Adaptive Video Streaming: Pre-encoded MPEG-4 with Bandwidth Scaling

A. Balk, M. Gerla, and M. Sanadidi
Network Research Laboratory, UCLA, Los Angeles, CA 90024 USA
{abalk, gerla, medy}@cs.ucla.edu

D. Maggiorini
Department of Informatics and Communication, Università degli Studi di Milano
dario@dico.unimi.it

Abstract

The increasing popularity of streaming video is a cause for concern for the stability of the Internet because most streaming video content is currently delivered via UDP, without any end-to-end congestion control. Since the Internet relies on end systems implementing transmit rate regulation, there has recently been significant interest in congestion control mechanisms that are both fair to TCP and effective in delivering real-time streams.

In this paper we design and implement a protocol that attempts to maximize the quality of real-time MPEG-4 video streams while simultaneously providing basic end-to-end congestion control. While several adaptive protocols have been proposed in the literature [28, 37], the unique feature of our protocol, the Video Transport Protocol (VTP), is its use of receiver side bandwidth estimation. Such estimation is transmitted to the sender and enables it to adapt to network conditions by altering its sending rate and the bitrate of the transmitted video stream. We deploy our protocol in a real network testbed and extensively study its behavior under varying link speeds and background traffic profiles using the FreeBSD Dummynet link emulator [31]. Our results show that VTP delivers consistent quality video in moderately congested networks and fairly shares bandwidth with TCP in all but a few extreme cases. We also describe some of the challenges in implementing an adaptive video streaming protocol.

1 Introduction

As the Internet continues to grow and mature, transmission of multimedia content is expected to increase and compose a large portion of the overall data traffic. Film and television distribution, digitized lectures, and distributed interactive gaming applications have only begun to be realized in today's Internet, but are rapidly gaining popularity. Audio and video streaming capabilities will play an ever-increasing role in the multimedia-rich Internet of the near future. Real-time streaming has wide applicability beyond the public Internet as well. In military and commercial wireless domains, virtual private networks, and corporate intra-nets audio and video are becoming commonplace supplements to text and still image graphics.

Currently, commercial programs such as RealPlayer [27] and Windows Media Player [24] provide the predominant amount of the streamed media in the Internet. The quality of the content delivered by these programs varies, but they are generally associated with low resolution, small frame size video. One reason these contemporary streaming platforms exhibit limited quality streaming is their inability to dynamically adapt to traffic conditions in the network during a streaming session. Although the aforementioned applications claim to be adaptive, there is no conclusive evidence as to what degree of adaptivity they employ as they are proprietary, closed software [28]. Their video streams are usually delivered via UDP with no transport layer congestion control. A large-scale increase in the amount of streaming audio/video traffic in the Internet over a framework devoid of end-to-end congestion control will not scale, and could potentially lead to congestion collapse.

UDP is the transport protocol of choice for video streaming platforms mainly because the fully reliable and strict in-order delivery semantics of TCP do not suit the real-time nature of video transmission. Video streams are *loss tolerant* and *delay sensitive*. Retransmissions by TCP to ensure reliability introduce latency in the delivery of data to the application, which in turn leads to degradation of video image quality. Additionally, the steady state behavior of TCP involves the repeated halving and growth of its congestion window, following the well known Additive Increase/Multiplicative Decrease (AIMD) algorithm. Hence, the throughput observed by a TCP receiver oscillates under normal conditions. This presents another difficulty since video is usually streamed at a constant rate (VTP streams are actually piecewise-constant). In order to provide the best quality video with minimal buffering, a video stream receiver requires relatively stable and predictable throughput.

Our protocol, the Video Transport Protocol (VTP), is designed with the primary goal of adapting an outgoing video stream to the characteristics of the network path between sender and receiver. If it determines there is congestion, the VTP sender will reduce its sending rate and the video encoding rate to a level the network can accommodate. This enables VTP to deliver a larger portion of the overall video stream and to achieve inter-protocol fairness with competing TCP traffic. A secondary goal of VTP is the minimal use of network and computer resources. We make several trade-offs to limit processing overhead and buffering requirements in the receiver. In general, VTP follows a conservative design philosophy by sparingly using bandwidth and memory during the streaming session.

In essence, the VTP sender asks the receiver the question: are

you receiving at least as fast as I am sending? If so, the sender increases its rate by a small amount to probe the network for unused bandwidth. If not, the sender immediately reduces its rate by an amount based on the receiver's bandwidth, the current sending rate and video bitrate.

An important aspect of VTP is that it is completely end-to-end. VTP does not rely on QoS functionality in routers, random early drop (RED), other active queue management (AQM) or explicit congestion notification (ECN). It could potentially benefit from such network level facilities, but in this paper we focus only on the case of real-time streaming in a strictly best effort network. Possible interactions between VTP and QoS routers, AQM or ECN are areas of future work.

VTP is implemented entirely in user space and designed around open video compression standards and codecs for which the source code is freely available. The functionality is split between two distinct components, each embodied in a separate software library with its own API. The components can be used together or separately, and are designed to be extensible. VTP sends packets using UDP, adding congestion control at the application layer.

This paper discusses related work in the next section and presents an overview of the MPEG-4 compression standard in Section 3. The VTP design is described in Section 4. Section 5 covers the VTP implementation and receiver buffering strategies. The experimental evaluation of VTP is treated in Section 6 and is followed by the conclusion.

2 Related Work

Recent research approaches to address the lack of a suitable end-to-end service model for multimedia streaming generally fall into two categories: 1) modifications or enhancements to AIMD congestion control to better accommodate streaming applications, or 2) model-based flow control based primarily on the results of [26]. We give several examples of each technique before presenting the motivation and design of VTP.

The Rate Adaptation Protocol (RAP) [28] is a rate based AIMD protocol intended for transmitting real-time video. The RAP sender uses receiver feedback about congestion conditions to make decisions about its sending rate and the transmitted video quality. The RAP algorithm does not result in fairness with TCP in many cases, but router support in the form of Random Early Drop (RED) can improve RAP's inter-protocol behavior to some extent.

A major difference between VTP and RAP is the degree to which they comply to AIMD. While RAP is a full AIMD protocol, VTP performs additive increase but it does not decrease its sending rate multiplicatively. Rather, it adjusts its sending rate to the rate perceived by the receiver. RAP and VTP also differ in the type of video encoding they stream. RAP is based on *layered* video encoding where the sender can decide how many layers can be sent at any given time. On the other hand, VTP assumes a *discrete* encoding scheme, where the sender chooses one of several pre-encoded streams and exclusively sends from that stream until it decides to change the video quality. Video compression is described in further detail in the next section.

In the spirit of RAP, N. Feamster proposes SR-RTP [12, 13], a backward compatible extension to the Real Time Protocol (RTP). SR-RTP uses a quality adaptation mechanism similar to RAP, but "binomial" congestion control reduces the congestion window size proportional to the square root of its value rather than halving it in response to loss. This is shown to assuage oscillations in the sending rate and produce smoother throughput. Binomial algorithms also display a reasonable amount of TCP fairness [6].

The main benefits of SR-RTP come from its features of selective retransmission of certain video packets and decoder post-processing to conceal errors due to packet loss. However, the effectiveness of selective retransmission depends strongly on the round trip time (RTT) between sender and receiver. Further, in [13], the receiver post-processing is performed offline for ease of analysis. It is not clear such recovery techniques are viable in real time or with limited processing resources.

The Stream Control Transmission Protocol (SCTP) [32] is a recently proposed protocol with many novel features designed to accommodate real-time streaming. SCTP supports multi-streaming, where a sender can multiplex several outgoing streams into one connection. This can potentially be very advantageous for compressed video formats since packets belonging to different parts of the video stream can be treated differently with respect to retransmission and order of delivery. The congestion control mechanism in SCTP is identical to TCP, where the congestion window is reduced by half in the event of packet loss. Like TCP, SCTP employs slow start to initially seek out available bandwidth and congestion avoidance to adapt to changing path conditions. This results in perfect fairness with TCP, but leads to high variability in throughput at the receiver. An investigation of the applicability of SCTP to MPEG-4 streaming is the subject of [4].

The work of J. Padhye, et. al. [26] has led to TCP-Friendly Rate Control (TFRC) [16]. TFRC is not itself a protocol, but an algorithm for maintaining the sending rate at the level of a TCP flow under the same conditions. The TFRC sender adjusts its rate according to an equation that specifies throughput in terms of packet size, loss event rate, RTT, and the retransmission timer value. TFRC is meant to serve as a congestion control framework for any applications that do not require the full reliability of TCP and would benefit from low variation in sending rate.

Application domains appropriate for TFRC include multimedia streaming, interactive distributed games, Internet telephony, and video conferencing. Several authors have applied the TFRC model to video streaming. In [34], a new error-resilient video compression method is developed which relies on simplified derivation of the TCP throughput equation. The relationship between the compression level and the congestion control model is examined. The Multimedia Streaming TCP-Friendly Protocol (MSTFP) is part of a comprehensive resource allocation strategy proposed in [37] which uses a TFRC model to adapt streaming MPEG-4 video.

