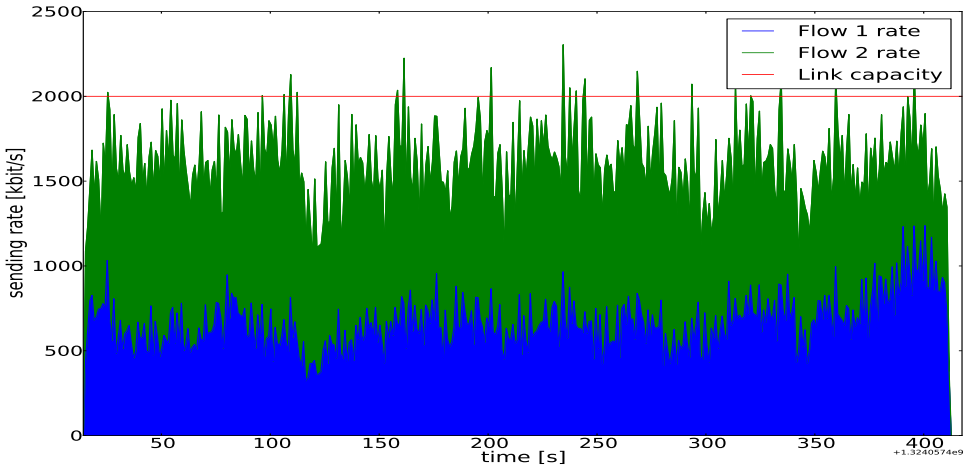
	Metric	FBRA	
		avg	dev
RTP flow 1	**Sending rate [kb/s]**	748.55	85.03
	Goodput [kb/s]	732.63	82.75
	Delivery ratio [%]	97.88	0.55
	FEC rate [kb/s]	95.92	8.59
	No. of lost packets	0	0
	No. of discarded packets	77.0	22.70
	Recovered packets	0	0
	PSNR [dB]	38.13	1.35
	Lossless PSNR [dB]	42.11	1.29
RTP flow 2	**Sending rate [kb/s]**	687.71	79.56
	Goodput [kb/s]	672.06	75.40
	Delivery ratio [%]	97.77	0.90
	FEC rate [kb/s]	88.90	6.44
	No. of lost packets	0	0
	No. of discarded packets	75.80	29.32
	Recovered packets	0	0
	PSNR [dB]	37.57	1.82
	Lossless PSNR [dB]	41.88	1.86

Emulation results for 240ms bottleneck delay:

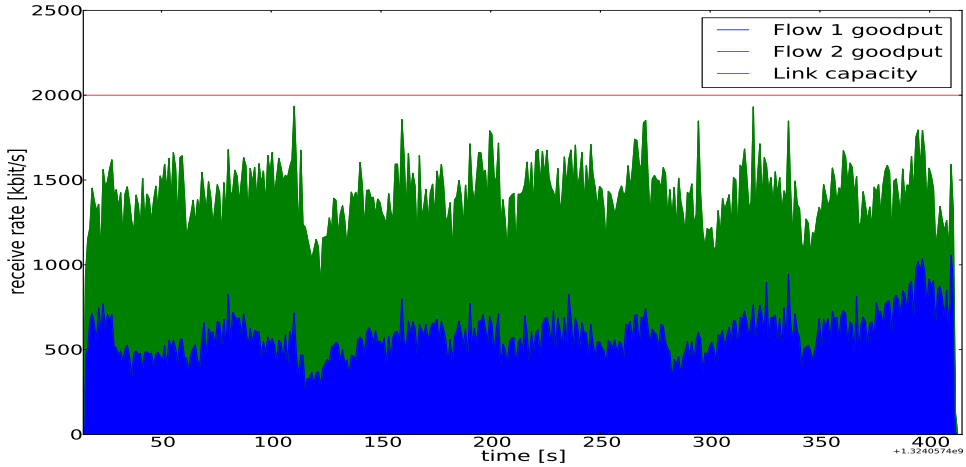
In the 240ms delay scenario, throughput of both flows is significantly lower than in previous cases. This decrease is caused by higher number of discarded packets caused by smaller delay budget in this scenario. Despite the fact that total sending rate of both flows never exceeds the link capacity limit, packet discards appear, and they again results from RTT issue in the Dummynet. Like in previous cases flows are fair to each other with just 30kbit/s difference of sending rates between both flows. Average PSNR value is also lower in this scenario and equals 30.78dB for the first flow, and 29.53dB for the second one. All overall metrics for this scenario are presented in the table 27. Figure 55 illustrates time charts of the one experiment run.

