High Performance Video Streaming on Low End Systems

M. Sc. Thesis in Electrical Engineering performed at
Telia Research AB, Nättjänster Division

Alexander Engman
KTH

Tutor at Telia Research AB:	Joakim Bergkvist
Examiner at KTH:	Gunnar Karlsson

Stockholm, Dec 1999

Abstract

Lately there has been vast improvement of bandwidth in networks. New technologies like wavelength division multiplexing and gigabit routers are moving the frontline of broadband communication. As motivation for these new innovations, the customers' presumed demand for video applications over the Internet is often mentioned.

The goal of this thesis has been to write a video streamer for pre-recorded data that could provide high bitrate output without demanding expensive equipment. This streamer could then be used to demonstrate the advantages of new network features like bandwidth reservation.

A streamer should preferably send the data smoothly in small blocks frequently rather than larger blocks with longer time in between. One of the difficulties in shaping the data output in this way is the limitation in time granularity that is experienced when running several processes on a single CPU.

Another difficulty is in handling the jitter and delay that inevitable appear when sending data over shared media. Furthermore, the client must display the video at the correct rate.

In consideration of these problems, a working client - streamer pair for uncompressed and H.263 coded video has been implemented. The streamer can deliver up to 10 Mbps to several clients, and it handles delay by buffering data at the receiver. The video is accessed via a web page.

Acknowledgements

I would like to express my gratitude to my tutor Joakim Bergkvist; as well as to David Ahlard, Istvan Cselenyi and other colleagues at Telia Research AB, Farsta for their help and support in this work. I would also like to thank my examiner Gunnar Karlsson at KTH for his guidance and patience.

HIGH PERFORMANCE VIDEO STREAMING ON LOW END SYSTEMS

1 Introduction

1.1 Background

Today Telia Research has a need for applications to demonstrate different capabilities of its network, such as resource reservation. Video is one application that can be used to show the potential advantages in this regard.

There are commercial video-streaming products available today, using dedicated hardware solutions, but they are expensive and difficult to modify for different tasks. There are also software-based streaming products, like Real Video, aimed at the consumer market. Real Video makes it possible to compress video (with degradation in quality) to such low bit rates that it is possible to transfer it via modem. Though Real Video is inexpensive, it is not capable of sending high quality video at Mb/s rates.

1.2 Purpose

The aim of this thesis is to develop a working client/streamer pair. The streamer application should run on a standard PC. When receiving requests, it streams the contents at high speed via UDP (user datagram protocol). Furthermore, the streamer should be able to compensate for the conditions of the shared media i.e. jitter and delay, and avoid sending the traffic in bursts. The streamer should also be able to handle several concurrent clients. The client should request, receive and display the contents. The system should be easy to modify to meet future demands.

1.3 Delimitations

1. The streamer need not support every video format without modification. It is primarily intended to present a working solution.

2. The streamer and client parts need not handle the problem that packets might arrive out of order when transferred over long distances.

3. The streamer will only handle constant bit rate video.

1.4 Structure of the report

Chapter 2 gives a brief introduction to some of the aspects that need to be considered when transferring digital video over packet-switched networks.

Chapter 3 discusses some of the problems with making a streamer and how they have been handled in this thesis.

Chapter 4 shows the experimental setup, the choice of video coding technique, streaming format, and the rate control mechanism.

Chapter 5 presents and discusses what has been accomplished and the results.

Chapter 6 presents additional measurements and analyzes.

Chapter 7 gives the conclusions and addresses future improvements.

2 Background to digital video and streaming

This chapter gives a brief introduction to digital video and some of the aspects that need to be considered when transferring digital video over packet-switched networks.

2.1 Video compression

The main purpose of compression is to reduce the size of video files in order to save bandwidth and storage space. Video takes up huge amounts of space. The television standard PAL[1] for example would need a bit rate of 216 Mbps (25 frames per second, 864 x 625 luminance[2] samples, 429 x 525 x 2 chrominance samples, 8 bits per sample resulting in a bit rate 25 x 8 x ((864 x 625) + (432 x 625 x 2)) = 216 Mbps) [22]. For a video studio this can be handled, but it is too much for transmission over public networks.

2.1.1 Lossy and lossless compression

There are two types of compression: lossy and lossless compression. The lossless compression removes statistically redundant information. The process is reversible and compression ratios of up to about 1:3 can be achieved. To compress the data even more, lossy compression has to be used. Ratios of over 1:20 is possible, at the price of reduced quality. The lossy compression removes the subjectively redundant information. This process destroys some of the data and is therefore not reversible. Lossless compression is almost exclusively used for medical and scientific purposes where distortion of the images can not be tolerated.

2.1.2 Forward error correction (FEC)

A drawback of removing most of the redundant information, is that compressed video is very sensitive to losses and errors. One way to improve the quality when there are frequent losses and errors is to use forward error correction (FEC).

With FEC, extra information is added by coding to support corrections of errors or losses. The amount that can be corrected depends on how much extra information that has been

[1] PAL (Phase Alternation Line television format) is the color television standard that is common in Europe.
[2] Since the eye has a higher sensitivity to light than color, the video has a higher resolution in light (luminance) samples than in color (chrominance) samples.

FEC might be good for reducing the loss rate, but the addition of redundant information means that more packets needs to be transmitted. The extra packets might in fact increase the congestion in the network and thus cause even more packet loss (if the stream represents a relatively large part of the traffic on a link).

2.2 Video coding techniques

There are several different coding techniques for digital video. Some of the most used standards are: MPEG1 (ISO/IEC 11172) and MPEG2 (ISO/IEC 13818), H.261 and H.263 (ITU-T).

2.2.1 MPEG1

The MPEG1 standard was developed for coding video at a bitrate of about 1.5 Mbps. It was developed for video-CD and CD-i media. Recommended image size is 360 x 240 pixels, and at the bitrate of 1.5 Mbps the quality is comparable to VHS.

An MPEG video sequence is divided into a group of pictures (GOP). There are three different types of coded pictures/frames in a GOP (see *Fig. 1*):

- I pictures (intracoded frames) are self-contained frames which act as reference for inter-coded P and B frames.
- P pictures (forward-predicted frames) are encoded using motion prediction from previous I or P pictures. The prediction error together with the motion vector are encoded.
- B pictures (bidirectionally-predicted frames) are encoded by using interpolated motion prediction between a previous I or P picture and a following I or P picture.

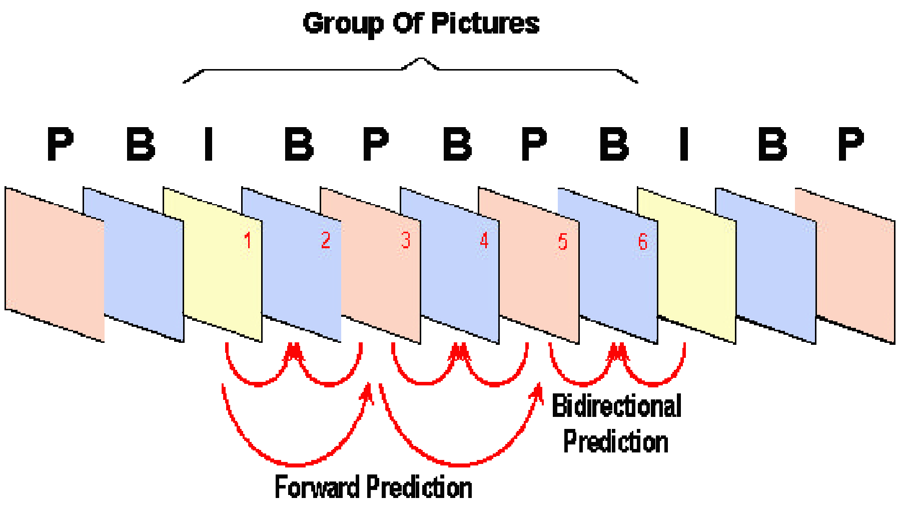


Fig. 1 GOP

With MPEG at least two or three frames have to be buffered at the destination in order to decode, given the interframe dependencies.

Techniques that are used to compress the I pictures are:

- Frequency based transform – discrete cosine transform (DCT) converts the color and brightness signals into frequency coefficients. The DCT is a lossless process that makes it easier to compress the signal.
- Quantization, results in lossy compression where it is possible to specify how much information that can be acceptably lost from the visable information. The image parts that are important to the human eye are represented more precisely while irrelevant information is represented with less precision.
- Huffman coding, which is a lossless compression that uses code tables based on statistics of the encoded data.

2.2.2 MPEG2

The MPEG2 standard was developed to meet demands for high quality video coding. It is similar to MPEG1 but has a greater collection of coding techniques. One example of improvements is that MPEG2 handles interlaced video (like the scanning pattern for television) better than MPEG1, which was developed for progressive (non-interlaced) video. For applications that use broadcast quality like DVD, streams of 4 to 6 Mbps are usually used. A proposed MPEG3 standard intended for high-definition television (HDTV) with bitrates of up to 40 Mbps, was merged into the MPEG2 standard at the beginning of the standardization work.

2.2.3 H.261

The H.261 standard is for a real-time coding with low delay. It has been developed for videoconferences and video telephony via integrated services digital network (ISDN). The coding algorithm allows transmission-rates in multiples (1-30) of 64 kbps. The image size is based on the CIF-format (Common Intermediate Format) (see *Fig. 2*) that has a resolution of 352 x 288 pixels. When transmitting over low bandwidth links like an ISDN channel (ISDN supports two channels of 64 kbps each) and analog modem, quarter CIF (QCIF) with a resolution of 176 x 144 is often used instead of CIF to get a better frame rate.

The basic idea of the algorithm is to divide each picture into macro blocks of 16 x 16 pixels, and try to find similarities between consecutive images, in order to reduce the information. Motion prediction is used to find out how parts in the image move.

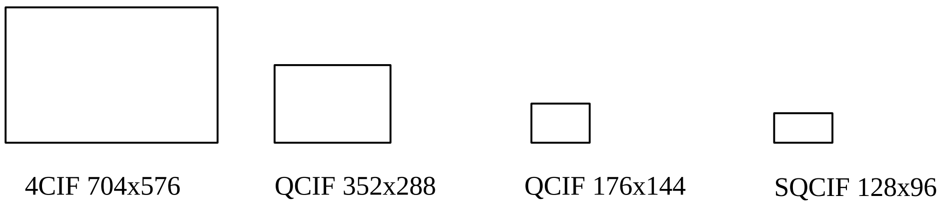


Fig. 2 Common intermediate format

2.2.4 H.263

The H.263 standard is an enhancement of H.261. It was developed to support the V.34 modem standard. Many improvements have been done to support the low bit rate of V.34 (28.8 kbps). Better motion compensation: a part of a predicted block is allowed to be outside the picture, advanced prediction mode and PB frames mode. PB is a P and a B frame coded as one unit (cf. MPEG coding). H.263 also supports a new format for low bandwidth applications, sub-QCIF (128 x 96) which has half as many pixels as QCIF. For higher resolution, there are new optional standards, 4CIF and 16CIF with resolutions of 704 x 576 and 1408 x 1152 pixels.

2.3 Layered coding

Layered coding is used to divide a video into bitstreams of different importance. First there is a base layer which only offer a rudimentary quality, then there are several add-on layers that increase the quality and in some cases the resolution. The advantage of using layered coding is that the base layer can be sent with high priority and the add-on layers with best effort. This means that the client will be guaranteed to get an image of basic quality and a better image if the resources allow it. When there are clients with different amounts of available bandwidth, they can receive as many layers as they can handle. When the network is congested the sender can chose to send only a few layers.

The drawback with layered coding is that it increases the complexity in encoding, decoding and transmission. It might also reduce the degree of compression due to the extra information that is needed to define the different layers.

2.4 Error concealment

Error concealment techniques in the decoder can reduce the impact of a lost packet. The more advanced the technique is, the better the result will be.