Ostensibly, any rate adjustment scheme derived from TCP would suffer the same limitations of TCP itself.[1] TCP's behaviors of poor link utilization in high-loss environments and un-

[1]The TCP throughput equation in TFRC is derived for TCP New Reno in particular.

fairness against flows with large RTTs have been documented repeatedly (see, for example, [2]). Although VTP decreases its sending rate in response to packet loss, the decrease decision does not assume that all packet loss is a result of overflowed router buffers. At the same time, the amount of decrease is sufficient to restrict the sending rate to within its fair share of the network bandwidth.

In this paper we argue that it is possible to build a stable and scalable network protocol that is not underpinned by AIMD. VTP borrows the idea of additive increase from AIMD, but its decrease step is not strictly multiplicative. VTP also uses network bandwidth estimation, but in a different way than the model-based approaches described above. By combining elements of AIMD and model-based congestion control while not directly following either, VTP attempts to benefit from the strengths of each approach. VTP aims to be adaptive and flexible by making minimal assumptions about the network and using network feedback as a rough indicator, not as rigorous set of input parameters. These principles encompass the motivating factors of the VTP design.

3 MPEG-4 Background

The MPEG-4 video compression specification [18, 25] has been developed as an open standard to encourage interoperability and widespread use. MPEG-4 has enjoyed wide acceptance in the research community as well as in commercial development owing to its high bitrate scalability and compression efficiency. Packetization markers in the video bitstream are another feature which make MPEG-4 especially attractive for network video transmission. MPEG-4 is a natural choice for VTP since abundant documentation exists and numerous codecs are freely available. Like other MPEG video compression techniques, MPEG-4 takes advantage of spatial and temporal redundancy in individual frames of video to improve coding efficiency. A unique capability of MPEG-4 is support for *object-based encoding*, where each scene is decomposed into separate video objects (VOs). A typical example of the use of object based encoding is a news broadcast, where the news person is encoded as a separate foreground VO while the background images compose another object. VO motion is achieved by a progression of video object planes (VOPs).

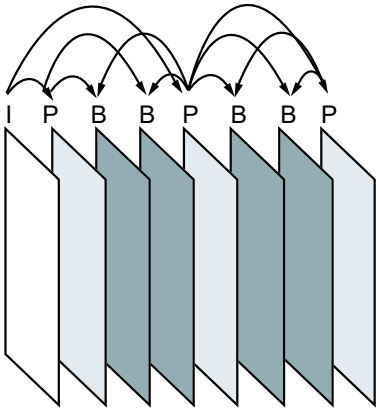

Figure 1: Group of Visual Object Planes (GOV) in MPEG-4.

There are three different types of VOPs in the MPEG-4 format: (1) Intra-coded VOPs (I-VOPs) that are encoded independently and can be considered "key" VOPs; (2) Predicted VOPs (P-VOPs) that depend on preceding I- or P-VOPs and contain predicted motion data and information about the error in the predicted values; and (3) Bi-directionally predicted VOPs (B-VOPs) that depend on both previous and next VOPs. Figure 1 shows a sequence of MPEG-4 VOPs, known as a Group of Video Object Planes (GOV), with the dependencies represented above each plane. If a VOP upon which other VOPs depend is damaged during network transmission, decoding errors will manifest in the damaged VOP as well as all its dependent VOPs, a phenomenon known as *propagation of errors*. RFC 3016[2] describes a structured packetization scheme that improves error resiliency, making error concealment and error recovery more effective to counteract error propagation.

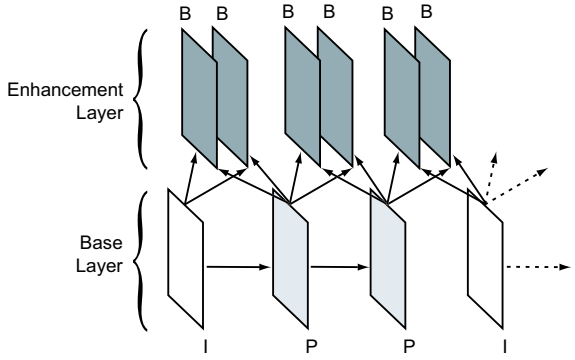

Figure 2: 2 Layered MPEG-4 encoding, VOPs at the head of an arrow depend on the VOPs at the tail.

Each VO can be composed of "layers". A *base* layer contains the basic representation of the VO and additional *enhancement* layers can be added by the codec to improve video resolution if needed. Figure 2 depicts a simple 2-layered MPEG-4 encoding, with B-VOPs comprising the enhancement layer. Since each VOP sequence can be accessed and manipulated independently, MPEG-4 encodes information about the scene composition in a separate stream within the video bitstream. The decoder's job is somewhat complex: in order to assemble a frame, it must calculate dependencies and perform the decoding algorithm for each layer of each VOP, build the scene according to the composition information, and synchronize between the independent VOP sequences, all while observing the play out time constraint.

The fundamental processing unit in MPEG-4 is a 16x16 block of pixels called a *macroblock*. Figure 3 shows a typical VOP composed of rows of macroblocks called *slices*. Macroblocks from I-, P-, and B-VOPs contain different kinds of data that reflect the particular dependency relationships of the VOP. A discrete cosine transform (DCT) is applied to each macroblock, and the resulting 16x16 matrix is then *quantized*. The range of the

[2]http://www.faqs.org/rfcs/rfc3016.html

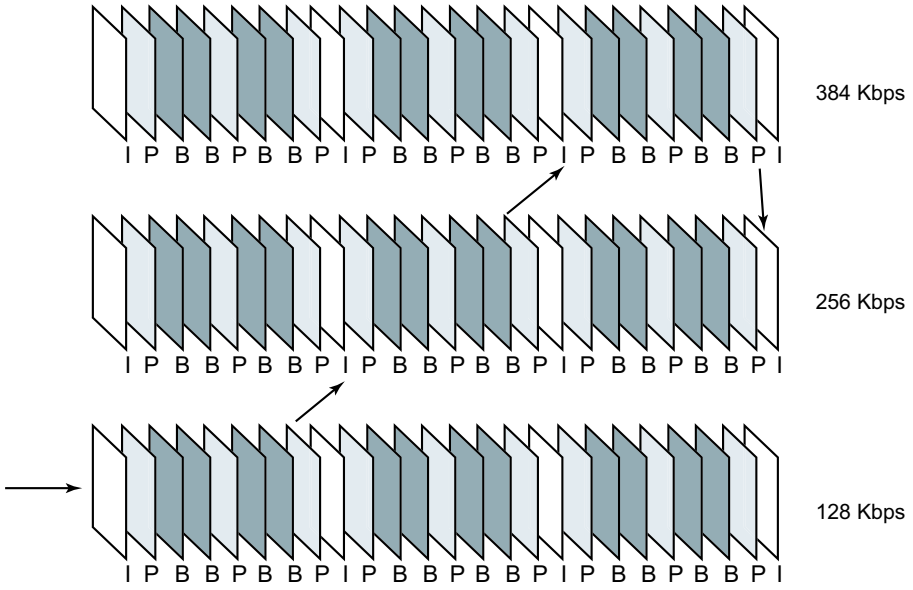

Figure 4: Example of video level switching in discrete encoding.

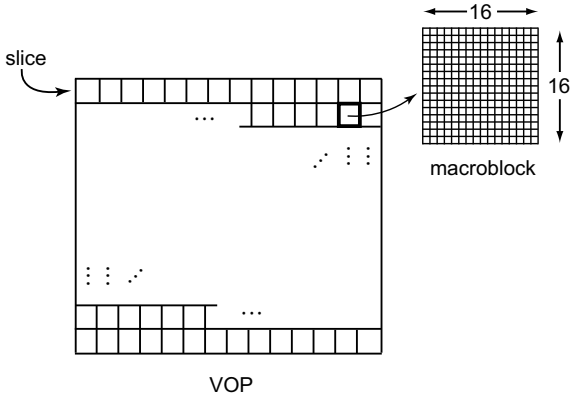

Figure 3: Macroblocks and slices in MPEG-4.

quantization parameters (QPs) is normally from 1 to 31, with higher values indicating more compression and lower quality. Ultimately, the bitrate of an MPEG-4 video stream is governed by the quantization scale of each DCT transformed macroblock.

Q. Zhang, et. al. [37] exploit this object based encoding structure by using network feedback to choose different quantizers for each VOP in real time. Foreground (more important) and background (less important) VOPs are weighted unequally, with QP values selected so that the quality of the background VOs is sacrificed first in times of congestion. The ranges of all quantizer values are such that the sum of bitrates of all the VOP streams equals the target bitrate of the whole video stream.