Table 27: Overall metrics for 240ms delay in two RTP flows competition scenario

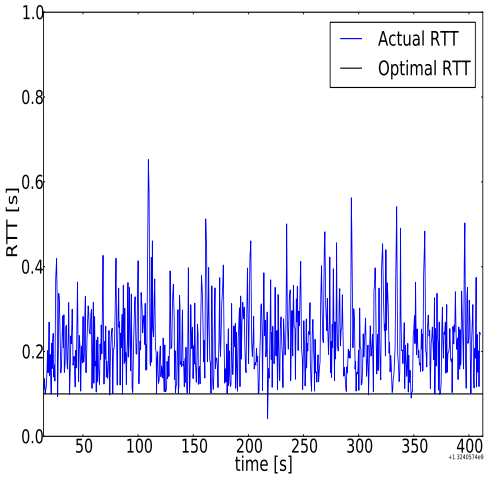
	Metric	FBRA	
		avg	dev
RTP flow 1	**Avg. sending rate [kb/s]**	395.52	88.90
	Goodput [kb/s]	381.63	87.73
	Delivery ratio [%]	96.37	1.07
	FEC rate [kb/s]	58.79	8.86
	No. of lost packets	0.1	0.3
	No. of discarded packets	129.0	34.07
	Recovered packets	0.1	0.3
	PSNR [dB]	30.78	2.52
	Lossless PSNR [dB]	32.11	2.46
RTP flow 2	**Sending rate [kb/s]**	365.26	93.15
	Goodput [kb/s]	350.51	89.51
	Delivery ratio [%]	95.94	0.57
	FEC rate [kb/s]	56.06	9.71
	No. of lost packets	0	0
	No. of discarded packets	128.5	21.88
	Recovered packets	0	0
	PSNR [dB]	29.53	1.21
	Lossless PSNR [dB]	32.01	1.17