A common way to conceal a lost part in an image is to use a spatially or temporally adjacent area. If a whole frame is lost, there are several ways to handle it (See *Fig. 3*). The easiest way (Alt. 1) is to skip the lost frame and pretend that it was never there. A better way (Alt. 2) might be to show the previous frame once more. The most advanced way (Alt. 3) is to keep the most recent frames in a buffer. The receiver shows a couple of frames before and after a loss longer times to cover the missing piece of the signal.

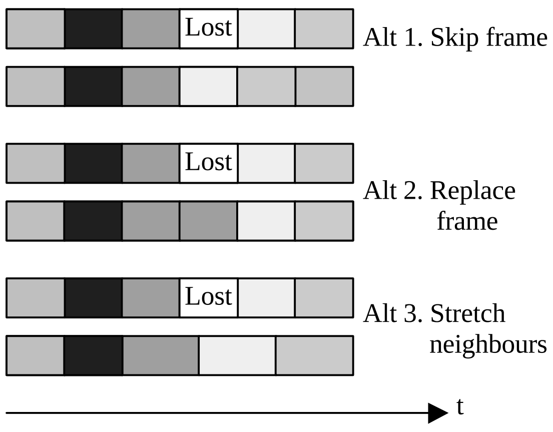


Fig. 3 Handling of frame loss

2.5 Jitter

Jitter can be explained as the difference between consecutive packets' transmission delays. Depending on how a video stream is multiplexed with other traffic, delays will appear and the jitter will thereby increase.

By smoothing the output from a server, induced jitter can be reduced. This can be done either by having a previously smoothed video file (constant bit rate CBR) or by using buffering to send a variable bit rate (VBR) coded file as CBR.

To accommodate the impact of jitter, a buffer can be used at the client side. The purpose of the buffer is to store at least the amount of data that is required to compensate for the expected jitter magnitude. A drawback with the buffer is that it introduces a delay at the receiver. The delay might be disturbing if the communication is two ways but has no particular drawback if the communication is one way.

2.6 Problems with VBR

One problem that appears when coding video is that the bitrate required to represent different frames varies heavily depending on the content. This kind of behavior is called VBR. When transmitting data on networks, it is preferable to have a constant bitrate because sudden peaks may overflow buffers and cause packet loss.

There are several techniques to handle the VBR behavior, some of them are presented under sections 2.7 Smoothing techniques and 2.8 Rate control.

2.7 Smoothing techniques

Smoothing is a way to reduce the variations in bitrate. There are several ways to smooth a videostream:

- Temporal multiplexing: using one buffer per stream to smooth the output can reduce the peak bandwidth requirements at the price of introduced jitter.

- By statistically multiplexing several independent streams together, we get a bandwidth demand that converges towards a normal distribution. This kind of smoothing introduces no delay.

- By using a work-ahead buffer[1] ([15], [17]), the variability can be reduced by introducing a delay (considering non real time content). There has to be enough buffer space in the receiver to regain the timing for the decoder.

In [15] and [17] the authors have worked on and evaluated work-ahead smoothing techniques that reduce the rate variability when transmitting stored video from a server to a client across the network. The problem they have focused on, is how to transmit a video as smooth as possible to a client with a fixed buffer without starvation or overflow. The solution to this problem is to schedule the transmission so that the variance and peak rate are minimized.

The scheduler has an algorithm that uses the frame sizes of the movie and the receiver's buffer size as inputs. The algorithm divides the movie in different segments by looking at the required bitrate for the frames. Each segment is then transmitted with a constant bitrate corresponding to the segments' average. With a VBR video as source, the peak rate and the standard deviation can be reduced by 70 to 80% using this technique (with a buffer of only 1 MB and for a video stream of 1.25 Mbps).

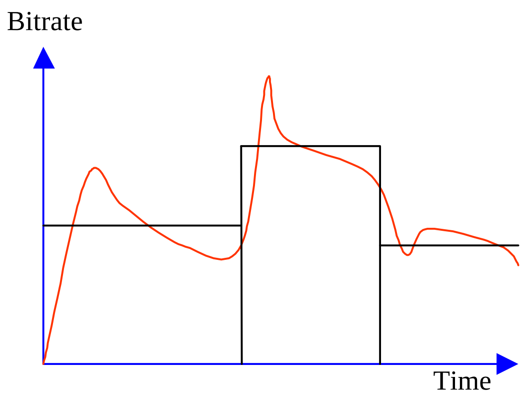


Fig. 4 Bitrate allocation over time

[1] A work-ahead buffer works like the buffer that was discussed under section 2.5 but in this case it stores much more data to accommodate variations in the movie bitrate as well as the jitter.

When one segment is being transmitted, the bandwidth is renegotiated for the next segment as early as possible (depending on the buffer capacity), in order to meet the coming demand (see *Fig. 4*). By prefetching a part of the next segment in advance, the bandwidth demands of the next segment can be reduced. A network manager keeps track of the network resources so that they are not overbooked.

The authors of [15] try to increase the bandwidth in smaller steps than the authors of [17]. The advantage is that it is more likely to get a little bit extra bandwidth from the network manager than a huge amount. The drawback with small increases is the amount of overhead. Therefore, all bandwidth negotiations are done in advance and the schedule is approved before the transmission starts.

If an increase in bandwidth is denied the sender has to dynamically decrease the quantiztation level to reduce the bitrate, choose another path, or allocate more buffer space in the client.

To cope with transmission jitter the algorithm must not empty or fill the buffer totally. A high- and low- watermark is calculated to handle early packets and the worst expected delay. If the jitter characteristics are known, it is possible to set the watermarks optimally and achieve better performance.

Network performance basically depends on peak-rate and burst characteristics. Therefore these smooth streams generated by scheduling and prefetching ought to consume less resources.

The authors of [15] have made a comparison of the use of smooth streams on two different kinds of network services, deterministic guaranteed service and renegotiated constant bitrate (RCBR).

Deterministic guaranteed service increases the performance by using temporal multiplexing. Statistical multiplexing is not allowed because of the strict quality guarantees. The advantage of deterministically guaranteed service is that the performance is guaranteed, but at the cost of added delay (due to the temporal multiplexing) and potentially low utilization.

RCBR is like constant bitrate (CBR) but with functionality for bandwidth renegotiation. RCBR does not use temporal multiplexing, performance is increased by using statistical multiplexing of network resources via a bandwidth renegotiation mechanism. The advantages with RCBR are low delay and potentially high utilization, the drawbacks are renegotiation overhead and lack of deterministic guarantees. When using RCBR together with the scheduling algorithm, capacity is reserved to handle the peak in each interval. For every new interval, the bandwidth is renegotiated.

2.8 Rate control

Bolot and Turletti (among many others) have worked on rate control mechanisms [11]. Rate control (see *Fig. 5*) is useful for two different things, to send a VBR video stream on a CBR channel and/or to adapt to congestion in the network. The main idea is to decrease the quality for the more complex scenes and increase it for the less complex ones to achieve a constant bitrate, or to adapt to congestion

There are different ways to adjust the output rate from an encoder. The rate can be reduced by decreasing the sampling rate, increasing the quantization step size, or by increasing the movement detection threshold. By changing these parameters in the opposite way the rate is increased. Common for all these techniques is that a reduction in bitrate implies a reduction in quality.

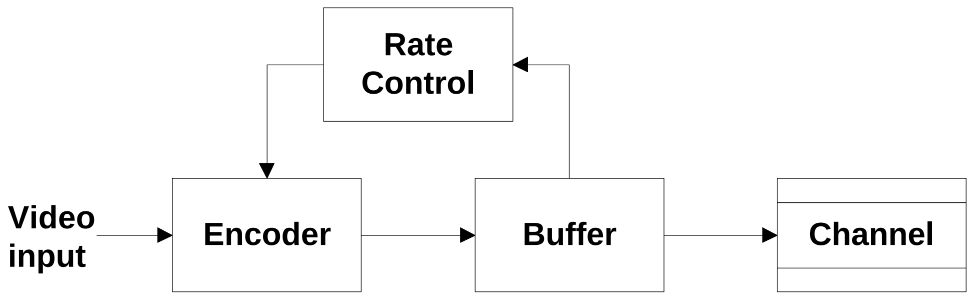


Fig. 5 Rate controler

2.9 New protocols for real-time traffic

During mid nineties and forward, several new protocols have been designed to support the transfer of real-time video.

The real-time stream transfer protocol (RSTP) offers client-server transfer functions for transmitting audio or video. The main idea with RSTP is to adopt the transfer rate to the conditions in the network. The video stream is first divided into cyclic timeslots. Depending on the situations in the network, the server and in the client, the data rate can be adjusted.

Real-time transport protocol (RTP) consists of two different protocols, the RTP that transports the payload data over UDP and the real-time transfer control protocol (RTCP), that is used to report traffic conditions from the clients to the server. Functions that are supported by RTP are loss detection for quality estimation and rate adaptation, sequencing of data, intra- and intermedia synchronization, source identification, and basic membership information.

The protocols use two different channels and the transmission is best effort. Using best effort may result in packets being lost, without retransmission (because UDP is used). All members on the RTCP-channel send regular reports about how much data they have sent (if any) and how well the data on the RTP-channel is being received. If certain amount of these messages get lost, the RTCP sees this lack of feedback as an indication of general packet-loss. When this accurs, RTCP request the sender to adapt its transmission-rate to avoid further packet-loss.

The resource reservation protocol (RSVP) makes it possible for applications to reserve resources along the path from the source to the destinations. RSVP has to be implemented in each router on the way, and in the end system. The end system reserves bandwidth and buffer space in each router passed from source to destination. In this way, a certain QoS[1] can be guaranteed. Both unicast and multicast delivery of data is supported by RSVP.

2.10 Traffic shaping

When sending stored video, it is desirable to do it smoothly and not all at once. The reasons are that:

1. Bursty transmission could congest the network.
2. A very large buffer would be needed to be able to see it.
3. In many cases, the capacity is low and it would take a lot of time to first download the video and then view it.

The solution to all these problems is to send the file in the same rate that is needed to view it. A way to accomplish this is to use traffic shaping

Shaping the traffic can save a lot of bandwidth, especially when the sources are bursty. However, this is done at the expense of increased delay, and the shaping is therefore limited by how much delay that is tolerable.

2.10.1 Leaky bucket

A common way to limit the burst size and transmission rate is to use the leaky bucket algorithm. The leaky bucket algorithm consists of two parts (see *Fig.* 6), a token pool of size N and tokens. Tokens are generated at a fixed rate of *R* tokens per seconds and are stored in the pool.

[1] QoS – Quality of service is a concept used to describe properties like throughput, delay and priority for a network service.

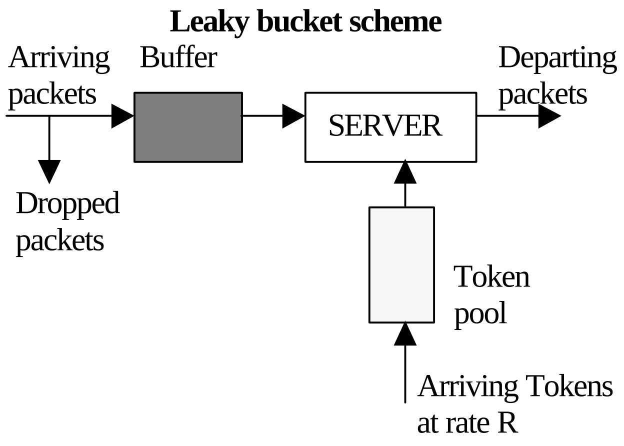


Fig. 6 Leaky bucket

The token pool cannot contain more than N tokens. When a packet departs, it consumes one token. If there are no tokens left in the pool the packet is queued. If the queue is full, the packet is lost. With a pool size of N = 1, the packet transmission rate becomes equal to R.

There are two ways to save buffer space: either to increase the pool size N or to increase the token generation rate R. But to smooth out bursts, the pool size must not be too large and it is thus better to increase R.

3 Discussion of problems related to streaming of video

This discussion addresses problems with streaming of data: jitter, delay, clock synchronization, sending rate, process scheduling and the clients' capacity.

