NTNU
Innovation and Creativity

Rate Adaptive Video Streaming over Wireless Networks

Explained & Explored

Torgeir Haukaas

Master of Science in Communication Technology
Submission date: June 2007
Supervisor: Yuming Jiang, ITEM
Co-supervisor: Arne Lie, Sintef

Norwegian University of Science and Technology
Department of Telematics

Problem Description

Through the use of the "Evalvid-RA" software package (http://www.item.ntnu.no/~arnelie/Evalvid-RA.htm), large networks supporting rate adaptive packet video can be simulated on a single computer. Typically, Internet congestion control mechanisms assume packet loss events are due to some link capacity contention. When one or more hops are wireless, packet loss events may have many other reasons. In such cases, the normal source rate control behavior may not be desirable. This project puts focus on how rate adaptive media will react when run over error prone wireless links. TFRC (DCCP) and
P-AQM are congestion control methods that Evalvid-RA currently supports. The student may also suggest modifications to make media transport over such networks more robust. WLAN shall be modeled using available packet drop modules.

The student must have a computer with ns-2 and Evalvid-RA installed, and some Internet protocol and programming skills. There exists written detailed background information other than provided on the web-page.

Assignment given: 17. January 2007
Supervisor: Yuming Jiang, ITEM

Abstract

In the search for improved methods of transporting real time video and VoIP over the Internet, new rate control mechanisms need to be developed and standardized. Ongoing research in the field has led to several new protocol suggestions. This thesis covers simulations of rate adaptive video transfers over wireless networks using one of them, TCP-Friendly Rate Control (TFRC). It is known that both TCP and TFRC have performance issues in wireless networks because their rate controllers react to random packet error as if it were congestion. On the other hand, UDP as used today for many real-time services, endanger the health of the network with unfair occupation of network resources. Moreover, the shared medium topology of many wireless networks add significant delay to the data transport. Through the use of *ns*-2 and Evalvid-RA, several simulation scenarios have been formed to illustrate how these problems affect real-time video as perceived by a wireless end user. The simulations illustrate the missing inter- and intra-flow fairness of UDP, reveal protocol and implementation vulnerabilities of TFRC, and show a remarkable delay caused by the IEEE 802.11b networks. The thesis also includes a study of past and current protocol refinement proposals and suggestions to novel re-design. The study reveals that the variety of proposals form an information stream that is difficult to follow, a scarcity of problem domain overview and decisive variations in the problem definition.

Preface

This is the master thesis for the degree MSc. in Communications Technology, carried out Spring 2007 at the Department of Telematics (ITEM) at the Norwegian University of Science and Technology (NTNU) in cooperation with Professor Yuming Jiang and Sintef-researcher and Ph.D. candidate at ITEM, Arne Lie.

It is a continuation of my Project Report ”Next Generation Internet and support for Streaming Media: Trace driven simulation” from Fall 2006, and it is aimed to go deeper under the surface of nearly all topics discussed in that report.

The thesis is written to be understandable for a fellow student within the fields of electronics, networking and computer science. If the reader is interested in the code and scripts used, they are attached in the back of this document together with a list of abbreviations.

I would like to thank Professor Yuming Jiang for support, and supervisor Arne Lie for invaluable help, support and patience during my work.

Contents

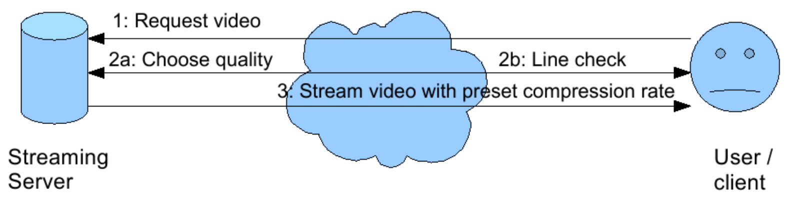
CHAPTER 1

Introduction

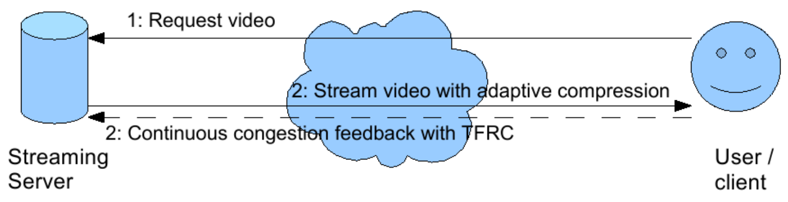
The available hardware and technology for consumers and service providers today allow for advanced multimedia services over IP-based networks. Hence, the popularity of video and audio streaming services such as Video-on-Demand (VoD), advanced on-line gaming, and video chatting and conferences are increasing. The demand for resource efficiency and robustness in the network follows. The current commercially available data transfer technology for streaming media does not adjust well to the best effort heterogeneous Internet architecture, and QoS demands are impossible to guarantee. A central problem with this development is that the Internet itself is extremely dynamic and unpredictable, while the real-time services we want to lay on top of it demand stable conditions for constant and uninterrupted throughput.

It is not only the demand for QoS that increases. As CPU and memory get physically smaller and more efficient, the number of applications that can be run on small devices increases as well. With this new flora of handheld computers, PDAs and cell phones, the wish for terminal mobility and easy wireless access to the same advanced services follows. Now, while the Internet itself has unpredictable throughput, the wireless access networks add even more uncertainty to the QoS-possibilities.

Most Internet based VoD services today rely on pre-compressed media, allowing consumers to choose between two or three different qualities to fit the given end to end connection (fig. 1.1(a)). For video conferencing and live streaming, the media is compressed on the fly, but the compression rate is normally controlled by an *offline* codec. This means that the video codec has no direct contact with the network and transport layer, where the actual transmission status is found. Thus, these approaches do not fully utilize the available resources at any given time and they might not provide the chosen quality during the entire session because of sudden changes in the network load somewhere along the path. With an *online* codec solution, the communicating parties of a multimedia transfer could adopt to the network state at any given time in the session. This is extra desirable for a mobile terminal which has a constantly changing radio coverage, or for a system that supports session-mobility, allowing the user to change terminal equipment



(a) Today


1: Request video
2: Stream video with adaptive compression
2: Continuous congestion feedback with TFRC
Streaming Server
User / client

(b) Tomorrow

Figure 1.1 Typical streaming service with (b) and without (a) congestion feeback.

without interrupting or restarting an ongoing session. Figure 1.1(b) illustrates the concept behind a solution based on continuous network feedback upstream from the client to the media server.

1.1 Problem

In the development of robust, scalable and intelligent new technologies for multimedia transfer over present and future wireless IP-networks, many concerns need to be taken into account. As will be further discussed in chapter 5, the solutions can be found by improvements in many levels of the ISO stack, or by novel protocol design. The challenges include:

- Calculating desired rates from current status and collected statistics.
- Transporting network feedback to the sender.
- Interpreting the feedback and notifying the transmitting application.
- Scaling the multimedia output rate from the sender.
- Providing stable transmission conditions for mobile users.
- Minimizing delay and jitter for real-time applications.
- Staying fair to concurrent traffic on the link.
- Differentiating between congestion loss and link error loss.

All of the challenges above have been addressed by separate research groups around the world, but where one solution is brilliant in the aspect discussed in that study, it fails to solve problems discussed in another. This thesis aims to summarize the problems found and the suggested solutions, and to account for the complexity of multimedia streaming over wireless networks. An

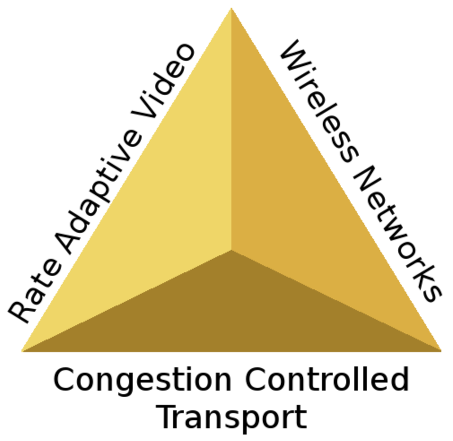


Figure 1.2 The combination of these three technologies is the essence of the problem domain.

extensive part of the thesis is used to explain how the technologies can be tested with simulations. Illustrations of the features and problems of current and new technologies will be presented through the described simulation framework.

1.2 Limitations

The topic of this thesis touches all protocol layers involved in a video streaming service. However, the presented simulation, status summary and analysis have limitations both in scope and possibilities.

1.2.1 Scope

The methods, mechanisms and protocols that are presented, tested and analyzed are discussed given the need and demand for a set of services with a certain quality. How the services are built up, and how the technology will and can be used in different scenarios and settings, is mentioned but not emphasized any further. The focus of this work is mainly on the transport and MAC layers of the protocol stack, and does not prioritize details from the physical or application layers.

The cost-efficiency of each technology in real-life implementation and business-related topics such as time to market, commercial support, industry cooperation and market potential, which are fundamental for a thourough understanding of a standard development process, are left for future studies.

The thesis seeks to enlighten the technological possibilities in terms of quality and performance, but an important factor such as security is left out of the scope. For every new product and service intended for online use today, security is a big concern that may be regarded as a part of the total QoS. Understanding topics such as service authentication, authorization and confidentiality, and how current technology in this field can be integrated with new media transport without breaking their performance and intention, is very important. Without this knowledge, technology that can replace today's well-established protocol stack (or parts of it) may not succeed.

1.2.2 Simulator Accuracy

A significant part of this thesis is the preparation and the simulation of video streaming with TFRC over a wireless network link. The *ns*-2 simulation environment, as described in section 2.4.3, is a complex mixture of code languages and application program interfaces (APIs). It's manual [1] is extensive but the content is partially outdated or inadequate[1]. The user interface to set up and control a *ns*-2 simulation is the Tcl scripting language (described in 2.4.7). However, to understand how the different parameters are set and how they work, the user is often forced to read the C++ source code due to the lack of sufficient documentation.

Many standard simulator functions can be learned in practice by reading and using Tcl-example files that are bundled with the *ns*-2 simulator. Alas, the quality and correctness of these scripts does not seem to have been thoroughly evaluated in all cases[2]. The user can often observe the simulator to crash because of bad parameter settings or input/output problems. Errors are seldom caught in a proper manner, and the user is not always informed what caused the error.

The mentioned complexity, improper documentation and scripts combined with untidy and experimental source code, gives the *ns*-2 simulator an advanced entry-level for students, scientists or other users. In the work presented in chapter 3, *ns*-2 add-ons are used to enable video trace simulations. This code is not widely used and tested, and is known to be highly experimental. One consequence of these factors is that it is time consuming to get a simple simulation running, but a more severe consequence is that it is very difficult for the user to ensure that obtained results are *correct*.

In [2] three different simulator environments, including *ns*-2, are used to simulate a simple mobile ad-hoc network. The paper concludes that the results differed both quantitatively and qualitatively to such an extent that the results from one single simulation environment does not have the credibility needed in serious science and development. Allthough this does not directly imply that *ns*-2 is inaccurate, it is a reminder that only real-life use and realistic testing can *prove* the functionalities or dysfunctionalities of new technology. In the light of all the above, it is not guaranteed that the results presented in this thesis are 100 % correct. In sections 2.4.3, 3.4, 3.6 and 4.1, the functionality, use and bugs of *ns*-2 will be further discussed.

1.3 Related work

The Internet research community is trying to find the best way to improve the quality and availability of video streaming services. One central topic for discussion is the differing operation of TCP and UDP traffic in the IP networks. The differences in their ways of adjusting to router congestion can result in situations where UDP traffic dominates the use of router resources since

[1]paragraphs like this can be found in the middle of the text: "but why/where is this information used??-answer awaited from CMU."

[2]comments like this can be found in the embedded scripts: "Poor hack. What should we do without a link object??"

it does not adjust the transmission rate automatically according to the given conditions. In [3], Floyd and Fall promotes the research on and use of protocols with congestion control. They introduce the term “TCP-friendliness” (which will be further discussed in section 5.1) and, as the article title says, promote “the use of end-to-end congestion control in the Internet”. The research effort has led to several suggestions to new congestion control mechanisms such as VBR over ATM ABR services [4], RAP [5], MPEG-TFRCP [6], LDA+ [7] and P-AQM+ECF [8], the latter being a NTNU proprietary protocol. Among the suggestions, TCP-Friendly Rate Control (TFRC) [9] has gained most attention and become a IETF standard that is intended to be used together with Datagram Congestion Control Protocol (DCCP, see section 2.5).

In [10] and [11], Evalvid, a framework for quality evaluation of streaming video is presented and tested. In [12] this framework (now called Evalvid-RA) is extended to work with rate adaptive video and with network and transport protocols based on transmission feedback from the client or the network. In [13], which is a pre-study to this master thesis, the Evalvid-RA framework is tested over an erroneous link simulated by a simple stochastic packet dropping mechanism. The results from the latter study show that the TFRC has performance problems when the link has random losses. This motivates for further study of TFRC over a more realistic wireless interface; The IEEE 802.11 MAC wireless protocol.

TFRC is designed with a rate adoption very similar to the one in TCP in order to be TCP-Friendly. In many studies such as [14], [15] and [16], the performance issues of TCP in wireless scenarios with random packet losses are discussed. The studies in this field has led to several suggestions to improved congestion control in TCP (i.e. TCP Westwood [17]). It is natural to assume that TFRC suffers from the same problems as TCP in the presence of wireless random errors.

A further discussion and analysis of related work, which is a central part of this thesis, can be found in Chapter 5.

1.4 Method and Structure

As seen in 1.1 and 1.3 above, there are many past and ongoing studies on the challenges regarding real time multimedia transport and QoS in wireless networks. This thesis will address some of these problems. While the TFRC protocol mainly has been tested and simulated over wired networks, this thesis presents simulation results over a realistic wireless access network. The Evalvid-RA framework is used to show realistic simulations of variable bit rate (VBR) video transfer. While many other studies have throughput and packet loss as the main evaluation criteria, this thesis uses measurements from the transferred and decoded video as evaluation criteria to see how the losses and rate adjustments affect it in practice. The simulations based on a combination of video trace driven simulation with rate adaptive video, transport protocols with congestion feedback and 802.11 wireless networks (fig. 1.2), results in the first out of two main contributions of this thesis. The simulations presented, is mainly intended to illustrate the benefits and disadvantages of TFRC compared to TCP and UDP in a realistic environment. The second main contribution is the analysis and comparison of different approaches and works surrounding the field of video transfer and QoS in wireless networks.

The work presented in this thesis consists of four main parts. In Chapter 2, some background information and relevant theory is gathered to ease the understanding of the following discussion. The environment, coding languages, codecs and protocols used in the practical work are described. The reader will also find relevant theory to the discussion on standard development in Chapter 5. In Chapter 3, the implementation and simulation setup is described for three different scenarios, while their results and analysis are found in Chapter 4. In Chapter 5, the problems, solutions, barriers, possibilities and trade-offs of different novel technologies in the domain are presented, before the thesis is summed up with the conclusions in Chapter 6.

CHAPTER 2

Background

In this section, background information on used tools, languages and theory will be presented. It is a brief overview, explaining only the terms that are used in this thesis and the technology and environment used in the simulations. Some of the sections in this chapter contain simulation specific background information, whilst others are background theory for the discussions in Chapter 5. The level of detail is sufficient to understand the further discussion without the need of pre-studies.

2.1 MPEG-4 Video

MPEG-4 video compression, as defined by the ISO/IEC Moving Picture Experts Group, is a standard for compressing and decompressing digital video aimed for distribution over media with limited bandwidth. The MPEG-4 standard suite includes 23 *parts* describing different aspects of multimedia compression and transfer. For instance, part 3 defines how to compress and encode audio to AAC-streams, and part 12, 14 and 15 describe the file and container formats. However, only two of the 23 parts describe how the raw video is encoded into a compressed data-stream suitable for network transfer. These are part 2 and part 10. The latter describes the newest and most advanced codec, namely Advanced Video Coding (AVC), often referred to as H.264[1]. The former is the one most commonly referred to as ”MPEG-4 video”. This is a coding scheme with several different *profiles*, where the most normal are Simple Profile and Advanced Simple Profile (ASP). In the rest of this document, the term MPEG-4 will refer to the standardized MPEG-4 part 2 ASP.

[1]In fact, the MPEG-4 part 10 does not describe H.264, but it is technically identical to H.264 which is defined by ITU-T

2.1.1 Uncompressed video

Raw video consists of a series of digital photos, each of them consisting of a grid of pixels (e.g. CIF 352x288 pixels). Each pixel is represented with three values of typically one byte each in the RGB format, which enables presentation of 256 intensities of each hue. Uncompressed, this produces a bit-stream that is impractical to store, process and distribute over normal available computers and network equipment due to its size. The following example shows the calculated size of one minute of CIF RGB video:

$$3\text{colors} \times 1\text{ byte/color} \times (352 \times 288\text{ pixels}) \times 25\text{ fps} \times 60\text{ seconds/minute } = 435\text{ MB/minute}$$

A more space efficient alternative to storing a video with three full colors, is to store it in the YUV format. This format models the human perception of color better. It stores the *luminance* and *chrominance* in different layers (given by Y, Cb and Cr in digital systems). Since the luminance is easier percepted by the human eye than chrominance, the latter need not be stored with equal resolution. Typical YUV subsampling schemes is YUV 4:2:0 and YUV 4:1:1. This notation indicates varying ways to arrange the chrominance and luminance layers in each frame. With both of these schemes, the number of bits needed for each pixel is reduced from 24 (RGB: 3x8) to 12. The uncompressed original and reconstructed videos in this work are in YUV 4:2:0 format. The size reduction introduced by YUV compared to RGB, which is of factor two, is not enough to make a video stream easy to handle for computers and networks. The video compression used in this project is described in the following.

2.1.2 MPEG frame types

With MPEG-compressing, none of the pictures in the raw-stream are stored or sent in its original form. Typically, the series of pictures or *frames* are divided into Groups Of Pictures (GOPs, see figure 2.1). The frames are then coded in various ways within each GOP. There are normally three types of frames in a MPEG GOP:

Intra coded frames (I) These are coded as single frames as in JPEG. They do not have references to any other frames and are relatively large.

Predictive coded frames (P) Are coded as the difference from a motion compensated prediction frame[2], generated from an earlier I-frame in the GOP. They are typically smaller than the I-frames, but bigger than the B-frames.

Bi-directional coded frames (B) Are coded as the difference from a bi-directionally interpolated frame, generated with motion compensation from earlier and later I or P frames in the sequence. B-frames are typically smaller than both I and P-frames.

The most important thing to note about this structure when talking about Quality of Service, is the fact that the I-frames are much more important to the quality of the video than the P-frames, which in turn are more important than the B-frames. Now, if a I-frame is lost during transmission

[2]Motion compensation basically means that instead of storing pixel-values, the difference vector from a reference frame is stored

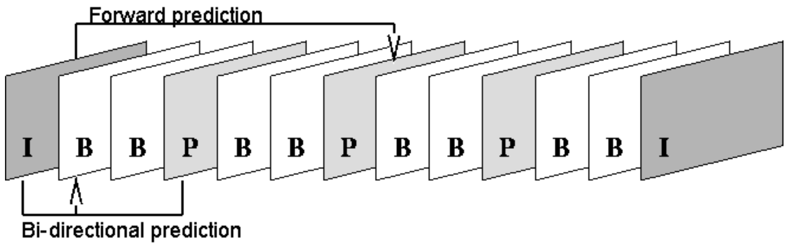


Figure 2.1 A typical MPEG GOP [18]

due to congestion or erroneous links, the impact on the result is big. However, smart decoders are able to reconstruct a usable image even if one or several packets from one frame is lost as long as the packet containing the frame header is received. The number of packets needed to transfer one frame depends on the ratio between frame-size and the packet transfer unit size, called Maximum Transfer Unit (MTU). The overall result can be seen on measures such as the Peak Signal-to-Noise ratio (PSNR, see section 2.2), Mean Opinion Score (MOS) or simply by watching the video in a MPEG-compatible player[3].

2.1.3 Quantization

The image compression of an I-frame is very similar to that of a JPEG-image. The image is split into smaller blocks (of e.g. 8x8 pixels), processed in a discrete cosine transform, quantized and entropy encoded. The quantization process is the one introducing the loss to the image quality in the encoding process. Hence, in addition to resolution and frame-rate (frames per second, fps), the number of quantization levels is the best indicator of the total bit-stream and the quality of the resulting compressed video. In fact, the quantization parameter is often referred to as a *quality*-parameter. The encoding of P- and B-frames, which include both JPEG-like encoding and motion prediction, is often referred to as a *hybrid encoding* (figure 2.2).

After the discrete cosine transform (DCT), each block of each layer Y, U and V of a frame can be represented in the frequency domain as a coefficient matrix. In the quantization process, each of these coefficients get divided by a quantization factor given in a quantization matrix. This matrix is predefined based on a quantization parameter, and ordered so that the least significant coefficients are reduced more than the most significant coefficients. In this way, the data that is most frequent in the frame gets the highest degree of detail, or *depth*, while other components are compressed harder. This process is not reversible (the data can not be 100 % restored), hence causing direct quality loss to the compressed data. However, with intelligent design of the quantization matrices, the sensibility of a human eye can be taken into account so that the loss is minimized [20].

[3]VideoLAN Client (VLC) is used for viewing compressed video in this work

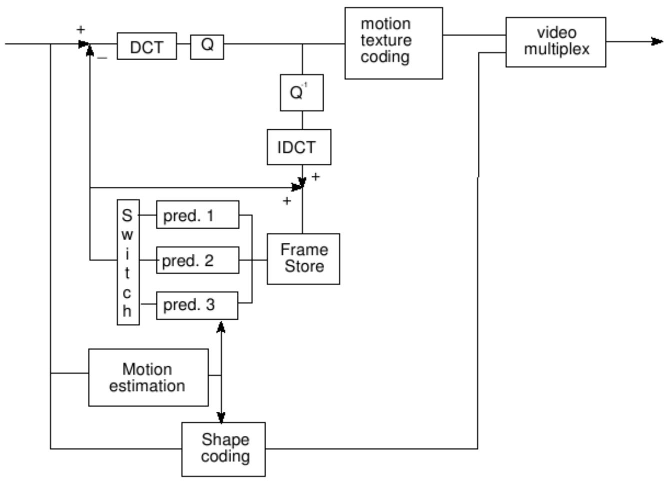


Figure 2.2 Basic block diagram of a MPEG-4 video encoder [19]

2.2 Quality Measure: PSNR

To measure the quality of a network service for video streaming, it is not sufficient to look at classical QoS terms such as jitter, delay and throughput. The way a video sequence is encoded with different frame types, and how it is split into smaller packets, results in a variable bit rate stream[4] where it is not irrelevant *which* packets that are lost. So the loss patterns of a congested router or an erroneous link can result in different video qualities for the same average loss rate. In the end, it is the perceived quality impression of the user that counts. Both subjective and objective approaches to quality measurements exists, but where subjective methods are expensive and requires much resources, objective measurement with the Peak Signal-to-Noise Ratio (PSNR) can be automated and simple [11].

To quantify the observed losses from compression, decompression and transfer of a video, a PSNR value can be calculated for the given video sequence. This value is the ratio between the maximum power of the original signal (data from the original video) and the power of the interfering noise that is introduced in the process. Because the signal quality can have a very wide dynamic range, the PSNR values are normally given in a logarithmic decibel scale.

The PSNR values used in this work, are calculated as the mean of the PSNR values for each frame in the video sequence. The PSNR for one frame is calculated by comparison with the respective original frame. First, the Mean Square Error (MSE) between each layer Y, Cr or Cb[5] is calculated as

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2 \tag{2.1}$$

[4]The term Constant Bit Rate (CBR) means that the rate is held constant over time, but will normally have bitrate peaks for each I-frame.

[5]Allmost all quality assessment studies that use PSNR, only calculate on the luminance layer (Y) since it is the most relevant to the quality perception[11].

where I and K are frame layers with resolution $m \times n$. Then the PSNR is defined as

$$PSNR = 20log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \tag{2.2}$$

where MAX_I is the maximum allowed pixel value. The PSNR introduced by a hybrid coder is normally in the range 30 to 40 dB when providing good image or video quality. Values below 30 dB are not necessarily satisfactory (depending on the range of application), while values below 20dB represent garbled results ([21], [20]).

2.3 Rate adaptive video

Today, a normal real-time video streaming service will provide the viewer with several different quality choices as parallel unicast streams. Often, the streaming server will perform some probing or end-to-end bandwidth check to be able to suggest a suitable bit-rate for the user. This is not a very elegant approach to the challenge of serving all kinds of users, whether it is a user with optical fiber access or a user with a 3G mobile phone, nor is it suitable for a mobile user who has constantly changing signal strength and bandwidth.

This thesis aims to investigate the robustness of the suggested transport protocols for rate adaptive video. The concept of rate adaptiveness is this: When a video is streamed real-time from a server to a client, the network capacity is reported back to the server continuously. Since it is a real-time service, the server can not simply adjust the bit-rate from its outgoing buffer, as this will break the continuity at the client side if the sent rate is lower than the rate of the compressed video. Therefore, rate adaptive transmitters adjust the *quality* of the video encoding directly instead, thus providing optimal streaming bit rate and full network utilization at any time.

The quality and bit rate of a video transmission can change upon adjustment of several different encoding parameters. These can be the number of frames per second, the frame size or the quantization used in the compression process. Which of these parameters that affect the quality of the resulting video the most depends on subjective measures, the purpose and content of the video and the display on which it is shown. In the Evalvid-RA framework used in this thesis, the network feedback affects the quantization parameter of the compression to adjust the transmitting rate. How this is done is explained in section 2.5.2 and 4.1.

A simplified version of the simulated architecture is given in figure 2.3. It illustrates that several compression qualities are available on the streaming server. These can be available as pre-compressed versions in the server storage, or they can be compressed on the fly with a real-time encoder that supports dynamically changing quality parameters. To store the full range of compression alternatives with the quantization parameter altering from 2 to above 30, some six times the storage space is required compared to only storing the best quality. In a real life implementation, a reduction of the number of quantization steps is more probable. In the case of live streaming, storing the video on disc is not an option since the video should not be buffered more than absolutely necessary. However, when serving Video-on-Demand, it is a trade-off between CPU-usage and storage space on the server(s)[6]. The Evalvid-RA framework is thus

[6]Further discussion of services and implementation issues are out of the scope of this thesis.

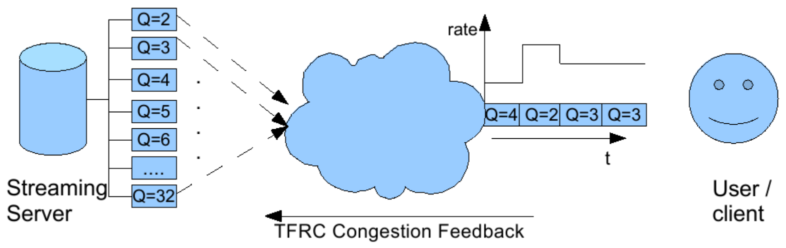


Figure 2.3 The server can provide several compression qualities simultaneously, and decides which one to serve the client based on congestion feedback. The rate indicated in the illustration, is the average bitrate for each quantization level.

mainly constructed to simulate *live* encoding.

2.4 Simulation environment

Testing of new transfer protocols in real networks would require big investments in design and implementation of new router software, in many cases even hardware. Additionally, one would need sufficient equipment with multiple sources of different types to support realistic big-scale end-to-end cases for heterogeneous networks. Since this is equipment that might crash during tests, this must be done on a separate network without any commercial or important traffic. These demands, among others, make real testing with physical data transfer too costly for most research institutions, at least in the early phases of development.

A local simulation environment enables the researcher to build customized fictive networks at a low cost. The simulations in this study are carried out on a single standard computer without any special hardware or costly software. The details are explained in this section.

2.4.1 Hardware

The only necessary hardware is a normal modern computer.

Description of the computer used:

- Fujitsu Siemens Computers S-Series Lifebook
- 1.8 GHz Intel Centrino CPU
- 1536 MB DDR RAM (512 MB sufficient)
- 64 MB Graphical memory (not necessary)
- 80 GB harddisc

It is worth mentioning here that if the rebuilt videos from all iterations of cases shall be kept for visual inspection and further analysis, several gigabytes of free space might be necessary. In this study, a YUV-file of 32.7 MB was used. The sum of all tracefiles, mp4-files and recreated YUV-files for all simulation cases sum up to 6.4 GB.

2.4.2 Operating system

Most of the simulation tools described below are parts of open source projects compiled on Unix-like machines using classical Unix development tools. However, they are successfully compiled and used on other platforms as well.

Description of the platform used:

- Kubuntu 7.04 (Feisty Fawn)
- Linux kernel 2.6.20-15 (for i586 architecture)
- Compilators: gcc / c++ 4.1.2
- C-library 2.4
- X-window system (for graphical output and video inspection): KDE 3.5.6

2.4.3 ns-2

The most central part of the simulation environment is the network simulator itself, *ns*-2 [1]. It is a discrete event based simulator supporting simulation of TCP/IP, multicast and unicast, wired and wireless networks, several routing protocols and more. It is a tool developed by the VINT-project [22], released as open source for any researcher to modify to own needs.

ns-2 is an object oriented simulator build on Otcl[7] and C++. The program is constructed with a C++ class hierarchy and a corresponding class hierarchy in OTcl. The Otcl is a front end that enables the user to combine the different C++-implemented objects and do function-calls from Tcl. The protocols and tools available to the user is made available from OTcl and instantiated with C++-classes. Due to this dependency, protocol design or bigger design changes must be done in both languages. *ns*-2 is build with two languages to support speed in data treatment at run-time (C++) as well as easy and fast re-configuration of parameters and settings without the need of recompilation (OTcl) [1].

The ns-software has some package-dependabilities, but all packages may be installed from a bundle containing [8]:

- ns-2.30
- nam-1.12 (see chapter 2.4.5)
- xgraph-12.1 (see chapter 2.4.5)
- tcl 8.4.13 (see chapter 2.4.7)
- tk8.4.13
- otcl-1.12

[7]Object oriented Tcl. Tcl is pronounced ”tickle”, often referred to together with the Tk GUI-toolkit: Tcl/Tk.
[8]for current latest stable build

- tclcl-1.18
- gt-itm (Georgia Tech Internetwork Topology Models)

2.4.4 Evalvid-RA

As mentioned, researchers can modify *ns*-2 code, or add new modules to it. One addition that is needed for these simulations, is the Evalvid-RA package [12]. This package enables simulation of rate adaptive multimedia transfer based upon tracefile generation of MPEG video. This means that instead of using the real binary multimedia content, which is resource demanding and time consuming, the simulation runs with tracefiles generated from the media files beforehand. A typical tracefile holds information on frame number, frame type, size, fragmentation into segments and timing for each video frame. While running, *ns*-2 can keep track of the timing and throughput of packets at each node including the receiver. With this information and the original compressed videofile, Evalvid-RA can rebuild the video the way it would have been received on a real network. The obvious benefit of this is that one can play the video and inspect the result visually. Additionally, the total noise introduced by compressing and transferring the video (measured in dB PSNR, see 2.2) and Mean Opinion Score can now be calculated.

Evalvid-RA is a branch of the open source tool EvalVid, invented by J. Klaue [11]. The addition to the original is made by Arne Lie [12] and consists of functionality for rate adaptive video, including support for congestion feedback mechanisms such as TCP-Friendly Rate Control (TFRC) [9]. The use of the tool-set is illustrated in figure 2.4.