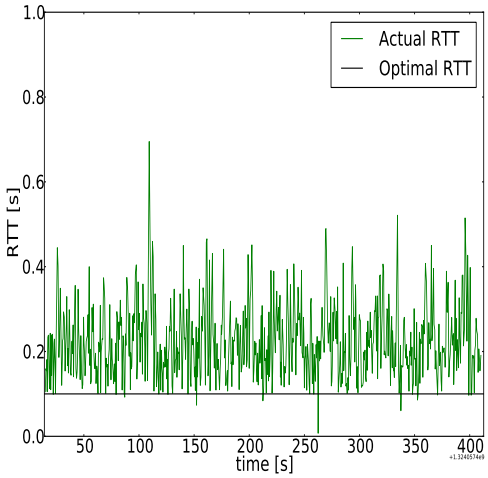
(a) Sending rate



(b) Goodput

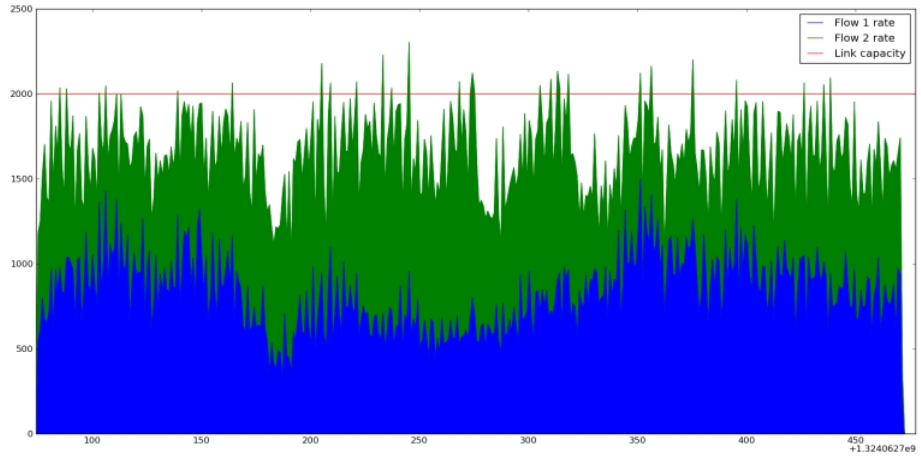


(c) Flow 1 RTT

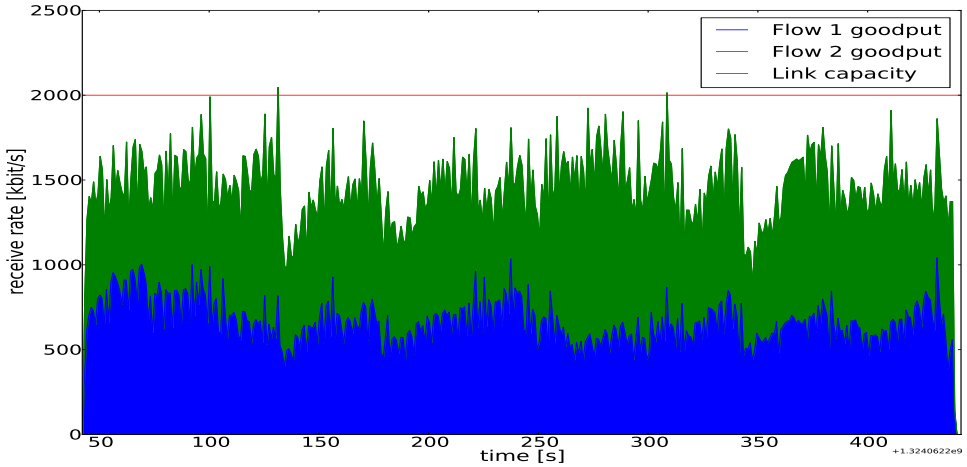


(d) Flow 2 RTT

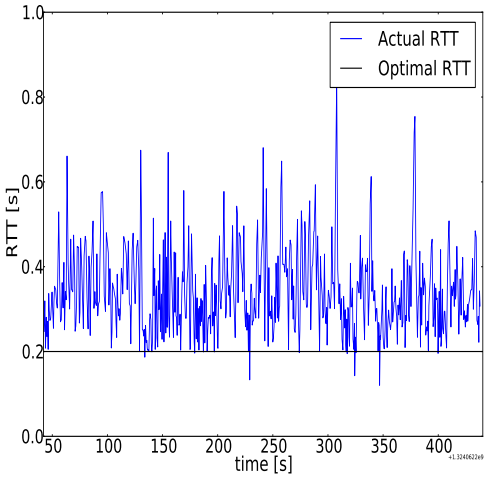
Figure 53: Time charts for 50ms delay RTP competition scenario