3.1 Dealing with jitter and delay

A common problem when sending data over a medium shared with other traffic is the introduction of jitter and delay on the way from the sender to the receiver. The jitter is due to bursts of traffic from other sources and is hard to avoid. Delay is introduced at bottlenecks like congested routers and by propagation.

The solution is (as described in 2.5) to have a buffer that removes the jitter and congestion control to minimize delays.

3.2 The issue of clock synchronization

There are two issues regarding synchronization that generally cause trouble:

- Synchronization time between sender and receiver clocks.
- The variation in time due to fluctuations in the server clock frequency.

Dedicated MPEG decoder set-top boxes, connected to a TV, are very sensitive to jitter in the data flow because it limits the possibility to phase lock on to the signal. NTSC[1], for example, needs a long-term signal drift of less than 0.1 Hz/s.

The clock that a computer relies on is by no means 100% accurate, but can have deviations of hundreds of ppm.

If the deviation that might occur in the server clock frequency can be anend, and proves to be approximately constant, then it can be compensated for. The most common solution though, is to use additional hardware offering greater clock accuracy. In this case the client application extracts the correct frame rate from information in the bit stream so there is no problem with clock synchronization.

[1] NTSC – National Television Standard Committee is a standard for color television in USA and other countries.

3.3 Sending at the correct rate

To stream data at the correct rate might be a problem. If the server sends faster than the viewer can display (due to lack of computing resources or when the client's viewing frame rate differs from the server's), a buffer overrun will occur in the client. On the other hand, if the server sends to slowly, the viewer has to make pauses intermittently.

The solution to this is to keep the receiver buffer at a reasonable level. This is implemented in the following way:

If the buffer starts to get empty, the client sends a message to the server to boost the sending rate. When the buffer reaches a safer level, the client asks the server to send at normal rate again.

If the client is temporary loaded with other work or for some reason doesn't have enough computing power to display at the given rate, it replays at a lower rate and buffers the incoming data. When the buffer starts to fill, the client sends a message to the server to lower the sending rate. If the buffer reaches a critical level, the client sends a halt message to the server and requests more data first when the buffer becomes empty.

3.4 Sending smoothly

Another difficulty with video streaming is to send a smooth stream of packets i.e. small packets often rather than large chunks more rarely. This problem can be solved to large extent by using dedicated hardware that forwards the packets in an even distribution over time. But without dedicated hardware, there are limitations in granularity from the shortest time a sending process can execute – 10 ms (see further in 3.5). In reality, a process can only get a timeslot for the CPU every 20 or 30 ms.

3.5 Scheduling contention

In UNIX, several processes can run concurrently. To accomplish this, a scheduler must divide the CPU time between the processes (since only one at a time can occupy the CPU). This usually causes a competition between the processes for CPU time, which is referred to as *scheduling contention*. Commonly the scheduling has a resolution of 10 ms time slices[1] (for example on an Intel processor system). Due to this fact, it is not possible to get a very good granularity in the scheduling.

[1] A time slice is the shortest time interval that a process is granted to run uninterrupted.

There are some tricks though: if a process has finished its work before its time slice has ended, it can force a *context switch*[1] by calling sleep.

To ensure that important processes like the *kernel*[2] get the time it needs, there are different levels of priority. If a process with high priority has work to do, it will preempt a process with lower priority. When processes with similar priority are running or waiting to run, the priority of the running processes is reduced for each time slice and a context switch will eventually occur.

For each time slice, the scheduler checks which process to run next. The choice of time slice length is a balance between being able to handle external events quickly enough and not to waste all time on scheduling, since each context switch takes about 0.1 ms.

3.6 Client capacity

One of the difficulties on the receiver side is that the replay of compressed video with a rate of several Mbps needs a lot of computing power. A standard PC (200 MHZ Pentium 64 MB) can hardly handle more then 1.5 Mbps of H.263 coded video (CIF, 8 fps) and a better PC maybe 3.2 Mbps with 12.5 fps. Most of the process time is consumed to decode each frame. The required computing power increases with the frame rate at constant bit rate. If uncompressed video is received, less computing power is needed and greater bit rates can be handled. But sending uncompressed video wastes a lot of bandwidth though - 3.1 Mbps of uncompressed video has about the same quality as 400 kbps of H.263 coded video. It is useful though to see how high bit rates that the receiver can manage (tests have shown that 6.2 Mbps of uncompressed video present no problem for a standard PC).

[1] Context switch occurs when the operating system stops one process and starts running another. A context switch is often caused by to external events like data being available on a device.
[2] The kernel is the most essential part in operating systems, responsible for resource allocation, low-level hardware interfaces, security etc.

4 Methods

The video server is software based and written in portable C-code that can run on a standard PC, with Mbps performance. By not relying on hardware solutions, the product becomes cost effective, flexible and easier to modify.

The video files are accessed via a web interface. It uses header files with description of where the video is located as well as necessary data to stream and view it.

In order to view video files, there is also a receiver part needed, which buffers and decodes the incoming data. The receiver handles uncompressed video and the H.263-coding format. The server, though, is not restricted to any format.

4.1 Experimental setup

The experimental setup (see *Fig. 7*) consists of a server[1] and "The Sniffer™" traffic analyzer connected to the common network via an Ethernet hub. Two clients[2] were also connected to the common network. Since both the clients and the server were connected to a switch, they did not have to compete with other computers for the bandwidth. The server, clients and analyzer were all equipped with 10 Mbps Ethernet cards.

The traffic analyzer was used to measure the inter-arrival times for each packet from and to the server, and the traffic load.

Several trials of each test were done and representative results were selected for presentation.

[1] A 180 MHz Pentium Pro 64MB running Linux (RedHat 5.1).
[2] A 200 MHz Pentium MMX PC with 64 MB RAM, running Windows NT (4.0) and a 400 MHz Pentium II PC with 128 MB RAM, running Windows 98.

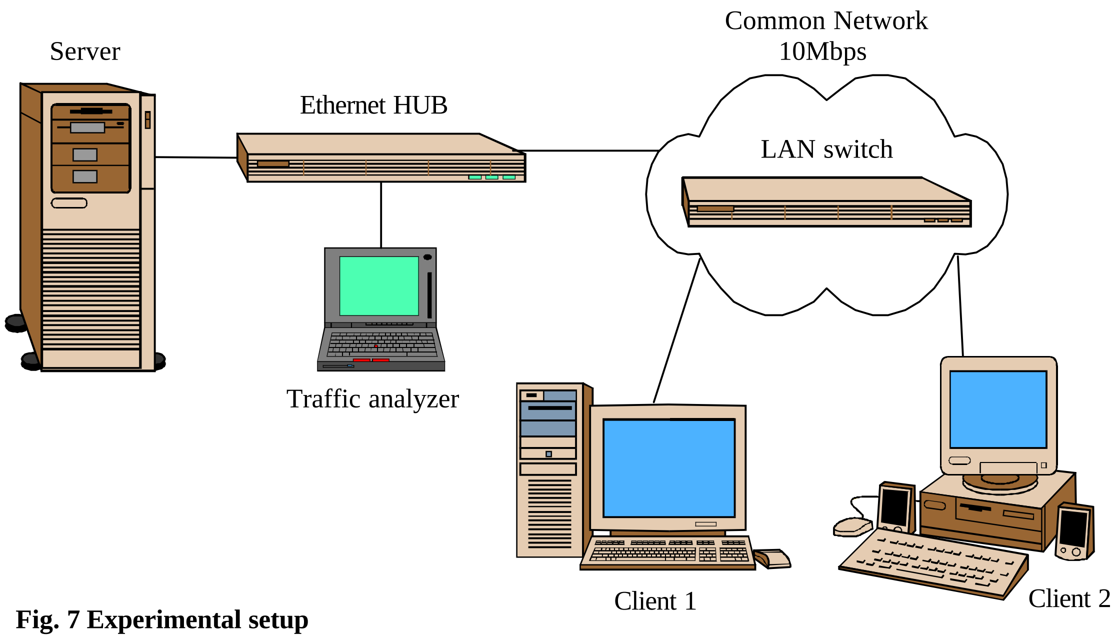


Fig. 7 Experimental setup

4.2 Video format

The video format that was used in this application is H.263 and uncompressed video. This format was chosen because:

1. H.263 is suitable for this kind of application since it can recover from loss of data.

2. Source code for a viewer was available for modification to fit the client.

3. An H.263 encoder was also available, which allowed different resolutions, frame rates and bit rates to be produced.

The CIF format (352 x 288) was chosen for the tests and frame rates of 8.3, 12.5 and 25 frames per second was coded in bitstreams 200 to 3200 kbps. The frame rates 8.3 and 12.5 might seem odd but origintes from the fact that the movie had 25 fps and when capturing every other frame, the output is 12.5 fps and for every third, the output is 25/3 which is appr. 8.3 fps.

Since only a short part of a movie was available as uncompressed video (90 sec corresponding to 450 MB), the server was set to loop through the source file.

The server is by no means restricted to send H.263 and uncompressed video but it might need modifications if there are specific needs for a particular format.

4.3 Sending with UDP

UDP is used to send the video because it is the only protocol (of TCP and UDP) that may send at a fixed rate even under high network loads and contending streams. TCP uses a mechanism called *multiplicative decrease congestion avoidance* – which means that when a packet is lost the sending rate halves. TCP would unavoidably lower its bit rate under congestion and the receiver could not view the video at the correct frame rate or would have to make pauses now and then to fill its buffer.

In shared networks, you can take advantage of TCP's back off and get a larger share of the bandwidth if you send via UDP. On the other hand, UDP gives no guarantees that all of the data will reach its destination. Therefore the viewer has a loss–recovery mechanism to make use of the data that comes through. H.263 have such a mechanism.

One might think that TCP could be used in conjunction with a receiver buffer to handle the decrease in rate under congestion. But when sending at high bit rates, it is hard to compensate longer fluctuations because it would require a huge buffer. Therefore, almost every video streaming application uses UDP.

4.4 Rate control mechanism

The rate control mechanism that handles the rate and shape of the streaming is located both in the client and in the server.

4.4.1 Server side

When the rate control mechanism gets a new job it assumes a timeslot every 20ms. Furthermore it gets the MTU as input, either as a parameter when the server is started or from a predefined value: 1500 bytes is the standard size MTU used on most networks which leaves (1500 -20 for IP header -14 for UDP header) 1476 bytes available for payload.

The next step is to calculate how many packets that need to be sent during each timeslot. If the timeslots occurs every 20 ms and rate is 409600 Bps and the available payload is 1476 byte we end up with (409600 * 0.02 / 1476) = 5.55. Since it is not possible to send 5.55 packets, the algorithm decides to send 6 packets per timeslot with a payload size of (1476 * 5.55 / 6 =) 1365 bytes. The alternative approach would be to send a different amount of packets each timeslot.

The scheduling works such that a process starts at a certain time and sends the amount of packets it is supposed to (6 in the example above) and calls sleep when done. During the time when the process is inactive, other processes can execute.

If the 20 ms assumption proves to be wrong or if the CPU is busy with other processes during some of the timeslots, actions has to be taken. As a reference, the process keeps track of how much data has been sent as well as for how long it has been sending, and consequently how much data that should have been sent. The *getttimeofday*[1] function is used to get an instantaneous estimation of how much data that should have been sent for the moment.

If the two amounts of data differs by more than 0.1 s times the bitrate then more or less packets are sent in each of following timeslot until the problem is corrected. To avoid conflicts with the other sending processes these corrections are averaged out over time.

One might think that in a system with several concurrent sending processes there should be a variation in how often a process gets a slot. Thorough tests have shown though that this is not the case. Each process gets to run almost every 20 ms. One idea of where the 20 ms originates from is that it might have been inherited from older days when the computers used the power frequency to control the timing.

[1] Gettimeofday is a function that returns the current time of the system clock with microsecond resolution.

4.4.2 Client side

The client is involved in the rate control in the way that it forwards requests about the bit rate to the server.