In contrast, VTP achieves adaptivity through a less complex approach with considerably looser semantics and lighter processing requirements. VTP is founded on the technique of discrete video encoding, where each video level is independent of the others. Each frame in the discrete encoded stream consists of only one rectangular VOP of fixed size,[3] which implies a one to one correspondence between VOPs and frames. In this sense, the MPEG-4 codec in VTP performs like a conventional frame-based encoder. In the remainder of this paper the terms "VOP" and "frame" are used interchangeably.

The VTP sender determines from which discrete stream to send video data based on receiver feedback, and sends from that level exclusively until a decision is made to change. The QPs across all frames in a single level are all within a pre-defined range. In effect, VTP adapts to one of the pre-encoded quantization scales in the video source instead of computing the quantizers in real time during the streaming session.

In Figure 4, three discrete levels of an example streaming session are shown with corresponding average bitrates. The variable in this diagram is frame size (in bytes); the frame rate and the GOV pattern are fixed between levels. The arrows indicate video quality changes during the sent stream. The stream starts at the lowest level – 128 Kbps, and then progresses to 256 Kbps and 384 Kbps as VTP determines bandwidth share is available. Later, VTP reduces the rate to 256 Kbps again as it notices contention for the link. All three streams are synchronized by frame throughout the transmission, but only one stream is sent at any given time. The quality change occurs only on I-frames, since the data in the P- and B-frames is predicted from the base I-frame in each GOV.

4 The Video Transport Protocol

A typical streaming server sends video data by dividing each frame into fixed size packets and adding a header containing,

[3] That is, there is only one video object in every scene.

for example, a sequence number, the time the packet was sent and the relative play out time of the associated frame. Upon receiving the necessary packets to reassemble a frame, the receiver buffers the compressed frame for decoding. The decompressed video data output from the decoder is then sent to the output device. If the decoder is given an incomplete frame due to packet loss during the transmission, it may decide to discard the frame. The mechanism used in the discarding decision is highly decoder-specific, but the resulting playback jitter is a universal effect. As predicted frames depend on key frames, discarding a key frame can severely reduce the overall frame rate.

The primary design goal of VTP is to adapt the outgoing video stream so that, in times of network congestion, less video data is sent into the network and consequently fewer packets are lost and fewer frames are discarded. VTP rests on the underlying assumption that the smooth and timely play out of consecutive frames is central to a human observer's perception of video quality. Although a decrease in the video bitrate noticeably produces images of coarser resolution, it is not nearly as detrimental to the perceived video quality as inconsistent, start-stop play out. VTP capitalizes on this idea by adjusting both the video bitrate and its sending rate during the streaming session. In order to tailor the video bitrate, VTP requires the same video sequence to be pre-encoded at several different compression levels. By switching between levels during the stream, VTP makes a fundamental trade-off by increasing the video compression in an effort to preserve a consistent frame rate at the client.

In addition to maintaining video quality, the other important factor for setting adaptivity as the main goal in the design is inter-protocol fairness. Unregulated network flows pose a risk to the stability and performance of the Internet in their tendency to overpower TCP connections that carry the large majority of traffic. While TCP halves its window in response to congestion, unconstrained flows are under no restrictions with respect to the amount of data they can have in the network at any time. VTP's adaptivity attempts to alleviate this problem by interacting fairly with any competing TCP flows.

The principal features of this design, each described in the following subsections, can be summarized as follows:

1. Communication between sender and receiver is a "closed loop," i.e. the receiver sends acknowledgments to the sender at regular intervals.
2. The bandwidth of the forward path is estimated and used by the sender to determine the sending rate.
3. VTP is rate based. There is no congestion window or slow start phase.

4.1 Sender and Receiver Interaction

VTP follows a client/sever design where the client initiates a session by requesting a video stream from the server. Once several initialization steps are completed, the sender and receiver communicate in a closed loop, with the sender using the acknowledgments to determine the bandwidth and RTT estimates.

The VTP video header and acknowledgment or "control packet" formats are shown in Figure 5. The symmetric design facilitates both bandwidth and RTT computation. The TYPE field

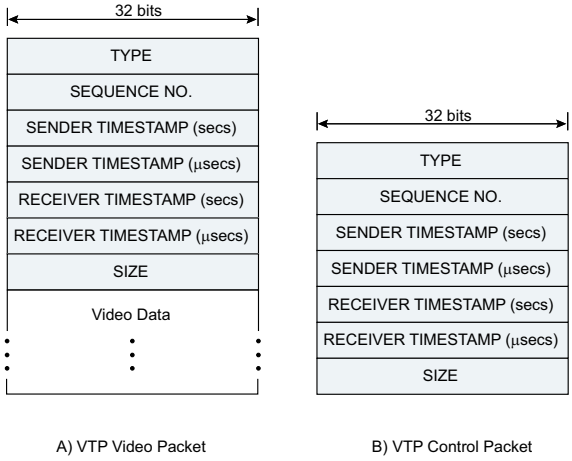

Figure 5: VTP packet formats for a) video packets and b) control packets.

is used by the sender to explicitly request a control packet from the receiver. For every k video packets sent, the sender will mark the TYPE field with an ack request, to which the receiver will respond with a control packet. The value of k is a server option that is configurable at run time by the user. The two timestamp fields for sender and receiver respectively are used for RTT measurement and bandwidth computation. VTP estimates the bandwidth available to it on the path and then calibrates its sending rate to the estimate, as detailed in the following paragraphs.

When the receiver receives a data packet with the TYPE field indicating it should send a control packet, it performs two simple operations. First, it copies the header of the video packet and writes its timestamp into the appropriate fields. Second, it writes the number of bytes received since the last control packet was sent into the SIZE field. The modified video packet header is then sent back to the sender as a control packet. This minimal processing absolves the receiver of bandwidth computation and frees it for decoding and video playback, which are highly time constrained.

Upon receipt of the control packet, the sender extracts the value in the SIZE field and the receiver timestamp. The sender is able to compute the time delta between control packets *at the receiver* by keeping the value of one previous receiver timestamp in memory and subtracting it from the timestamp in the most recently received packet. The value of the SIZE field divided by this time delta is the rate currently being achieved by this stream. This rate is also the "admissible" rate since it is the rate at which data is getting through the path bottleneck. In essence, the measured rate is equal to the bandwidth available to the connection. Thus, it is input as a bandwidth sample into the bandwidth estimation algorithm described in the next section.

The sender uses its own timestamps to handle the RTT computation. When the sender sends a video packet with the TYPE field marked for acknowledgment, it remembers the sequence number. If the sequence number on the returning control packet matches the stored value (recall the receiver simply copies the header into the control packet, changing only its own timestamp

and the SIZE field), the sender subtracts the sender timestamp in the control packet from the current time to get the RTT sample.

If either a data packet that was marked for acknowledgment or a control packet is lost, the sender notices a discrepancy in the sequence numbers of the arriving control packets. That is, the sequence numbers do not match those that the sender has recorded when sending out video packets with ack requests. In this case, the sender disregards the information in the control packets. Valid bandwidth or RTT samples are always taken from two consecutively arriving control packets.

4.2 Bandwidth Estimation and Rate Adjustment

Bandwidth estimation is an active area of research in its own right [1, 7, 8, 20]. In this paper we provide only a brief summary following [8]. Recall from the previous section that the achieved rate sample b_i can be obtained by dividing the amount of data in the last k packets by the inter-arrival time between the current and $k-1$ previous packets. As a concrete example, suppose $k = 4$ and four packets arrive at the receiver at times $t_1, ..., t_4$, each with $d_1, ..., d_4$ bytes of data respectively. The sum $\sum_{i=1}^{4} d_i$ is sent to the sender in the SIZE field of the control packet.

The sender, knowing t_1 from the last control packet and t_4 from the current control packet, computes

$$b_i = \frac{\sum_{i=1}^{4} d_i}{(t_4 - t_1)} \quad (1)$$

Exponentially averaging the samples using the formula

$$B_i = (\alpha)B_{i-1} + (1-\alpha)\left(\frac{b_i + b_{i-1}}{2}\right) \quad (2)$$

yields the bandwidth estimate B_i that is used by the sender to adjust the sending rate. The parameter α is a weighting factor that determines how much the two most recent samples should be weighed against the history of the bandwidth estimate. In experimental trials, it was determined that VTP performs best when α is a constant close to 1. Packet loss is reflected by a reduction in the achieved rate and thus in the bandwidth estimate. Since the bandwidth estimation formula takes into account losses due to both congestion and random errors, using an exponential average prevents a single packet drop due to a link error from causing a steep reduction in the estimate.

Through the estimate of the connection bandwidth, the VTP sender gains considerable knowledge about the conditions of the path. The sender uses the estimate as input into an algorithm that determines how fast to send the data packets and which pre-encoded video to use. We describe the algorithm in terms of a finite state machine (FSM), shown in Figure 6. Assuming three video encoding levels, the states Q0, Q1 and Q2 each correspond to one distinct video level from which VTP can stream. We use three levels throughout this example for simplicity, but $n > 3$ levels are possible in general. Each of the IR states, IR0, IR1, and IR2, represent increase rate states, and DR represents the decrease rate state. In Figure 6, the states and transitions involved in a quality level increase are highlighted with dashed lines.