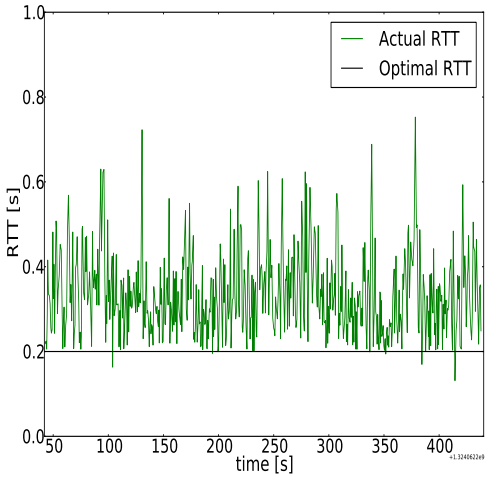
(a) Sending rate



(b) Goodput

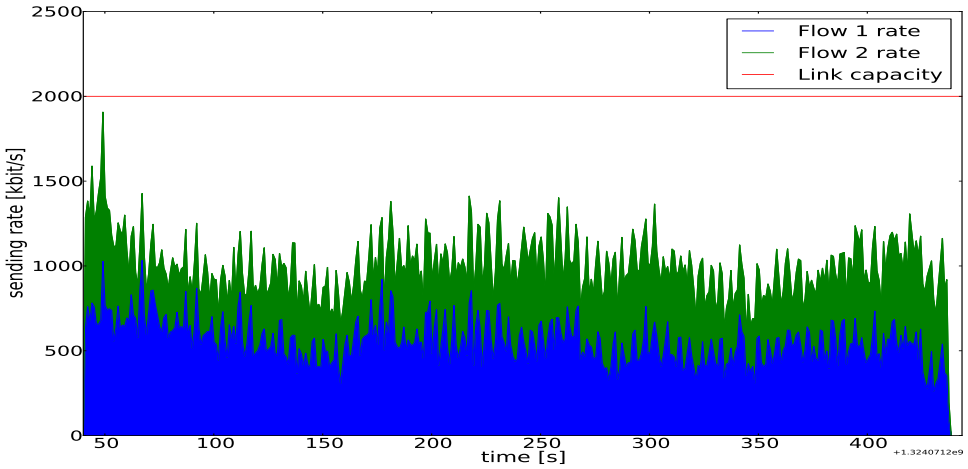


(c) Flow 1 RTT

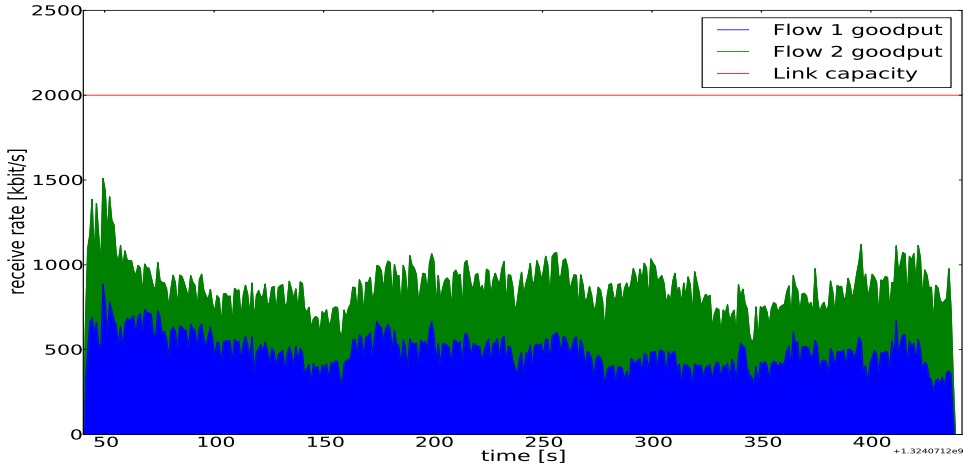


(d) Flow 2 RTT

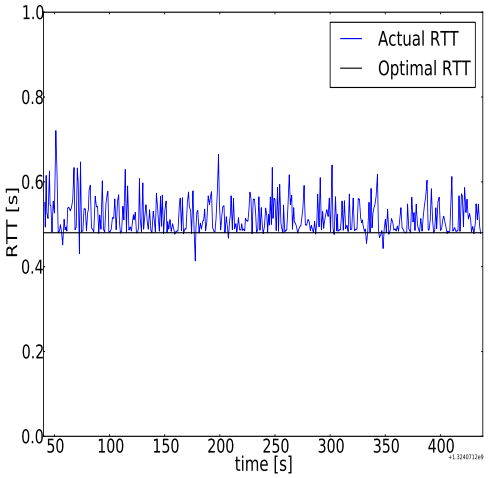
Figure 54: Time charts for 100ms delay RTP competition scenario