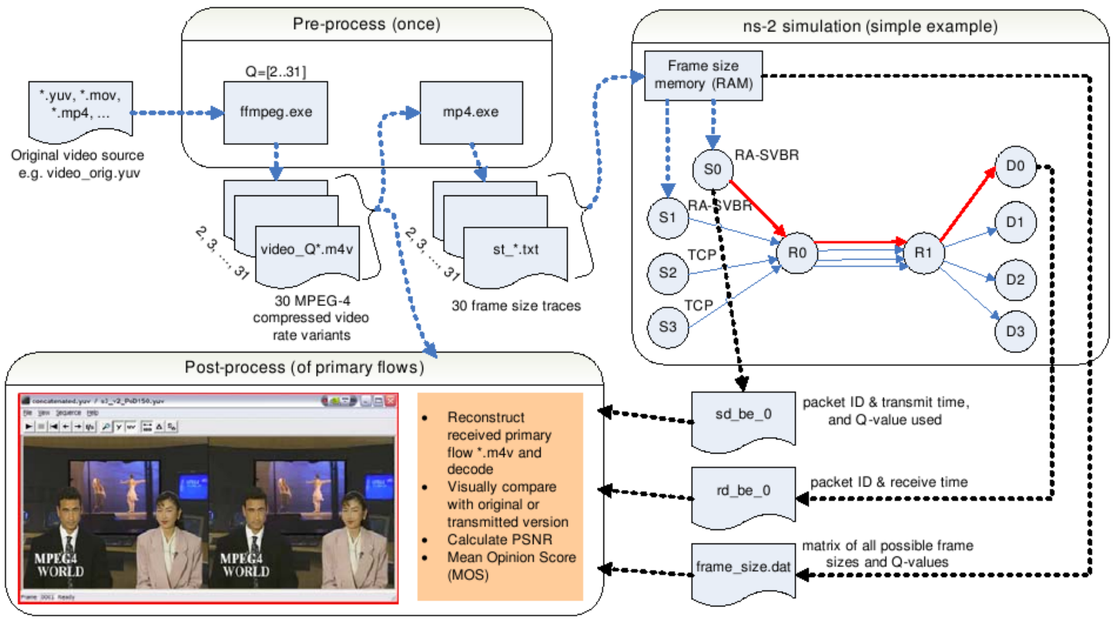

Pre-process (once)
ns-2 simulation (simple example)
*.yuv, *.mov, *.mp4, ...
Original video source e.g. video_orig.yuv
Q=[2..31]
ffmpeg.exe
mp4.exe
video_Q*.m4v
2, 3, ..., 31
30 MPEG-4 compressed video rate variants
st_*.txt
30 frame size traces
Frame size memory (RAM)
RA-SVBR
RA-SVBR
TCP
TCP
S0
S1
S2
S3
R0
R1
D0
D1
D2
D3
Post-process (of primary flows)
Reconstruct received primary flow *.m4v and decode
Visually compare with original or transmitted version
Calculate PSNR
Mean Opinion Score (MOS)
sd_be_0
packet ID & transmit time, and Q-value used
rd_be_0
packet ID & receive time
frame_size.dat
matrix of all possible frame sizes and Q-values
MPEG4 WORLD

Figure 2.4 The pre-process, simulation and post-process of the Evalvid-RA framework.[12]

The Evalvid-RA tool-set includes several smaller programs, namely *mp4*, *et_ra*, *fixyuv_ra* and *psnr*. A short description of each one of them follows.

mp4 is a program used in the pre-process to generate tracefiles including information about each frame in the video. See first GOP of the reference video in code snippet 1.

et_ra (Evaluate Traces - Rate Adaptive) is a program that performs the actual calculation of jitter and packet-loss. Further, it combines the data from input (one mp4-file and one tracefile for each compression level) and output (packet-arrival statistics from the ns-simulation) to form a resulting m4v-video. Due to complication with the decompressor (ffmpeg), this program is tuned to work in *frame*-mode. That means that if the tracefile contains information about one lost packet, the entire frame the packet belongs to is considered lost (more about this in section 4.2.2).

fixyuv_ra is a program that fixes broken parts of the received video. This is necessary because the resulting sequence must be of equal length as the original sequence for objective overall quality-measures. Fixyuv detects missing frames, and duplicates last received frame until new frames arrive (i.e. until the received video is synchronized with the original).

psnr is the last program being used in the simulation and calculation sequence in this thesis. It takes the original YUV-sequence as input and compares it to the recovered and fixed YUV-sequence obtained at the receiver.

For an example of how these tools can be combined, which parameters they take and give, see 3.5, and how everything can be triggered from a shell-script, see Appendix B.

Code Snippet 1 The first 13 frames indicated in a video trace file. The columns are: Frame number, frame-type (H for header), frame size, number of segments (i.e. number of packets for that frame) and offset from start of video (B-frames are not used by FFmpeg in this compression).

```
0     H     52      1 segm  at    0 ms
1     I     9209   10 segm  at    0 ms
2     P     3411    4 segm  at    0 ms
3     P     3992    4 segm  at    0 ms
4     P     3827    4 segm  at    1 ms
5     P     4251    5 segm  at    1 ms
6     P     3817    4 segm  at    1 ms
7     P     4207    5 segm  at    1 ms
8     P     4121    5 segm  at    1 ms
9     P     4057    5 segm  at    2 ms
10    P     4095    5 segm  at    2 ms
11    P     4071    5 segm  at    2 ms
12    P     3994    4 segm  at    2 ms
```

2.4.5 Presentation tools

To analyze the results of a simulation, tools for graphical presentation often make the output more understandable. In this project, the tools *xgraph* and *nam* where used both to present the final results as well as to check correctness of coding and scripting in the development progress.

Xgraph is a simple tool to draw a graph based on text-files with data. It is very useful to indicate for instance if a newly implemented protocol or simulation function works or not (see figure 2.5).

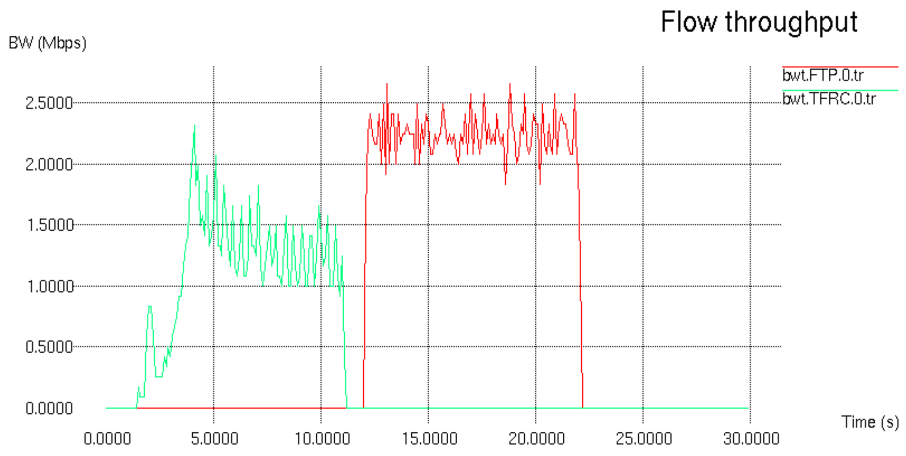


Figure 2.5 An example showing the throughput from a TFRC source (green) and a TCP source (red) at a mobile node using 802.11 wireless interface.

The Network Animator (nam) is a tool that is tightly coupled with *ns*-2 to display the transmission of packets in the chosen network architecture. It is capable of drawing all the nodes and lines that are set up, and animates how the packets flow through the given topology. It can also track a specific packet end-to-end, display link throughput and give further pinpoints to the network performance in terms of routing-delay, packet drops, congestion and more. Unfortunately, it lacks the capability of tracing the packet in a wireless zone. In the simulations performed in this work, the packet can be tracked from the server to the base-station, but to see that the packet arrives to the correct wireless client, the tracefiles must be investigated.

2.4.6 Other

The list of software and tools is long, and in addition to the mentioned software that comes included with *ns*-2 and Evalvid-RA, two additional tools where used to run the complete simulation in this study. These tools are *FFmpeg* and *mp4creator*. For raw video inspection, *VidView* was used.

Mp4creator is a program from the MPEG4IP-project, a open source project aimed to provide a fully open and standards based solution for video streaming applications. In this work has been used to convert the output from *et_ra*, a m4v-file, to a playable mp4-file. Mp4creator adds metadata and header information, called a media *container*, so that a video player can read the necessary parameters for correct play back.

FFmpeg is "*a complete solution to record, convert and stream audio and video*" [23]. It is a free program mainly intended for Unix-like platforms to compress and decompress video and audio in several formats. These include open standards such as H.264 and closed proprietary codec-formats such as Windows Media Video (wmv). In this thesis it is the chosen codec implementation and it is used to convert a binary YUV-sequence into mp4-files with the given quantizations (described in 2.1) in the pre-process. In the post-process, FFmpeg decodes the reassembled mp4-file into a sequence of YUV-images.

Sometimes, the calculated PSNR-numbers can not tell the entire truth about the video quality. For example, one video might have excellent resolution and low quantization but has lots of dropped frames, while another video is harder compressed but with a lower amount of frame drops. These two videos have very different characteristics, and the user preferences here are subjective. To be able to tell the difference, and to inspect the effects of compression and transfer visually, the program *VidView* is excellent. This tool can read YUV videos and compare two instances in parallel. It enables inspection of the differences between the videos by showing only the delta values of the degraded version, and separation into the layers Y, Cr and Cb.

2.4.7 Scripting and coding languages

To be able to understand the mechanisms of *ns*-2, several programming-languages need to be understood. The *ns*-2 software is build with both C++ and object oriented Tcl code (OTcl). To make simple scripts and topologies with well documented standard protocols, a basic understanding of Tcl is sufficient. However, once the goal is to simulate with protocols in the development stage, editing of both Tcl/OTcl and the relevant C++-object instantiations might be necessary.

To run Evalvid-RA with *ns*-2, the latter needs to be recompiled together with some new source code. Evalvid-RA introduces changes in the *ns*-2 TFRC-objects, the objects `agent`, `packet` and `scheduler` in addition to a set of new objects. The Evalvid-RA package also includes a Tcl-script that acts as a topology- framework for generation of both TCP and UDP streams, as well as treatment of trace files for the video streaming study. The methods, syntax and semantics found in this example script has been used as a basis for constructing a new Tcl-script to define the topology and components used in the simulations described in Chapter 3, and a copy can be found in appendix D.

If a special printout of data is wanted, the AWK general purpose computer language can be used. It is a scripting language that is designed for processing text-based data, either in files or data streams. In this thesis, AWK has been used to gather relevant data-fields from the program execution output, to print these data to new text-files and to make computations for correct results that can be presented with *xgraph*.

The simulation process and the iterations of it is coupled together with *shell-scripts*. These are basically collections of Unix/Linux bash shell commands, with the addition that they allow variables to be stored temporarily. All the shell-scripts used in this project are given in appendix B.

2.5 Feedback mechanisms and rate control

In this work, the flexibility and adaptivity issues of multimedia flows in the Internet are discussed and analyzed. Today's client, server and router software mainly support two branches of transport protocol types; TCP and UDP. The TCP protocol was originally designed to offer a reliable connection oriented transmission (RFC 793 from 1981), where the sender gets all data acknowledged before the session ends. Since router congestion causes packet drops and forces TCP retransmissions, the TCP-designers added rate controlling mechanisms (RFC 1122 from 1989 and RFC 2581 from 1999). With the development over the years, the rate controlling mechanisms have been modified and improved in several suggestions, and with the enormous popularity of Internet services, the importance of a fair and correct rate controller is greater than ever. Some of the TCP improvements include TCP Vegas, TCP Westwood[17], High Speed TCP and Fast TCP. These studies are interesting in this context because of one big challenge for TCP-traffic: *How to separate congestion loss in wired links from sporadic packet loss in erroneous wireless links?*

The solutions chosen for this problem in the future TCP version is important to the remaining Internet traffic because of the need for inter-flow fairness. The UDP protocol is believed to be phased out and replaced by a more refined and intelligent protocol with throughput feedback and rate control for multimedia useage [24]. In other words, the future rate controllers for real-time traffic should be developed in corelation to the future TCP rate control in order to be TCP-friendly.

In this section, the needed background information to understand the simulations is presented. More about the different approaches to defining a new real-time protocol standard and an analysis of some suggestions can be found in Chapter 5.

2.5.1 DCCP

As mentioned earlier, the Datagram Congestion Control Protocol is a IETF standardized unreliable transport protocol with end-to-end congestion control. It is intended for use with applications such as streaming media or on-line gaming, where the real-time constraint is most important and packet retransmissions are uninteresting.

The UDP protocol was originally designed for short request-response transfers, such as DNS and SNMP, where the TCP handshake routines are unnecessary overhead. With the use and popularity of applications such as radio-streaming, IP-TV, VoIP and massive multiplayer online games, the dominant UDP-traffic pattern in the Internet has moved from short-lived and small to long-lived and massive. The observed growth of these inelastic flows is feared to pose a threat to the health of the Internet, despite the fact that some applications implement their own congestion control mechanisms. DCCP is planned to be the Internet savior in this context because of the following [24]:

- Avoiding congestion collapse by introducing congestion control on unreliable flows.
- Defining a standard solution for reliable congestion feedback and congestion parameter negotiation for unreliable flows.

- Implementing a standard usage of Explicit Congestion Notification (ECN, see section 2.5.3).
- Providing a pre-defined and easy to use TCP-Friendly congestion control.
- Allowing easier NAT- and firewall-traversal.
- Providing low per-packet byte overhead.
- Maintaining the properties that makes UDP so much used:
  - Low start-up delay compared to TCP three-way handshake.
  - Statelessness.
  - Prioritizing timely data arrival instead of reliable arrival.

All of the above should be beneficial arguments for application developers if a common standard protocol is available and enabled in network and consumer equipment, since they remove the congestion control "burden" from the application layer and the developer.

The DCCP protocol as such is *not* used in the simulations presented here. However, the DCCP standard defines several sets of congestion control mechanisms, each of whom are identified by a Congestion Control Identifier (CCID). To determine which congestion control to use, the server and client negotiate at connection setup or possibly thereafter by exchanging CCIDs. They can request to change congestion control mechanism during the connection, and different choices can be prioritized by listing CCIDs in datagram header. As of 2006 and DCCP standard document RFC4340 [25], only two types of congestion control are defined:

- CCID 2: TCP-like Congestion Control
- CCID 3: TCP-Friendly Congestion Control

The CCIDs 0-1 and 4-255 are reserved for other and future congestion control mechanisms. DCCP requires that any implementation supports at least CCID 2, but for multimedia transfer, CCID 3 is likely to be more popular. Therefore, CCID 3, TFRC, is used in the simulations presented here. TFRC is described in the next section.

2.5.2 TFRC

UDP is often referred to as "greedy" because it has no congestion control mechanism, thus transmitting at a given pre-defined rate even though the path might be congested. This greedy nature of UDP will in competition with TCP traffic let the latter suffer because it reacts to returned network load parameters such as round-trip-time and packet loss probability by adjusting its transmission rate accordingly [6].

To make sure that a certain degree of traffic fairness is obtained in the network, the definition for TCP-friendliness must be met: "*A non-TCP connection should receive the same share of bandwidth as a TCP connection if they traverse the same path*"[26], [3].

TCP-Friendly Rate Control (TFRC) is a young protocol, first standardized by IETF in 2003 with RFC 3448 [9] and in 2006 included in the Datagram Congestion Control Protocol (DCCP).

It is described as the DCCP Congestion Control ID 3 (CCID 3) in RFC 4342 [27]. The protocol operates on unicast flows in a best effort Internet environment. It generates bit rates comparable to TCP connections on the same link, but with smoother transmission rate transitions than TCP has. Therefore, it is specially apt for applications that require smooth streams, like video conferences or other streaming media [9]. TFRC works according to the following four general steps:

1. Loss events are measured at the receiver and sent back to the sender.
2. The sender calculates round-trip-time (RTT) based on the feedback.
3. The loss event rate (from receiver) and calculated RTT is used as input for calculation of a new transmission rate.
4. The sender adjusts the rate accordingly.

The new transmission rate is calculated using the TCP throughput equation:

$$X = \frac{s}{R \times \sqrt{2 \times b \times \frac{3}{p}} + t\_RTO \times 3 \times \sqrt{3 \times b \times \frac{p}{8}} \times p \times (1 + 32 \times p^2)} \tag{2.3}$$

Where X is the new rate, s is the packet size (bytes), R is RTT (seconds), p is loss event rate, t_RTO is the TCP retransmission timeout (seconds) and b is the number of packets acknowledged in one feedback packet. A feedback timer at the sender controls when the rate should be adjusted if no feedback is reported.

The TFRC specification does not detail the content and set up of data transport, as it is meant to work alongside different types of transport protocols. However, some general requirements need to be met, i.e. sequence numbers, timestamps and a timestamp estimate. With the Evalvid-RA framework, it is possible to simulate pure TFRC transport directly on top of the network layer. In this setup, the TFRC header is 16 byte. With a maximal transfer unit (MTU) of 1000 byte and a 20 byte IP header, the total packet size is 1036 bytes.

Through the TFRC interface to the application layer, Evalvid-RA fetches information about the current packet rate and adjusts the video compression accordingly. This interface is discussed further in section 4.1.

2.5.3 Network intelligence

In the simulations of Chapter 3, tests of ECN, RED and P-AQM where originally planned. Due to scope limitation and other constraints, the simulations now focus more in depth on TFRC. Nevertheless, the technologies introduced here are discussed in Chapter 5 as some of several possible solutions to the challenge of providing more reliable real-time services over Internet.

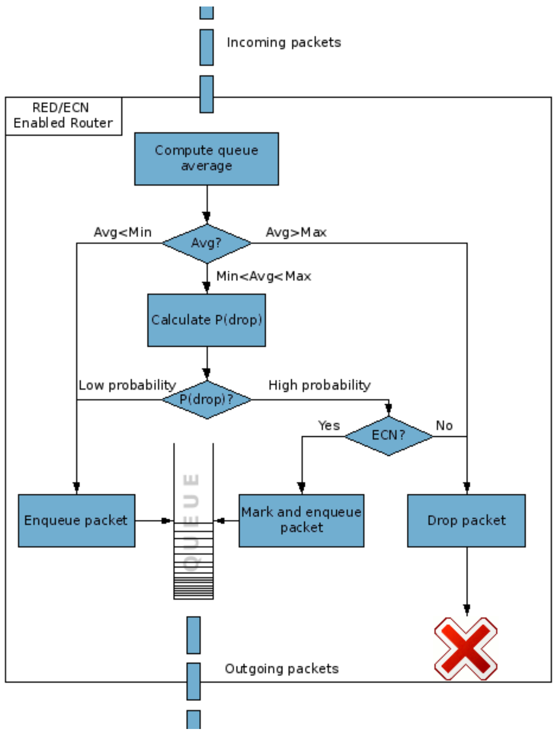
ECN & RED

In a traditional TCP network, the packets are dropped if the input buffers on the routers are full according to the drop tail queuing algorithm. First when reports of packet drops are returned to the sender(s), the transmission rate is adjusted and the dropped packet is queued for retransmission. With a Random Early Detection (RED) queuing algorithm, the congestion can be foreseen and reported back to the sender(s) *before* any packets are dropped. Thus, the throughput and resource utilization is improved and the phenomenon called global TCP synchronization[9] can be avoided. RED monitors the average queue-length, and drops packets randomly according to a statistical distribution and minimum and maximum queue length thresholds (see fig. 2.6(b)). The packet dropping is simply a indication to the sender that the queues *might* overflow if its current rate is kept or raised. A problem with this solution is that the drop forces the sender to retransmit the “unlucky” packet, so it would be better to inform the sender about the congestion possibilities in another fashion [28].

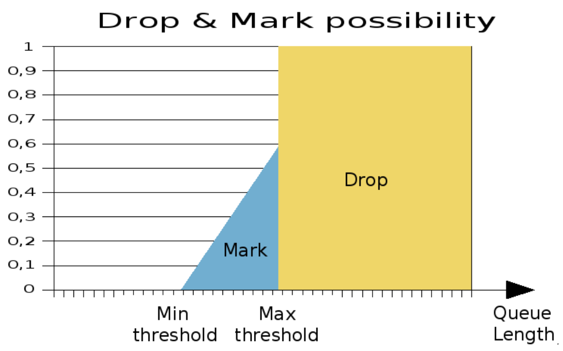
ECN is a mechanism to do this congestion feedback without dropping any packets as long as they are not overflowing the router buffer. It is an extention to the IP-layer that can be used if both sender and receiver support the use of it. It is also depending on router support, since *a)* there is no point in using ECN if no routers knows how to use it for congestion indication, and *b)* some old IP routers might discard an IP packet with active ECN field (mistaking it for being corrupt). The addition to IP consists of a header field of two bits in the DiffServ-field (called the Type Of Service (TOS) field and the Traffic Class Octet in IPv4 and IPv6 respectively), see fig. 2.7. This header field is called the Congestion Experienced (CE) codepoint, and can have the values `01` or `10` to indicate that ECN is enabled (ECN-Capable Transport, ECT), or `11` to flag that congestion is observed (CE) [29].

To implement ECN in the Internet, it is necessary to enable support in the interface between network layer and transport layer so that the transport protocol receives information correctly. With TCP, this can be done as an evolution, since the use of ECN can be determined when the TCP connection is established. With UDP on the other hand, there is no two-way communication on the transport layer, so the sender has no way of knowing if the CE-flag has been set or whether the receiver supports ECN at all. One solution could be to allow applications access to the ECN mechanisms on the IP layer through the UDP sockets. Another solution, which would be more of a transport layer revolution, is to convert to DCCP, which has ECN-support by default [24].

[9]This occurs when several TCP sources are notified of packet drops simultaneously, thereby increasing/decreasing output rates simultaneously. Results are bursty traffic and poor utilization.



(a) RED & ECN enabled router



(b) Drop or mark possibility

Figure 2.6 A conceptual flow-chart of a RED & ECN enabled router (a) and a graph showing the drop possibility P(drop) of a packet when the queue average is between the two thresholds (Max/Min).

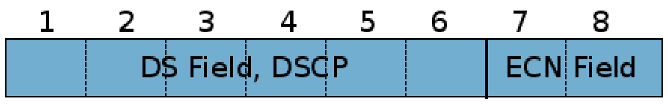


Figure 2.7 The Differentiated Services and ECN fields in IP. (DSCP = Differentiated services Code Point)

P-AQM

DCCP is the IETF choice for the new generation streaming media transport protocol. It's use of binary congestion level feeback[10] has been questioned, and claimed to limit the accuracy and speed of rate adaption [8]. The Proportional Active Queue Management (P-AQM) introduces a different form of router queuing and a more detailed feedback report. P-AQM is not a transport protocol in itself and must be combined with UDP as a data carrier. The P-AQM suggestion consists of two functionalities; The queue management and the feedback mechanism.

[10]"Binary" because the packet is reported dropped or not dropped, or the 1-bit ECN field of ECN enabled routers is set.

AQM is a queuing method that can warn about congestion before a buffer overflow occurs. It uses queue length limits smaller than the buffer capacity and activates congestion feedback for all packets exceeding this limit. P-AQM uses a two-queue scheduler (see figure 2.8) that separates adaptive flows (TCP) and aggressive flows (UDP). While the TCP queue is kept under control by TCP's own rate controll mechanism, the P-AQM system takes care of congestion feedback for the UDP queue.

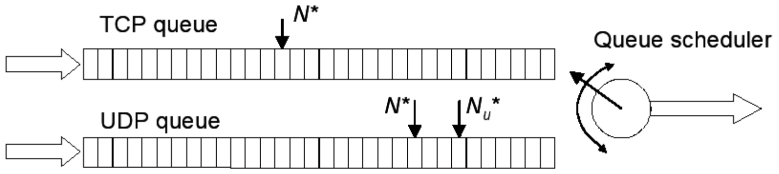


Figure 2.8 The two-queue scheduler of P-AQM[8]

Explicit Congestion Feedback (ECF) or Explicit *Rate* Feedback (ERF) are the two tested and analyzed methods for calculating and returning feedback to the stream source in [8] and [30] respectively. Both variants are suggested to be sent via Internet Control Message Protocol (ICMP) Source Quench packets directly to the sender, since neither TCP, UDP nor IP has suitable fields for this use. These packets have a 32 bit unused field that can carry more detailed feedback information than the ECN system.

In the ECF variant, the congestion level is indicated by a value below 1 to report congestion and above 1 to indicate available resources. The values are calculated as Additive Increase / Multiplicative Decrease (AIMD). In P-AQM, this value is not calculated to ensure TCP-fairness, since this is taken care of by the router queue scheduler (see eq. (2.5)). The calculated value is instead optimized to ensure intra-UDP flow fairness and to make the rate adoption smooth and dynamic to fast traffic load changes. When the server receives the ECF value, it calculates the new rate with

$$r(n+1) = \begin{cases} ECF(n) \times r(n) & ECF(n) < 1.0 \\ min\{(n) + ECF(n), r_{max}\} & ECF(n) > 1.0 \end{cases} \tag{2.4}$$

where r_{max} is the original or maximum rate of the streamed source. Also, the server has to take into account that several router nodes my be sending reports. In that case, the report with the lowest value should be used as $ECF(n)$ in eq. (2.4). This rate adaption ensures max-min fairness, which is a traffic shaping property that assigns a fair share of the bandwidth to both low and high bit rate data flows. DCCP in contrast, uses RTT measures to achieve proportional fairness and TCP-friendliness.

The queue scheduler is responsible for fairness between elastic (TCP) and inelastic (UDP) traffic. In P-AQM, the available bandwith to non-elastic traffic is given by

$$c_{udp}(n+1) = c \times \frac{nofUDP_f(n)}{nofUDP_f(n) + nofTCP_f(n)}, \tag{2.5}$$

where c is the total output capacity, $nofTCP_f(n)$ is the number of long running TCP-flows and $nofUDP_f(n)$ is the number of UDP flows chosen based on the proportional bandwidth demand of each present flow. Output capacity available for TCP is thus

$$c_{tcp}(n+1) = c - c_{udp}(n+1). \tag{2.6}$$

This queue scheduling method is not claimed to be optimally TCP-friendly for all combinations of short/long-lived, bursty/stable and small/big flows, but it has proved good performance when the TCP flows are long-lived and the UDP flows are persistent (Section sources: [8], [30], [31]).

2.6 Wireless transmission: 802.11

To get a thorough understanding of the problems connected to transport layer feedback mechanisms in wireless environments, it is important to have a brief overview of the relevant technology in wireless networks. This master thesis focuses on the most widespread wireless broadband access today; the ANSI/IEEE 802.11 wireless LAN system.

2.6.1 The 802.11 group

The 802.11-standard has been developed since the mid-90s, leading to a comprehensive group of standards and enhancements today.

The first edition of the 802.11 standard was approved in 1997, and offered bitrates of 1 or 2 Mbps. Two years later, the first substandard called *802.11b* introduced bitrates of 5 or 10 Mbps, working in the 2.4-2.5 GHz-band. In a parallel research process, another substandard called *802.11a* was defined. This standard operates in the same frequency band as the legacy 802.11 (5 GHz), a band that is not as heavily used as the 2.4 GHz-band of 802.11b. It allows for faster signaling and less interference, so the 802.11a standard works on velocities from 6 to 54 Mbps. However, due to the high carrier frequency, the standard has limitations on distance and coverage. In practice, the 802.11a standard has a reach of about 30 meters and is limited to almost line of sight [32].

In 2003, a new amendment joined the 802.11 group, ratified and known as 802.11g. This variant combines benefits from both its predecessors by allowing up to 54 Mbps in the 2.4GHz-band. Hence, faster data transfer can be achieved in a larger coverage area, and walls or obstacles do not interrupt the traffic to the same extend as with 802.11a. Today, most consumer equipment supports a ”tri-band” mode covering all three substandards in the 802.11 group [33].

To keep up with the velocities of wired ethernet LAN, a group was formed in 2004 to develop *802.11n*. The working drafts of this group has recently passed a 802.11 working group voting, and are estimated to be published in September 2008 [34]. 802.11n is supposed to support bitrates of up to about 700 Mbps, using two antennas transceiving 360 Mbps each in ”Advanced beamforming mode” [33].

In addition to the the original standard and these three amendments, the 802.11 standard covers security (802.11i), bridge operation procedures (802.11c), roaming extensions (802.11d) and QoS (802.11e) among others. The latter can have a significant impact on video streaming speed and quality, and will be covered in section 2.6.5.

2.6.2 General architecture and specifications

The 802.11 standard specifies the use of both the physical (PHY) and the Media Access Control (MAC) layer. The legacy specification included three alternative PHY implementations; Frequency Hopping spread spectrum, Direct Sequence Spread Spectrum (DSSS) and Infrared. However, only DSSS has been further developed in the later amendments (802.11b).

Both layers of the 802.11 stack have a layer management entity to provide management interfaces within and between the layers. Additionally, each 802.11 unit has a station management entity that operates on both layers. The PHY layer carries out the modulation and encoding.

802.11a and 802.11g uses Orthogonal Frequency Division Multiplexing (OFDM) but with different types of subcarrier modulation. 802.11b uses Complementary Code Keying (CCK) which is a variation of Code Division Multiple Access (CDMA).

The MAC layer has several responsibilities. The standard defines:

- An asynchronous data service with the upper layer, normally 802.2 Logical Link Control (LLC) layer. This includes fragmentation and reassembly of packets and MAC Protocol Data Units (MPDUs), and framing procedures. Unlike Ethernet (802.3), 802.11 depends on framing from the 802.2 LLC.
- Security services: Originally, the security was covered by Wired-Equivalent Privacy (WEP), but as this encryption proved to be insecure, the 802.11i-group was formed. Before the 802.11i amendment was complete, Wi-Fi-Protected Access (WPA) was released to the market. This protocol implements parts of the 802.11i specifications, while the newer WPA2 satisfies all requirement stated in 802.11i.
- MPDU ordering of multicast, broadcast and directed MPDU traffic.
- Access procedures, which is covered in section 2.6.3.
- Association service: Controls the association state between STAs and APs. This service includes media scanning (passive or active) and synchronization.

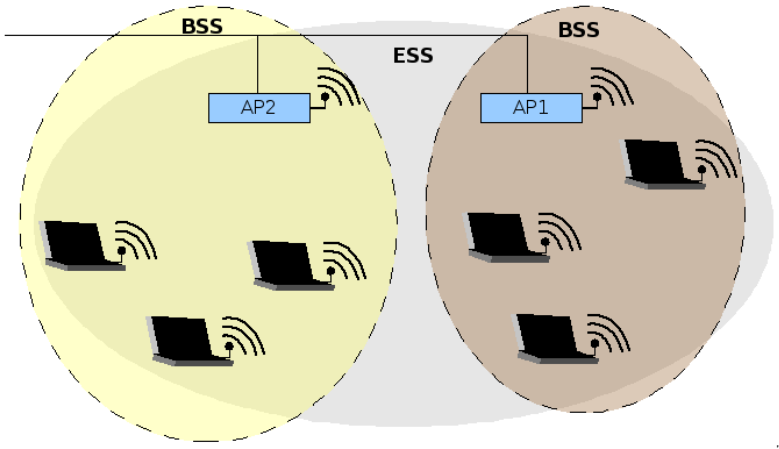


Figure 2.9 STAs connected to AP1 are part of one BSS, STAs connected to AP2 are part of another. Together, the two BSS form an ESS.

A wireless 802.11 network can have two distinct topology modes; Ad-Hoc or Managed. In the former mode, at least two mobile stations (STA) are interconnected without external management. The latter and most commen mode consists of one or several Access Points (APs) and STAs. Each STA is associated to one AP and this AP is usually connected to a wired LAN or backbone network through a Portal (PO). Together this forms an infrastructure Basic Service Set (BSS). An interconnected WLAN with several APs is called a Extended Service Set (ESS), see figure 2.9. In the simulations presented later in this report, a topology with support for one multi-BSS ESS will be used.

2.6.3 MAC Access procedures and timing

To avoid collision of traffic on the same channel and minimize the chances of network internal interference, the MAC layer must have rules to regulate how and when the mobile network nodes (STAs) access the medium. There are three types of access procedures defined in the current 802.11 standard:

PCF The Point Coordination Function access procedure requires one central node to coordinate which station that is allowed to send at each given time. It can be used in point to point connection (ad-hoc) between to stations, or it can be the job of an AP. PCF provides contention free frame transfer and thus allows for better QoS, but it does not define traffic classes, nor does it work with RTS/CTS scheme (described later). PCF is not widely implemented because it is optional, and will not be used in simulations in this report.

HCF The Hybrid Coordination Function is a part of the new 802.11 QoS-amendment; 802.11e. It will be fully described in 2.6.5.

DCF The Distributed Coordination Function is the most common form of media access control in 802.11 networks. The DCF technique used in 802.11 is called Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). It will be further described in the following.

For elastic traffic types such as normal web-traffic or FTP-traffic, the main priority is to get every sent byte to the receiver, so variable delay and retransmissions are acceptable. For non-elastic traffic such as real-time video and audio however, the delays should be minimized and held as constant as possible. Since the DCF access procedure of 802.11 is the most used and implemented technique in wireless networks today, it is interesting to study how it impacts the end to end delay of a multimedia session. This will be covered in the simulation case III, section 3.7.3.