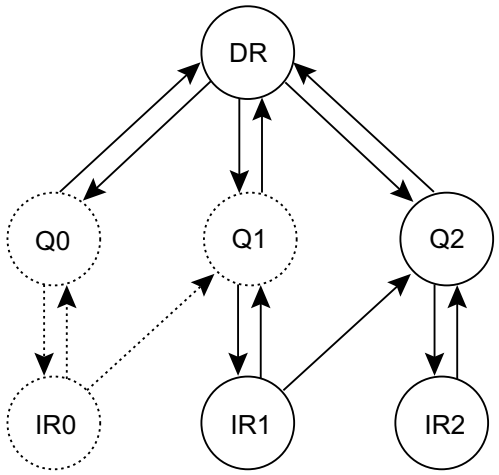

Figure 6: VTP finite state machine with states and transitions involved in a video quality level increase represented with dashed lines.

Starting in state Q0, a transition to IR0 is initiated by the reception of a bandwidth estimate that is equal to or greater than the current sending rate. Being in state Q0 only implies the VTP server is sending the lowest quality level, it says nothing about the sending rate. In state IR0, the server checks several conditions. First, it checks if the RTT timer has expired. If it has not, the server returns to Q0 without taking any action and awaits the next bandwidth estimate. If one RTT has passed, it remains in IR0 and investigates further. It next determines whether the sending rate is large enough to support the rate of the next highest level (level 1 in this case). If not, the server increases the sending rate by one packet size and returns to state Q0. If, on the other hand, the sending rate can accommodate the next quality level, the server checks the value of a variable we call "the heuristic."

The heuristic is meant to protect against over ambitiously increasing the video quality in response to instantaneous available bandwidth on the link that is short-lived and will not be able to sustain the higher bitrate stream. If the heuristic is satisfied, the server increases the sending rate by one packet size and transitions to state Q1. If the heuristic is not met, the server increases the rate by one packet and returns to state Q0. In normal operation, the server will cycle between states Q0 and IR0 while continually examining the RTT timer, the bandwidth estimate and the heuristic, and adjusting the sending rate. When conditions permit, the transition to Q1 occurs. The process repeats itself for each of the quality levels.

In the current implementation the heuristic is an amount of time, measured in units of RTT, to wait before switching to the next higher level of video quality. Ideally, the heuristic would also take into account the receiver buffer conditions to ensure a video quality increase would not cause buffer overflow. Since the receiver is regularly relaying timestamp information to the sender, it would be expedient to notify the sender of the amount of buffer space available in the ack messages. The sender would then be able to make the determination to raise the video quality with the assurance that both the network and the receiver can

handle the data rate increase. [29] examines the factors that need to be taken into account in quality changing decisions in detail.

In a rate and quality decrease, the transition to DR is initiated when the server receives a bandwidth estimate less than its current sending rate. In DR, the server checks the reference rate of each constituent quality to find the highest one that can fit within the bandwidth estimate. The server sets its sending rate to the bandwidth estimate and transitions to the state corresponding to the video quality that can be supported. Unlike the state transitions to increase quality levels, the decrease happens immediately, with no cycles or waits on the RTT timer. This conservative behavior contributes greatly to the fairness properties of VTP discussed in Section 6.2.

As the FSM suggests, the selection of the encoding bitrates is important. VTP observes the rule that a particular video encoding level must be transmitted at a rate greater than or equal to its bitrate and will not send slower than the rate of the lowest quality encoding. This could potentially saturate the network and exacerbate congestion if the lowest video bitrate is frequently higher than the available bandwidth. Additionally, if the step size between each reference rate is large, more data buffering is required at the receiver. This follows from the fact that large step sizes lead to the condition where VTP is sending at a rate that is considerably higher than the video bitrate for long periods of time.

4.3 Rate Based Congestion Control

The stability of the Internet depends on the window based AIMD algorithm of TCP. Any protocol that does not observe the AIMD scheme requires justification to be considered viable, especially for large-scale deployment. VTP has no congestion window, does not perform slow start, and does not halve its sending rate on every packet loss. However, VTP uses resources in a minimal way and relinquishes them on the first indication of congestion. Justification for the plausibility of VTP is based mainly on the practical observation that the threat to Internet stability is not posed by flows using congestion control schemes that are non-compliant to AIMD, but rather by flows under no end-system control at all – flows that are completely impervious to network conditions.

It has not been proven that Internet stability requires AIMD, but some form of end-to-end congestion control is necessary in order to prevent congestion collapse [16]. Even though VTP is not founded on AIMD, it is still able to fairly share links with TCP competitors as evidenced by the experimental results of Section 6.2. Inter-protocol fairness of VTP notwithstanding, any end-to-end mechanism that limits the flow of the real-time traffic in an environment where it competes with TCP is advantageous from the perspective of fairness. Furthermore, unlike TCP, VTP is aimed at preserving minimum variance in delivery rate at the receiver. Streaming applications that eschew TCP due to its oscillatory steady state nature can benefit from the smooth delivery rate of VTP while during times of congestion their data load on the network will be judiciously constrained.

By default, VTP performs a type of congestion avoidance: it increases its rate by a small amount on every estimated RTT. Normally the rate increase is one packet size per RTT, but it can be tuned to compensate for large RTTs. The gradual rate increase seeks out available bandwidth and enables VTP to "ramp up" the video quality if network conditions remain accommodating. This behavior parallels the additive increase phase of AIMD so that rate increases in VTP and TCP are comparable.

Throughout the duration of the connection, VTP estimates the forward path bandwidth. If the bandwidth estimate falls below the sending rate, VTP takes this as an indication of network congestion and reduces its rate. In summary, the protocol behaves conservatively by slightly increasing the send rate every RTT and cutting the rate immediately upon the arrival of "bad news."

5 VTP Implementation

We implemented VTP on the Linux platform and performed extensive evaluations using the Dummynet link emulator [31]. We developed a technique to *smooth* the bandwidth required by the outgoing video stream and compute the client buffer requirement for specific pre-encoded video segments. In this section we cover the software implementation of VTP and our approach to client buffering.

5.1 Software Architecture

The VTP implementation effort has strived to build a fully functioning video streaming platform. VTP software accepts standard Audio/Video Interleaved (AVI) files as input. For each video segment, VTP requires multiple AVI files, each of a different level of MPEG-4 compression. Two main functional units comprise the VTP architecture. A transport layer component called NetPeer provides an interface that returns an estimate of the bandwidth share of the connection. A middleware component called FileSystemPeer manages the source video data and determines the sending rate based on the estimate provided by NetPeer.

For each set of AVI files, a binary file is created that contains the discrete encoded video along with packet delimiters to guide the server in selecting the right frame when a level change needs to be made. Figure 7 shows the fields in a single record of the binary file. Within the file the video data is packetized and sorted first by frame number, then by video encoding level, and finally by number of the packet within the frame. This organization enables the FileSystemPeer to find the right packet on a video level change without performing "seeks" on the file. Audio and video portions of the AVI files are de-multiplexed in the process of creating the binary file and only the video data is stored and transmitted. Streaming audio and video in combination with VTP is a subject of future research. Upon receiving the client's request to start a stream, the FileSystemPeer opens the binary file and begins to send data at the lowest quality encoding. As the session progresses, the FileSystemPeer changes the video level in response to the NetPeer feedback.

The client and server communicate over two separate sockets: one UDP socket for data and one UDP socket for control information. Timestamps are gathered using the Berkeley Packet Filter utility (BPF)[4] running in a separate thread to minimize the in-

[4] Available from http://www-nrg.ee.lbl.gov/.