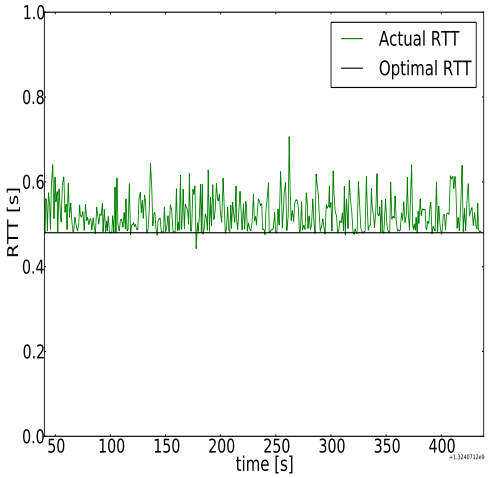
(a) Sending rate



(b) Goodput



(c) Flow 1 RTT



(d) Flow 2 RTT

Figure 55: Time charts for 240ms delay RTP competition scenario

8 Conclusions and future work

In this thesis, possibility of using FEC for rate adaptation together with error recovery was studied. During the work on the thesis the FEC Based Rate Adaptation algorithm (FBRA) was developed. Initially the Non-FEC Based Rate Adaptation algorithm (N-FBRA) was not planned to be developed, however, as the performance of the FBRA was very impressive, the N-FBRA was created from the FBRA by removal of the FEC part. Algorithm evaluation was divided into 2 parts. In the first one, performance of these algorithms was evaluated in the *ns-2* simulator. In the second one, the FBRA algorithm was tested in the Dummynet network emulator by usage of the *AMuSys* platform, which was specially design during the work on this thesis. The N-FBRA algorithm was not evaluated in the *AMuSys*, as the main focus of the work was put on the FEC issues.

Simulations conducted in the *ns-2* simulator show that FEC can be successfully used for both rate adaptation and error recovery simultaneously. In the variable link capacity scenario the FBRA algorithm performs superior to the N-FBRA one. However, in the scenario where single RTP flow competes against many TCP flows the N-FBRA algorithm achieves higher average throughput by a huge margin. Although the delivery ratio in the FBRA algorithm is slightly better, the following question arises. Is it beneficial to use the FBRA algorithm if the N-FBRA can perform so excellently? The FEC component in the FBRA algorithm adds possibility of error recovery in case of overshooting, and by introduction of two additional states ($s+$, $s++$) guarantees more stability between changes of the sending rate. However, as the scenario with constant link capacity and multiple flows competing for the bandwidth is much more realistic than the variable link capacity case, it seems reasonable to conclude that the N-FBRA algorithm is more useful in the Internet environment than the FBRA one.

The Dummynet experiments do not add much to the *ns-2* results because of previously mentioned RTT issue in the Dummynet. This problem prevents the FBRA algorithm from performing at its best and reproduction of the scenarios which were simulated in the *ns-2*. Therefore no exact conclusions can be drawn from this part. However, what can be definitely stated is the fact that the FBRA algorithm correctly reacts to appearing congestion cues.