The DCF access procedure is carried out in several steps. First, the STA senses the medium before transmitting. If the medium is free for a given period, called the Distributed Interframe Space (DIFS), the STA is allowed to send its data. While the STA transmits, other STAs can sense the transmission and defer access. When the Transmission is over, other stations can send after a DIFS period. If the medium is not available after a DIFS, the STA activates a backoff algorithm. This is called a *binary exponential backoff algorithm* and the backoff interval is chosen randomly to form a uniform distribution. The reason for this is that several STAs may be waiting to transmit at the same time. When they sense that the medium is available, the random backoff interval will make it less possible that they start transmitting simultaneously. In addition to being triggered when a STA senses that the medium is busy, the backoff algorithm is triggered after each transmission and retransmission. The backoff time is upwards limited with the contention window parameter. The contention window is increased exponentially for every retransmission n of a MPDU following $CW = 2^n - 1$ [transmission slots], but with an upper

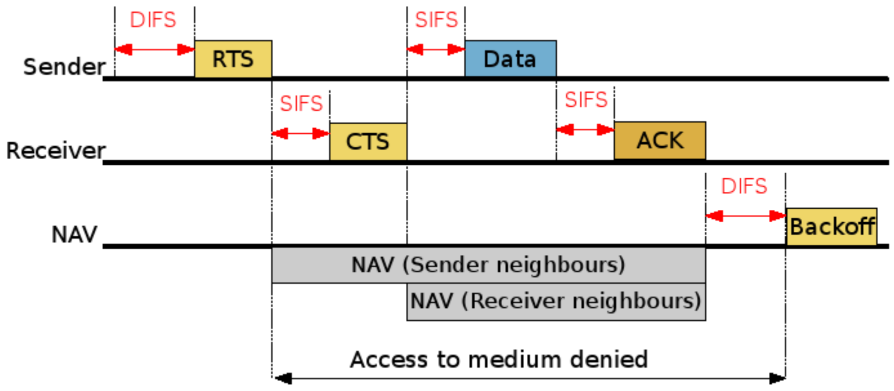


Figure 2.10 Complete CSMA/CA procedure for transmitting one frame with RTS/CTS. (Note that the packet sizes are not displayed in realistic order of magnitude.)

boundary set by the physical layer. For example, the DSSS type limits the maximum contention window to 1023 transmission slots[11] .

Most unicast frametypes are required to be acknowledged by the 802.11 MAC protocol. This is sent with an acknowledgment packet after a period called Short Interframe Space (SIFS). The entire MPDU transmission and acknowledgment process is illustrated in figure 2.10. Broadcast or multicast traffic will not be acknowledged.

An optional CSMA/CA feature is the exchange of packets called Request To Send (RTS) and Clear To Send (CTS). This is mainly used to solve the *hidden station problem*, which is as follows: Two STAs wants to transmit data at the same time. They can both reach and sense traffic from the associated AP, but due to distance, walls or other physical obstacles, they cannot sense each other. Both stations will assume that they have available transmit time, but the AP will receive from both STAs, resulting in interference and failed reception.

The RTS/CTS carrier sense mechanism forces STAs to ask for a pre-transmission application with a RTS package containing the length of the transmission, as seen in fig. 2.10. All STAs do not necessarily hear the RTS transmission, however, all STAs *must* hear the AP. So, if the transmission application gets approved by the AP, a CTS, which everyone hears, is returned. All STAs with the same AP association will then set a Network Allocation Vector (NAV) which indicates the length of a pre-transmission SIFS, the transmission, a post-transmission SIFS and the acknowledgment. The NAV effectively blocks any transmission from the other STAs when it is set. When the NAV is released, all STAs must wait one DIFS and the random backoff period before they can apply for a transmission (Sources: [32], [33], [36]).

2.6.4 Link stability

As mentioned earlier in this report, the wireless transmission medium, air, can be error-prone. Unlike wired connections, it is difficult to guarantee throughput, bandwith, delay or jitter, especially when the wireless nodes are allowed to move while transceiving and the current frequency

[11] 802.11a and .11g have a slot time of 9 μs while 802.11b has 20 μs. 802.11g scales down to 20 μs if some stations lack the high speed mode [35].

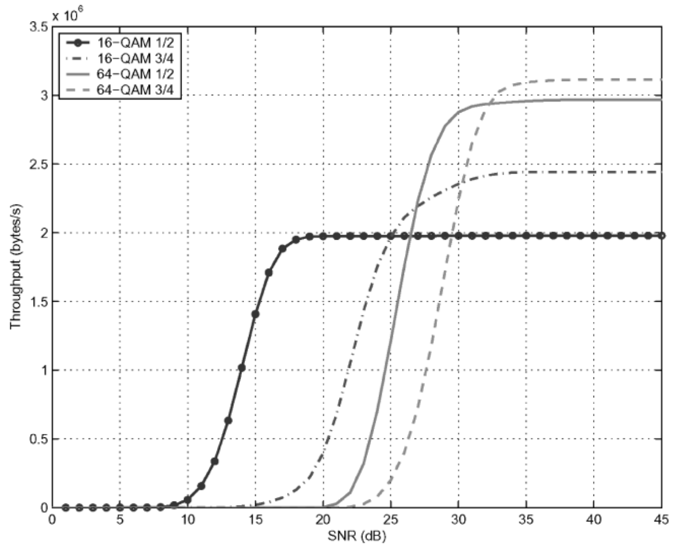


Figure 2.11 The different rates have different modulations which results in different SNR [37].

band is not 100% reserved for the service. Some of the reasons for link instability are briefly summarized in the following.

Rate control

Rate control in the 802.11 network interface is used to make a STA flexible to the big signal strength variations observed when the STA is moving or interference fluctuations occur. The PHY layer of 802.11 provides different discrete rate services to the above layers[12]. Each discrete level corresponds to a coding and modulation scheme, all differing in radio coverage and noise sensitivity. For instance, the 64-QAM $3/4$ offers large throughput but is noise sensitive and thus provides a narrower coverage, see figure 2.11.

It is up the device driver to choose which rate service it currently wants based on information on the link conditions, called Channel State Information (CSI). For real-time applications, it is important that the algorithm that chooses desired rate is fast and clever enough to adjust the rate according to temporary and constant signal strength changes. Normally, this algorithm is based on feedback statistics calculated over time. Now, a short and temporary drop in signal strength might cause packet loss and trigger MAC layer retransmission. In the rate controller, the throughput is smoothed by the statistical process, so the short drop-out will have no effect. To elastic traffic, this is not a problem, in fact, it is the most effective solution. For real-time traffic however, it may cause severe quality degradation. 802.11 rate controller algorithms for real-time traffic is discussed further in [37] (paragraph source).

[12]i.e. 6, 9, 12, 18, 24, 36, 48, and 54 Mbit/s in 802.11a

Fading and shadowing

Fading and shadowing are typical radio issues that decrease the QoS of a 802.11 link. Shadowing occurs when an object cuts off the line of sight between two communicating STAs or STA to AP. The effect of such a blockade of the direct signal depends on the signal strength and the amount and quality of reflected signals. In open air, there is ideally no reflected signal components, while on open ground there is at least one set of reflected rays[13]. In a home or office network, all objects, walls and buildings will cause a mesh of scattered reflections. While these reflections can reduce the consequences of direct shadow, they introduce another problem: Multipath fading.

When a radio antenna receives both a direct signal and a reflected signal, it will observe that the reflected signal is slightly delayed because of the added distance. Reception of several delayed signal can lead to both destructive and constructive interference and phase shifting. Destructive interference happens when the signal is partially cancelled by a phase-shifted variant of itself. This phenomenon is often referred to as Rayleigh fading or Rician fading. The Rayleigh fading distribution model is most applicable if there is no line of sight between sender and receiver (source [38], [33]).

Interference

Interference is common in 802.11 networks because of the frequency band licensing. The 2.4 GHz band[14] which is used by both 802.11b, 802.11g and 802.11n, is not a dedicated channel for WLAN only. A normal household appliance today that has emissions in the ISM-band is the microwave oven. A normal oven for private use operates on a relatively narrow band close to 2.45 GHz and transmits on that frequency only every 20 ms. Therefore, its external radiation is only disturbing some of the channels used by 802.11, see figure 2.12. A professional restaurant oven on the other hand, uses two instead of one magnetron tubes (transmittors) which occupy a broader frequency range twice as often. Allthough the commercial ovens normally are better shielded than residential ones, the radiation is sufficient to cause problems for several 802.11 channels a majority of the time [39].

Another widely discussed source of interference to 802.11 is Bluetooth. This wireless network protocol is designed for small distances, and the senders have a power output of about 20 dBm. Still, a significant interference is observed if transmitting Bluetooth devices are within about 5 meters from a receiving 802.11 STA. The effect is stronger the further away from the AP the STA is because the Signal-to-Interference (S/I) ratio decreases (Sources: [40], [41]).

2.6.5 Quality of Service enhancement: 802.11e

The IEEE 802 working group fully accepted the new standard amendment for enhanced Quality of Service, 802.11e, in September 2005, almost five years after the task force was originally formed [34]. The need for improved jitter and delay control was increasing as the use of both

[13]Called Two Ray Ground Propagation, discussed more in section 3.7.2

[14]Called the ISM-band (Industry, Science and Medical applications).

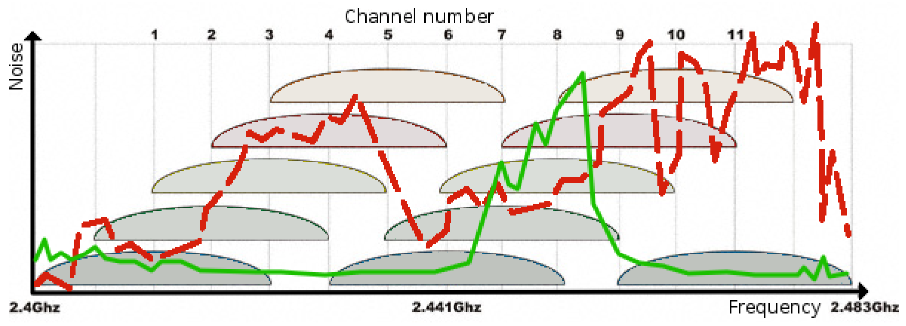


Figure 2.12 Commercial/professional microwave ovens (red) generate noise in a broader band and interferes with more 802.11 channels than residential ovens (green). (Note that the channel allocation on the y-axis has no direct relation to the noise value.)

WLANs and realtime media escalated. To give this traffic type better conditions three things had to be improved with the new amendment:

1. A transmitting STA must get a guarantee of a certain minimum of medium access to avoid buffer delay and jitter problems.

2. A system for traffic flow classification and differentiation must be available.

3. The transmission overhead should be minimized without making the transmission more vulnerable to interference and errors.

The 802.11e amendment introduces solutions to these three points through a new scheme for grouping and acknowledging MPDUs and two new schemes for medium access. The acknowledgment scheme is called block-ACK, and works in the following way.

The transmitter is allowed to transmit several MPDUs in a row without waiting the duration of two SIFS and an acknowledgment between each MPDU. This is called a Transmission Opportunity (TXOP). At the end of this TXOP, it sends a block-ACK request containing the MPDU sequence numbers contained in the burst. The receiver replies with a block-ACK for all the correctly received MPDUs. The missing MPDUs will be buffered at the sender until they have been acknowledged or their packet life time expires. The receiver MAC buffers the incomplete burst until the remaining MPDUs are received or it receives MPDUs with higher sequence number than the lost ones. This indicates that the lost MPDU's lifetime at the sender has expired. Note that 802.11e enables the MAC entities to operate with packet lifetime instead of retry limits. This gives a better delay constraint for real time applications. An effect of the TXOP / block-ACK mechanism is that the receiver MAC delivers the MPDU-bursts to the above layers only when the entire burst is received or the MPDU lifetime has expired. Hence, application layer bandwith estimators may be fooled to compute wrong results.

The new schemes for medium access in 802.11e are improvements to PCF and DCF through the new *Hybrid Coordination Function* (HCF). Two variations are defined; The *Enhanced Distributed Channel Access* (EDCA) and *HCF Controlled Channel Access* (HCCA).

EDCA

EDCA is an improvement to the legacy DCF and was therefore formerly known as Enhanced Distributed Coordination Function (EDCF). It provides relative qualitative differentiation of traffic based on traffic classes at each STA. It is however not able to provide any hard QoS guarantees. The legacy DCF mechanisms ensures a inter- and intra-STA traffic fairness by giving all STAs equal chances of acquiring a transmission slot in the backoff period. Additionally, DCF uses only one queue for outbound traffic. These two issues limits the QoS differentiation significantly. EDCA addresses both problems by allowing a certain traffic unfairness and by splitting traffic classes into separate queues.

Four *Access Categories* (ACs) are defined in EDCA, listed in prioritized order: Voice, video, best effort and background. These ACs are treated with different Contention Windows. The high priority AC is assigned with a smaller CW than the lower prioritized ACs so that it will always have a higher probability of acquiring the medium. Additionally, EDCA allows the ACs to use different Interframe Spaces (IFS). While all traffic in DCF uses fixed values for DIFS and SIFS, EDCA assigns a bigger IFS to the low prioritized ACs and thereby forces them to wait longer than the prioritized ACs before initiating the backoff algorithm.

HCCA

HCCA is an improvement of the legacy PCF, the 802.11 contention-free access method. HCCA is the most complicated of the available and standardized access methods, but is also the one which can promise the strongest QoS. As with PCF, the HCCA divides the timeline into a Contention Period (CP) and a Contention-Free Period (CFP). While PCF uses polling from the AP to allow STAs to transmit in CFP and uses legacy DCF in the CFP, HCCA uses the more advanced and effective EDCA in CFP. Moreever, the hybrid controller (which is the AP), can define both traffic classes and traffic streams, which enables it to prioritize on a session-level. Each STA can report its queue lengths for every AC. With this information, the AP can make improved decisions and differentiation for better over-all QoS performance.

Despite that HCCA can provide the best QoS available following the IEEE 802.11 standard, it is rarely implemented in consumer equipment as is the case with PCF. The international *Wi-Fi-Alliance* has developed an equipment certificate based on the 802.11e amendment called Wireless Multimedia Extension (WME) or Wi-Fi Multimedia (WMM). This is mainly used to certify Voice over IP and WLAN equipment such as IP-telephones (sources: [42], [35], [43]).

CHAPTER 3

Simulation background and layout

This section of the report covers the details of the simulations carried out, and how the system was set up to obtain interesting results. The reader will find discussions on how the different parameters where chosen, and how they affect the system in theory. Before the performed simulations are described, their goal is further defined and the simulator implementation is described. The results and analysis of the obtained results can be found in Chapter 4.

3.1 Goal

As stated in the introduction of this document, the primary goal of the simulations in this thesis is to illustrate the advantages and disadvantages with TFRC in a realistic wireless environment and compared to TCP and UDP traffic. The protocols in question have all been tested thouroughly over *wired* networks with different types of cross-traffic, topologies, equipment, capacity and levels of realism. Both UDP and TCP traffic have been tested and simulated over *wireless* networks, but TFRC performance in wireless networks is not so thoroughly documented. The simulation results shall be a basis for further discussion of TFRC and DCCP as a multimedia carrier, but also to pinpoint the performance of 802.11 for real-time traffic.

In the master pre-study [13], the focus of the simulations was to see how TFRC performed when unavoidable packet losses occurred due to an unstable link. That study is continued here with a proper 802.11 MAC and PHY implementation in *ns*-2. The TFRC and 802.11 performance is evaluated through a set of cases with the following hypothesis':

- **Case I:** "TFRC gives better results than UDP with high traffic load and interference" (sec. 3.7.1).
- **Case II:** "UDP gives better results than TFRC with low link quality or noise" (sec 3.7.2).
- **Case III:** "802.11 MAC gives significant added delay" (sec 3.7.3).

The results from these cases are meant to illustrate a set of effects that are of importance for a real-time multimedia service delivered to a wireless client. The goal is to find the vital deviations in behavior and performance in a relatively coarse scale.

3.2 Topology

With *ns*-2, most topologies can be defined with Tcl-code. The Tcl-script base used (see Appendix D) is written completely from scratch to ensure a tidy and structured code that fits the simulation purpose optimally. It defines a 'dumbbell'-topology as illustrated in figure 3.1. The setup defines a server access network, a simple core network, a base station network and wireless access networks[1]. To simulate l streams to m users in n formats or types, the script automatically sets up n separate servers, m clients and the l connections between them. The supported applications and protocols are:

- VBR Video/UDP
- Rate-Adaptive VBR Video/TFRC
- FTP/TCP
- Web(HTTP)/TCP
- Ping/ICMP

All of these will be used in different numbers and combinations to get the desired scenarios described later.

When studying the impact of errors on a link, it is important to isolate the error source in the analysis. To avoid confusion from queue delays and contention in other part of the network, the Tcl-script is set up to let all data stream without long propagation and contention delay from their respective sources to the wireless access network.

Figure 3.1 shows the final topology with all application types enabled. It has five source nodes, or media servers, and seven consuming nodes, or clients. Note that while two FTP clients are connected to one FTP server, and two VBR Video/UDP clients are connected to one VBR Video/UDP server, each VBR Video/TFRC client requires a separate VBR Video/TFRC server. This is done to make the interaction with the Evalvid-RA toolset smooth. The toolset in its current state is only capable of monitoring and rebuilding *one* video flow per simulation for further analysis. This video can be transported with either UDP or TFRC, but to compare the results of two video streams, two separate simulation iterations must be executed. The Evalvid-RA toolset simply monitors the flow that starts first. How the different simulation iterations are controlled by external scripts are described in section 3.5 and under each case description (sections 3.7.1, 3.7.2 and 3.7.3).

The simulations carried out to test TFRC and 802.11 performance here are similar to classical client-server service. However, the technologies in question are equally interesting to use

[1]The latter two are also known as Extended Service Set (ESS) and Base Service Set (BSS) respectively in 802.11 literature.

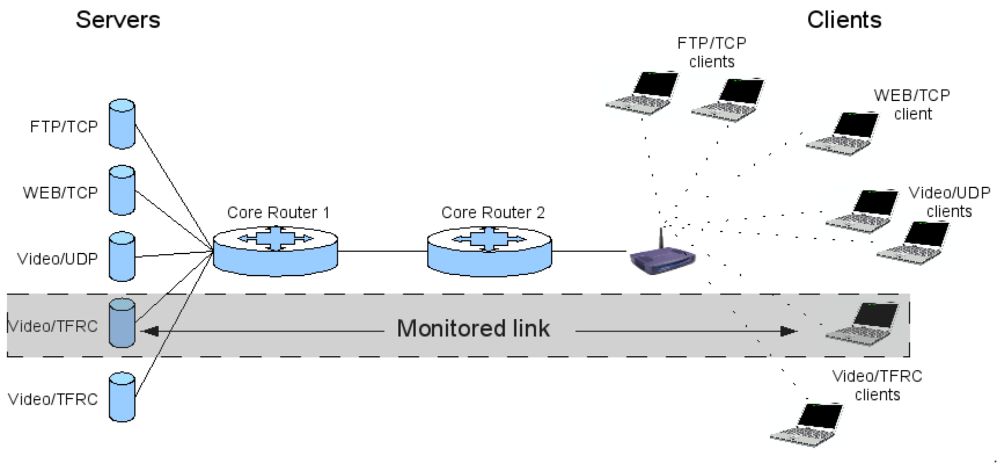


Figure 3.1 The implemented topology with utilization of all application types.

in video-conferencing, where the communicating nodes act as both servers and clients simultaneously. Allthough the script can be used for the latter service with only minor modifications, the results from the presented simulations apply to both service types as is. The most important difference in QoS demand is short RTT for video-conferencing, a parameter that can easily be tuned in the simulation framework.

3.3 Simulation controller: 802.11.tcl

The main part of the programming and implementation challenge in the performed study is the design and testing of the script that sets up and controls the simulation. As mentioned in section 2.4.3, the *ns*-2 simulator has a core of compiled C++-objects which can be controlled and linked together through an interface with OTcl. The OTcl framework of *ns*-2 is in turn controlled by a user made script written in Tcl. The script that sets up and controls the simulation of this study is called `802.11.tcl`, can be found in Appendix D, and will be briefly explained here.

The complete interaction of one end-to-end video transmission is not totally controlled by this script, but includes all elements of the Evalvid-RA framework described in section 2.4.4. How this framework is controlled, is described in section 3.5. Now, since the 802.11.tcl-script is not the top-controller, it should be controllable through an interface to other scripts or programs. Therefore, the script execution starts with a method that reads in all parameters delivered from command-line and sets the correct internal parameters accordingly. For example, the shell command

```
$ ns 802.11.tcl -nofTFRC 1 -nofUDP 1 -doPing 1 -disableRTS_CTS 1
   -prop Propagation/Shadowing -verbose 0
```

tells the simulator to set up one TFRC server and client, one UDP server and client (`nof` = “number of”), to do an initial ping to check the connection, to turn off the MAC-layer RTS/CTS-mechanism (described in section 2.6), to use a shadowing propagation model, and not to write

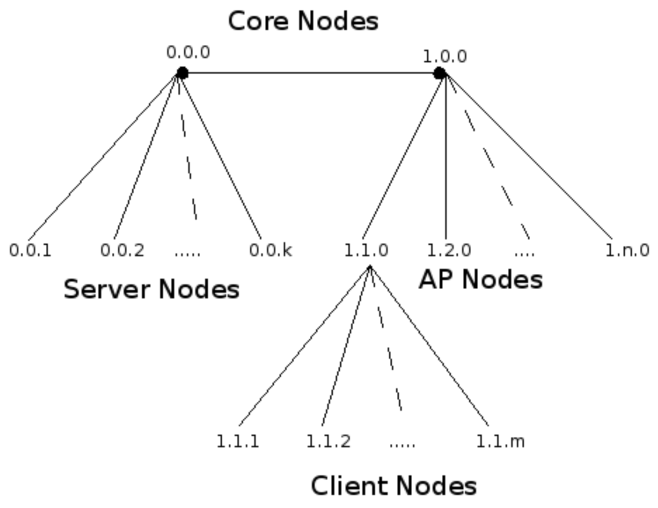


Figure 3.2 The network layer address hierarchy

out information at run-time[2]. The parameters that are *not* set through this command-line interface, are used based on default values set in the beginning of the script (i.e. TwoRayGround is default propagation, pinging is disabled by default etc.).

The next thing the script does, is to count all source and destination nodes required, and to set up the topology with nodes and links accordingly. All nodes are addressed in an IP-like manner, with a three layer hierarchical model as seen in figure 3.2. They are set up with specified queuing types, routing algorithms and link layer types, and are connected with links with the specified bandwidth and propagation delay. All MAC- and PHY-layer parameters for the 802.11 network are set in this phase.

The next task for the script is to prepare input and output streams for the Evalvid-RA cooperation. The script tells the simulator where to find video tracefiles and what to do with them. It opens files to write the simulation results such as video frame send and arrival time from and to the Evalvid-RA application layer implemented in *ns*-2. So far in the script-execution, the nodes are set up and the input/output is prepared, but no services are defined. Each service-type (FTP, WEB, Video/UDP and Video/TFRC) has a dedicated setup-function that instantiates the application server and client at each node and connects them. Once this is done, the simulator is fed with the transmission schedule, which is the the start and stop times of each application. In addition to this setup, the script has functions to record the data throughput, and connectors for incoming/returned data to the Ping/ICMP and Video/TFRC instances.

When the Tcl-script has been executed, all necessary variables and settings are loaded into the compiled *ns*-2 core and the simulation starts.

[2]This function can only control the output of the Tcl-script, not the compiled code.

3.4 ns-2 modifications

The current *ns*-2 framework contains all the necessary code for both wireless networking with 802.11 and the transport layer protocols UDP, TCP and TFRC. Virtual application servers for web and FTP traffic are also included, but to enable the rate adaptive video application, some changes and add-ons must be made. To enable this, code from the Evalvid-RA project [12] must be plugged in with a seven step manual installation procedure. When the *ns*-2 package with the new add-on is recompiled, a rate adaptive video service can be added to a server node via the Tcl interface.

To enable the Evalvid-RA video application to run over both TFRC and UDP at the same time (from different servers, but with the same simulation source code), some changes in the Evalvid-RA source code had to be made. The source file called `vbr_rateadapt.cc` had predefined states with static variables to enable or disable the TFRC header type[3]. To provide dynamic changing of header type, code was added to receive and store this setting directly from the Tcl script via a boolean-style variable called `isTFRC`.

The standard protocols in *ns*-2 have a variable called `bytes_` to store the number of bytes received in the protocol's sink entity. The modified UDP protocol `eraUDP` however, lacks this variable. Thus, the throughput to an eraUDP-node can not be recorded with the same methods as for other protocol types. To change this, the OTcl-to-C++ interface for `bytes_` was added together with a line to ensure correct updates of the value in `evalvid_ra_sink.cc` and `evalvid_ra_sink.h`.

3.5 IO-handling

As mentioned in several sections earlier, the simulation process is not one single program thread but a compilation of several programs, all executed from various shell-scripts. Most of the programs take different relevant parameters as input and write results to binary files, text files and/or to the standard output buffer of the shell. Each program can easily be run directly from the command line, but because many of the simulation scenarios are best tested as iterative processes, shell scripts with loops are necessary (see Appendix B).

To coordinate the program executions, correct input must be given to each program, and their respective output must be named and stored in a place where the next program can find it, if necessary. This master study has led to the development of several shell-scripts:

manyQ.sh This script comes with the Evalvid-RA package, but is slightly cleaned up and modified to fit the video input. The script takes minimum and maximum quantization and filename as input, and executes *FFmpeg* and *mp4* iteratively. For example, FFmpeg is controlled with the following:

```
ffmpeg -r 30 -s 176x144 -i [original YUV video] -vcodec mpeg4
-g 12 -sc_threshold 20000 -qscale [Q-param] -s 176x144 -r 30
-y videoVariants/[video-name]_Q[Q-param].m4v
```

[3]A TFRC packet has 36 bytes of header while a UDP packet has only 28 bytes.

This line means that a video specified with the `-i` parameter, a frame rate of 30 fps and qcif size (176x144) should be compressed with mpeg4 to a m4v-file with quality specified in `-qscale`. When looped several times in manyQ.sh with different `[Q-param]`, this creates a library of the different available compression levels on the streaming server. Note that while the total system described here simulates an *online* encoder, the library of video files are encoded *offline*. Next, the mp4-utility uses the newly constructed compressed files as basis for trace file generation. Since it runs through all necessary actions to prepare a Evalvid-RA *ns*-2 simulation, it is called the *pre-process*. Since it is normally only needed to run this script once for each raw video that is used in the simulation, and since the process is time consuming, the pre-process is not included in the case controlling scripts.

case(n).sh Each case uses a shell script (the three variants are found in Appendix B) to run the simulation and its *post-process*. Each of these scripts enable the user to specify the number wanted of each service type, and to control which parameters are changed for each iteration of a simulation. It executes all programs necessary to obtain the wanted data (i.e. PSNR-value(s), delay or throughput) and displays the results in a graph with suitable layout and axis-texts. Figure 3.3 shows the typical sequence of an Evalvid-RA process end to end.

clean.sh This is a simple script to remove all temporary output from previous simulation, so that the next simulation is not disturbed by previous values. For instance, *mp4creator* concatenates its output to allready existing output with same name, so that a mp4-file might be several times bigger than it should be. Clearly, this affects the following programs in the process chain. Since all of the different processes provide runtime-messages and write one or several output files, the working folder can be quite messy and the user looses overview if the temporary files are not removed from time to time.

To sort out the relevant data from various tracefiles and other simulation output, Awk-scripts have been made (see Appendix C). These scripts search through the given input data line by line for a specified pattern, performs calculations on the found data and prints the result to standard out in the shell.

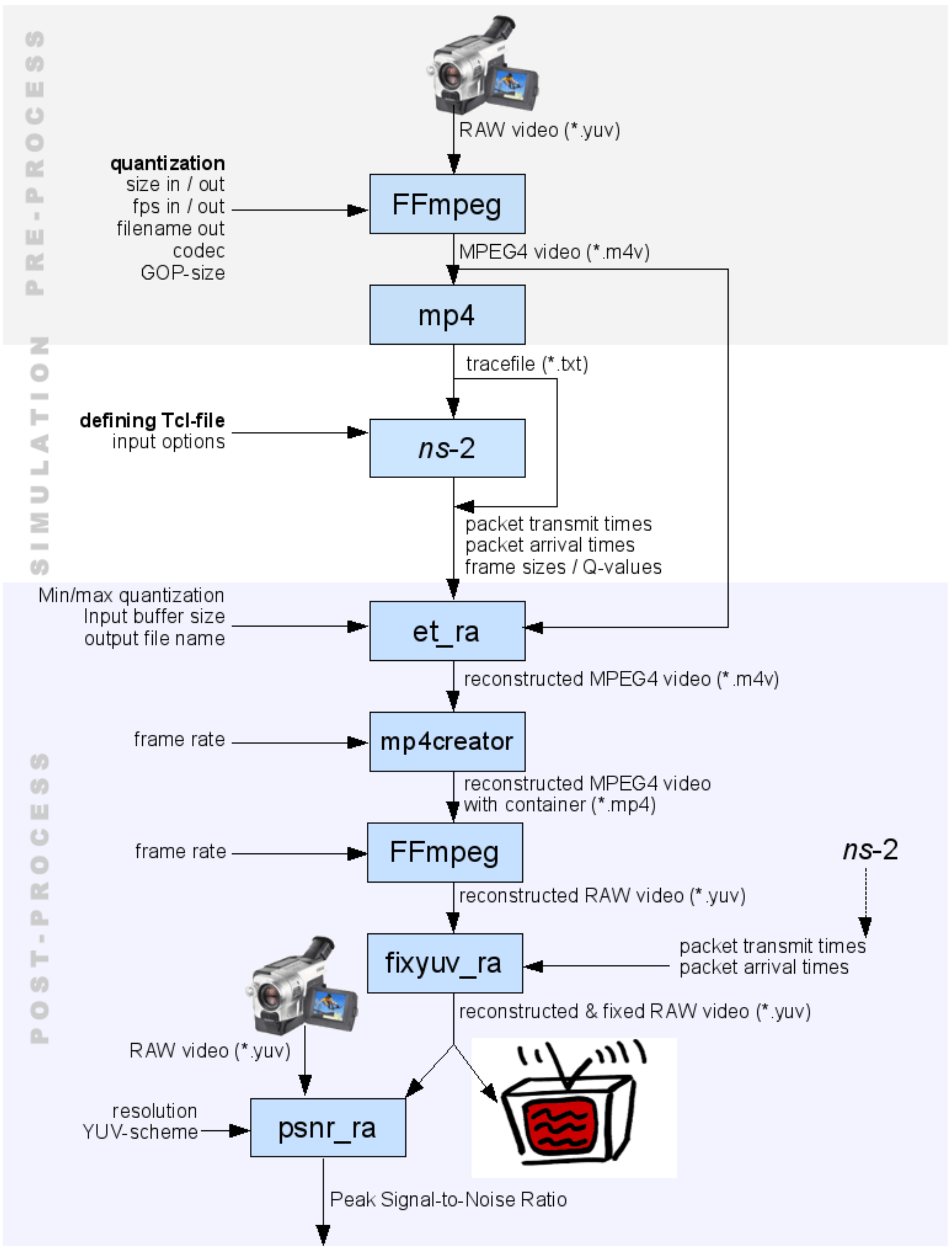


Figure 3.3 The complete interaction model between program instances for pre-process, simulation and post-process. This process can be controlled by shell scripts containing loops to do it several times with differing input parameters.

3.6 Wireless channel

The *ns*-2 comes with code and framework to simulate 802.11b networks with both ad-hoc and infrastructure topology. The implemented protocols feature MAC and PHY layer simulation with the possibility to set parameters to get close to specific real implementations. This is useful for all scenarios where the researcher wants to be as realistic as possible and if the results from a physical installation is to be compared with simulation results. However, the control interface in Tcl is not as intuitive as one might think and the default values of *ns*-2 are not always very useful.

In [44], J. Robinson walks through a set of features that should be added or modified to enable close to real-life simulations of 802.11b:

- The data rate of *ns*-2 is only 2Mbps by default. Today most 802.11-cards support 802.11*g* or even 802.11*n* with 54Mbps and 270Mbps+ theoretical capacity respectively, but since *ns*-2 seems to be most tested with 802.11*b*, the data rate should be set to 11Mbps with Tcl (`MAC/802_11 set dataRate_ 11Mb`). An unfortunate limitation with the *ns*-2 implementation, is that it does not support Auto Rate Fallback (ARF). The consequences of this is discussed further in section 4.2.2.

- In 802.11 MAC, the standard defines that a Request To Send packet (RTS) should be sent if the PDU size is above a RTS threshold. According to [44], most commercial 802.11 equipment for non-professional use is stripped for the entire RTS/CTS functionality to save transmission overhead. To simulate a home wlan, which is relevant in this study, the RTS/CTS can be effectively turned off in *ns*-2 by setting the RTS threshold to a value higher than the MTU of the protocol (`MAC/802_11 set RTSThreshold_ 3000`). This will be tested in Case III, section 3.7.3 (results in 4.2.3).