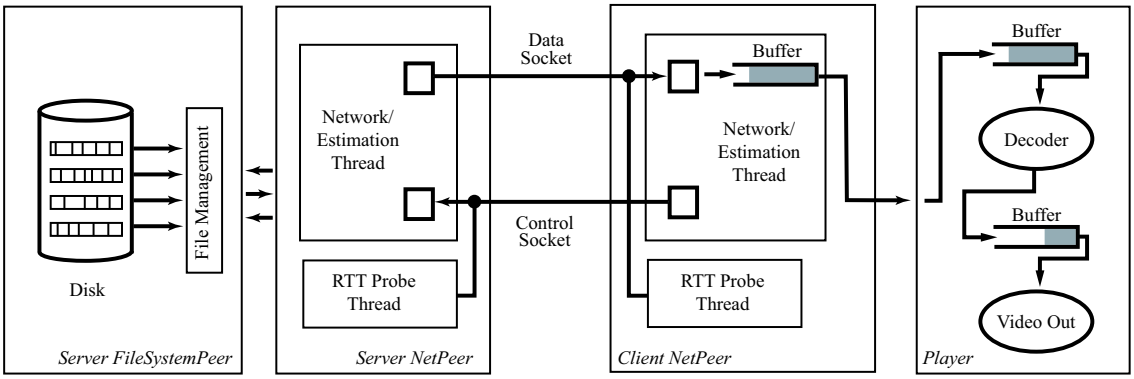

Figure 8: VTP Software Architecture.

field	bytes	description
timestamp	4	application level video playout time
sequence #	4	sequence number of packet within current frame
frame #	4	frame number
size	4	number of bytes of video data for this record
video level	4	video encoding level
rate	4	nominal sending rate for this packet
frame_done	1	indicates if this is the last packet for this frame
video level	4	video encoding level
I_frame	1	indicates if this packet belongs to an I frame
video data	variable	MPEG-4 compressed video data

Figure 7: Fields included in the VTP binary input file. The timestamp, sequence number, and size fields are different from and independent of the fields of the video packet header.

fluence of the data processing on the RTT value. The BPF allows the user mode player and server processes to collect timestamps at the network interface level that exclude operating system and protocol overhead time. The minimum measured RTT during the connection is used as the RTT value in the rate adjustment algorithm. Figure 8 presents a functional diagram of the VTP software architecture. Each of the two server components of VTP is independent and could potentially be used with other software modules. Similarly, the client NetPeer is intended to function as a generic plug-in to any software player that supports modular input. In this implementation we used the xine video player [36] for Unix systems.

A VTP software server may be implemented easily by linking the FileSystemPeer and NetPeer modules and providing a main routine to form an executable. The client side NetPeer includes buffering capability to accommodate network level buffering of video data.

The FileSystemPeer API provides two major functions:

```
is_eof = getPacket(qual, buffer, size);
rate = setRate(rtt_val, bw_est, &qual);
```

The `getPacket` function fills the `buffer` field with a header and `size` bytes of video data from video quality `qual`, where `qual` corresponds to one of the pre-encoded compression levels in the binary file. A flag is returned indicating if this is the last packet in the file. The `setRate` function realizes the algorithm in Section 4.2. The values for the parameters `rtt_val` and `bw_est` are provided by NetPeer (see NetPeer API below). The last parameter, `qual`, is passed by reference and is set by the `setRate` function and used as input in the next call to `getPacket`. It should be noted that both `getPacket` and `setRate` maintain state between calls.

The NetPeer API provides three functions:

```
bw_est = getBWE();
rtt_val = getRTT();
sendData(rate, buffer);
```

The sender uses `getBWE` to get the latest bandwidth estimate from its NetPeer. Internally, NetPeer performs non-blocking reads on the control socket to obtain the latest acknowledgment from the receiver. From the information in the ack, it computes a bandwidth estimate which is the return value of the function. The sending rate can then be computed by calling the `setRate` function of the FileSystemPeer with the bandwidth estimate as the second parameter. `GetRTT` returns the latest value of the RTT estimate. The `sendData` function determines the amount of time to wait from the `rate` parameter and then sends the `buffer` containing the header and video data.

In addition to these exported functions, several other functions are provided to handle connection initiation, opening the source video files, and other initialization and configuration tasks. The k parameter, the value of the heuristic variable (in units of RTT), and the port numbers that VTP uses are all user configurable.

5.2 Transmission Schedules for Variable Bitrate Video

In a constant bitrate (CBR) video source, the quantization parameters are continuously adjusted to maintain the target bitrate of the overall video stream. This is beneficial for network transmission but leads to varying video quality from frame to frame, and can have an unpleasant effect on the viewer's perception. MPEG-4 preserves consistent quality by increasing the bitrate at times of high motion or detail, producing a variable bitrate (VBR) encoding. In some instances the bitrate can change dramatically during the course of a video clip. The amount of rate variability is codec-dependent. In this research we investigated three MPEG-4 video codecs: DivX 4.2 [11], FFmpeg 0.4.6 [15], and Microsoft MPEG-4 version 2 [24]. After several initial tests, the Microsoft codec was found to be inappropriate for VTP. This codec uses an algorithm that drops entire frames to achieve the desired compression level, conflicting with the VTP assumption of a similar frame pattern across the set of encodings. Moreover, dropping frames for the purpose of compression has other highly undesirable effects: inconsistent frame rates, shortening of the duration of the video, and an awkward, "jumpy" playback. The rest of this section assumes that the video source is compressed with a codec that does not skip frames to affect the level of compression; such is the case with DivX and FFmpeg.

Since it would be ineffective to transmit video data at uneven, bursty rates, we developed a method for determining a transmission schedule for VBR MPEG-4 video that leads to a piecewise-constant nominal sending rate. By taking advantage of a priori knowledge of the bitrates of the stored video files, the peak bandwidth requirements and rate variability of the transmission can be significantly reduced. An appropriate sending rate can be incrementally computed by averaging the video bitrate over discrete intervals.

Let $V(t)$ represent the cumulative amount of bytes consumed by the client from the start of the streaming session to time t. In other words, if the video is encoded at a variable rate $v(\tau)$,

$$V(t) = \sum_{\tau=0}^{t} v(\tau) \tag{3}$$

As a starting point for a constant rate transmission plan, let $C(t)$ be the cumulative amount of bytes received at the client under a very simple CBR schedule: the constant rate equal to the size of the entire video segment (in bytes) divided by the duration.

Figure 9 shows $V(t)$ and $C(t)$ for a 16 MB, 130 second sample MPEG-4 video from a scene of the movie "TRON," encoded with DivX. If this video sample is sent at a constant rate equal to the slope of $C(t)$, the function

$$U(t) = C(t) - V(t) \tag{4}$$

in the bottom plot in Figure 9 leads to several basic observations that are of interest.

Intuitively, $U(t_0) < 0$ for a particular t_0 signifies that transmitting the video stream at simply the average bitrate of the entire segment would lead to buffer underrun at time t_0. The maximum positive value of $U(t)$ corresponds to the largest buffer occupancy in bytes under the same constant transmission rate. A straightforward generalization of this approach involves shortening the interval over which the average is taken, and connecting several CBR "runs" to form a sequence of transmission rates.

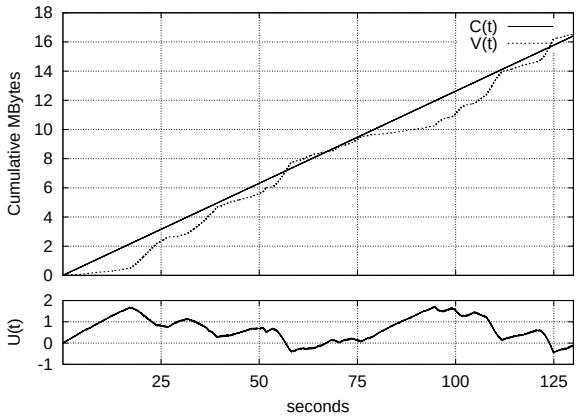

Figure 9: Cumulative amount of bytes received, $C(t)$, when the entire video segment is transmitted at a constant rate. Shown with the consumption rate, $V(t)$.

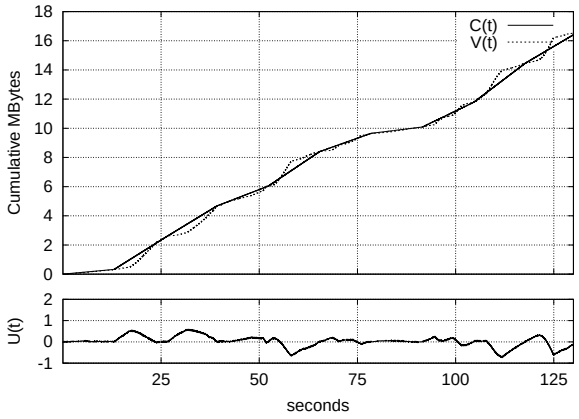

Figure 10: Cumulative amount of bytes received based on a piecewise-constant rate schedule with ten segments of equal duration.

Figure 10 shows the same video source dissected into ten intervals of equal length. Within each interval, the rate is computed as the average of the video bitrate, as before. In this figure, $C(t)$ stays closer to the $V(t)$ curve, resulting in smaller peaks in $U(t)$. The use of ten segments in particular was found in experimental trials to be a good compromise between the length and number of separate intervals. Under this plan, the sender adjusts its sending rate ten times during the course of the stream. Each sending rate is exactly the slope of the line segment for the corresponding interval. The bottom plot shows that the condition $U(t) < 0$ still holds at around $t = 60$, $t = 110$, and $t = 125$, indicating buffer underruns at these times for this sending plan. The next section addresses eliminating these underruns, and finding the minimum buffer size required in the case of equal length, constant rate intervals.