The question of using FEC for rate adaptation remains unanswered. It was shown that the RTP flow driven by the FBRA algorithm can do very well, but is not able to fight for the network bandwidth as effective as the N-FBRA counterpart. However, it is important to point out that the FBRA algorithm is more reliable due to the FEC addition, and therefore we predict that its value increases as the network link becomes lossier, or less stable. One possible direction of future work is evaluation of the FBRA algorithm in lossier network conditions like mobile or heterogeneous environments. In addition, it would be interesting to check the FBRA algorithm behaviour in case of Active Queue Management (AQM) presence in the network, or if Explicit Congestion Notification (ECN) for RTP is used. Finally, for the Internet environment, it seems sensible to continue work on FEC usage by creating a combination of these two algorithms. The newly created algorithm should

be able to sense the network conditions and based on them use either the FBRA algorithm, or the N-FBRA one. In case of change of condition, it should be able to perform a smooth switch from the one algorithm to the other.

References

[1] Sandvine. [Online]. Available: http://www.sandvine.com/

[2] J. Nielsen. Nielsen's Law of Internet Bandwidth. [Online]. Available: www.useit.com/alertbox/980405.html

[3] V. Singh, "Rate-control for H.264 Conversational Video Communication in Heterogeneous Networks." Aalto University, Master thesis, 2010.

[4] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Application (RFC 3550)," July 2003.

[5] J. Ott and V. Singh, "Networked Multimedia Protocols and Services course."

[6] A. Li, "RTP Payload Format for Generic Forward Error Correction(RFC 5109)," December 2007.

[7] T. Friedman, "RTP Control Protocol Extended Reports (RTCP XR) (RFC 3611)," November 2003.

[8] L. Gharai and C. Perkins, "RTP with TCP Friendly Rate Control (draft-gharai-avtcore-rtp-tfrc-00)," March 2011.

[9] comScore. [Online]. Available: http://www.comscore.com/

[10] You Tube. [Online]. Available: http://www.youtube.com/

[11] Cisco. [Online]. Available: http://www.cisco.com/

[12] Skype. [Online]. Available: http://www.skype.com/

[13] Gizmo. [Online]. Available: http://www.google.com/gizmo5/

[14] BitTorrent. [Online]. Available: http://www.bittorrent.com/

[15] J. Nagle, "Congestion Control in IP/TCP Internetworks (RFC896)," January 1984.

[16] V. Jacobson, "Congestion avoidance and control," *SIGCOMM*, 1988.

[17] P. S. F. L. A. R. S. M. M. Mathis, J. Mahdavi, "TCP Selective Acknowledgment Options (RFC 2018)," October 1996.

[18] V. Jacobson, LBL, R. Braden, ISI, D. Borman, and Cray Research, "TCP Extensions for High Performance (RFC 1323)," May 1992.

[19] S. McCanne and S. Floyd, "ns-2: Network Simulator." [Online]. Available: http://www.isi.edu/nsnam/ns/

[20] ITU-T, "ITU-T Recommendation H.264 (2003): Advanced video coding for generic audio-visual services ISO/IEC 14496-10:2003: Information Technology - Coding of audio-visual objects Part 10: Advanced Video Coding," ITU-T, Tech. Rep., 2003.

[21] S. Wenger, "H.264/AVC over IP," *IEEE Transactions on Circuits and Systems*, vol. 13, no. 7, July 2003.

[22] S. Wenger, M. Hannuksela, T. Stockhammer, M.Westerlund, and D.Singer, "RTP Payload Format for H.264 Video (RFC 3984)," February 2005.

[23] B. Briscoe, "A Fairer, Faster Internet Protocol," *IEEE Spectrum Magazine*, vol. Dec 2008, pp. 38–43, dec 2008.

[24] Eli Brosh, Salman A. Baset, Dan Rubenstein, and Henning Schulzrinne, "The Delay-Friendliness of TCP," in *SIGMETRICS'08*, Annapolis, Maryland, USA, June 2008.

[25] V. Balan, L. Eggert, S. Niccolini, and M. Brunner, "An Experimental Evaluation of Voice Quality over the Datagram Congestion Control Protocol," *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE.*, pp. 2009–2017, May 2007.

[26] J. Ott, S. Wenger, N. Sato, C. Burmeister, and J. Rey, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP) - Based Feedback (RTP/AVPF) (RFC 4585)," July 2006.