- The preamble used in the 802.11 PHY to synchronize sender and receiver directly in front of a packet transmission, has different size from implementation to implementation. However, the preamble size will be the same for all protocols in the simulations presented here and the possible inaccuracy involved is assumed to be minimal.

- The physical channel implementation models of *ns*-2 are said to be simple and inaccurate. For example, [45] argues that the threshold parameters in the simulator should be used together with a Bit Error Rate (BER) calculation to provide accurate real-life imitation. [44] recommends the use of specially designed channel model add-ons to simulate more realistic scenarios. An option is to integrate Rayleigh and Ricean fading models, but the pitfalls and added complexity of adding more unknown code to the *ns*-2 core taken into account, the existing channel models are considered sufficient for the simulations carried out here.

- A final interesting observation made in [44], is that the maximal achieved throughput varies substantially with different transport layer packet size. Evalvid-RA has 1000 bytes as MTU for both TFRC and UDP. This can imply a bandwidth reduction of about 1Mbps in standard 802.11b networks.

During the development and setup of the simulator controller script, several pitfalls where discovered. One example is the use of `Phy/WirelessPhy set bandwidth_`, a Tcl-command that can be seen in several of the example scripts that comes with *ns*-2. It was discovered that the values set here had no effect on the wireless throughput. After some testing and searching, the cause was found: the `bandwidth_`-parameter in `wireless-phy.cc` was commented out and in fact never used in the remaining source code.

Another example of a pitfall is that the interface queue of the wireless access point consistently drops the two first packets in a simulation if they are from a UDP flow. For video transfer, this is crucial since the first packet contains header information that is needed to decode the rest of the video stream. When these two packets are dropped, the ffmpeg decoder is not able to decode any of the subsequent data. The cause of this problem is not found, but it is solved by sending a downstream ping message from server to client before the initial video packets[4].

A third and last example is that the user is not notified by a misspelled parameter setting in Tcl. The consequence is that *ns*-2 runs just as normal, but with another parameter setting than the user expects, until the misspelling is discovered.

3.7 Scenarios

To test the simulation environment, link types, transport protocols and applications, different scenarios, or use-cases, had to be defined. In each of these, some result should be expected based on previous simulations and theoretical background information.

In this section three different cases are defined and explained. The two first cases are formulated based on results from [13] and are focused on the TFRC performance in congested and erroneous environments. The third case is made to illustrate how the MAC algorithms in 802.11 affect the performace in terms of delay and delay-variations (jitter), which are of high importance to real-time services. While the cases I, II and III are introduced here, their results are found in the results chapter under sections 4.2.1, 4.2.2 and 4.2.3 respectively.

3.7.1 Case I: Congested wireless link

"TFRC gives better results than UDP with high traffic load and interference"

This first case is based on the original theories behind rate adaptive video, namely that the video quality is better when the compression and transmission rate is adjusted to the changes in network congestion compared to static compression and high packet loss rate. Evalvid-RA is tested and verified to provide good scaling and QoS over wired congested networks, so in this work the performance in a congested wireless network will be tested. While the real bottleneck of a wireless network is the physical layer bandwidth limitations, the most interesting aspect of the network performance is how the MAC layer allocates the resources to the different traffic types and connected nodes under high traffic load. The CSMA/CA scheme with RTS/CTS generates a significant amount of chatting and overhead on the link. How this affects the video

[4]the script must get `-doPing 1` as input, see `proc ping` in Appendix D

quality with TFRC (rate adaptive video) and UDP (non-rate adaptive video) will be measured with PSNR and manually inspected.

The case is separated in two blocks with four different runs on each:

1. A TFRC-flow is monitored while different traffic types runs simultaneously:
   - From one to eight concurrent TFRC-flows.
   - From one to eight concurrent TCP-flows.
   - From one to eight concurrent UDP-flows.
   - From one to eight concurrent flows of mixed type.
2. A UDP-flow is monitored while different traffic types runs simultaneously:
   - From one to eight concurrent UDP-flows.
   - From one to eight concurrent TFRC-flows.
   - From one to eight concurrent TCP-flows.
   - From one to eight concurrent flows of mixed type.

The reason for this high number of different tests is to be able to pinpoint which "competing" traffic types that are critical to the video quality and to check inter- and intra-protocol friendliness (see definition in section 5.1).

To execute the simulations in this case, the shell-script `case1.sh` (found in Appendix B) has been made. The script basically does the entire simulation and post-process described in section 3.5 and seen in figure 3.3 for different numbers of "competing" traffic. Each iteration generates a bandwidth graph that displays the throughput to each wireless node. This is very useful to see the inter- and intra-friendliness of the different flows. At the end of the run, a graph displays the average PSNR-value for each iteration to illustrate how the over-all video quality is affected by an increasing degree of congestion as the number of concurrent flows grow. In addition to the program executions of `ns`, `ffmpeg` and so on, a little Awk-command is inserted to read out the resulting PSNR-value from the `psnr` program output. The script also arranges the resulting files for archiving and analysis.

3.7.2 Case II: Node movement and variable link quality

"UDP gives better results than TFRC with low link quality or noise"

Whereas Case I is meant to illustrate a *benefit* of using TFRC to control the video data transfer in a wireless environment, this case is meant to point out a *drawback* caused by the congestion control mechanism. The problem found in the pre-study [13] is that TFRC does not distinguish congestion losses from link error losses. In a wired environment, a packet is typically lost because of buffer overflow due to queues in intermediate routers. TFRC catches this event and adjusts the transmitting rate accordingly so that the subsequently sent packets get through the router queue. When TFRC is used as a bearer of rate adaptive video, reduced sending rate implies reduced video quality due to harder compression (higher quantizer value in MPEG-4). However, temporary reduced video quality is considered better for the overall quality perception than interrupts and freezes because of dropped frames and packets. In a wireless environment on the other hand, a packet can be lost because of a sudden propagation shadow, a hand-over, interference from other transmitting equipment or signal fading. In this case, it is not always helpful to reduce the bit rate, frame rate or packet rate since the link quality may still be poor and packets will be dropped anyway. The expected results from the tests performed in this case, are that beyond some link quality threshold, the video transported with TFRC will experience *both* a quality reduction due to quantizer increasment *and* a reduction due to link error drops.

To simulate varying link quality with *ns*-2 and 802.11b, the general idea was to adjust the distance of the nodes to and from the access point, possibly using different link layer propagation models provided by the simulator. In the development of a control script for this case, several problems were encountered. First, adjusting the distance of the nodes had no effect up to a limit of 16 meters. Beyond 16 meters the throughput fell from 100% to 0%. Second, the two models `FreeSpace`, which is signal propagation with no reflection, and `TwoRayGround`, which is propagation with one reflector surface, gave totally identical results for all distances. Third, the *ns*-2 option of using random node movement did not work[5]. Fourth, the more complex `Shadowing` propagation model gave segmentation faults at run-time.

Now, as recommended by [45], the Rayleigh and/or Ricean fading models could be implemented instead of the existing shadowing model, but luckily a satisfactory solution was found with the latter when a bug was discovered. The bug was related to the distance between mobile nodes and access points, and the reference distance used in the calculation of the received power within the shadowing model. When the reference distance d_0 and the node-to-AP distance d had a difference of more than one meter, the segmentation fault was avoided. The shadowing model then uses the following representation for power calculation:

$$\left[\frac{\overline{P_r(d)}}{P_r(d_0)}\right]_{dB} = -10\beta log\left(\frac{d}{d_0}\right) + X_{dB} \tag{3.1}$$

Here, P_r is the reception power, β is called the path loss exponent (controllable with `Propagation/Shadowing set pathlossExp_` in *ns*-2), and X_{dB} is a Gaussian random variable with zero mean and standard deviation σ_{dB} controlled by `Propagation/Shadowing set std_db_` in *ns*-2 [1, p.189].

To ensure a relatively linear signal quality degradation when increasing the distance to the access point, the values $\beta = 2.0$, $\sigma_{dB} = 4.0$ and $d_0 = 1$ were found to be most apt. The controller script now only has to vary the distance parameters in y and x directions to change the received signal quality. The final script is almost equal to the one described in Case I, only with iterative changes in the distance instead of iterative changes in flow numbers (see Appendix B).

3.7.3 Case III: MAC layer efficiency

"802.11 MAC gives significant added delay"

[5]In fact, the following message was found in the *ns*-2 manual: "We have not used the second method and leave it to the user to explore the details."

Since the wireless access network is a shared medium, the 802.11 MAC layer has buffer capabilities to hold the traffic until an available slot in the medium is available. The buffer size and queuing time depends on the algorithms used in the allocation of transfer times. Thus, the algorithm has a direct impact on the delay and jitter of traffic. As explained in section 2.6.3, the legacy 802.11 MAC uses several types of interframe spaces, a backoff procedure and the RTS/CTS mechanism to avoid in-air collisions of traffic from different STAs. This simulation case will illustrate the total delay introduced by these collision avoidance techniques. The results can be used to show the need for 802.11e as described in section 2.6.5.

Four different comparisons are made in this simulation case:

1. Comparing the delay variations to the throughput of one flow.
2. Comparing the average delay for different network congestion levels.
3. Comparing the average delay for different RTT values.
4. Comparing the delay with and without RTS/CTS.

To generate data for these comparisons, the VBR UDP traffic source from the Evalvid-RA is used. This traffic type is chosen because a) the video source generates variable bitrate which in turn is expected to give variable delay, b) the source is not adjusting to congestion, buffer overflow and jitter in the network and c) because it is band-limited[6]. Note that UDP traffic only generates one way traffic, thus giving the AP full control of all the medium access. To check how bandwidth variations and different input pressure affects the delay, the measurements are made with between one to six concurrent UDP flows to the wireless network. With more than four to five flows simultaneously, the network is assumed to be completely congested since the videos generate about 1.20 Mbps in average and the actual throughput of the link is found to be about 5Mbps[7].

The controlling script for this case, as seen in Appendix B, consists of a for-loop to execute three commands iteratively i times ($i = \{1, ..., 6\}$). The commands are the ns execution with `-nofUDP $i -doPing 1 -disableRTS_CTS 1` (the last parameter is 1 or 0), an Awk-script to gather all timing information for packets on the monitored flow (`added-delay.awk`, found in Appendix C) and one Awk-script to calculate the average delay for all packets in one flow (`avg-delay.awk`, found in Appendix C).

Since a resulting decimal number in milliseconds is only considered interesting information if the reader is experienced in the field and knows which values are common, the results are presented as the 802.11 *share* of the total end-to-end delay. With this presentation, the reader can more easily understand the impact of a wireless link. However, since these results can be misleading if the rest of the network is either very fast or very slow, the tests are run with several scenarios (that is, with propagation delays from 6 ms to 200 ms from sender to access point). The actual decimal values are also discussed in the contexts where it is useful.

[6]FTP-traffic as an alternative will utilize the full possible TCP throughput both for one and several concurrent flows.

[7]A statement that is confirmed by the results of all cases.

CHAPTER 4

Results and analysis

In this section, the most relevant and interesting results from the simulations will be presented. The results will be discussed, analyzed and explained next to illustrating graphs and tables. First, a TFRC feature or implementational bug will be presented as a vulnerability to the system, then the different simulation scenarios will be discussed, before the most important findings will be repeated in a summary at the end.

4.1 Vulnerability

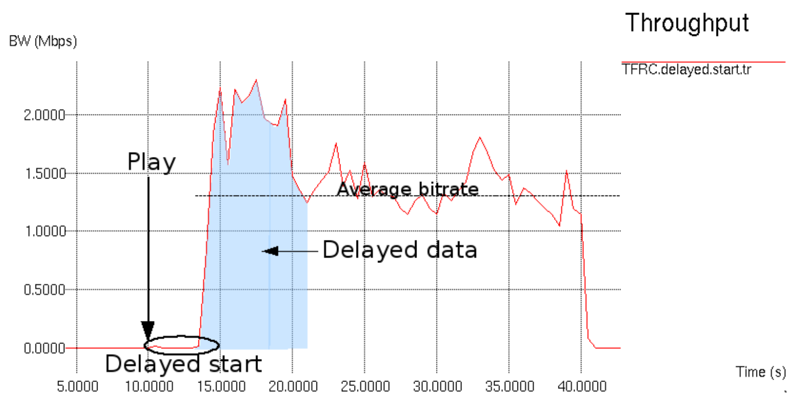


Figure 4.1 This graph shows the video throughput rate when the first TFRC acknowledgment is lost.

As discovered in [13], the TFRC protocol is vulnerable if the first packet is not acknowledged by the receiver. Figure 4.1 illustrates that the problem exists with the new topology and architecture as well. The example shows the flow from a TFRC source that is started at time

$t = 10s$. As can be seen on the figure and verified by the trace file, the first packet seems to arrive correctly (note the little bump in the circle). But since the acknowledgment is lost, the TFRC source refuses to continue the transfer. TFRC believes that the disappearance of the first packet is an indicator of a congested network, which triggers it to wait and retry. After about three seconds the sender gets an acknowledgment and starts the remaining transfer. The TFRC RFC [9] explains this behavior in chapter 4.2; TFRC has a transmit rate X that is initialized to 1 packet/second and a feedback timer that expires after 2 seconds. This suits well with the delay of about 3 seconds seen in figure 4.1.

Another observation from figure 4.1, is that when the traffic starts, it accelerates to a bitrate that is higher than what the compressed video can produce at best quality quantization. The video used in all simulations has a bitrate peak in the first seconds, but not as dramatic as seen in the shaded area of figure 4.1. The average bitrate at minimum compression (`q=2`) is

$$\bar{r} = \frac{size(q)}{duration} + overhead(pdu) = \frac{4.2MB}{30s} + 0.08 = 1.20 \text{ Mbps} \tag{4.1}$$

where $overhead(pdu)$ is the network and transport layer overhead in data unit framing and packet headers. Since the bitrate on the figure peaks at well above 2.1 Mbps, an explanation needs to be found.

As mentioned earlier, the feedback mechanism makes the video compression rate adjust dynamically to the reported congestion and loss statistics of the network. The TFRC daemon receives the feedback packets and adjusts its packet rate thereafter. Now, how is the application level being notified of the rate change and how may it react?

TFRC has an interface to the network layer, which lies beneath it in the protocol stack, and an interface to the application level above. The Evalvid-RA implementation uses the latter interface to read out the current transmission rate in use by TFRC. With this information, it calculates a new suitable video bitrate and a new quantizer parameter. The video frames with the new compression rate are fetched and queued in the input buffer of the TFRC application level interface. In the case shown in figure 4.1, the video transmission is started on the application level, but because TFRC does not receive the first acknowledgment from the client, it refuses to forward the flow it receives from the application level. Instead, this flow is buffered in the TFRC input queue until the network path is cleared for transmission. When TFRC finally gets its retransmitted header packet acknowledged, the queue will contain several seconds with video ready for transmission. Until TFRC stabilizes or receives new congestion feedback, the queue will burst out on the network.

The Evalvid-RA application starts by default at a quantization rate of Q=8 in this implementation. This is applied to the first GOP in the transmitted video. Normally, TFRC should receive one feedback for the very first packet, and thereafter with one or two RTT intervals. So by the time the first GOP, which can consist of several tens of packets depending on the video resolution and contents, is sent, one or several feedback messages is assumed to have arrived. If Evalvid-RA is not informed of a congestion by this time, it lowers the quantizer and is soon transmitting with full quality. The problem, or *implementational bug*, seems to be that the TFRC agent does not report that the first feedback message is missing to the application.

A solution suggestion is that the video application compresses its output maximally until proper feedback is received, thus minimizing the chances of packet loss when the buffered top quality data bursts out on the (possibly congested) network. The data backlog in the TFRC queue is illustrated with the shaded area in figure 4.1, and the interface between TFRC and the application is conceptually illustrated in 4.2.

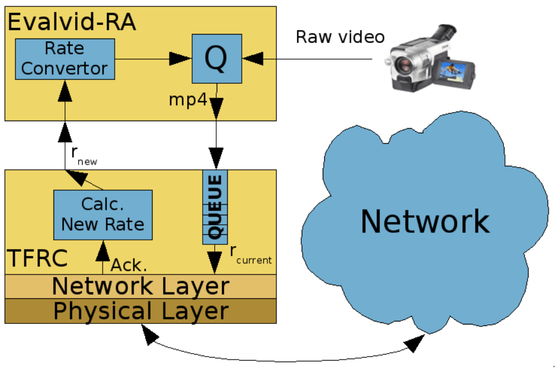


Figure 4.2 This figure shows how the incoming acknowledgment triggers the TFRC rate calculation mechanism which can report the new rate to the application layer. Here, the rate parameter is converted to a suitable quantizer value for the video compression. Compressed video is sent to TFRC input buffer before transmitted through the network.

4.2 Scenarios

The simulations are mainly presented with graphs showing PSNR values or throughput values. An example of a throughput graph is given in figure 4.3. This graph shows the basic shape of the video with minimum compression, Q=2. It is beneficial to know the basic shapes of the videos in order to easily interpret the results when the video-flow is damaged or band limited later on[1]. Three main observations should be made in this case. First, the video used is 30s long and comprised by 3 equal sequences of 10s (300 frames at 30fps). This can be seen by the three peaks of each transfer type in figure 4.3. Second, the TFRC throughput is slightly higher than the UDP throughput. This can be explained with the bigger header size of TFRC packets. Third, the TFRC flow is less aggressive in the first seconds. This is due to the slow start mechanisms employed in the congestion controller

.

4.2.1 Case I: Congested wireless link

In this case, the benefits of rate adaption and dynamic quality scaling is tested. To prove the hypothesis (“TFRC gives better results than UDP with high traffic load and interference”), expected results would be to see that the overall PSNR value for a video sequence would be higher with TFRC transport than with plain UDP transport.

[1]The same video is used in all the simulations presented here.

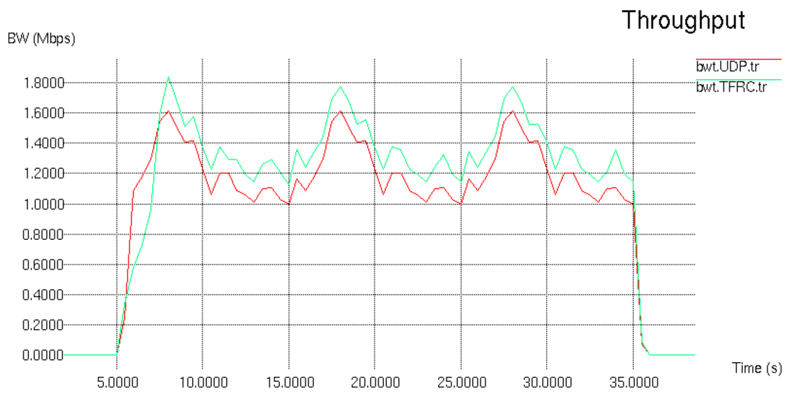
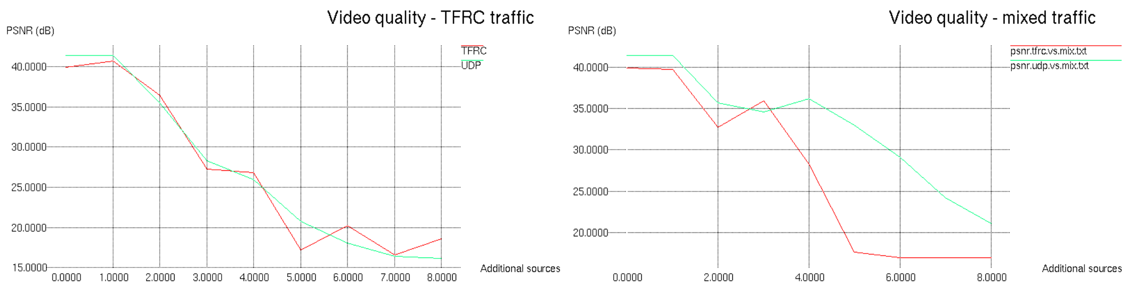


Figure 4.3 This graph compares maximal uninterrupted throughput for one separate UDP and one separate TFRC simulation run.

In figure 4.4, the results from both UDP and TFRC objective quality is shown for (a) only TFRC traffic running concurrently and (b) mixed traffic sources running concurrently (UDP, FTP, WEB and TFRC), and in figure 4.7(a) the results with only UDP competition. None of these graphs confirm that TFRC is superior in any way. In fact, the most realistic case which is the mixed traffic shown in 4.4(b), indicate that the UDP traffic provides the overall best results in objective quality.



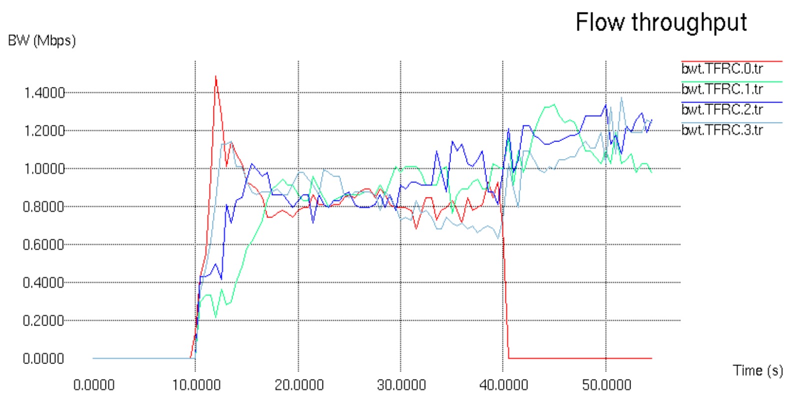
(a) With TFRC traffic (b) With mixed traffic

Figure 4.4 These graphs show UDP (green line) and TFRC (red line) video compared with concurrent traffic of two different types.

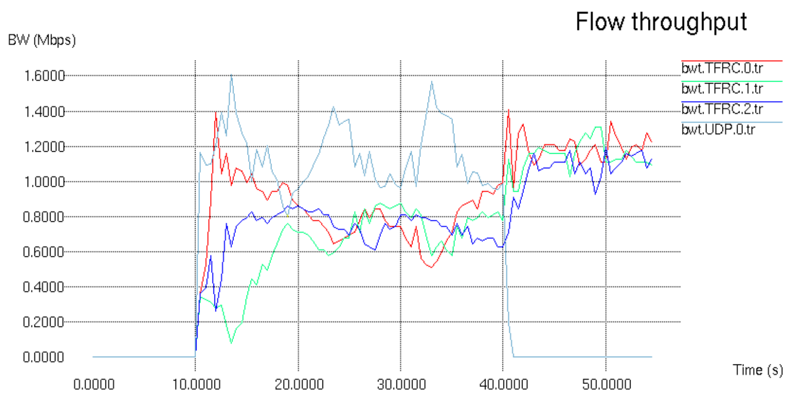
To take a closer look at what happens here, let us step into the details of the simulation iterations with 3 concurrent TFRC sources[2]. Figure 4.5 shows the throughput for all the simultaneous flows, where the first TFRC flow is monitored in a) and the UDP flow is monitored in b). What we observe is as expected: The UDP-flow has a significantly higher throughput to the end user in b) under the same circumstances as the TFRC flow in a). Now, while this graph shows the throughput, it does not show the amount of lost packets, a metric that is important due to the the interpacket dependency of MPEG video explained in section 2.1. An analysis of the receiver trace file produced by Evalvid-RA can give information on how many frames that

[2]The number 3 is randomly chosen for this example.

where accepted at the receiver node. The results here are also as expected for the UDP traffic; 332 out of 4958 packets where lost in this flow, which gives a percentage packet loss of 7%. With such a high loss rate, many header and reference frames (I-frames) may have been lost, causing severe damage to the video even if the the frames that arrive correctly have minimum compression.



(a) Monitored TFRC (red line)



(b) Monitored UDP (bright blue line)

Figure 4.5 These graphs show the throughput of the different concurrent flows when TFRC-flows are the "competitors"

In the TFRC flow, one would expect to have close to zero packet loss since TFRC should adjust the transmission rate until all packets get through. This is not the case here; The receiver trace file says that 51 out of 3108 packets, or 2%, where lost with compression rates of Q=3 and Q=4. So, TFRC still observes packet loss and the effect expected to be seen in Case II, that the adaptive stream is degraded due to both packet loss and quantization, is already present here in Case I. The fact that TFRC can not adjust perfectly to the loss type and rate is crucial, as will be discussed further in Case II (section 4.2.2).

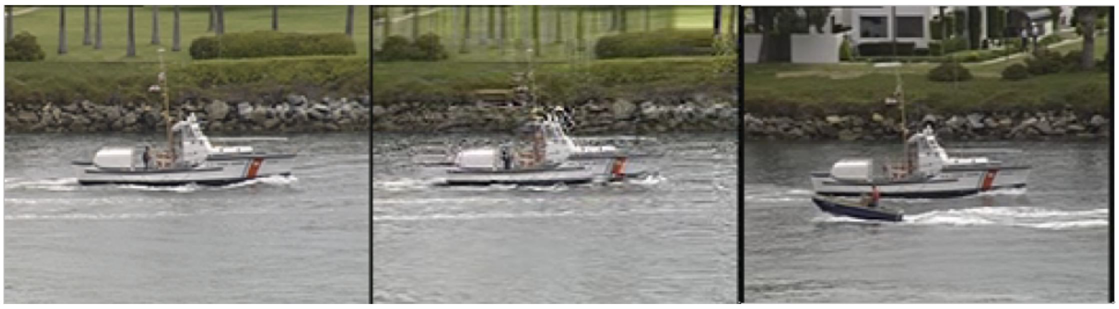


Figure 4.6 This is the same frame from three different versions of a video: The original (left), the non-adaptive (center) and the adaptive (right).

Subjective quality

So far we have seen that TFRC adjusts the transmission rate and forces Evalvid-RA to adjust the quantization level up. The adjustment does not seem to be enough, since the PSNR graph indicates that the all-over quality of the non-adaptive video is almost the same or better then the quality of the adaptive video from Evalvid-RA. As mentioned earlier, there are many ways to measure the quality of a video, but none of them are cheap, scalable and always correct. PSNR is one of the cheapest and easiest ways of measuring the quality of a video, but it has some flaws. For instance, while the PSNR measurement can see defects such as missing blocks or pixel errors, it fails to see if a series of frames compose a smooth video sequence or if it is jagged and uneven. Another is that it is not intelligent enough to synchronize the video in case of a bigger delay. The effect of delayed or early frames is not necessarily too bad for the end user, while the PSNR calculation compares each incoming frame with a reference frame from a time-shifted part of the original video, resulting in low SNR-values.

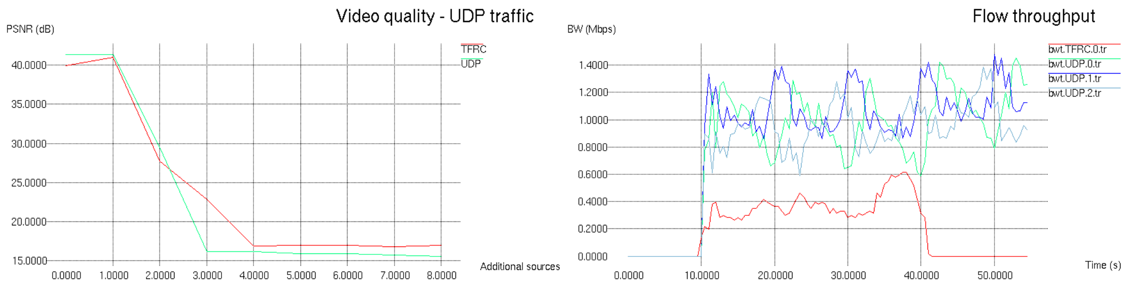
One of the superior qualities with the Evalvid-RA tool-set, is the possibility to inspect the resulting video for each simulation run. Together with the raw-file viewer VidView (referred to in section 2.4.6), one can expose if the PSNR value is correct or not for the sequence. Figure 4.6 illustrates how videos with the same PSNR value can have different appearance. Again, the example is from UDP non-adaptive versus three TFRC adaptive sources (4.6, center) and four TFRC adaptive sources (4.6, right). What can be observed is that the non-adaptive flow has had recent packet losses, disturbing the picture noteably but not crucially, while the adaptive flow shows a delayed frame (compared to the original (4.6, left)).

Inter and intra-protocol friendliness

The discussion in Chapter 5 will address properties of a *TCP-friendly* flow, and the different definitions of the term. Here, the term friendliness will be extended to represent if a flow is or is not damaging to the throughput and stability of other concurrent flows of the same or other type. The graphs in figure 4.7 illustrate the problems connected to UDP traffic, in fact the main reason for the wish to change UDP with DCCP: UDP traffic does not scale at all to the existing conditions in a router or a link. In figure 4.7(a) concurrent UDP traffic not only destroys the throughput and quality of the more discreet TFRC, but also destroys for flows of its same kind.

Allready with 3 concurrent flows, the monitored UDP flow is unusable ($PSNR < 20dB$, see section 2.2 for more), while the TFRC flow scales the rate to provide a barely watchable result. The throughput of this flow is seen in figure 4.7(b).

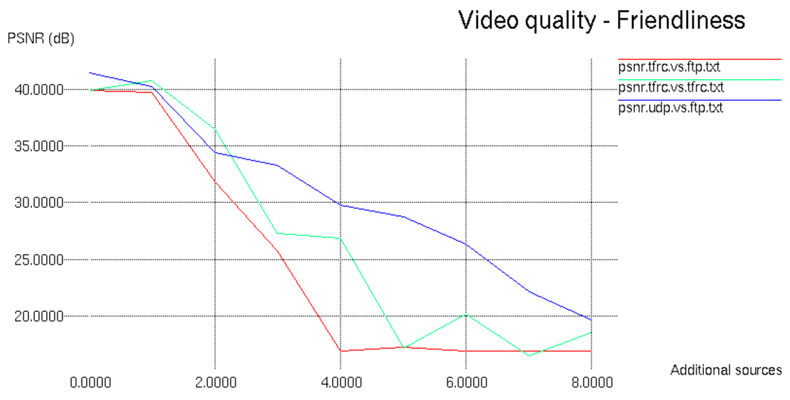
It is interesting to watch how the PSNR-to-throughput-ratio differ between the two flows in this example. While the PSNR values are nearly equal for all but one measurement, the throughput differ significantly. The UDP flows compete to get the full rate through, thereby destroying each other's flows with interference, while TFRC is forced to scale down to the lowest quantization values. The best example from the performed simulation is shown in figure 4.7, where the TFRC flow provides *better* PSNR than the UDP flow, while having a considerably lower bandwidth consumption with an average throughput of about 380 Kbps.



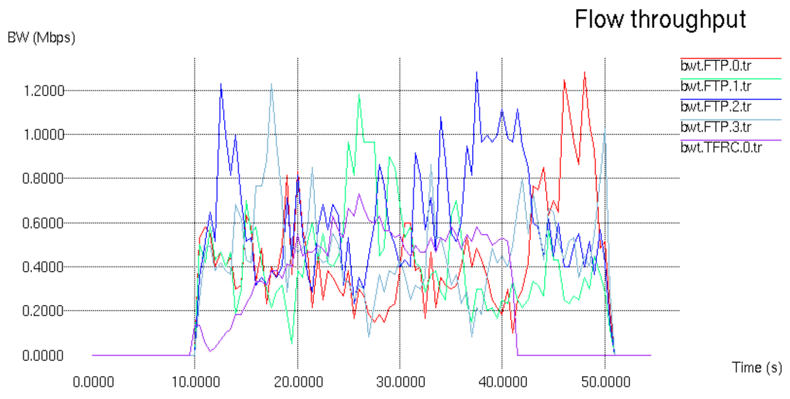
(a) PSNR (b) Throughput (TFRC=purple line)

Figure 4.7 This graph displays the quality of a video from a UDP or TFRC source measured in PSNR (a) and throughput (b) when the concurrent traffic on the link is only UDP.

The classical friendliness test does not include UDP because there is no doubt about its lack of friendliness. A new transport protocol suggestion is rather compared against TCP traffic, since this constitutes the main share of today's Internet traffic. In these simulations, rate adaptive video over TFRC was tested against FTP over TCP. These simulations reveal that TFRC is apparently TCP-friendly, but not very "TCP-robust". What can be seen in figure 4.8 is that the TFRC flow adjusts its rate to fit the bandwidth requirements of the concurrent TCP flows (4.8(b)). In 4.8(a), we can see how TFRC "suffers" from its friendliness (red line) whereas UDP gets a far better performance, strangling the other flows on the link (blue line). Since TFRC and TCP have almost equivalent rate control mechanisms, one would expect to see that a monitored TFRC flow has the same performance in an environment with TCP or TFRC traffic. Figure 4.8(a) shows something else: The TFRC flow in the TFRC environment (green line) shows an all-over better performance than the TFRC flow in the TCP environment (red line). Figure 4.8(b) is taken from the inflection point where four TCP flows result in an unwatchable video with PSNR below 20dB. Compared to figure 4.5(a), where TFRC shows superb intra-friendliness as the four flows adjust to almost equal rate in a few seconds, figure 4.8(b) shows a highly fluctuating behavior from all the TCP flows that results in poor bandwidth utilization and interference-caused packet loss. This is the observation referred to as not "TCP-robust" above.