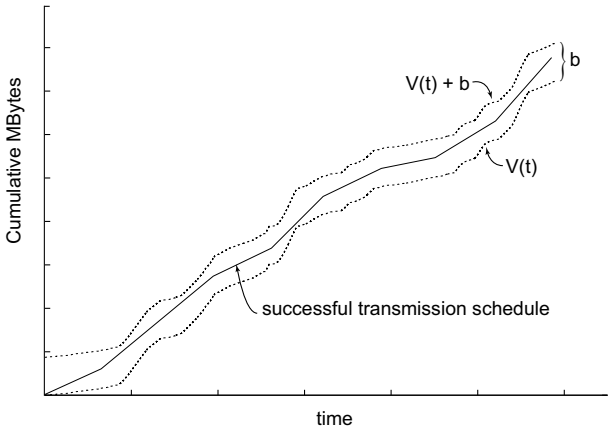

Figure 11: Cumulative bytes received under successful transmission rates.

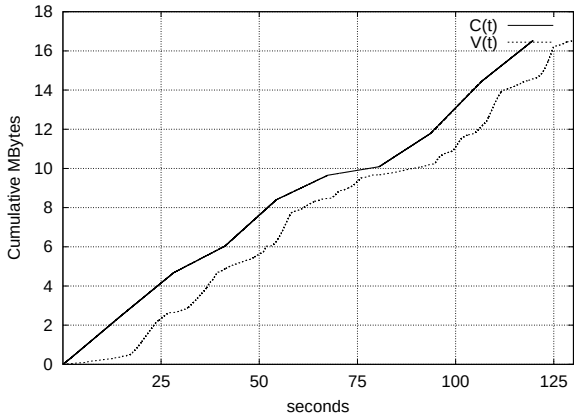

Figure 12: Cumulative bytes received, $C(t)$, under the piecewise CBR transmission schedule computed for "TRON."

5.3 Minimizing Client Buffer Requirements

The technique in the previous section can be extended to optimize the transmission schedule to ensure minimal use of receiver memory for video data buffering. Our approach follows closely the PCRTT algorithm of [23]. Given the consumption rate $V(t)$ and a buffer size b at the client, a successful transmission rate would deliver at least $V(t)$ but not more than $V(t) + b$ bytes to the client at any time t, as illustrated in Figure 11.

To find the minimum b required for a particular video stream while protecting against buffer underruns, we consider again the function $U(t)$ from equation 4. The maximum of $U(t)$ is the amount of data that the piecewise-constant rate plan will send ahead of the consumption rate. The client must allocate a buffer of at least $\max U(t)$ bytes of data to avoid data loss due to overflowing buffers. The minimum value of $U(t)$ corresponds to the greatest difference between the consumption rate and the server sending rate, i.e., the point where the sender falls most behind the receiver.

If $\left|\min U(t)\right|$ bytes could be transmitted before the time at which $\min U(t)$ occurs, underruns would be prevented. Suppose that a time t_j is chosen such that $\left|\min U(t)\right|$ bytes of data is sent in the interval $[0, t_j]$ in addition to the amount of data that needs to be sent under the constant rate plan. This way, we add $\left|\min U(t)\right|/t_j$ bytes/second to the rate computed by the piecewise-constant method to all the rates that lie in the interval $[0, t_j]$. In the worst case, time t_j can fall precisely when $U(t)$ is at its maximum. The client would then have to be able to buffer both the maximum of $U(t)$ and $\left|\min U(t)\right|$ at the instant t_j. Hence, if a $b_{\min}$ byte buffer is allocated at the client, where

$$b_{\min} = \left|\max U(t)\right| + \left|\min U(t)\right| \qquad (5)$$

both underruns and overruns will be prevented. The time t_j must be chosen before the time that $\min U(t)$ occurs, but ideally it should be chosen to be before the time of the first negative value of $U(t)$. Figure 12 shows the adjusted CBR ten-segment transmission plan for the TRON video sources with $t_j = 5$. The cumulative amount of bytes received under the new plan, $C(t)$, is always above the consumption rate $V(t)$. The value of $b_{\min}$ for this video segment is is a relatively modest 2.7 Mbytes, which is required around time $t = 17$. In general, choosing a suitable t_j depends on the size of $\left|\min U(t)\right|$ and the time at which $\min U(t)$ occurs. If $\left|\min U(t)\right|$ is large, some care must be taken so that t_j is not too small. That is, the additional bytes that need to be sent are spread out over time and not sent in a burst at the beginning of the stream.

With discrete encoding, each video is stored on disk as several distinct streams differing in their level of compression. Each stream has an associated $b_{\min}$ and the transition points between segments occur at the same time points in each level. The $b_{\min}$ for the whole segment is simply chosen to be the maximum $b_{\min}$ of the constituent streams. Since the sending rates for all the video levels are pre-computed, this value of $b_{\min}$ is known before any video data is sent. The next figures present the encoded bitrates and resulting sending rate profiles for the two sets of video sources used throughout the experimental evaluation of VTP.

The left plot of Figure 13 shows the encoded bitrate of three levels of quantization for the "TRON" video segment. The bitrates of this DivX compressed MPEG-4 video exhibit a great deal of variability – from 500 Kbps to more than 4.5 Mbps in the case of the stream with QPs in the 2 to 10 range. The corresponding piecewise-constant sending rates computed with $t_j = 5$ are shown in the right plot of Figure 13. The peak rate is reduced significantly to around 1.6 Mbps. The left plot of Figure 14 presents the bitrate trace for a trailer for the movie "Atlantis," compressed with the FFmpeg MPEG-4 video codec. The FFmpeg codec produces video that is markedly less variable rate than DivX. Another interesting point to note is that files created with FFmpeg are smaller and use less bandwidth than DivX for the same QPs. The right plot of Figure 14 shows the rate profile for the "Atlantis" sequence, with t_j again set at 5 seconds.

An alternative and commonly used approach to "pre-sending" extra bytes for protecting against underruns is to delay the initial playback at the client while it accumulates some amount of buffered video data. The video player, however, usually has its own requirements for buffering in addition to buffering done at

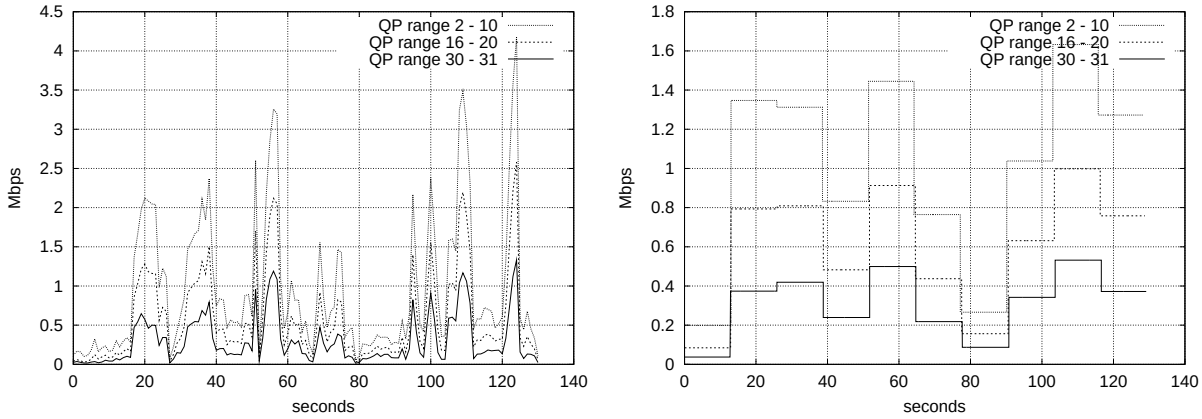

Figure 13: Source bitrates (left) and sending rate profile (right) produced for "TRON."

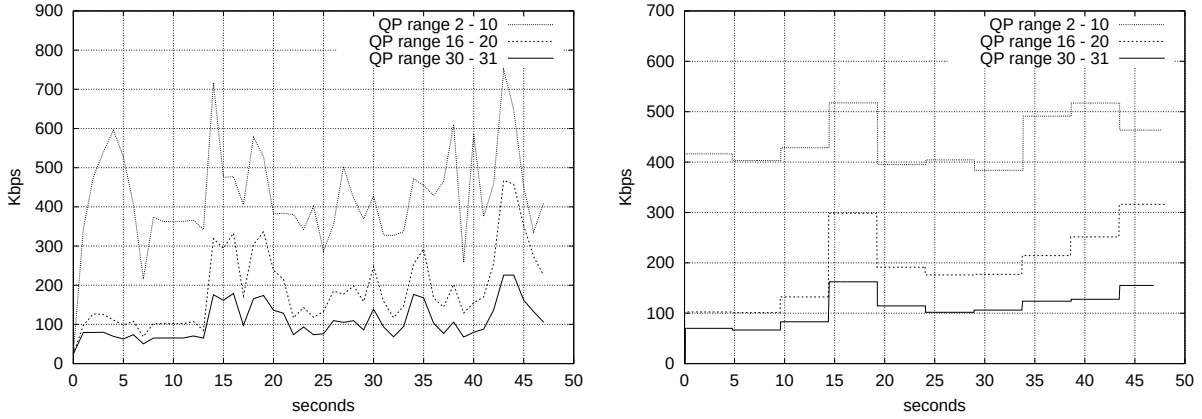

Figure 14: Source bitrates (left) and sending rate profile (right) produced for "Atlantis."

the network level. As can be seen in the architecture diagram (Figure 8), the player buffers data between the codec and the video output system to synchronize the display of consecutive frames. Buffers also need to absorb the small time scale rate changes due to variance in delay or "jitter." VTP is designed modularly to operate with many video players, hence it does not place any restrictions on the player with regard to play out start time. VTP offers $b_{\min}$ as a guard against buffer overruns and underruns resulting from differences between the sending rate and consumption rate. The decisions of exactly how much buffer space to allocate and when to start play out are left to the player.