There are four states in which the server can stream data to the client:

Normal	The server sends at the same rate as the coded bit rate.
Slow	The server sends slower – set to 70% of normal rate.
Boost	The server sends faster – set to 120% of normal rate.
Halt	The server halts and waits for a new command from the client.

The following rules are set up to handle the client buffer:

If	more than	85 % in buffer		Send Halt
If	more than	30 % in buffer	And current state halt	Wait
If	less than	30 % in buffer	And current state halt	Send Slow
If	more than	75 % in buffer		Send Slow
If	more than	2 s of video in b.	And current state boost	Send Normal
If	less than	1 s of video in b.		Send Boost
If	less than	1.25 s of video in b.	And current state slow	Send Normal

The buffer size is set to 2 MB and consequently it can handle different amounts of video (max 5 s for a 3.2 Mbps stream and up to 80 s for a 200 kbps stream). The aim is to have at least 1s of content available in the buffer and still space to store an incoming burst of data.

5 Presentation and discussion of results

5.1 The application

The video streamer application consists of two parts: the server and the client.

The server part handles requests from several clients and streams the requested data at the same rate as the video was coded for.

On the client side, an application has been written to work as an interface between the server and a viewer. It sends requests to the server, receives data, buffer and forward the incoming data to a viewer.

The server runs in a UNIX environment (in this case Linux) and is programmed in C to make it portable and platform independent. The client (also written in C) is built for personal computers running Microsoft Windows ™.

The user interface is a web page where the user clicks on the video stream that he or she wants to access. The link on the web page points to a header file that contains necessary information to establish a connection to the video stream server.

See figure 8 for an overview of the system.

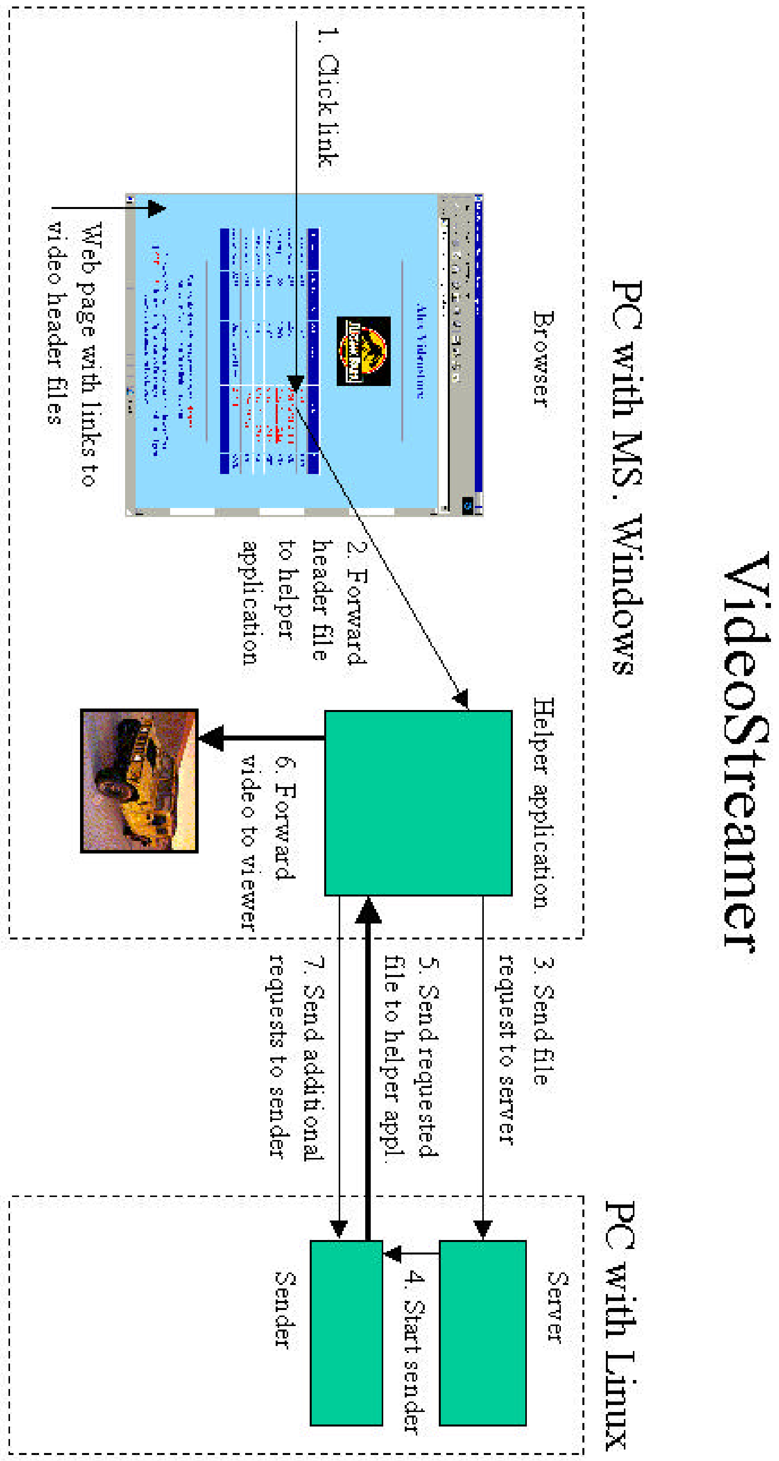

VideoStreamer
PC with MS. Windows
Browser
1. Click link
Web page with links to video header files
2. Forward header file to helper application
Helper application
6. Forward video to viewer
3. Send file request to server
5. Send requested file to helper appl.
7. Send additional requests to sender
PC with Linux
Server
4. Start sender
Sender

Fig.6 Overview

Fig. 8 Overview

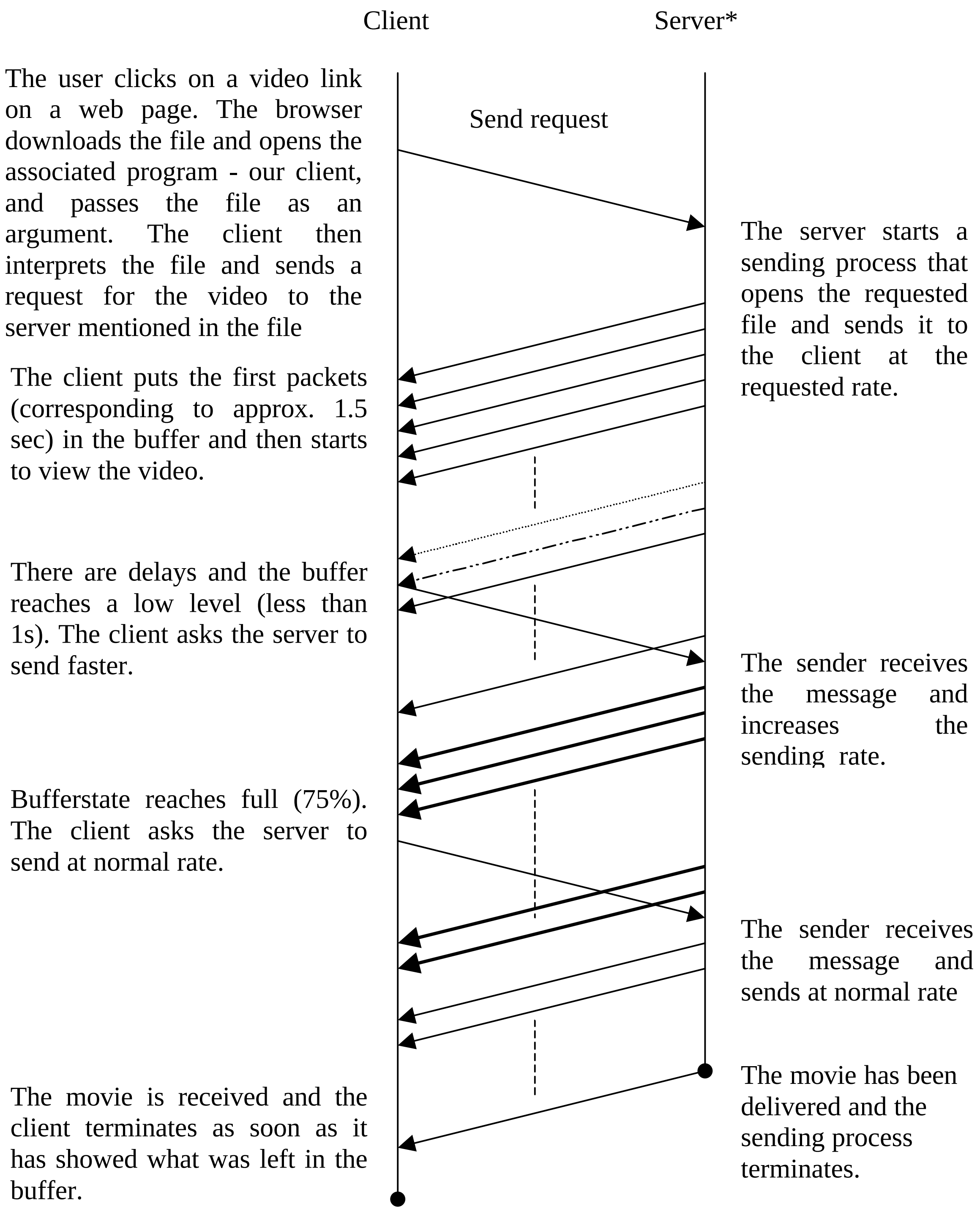
5.2 How the receiver/client part works

When a user clicks on a link to a video header file on the web server, the browser retrieves that file. The file extension of the header file is associated with the receiver program, which is started and gets the downloaded header filename as input. The header file contains information about the server that has the file: the actual filename, the video format, the coded bit rate of the file etc. The client program then allocates a local *port*[1], and sends a request to the server address and port indicated in the header file. The request consists of the video filename and the bitrate. The client then waits for the server to begin streaming the requested file on the same local port as it sent the request on. The client then waits for about 1.5s before it starts to display the video. This is done to fill the buffer with at least 1s of video to handle jitter and delay.

A feature with the displayer in the client is that it is possible to expand the visible image fourfold. This feature adds 25 to 40% to the CPU load that the client already causes.

See scenario (*Fig. 9*) for the course of events.

[1] The TCP/IP protocols use ports (also referred to as sockets) to distinguish among multiple destinations on a computer with a certain address. TCP/IP identify the ports by using small integer numbers (usually 1-65536). An application is almost free to chose the port number it wants to be associated with, but some are reserved for common services like email, ftp and telnet.


Client
Server*
The user clicks on a video link on a web page. The browser downloads the file and opens the associated program - our client, and passes the file as an argument. The client then interprets the file and sends a request for the video to the server mentioned in the file
Send request
The server starts a sending process that opens the requested file and sends it to the client at the requested rate.
The client puts the first packets (corresponding to approx. 1.5 sec) in the buffer and then starts to view the video.
There are delays and the buffer reaches a low level (less than 1s). The client asks the server to send faster.
The sender receives the message and increases the sending rate.
Bufferstate reaches full (75%). The client asks the server to send at normal rate.
The sender receives the message and sends at normal rate
The movie has been delivered and the sending process terminates.
The movie is received and the client terminates as soon as it has showed what was left in the buffer.

* When the server receives a request, it starts a new process that handles the client. Then it continues to listen for new clients. In this figure the server line represents the server receiving the request and the sender doing the rest of the actions.

Fig. 9 Scenario of a video stream session

The receiver has an independent *thread*[1] that receives the incoming data and puts it into a buffer. The receiver thread runs at high priority to minimize the possibility that packets are lost if the socket's receive buffer gets full. If the buffer is becoming full, a message is sent to slow down the sending rate, and if the buffer state reaches a critical level, a halt message is sent. As soon as there is enough space in the buffer a "start send" message is issued to the server.

If the buffer tends to empty, the client sends a request to get a higher rate from the server.