(a) PSNR



(b) Throughput

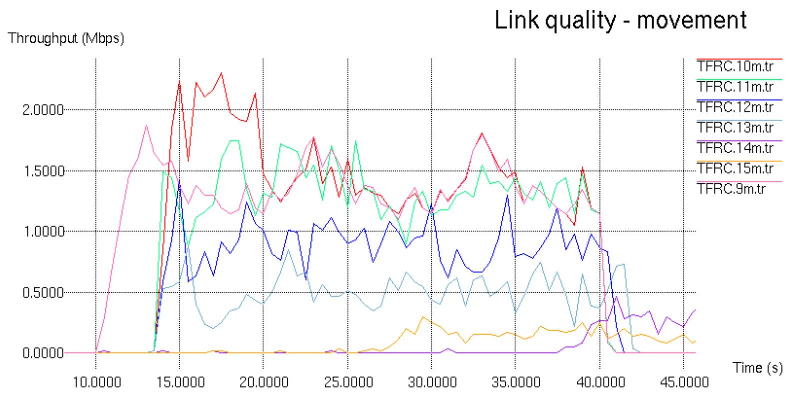
Figure 4.8 This graph illustrates the TCP-fairness of TFRC in PSNR (a) and throughput (b).

4.2.2 Case II: Node movement and variable link quality

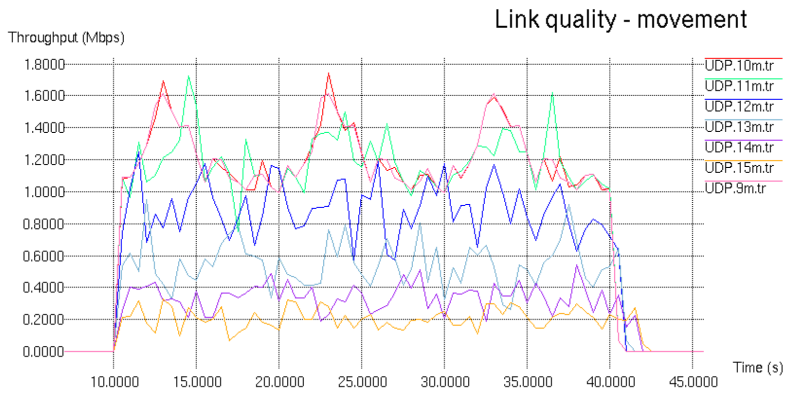
In Case I, the video flows where tested against flows from different transport protocols to see how the videos where influenced by the different interference pattern the concurrent traffic formed. In Case II on the other hand, interfering traffic is not interesting and adaptive and non-adaptive traffic will be compared by results from separate simulation runs. The only variable to be changed here is the distance between the client and the access point, thereby creating controlled variance in signal strength. As stated earlier, the expected result here is that the adaptive video source will experience an “amplified loss” due to both high compression quantization and random packet loss in the wireless link.

Figure 4.9 shows the throughput from the two transfer types in question; UDP and TFRC. Each line describes the throughput for one separate simulation of one flow over the specified distance. For all distances from 0 to 8 meters, the throughput is more or less equal for all flows with the propagation model set up in section 3.7.2, but when the receiver node is moved further away, the link fading gets an effect on the received throughput.

The figure shows us what one would expect; The UDP throughput is higher than the TFRC



(a) TFRC

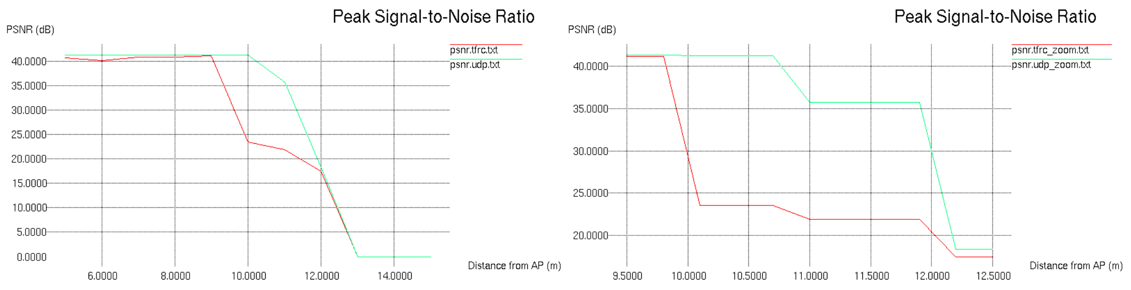


(b) UDP

Figure 4.9 Throughput graphs for TFRC (a) and UDP (b) for distances 9 to 15 meters from the access point.

throughput for all distances where the signal strength is close to the reception threshold of the receiving wireless interface. In these cases, the UDP flow would push through on a best effort basis, thus exploiting the available resources maximally, while the TFRC flow adjusts the transmitted rate down in an attempt to avoid what it assumes to be congestion. On distance 14 and 15 meters we see that TFRC has adjusted all the way down to the initial base rate of one packet per second with two seconds feedback timer (figure 4.9(a), note the small bumps circa every third second), while the UDP flow works at a 200 to 400 Kbps bitrate. Intuitively, this would imply that UDP performs better at these distances. However, as can be seen in the PSNR graph in 4.10(a), the video client can not decode any of the received material because the loss ratio is too big and the required throughput of a flow that transfers data with minimal compression is high.

Figure 4.10 shows two versions of the same effect. Figure 4.10(a) shows the over all PSNR values for each distance from 5 to 15 meters for both adaptive and non adaptive video. The observations agree with the throughput results since the PSNR values are almost equal up to about 9 meters, and a PSNR of zero is indicated from 13 meters and beyond. Now, since the only deviation is in the area between 9 and 13 meters, a variant of the simulation described in section 3.7.2 was made to get a higher resolution of the results here. The output from those simulations is shown in figure 4.10(b). This graph confirms our suspicions; *When packet loss is frequent, random and not due to congestion, TFRC provides poor performance compared to UDP as a video bearer*.



(a) 5 to 15 meters (b) 9.5 to 12.5 meters

Figure 4.10 In (a), the PSNR results from a video streamed to a client with different distances to the AP is shown. Figure (b) is a re-sampling of the most interesting area of the curves in (a).

The assumptions made to form the hypothesis for this case (“UDP gives better results than TFRC with low link quality and noise”) was, as stated repeatedly, TFRC’s ”amplified loss“. But while the PSNR-graphs in figure 4.10 alone confirms the *result* of this effect, the throughput graphs in 4.9(a) gives reason to do a further analysis of the *cause* of the effect. What can be observed in the latter figure is namely that the delayed-start effect discussed in 4.1 seems to occur for all simulations with distance above 9 meters. Therefore, it is possible that this alone is the cause of the inferior PSNR values in the 9 to 13 meter region. To find the answer, the tracefiles for the different simulations must be studied.

Distance (m)	$\overline{Q}$	Lost (pkts.)	Total (pkts.)	Loss rate (%)	PSNR
9.8	2.04	0	4856	0.00	41.14
10.1	2.07	2	4777	0.04	23.51
11.0	2.34	16	4308	0.37	21.89
12.2	3.49	11	2946	0.37	17.47
13.0	6.90	30	1751	1.71	0.00
14.0	2.42	30	644	4.60	0.00
15.0	7.90	66	609	10.84	0.00

Table 4.1 This table summarizes the loss statistics of the simulated TFRC flows.

Several things should be noted about table 4.1 and 4.2[3].

First the numbers shown, are not from measurements with equal distance difference in meters. Actually, allthough the simulations where done with a resolution of 3 dm between 9.5 and

[3]Note that the results are partly based on random processes, which produces slightly different numeric results with other seeds.

Distance (m)	$\overline{Q}$	Lost (pkts.)	Total (pkts.)	Loss rate (%)	PSNR
9.8	2	0	4959	0.00	41.38
10.1	2	2	4959	0.04	41.26
11.0	2	18	4959	2.37	35.70
12.2	2	1339	4959	27.00	18.36
13.0	2	2766	4959	55.78	0.00
14.0	2	3563	4959	71.85	0.00
15.0	2	4097	4959	82.62	0.00

Table 4.2 This table summarizes the loss statistics of the simulated UDP flows.

12.5 meters, the results where identical for all samples left out of the tables. This can also be seen by the flat segments in figure 4.10(b). The reason for this remains unknown, but presumably, it has to do with the construction of the *ns*-2 implementation of the propagation model.

Second, the total number of packets depends on the quantization level and the MTU used. Both TFRC and UDP have MTU=1000 bytes. The UDP transfer has Q=2 for all GOPs, which gives a total of 4959 packets to be transmitted for all simulation runs, header packets included.

Third, while the UDP transfers show strongly increasing and logical loss rates, the TFRC flows do not. While the number of lost packets rise from 2 to 16 between 10.1 and 11.0 meters, it falls to 11 on 12.2 meters. Still, the PSNR value for 12.2 meters is far lower than for 11.0. What we see is that the average quantization level, $\overline{Q}$, rises notably and the number of total packets sinks correspondingly.

Fourth, the numbers given for distances 13-15 in table 4.1 do not tell the entire truth. The quantization parameter should have been close to 31 (maximum) for these simulations, but since the client feedback never arrives the sender, the TFRC agent keeps transmitting with initial settings (1 packet/second), but does not tell the Video application that the network feedback is missing. Therefore, Evalvid-RA will continue to produce video at Q=2 until a proper feedback has been reported. This happens at t=37s for 14 and t=25 for 15 meters wireless distance. This start-up delay causes most of the video sequence to arrive the client *after* the simulation is finished, which is at t=55s.

Now, with all of the above in mind, three questions arise:

1) How does the variable delay on the adaptive video affect the end result?

In this case, the UDP traffic is never delayed since there is no congestion in any parts of the network. The TFRC traffic is delayed if the first feedback is lost. This results in a delay of several seconds of the video, which makes the PSNR-calculator compare each incoming frame with an offset to the original. For example, with 3 seconds initial delay, 10% of the sequence will be delayed at the receiver. Now, in a real-time system, such delays are not acceptable. Neither are they in this simulation environment. When the video is received and reconstructed, the system uses a parameter called Play out Buffer (PoB) to limit how big the delay can be. Figure 4.11 shows the PSNR values for each reconstructed raw frame. The PoB (set to 250 ms

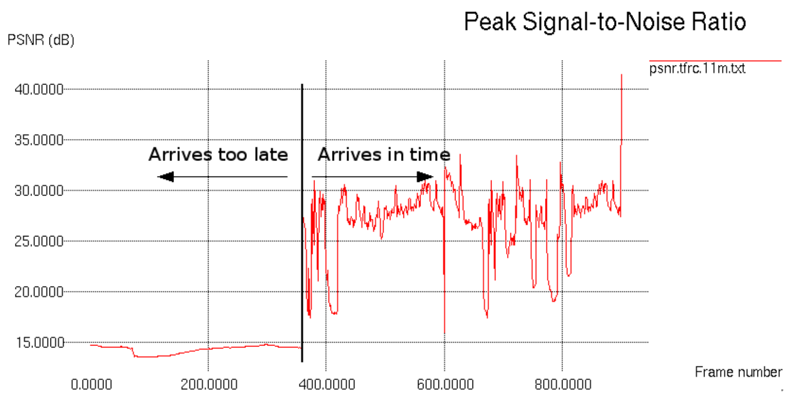


Figure 4.11 The PSNR graph for all frames in the video sequence transmitted with 11m wireless distance.

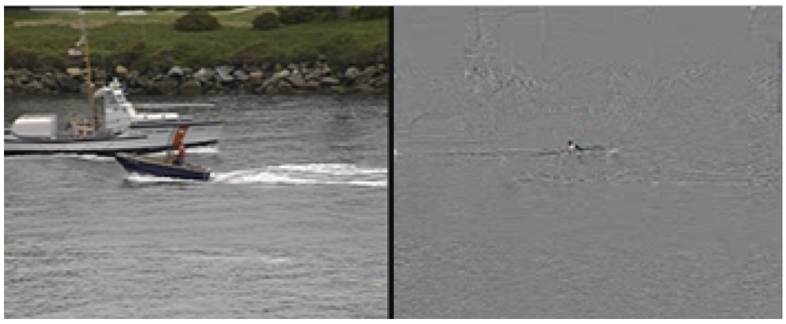
in all simulations) limits which frames that are included in the final video and which are not.

As earlier, VidView allows us to check these results by manual inspection of the reconstructed video. In figure 4.12, the original frame is compared to the reconstructed frame after transmission. First, frame 350, which is in the ”Arrives too late“ area of 4.11 is compared to the original frame 350. Ten frames later the first actually accepted and reconstructed frame arrives with good quality. This frame is sent from the server application at t=12s and received by the client application at t=15s, which means that it is the first of the buffered frames to be accepted.

2) How important are the header frames in the loss statistics?

When Evalvid-RA downgrades the video quality, the video stream from the application down to the TFRC-agent is reduced. Therefore, the TFRC agent does not have to segment each video frame into as many packets as is the case for a higher quality stream. One effect of this packet-rate reduction, is that the rate, or density, of absolutely necessary packets increases. As explained in section 2.1, the I-frames have a much bigger importance than the other frames, and the packet containing the first segment of an I-frame *must* be received in order to decode the rest. Furthermore, the first header-frame, which is sent before the first I-frame, is also necessary for proper decoding. To prioritize the importance of the packets:

1. The first packet contains the first header-frame, which the decoder *must* have in order to decode the subsequent stream.
2. The first packet of each I-frame, which contains the quantizer scale parameter for the entire frame and is necessary to decode the frame.
3. The first packet of each P-frame, which contains the quantizer scale parameter for the entire frame and is necessary to decode the frame.
4. The first packet of each B-frame, which contains the quantizer scale parameter for the entire frame and is necessary to decode the frame.


(a) Frame 350

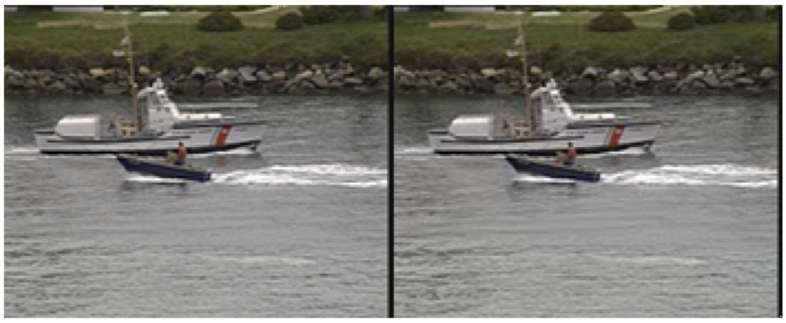

(b) Frame 360

Figure 4.12 A comparison of the reconstructed frames 350 (arrives too late) and 360 (arrives in time) to the original frames.

Now, an increased density of ”crucial“ packets will increase the probability of a crucial packet drop. For example, from table 4.1 and 4.2, it can be seen that TFRC and UDP have sent 2946 and 4959 packets respectively. If we define the two first types above as crucial, both streams will have H-packet + [header I-packets] $= 1 + 75 = 76$ ”crucial“ packets. The TFRC stream will have $\frac{76}{2946} = 2.5\%$ density while the UDP stream will have $\frac{76}{4959} = 1.5\%$, *a reduction of 40%.*

How can this effect be seen directly on the result presented here? Well, they can not. The reason is mentioned in section 2.4.4: The Evalvid-RA program that simulates the packet-to-video-stream reassembly, defines one entire frame as dropped if one or more belonging packets are lost. Since the TFRC stream under lossy conditions produces fewer packets per frame, it is reason to believe that the density effect is reversed, making the UDP stream most vulnerable. The total effect and analysis of packet drop versus frame drop is left for future work.

3) Finally, is the quality degradation of the adaptive flow due to delay or ”amplified loss“?

The above analysis of trace files and PSNR data started due to the suspicion that the degraded quality of the adaptive flow was caused by the somewhat buggy start-up delay, or if it is caused by packet loss in the weak wireless link. The data presented above does not provide a clear answer. The TFRC flow is delayed, something that definitely has an effect on the overall result, but additionally, the flow does experience packet losses even if the rate is reduced. The conclusion must be that in this case, UDP is superior because TFRC has controlled quality reduction, random packet losses *and* a start-up delay. The works of TFRC with a better implementation where the traffic is not jammed initially is left for future work.

The results in this case have limitations when compared to a real-life situation. As mentioned in section 1.2, *ns*-2 limits the simulation by lacking an Auto Rate Fallback (ARF) implementation. In all simulations presented here where the link quality is discovered to be insufficient, a real 802.11b network would fall back to a more robust channel encoding with a lower bit-rate. As discussed in [46], the available algorithms for ARF depends on the PHY-layer implementation and chip-sets, which can explain why it has no proper *ns*-2 implementation known to the author.

4.2.3 Case III: MAC layer efficiency

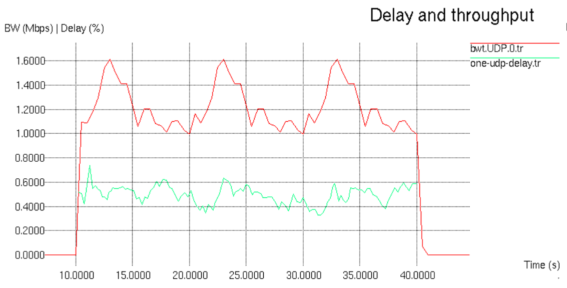
The hypothesis of this case is "802.11 MAC gives significant added delay". As stated in section 3.7.3, the theory about 802.11 MAC gives reason to believe that this wireless transfer protocol introduces a significant delay to the traffic. The case description promised the following four comparisons:

1) Comparing the delay variations to the throughput of one flow:

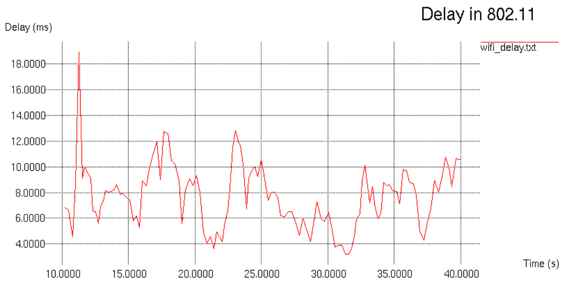
This first test is meant to show how the delay varies with different input rates to the shared medium. The test is performed with a VBR Video with maximum MPEG-4 quality (Q=2) and UDP transport protocol. The delay is measured and compared to the total end-to-end delay of the link. In this first test, the three links from the server to the AP have a delay of 2ms each, thus modelling a small and fast network. The result is given in figure 4.13(a). What can be observed here is mainly that the delay is variable with an average delay share of about 50%. No jitter values are caluculated from the simulations presented here, but the delay variation observed is 15.7 ms from extreme to extreme (for t=11.3 and t=31.6 respectively), *more than two and a half times the aggregated delay of the wired links*. This fluctuating delay is better illustrated in figure 4.13(b). It can also be observed that the delay has a tendency to grow when the bandwidth peaks, however the shape of the delay curve is not strongly following the bandwidth curve, and since the simulation depends on random processes, the empirical value of this observation is low.

2) Comparing the average delay for different network congestion levels:

Because the amount of overhead on the 802.11 link is strictly dependent on the amount of traffic demanding access to the shared medium, it is interesting to see how the delay grows with increasing competition for transmission time. Figure 4.14 shows the delay share of total end-to-end delay for one to six concurrent flows. In figure 4.14(b) we see the delay for one monitored UDP flow with different amounts of competing flows. We see that the variation patterns show little or no corelation, and the delay for a flow with one competitor is not strictly higher nor lower then the delay for a flow with zero or two competitors. However, when calculating the delay average for each flow, as is done in figure 4.14(a), the delay is almost linearly increasing



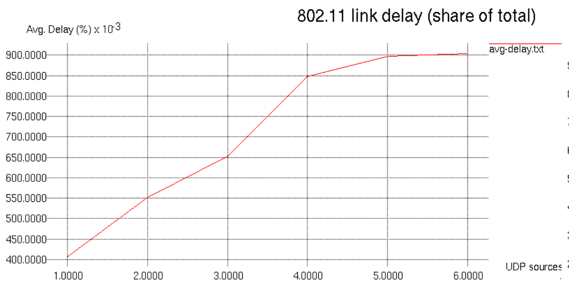
(a) Delay share and throughput



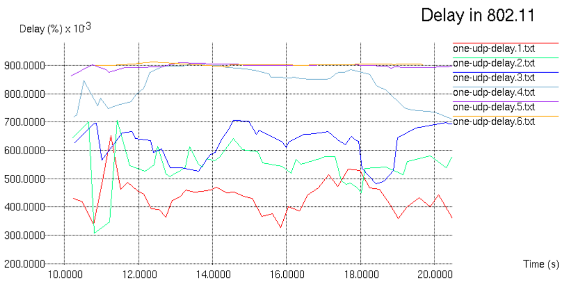
(b) 802.11 delay in ms

Figure 4.13 (a) shows the data throughput (red) and corresponding MAC layer delay share (green) on a shared axes (the y-axis is share 0 - 100% for green curve and bandwidth in Mbps for red curve). (b) shows the fluctuating delay measured from the AP to the client node.

up to a point of total network congestion between four and five concurrent video flows. Despite that the delays for the totally congested network constitute almost ten times the wired delay, it is more interesting to note that allready at three concurrent flows, which sum up to about 3.6 Mbps, the MAC layer seems to have problems delivering all data in real-time. At this point, the 802.11 MAC layer spends 65 %, or 11 ms, of the total end-to-end delay experienced by the packet.



(a) Average delay share



(b) Delay for one flow with increasing congestion

Figure 4.14 (a) shows a linearly increasing delay share for increasing traffic pressure. (b) shows the packet by packet delay for the same flows in a 10s period of the transfer.

Now, the way the results are presented here, it seems like the 802.11 MAC is a hopelessly breaking obstacle for all real-time services. However, the delays are compared to a fast network with only 6 ms total delay from server to AP. Compared to the tolerable delay in a video conference, which is about 150 ms, or compared to the play out buffer limit used in the previous cases (PoB=250 ms), a delay of 11 ms is not so hopeless after all. This can best be seen if this experiment is done with a slower transport network and higher RTT values.

3) Comparing the average delay for different RTT values:

Delay and delay variation are especially troublesome for duplex services, such as video conferencing. To test how the wireless link affects the total delay for some different realistic scenarios, the delay recording simulation has been run with different core delay as input (`ns`

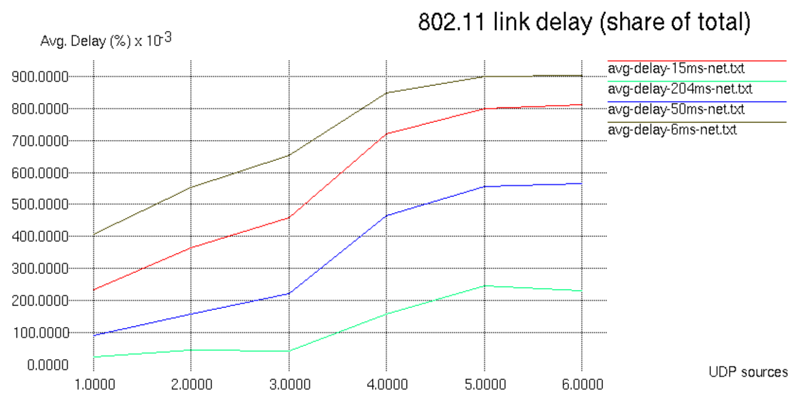


Figure 4.15 This figure illustrates the impact of using a wireless access network in a regional/national (brown), national/multi-national (red), continental (blue) and global (green) connection.

`-core_prop [delay]`)[4]. The original topology has a total delay of 6 ms, which is a typical delay for a regional to national network. The simulations have been run with 15 ms total delay to imitate a national to medium range international networks (i.e. for a Scandinavian video conference), 50 ms total delay to imitate a continental network (i.e. with end nodes in Norway and Spain), and finally with 204 ms total delay to imitate a global network (i.e. Norway to Japan).

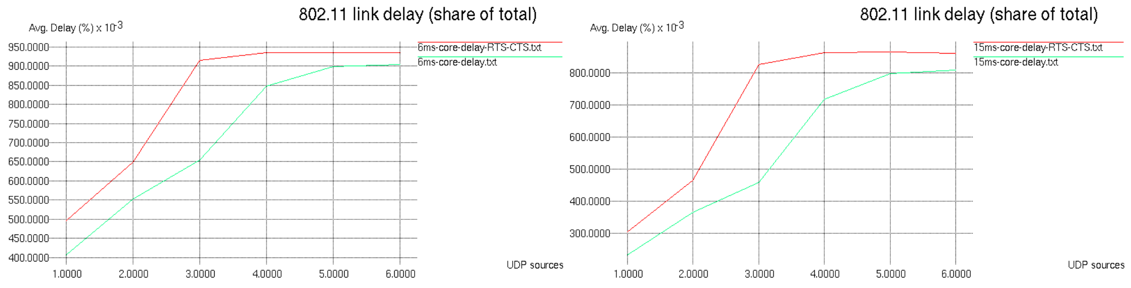
As expected, when the total delay increases, the 802.11 share of the delay decreases. The illustration is useful to see that the wireless network has a considerable effect on the QoS of a video conferencing service. For example, if a user wants to connect to a video meeting with colleagues in Madrid via a moderately crowded AP, the first and last meters from the client to the AP constitutes 22% of the total delay for each packet travelling the distance of about 3700 km. This delay can be be the factor that prohibits sufficient QoS for a requested real time service.

4) Comparing the delay with and without RTS/CTS:

The simulation results shown so far in this case have illustrated how a wireless network can be problematic for a real-time service, however, as described in section 3.6, many normal commercial APs today do not have the RTS/CTS handshake mechanism described in section 2.6.3 enabled or implemented. So all results shown above are actually *without* RTS/CTS. Now it is time to see if and how the delay increases when it is enabled.

The two graphs in figure 4.16 compare the average delay for a UDP flow with zero to five other flows demanding resources on the same link with and without RTS/CTS. At least three observations should be made. The first and most obvious is that RTS/CTS does, as expected, add a significant extra delay to the medium access process. Second, the RTS/CTS overhead increases with the traffic pressure. The third and most important is that with the added delay, the RTS/CTS curves reach an upper limit (when the network is totally congested) at a smaller traffic pressure than the non-RTS/CTS curves. With two video streams of totally 2.4 Mbps on a

[4]The estimated propagation time values are made based on ping polling of different national and international servers on a Saturday morning (assuming low to medium network congestion in average in this period).



(a) 6 ms transport network delay (b) 15 ms transport network delay

Figure 4.16 This graph compares the delay share of 802.11 MAC with and without the RTS/CTS mechanism.

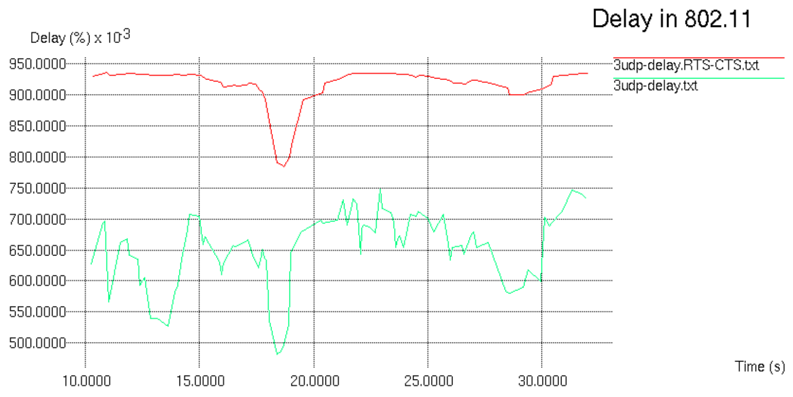


Figure 4.17 A comparison between the continuous delay share of a flow when RTS/CTS is turned off (green) and on (red) with two additional competing flows in the network.

link with 6 ms propagation to the AP, the wireless interface utilizes *65% of the total end-to-end delay* according to figure 4.16(a)(UDP sources = 2).

Figure 4.17 illustrates the packet by packet delay for one of three concurrent UDP flows in an 802.11 network with 6 ms propagation delay from the server. The example given shows the biggest observed difference between a simulations with and without RTS/CTS. At first glance, this presentation says that the RTS/CTS solution has about twice the delay of the other, but with far less jitter. However, both figure 4.17 and all the above that present the *share* of total delay do not tell the entire through about the delay size. Let us consider an example from the biggest gap in figure 4.17. What is the difference between 52.5% and 92.5% delay share? With 6 ms transport delay up to the AP, 52.5% share is 6.63 ms. When the 802.11 network constitute 92.5%, the *delay size has increased to 74 ms*.

4.3 Summary

Case I could not definitely prove that TFRC is superior to UDP when the wireless network is the bottleneck. The results show that TFRC does not adjust perfectly, and packet losses occur even if the video quality is adjusted downward (higher Q-values) with Evalvid-RA. However, it is shown that TFRC obtains a better quality with far lower throughput than UDP. This indicates that TFRC has a more economic resource utilization than UDP. Further protocol comparisons show that TFRC has a smoother sending rate than TCP, but the fluctuating TCP-peaks seem to force the TFRC throughput to be slightly lower than it should be for ideal friendliness.

Case II show interesting results by indicating the expected “amplified loss”, alas, due to the vulnerability explained in section 4.1, the results are not good enough to illustrate this expected effect. It is however no doubt that it is present in real life (as stated in e.g. [47], [48] and [49] for TFRC and numerous TCP articles).

Case III illustrates that the delay introduced by the wireless last hop is significant both with and without the RTS/CTS scheme. These results support the expectations of 802.11 being a challenging hindrance for strict QoS in high bit-rate real-time services.

CHAPTER 5

Standards development

So far, the theories behind a virtual video streaming solution have been given account for, the simulator layout has been described and simulation results have been presented for TFRC as a congestion controller for real-time video, and for 802.11 as an access network for mobile nodes. Neither TFRC nor 802.11 has proved to be perfect for the task when strict QoS is needed for a real-time service in a wireless environment. The protocol mostly used for this today, UDP, causes as shown severe problems for other traffic flows on the shared resource. Thus, the service providers, the network owners, the end users and the Internet itself demand a new common platform and standard to transport real-time video reliably, effectively and in an economic, scalable and TCP-friendly fashion.

This chapter aims to provide an overview of some of the different approaches to solve this problem (from here on referred to as *solution suggestions*) and to highlight some essential problems connected to the development of a new Internet standard.

5.1 Definition: TCP-friendliness

An essential part of the requirements for a new protocol is that it must be TCP-friendly or TCP-fair. This term has been used and referred to several times in this thesis in the sense that a given TCP-friendly protocol should not be greedy with the resources, thus strangling the throughput of congestion controlled flows such as TCP. Since this property is a absolute demand for a new real-time media bearer [3], many of the researchers use it to test weather their solution suggestion is acceptable or not. Now, if there is no clear definition of this requirement, the research communities will pull in each their direction with each their clear idea of how to measure TCP-friendliness. The paper of S. Floyd and K. Fall from 1999 [3] is often referred to in this need for a clear and credible definition. It says:

"We say a flow is TCP-friendly if its arrival rate does not exceed the arrival of a conformant TCP connection in the same circumstances."

The fairness index of Chiu and Jain [50] from the 1980's, a time when neither today's network topologies nor traffic patterns where considered, is often applied to verify if a non-TCP-flow fulfills the definition from [3]. In recent research, other definitions have come up. For instance, in [51] by Merani et. al. from 2004, TCP-friendliness is defined as:

"The non-TCP flow's long-term throughput should be neither higher, nor lower than the long-term throughput of a TCP connection running on the same path, experiencing the same network conditions."

Why can not the non-TCP flow have a lower throughput than the TCP-flow? It is doubtingly that a non-TCP flow would harm the TCP-throughput by applying a *lower* pressure on the resources. Additionally, and more important, in an environment where TCP does not perform optimally, should not a non-TCP flow be able to perform better than TCP and still be TCP-friendly? With this definition as a basis, Merani et. al. conclude that TFRC tends to "over shoot" the competing TCP flows, hence motivating for a parameter tuning in TFRC. The authors quote an article from Widmer et. al. [52] from 2001, where a more general definition is given just below the one used by Merani et. al. in numerous articles ([51], [53] and [54], all studies of TFRC performance in wireless settings). Widmer's definition is as follows:

"TCP Friendliness for Unicast - A unicast flow is considered TCP-friendly when it does not reduce the long-term throughput of any coexistent TCP flow more than another TCP flow on the same path would under the same network conditions.
TCP Friendliness for Multicast - A multicast-flow is defined as TCP-friendly when for each sender-receiver pair, the multicast flow has the property of being unicast TCP-friendly."