6 Experimental Evaluation

The goals of our experimentation with VTP were to assess inter-protocol fairness between VTP and TCP and to evaluate the quality of the transmitted video played by the client. We streamed both the "TRON" and "Atlantis" video sources under various scenarios differing in the k parameter, the number of connections, and link capacity.

6.1 Basic Protocol Behavior

One of the main goals of VTP is to fairly share network resources with other traffic. VTP attempts to achieve fairness with TCP by reducing its sending rate whenever the bandwidth estimate indicates that the current rate cannot be supported. Depending on the difference between the estimate and the current rate, VTP can take several steps to back off, freeing network resources to ensure other flows obtain an even share of the link.

Figure 15 shows the behavior of VTP sending the "Atlantis" segment isolated on a 10 Mbps link with a 10 millisecond RTT. This single connection is an unlikely scenario but it clearly illustrates VTP progressing through its rate change algorithm. The plot on the left displays the sending rate and computed bandwidth estimate, while the plot on the right displays which pre-encoded video stream VTP is sending at the corresponding time.

Each video packet contains 1 Kbyte of video data, and the k parameter, which determines how often to send control packets, is set to 5. For the purpose of this example, these settings strike a balance between minimizing protocol overhead resulting from acknowledgments and the need to keep packet sizes small

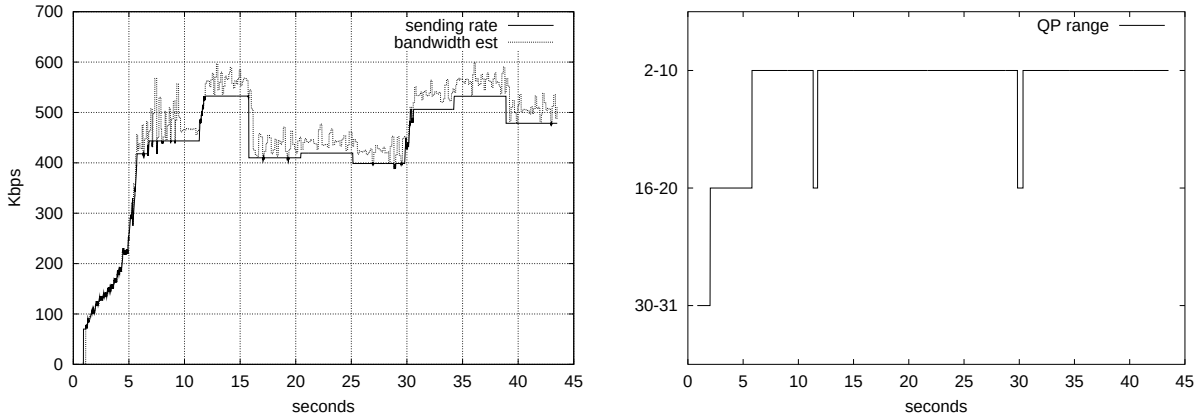

Figure 15: VTP isolated on a 10 Mbps, 10 millisecond RTT link.

to promote even packet flow. Later in this section we examine the effect of the k parameter more closely. The so-called heuristic variable, which tells VTP how long to wait before moving to the next higher video quality, is set to 2 RTTs.

In the initial phase of the "Atlantis" session, the protocol starts sending the video segment at the rate of the transmission schedule for the lowest quality video. Since there is plenty of bandwidth available on the free 10 Mbps link, VTP raises its sending rate and the quality of the video stream. By about $t = 7$ seconds, the highest quality video is being sent (with QPs in the 2 to 10 range). For the remainder of the flow, VTP sends the highest video quality at the rate prescribed in the transmission plan, with the exception of times 12 and 30 seconds. At these times VTP reduces the video quality one level for a brief time and then returns to sending the high quality video.

The reason behind these quality "valleys" can be understood by referring to the "Atlantis" transmission plan, the right plot of Figure 14. According to the plan, the rate requirement for the highest video quality suddenly increases by roughly 100 Kbps at about $t = 14$ and again at $t = 34$ seconds. In the interest of fairness, VTP does not instantaneously increase its rate by such large amounts. Instead, it switches to sending video that is one quality level lower, and continues to probe for available bandwidth by increasing the sending rate by 1 packet per RTT. After 1 second, the sending rate reaches the rate required for the highest video level and the heuristic is satisfied. This allows VTP to switch back to the highest quality video. A threshold is applied to the sending rate so that if the difference between the sending rate and the reference rate is small, the VTP server can increase its rate without performing bandwidth exploration. This happens, for example, at $t = 21$ seconds in Figure 15. This way, VTP conservatively favors fairness when the prescribed rate increase is large, but it does not rapidly change video streams on every minor rate adjustment in the send plan. The threshold is configurable by the user at run time. In this experiment, the threshold was set to 1 Kbps.

6.2 Fairness with TCP

The following experiments were designed to quantitatively measure how much bandwidth TCP and VTP attain when competing directly with each other. The experiments were performed using a relatively simple network topology in which two independent LANs were connected through a PC running FreeBSD acting as a gateway. The sender and receiver machines were located on separate LANs so that all traffic passed through the gateway which emulated a bottleneck router along an Internet path. The Dummynet utility and the Iperf program[5] were used to vary the link capacity and generate background TCP traffic respectively. In this environment all packets arrive in order, so any gap in sequence numbers can immediately be interpreted by the VTP receiver as packet loss.

Figure 16 presents the normalized throughput of VTP sending the "Atlantis" segment on a 3 Mbps, 10 ms RTT link with various numbers of TCP flows. Each column of data points represents a separate experiment where a single VTP flow several and TCP flows share the link. The x axis is labeled with total number of flows (e.g. the column labeled "16" is the result of one VTP and 15 TCP flows). As before, k is set at 5, the heuristic is 2 RTTs, and 1 Kbyte video packets are sent. The normalized throughput is computed by simply dividing the average bandwidth received by each flow by the fair share bandwidth value for each case. Perfect inter-protocol fairness would be exhibited by both VTP and TCP scoring a normalized throughput of 1. The vertical bars show the standard deviation of the TCP bandwidth values for cases where there is more than 1 TCP connection.

In the case of 2 connections, TCP obtains much more bandwidth simply because VTP has no need to transmit faster than about 450 Kbps, the average rate of the sending plan for the highest video quality (see Figure 14). As the number of connections increases, VTP and TCP compete for the limited resources of the link. VTP shares the link relatively fairly except for the case of 32 connections. In this case, the fair share value is $3000/32 = 93.5$ Kbps, which is roughly three quarters of the rate of the lowest video quality according to Figure 14. Since

[5] http:// dast.nlanr.net/Projects/Iperf/

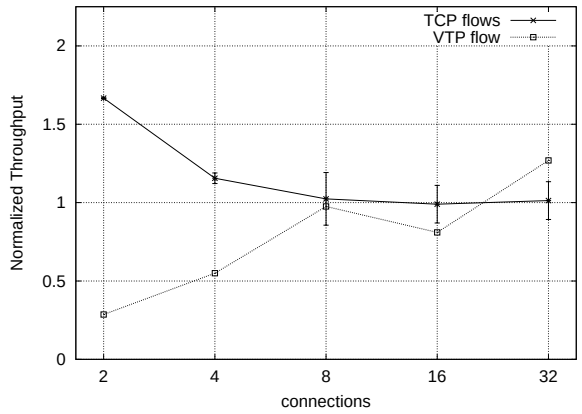

Figure 16: Single VTP flow competing with TCP on a 3 Mbps link.

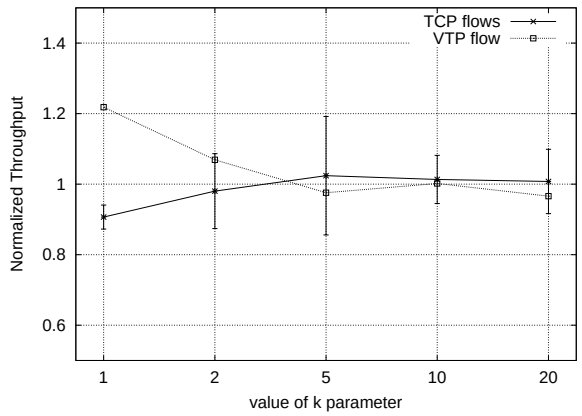

Figure 17: Effect of control packet frequency on fairness.