5.3 How the sender/server part works

The server waits for requests on a certain port that is specified at startup (see *Fig. 10*). When a request arrives, the server starts a new process that handles the request while the other process waits for further requests. The new process interprets the request and starts sending the requested file at the requested rate to the client that sent the request. The address and port issued request identify the client.

If the receiver/client stops to receive packets (the user quits the application for example), an error condition[2] occurs and the sending process stops to send and terminates. The sending process also listens for messages from the client. Messages could be to slow down, halt, increase the rate etc.

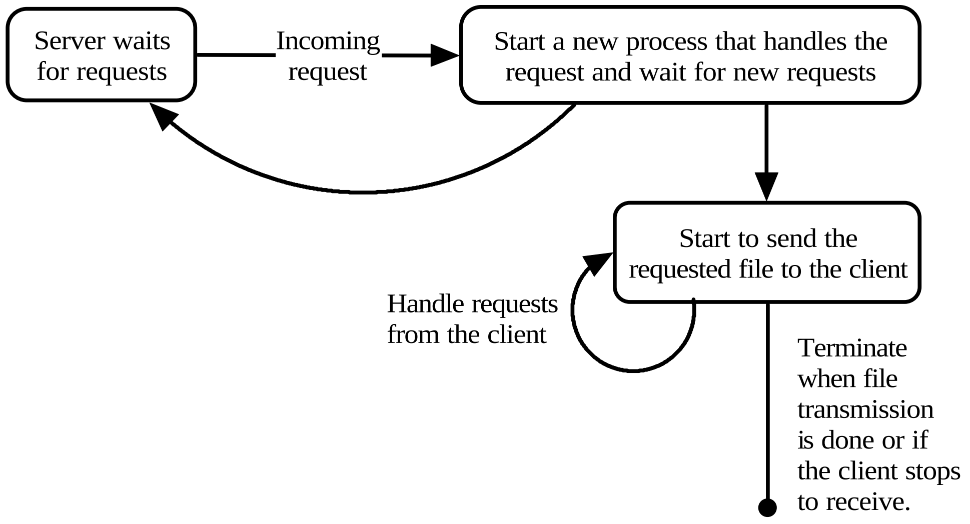

Server waits for requests
Incoming request
Start a new process that handles the request and wait for new requests
Start to send the requested file to the client
Handle requests from the client
Terminate when file transmission is done or if the client stops to receive.

Fig. 10 Server overview

[1] Threads are in large extent similar to running several processes on a single CPU. The main idea is that you want to let several processes run concurrently.
[2] When the client stops to receive packets, the client computer sends an ICMP (Internet control management protocol) message to the server computer that asks it to stop sending. As a consequence of this message the server computer shuts down the port it sends on. Then the sending process will receive a "port unreachable" error when it tries to send and quits streaming.

5.4 Performance

The implementation of the video server offers a time granularity of 20 ms per stream, but with three concurrent streams for example, two can run on even and one on odd tenths of milliseconds (see *Fig. 11*). Hence a 3.15 Mbps stream would send 8 kB of data every 20 ms distributed over 8 UDP packets of 1 kB.

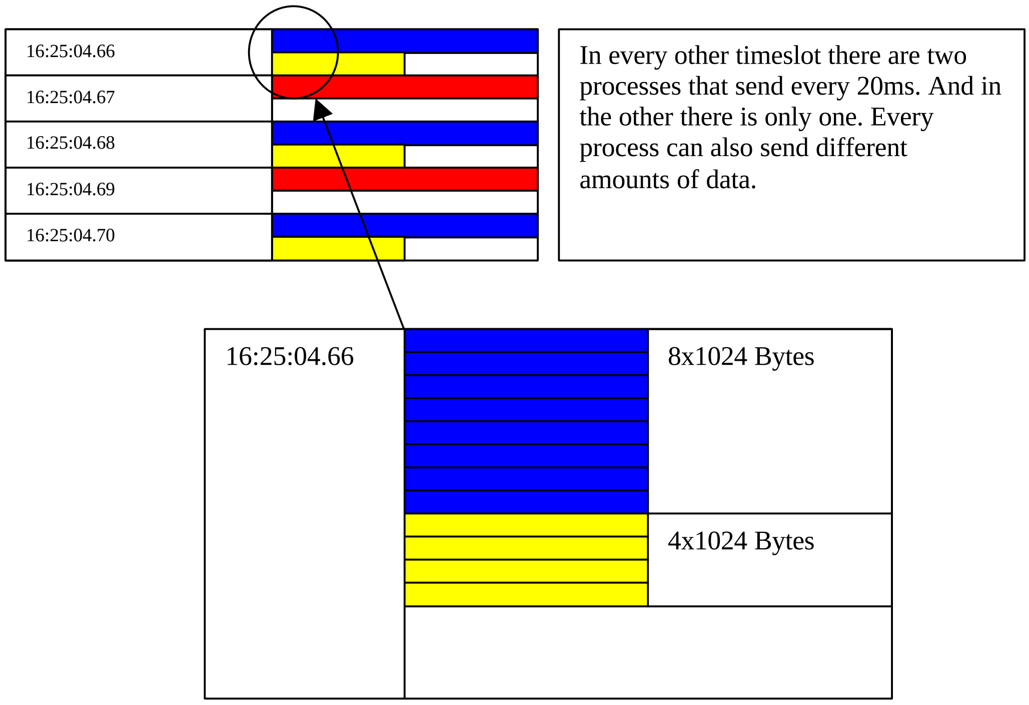

16:25:04.66
16:25:04.67
16:25:04.68
16:25:04.69
16:25:04.70
In every other timeslot there are two processes that send every 20ms. And in the other there is only one. Every process can also send different amounts of data.
16:25:04.66
8x1024 Bytes
4x1024 Bytes

Fig. 11 Example of the scheduling of two 3.15Mbps and one 1.6Mbps stream

See Appendix A for the values from a measurement of a run with two 3.15 Mbps and one 1.6 Mbps stream

The server (running on an average PC – Pentium Pro 180 MHz, 64 MB) is capable of handling up to 10 Mbps (due to network configuration constrains using 10 Mbps Ethernet) without any degradation in performance. Tests have been done with three concurrent streams of 3.15 Mbps each.

The server CPU utilization when running this test was less than 5% why one can say that CPU power is not a concern. One might be tempted to use the free capacity for something else, but then, the sending performance might be reduced severely because the sending processes would not get their sending time-slot as frequently as needed.

What is more important than a fast CPU to achieve high performance is a disk drive that is capable of delivering several concurrent streams under a prolonged time. In this case a simple IDE-drive[1] has been used, but if one wants to serve many clients, ultra-SCSI2[2] drives are preferred.

If one is to provide a video on demand server, many disk drives are required to store the movies. Hence the load on each disk drive would be less than in the single case (if not all customers want to see the same movie at the same time).

What is also very important, is that the video files is not fragmented on the disk. As files are created and deleted in the file system, the free space becomes split into smaller non-contiguous blocks. When a large file is saved on a system where the free space is split into several small blocks, the file cannot be stored in one piece, but becomes scattered across the file system. Later, when the file is accessed, the file system has to seek for the parts to retrieve the data. The seek time degrades the performance.

The solution to fragmented files is to use defragmentation software that rebuild the file structure and consolidates the fragmented files into contiguous blocks.

[1] IDE – Integrated Drive Electronics, is an interface used to connect internal storage devices to the computer's bus. One IDE interface can handle two devices, one of them referred to as Master runs with priority and the other is referred to as Slave.

[2] SCSI – Small Computer System Interface, is an interface used to connect devices like hard-drives, CD-ROM, printers, scanners etc. to one single adapter card on the computer's bus. Ultra-SCSI2 is an extension of SCSI that allows a higher throughput rate (40 MBps compared to 5 MBps for SCSI). Besides the ability to connect up to seven accessories in a chain – both internal and external, SCSI devices also consume less CPU-time compared to IDE-devices.

6 Measurements under load

In order to get a better view of the client-sever capabilities in a network with heavy background traffic, a new experimental setup was established. Although several measurements series have been carried out, only one will be presented here because the behavior was quite similar.

In the new setup, a traffic simulation application - Chariot from Ganymede, was used to generate background traffic. A network monitor and analyzer, NetVCR from Niksun was used instead of the previous monitor “The Sniffer”. To get a better understanding of the new setup the NetVCR monitor and the Chariot software is discussed briefly.

6.1 NetVCR network monitor

The NetVCR network monitor is an instrument developed by Niksun Inc. for passive monitoring, analysis and management. The fact that the instrument is passive means that it does not affect the network that it measures in any way. When a measurement is carried out, the instrument is connected to a point in the network where the interesting traffic passes. A typical choice of point is an aggregation point like a hub, switch or router. In the switch and router case the instrument is connected to a port that mirrors the traffic on the other ports to be able to see all the traffic.

The NetVCR monitor is based on a portable high performance PC (see Fig. 12) running FreeBSD UNIX. As standard configuration the monitor has two fast Ethernet network interfaces for measurement and one interface for remote control of the device.

At the beginning of a measurement, all data (the complete packets) are stored in the monitor, but when the storage space becomes full, the monitor discards the complete packets of the oldest data and revert to storing only the statistics about the packets.

To make it easy to access the instrument remotely from virtually any platform, the graphical user interface is built up as a web page and the instrument runs a web server to provide the content. Since the instrument stores packets from networks that it monitors, a firewall is built in to prevent unwanted access of the data. The most secure way though to operate the system is to disconnect the remote control interface.

Fig. 12 NetVCR network monitor

The statistics that the monitor collects can be presented in many ways. First of all the time interval of interest is entered (monitoring can occur for several weeks or months). Next the monitor shows the link level statistics – how much of the traffic that is represented by each link protocol (IP, ARP etc) as well as a graph of bit and packet rate over time. Another list shows the destination and source of the traffic. At this stage it is possible to follow the tree like structure of the statistics in many ways by clicking on a certain protocol or host.
Example of a sequence might be: IP > UDP > host 131.115.159.214 which will display all the UDP traffic from host 131.115.159.214 under the chosen time interval in a two graphs – bits vs. time and packets vs. time.
Logical expressions can be entered to do more advanced filter functions.

6.2 Chariot

Chariot is a traffic simulation application from Ganymede Software Inc. It consists of a console application and several node applications. The application is used to test network equipment and software by generating real traffic between its nodes. Each node runs on its own computer and traffic is generated between pairs of nodes or by multicast. All configurations are done from the console application, which also collects statistics about the simulation. Chariot uses scripts to describe each traffic type like telnet, http text and NetMeeting.

6.3 Measurement setup

First a traffic mix was set up in Chariot consisting of:

Traffic type:	**Protocol:**
Http text transfer	TCP
Http gif transfer	TCP
File transfer	TCP
NetMeeting	UDP
POP3	TCP
Telnet	TCP

The test network (see Fig. 13) consisted of five computers and a monitor connected to each other via a hub.

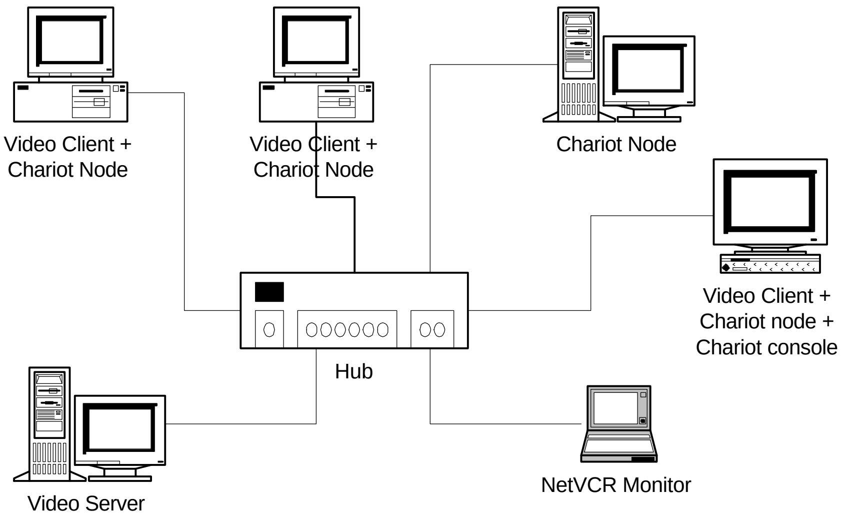


Fig. 13 Measurement test network

Three of the computers (running windows) acted as clients for the video and all computers except the server acted as nodes for Chariot. The test sequence begun by first starting the background load from the Chariot console, then five clients were started, one at a time at the 3 windows computers. The background load was about 8 Mbps (see *Fig. 14*) before the clients started to retrieve its data. Of the 8 Mbps, the file transfer represented the largest part.