[27] C. Borman, L. Cline, G. Deischer, and T. Gardos, "RTP Payload Format for the 1998 Version of ITU-T Rec. H.263 Video (H.263+) (RFC 2429)," October 1998.

[28] J. Rey and D. Leon, "RTP Retransmission Payload Format (RFC 4588)," July 2006.

[29] ITU-T, "ITU-T Recommendation H.263 Video Coding for Low Bit Rate Communication," *wwwituintrec*, November 2000.

[30] J. Rosenberg and H. Schulzrinne, "An RTP Payload Format for Generic Forward Error Correction (RFC 2733)," December 1999.

[31] J. Devadoss, V. Singh, C. Liu, Y. Wang, J. Ott, and I. Curcio, *Evaluation of Error Resilience Mechanisms for 3G Conversational Video*. Tenth IEEE International Symposium on Multimedia, December 2008.

[32] V. Singh, J. Ott, and I. Curcio, "Rate Adaptation for Conversational 3G Video," *Communications Society*, pp. 1–7, 2009.

[33] S. Floyd and M. Handley, "TCP Friendly Rate Control (TFRC): Protocol Specification (RFC 5348)," September 2008.

[34] S. Wenger and U. Chandra, "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF) (RFC 5104)," February 2008.

[35] J. Ott and I. Curcio, "Real-time Transport Control Protocol Extension Report for Run Length Encoding of Discarded Packets (draft-ott-avt-rtcp-xt-discard-metrics-02)," November 2010.

[36] G. Hunt, "RTCP XR Report Block for Discard metric Reporting (draft-ietf-avt-rtcp-xr-discard-02)," May 2009.

[37] B. Ngamwongwattana and R. Thompson, "Measuring One-Way Delay of VoIP Packets Without Clock Synchronization," in *2009 IEEE Intrumentation and Measurement Technology Conference*. IEEE, 2009, pp. 532–535.

[38] B. Ngamwongwattana and R. Thompson, "Sync & Sense: VoIP Measurement Methodology for Assessing One-Way Delay Without Clock Synchronization," *IEEE Transactions On Instrumentation And Measurement*, vol. 59, no. 5, pp. 1318–1326, 2010.

[39] E. Ong, X. Yang, W. Lin, Z. Lu, S. Yao, X. Lin, S. Rahardja, and B. Seng, "Perceptual quality and objective quality measurements of compressed videos," *Journal of Visual Communication and Image Representation*, vol. 17, no. 4, pp. 717–737, 2006.

[40] ITU-T, "ITU-T J.247 Objective perceptual multimedia video quality measurement in the presence of a full reference," *wwwituintrec*, 1999-2008. [Online]. Available: http://www.itu.int/rec/T-REC-J.247/en/

[41] Z. Li, "A saliency map in primary visual cortex," *Trends in Cognitive Science*, vol. 6, no. 1, pp. 9–16, January 2002.

[42] S. Floyd, "TCP Friendly Rate Control (TFRC): The Small-Packet (SP) Variant (RFC 4828)," August 2007.

[43] S. Floyd and E. Kohler, "Internet Research Needs Better Models," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, January 2003.

[44] M. Montagud and F. Boronat, "A new network simulator 2 (NS-2) module based on RTP/RTCP protocols to achieve multimedia group synchronization," in *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, ser. SIMUTools '10, ICST, Brussels, Belgium, 2010.

[45] S. Floyd, "Difficulties in Simulating the Internet," *IEEE/ACM Transactions on Networking*, vol. 9, no. 4, August 2001.

[46] "GStreamer Open Source Multimedia Framework." [Online]. Available: http://gstreamer.freedesktop.org/

[47] JRTPLIB. [Online]. Available: http://research.edm.uhasselt.be/~jori/

[48] "Dummynet Network Emulator." [Online]. Available: http://info.iet.unipi.it/~luigi/dummynet/

[49] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, October 2004.