With this definition a non-TCP flow can outperform TCP and still be TCP-friendly. This property must be adopted in order to understand the challenges involved in the development of a new and improved transport protocol solution for heterogeneous networks. Sanadidi et. al. have done this when evaluating TFRC with the performance issues of both TFRC and TCP over lossy links in mind [47]. They introduce the term *opportunistic friendliness* with a similar def intion as used by Widmer:

"Opportunistic friendliness is the ability of a new flow to use the bandwidth that would be left unused by legacy flows."

This definition is not totally clear, but the article [47] describes it further; A new flow can use the bandwith that a legacy flow would have used under the same circumstances *and* all bandwith left unused by the existing flows. Note that in this definition, the word "TCP" is left out. Thus, it can also reffer to another flow type like TFRC. In [55], the authors write that they do not know of any definition that applies to both wired and wireless Internet, so they form their own, which in essence is equal to the two above:

"A protocol is TCP-friendly if it is friendly to TCP in wired networks (according to Floyd's definition), and can achieve better performance than TCP in wireless networks.

The definitions mentioned here are most concerned with the impact of flow A to flow B, or inter-protocol fairness. However, *intra*-protocol fairness is equally important. This is the property of staying fair to other flows of its same kind so that any new flow A_n is friendly to all existing flows A_1 to A_{n-1}.

The different definitions above, allthough they can seem equivalent at first glance, has lead to different conclusions from the researchers, possibly lowering the development speed of a common new standard. In an increasingly mobile Internet environment, the protocol performances in wireless access networks must be prioritized. Since the traditional versions of TCP do not perform optimally, it does not make sense to limit throughput to TCP. Therefore, the three latter definitions seem most apt for current and future work in this domain.

5.2 Network or client intelligence?

Two elementary strategies are considered in all forms of data network design; The placement of most or all network intelligence and data processing in the network core and intermediate nodes on one hand, and placement of all intelligence and processing in the end nodes on the other. The simulations presented in Chapter 3 present a topology where all transport and service level communication is end-to-end, and all calculations of jitter, delay, loss, transmission and quantization rates happen at either of the two end nodes. Thus, this layout with client-server service, IP and TFRC (or DCCP) is a typical client intelligent example.

The problems discussed in this study, is that the end-nodes, especially the server, are not capable of fine tuning their parameters to the exact network condition because they only know the packet's arrival status and timing. If elements of network intelligence where introduced, the intermediate nodes in all sorts of topologies could report back to the server about the link state in and out of the node. With such a system, the video server could easily know the exact reason for any packet drop or delay.

The P-AQM solution suggestion described in 2.5.3 shows one way to distribute the calculation and feedback "burden" over several nodes. This is done by letting P-AQM enabled routers employ smarter and more complex queuing schemes and individually send state feedback to the sender. With this system, the responsibility for congestion control and rate adaption is moved from the transport protocol to the server application and the routers. Thus, the use of legacy UDP and TCP side by side can be continued without jeopardizing the Internet health. TFRC and TCP have problems with loss differentiation that P-AQM avoids by sending feedback only if congestion is observed. Random error losses in wireless networks are simply ignored by the video bearer, UDP.

A suggestion to solve the loss differentiation problem of heterogeneous networks (wired and wireless) could be to use the APs to mark the returned traffic in both DCCP/TFRC and TCP flows with a signal strength indicator, thus giving the server better chances of estimating the loss causes. This solution requires cross-layered design, which breaks the network stack abstraction philosophy. Another use of the AP in a network intelligent fashion is to use Snoop, evaluated in [56] and [48]. This is a link layer approach with local retransmission where TCP (or TFRC with some improvements in Snoop) packets are recorded temporarily, and their ACK packets are checked to see if the data was lost in the wireless link or received properly. It retransmits all packets that are lost in the last hop and suppresses the ACK, thus sparing the serving TCP agent for the loss discrimination job and unnecessary rate adjustments. A more client intelligent approach is to let the TFRC client mark the ACKs with the current signal strength. When a loss occurs, the TFRC server can look at the last received signal strength value to determine whether the loss was due to congestion or link error. Alas, this solution is also dependent on a cross-layered design.

If the purpose is to make a specially crafted proprietary network not intended for Internet connection, the list of improvements in all layers and protocols could be long, and the result could probably be significantly improved with the technology and knowledge available today. However, because any new Internet standard today must meet requirements for a certain backward compatibility, interoperability, scalability, industry support and roll-out cost, many suggestions such as the those mentioned above can be rejected. A typical problem with solutions such as P-AQM is that it is not very effective in practice without a degree of penetration in the Internet routers worldwide. This constitutes a considerable cost and requires full industry support, and is one of the reasons that client intelligent and end-to-end protocols are more emphasized in Internet literature and research [47]. The following part of this section covers a selection of end-to-end solutions.

[48] evaluate different solution suggestions to wireless streaming performance. Many suggestions are made that benefit from the use of end-to-end packet transfer statistics. One suggestion interprets the variations in one-way delay to differentiate congestion loss from wireless loss, another uses the variation in RTT to differentiate, while a third suggestion combines packet inter arrival times with one-way delay to decide whether the loss is due to congestion or not. The key assumption is that the delay variations are bigger if there is contention in the network, while a report of a packet that has been lost due to an error should not experience much added delay. While these solutions keep the costly updates of network equipment unnecessary and avoid using cross-layered design solutions, they have a rather high *misclassification* rate that result in under-utilization of the wireless bandwidth [48].

Voelker et. al. also study end-to-end schemes for loss differentiation in [49]. In this article, the so-called Loss Discrimination Algorithms (LDAs) are studied in detail. It reveals that most LDAs based on inter-arrival times do not perform well when there is competition for the wireless link, i.e. both congestion and possible wireless loss. The simulation scenarios and test-beds used in the development of these LDAs are too simple to prove that the solutions can survive in a real-life network. For example, many LDAs and other solution suggestions assume that the wireless link is in the last hop. These methods fails if tested in a topology where a wireless link is in between two wired networks. Voelker et. al. extend TFRC to support different LDAs, compare the LDA performance in terms of effectiveness, smoothness and fairness, and suggest two new LDAs. The LDAs they test are the Biaz and Spike schemes, and the proposals they make are called ZigZag scheme and Hybrid LDA. The article concludes that *no single LDA can provide good performance with all kinds of network topologies and concurrent traffic types*. Their own invention the Hybrid LDA however proves good performance in all scenarios tested by selecting the best “basis”-LDA at any given time.

5.3 Multiple streams

In the simulations performed here, the performance issues of TFRC have been illustrated through two cases. However, the results given do not directly prove that TFRC under-utilize the wireless network bandwidth due to the system vulnerability found in section 4.1 and the combination of relatively high bandwidth video (1.2 Mbps) and high bandwidth wireless network (theoretically 11 Mbps, but about 5 Mbps in practice). In [48] on the other hand, the video source bandwidth

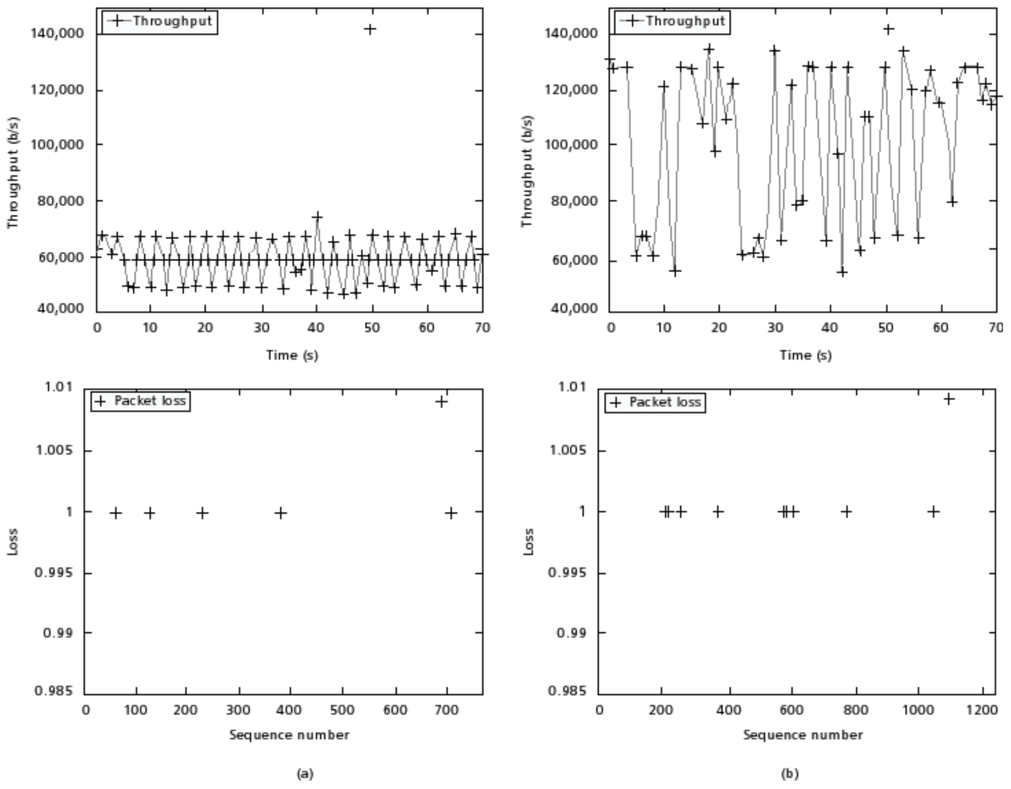


Figure 5.1 Throughput and packet loss details for (a) one TFRC, and (b) MULTFRC [48].

is large compared to the capacity and error rate of the CDMA wireless network, so a proof of network under-utilization with TFRC is possible and performed. In this article, Zakhor et. al. present a solution suggestion that is simple but effective. The approach is to use several TFRC streams in parallel to transport one single video stream. With this solution, they can show improved performance and link-utilization for a video streaming application without introducing any further network intelligence or protocol alteration. Their suggestion, MULTFRC, is simply a application layer system that dynamically adds or drops normal TFRC streams based on RTT feedback read from the TFRC server agent. The RTT feedback is used in a connection controller inspired by the traffic statistics loss differentiation methods described in section 5.2. Thus, it can add TFRC connections until the connection controller discovers congestion. Since each TFRC stream can discover and adjust to this congestion on its own, the added congestion check from MULTFRC is made to avoid overhead and starvation of other TCP or TFRC flows to and from the connected server and client. The results from their study is illustrated in figure 5.1.

Allthough simple and effective, the MULTFRC approach can not be claimed to be very elegant, it introduces overhead in resource utilization on both interacting nodes (especially computational overhead in video segmentation and reassembly, which can be critical for power scarce mobile equipment) and shows an unstable and bursty throughput (ref. figure 5.1(b)).

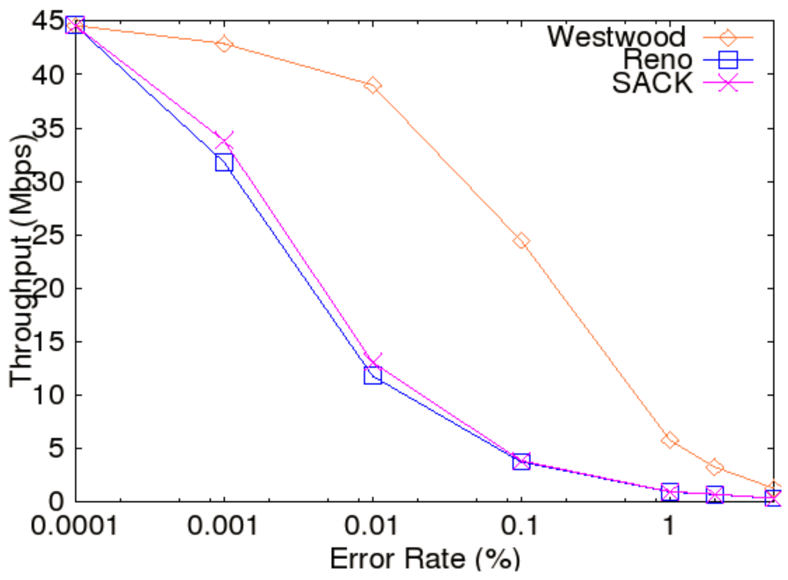


Figure 5.2 Throughput rates for TCP Westwood compared to TCP Reno and SACK, showing how the former handles random errors in a more efficient manner [57]

5.4 Designing from scratch

Many of the solution suggestions relevant to TFRC video streaming are from the research done to improve TCP performance in wireless links. Many of these discuss different approaches to loss differentiation (or loss discrimination) that will enable the TCP agents to understand the cause of the packet losses observed. Since the TFRC performance problems have coherence or similarity with those of TCP, solutions for TCP are often applicable for TFRC [48].

Sanadidi and Gerla, the same researchers that defined opportunistic friendliness, have done considerable work in the development of the new TCP version TCP Westwood. This version is aimed to give better throughput for ”large leaky pipes”, i.e. links with big bandwidth delay product and a limited amount of random error losses [17]. It is a sender side improvement to TCP NewReno, developed due to the increasing penetration of wireless access networks, powerful mobile clients and demanding ubiquitous services. The performace of TCP Westwood in a lossy environment with random packet loss, a test very similar to the one performed with TFRC in [13], is shown in figure 5.2.

The concepts and ideas behind this work could be a direct guideline to a refinement of TFRC for better wireless support. Sanadidi and Gerla, however, have developed a new transport protocol for adaptive video streaming from scratch. In [47], their solution suggestion Video Transport Protocol (VTP) is introduced. Two mechanisms are essential for VTP: Achieved Rate (AR) and Loss Discrimination Algorithm (LDA). AR is the rate that the receiver can measure and is used by the sender agent’s rate adaption mechanism. The LDA described here is the Spike scheme evaluated in [48] and referred to in 5.2. This scheme is based on RTT measures and the simple assumption that if a packet loss occurs when the RTT is close to the minimum

RTT observed, then there is *no* congestion and the loss must be due to the erroneous link. This assumes that the wireless link is also the bottleneck of the connection, which is correct in most cases. However, with today's fastest 802.11n equipment installed at a customer with a cheap ADSL line, the bottleneck will clearly not be in the wireless link. The rate controller is crafted to imitate TCP's additive increase until congestion is detected. The multiplicative decrease of TCP is omitted to ensure smooth rate variation at the video consumer. Instead of the drastic and short rate change of TCP, VTP will reduce the sender rate with small steps and keep the changes longer before increasing again, thus ensuring TCP-similar throughput in average.

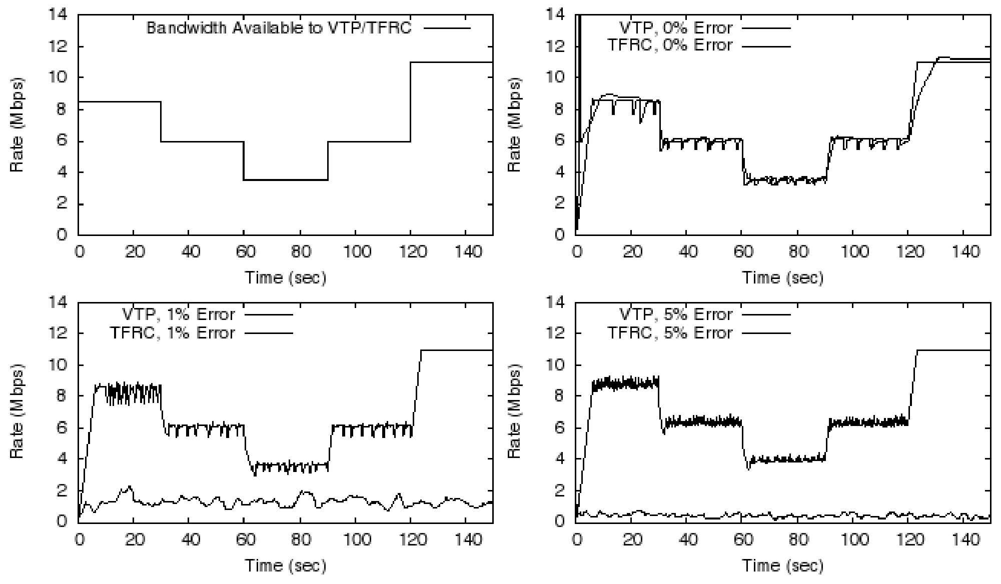


Figure 5.3 These graphs compare VTP and TFRC with different amount of random link loss [47].

The VTP protocol was tested in a similar manner to the tests performed and described in this thesis, only with a less realistic and more general wireless interface implementation. Figure 5.3 shows how the protocol outperforms TFRC in a lossy environment. Figure 5.4 illustrates the idea of opportunistic friendliness with VTP and TCP flows at different error rates. Note how the TCP flows running with VTP (left) and TCP (right) has the same throughput while VTP is allowed to enjoy the rest of the link capacity.

5.5 Application and cross-layer solutions

As stated in the introduction section 1.2.1, the service and application levels are regarded out of the scope for this work. However, several solution suggestions span both application, transport and MAC layers by different forms of cross-layer designs, and deserve being mentioned.

One of them is a new suggestion (2007) called RTP-Primal. This is a strategy that guarantees flow smoothness and congestion control for UDP flows based on application level adjustments of RTP and RTCP [58].

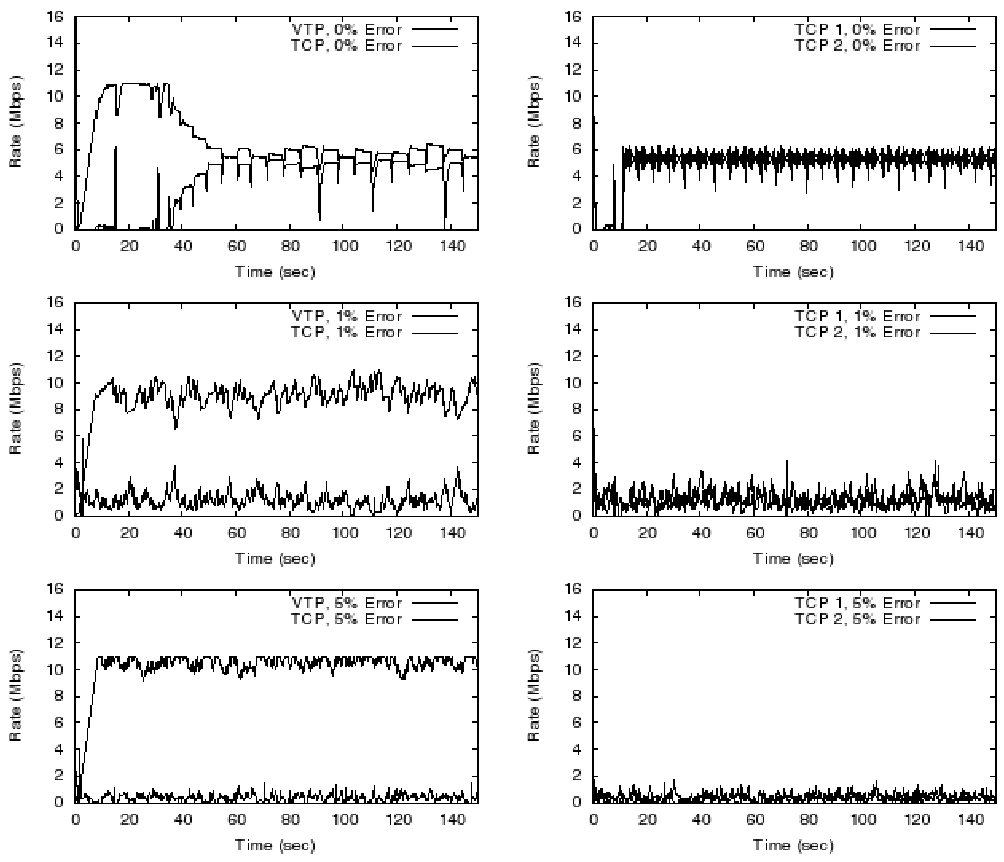


Figure 5.4 One VTP and one TCP flow to the left, and two TCP flows to the right. The graphs illustrate how the TCP flows have equivalent throughput, thus proving VTP opportunistic friendliness [47].

Another suggestion is a novel combination of the error resilience tools in H.264 and the possibilities of the new 802.11e in a cross-layer architecture [59]. The idea with this architecture can be briefly described as follows: H.264 has a function called data partitioning, which means that it separates data of different importance in different partitions (ref. discussion about packet importance in section 4.2.2). The QoS features of 802.11e allow for separated prioritized queues called access categories (described in section 2.6.5). Lower loss rate and delay, and higher overall quality can be achieved if the most important video partitions of H.264 can be enqueued in the most prioritized access category of 802.11e, as shown in [58].

A third solution suggestion is the Wireless Multimedia Streaming TCP-Friendly Protocol (WMSTFP) [55]. This solution emphasizes both network adaption and media adaption, and include mechanisms in application, transport and link layers. In the application layer, it proposes to use a special video codec with layered video and improved dynamic Forward Error Correction. It benefits from link-layer information from the wireless last hop to differentiate loss types, which makes it more robust to lossy wireless link than the reference protocol TFRC.

The above mentioned suggestions are only a few of a long list of suggestions. A tendency is that the most effective solutions are the ones who require most alteration in several layers of the protocol stack, an unfortunate trade-off if the Internet layer abstraction should be maintained.

5.6 Summary

As this chapter is meant to illustrate, there are numerous articles and suggestions in the problem domain emphasized in this work. Although DCCP with TFRC has become the most popular successor for UDP, considerable work is done and under progress to find a solution that fulfills the requirements of DCCP stated in [24].

Many of the suggestions are more specific and for narrower use than DCCP is intended, thereby increasing the effectiveness for one given task but failing to perform for many others. A general problem with the suggestions mentioned here is that the amount of information and work in the domain is so big, that no single proposal manages to cover all the aspects of this complex challenge. Where a suggestion is excellent with one type of metric and requirement, it fails in others. Most are tested with one single and oversimplified topology, and are not close to imitating the Internet.

The best example of redundant research and misunderstandings due to the "information overload" is the definition problem explained in 5.1. It is remarkable that researchers within the very same domain can work and publish without any knowledge or reference to each other.

The discussion concludes that many cross-layer approaches and intelligent networks show the best performance. P-AQM, which is explained in greater detail than the others (section 2.5.3), can perform good because it can be used to adapt streams only according to congestion while the error-losses are ignored. However, cross-layered design breaks with layer abstraction principles, and network intelligence can be very expensive. An one-layer end-to-end approach proposal is to use a modified TFRC that differentiates losses based on the ECN mark, but since ECN requires router support, it is not sufficient only to change the software at the network edges.

A common one-layer QoS upgrade is preferred. The 802.11e can be a big step in the right direction, as pointed out by [42], [43],[60], [61] and [62]. With this new protocol amendment, video streams over the most used office and home WLANs can be improved. But still, the streaming problem persists for other types of wireless networks such as CDMA, WiMax and older WLAN standards.

CHAPTER 6

Conclusion

This thesis has aimed to enlighten some of the problems involved in the combination of real-time services, existing transport protocols and wireless networks. To achieve this, a simulation topology has been set up to facilitate support for different relevant scenarios, transport types, link states and services. This has been a tool to illustrate some of the features and problems with today's protocols in comparison with the IETF suggestion for a new real-time rate controller, TFRC.

Furthermore, the thesis has aimed to clarify some of the issues in the standard development process. This has been done by pointing at the vast variety of different protocol proposals with overlapping or contradictory strategies.

One simulator implementation with one specific set-up is seldom enough to provide definitive conclusions. This also applies to the work that has been presented here. The thesis ends with a summary of suggested future work to improve the empirical quality of the simulations and to provide more certain conclusions.

6.1 The simulations

Through three different cases, the Evalvid-RA toolset has been used to imitate real video streaming by the use of tracefiles and a specially crafted packet reassembly and video reconstruction mechanism. With this toolset and the network models of *ns*-2, a powerful and complex framework has been available to perform realistic simulations of rate adaptive video-on-demand streaming services. The results have been presented in terms of throughput to the end users and PSNR of received video as compared to the uncompressed original. The framework has also allowed for manual inspection of the videos in order to compare actual perceived quality.

The simulation cases have shown how the "brute force" mode of operation of UDP traffic threatens the health of other data flows both of its own and different types. We have seen how

TFRC, on the other hand, scales well to adopt to other TFRC and TCP flows, but fails to eliminate packet loss in the wireless link. The first packet transmissions of a TFRC session have proved to be critical for the flow quality when we have real-time buffer constraints. The design and objective of TFRC makes it vulnerable to packet loss that is not due to congestion and buffer overflow, but a result of varying link quality and random errors. Also, the connection between Evalvid-RA acting as application layer and TFRC acting as transport layer has an implementational flaw or feature that distorts the test results *if* the first packet or its acknowledgment is lost.

Further, the simulations have illustrated the degree of delay introduced by a IEEE 802.11b wireless network in the last hop. While most modern wired access and transport networks have hierarchical topologies and dedicated channels between the nodes, the wireless network has flat topology and shared medium. Thus, it must spend considerable amounts of overhead time to avoid transmission collisions and maintain fairness between the nodes. The standardized mechanism to solve the hidden station problem, RTS/CTS, is often left out of 802.11 access point implementations. The simulations show why this can be a good idea, not only with the implementation cost in mind: The difference in added delay with or without RTS/CTS is not negligible, and can constitute the decisive factor that makes QoS possible or not for strict real-time services.

6.2 The development

The problem domain discussed in this thesis is complex since it involves considerations spanning from market demand and service in one end to radio frequency bands and physical channel encoding in the other. This complete picture is too comprehensive to be a part of the scope of this thesis, yet with the focus narrowed down to transport and MAC layers, the list of proposals and approaches is extensive. The work presented here reveals that the definition of one single term can result in very differing research results and contradictions.

The proposal examples discussed show that many approaches exist that can improve video streaming performance and QoS by adding more network intelligence or implementing a cross-layer design. However, while these suggestions perform optimally in a simulation environment, many would fail in the heterogeneous complexity of Internet. Moreover, many of the designs require that some or all nodes involved in the end-to-end path have upgrades in several layers, a requirement that breaks with the evolutionary principles and layer abstractions of the Internet protocol suite.

6.3 Yet to come

In the progress of the work presented here, several additional aspects and possible simulation improvements have been discovered. To limit the scope, and to go more in depth in some chosen topics, these have been postponed to future work.

The simulations here have been tested with a 30 seconds long QCIF video. To have more generic results and avoid stochastics, the simulations should be run with a rich library of videos of different resolution, length, content and encoding. The scenarios should be extended to include a higher degree of duplex traffic to imitate video conferencing, as this is especially interesting for the delay variations of the 802.11 MAC. A topology where the wireless network is *not* in the lost hop should also be tested to avoid the limitations found in many of the discussed solution suggestions.

Further, the new IEEE QoS amendment to the 802.11 family, 802.11e, should be included in the framework and compared to the results found with legacy 802.11b. The P-AQM solution should be tested on the wireless simulation topology presented here. Since P-AQM only delivers *congestion* feedback instead of *loss* feedback, it is highly relevant to test its performance in the problem domain defined.

A modification of TFRC to differentiate loss based on ECN flags would be very interesting to see if an updated version of the IETF proposal can satisfy *all* performance requirements in the new generation mobile Internet. Since the Evalvid-RA toolset as used here drops an entire video frame if one of the belonging packets is lost, the rebuilt video does not entirely represent the amount and quality of data that has actually been received. Additionally, the vulnerability of TFRC regarding the first packet arrival and acknowledgment can be caused by the interaction between TFRC and Evalvid-RA. A more precise implementation of the simulator tool-set could deliver more exact answers through the use of PSNR measurements.

Finally, this thesis has been strictly technical, and most business and market related topics relevant to realize a technology have been out of the scope. A study of the market potential, time to market and industry support for each of the proposals is vital to understand which strategy that eventually reaches the end-user.

Bibliography

[1] Kevin Fall and Kannan Varadhan (Editor). *The ns Manual*. A Collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC, 2007. The VINT Project.

[2] David Cavin, Yoav Sasson, and Andre Schiper. On the Accuracy of MANET Simulators. *Proc. of POMC'02*, 2002.

[3] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking (TON)*, 7(4):458–472, 1999.

[4] T. V. Lakshman, P. P. Mishra, and K. K. Ramakrishnan. Transporting compressed video over ATM networks with explicit rate feedback control. *INFOCOM'97: Proceedings of the INFOCOM'97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies.*, 1997.

[5] R. Rejaie, M. Handley, and D. Estrin. RAP: End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet. *Proc. of IEEE International Conference on Communications (ICC)*, 2001.

[6] Masaki Miyabayashi, Naoki Wakamiya, Masayuki Murata, and Hideo Miyahara. MPEG-TFRCP: Video Transfer with TCP-friendly Rate Control Protocol. 2001.

[7] D. Sisalem and A. Wolisz. LDA+ TCP-Friendly Adaption: A Measurement and Comparison Study. *Proc. of NOSSDAV*, 2000.

[8] Arne Lie, O. M. Aamo, and L. A. Rønningen. A Performace Comparison Study of DCCP and a Method with non-binary Congestion Metrics for Streaming Media Rate Control. *Proc of 19th International Teletraffic Congress (ITC'19)*, 2005.

[9] m. Handley, S. Floyd, J. Widmer, and J. Padhye. RFC3448: TCP-Friendly Rate Control (TFRC): Protocol Sepcification. 2003. `http://www.ietf.org/rfc/rfc3448.txt`, link last active: May 2007.

[10] Ku-Lan Kao, Chih-Heng Ke, and Ce-Kuen Shieh. An Advanced Simulation Tool-set for Video Transmission Performance Evaluation. 2006.

[11] Jirka Klaue, Berthold Rathke, and Adam Wolisz. EvalVid - A Framework for Video Transmission and Quality Evaluation. *Proc of 13th International Teletraffic Congress (ITC'13)*, 2003.

[12] Arne Lie and Jirka Klaue. Evalvid-RA: Trace Driven Simulation of Rate Adaptive MPEG-4 VBR Video. Not yet published.

[13] Torgeir Haukaas. Next Generation Internet and Support of Streaming Media: Trace Driven Simulation. *Available at* `http://folk.ntnu.no/torgeiha/div/ra-vbr_video_over_wifi.pdf` *(link active: May 2007)*, 2006.

[14] M. Bottigliengo, C. Casetti, C.-F. Chiasserini, and M. Meo. Short-term Fairness for TCP-flows in 802.11b WLANs. *IEEE INFOCOM*, 2004.

[15] M. Franceschinis, M. Mellia, M. Meo, and M. Munafo. Measuring TCP over WiFi: A Real Case. *1st workshop on Wireless Network Measurements (Winmee), Riva Del Garda, Italy*, 2005.

[16] D. Barman and I. Matta. Effectiveness of loss labeling in improving TCP performance in wired/wireless networks. *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on*, pages 2–11, 2002.

[17] Ren Wang, Massimo Valla ans Bryan Kwok Fai Ng, Mario Gerla, and M.Y. Sanadidi. Efficiency/Friendliness Tradeoffs in TCP Westwood. *Proc. of the 7th IEEE Symposium on Computers and Communications*, 2002.

[18] John Murphy. Resource allocation in ATM networks (PhD report). 1996. `http://www.eeng.dcu.ie/~murphyj/the/the/the.html`, link last active: May 2007.

[19] Rob Koenen. *MPEG-4 Overview*. International Organisation for Standardisation(ISO), 2002.

[20] Gonzales and Woods. *Digital Image Processing*. Prentice Hall, 2nd edition, 2002.

[21] Wikipedia. Peak Signal-to-Noise Ratio. `http://en.wikipedia.org/wiki/PSNR`, link last active: May 2007.

[22] VINT. *Virtual InterNetwork Testbed*. A Collaboration among USC/ISI, Xerox PARC, LBNL, and UCB, 1996. `http://www.isi.edu/nsnam/vint/`, link last active: May 2007.

[23] FFmpeg homepage. `http://ffmpeg.mplayerhq.hu/`, link last active: May 2007.

[24] S. Floyd, E. Kohler, and J. Padhye. RFC4336: Problem Statement for the Datagram Congestion Control Protocol (DCCP). 2006. `http://www.ietf.org/rfc/rfc4336.txt`, link last active: May 2007.

[25] S. Floyd, E. Kohler, and J. Padhye. RFC4340: Datagram Congestion Control Protocol (DCCP). 2006. `http://www.ietf.org/rfc/rfc4340.txt`, link last active: May 2007.

[26] J. Padhye, J. Kurose, D. Towsley, and R. Koodli. A model based TCP-friendly rate control protocol. *UMASS-CMPSCI Technical Report*, 1998.