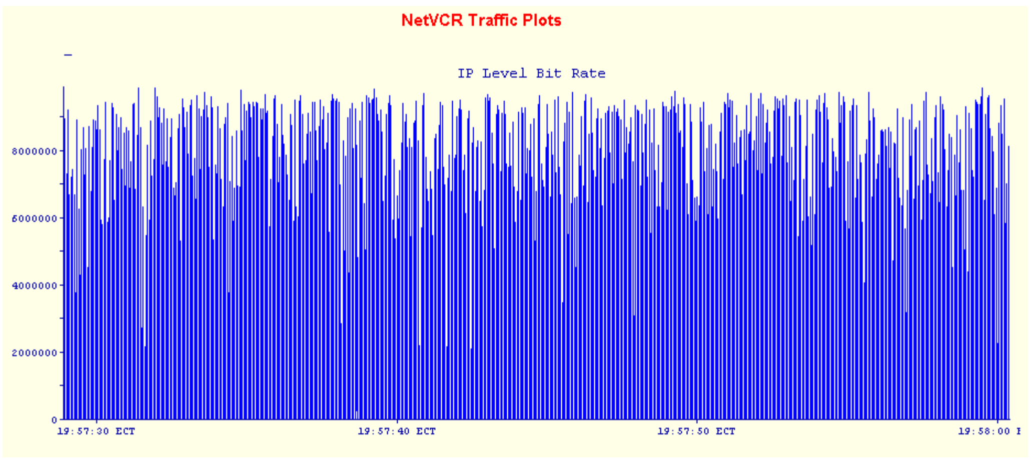


Fig. 14 Measurement of Chariot output (bits / time)

When the streaming of video started the background load went down to an average of 2.1 Mbps according to Chariots own analysis with peaks above 8 Mbps (see Fig. 15).

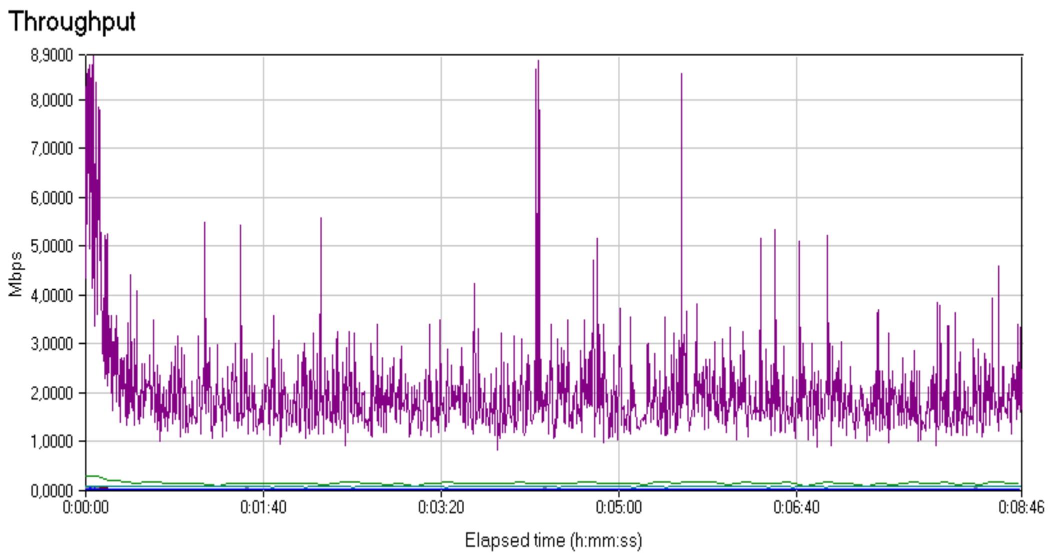


Fig. 15 Chariot throughput

The five data streams were divided so that the computer running the Chariot client received 1.6 Mbps while the other two received 1.6 Mbps + 400 kbps and 1.6 Mbps + 200 kbps. In (Fig. 16) the total amount of data is represented by the top line while the bars represent the video data from the server. The top picture shows bytes / time and the bottom picture shows packets / time.

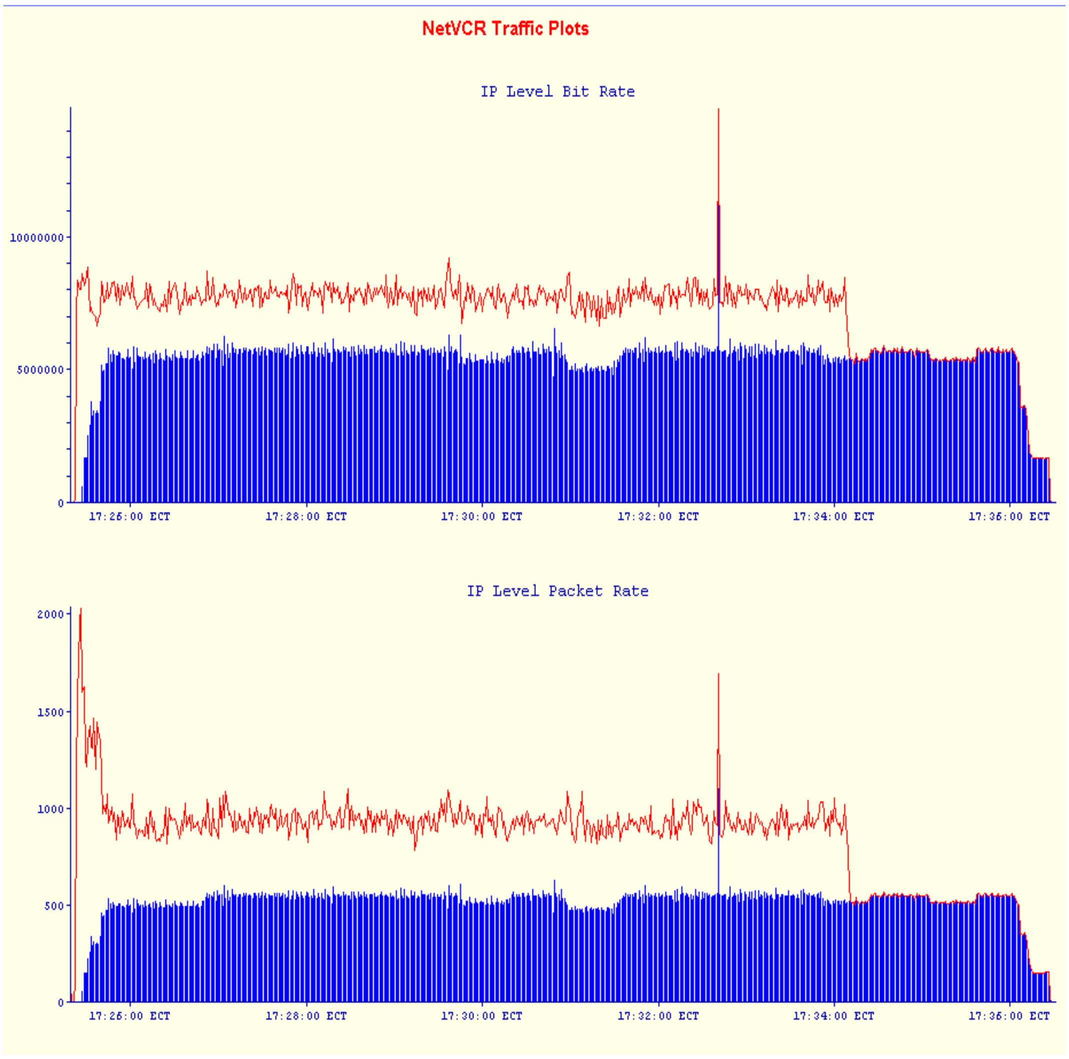


Fig. 16 Measurement of Chariot and video traffic

As said earlier the measurement started with only the Chariot traffic and the bottom graph in the picture above shows that the Chariot traffic is pushed back. Since most of the Chariot generated traffic consists of TCP, it lowers its sending rate when it meets the UDP video traffic. Had the Chariot traffic mix contained more UDP traffic the video streams would have faced more resistance. During the end of the measurement, the Chariot traffic was turned of and consequently the top line that represents video and Chariot traffic goes down.

The high peak (15 Mbps) two thirds away in the diagram was rejected as a flaw in the instruments plotting routine. It disappeared when the peak area was enlarged.

The longer fluctuations in the diagrams above are due to decreases in the clients receive rate. As mentioned earlier the client asks the server to slow down when it can’t display fast enough. One explanation as to why the clients lay behind might be that the Chariot program and the traffic it produces also loads the client CPU.

In the end of the graph the clients are turned of one at a time.

In Fig. 17 each client’s traffic is plotted and compared to the total amount of video traffic.

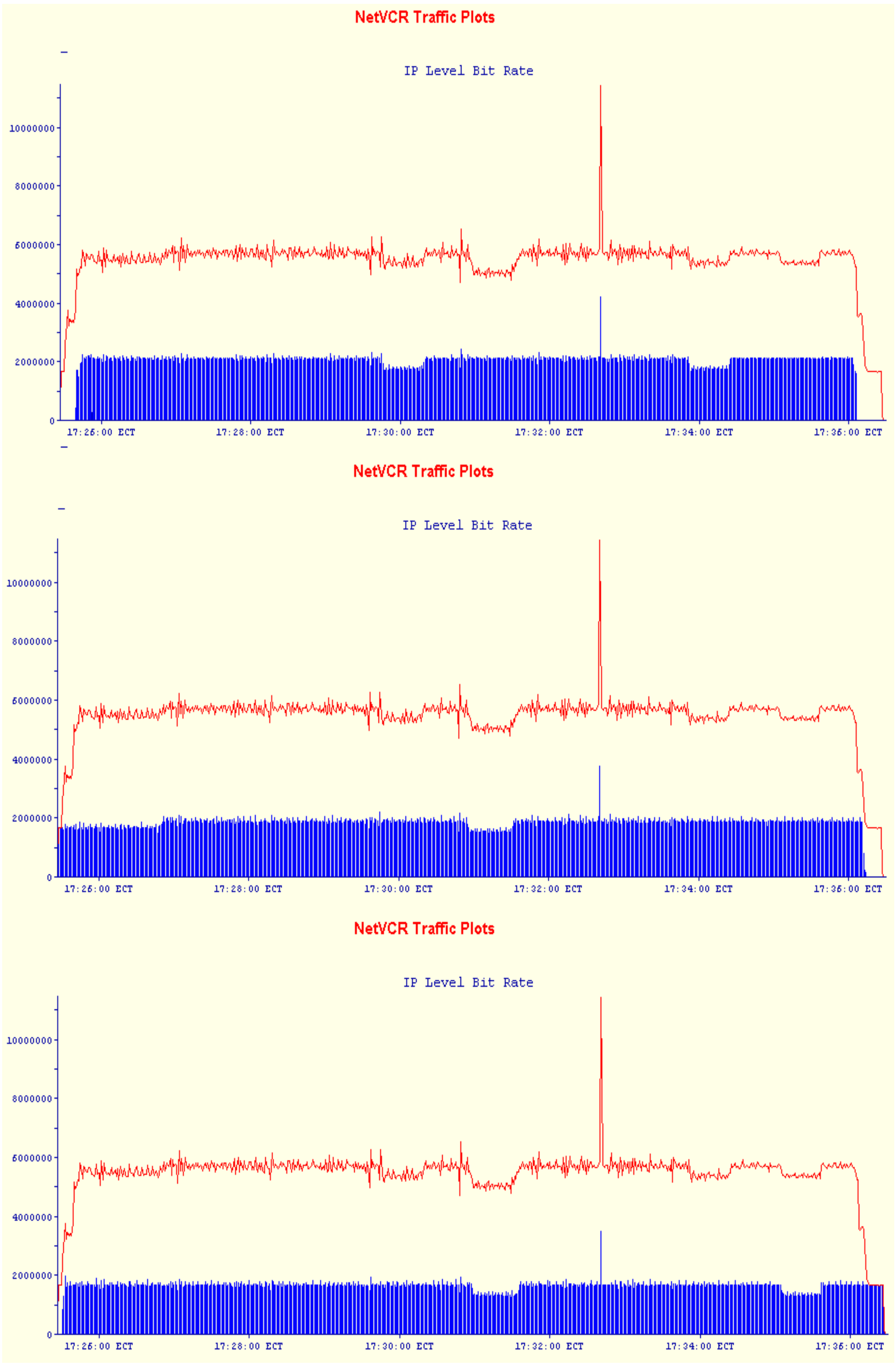


Fig. 17 Client traffic compared to all clients’ traffic

With NetVCR it is possible to zoom in time and Fig. 18 shows a close-up of the packets to the five clients. The six bars at about 1400 bytes are to two 1.6 Mbps clients, the two long bars are also to a 1.6 Mbps that receives at reduced rate. The bar at 1050 bytes is to the 400 kbps client and the shortest bar is to the 200 kbps client.

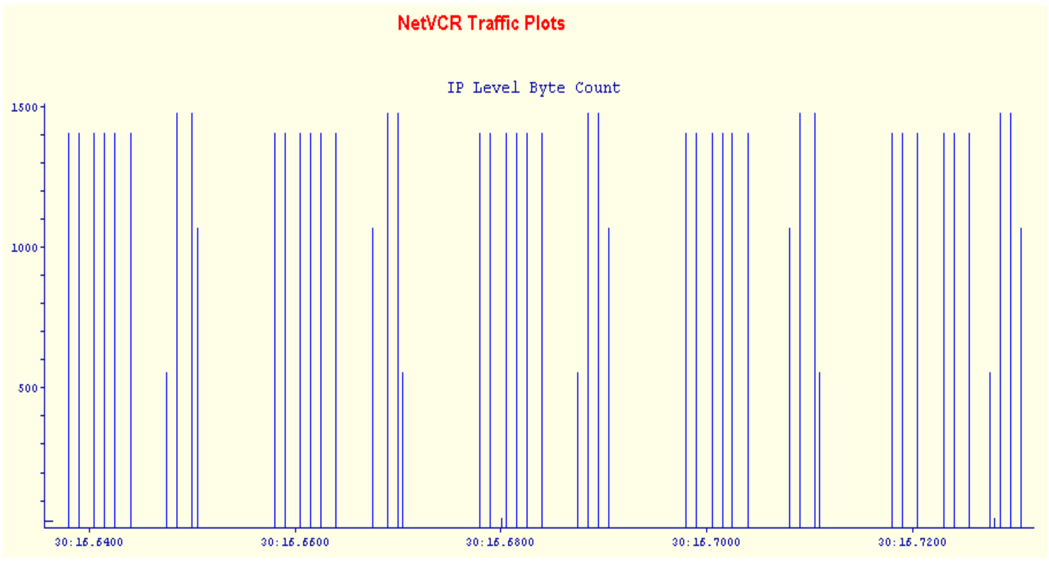


Fig. 18 Close-up of streams

The data corresponding to the bars in Fig. 18 above is represented in the table below:

Time	Source addr. Port nr		Dest. addr. Port nr	Prot	Payload
17:30:16.638258	131.115.159.214.1296	>	131.115.157.185.4858:	udp	1365
17:30:16.639407	131.115.159.214.1296	>	131.115.157.185.4858:	udp	1365
17:30:16.640555	131.115.159.214.1296	>	131.115.157.185.4858:	udp	1365
17:30:16.641703	131.115.159.214.1297	>	131.115.158.198.2858:	udp	1365
17:30:16.642851	131.115.159.214.1297	>	131.115.158.198.2858:	udp	1365
17:30:16.644000	131.115.159.214.1297	>	131.115.158.198.2858:	udp	1365
17:30:16.647675	131.115.159.214.1300	>	131.115.157.185.4865:	udp	512
17:30:16.648878	131.115.159.214.1298	>	131.115.157.165.1989:	udp	1433
17:30:16.650081	131.115.159.214.1298	>	131.115.157.165.1989:	udp	1433
17:30:16.650956	131.115.159.214.1299	>	131.115.157.165.1990:	udp	1024
17:30:16.658265	131.115.159.214.1297	>	131.115.158.198.2858:	udp	1365
17:30:16.659413	131.115.159.214.1297	>	131.115.158.198.2858:	udp	1365
17:30:16.660561	131.115.159.214.1297	>	131.115.158.198.2858:	udp	1365
17:30:16.661710	131.115.159.214.1296	>	131.115.157.185.4858:	udp	1365
17:30:16.662858	131.115.159.214.1296	>	131.115.157.185.4858:	udp	1365
17:30:16.664006	131.115.159.214.1296	>	131.115.157.185.4858:	udp	1365
17:30:16.667989	131.115.159.214.1299	>	131.115.157.165.1990:	udp	1024
17:30:16.669192	131.115.159.214.1298	>	131.115.157.165.1989:	udp	1433
17:30:16.670395	131.115.159.214.1298	>	131.115.157.165.1989:	udp	1433
17:30:16.670861	131.115.159.214.1300	>	131.115.157.185.4865:	udp	512
17:30:16.678258	131.115.159.214.1296	>	131.115.157.185.4858:	udp	1365
17:30:16.679406	131.115.159.214.1296	>	131.115.157.185.4858:	udp	1365
17:30:16.680554	131.115.159.214.1296	>	131.115.157.185.4858:	udp	1365
17:30:16.681703	131.115.159.214.1297	>	131.115.158.198.2858:	udp	1365

17:30:16.682851	131.115.159.214.1297	>	131.115.158.198.2858:	udp	1365
17:30:16.684000	131.115.159.214.1297	>	131.115.158.198.2858:	udp	1365
17:30:16.687567	131.115.159.214.1300	>	131.115.157.185.4865:	udp	512
17:30:16.688769	131.115.159.214.1298	>	131.115.157.165.1989:	udp	1433
17:30:16.689972	131.115.159.214.1298	>	131.115.157.165.1989:	udp	1433
17:30:16.690847	131.115.159.214.1299	>	131.115.157.165.1990:	udp	1024
17:30:16.698274	131.115.159.214.1297	>	131.115.158.198.2858:	udp	1365
17:30:16.699422	131.115.159.214.1297	>	131.115.158.198.2858:	udp	1365
17:30:16.700570	131.115.159.214.1297	>	131.115.158.198.2858:	udp	1365
17:30:16.701718	131.115.159.214.1296	>	131.115.157.185.4858:	udp	1365
17:30:16.702867	131.115.159.214.1296	>	131.115.157.185.4858:	udp	1365
17:30:16.704015	131.115.159.214.1296	>	131.115.157.185.4858:	udp	1365
17:30:16.708025	131.115.159.214.1299	>	131.115.157.165.1990:	udp	1024
17:30:16.709227	131.115.159.214.1298	>	131.115.157.165.1989:	udp	1433
17:30:16.710566	131.115.159.214.1298	>	131.115.157.165.1989:	udp	1433
17:30:16.711144	131.115.159.214.1300	>	131.115.157.185.4865:	udp	512
17:30:16.718317	131.115.159.214.1296	>	131.115.157.185.4858:	udp	1365
17:30:16.719466	131.115.159.214.1296	>	131.115.157.185.4858:	udp	1365
17:30:16.720613	131.115.159.214.1296	>	131.115.157.185.4858:	udp	1365
17:30:16.723125	131.115.159.214.1297	>	131.115.158.198.2858:	udp	1365
17:30:16.724405	131.115.159.214.1297	>	131.115.158.198.2858:	udp	1365
17:30:16.725667	131.115.159.214.1297	>	131.115.158.198.2858:	udp	1365
17:30:16.727570	131.115.159.214.1300	>	131.115.157.185.4865:	udp	512
17:30:16.728772	131.115.159.214.1298	>	131.115.157.165.1989:	udp	1433
17:30:16.729975	131.115.159.214.1298	>	131.115.157.165.1989:	udp	1433
17:30:16.730856	131.115.159.214.1299	>	131.115.157.165.1990:	udp	1024

In the destination column above, each client computer is identified by its address and the port numbers identify each client.

6.4 Comments

The hard part of this measurement has been to quantify how well the video streaming works. On average, each client had a couple of failures each minute due to lost data. In approximately half of these failures, it resulted in a visible flaw – minor part (about 16x16 pixels) of the picture in wrong color for example. Depending on the application this might be acceptable or not. Some of the clients had to request a temporary increase in the data rate (2-3 times) to cover the lost data during a 10-minute period.

Although this experimental setup tried to simulate a real network, it is a cumbersome goal to fulfill. What is missing in this test is the delay that occurs in routers on a large network. With high delays the regulating algorithm in the server and client would be facing a tougher job.

In the beginning of the tests, the Chariot traffic and the video traffic accidentally mixed with the traffic on the Telia Research LAN. This proved to cause a constantly lit collision LED on the hub and more losses than the client’s decoder could correct - the image quality became really poor. But then again this is not the typical situation in a common network.

7 Conclusions

The server and client that was built in this thesis shows that it is possible to deliver 10 Mbps of data shared by several streams from a standard PC. The system has proved to work well with background traffic.

The data for each stream is delivered every 20 ms. Due to the 10 ms constrains in the scheduling, it is difficult to get a much better solution with several streams without using additional hardware.

7.1 Future improvements and additions

In order to get a better shape of the traffic coming out of the server, the data could be better interleaved. Today, every process sends the packets that are needed in its timeslot. With a single process that handles the transmission of all the streams, you could output one packet at a time from each stream in contrast to the current concept where 8 packets are sent in a row from each stream. This new sending mechanism would also put less pressure on the receiver, which then gets packets in a more time-distributed way. There has not been enough time to implement this though, but it is an interesting idea to continue with.

By using a viewer that utilizes the computing power in today’s graphic cards and the abilities in the processors multimedia instructions i.e. MMX[1] in Intel and 3DNOW[2] in AMD processors, higher frame rates could be handled with a common PC.

Another solution is to use a hardware decoder. An MPEG2 decoder card for example would let a common PC decode and view MPEG2 coded video at full rate. Additional hardware would bring higher costs for the client though and the concept would become more hardware dependent.

An interesting project would be to make the server recognize different video formats and adopt the sending rate according to this instead of relying on the info in the header file.

It would also be interesting to test the server in a 100 Mbps or Gigabit Ethernet environment network to get a better view of its capabilities.

[1] MMX is an acronym for Matrix Math eXtensions is a set of 57 extra instructions in Intel’s newer processor families. The extra instructions are used to perform parallel operations on a matrix of data. This makes it possible to speed up the process of coding/decoding video.
[2] 3DNOW is a similar approach from AMD which is implemented in their processor families.

8 References

[1] T. C. Kwok: Residential Broadband Internet Services and Applications Requirements. IEEE Communications Magazine, June 1997.

[2] A. Banerjea: On the Use of Dispersity Routing for Fault Tolerant Real-time Channels. European Transactions on Telecommunications, Vol. 8, No. 4, July - August 1997.

[3] C. Perkins, J. Crowcroft: Real-time Audio and Video Transmissions of IEEE GLOBECOM ’96 over the Internet. IEEE Communications Magazine, April 1997.