[27] S. Floyd, E. Kohler, and J. Padhye. RFC4342: Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC). 2006. `http://www.ietf.org/rfc/rfc4342.txt`, link last active: May 2007.

[28] S. Floyd and V. Jacobson. Random early detection for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[29] K. Ramakrishnan, S. Floyd, and D. Black. RFC3168: The Addition of Explicit Congestion Notification (ECN) to IP. *Internet RFCs*, 2001.

[30] Arne Lie. P-AQM: low delay max-min fairness streaming of scalable real-time CBR and VBR media. 2007.

[31] Arne Lie. A Proposal for Low Delay Max-min Fairness Control of Streaming Media. 2006.

[32] Prasad and Ramjee. *WLANs and WPANs towards 4G Wireless*. Artech House Inc., 2003.

[33] Matthew Gast. *802.11 Wireless Networks: The Definitive Guide*. O'Reilley, 2nd edition, 2005.

[34] IEEE Working Group 802.11. IEEE 802.11 official timelines. *Link accessed and active: 2. March 2007*. `http://grouper.ieee.org/groups/802/11/Reports/802.11_Timelines.htm`, link last active: May 2007.

[35] T. Ulseth and P. Engelstad. Voice over WLAN (VoWLAN)–A wireless voice alternative. *Telektronikk 1.06*, 2006.

[36] ANSI and IEEE. *Std 802.11 - Medium Access Control (MAC) and Physical (PHY) Specifications*. ANSI / IEEE, 1999.

[37] I. Haratcherev, J. Taal, K. Langendoen, R. Lagendijk, and H. Sips. Automatic IEEE 802.11 rate control for streaming applications. *Wireless Communications and Mobile Computing*, 5(4):421–437, 2005.

[38] C. Steger, P. Radosavljevic, and JP Frantz. Performance of IEEE 802.11 b wireless LAN in an emulated mobile channel. *Vehicular Technology Conference, 2003. VTC 2003-Spring. The 57th IEEE Semiannual*, 2, 2003.

[39] A. Kamerman and N. Erkocevic. Microwave oven interference on wireless LANs operating in the 2.4 GHz ISM band. *Personal, Indoor and Mobile Radio Communications, 1997.'Waves of the Year 2000'. PIMRC'97., The 8th IEEE International Symposium on*, 3, 1997.

[40] N. Golmie, RE Van Dyck, and A. Soltanian. Interference of bluetooth and IEEE 802.11: simulation modeling and performance evaluation. *Proceedings of the 4th ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 11–18, 2001.

[41] RJ Punnoose, RS Tseng, and DD Stancil. Experimental results for interference between Bluetooth and IEEE 802.11 b DSSS systems. *Vehicular Technology Conference, 2001. VTC 2001 Fall. IEEE VTS 54th*, 1, 2001.

[42] Q. Ni. Performance analysis and enhancements for IEEE 802.11 e wireless networks. *Network, IEEE*, 19(4):21–27, 2005.

[43] Mehmet U. Demircin and Peter van Beek. Bandwidth Estimation and Robust Video Streaming over 802.11e Wireless LANs. 2006.

[44] Joshua Robinson. Making NS-2 simulate an 802.11b link. *`http://www.ece.rice.edu/~jpr/ns/docs/ns-802_11b.html`, link last active: May 2007.*

[45] Wu Xiuchao. Simulate 802.11b Channel within NS2.

[46] M. Lacage, M.H. Manshaei, and T. Turletti. IEEE 802.11 rate adaptation: a practical approach. *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 126–134, 2004.

[47] Guang Yang, Mario Gerla, and M.Y. Sanadidi. Adaptive Video Streaming in Presence of Wireless Errors. *IFIP/IEEE MMNS 2004*, 2004.

[48] Minghua Chen and Avideh Zakhor. Rate Control for Streaming Video over Wirless . 2004.

[49] Song Cen, Pamela C. Cosman, and Geoffrey M. Voelker. End-to-End Differentiation of Congestion and Wireless Losses. 2003.

[50] D.M. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17(1):1–14, 1989.

[51] M. Borri, M. Casoni, and M.L. Merani. Performance and TCP-fairness of TFRC in an 802.11g WLAN: Experiments and Tuning. *Wireless Communication Systems, 2004. 1st International Symposium*, 2004.

[52] J. Widmer, R. Denda, and M. Mauve. A survey on TCP-friendly congestion control. *Network, IEEE*, 15(3):28–37, 2001.

[53] M. Borri, M. Casoni, and M.L. Merani. An Experimental Study on Congestion Control in Wireless and Wired Networks. *Proc. of IEEE ICC 2005, May 2005*, 2005.

[54] M. Borri, M. Casoni, and M.L. Merani. Effectiveness and Issues of Congestion Control in 802.11g wireless LANs. *Journal of Wireless Networks, Springer*, 2006.

[55] F. Yang, Q. Zhang, W. Zhu, and Y.Q. Zhang. End-to-end TCP-Friendly Streaming Protocol and Bit Allocation for Scalable Video Over Wireless Internet. *IEEE Journal on selected areas in communications, vol. 22, no. 4, May 2004*, 2001.

[56] H. Balakrishnan, VN Padmanabhan, S. Seshan, and RH Katz. A comparison of mechanisms for improving TCP performance overwireless links. *Networking, IEEE/ACM Transactions on*, 5(6):756–769, 1997.

[57] Ren Wang, Massimo Valla, Mario Gerla, and M.Y. Sanadidi. TCP Westwood: Efficient Transport for High-speed Wired/Wireless Networks. *Presentation in High Speed Workshop, IEEE Infocom*, 2002.

[58] M. Borri, M. Casoni, and M.L. Merani. RTP-Primal: A New Congestion Control Scheme Suitable for Smooth Multimedia Delivery. *Proc. of IEEE Wireless Communications and Networking Conference, IEEE WCNC 2007*, 2007.

[59] Adlen Ksentini, Mohamed Nimi, and Abdelhak Guerouni. Toward an Improvement of H.264 Video Transmission over IEEE 802.11e through a Cross-Layer Architecture. *IEEE Communications Magazine, January 2006*, 2006.

[60] D.J. Leith, P. Clifford, D. Malone, and A. Ng. TCP Fairness in 802.11e WLANs. 2005.

[61] Y. Xiao and H. Li. Evaluation of distributed admission control for the IEEE 802.11 e EDCA. *Communications Magazine, IEEE*, 42(9):S20–S24, 2004.

[62] A. Banchs, A. Azcorra, C. Garcia, and R. Cuevas. Applications and challenges of the 802.11 e EDCA mechanism: an experimental study. *Network, IEEE*, 19(4):52–58, 2005.

APPENDIX A

List of abbreviations

AAC Advanced Audio Coding

AC Access Category

AIMD Additive Increase / Multiplicative Decrease

ANSI American National Standards Institute

AP Access Point

API Application Program Interface

AQM Active Queue Management

AR Achieved Rate

ARF Auto Rate Fallback

ASP Advanced Simple Profile

AVC Advanced Video Coding

AWK Aho Weinberger Kernighan (script language)

BSS Basic Service Set

CCID Congestion Control ID

CCK Complementary Code Keying

CDMA Code Division Multiple Access

CE Congestion Experienced

CFP Contention Free Period

CIF Commom Intermediate Format

CP Contention Period

CSI Channel State Information

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance

CTS Clear To Send

DCCP Datagram Congestion Control Protocol

DCF Distributed Coordination Function

DCT Discrete Cosine Transform

DNS Domain Name System

DSSS Direct Sequence Spread Spectrum

ECF Explicit Congestion Feedback

ECN Explicit Congestion Notification

ECT ECN-Capable Transport

EDCA Enhanced Distributed Channel Access

ERF Explicit Rate Feedback

ESS	Extended Service Set
FTP	File Transfer Protocol
GOP	Group Of Picture
HCCF	HCF Controlled Channel Access
HCF	Hybrid Coordination Function
ICMP	Internet Control Message Protocol
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IETF	The Internet Engineering Task Force
IFS	InterFrame Space
IO	Input / Output
IP	Internet Protocol
ISM	Industry, Science and Medical applications (radio band)
ISO	International Organization for Standardization
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector
JPEG	Joint Photographic Experts Group
LDA	Loss Discrimination Algorithm
LLC	Logical Link Control
MAC	Media Access Control
MPDU	MAC Protocol Data Unit
MSE	Mean Square Error
MTU	Maximum Transfer Unit
MOS	Mean Opinion Score
MPEG	Motion Picture Experts Group
nam	Network animator
NAT	Network Adress Translation
NAV	Network Allocation Vector
ns	Network simulator
OFDM	Orthogonal Frequency Division Multiplexing
PAL	Phase-Alternating Line
PCF	Point Coordination Function
PDA	Personal Digital Assistant
PoB	Play out Buffer
PSNR	Peak Signal-to-Noise Ratio (in dB)
P-AQM+ECF	Propotional Active Queue Management with Explicit Congestion Feedback
QAM	Quadrature Amplitude Modulation
QCIF	Quarter CIF
QoS	Quality of Service
RA	Rate Adaptive
RED	Random Early Detection
RGB	Red, Green, Blue
RFC	Request For Comments
RTCP	RTP Control Protocol
RTP	Real Time Protocol
RTS	Request To Send
RTSP	Real Time Streaming Protocol
RTT	Round-trip-time
SIFS	Short Interframe Space
SNMP	Simple Network Management Protocol
STA	802.11 Wireless Station
SVBR	Shaped Variable Bit Rate
Tcl	Tool Command Language
TCP	Transmission Control Protocol
TFRC	TCP-Friendly Rate Control

TOS	Type Of Service
TXOP	Transmission Opportunity
UDP	User Datagram Protocol
VBR	Variable Bit Rate
VLC	VideoLAN Client
VoD	Video-on-Demand
VTP	Video Transport Protocol
WEP	Wired-Equivalent Privacy
WLAN	Wireless Local Area Network
WME	Wireless Multimedia Extension
WMM	Wi-Fi Multimedia
WMSTFP	Wireless Multimedia Streaming TCP-Friendly Protocol
WMV	Windows Media Video
WPA	Wi-Fi Protected Access
YUV	Luminance and chrominance color space

APPENDIX B

Shell scripts

Code Snippet 2 , manQ.sh: This script is used to run the simulation pre-process.

```
#!/bin/bash
#############################################################################
# Script to execute ffmpeg and generate many versions of VBR open loop video.
# Input arguments:
# $1: Quantizer minimum.
# $2: Quantizer maximum
# $3: input file name (the video file that ffmpeg knows how to decode, i.e. trascode)
#
# Output is a *.m4v file (mpeg4 video bit stream) file
#
# Then this file is stripped for eventially repeated headers using
#           mp4 -strip
# Then the send trace is generated with
#           mp4 -send
# from Evalvid.
#
# Last edited by Torgeir Haukaas - 2006
############################################################################

######################################
# ffmpeg parameters:
# (Note that parameters can reflect input and output files. The file the switch belongs
# to is the first file mentioned right of the switch.)
# -r is fps, and is set twice: forces 30fps of input, and 30fps of output
# -s is set to CIF
# -vcodec is set to mpeg4 (format of output)
# -qscale is fixed quantizer (==> VBR open loop)
# -g is GOP size in numb. of frames. Note that ffmpeg will modify this to
#    shorter GOPs when needed.
# -y forces output file overwrite in case it exists.
#####
# mp4 parameters:
# -strip searches input file for repeated headers and removes them for output
# -send <stream dest. IP address> <port numb> <fps> <source file> > <stream data>
# Note: I have speeded up this function by indicating very high frame rate!! The time
# schedule will nevertheless be overruled by the ns-2 program.
######################################
for ((  i=$1 ;  i<=$2;  i++  ))
do
        ffmpeg -r 30 -s 176x144 -i $3 -vcodec mpeg4 -g 12 -sc_threshold 20000 -qscale
            $i -s 176x144 -r 30 -y videoVariants/$3_Q$i.m4v
        mp4 -send 224.168.1.100 1234 1000 30000 videoVariants/$3_Q$i.m4v > nsin/
            st_$3_Q$i.txt
done

exit 0
```

Code Snippet 3 , case1.sh

```
#!/bin/bash
# Torgeir Haukaas - NTNU 2006
# This script executes i iterations of a ns-simulation to make
# a comparison of video qualities whith different link congestion.
##############################################################################
#
type=udp
TYPE=TFRCs
UDP=1
TFRC=8
FTP=0
WEB=0
VIDEO=3xCoastguard_qcif
POB=250
folder=/home/torgeir/devel/master/case1
#i=$((FTP+UDP+WEB+TFRC-1))
#rm -f $folder/psnr-all.$type.txt
for (( i=0 ; i<=$TFRC; i++ ));
do
        echo "---->NEW ITERATION: -nofUDP $UDP -nofTFRC $i -nofFTP $FTP -nofWEB $WEB
            <-------"
        mkdir $folder/nsout.$i
        ns 802.11.tcl -nofUDP $UDP -nofTFRC $i -nofFTP $FTP -nofWEB $WEB -doPing 1 -
            seed 8 > $folder/nsout.$i/std_out.$type.iteration$i.txt 2>&1
        echo "ns finished"
        et_ra nsout/sd_be.$type.0 nsout/rd_be.$type.0 nsin/st_$VIDEO.yuv_Q2.txt nsout/
            video2.dat videoVariants/$VIDEO.yuv_Q 2 31 nsout/frame_size.dat $VIDEO.m4v
             01 $POB
        echo "et_ra finished, $VIDEO.m4v created "
        mp4creator -r 30 -c $VIDEO.m4v $folder/nsout.$i/$VIDEO.$i.mp4 > mp4creatorOut/
            out.$i.txt 2>&1
        echo "mp4creator finished, $VIDEO.$i.mp4 created"
        rm $VIDEO.m4v
        ffmpeg -i $folder/nsout.$i/$VIDEO.$i.mp4 -r 30 $folder/nsout.$i/$VIDEO.
            reconstructed.yuv > ffmpegOut/out.$i.txt 2>&1
        echo "ffmpeg finished, $VIDEO.reconstructed.yuv created "
        echo ""
        fixyuv nsout/sd_be.$type.0 nsout/rd_be.$type.0 new_st.txt $folder/nsout.$i/
            $VIDEO.reconstructed.yuv $folder/nsout.$i/$VIDEO.reconstructed.fixed.$i.
            yuv $POB > fixyuvOut/out.$i.txt 2>&1
        echo "fixyuv_ra finished "
        echo ""
        rm -f $folder/nsout.$i/$VIDEO.reconstructed.yuv
        psnr 176 144 420 $VIDEO.yuv $folder/nsout.$i/$VIDEO.reconstructed.fixed.$i.yuv
            > $folder/nsout.$i/psnr_$VIDEO.$type.it$i.txt 2>$folder/nsout.$i/
            psnr_result.$type.$i.txt
        echo "psnr finished "
        awk '{print '$i'" " $8}' $folder/nsout.$i/psnr_result.$type.$i.txt >>$folder/
            psnr_all.$type.txt

        cp nsout/rd_be.$type.0 $folder/nsout.$i/rd_be.$type.0
        cp nsout/sd_be.$type.0 $folder/nsout.$i/sd_be.$type.0
        cp nsout/802.11.tr $folder/nsout.$i/802.11.tr
        cp nsout/bwt.* $folder/nsout.$i/
        ./xgr2.sh
        echo "Press Y to clean N to keep tmp-files:"
        read answer
        if (( (($answer == Y)) || (($answer == y)) )); then
                ./clean.sh
        else
                echo "keeping tmp-files"
        fi
done
cd ../case1/
xgraph psnr_all.$type.txt -nb -geometry 800x400 -bg white -t "Peak Signal-to-Noise
    Ratio" -y "PSNR (dB)" -x "number of $TYPE" &

exit 0
```

Code Snippet 4 , case2.sh

```
#!/bin/bash
# Torgeir Haukaas − NTNU 2006
# This script executes i iterations of a ns−simulation to make
# a comparison of video quality with different signal quality.
############################################################################
#
type=tfrc
UDP=0
TFRC=1
FTP=0
WEB=0
VIDEO=3xCoastguard_qcif
POB=250
folder=/home/torgeir/devel/master/case2/$type
dev=4
pathloss=2
dist=15
#i=$((FTP+UDP+WEB+TFRC−1))
#rm −f $folder/psnr−all.$type.txt
for ((  i=5 ;  i<=$dist;  i++  ));
do
        echo "---->NEW ITERATION: pathloss=$pathloss deviation=$dev nodeDist=$i -nofUDP
             $UDP -nofTFRC $TFRC <-------"
        mkdir $folder/nsout.$i
        ns 802.11.tcl −nofUDP $UDP −nofTFRC $TFRC −nofFTP $FTP −nofWEB $WEB −doPing 0 −
            pathlossExp $pathloss −std_db $dev −nodeDist $i > $folder/nsout.$i/std_out
            .$type.iteration$i.txt 2>&1
        echo "ns finished"
        et_ra nsout/sd_be.$type.0 nsout/rd_be.$type.0 nsin/st_$VIDEO.yuv_Q2.txt nsout/
            video2.dat videoVariants/$VIDEO.yuv_Q 2 31 nsout/frame_size.dat $VIDEO.m4v
             01 $POB
        echo "et_ra finished, $VIDEO.m4v created "
        mp4creator −r 30 −c $VIDEO.m4v $folder/nsout.$i/$VIDEO.$i.mp4 > mp4creatorOut/
            out.$i.txt 2>&1
        echo "mp4creator finished, $VIDEO.$i.mp4 created"
        rm $VIDEO.m4v
        ffmpeg −i $folder/nsout.$i/$VIDEO.$i.mp4 −r 30 $folder/nsout.$i/$VIDEO.
            reconstructed.yuv > ffmpegOut/out.$i.txt 2>&1
        echo "ffmpeg finished, $VIDEO.reconstructed.yuv created "
        echo ""
        fixyuv nsout/sd_be.$type.0 nsout/rd_be.$type.0 new_st.txt $folder/nsout.$i/
            $VIDEO.reconstructed.yuv $folder/nsout.$i/$VIDEO.reconstructed.fixed.$i.
            yuv $POB > fixyuvOut/out.$i.txt 2>&1
        echo "fixyuv_ra finished "
        echo ""
        rm −f $folder/nsout.$i/$VIDEO.reconstructed.yuv
        psnr 176 144 420 $VIDEO.yuv $folder/nsout.$i/$VIDEO.reconstructed.fixed.$i.yuv
            > $folder/nsout.$i/psnr_$VIDEO.$type.it$i.txt 2>$folder/nsout.$i/
            psnr_result.$type.$i.txt
        echo "psnr finished "
        awk '{if ($8=="") print '$i'" " 0; else print '$i'" " $8}' $folder/nsout.$i/
            psnr_result.$type.$i.txt >>$folder/psnr_all.$type.txt

        cp nsout/rd_be.$type.0 $folder/nsout.$i/rd_be.$type.0
        cp nsout/sd_be.$type.0 $folder/nsout.$i/sd_be.$type.0
        cp nsout/802.11.tr $folder/nsout.$i/802.11.tr
        cp nsout/bwt.∗ $folder/nsout.$i/
        ./xgr2.sh
done
cd $folder
xgraph psnr_all.$type.txt −nb −geometry 800x400 −bg white −t "Peak Signal-to-Noise
    Ratio" −y "PSNR (dB)" −x "Distance (m)" &

exit 0
```

Code Snippet 5 , case3.sh: This script runs i iterations of ns with $1,...,i$ UDP sources, collects transmit and reception timestamps from one monitored flow, calculates the delay average per flow and presents them in a graph.

```
#!/bin/bash
# Torgeir Haukaas - NTNU 2006
# This script executes i iterations of a ns-simulation to make
# a presentation of 802.11 link delay (in %) for different input rates.
############################################################################
#
type=udp
UDP=6
TFRC=0
FTP=0
WEB=0
VIDEO=3xCoastguard_qcif
POB=250
folder=/home/torgeir/devel/master/case4/many-udps

#i=$((FTP+UDP+WEB+TFRC-1))
#rm -f $folder/psnr-all.$type.txt
for (( i=1 ; i<=$UDP; i++ ));
do
        echo "---->NEW ITERATION: -nofUDP $i <-------"
        mkdir $folder/nsout.$i
        ns 802.11.tcl -nofUDP $i -doPing 1 -snet_prop 3ms -core_prop 9ms -bsnet_prop 3
            ms > $folder/nsout.$i/std_out.$type.iteration$i.txt 2>&1
        echo "ns finished"
        awk -f awk/added-delay.awk nsout/802.11.tr > nsout/one-udp-delay.$i.txt
        echo "added-delay.awk finished"
        awk -f awk/avg-delay.awk nofudp=$i nsout/one-udp-delay.$i.txt >> $folder/avg-
            delay.txt
        echo "avg-delay.awk finished"

        cp nsout/* $folder/nsout.$i/
        #./xgr2.sh

done
cd $folder
xgraph avg-delay.txt -nb -geometry 800x400 -bg white -t "802.11 link delay (share of
    total)" -y "Avg. Delay (%)" -x "UDP sources" &

exit 0
```

Code Snippet 6 , clean.sh: This shell-script removes all the trace files and temporary results/output from a simulation.

```
#!/bin/bash
echo -n "cleaning up the simulation mess!         "
echo -n "."
rm -f nsout/* > err 2>&1
echo -n "."
rm -f ffmpegOut/* > err 2>&1
echo -n "."
rm -f fixyuvOut/* > err 2>&1
echo -n "."
rm -f mp4creatorOut/* > err 2>&1
echo -n "."
rm -f psnr2mos/* > err 2>&1
echo -n "."
rm -f *reconstructed* > err 2>&1
echo -n "."
rm -f psnr_*.txt > err 2>&1
echo -n "."
rm -f coastguard*.mp4 > err 2>&1
echo -n "."
rm -f packets_lost.txt > err 2>&1
echo -n "."
rm -f result.txt > err 2>&1
echo -n "."
rm err
echo -n "."
echo " done."
```

APPENDIX C

AWK scripts

Code Snippet 7 , web-throughput.awk.

```
# Awk script to find the throuput to a given destination node.
# Used to monitor web-traffic throughput.
# Input:
#       dn = destination node
# Torgeir Haukaas -NTNU- 2006
# (Based on script by Lloyd Wood, University of Surrey)
####################################################################
BEGIN {
        highest_packet_id = 0;
        n = 1;
        st=0.5;
        web_flow_id=2;
}

/-Hs/ && /r/ {
        time = $3;
        node_1 = $5;
        node_2 = $7;
        bytes = $37;
        flow_number = $39;

        # The following while-loop ensures that empty time samples are taken
        # care of.
        while (time > st*n) {
                bps[n] = acc_byte[n]*8/(st*1000000);
                n++;
        }

        # Accumulate all received bytes at final destination for the flow of interest
        if (node_1 == dn && flow_number == web_flow_id)
                acc_byte[n] += bytes;
}

# End up with printing out the results
END {
        for ( t = 1; t <= n; t++ ) {
                printf("%f %f\n", t*st, bps[t]);
        }
}
```

Code Snippet 8 , added-delay.awk.

```
# This script monitors the flow between specified source and sink ,
# and calculates delay caused by the 802.11−link .
# Output is arrival time of packet [j] and the average delay of the last
# 10 packets .
# Torgeir Haukaas –NTNU– 2006
###################################################################
BEGIN {
        source=2;       # refers to ns node numbers
        ap=3;           # refers to ns node numbers
        sink=4;         # refers to ns node numbers
        limit=11000;    # limit calculation to the k first packets
        sample=40;      # step size for delay sampling (in nof pkts)
}
(/^\+/ && $3==source) || (/^r/ && $4==ap) || (/^r/ && $5==sink) {
        for(k=0;k<limit;k=k+sample) {
                if ( $1=="+" && $12==k ) starttime[k]=$2;
                if ( $1=="r" && $12==k ) aptime[k]=$2;
                if ( $1=="r" && $41==k ) stoptime[k]=$3;
        }

}

END {
#       for (idx in starttime) print idx, starttime[idx], aptime[idx], stoptime[idx];

        j=1;
        for(i=0;i<limit;i=i+20) {
                if(starttime[i]>0 && aptime[i]>0 && stoptime[i]>0){
                        total_delay=stoptime[i]−starttime[i];
                        wifi_delay=stoptime[i]−aptime[i];
                        share[j]=wifi_delay/total_delay;
                        if(j>5) {
                                running_avg=(share[j−4]+share[j−3]+share[j−2]+share[j
                                    −1]+share[j])/5;
                                print stoptime[i], running_avg;
                        } else if(j>10) {
                                running_avg=(share[j−9]+share[j−8]+share[j−7]+share[j
                                    −6]+share[j−5]+share[j−4]+share[j−3]+share[j−2]+
                                    share[j−1]+share[j])/10;
                                print stoptime[i], running_avg;
                        } else print stoptime[i], share[j];
                        j++;
                        #printf(”id %i start %f ap %f stop %f share %f\n”, i, starttime
                            [i], aptime[i], stoptime[i], share);
                }
        }
}
```

Code Snippet 9 , avg-delay.awk.

```
# this script calculates average delay cause by the 802.11-link.
# run added-delay.awk on a ns trace-file and
# apply this to the results.
# Input:
#        nofudp = number of concurrent flows in the monitored
#                 802.11 network.
# Torgeir Haukaas -NTNU- 2006
####################################################################
BEGIN {
        i=0;
}
{
        delay[i]=$2
        i++;
}

END {
        for(k=0;k<i;k++) sum=sum+delay[k];
        avg=sum/i;
        print nofudp, avg;
}
```

Code Snippet 10 , avg-quantizer.awk.

```
# this script calculates average quantizer level used by
# Evalvid-RA. Apply to a file of the type sd_be[x]
# Input:  none
# Torgeir Haukaas -NTNU- 2006
####################################################################
BEGIN {
        i=0;
}
{
        q[i]=$6
        i++;
}

END {
        for(k=0;k<i;k++) sum=sum+q[k];
        avg=sum/i;
        print avg;
}
```