[4] T.-H. Wu, I. Korpeoglu, B.-C. Cheng: Distributed Interactive Video System Design and Analysis. IEEE Communications Magazine, March 1997.

[5] G. Karlsson: Asynchronous Transfer of Video. IEEE Communications Magazine, August 1996.

[6] N. E. Andersen, P. M. N. Nordeste, A. M. O. Duarte, H. E. Lassen, A. Ekblad, A. R. Pach, K. Amborski, L. Dittmann: Broadbandloop: A Full-Service Access for Residential and Small Business Users. IEEE Communications Magazine, December 1997.

[7] D. J. Wright: Assessment of Alternative Transport Options for Video Distribution and Retrieval over ATM in Residential Broadband. IEEE Communications Magazine, December 1997.

[8] S. Kalyanaraman, R. Jain, S. Fahmy, R. Goyal: Performances and Buffering Requirements of Internet Protocols over ATM, ABR and UBR Services. IEEE Communications Magazine, June 1998.

[9] T.-H. Lee, K.-C. Lai: Characterization of Delay-Sensitive Traffic. IEEE/ACM Transactions on Networking, Vol. 6, No. 4, August 1998.

[10] G. Chirivolu, R. Sankar, N. Ranganathan: Addaptive VBR Video Traffic Management for Higher Utilization of ATM Networks. ACM Sigcomm Computer Communication Review. Vol. 28, Nr. 3, July 1998.

[11] J-C. Bolot, T. Turletti: Experience with Control Mechanisms for Packet Video in the Internet. ACM Sigcomm Computer Communication Review. Vol. 28, Nr. 1, January 1998.

[12] S. Gringeri, B. Khasnabish, A. Lewis, K. Shuaib, R. Egorov, B. Basch: Transmission of MPEG-2 Video Streams over ATM. IEEE Multimedia, January-March 1998.

[13] C. A. Fulton, S.-q. Li: Delay Jitter First-Order and Second-Order Statistical Functions of General Traffic on High-Speed Multimedia

Networks. IEEE/ACM Transactions on Networking, Vol. 6, No. 2, April 1998.

[14] R. Gopalakrishnan, G. M. Parulkar: Efficient User-Space Protocol Implements with QoS Guarantees Using Real-time Upcalls. IEEE/ACM Transactions on Networking, Vol. 6, No. 4, August 1998.

[15] J. D. Salehi, Z.-L Zhang, J Kurose, D. Towsley: Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements Through Optimal Smoothing. IEEE/ACM Transactions on Networking, Vol. 6, No. 4, August 1998.

[16] E. C. Lin: The Effects of Scheduling Contention on Workstation Traffic. Licentiate Thesis, Department of Teleinformatics, Royal Institute of Technology, 1996.

[17] W.-C. Feng, S. Sechrest: Critical bandwidth allocation for the delivery of compressed video. Computer Communications, Vol. 18. No. 10, October 1995.

[18] H. Jinzenji, K. Hagishima: Real-time Audio and Video Broadcasting of IEEE GLOBECOM ’96 over the Internet Using New Software. IEEE Communications Magazine, April 1997.

[19] R. O. Onvural: Asynchronous Transfer Mode Networks – Performance Issues 2:nd ed. Artech House Publishers, 1995.

[20] D. E. Comer, D. L. Stevens: Internetworking with TCP/IP, Volume II. Prentice-Hall International, 1991.

[21] D. E. Comer: Internetworking with TCP/IP, Volume I, 3:rd ed. Prentice-Hall International, 1995.

[22] M. J. Riley, I. E.G. Richardson: Digital Video Communications. Artech House Publishers, 1997.

Drafts:

1. ITU-T Recommendation H.263, 1995

9 Abbreviations

CBR	Constant Bit Rate
CODEC	Encoder/Decoder
CBR	Constant Bit Rate
CRC	Cyclic Redundancy Check
FEC	Forward Error Correction
H.261	Video coding standard
H.263	Video coding standard
IDE	Integrated Drive Electronics
IEC	International Electrotechnical Commission
IP	Internet Protocol
ISDN	Integrated Services Digital Network
ISO	International Organization for Standardization
ITU	International Telecommunication Union
MPEG	Moving Pictures Expert Group
MPEG2	Generic video coding standard
MMX	Matrix Math eXtensions
MTU	Maximum Transmission Unit
QoS	Quality of Service
PAL	Phase Alternation Line (television format)
RCBR	Renegotiated Constant Bitrate
RSTP	Real-time Stream Transport Protocol
RSVP	Resource Reservation Protocol
RTP	Real-time Transport Protocol
RTCP	Real-time Control Transfer Protocol
SCSI	Small Computer System Interface
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
UDP	User Datagram Protocol
VBR	Variable Bit Rate

Appendix A

Fig. 20 presents a dump of data from the traffic analyzer "The Sniffer" of three streams from the video streamer. The average load on the network before the test was below 1% = 100kbps with shorter peaks of 10-30%.

The first column represents the relative time, that was inserted as a reference instead of the real time, since it was easier to use.

The "Delta T" might seem miscalculated compared to the relative time scale, but this is not the case. The mismatch is due to limitations in number of visible decimals in "The Sniffer".

Stream "1" of 3200kbps was sent to one computer, two streams – "2" of 3200kbps and "3" of 1600kbps were sent to another computer.

The "Length" column represent the packet size with UDP header (8 bytes), payload is 1024 bytes. If we include the IP header of 20 bytes, the packet size increases to 1052 bytes. But this is not enough, the packet is sent over Ethernet, which add its own header called DLC of 14 bytes. Of the 14 bytes, 2*6 bytes is used for the source and destination Ethernet addresses and 2 bytes for the Ethertype which in this case is IP. Finally we end up with the total packet size of 1066 bytes (see figure 19).

Consequently we have an overhead of 42 bytes, which is 4% of the total packet size when using a 1024 bytes payload. The overhead decreases the usable bandwidth from 10Mbps to 9.6Mbps.

DLC 14 bytes	IP header 20 bytes	UDP 8 bytes	Payload 1024 bytes

Fig. 19 Ethernet packet

The maximum transmission unit (MTU) (consisting of payload + UDP and IP headers) is usually set to 1500 bytes but might be as low as 576 on certain networks. A packet that is larger than the MTU on a certain network will get fragmented. It is usually desired to avoid fragmentation therefore it is better not to send packets larger than the smallest allowed MTU on the path from the sender to the receiver. If it is known in advance what the MTU is in the environment where the server will run, it can be set when the server is started. If nothing is entered the server assumes the MTU to be 1500 bytes.

Fig. 20 below presents the three different streams and their different send time.

As can be seen in the table, each stream sends every 20ms.

Rel. Time [s]	Delta. T [s]	Stream	Length [B]
0.0000	0.0000	1	1032
0.0009	0.0009	1	1032
0.0018	0.0009	1	1032
0.0027	0.0009	1	1032
0.0035	0.0009	1	1032
0.0044	0.0009	1	1032
0.0053	0.0009	1	1032
0.0062	0.0009	1	1032
0.0070	0.0009	2	1032
0.0079	0.0009	2	1032
0.0088	0.0009	2	1032
0.0096	0.0009	2	1032
0.0105	0.0009	2	1032
0.0114	0.0009	2	1032
0.0123	0.0009	2	1032
0.0131	0.0009	2	1032
0.0140	0.0009	3	1032
0.0149	0.0009	3	1032
0.0158	0.0009	3	1032
0.0166	0.0009	3	1032
0.0200	0.0034	1	1032
0.0209	0.0009	1	1032
0.0218	0.0009	1	1032
0.0227	0.0009	1	1032
0.0235	0.0009	1	1032
0.0244	0.0009	1	1032
0.0253	0.0009	1	1032
0.0261	0.0009	1	1032
0.0270	0.0009	2	1032
0.0279	0.0009	2	1032
0.0288	0.0009	2	1032
0.0297	0.0009	2	1032
0.0305	0.0009	2	1032
0.0314	0.0009	2	1032
0.0323	0.0009	2	1032
0.0331	0.0009	2	1032
0.0340	0.0009	3	1032
0.0349	0.0009	3	1032
0.0358	0.0009	3	1032
0.0366	0.0009	3	1032
0.0400	0.0034	1	1032

Fig. 20 Dump of 3 concurrent streams, 1 and 2 are 3200kbps, 3 is 1600kbps

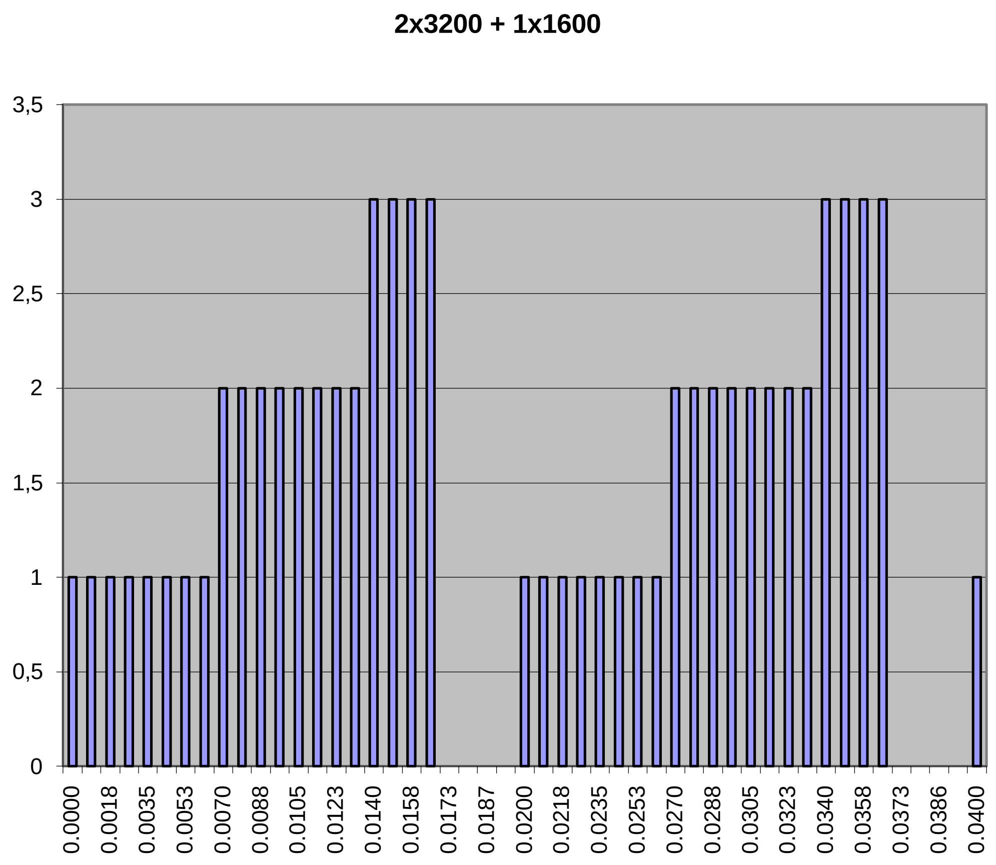


Fig. 21 Packet sending time for three concurrent streams

The heights 1, 2 and 3 (in *Fig. 14*) represent the three different streams. The empty "slots" between 0.0166 - 0.200 and 0.0366 - 0.0400 are due to the fact that one of the streams only send at half the maximum rate (when using three concurrent streams of 3200kbps each).

Appendix B

The header file is used to describe the video (type, rate, and name), where it is located (address) and how to access it (port number). The header file has the suffix *vhf (video header format)* which is used to associate the file to the client application in the Microsoft Windows environment.

Structure of the header file:

```
#Header file for streaming video
FILE=jur400cif25.263
PORT=6563
SERVER=131.115.157.234
RATE=3200
TYPE=0
#End of file
```

FILE is the filename of the video file on the server.
PORT is the port number to which the server is listening for requests
SERVER is the IP address of the server
RATE is the desired rate (in kbps) for sending the data - set to the same rate as the video was coded to.
TYPE specifies which type of video it is, in this case 0 = 263 and 1 = uncompressed.