APPENDIX D

Tcl scripts

Code Snippet 11 This is the main Tcl-script controlling the *ns*-2 simulation.

```
# This script sets up a simulation environment to test streaming media
# over 802.11 wireless interfaces with adjustable link quality.
# A dumbbell topology with wireless nodes on one end and media servers
# on the opposite end will be demonstrated.
######################################################################
# No rights reserved: Torgeir Haukaas 2007

# ====================================================================
# General Parameters
# ====================================================================
set opt(title)          "802.11"        ;# script-name
set opt(seed)           13.37           ;# general random seed for simulation traffic
    flow
set opt(stop)           55.0            ;# simulation time
set opt(verbose)        1               ;# 1 to print messages 0 to be silent
# ====================================================================
# Channel properties (wifi-link)
# ====================================================================
set opt(chan)           Channel/WirelessChannel
set opt(prop)           Propagation/TwoRayGround ;# FreeSpace, TwoRayGround or
    Shadowing
set opt(netif)          Phy/WirelessPhy
set opt(mac)            Mac/802_11
set opt(qm)             DropTail        ;# RED
set opt(ifq)            Queue/DropTail
set opt(ll)             LL
set opt(ant)            Antenna/OmniAntenna
set opt(adhocRouting)   DSDV            ;# possible ad-hoc routing protocols:
                                        # DSDV DSR TORA AODV
set opt(disableRTS_CTS) 1               ;# RTS / CTS is disabled by default (=1).
    ENABLE (=0) if mobile nodes
                                        # cannot see eachother!!
# For Propagation/Shadowing:
set opt(pathlossExp)    2.0             ;# path loss exponent
set opt(std_db)         4.0             ;# deviation (dB)
set opt(dist0)          1.0             ;# reference distance (m) (see ns manual p.
    189)
# ====================================================================
# Topology
# ====================================================================
set opt(x)              670             ;# X dimension of the topography
set opt(y)              670             ;# Y dimension of the topography

set opt(ifqlen)         50              ;# max packet in ifq
set opt(nofmn)          1               ;# number of wireless nodes OBS! max 9 for now
set opt(nofbs)          1               ;# number of base stations
set opt(nofsn)          1               ;# number of server nodes
set opt(mn_motion)      0               ;# random motion of mobile nodes
set opt(nodeDist)       2               ;# distance in meters between WLAN nodes
set opt(ecn)            0               ;# 1 to enable ECN in routers
set opt(adaptive)       true
set opt(minth)          0
set opt(maxth)          0
# ====================================================================
# Evalvid parameters
# ====================================================================
set opt(videoname)      "3xCoastguard_qcif"     ;# base of video file name
set opt(vbr_rate)       3Mb                     ;# max VBR-output
set opt(q_variants)     30                      ;# quantizer scale range
set opt(ra_mtu)         1000                    ;# max fragmented size, MTU
set opt(fps)            30                      ;# frames per second (all video is
    treated with equal fps)
set opt(tfrc_pktsize)   [expr $opt(ra_mtu) + 36] ;# DCCP/TFRC header (12+4=16B) and
    IP-header (20B)
set opt(udp_pktsize)    [expr $opt(ra_mtu) + 28] ;# UDP header (8B) and IP-header (20B)
set opt(cbr_true)       0
set opt(GoP_size)       12
# ======================================================================
# Traffic generation.
# ======================================================================
set opt(nofTCP)         0       ;# number of TCP flows
set opt(nofWEB)         0       ;# number of WEB flows
set opt(nofFTP)         0       ;# number of FTP flows
set opt(nofUDP)         0       ;# number of UDP flows
set opt(nofTFRC)        0       ;# number of TFRC flows
#
set opt(window)         30      ;# window for long-lived traffic
set opt(tcp_pktsize)    1460
set opt(tcpTick_)       0.01
set opt(doPing)         0       ;# 1 to ping servers on startup, 0 to disable
# WEB-settings
set opt(webSeed)        0       ;# seed for RNG for web traffic generation
set opt(interPage)      1       ;# average value for exponentially distributed inter
    page time
set opt(pageSize)       3       ;# nof objects per page
set opt(nofWebPages)    10      ;# number of pages per session
set opt(nofWebSessions) 4       ;# number of web sessions
set opt(webObjectSize)  20      ;# average size of web object in pkts.
set opt(paretoShape)    1.05    ;# shape parameter for Pareto distribution of web size
                                 # a larger parameter means more small objects
# ======================================================================
# Plotting statistics.
# ======================================================================
set opt(out_dir)        nsout
set opt(tf)             $opt(out_dir)/$opt(title).tr     ;# trace file
set opt(namtf)          $opt(out_dir)/$opt(title).nam    ;# nam trace file
set opt(doTrace)        1       ;# 1 to do tracing TODO: remove or fix: this value
    cannot be 0
set opt(nam)            1       ;# 1 to trace and run nam
set opt(sampleTime)     0.5     ;# time between each bw-sample written by the
    record-function
set opt(macTrace)       OFF
set opt(agentTrace)     ON
set opt(routerTrace)    OFF
set opt(movementTrace)  OFF
# ======================================================================
# Link characteristics.
# ======================================================================
set opt(errUnit)        packet  ; # unit of errors, per packet or per bit
set opt(errRateUL)      0       ; # the rate of errors in uplink
set opt(errBurstUL)     0       ; # coefficient 0..1 of how bursty errors are
set opt(errSlotUL)      0       ; # time in sec of low error period
set opt(errRateDL)      0       ; #
set opt(errBurstDL)     0       ; #
set opt(errSlotDL)      0
# ======================================================================
set opt(tfrcFB)         1               ; # number of feedback reports in TFRC per RTT

set opt(snet_bw)        100Mb           ;# link BW on server access net
set opt(snet_prop)      2ms             ;# link propagation delay on server access net
set opt(core_bw)        100Mb           ;# link BW on core net (router0<--->router1)
set opt(core_prop)      2ms             ;# link propagation delay on core net (router0<
    --->router1)
set opt(bsnet_bw)       100Mb           ;# link BW on base station access net
set opt(bsnet_prop)     2ms             ;# link propagation delay on base station
    access net
set opt(wifi_bw)        11Mb            ;# link BW on wifi net

# ==================== END OPTIONS =====================================
# ==================== START PROCEDURES ================================
proc getopt {argc argv} {
        global opt
        for {set i 0} {$i < $argc} {incr i} {
                set arg [lindex $argv $i]
                if {[string range $arg 0 0] != "-"} continue
                set name [string range $arg 1 end]
                set opt($name) [lindex $argv [expr $i+1]]
                puts2 "opt($name): $opt($name)"
        }
}

proc puts2 {output} {
        global opt
        if { $opt(verbose)==1 } {
                puts $output
        }
}
proc setnofnodes {} {
        global opt ns
        set opt(nofmn) [expr $opt(nofTFRC)+$opt(nofUDP)+$opt(nofFTP)+$opt(nofWEB)]

        if { $opt(nofmn)>9 } {
                puts2 "nofmn>9. extend adress array!"
                puts2 "NS EXITING..."
                $ns halt
                exit 0
        }
        set opt(nofsn) 0
        if {$opt(nofTFRC)>0 } {
                set opt(nofsn) [expr $opt(nofsn) + $opt(nofTFRC)]
        }
        if {$opt(nofUDP)>0 } {
                incr opt(nofsn)
        }
        if {$opt(nofFTP)>0 } {
                incr opt(nofsn)
        }
        if {$opt(nofWEB)>0 } {
                incr opt(nofsn)
        }
        puts2 "Ordered one server per source type. nof servers: $opt(nofsn)"
}

proc wlan_topo_complex {} {
        global ns nodes opt mn sn rn bs
        puts2 "Building topology..."
        ########################################################
        #  Server,      Router,   Base-station and Mobile nodes
        #
        #  sn_0---\                              /----mn_0
        #          \                            /
        #  sn_1-----rn_0----------rn_1------bs_0------mn_1
        #            /               \          \        ...
        #  ...      /                 \    ....   \
        #          /                   \           \----mn_n
        #  sn_n--/                      \----bs_n
        #
        #  0.0.n    0.0.0        1.0.0    1.n.0      1.n.m
        ##########################################################

        # Configuration for Orinoco 802.11b 11Mbps PC card with ->22.5m range
        Phy/WirelessPhy set Pt_ 0.031622777              ;# transmit power
        Phy/WirelessPhy set L_ 1.0                       ;# System loss factor
        Phy/WirelessPhy set bandwidth_ 1 ;#$opt(wifi_bw)
        Phy/WirelessPhy set freq_ 2.472e9                ;# channel-13. 2.472GHz
        Phy/WirelessPhy set CPThresh_ 10.0               ;# reception of simultaneos
             packets if P1 > CPThresh_ x P2
        Phy/WirelessPhy set CSThresh_ 5.011872e-12       ;# carrier sensing threshold
        Phy/WirelessPhy set RXThresh_ 5.82587e-09        ;# reception threshold

        # LIMITATION: ns does not support ARF (Auto Rate Fallback, dynamic channel rate
             adjustment)
        Mac/802_11 set dataRate_ $opt(wifi_bw)

        # Most commercial 802.11b-cards have RTS/CTS disabled by default. RTSThreshold_
             = 3000 will
        # enable RTS only for packets above 3KB, which should be never.
        if { $opt(disableRTS_CTS)==1 } {
                puts2 "Disabling RTS/CTS"
                Mac/802_11 set RTSThreshold_ 3000
        }
        Mac/802_11 set basicRate_ 1Mb                    ;# for broadcast packets

        # address scheme set-up
        $ns node-config -addressType hierarchical
        AddrParams set domain_num_ 2             ;# sets two domains: 0.0.0 and 1.0.0
        if { $opt(nofbs)>1 } {
                puts2 "no support for more than one base station node"
                puts2 "NS EXITING..."
                $ns halt
                exit 0
        }
        lappend cluster_num  1 2                 ;# sets nof clusters pr domain (2 = one
             for router, one for each bs)
                                                ;# one cluster of servers and one
                                                    cluster of mobile nodes
        AddrParams set cluster_num_ $cluster_num
        # setting nof nodes pr cluster [clients+router], router, [clients+base station]
            :
        lappend eilastlevel [expr $opt(nofsn) + 1] 1 [expr $opt(nofmn) + 1]
        AddrParams set nodes_num_ $eilastlevel

        set topo [new Topography]
        $topo load_flatgrid $opt(x) $opt(y)
        # god needs to know the number of all wireless interfaces MN+BS
        create-god [expr $opt(nofmn) + $opt(nofbs)]

        # creating wired nodes: sn = server nodes and rn = router nodes
        set rn(0) [$ns node {0.0.0}]
        set rn(1) [$ns node {1.0.0}]
        puts2 "Router nodes:"
        puts2 [$rn(0) node-addr]
        puts2 [$rn(1) node-addr]
        # create more sn-adresses if necessary...
```

```
        if { $opt(nofsn)>9 } {
                puts2 "create for-loop to address more server nodes"
                puts2 "NS EXITING..."
                $ns halt
        }
        set sn_adr {0.0.1 0.0.2 0.0.3 0.0.4 0.0.5 0.0.6 0.0.7 0.0.8 0.0.9}
        puts2 "Server nodes:"
        for {set i 0 } { $i < $opt(nofsn) } { incr i } {
                set sn($i) [$ns node [lindex $sn_adr $i]]
                puts2 [$sn($i) node-addr]
        }

        if {$opt(prop) == "Propagation/Shadowing" } {
                # first set values of shadowing model
                Propagation/Shadowing set pathlossExp_  $opt(pathlossExp)       ;# path
                     loss exponent
                Propagation/Shadowing set std_db_       $opt(std_db)            ;#
                    shadowing deviation (dB)
                Propagation/Shadowing set dist0_        $opt(dist0)             ;#
                    reference distance (m)
                Propagation/Shadowing set seed_         $opt(seed)              ;# seed
                     for RNG
                puts2 "values for shadowing propagation are set"
        }
        # configuring node for 802.11-interface.
        $ns node-config -adhocRouting $opt(adhocRouting) \
                -llType $opt(ll) \
                -macType $opt(mac) \
                -ifqType $opt(ifq) \
                -ifqLen $opt(ifqlen) \
                -propInstance [new $opt(prop) ] \
                -antType $opt(ant) \
                -phyType $opt(netif) \
                -wiredRouting ON \
                -channel [new $opt(chan)] \
                -topoInstance $topo \
                -agentTrace $opt(agentTrace) \
                -routerTrace $opt(routerTrace) \
                -macTrace $opt(macTrace) \
                -movementTrace $opt(movementTrace)
#       -propType $opt(prop) \

        # create base-station
        if { $opt(nofbs)>1 } {
                puts2 "create for-loop to add several base-stations"
                puts2 "NS EXITING..."
                $ns halt
                exit 0
        } else {
                # creating and setting network hier-address
                set bs(0) [$ns node {1.1.0}]
                # bs is not moving:
                $bs(0) random-motion 0
                $bs(0) set X_ 1.0
                $bs(0) set Y_ 1.0
                $bs(0) set Z_ 0.0
                puts2 "Base station nodes:"
                puts2 [$bs(0) node-addr]
        }

        # configure for "nofmn" mobilenodes
        $ns node-config -wiredRouting OFF
        if { $opt(nofbs)>1 } {
                # TODO: legg til kode for flere bs om det trengs
                puts2 "create for-loop to address mobile nodes"
                puts2 "under several base-stations"
                puts2 "NS EXITING..."
                $ns halt
                exit 0
        } else {
                set x_pos 0.0
                set y_pos 0.0
                # create more mn-adresses if necessary...
                if { $opt(nofmn)>9 } {
                        puts2 "create for-loop to address more mobile nodes"
                        puts2 "NS EXITING..."
                        $ns halt
                        exit 0
                }
                set adr {1.1.1 1.1.2 1.1.3 1.1.4 1.1.5 1.1.6 1.1.7 1.1.8 1.1.9}
                puts2 "Mobile nodes: "
                for {set j 0 } { $j < $opt(nofmn) } { incr j } {
                        set mn($j) [$ns node [lindex $adr $j]]
                        puts2 [lindex $adr $j]
                        # setting coordinates and motion
                        $mn($j) random-motion $opt(mn_motion)
                        $mn($j) set X_ $opt(nodeDist) ;#[expr $x_pos + $opt(nodeDist)
                            /2]
                        $mn($j) set Y_ $opt(nodeDist) ;#[expr $y_pos + $opt(nodeDist)
                            /2]
                        $mn($j) set Z_ 0.0
                        # each new mn is longer away from bs:
                        set x_pos [expr $x_pos + $opt(nodeDist)/2]
                        set y_pos [expr $y_pos + $opt(nodeDist)/2]
                        $mn($j) base-station [AddrParams addr2id [$bs(0) node-addr]]
                        $ns initial_node_pos $mn($j) 20
                }
        }

        # linking up servers to first core router
        set link_angle 135
        set angle_offset [expr 90/$opt(nofsn)]
        for {set k 0 } { $k < $opt(nofsn) } { incr k } {
                $ns duplex-link $sn($k) $rn(0) $opt(snet_bw) $opt(snet_prop) $opt(qm)
                #$ns duplex-link-op $rn(0) $sn($k) orient $link_angle
                set link_angle [expr $link_angle + $angle_offset]
        }
        # core link
        $ns duplex-link $rn(0) $rn(1) $opt(core_bw) $opt(core_prop) $opt(qm)
        #$ns duplex-link-op $rn(0) $rn(1) orient right

        # linking up base stations to second core router
        for {set l 0 } { $l < $opt(nofbs) } { incr l } {
                $ns duplex-link $rn(1) $bs($l) $opt(bsnet_bw) $opt(bsnet_prop) $opt(qm)
                #$ns duplex-link-op $rn(1) $bs($l) orient right-up
        }

        puts2 "done setting up topology"
}

proc ping {} {
        global opt ns mn sn
        set tmp 0
        set tmp2 0
```

```
# establish route with ping. each type to a seperate server:
for {set m $tmp2 } { $m < $opt(nofFTP) } { incr m} {
        set ping_downstream [$ns create-connection Ping $sn($tmp) Ping $mn($m)
            9]
        set ping_upstream [$ns create-connection Ping $mn($m) Ping $sn($tmp) 9]
        puts2 "FTP-link: ping from [$sn($tmp) node-addr] to [$mn($m) node-addr]
             and back!"
        $ping_downstream oneway
        $ping_upstream oneway
        $ns after 0.1 "$ping_downstream send"
        $ns after 0.2 "$ping_upstream send"
        if {$m == [expr $opt(nofFTP) - 1] } {
                incr tmp
        }
        incr tmp2
}

set tmp3 $tmp2
for {set m $tmp2 } { $m < $opt(nofWEB) } { incr m} {
        set ping_downstream [$ns create-connection Ping $sn($tmp) Ping $mn($m)
            9]
        set ping_upstream [$ns create-connection Ping $mn($m) Ping $sn($tmp) 9]
        puts2 "WEB-link: ping from [$sn($tmp) node-addr] to [$mn($m) node-addr]
             and back!"
        $ping_downstream oneway
        $ping_upstream oneway
        $ns after 0.3 "$ping_downstream send"
        $ns after 0.4 "$ping_upstream send"
        if {$m == [expr $opt(nofWEB) - 1] } {
                incr tmp
        }
        incr tmp2
}

set tmp3 $tmp2
for {set m $tmp2 } { $m < [expr $opt(nofUDP) + $tmp3] } { incr m} {
        set ping_downstream [$ns create-connection Ping $sn($tmp) Ping $mn($m)
            9]
        set ping_upstream [$ns create-connection Ping $mn($m) Ping $sn($tmp) 9]
        puts2 "UDP-link: ping from [$sn($tmp) node-addr] to [$mn($m) node-addr]
             and back!"
        $ping_downstream oneway
        $ping_upstream oneway
        $ns after 0.5 "$ping_downstream send"
        $ns after 0.6 "$ping_upstream send"
        if {$m == [expr $opt(nofUDP) + $tmp3 - 1] } {
                incr tmp
        }
        incr tmp2
}
set tmp3 $tmp2
for {set m $tmp2 } { $m < [expr $opt(nofTFRC) + $tmp3] } { incr m} {
        set ping_downstream [$ns create-connection Ping $sn($tmp) Ping $mn($m)
            9]
        set ping_upstream [$ns create-connection Ping $mn($m) Ping $sn($tmp) 9]
        puts2 "TFRC-link: ping from [$sn($tmp) node-addr] to [$mn($m) node-addr
            ] and back!"
        $ping_downstream oneway
        $ping_upstream oneway
        $ns after 0.7 "$ping_downstream send"
        $ns after 0.8 "$ping_upstream send"
        if {$m == [expr $opt(nofTFRC) + $tmp3 - 1] } {
```

```
                        incr tmp
                }
                incr tmp2
        }
}

proc stop {} {
        global ns opt mn sn rn bs

        for {set j 0 } { $j < $opt(nofmn) } { incr j } {
                $ns at $opt(stop).0000010 "$mn($j) reset";
        }
        for {set i 0 } { $i < $opt(nofsn) } { incr i } {
                $ns at $opt(stop).0000010 "$sn($i) reset";
        }
        $ns at $opt(stop).1 "puts2 \"NS EXITING...\" ; $ns halt"

        exit 0
}

proc record { sink bwt } {
        global opt
        #Get an instance of the simulator
        set ns [Simulator instance]
        #Set the time after which the procedure should be called again
        set time $opt(sampleTime)
        #How many bytes have been received by the traffic sink?
        set bw [$sink set bytes_]
        #Get the current time
        set now [$ns now]
        #Open a file to write bandwidth-trace
        #Calculate the bandwidth (in MBit/s) and write it to the file
        puts $bwt "$now [expr $bw/$time*8/1000000]"
        #Reset the bytes_ values on the traffic sinks
        $sink set bytes_ 0
        #Re-schedule the procedure
        $ns at [expr $now+$time] "record $sink $bwt"
}

proc prepare_evalvid_trace {} {
        global ns opt trace_file trace_file_id
        puts2 "------------------------trace prepare
            start------------------------------"
        puts2 "Start making GOP and frame trace files for $opt(q_variants) rate
            variants"
        for {set i 1} {$i <= $opt(q_variants)} {incr i} {
                set original_file_name($i) nsin/st_$opt(videoname).yuv_Q[expr $i + 1]
                    .txt
                set original_file_id($i) [open $original_file_name($i) r]
        }
        set trace_file_name $opt(out_dir)/video2.dat
        set trace_file_id [open $trace_file_name w]
        set trace_file [new vbrTracefile2]
        $trace_file filename $trace_file_name
        set frame_count 0
        set last_time 0

        # AL: toggle between multiple input files!
        # set original_file_id $original_file_id(1)
        set source_select 1

        set frame_size_file     $opt(out_dir)/frame_size.dat
        set gop_size_file       $opt(out_dir)/gop_size.dat
        set frame_size_file_id  [open $frame_size_file w]
        set gop_size_file_id    [open $gop_size_file w]

        for {set i 1} {$i <= $opt(q_variants)} {incr i} {
                set frame_size($i) 0
                set gop_size($i) 0
        }
        set gop_numb 0
        # Convert ASCII sender file on frame size granularity to ns-2 adapted internal
            format
        while {[eof $original_file_id(1)] == 0} {
                for {set i 1} {$i <= $opt(q_variants)} {incr i} {
                        gets $original_file_id($i) current_line($i)
                }

                scan $current_line(1) "%d%s%d%s%s%s%d%s" no_ frametype_ length_ tmp1_
                    tmp2_ tmp3_ tmp4_ tmp5_
                #puts2 "$no_ $frametype_ $length_ $tmp1_ $tmp2_ $tmp3_ $tmp4_ $tmp5_"

                set time [expr 1000 * 1000/$opt(fps)] ;# Note that this is time between
                     frames in number of us

                if { $frametype_ == "I" } {
                        set type_v 1
                        set time 0
                }
                if { $frametype_ == "P" } {
                        set type_v 2
                }
                if { $frametype_ == "B" } {
                        set type_v 3
                }

                # Write to GOP size file after each H-frame found:
                if { $frametype_ == "H" } {
                        set puts_string "$gop_numb"
                        for {set i 1} {$i <= $opt(q_variants)} {incr i} {
                                set puts_string "$puts_string $gop_size($i)"
                        }
                        puts $gop_size_file_id $puts_string
                        incr gop_numb
                        set type_v 0 ;# Must have different type than I-frame so that
                            the LB(r,b) algorithm finds it!
                }
                # Write to frame_size.dat:
                set puts_string "$no_"
                for {set i 1} {$i <= $opt(q_variants)} {incr i} {
                        set puts_string "$puts_string $gop_size($i)"
                }
                puts $frame_size_file_id $puts_string

                for {set i 1} {$i <= $opt(q_variants)} {incr i} {
                        scan $current_line($i) "%d%s%d%s%s%s%d%s" no_ frametype_ length
                            ($i) tmp1_ tmp2_ tmp3_ tmp4_ tmp5_
                        set gop_size($i) [expr $gop_size($i) + $length($i) ]
                }

                # Write to video2.dat:
                set puts_string "$time $length_ $type_v $opt(ra_mtu)"
                for {set i 2} {$i <= $opt(q_variants)} {incr i} {
                        set puts_string "$puts_string $length($i)"
                }
                puts  $trace_file_id $puts_string
                if { $frametype_ != "H" } {
                        incr frame_count
                }
        }
        puts2 "#of frames written to GOP and Frame trace files: $frame_count"
        #close $original_file_id
        close $trace_file_id  ;# Note that this new trace file is closed for writing
            and
        # opened below for reading through being a new Tracefile in
            eraTraceFile2::setup()
        puts2 "------------------------trace prepare
            stop-------------------------------"
}

proc attach_FTP_traffic { startnode endnode connection_number} {
        global ns opt tcp tcp_in_use ftp_server
        $startnode color red
        $endnode color red
        set start_id [$startnode id]
        set end_id [$endnode id]
        set tcp($connection_number) [new Agent/TCP]
        $tcp($connection_number) set fid_ 40
        set tcpsink($connection_number) [new Agent/TCPSink]
        $ns attach-agent $startnode $tcp($connection_number)
        $ns attach-agent $endnode $tcpsink($connection_number)
        $ns connect $tcp($connection_number) $tcpsink($connection_number)
        set ftp_server($connection_number) [new Application/FTP]
        $ftp_server($connection_number) attach-agent $tcp($connection_number)

        set bwt [open $opt(out_dir)/bwt.FTP.$connection_number.tr w] ;#$opt(
            bwtDelimiter).
        $ns at 0.0 "record $tcpsink($connection_number) $bwt"
        puts2 "added TCP connection number $connection_number  between nodes $start_id
            and $end_id"
}
# ======================================================================
# Web traffic generator by Sally Floyd (from web.tcl in ns tcl/ex)
# Modified by TH
# ======================================================================
proc attach_WEB_traffic {startnode endnode connection_number} {
        global ns opt pool

        $pool set-server 0 $startnode
        $pool set-client $connection_number $endnode

        set start_id [$startnode id]
        set end_id [$endnode id]
        $pool set-num-session $opt(nofWebSessions)
        for {set i 0} {$i < $opt(nofWebSessions)} {incr i} {
                set interPage [new RandomVariable/Exponential]
                $interPage set avg_ $opt(interPage)
                set pageSize [new RandomVariable/Constant]
                $pageSize set val_ $opt(pageSize)
                set interObj [new RandomVariable/Exponential]
                $interObj set avg_ [expr 0.01]
                set objSize [new RandomVariable/ParetoII]
                $objSize set avg_ $opt(webObjectSize)
                $objSize set shape_ $opt(paretoShape)
                set tmp [new RandomVariable/Uniform]
                set startTime [expr 1.0 * [$tmp value] * $opt(stop) ]
                # Poisson start times for sessions.
                # when does a web session end?
                puts [format "WEB session $i startTime %0.3f" $startTime ]
                $pool create-session $i $opt(nofWebPages) $startTime $interPage
                    $pageSize $interObj $objSize
                # All of the web traffic uses the same packet size.
                # webcache/webtraf.cc
                # doc/webcache.tex
                # Feldmann99a, Section 2.4, paragraph 3-4 and the appendix A.1.
                # empweb/empftp.cc
                # doc/session.tex
                # When does each web session start?
        }
        # using web-throughput.awk to log throughput
        puts2 "added WEB connection number $connection_number with $opt(nofWebSessions)
             sessions  between nodes $start_id and $end_id"
}

proc attach_UDP_traffic { startnode endnode connection_number} {
        global ns opt udp vbr_in_use vbr trace_file
        $startnode color blue
        $endnode color blue
        set start_id [$startnode id]
        set end_id [$endnode id]
        set udp($connection_number) [new Agent/eraUDP]
        $udp($connection_number) set fid_ $connection_number
        $ns attach-agent $startnode $udp($connection_number)
        set vbr($vbr_in_use) [new Application/Traffic/eraVbrTrace]
        $vbr($vbr_in_use) attach-agent $udp($connection_number)

        $udp($connection_number) set packetSize_ $opt(udp_pktsize)
        $udp($connection_number) set TOS_field_ 1     ;# New 120905: tag ECF enabled
            sources! 0 is default.
        # Limit the number of send trace files:
        if {$connection_number < 15} {
                # Connect a file name to UDP source to write transmit trace data
                $udp($connection_number) set_filename $opt(out_dir)/
                    sd_be.udp.$connection_number
        }
        $vbr($vbr_in_use) set running_ 0
        $vbr($vbr_in_use) set r_ $opt(vbr_rate) ;# Set the rate instead of packet
            interval
        $vbr($vbr_in_use) attach-tracefile $trace_file          ;# video2.dat type
            vbrTracefile2
        $vbr($vbr_in_use) set b_ 1.5
        $vbr($vbr_in_use) set q_ 1
        $vbr($vbr_in_use) set GoP_ $opt(GoP_size)
        $vbr($vbr_in_use) set fps_ $opt(fps)
        $vbr($vbr_in_use) set isTFRC_ 0

        set udpsink($connection_number) [new Agent/eraUdpSink]
        $ns attach-agent $endnode  $udpsink($connection_number)
        $ns connect $udp($connection_number) $udpsink($connection_number)
        if {$connection_number == 0} {
                # Connect a file name to TFRC sink to write receivce trace data
                $udpsink($connection_number) set_trace_filename $opt(out_dir)/
                    rd_be.udp.$connection_number
        }
        $udpsink($connection_number) set bytes_ 0
        set bwt [open $opt(out_dir)/bwt.UDP.$connection_number.tr w] ;#$opt(
            bwtDelimiter).
        $ns at 0.0 "record $udpsink($connection_number) $bwt"
        puts2 "added UDP connection number $connection_number (vbr($vbr_in_use))
            between nodes $start_id and $end_id"
        incr vbr_in_use
}

# ========================================================================
# Add Evalvid−RA RA−SVBR sources
# ========================================================================
proc attach_TFRC_traffic {startnode endnode connection_number} {
        global ns tfrc opt trace_file vbr_in_use vbr
        $startnode color green
        $endnode color green
        set start_id [$startnode id]
        set end_id [$endnode id]
        set tfrc_agent($connection_number) [new Agent/TFRC]
        $tfrc_agent($connection_number) set fid_ $connection_number
        $ns attach-agent $startnode $tfrc_agent($connection_number)
        set vbr($vbr_in_use) [new Application/Traffic/eraVbrTrace]
        $vbr($vbr_in_use) attach-agent $tfrc_agent($connection_number)

        $tfrc_agent($connection_number) set packetSize_ $opt(tfrc_pktsize) ;# this is
            the MTU for the TFRC
        $tfrc_agent($connection_number) set TOS_field_ 1                ;# New 120905:
            tag ECF enabled sources! 0 is default.

        if {$connection_number < 15} {
                # Connect a file name to TFRC source to write transmit trace data:
                $tfrc_agent($connection_number) set_filename $opt(out_dir)/
                    sd_be.tfrc.$connection_number
        }

        $vbr($vbr_in_use) set running_ 0
        $vbr($vbr_in_use) set r_ $opt(vbr_rate) ;# Set the rate instead of packet
            interval
        puts2 "r_ = $opt(vbr_rate)"
        $vbr($vbr_in_use) attach-tracefile $trace_file
        $vbr($vbr_in_use) set b_ 1.5
        $vbr($vbr_in_use) set q_ 4
        $vbr($vbr_in_use) set GoP_ $opt(GoP_size)
        $vbr($vbr_in_use) set fps_ $opt(fps)
        $vbr($vbr_in_use) set isTFRC_ 1

        set tfrcsink($connection_number) [new Agent/TFRCSink]
        $ns attach-agent $endnode  $tfrcsink($connection_number)
        $ns connect $tfrc_agent($connection_number) $tfrcsink($connection_number)

        if {$connection_number == 0} {
                # Connect a file name to TFRC sink to write receivce trace data:
                $tfrcsink($connection_number) set_trace_filename $opt(out_dir)/
                    rd_be.tfrc.$connection_number
        }
        set bwt [open $opt(out_dir)/bwt.TFRC.$connection_number.tr w] ;# $opt(
            bwtDelimiter).
        $ns at 0.0 "record $tfrcsink($connection_number) $bwt"
        puts2 "added TFRC connection number $connection_number (vbr($vbr_in_use))
            between nodes $start_id and $end_id"
        incr vbr_in_use
}

# ========================================================================
# The interface between TFRC Receive (of ACKS)
# and the adaptive rate controller of the VBR application
# ======================================================================
Agent/TFRC instproc tfrc_ra {bytes_per_sec backlog} {
        global vbr ns opt sn_in_use
        set now [$ns now]
        $self instvar node_
        set node_id [$node_ id]
        # the following is needed to find correct value for $vbr_number,
        # which is the index for the server instance (not the physical node,
        # but the agent) that is to receive the TFRC ack-packets:
        set vbr_number [expr $node_id + $opt(nofUDP) - 2] ;# adding values for
            allocated udp-sources, subtracting the core nodes (numbered 0 and 1)
        if { $opt(nofFTP) > 0 } {
                set vbr_number [expr $vbr_number - 1]
        }
        if { $opt(nofWEB) > 0 } {
                set vbr_number [expr $vbr_number - 1]
        }
        if { $opt(nofUDP) > 0 } {
                set vbr_number [expr $vbr_number - 1]
        }
        puts2 "tfrc_ra behandler ack til vbr($vbr_number) (node($node_id)) B/s:
            $bytes_per_sec"
        #puts "In TFRC instproc. rate = $bytes_per_sec (B/s), node_id = $node_id"
        #TODO: dobbeltsjekk at det blir riktig node her!!
        #puts "TCL: before vbr($vbr_number) TFRC_rateadapt rate=$bytes_per_sec
            node=$node_id"
        $ns at [expr $now] "$vbr($vbr_number) TFRC_rateadapt $bytes_per_sec $node_id
            $backlog"
        #puts "TCL: after vbr($vbr_number) TFRC_rateadapt rate=$bytes_per_sec
            node=$node_id"
}

# ======================================================================
# Function neede to receive pings properly.
# ======================================================================
Agent/Ping instproc recv {from seq ff bf} {
        $self instvar node_
        puts "node [$node_ id] received ping answer from \
                $from seq $seq with forward delay $ff and backward $bf ms."
#       puts $rttf "$seq [expr $ff/1000] [expr $bf/1000]"  ;# for RTT-logging, this can
     be used
}

# ==================== END PROCEDURES ==================================
# ==================== START EXEC: ENVIRONMENT =========================

set ns [new Simulator]

getopt $argc $argv

#set opt(bwtDelimiter)  -$opt(pathlossExp)-$opt(std_db)-$opt(nodeDist)m-$opt(ecn)ecn
        ;# variable(s) to index the bwt-files

puts2 "\n--------new sim---------"

if {$opt(seed) > -1} {
        puts2 "Seeding Random number generator with $opt(seed)"
        ns-random $opt(seed)
}
#
# Create trace
#
```

```
$ns use-newtrace
if { $opt(doTrace) } {
        set tf [open $opt(tf) w]
        $ns trace-all $tf
}
if {$opt(nam)} {
        #Open the files for nam trace
        set nf [open $opt(namtf) w]
        $ns namtrace-all $nf
        #$ns namtrace-all-wireless $nf $opt(x) $opt(y)
}

#defining colors for better readability
$ns color 40 red
$ns color 41 blue
$ns color 42 green
$ns color 43 chocolate

#
# RED and TCP parameters
#
if {$opt(ecn) == 1} {
        set opt(qm) RED
        set opt(ifq) Queue/RED
        Queue/RED set setbit_ true
}
if {$opt(qm) == "RED" } {
        Queue/RED set summarystats_ true
        Queue/RED set adaptive_ $opt(adaptive)
        Queue/RED set q_weight_ 0.0
        Queue/RED set thresh_ $opt(minth)
        Queue/RED set maxthresh_ $opt(maxth)
}
if {$opt(qm) == "DropTail" } {
        #Queue/DropTail set summarystats_ true
        #Queue/DropTail set shrink_drops_ true
}

if {$opt(nofTCP) > 0 } {
        Agent/TCP set ecn_ $opt(ecn)
        Agent/TCP set packetSize_ $opt(tcp_pktsize)
        Agent/TCP set window_ $opt(window)
}

#
# Setting global TFRC defaults:
#
if {$opt(nofTFRC) > 0 } {
        Agent/TFRC set ss_changes_ 1            ;# Added on 10/21/2004
        Agent/TFRC set slow_increase_ 1         ;# Added on 10/20/2004
        Agent/TFRC set rate_init_ 2
        Agent/TFRC set rate_init_option_ 2      ;# Added on 10/20/2004
        Agent/TFRC set SndrType_ 1
        Agent/TFRC set oldCode_ false
        Agent/TFRC set packetSize_ $opt(tfrc_pktsize)
        Agent/TFRC set maxqueue_ 500
        Agent/TFRC set printStatus_ true
        Agent/TFRC set ecn_ $opt(ecn)           ;# Enable ECN
        #Agent/TFRC set useHeaders_ false       ;# "useHeaders_" is removed from the
            evalvid adjusted tfrc.cc (TH)
        #Agent/TFRCSink set NumFeedback_ $opt(tfrcFB)
}
```

```
#
# Create wlan topology
#
setnofnodes              ;# adjusts total nof servers and mobile nodes according to
     specified nof independent streams
wlan_topo_complex        ;# sets up the topology with chosen amount of streams, one
     server for each source type
                         ;# and one mobile node for each stream

set mn_in_use 0
set sn_in_use 0
set vbr_in_use 0

if { $opt(nofTFRC)>0 || $opt(nofUDP)>0 } {
        prepare_evalvid_trace
}

#
# Set up forward FTP connection between server and mobile node
#
if {$opt(nofFTP) > 0} {
        for {set i 0} {$i < $opt(nofFTP)} {incr i} {
                attach_FTP_traffic $sn($sn_in_use) $mn($mn_in_use) $i
                incr mn_in_use
        }
        incr sn_in_use
}
#
# Set up forward WEB connection between server and mobile node
#
if {$opt(nofWEB) > 0} {
        PagePool/WebTraf set FID_ASSIGNING_MODE_ 2
        # gen/ns_tcl.cc, for 2, use sameFid_
        # for 1, increment FID.
        # for 0, use id_.
        set pool [new PagePool/WebTraf]
        $pool set-num-client $opt(nofWEB)
        $pool set-num-server 1  ;# all clients connect to the same web-server for
             simplicity
        # $pool set sameFid_ $count
        $pool set sameFid_ 2
        for {set i 0} {$i < $opt(nofWEB)} {incr i} {
                attach_WEB_traffic $sn($sn_in_use) $mn($mn_in_use) $i
                incr mn_in_use
        }
        incr sn_in_use
}

#
# Set up forward UDP connection between server and mobile node
#
if {$opt(nofUDP) > 0} {
        for {set i 0} {$i < $opt(nofUDP)} {incr i} {
                attach_UDP_traffic $sn($sn_in_use) $mn($mn_in_use) $i
                incr mn_in_use
        }
        incr sn_in_use
}

#
# Set up forward TFRC connection
```

```
#
if {$opt(nofTFRC) > 0} {
        for {set i 0} {$i < $opt(nofTFRC)} {incr i} {
                attach_TFRC_traffic $sn($sn_in_use) $mn($mn_in_use) $i ;# $bs(0)
                incr mn_in_use
                incr sn_in_use          ;# each TFRC session needs a seperat source to
                    allow seperat video-feeds
        }
}


# ======================================================================
# Starting and stopping sources
# ======================================================================
if {$opt(doPing)==1} {
        ping
}
for {set i 0} { $i < $opt(nofFTP) } {incr i} {
        # start ftp sources
        $ns at [expr 10] "$ftp_server($i) start"
        $ns at [expr 50] "$ftp_server($i) stop"
}
#
# start vbr sources
#

# start UDPs
#$ns at 9.98 "$vbr(0) start"
for {set i 0} { $i < $opt(nofUDP) } {incr i} {
        $ns at 10 "$vbr($i) start"
        #$ns at 50 "$vbr($i) stop"
}
# start first TFRC
#$ns at 9.8 "$vbr($opt(nofUDP)) start"
# start the rest of the TFRCs
for {set i $opt(nofUDP)} { $i < $vbr_in_use } {incr i} {
        $ns at 10 "$vbr($i) start"
        #$ns at 50 "$vbr($i) stop"
}

$ns at $opt(stop) "stop"
puts2 ""
puts2 "-----Starting simulation!-----"
$ns run
```