VTP does not send slower than the rate of the transmission plan for the lowest video quality (about 125 Kbps according to Figure 14) it uses slightly more than the fair share value of the bandwidth. It is important to note that this unfairness is not an inherent limitation of VTP, but a circumstance of the relationship between the link capacity and the video encoding. The case where VTP shares the link with 7 TCP connections results in near perfect fairness.

Looking at this case in more detail reveals the significance of the frequency at which the control packets are sent. Figure 17 shows 5 values for the k parameter, which determines how often the VTP sender will mark a video packet with a request for an acknowledgment. The experimental scenario is the same as in the previous figure where 1 VTP flow shares a 3 Mbps, 10ms RTT link with 7 TCP flows; only the value of k differs. When k is small, the bandwidth samples are taken over a shorter period. This leads to higher variability in the sampled values and overestimation of available bandwidth to a slight degree. The protocol aggressively uses more bandwidth than its fair share in cases $k = 1$ and $k = 2$. With increasing k, unfairness is eliminated. Even with $k = 20$, VTP shows a reasonable amount of efficiency

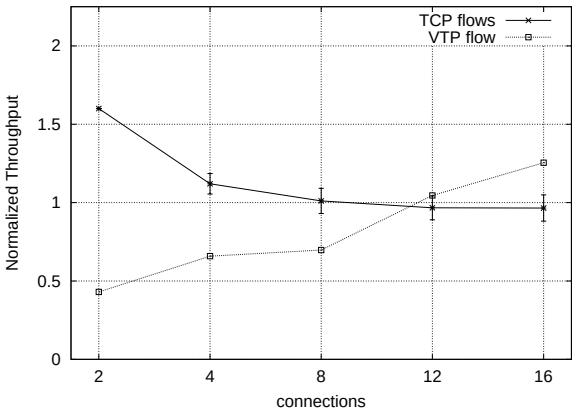

Figure 18: "TRON" video stream transmitted using VTP sharing a 5 Mbps link with TCP connections.

in its estimation and only loses to TCP by a modest amount. Additionally, as k increases, protocol overhead from sending acknowledgment decreases. Figure 17 clearly shows that VTP can maintain fairness with TCP without relying on constant receiver feedback.

In Figure 18, VTP sends the "TRON" video segment on a 5 Mbps, 10 ms RTT link against background TCP traffic. The "TRON" send plan requires significantly higher bitrates than "Atlantis," thus we set the link speed correspondingly higher. The "TRON" transmission plan also contains larger instantaneous jumps in send rate, as much as 1 Mbps for the highest video quality (see Figure 13). Both of these differences are a result of the dissimilar bitrate profiles produced by the DivX and FFmpeg codecs, as evident in Figures 13 and 14. Figure 18 shows that VTP uses less than or equal to its fair share of bandwidth in all cases except that of 16 connections, where again the link limitation is reached. The figure verifies the "Atlantis" experiments: VTP behaves fairly, in some cases generously leaving bandwidth unused, if its bandwidth share allocation is at least enough to stream the lowest quality of video.

6.3 Random Link Errors

It is of interest to determine the effect of lossy network conditions, e.g. noisy wireless links, on the performance of VTP. In Figure 19, VTP and TCP share a link on which the percentage of random errors ranges from 0.1 to 5, configured by setting Dummynet parameters accordingly. The scenario is the same as in the previous experiment where VTP streams the "TRON" segment on a 5 Mbps link and competes with 11 TCP connections (for a total of 12 connections). The average throughput of VTP is shown alongside the average throughput of the all the TCP flows for each error percentage. As evident from the figure, VTP throughput increases slightly as the error percentage increases. This results from the dynamics of the combined random loss and congestion network conditions. When TCP reacts to lost packets by halving its congestion window, it leaves bandwidth open which VTP is able to utilize since it is not so

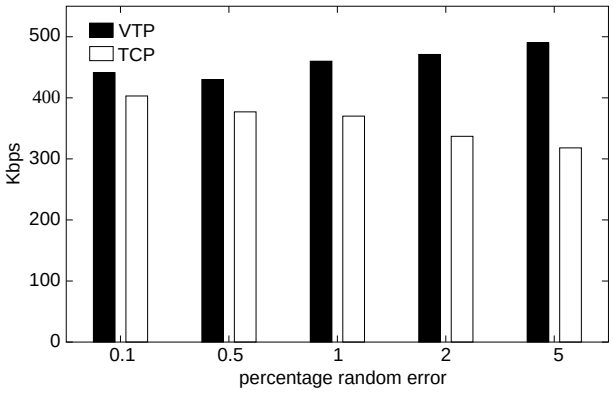

Figure 19: Average VTP and TCP throughput on a 5 Mbps, 10ms RTT link with 12 total connections and varying degrees of random errors.

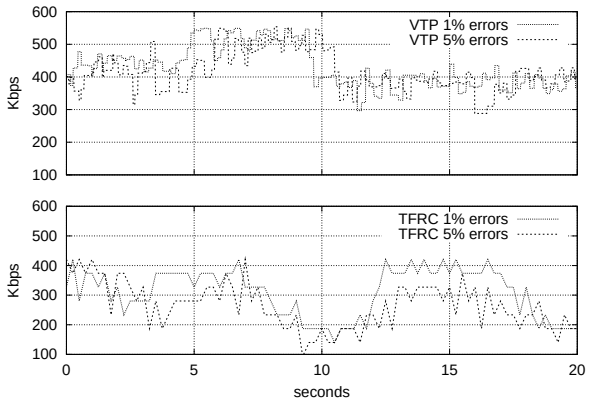

Figure 20: VTP and TFRC throughput under 1% and 5% random loss.

severely affected by the random errors. This clearly illustrates the advantage of the estimation based design of VTP in random loss scenarios.

In summary, we have demonstrated that VTP uses network resources fairly when facing competition from the AIMD based congestion control of TCP. In lightly loaded networks, VTP uses only the bandwidth required to transmit at the rate of the highest quality video stream, the remaining bandwidth can be claimed by other connections. In environments of moderate congestion, VTP fairly shares the link as long as its fair share is at least the rate of the lowest quality video. We have shown the relationship between VTP fairness and control packet frequency. We found that VTP's fairness properties are not codec specific, and that it is able to maintain stable sending rates when streaming source video with significantly different transmission plans.

6.4 Comparison with TFRC

Several recently proposed protocols [34, 35, 37] use the TFRC [16] model-based approach to adapt video streams to network conditions. As mentioned in Section 2, TFRC is a method for setting the transmit rate based on a closed form equation that estimates the throughput TCP would receive under similar packet loss conditions. Namely, an upper bound for the transmit rate T is computed by

$$T = \frac{s}{R\sqrt{\frac{2p}{3}} + t_{RTO}\left(3\sqrt{\frac{3p}{8}}\right)p\left(1+32p^2\right)} \qquad (6)$$

where s is the packet size, R the RTT, p the steady state loss event rate, and t_{RTO} the value of the TCP retransmit timeout [26]. Since, like TCP, equation 6 assumes packet loss indicates congested router buffers along the sender-receiver path, the problem of poor efficiency in noisy environments resurfaces in TFRC. In this section, we show that VTP displays an advantageous characteristic in comparsion with TFRC in network environments with random link errors.

Using the publicly available TFRC experimental code, we developed an experiment to analyze the behavior of VTP and TFRC under the same lossy conditions. We used our test LANs to submit TFRC and VTP to the same random loss environment configured with Dummynet. TFRC was set to use the Average Loss Interval method with $n = 8$ and w_1 through w_8 set to 1, 1, 1, 1, 0.8, 0.4, 0.6, 0.2 respectively. Figure 20 shows the throughput acheived by VTP and TFRC for two relatively high cases of loss: 1% and 5%. Each plot shows 20 seconds of steady state solitary flow for both schemes, with TFRC set to transmit at 400 Kbps and VTP sending the "Atlantis" video segment at the highest video quality, also at 400 Kbps. We used the TCPDump utility[6] to measure the performance of TFRC.

In the figure, the TFRC flow succumbs to random loss as throughput severely drops in both cases, falling below 200 Kbps in the 1% case, and to 100 Kbps for the 5% case. This is a consequence of the TCP-inspired nature of the TFRC scheme, where multiplicative decrease takes effect whenever it is determined that packet loss has occured. In comparison, VTP throughput is reduced by a smaller fraction and for shorter time periods since the bandwidth estimation acts as a filter against the effect of random loss. After every reduction in throughput, VTP reacts quickly and is able to regain its previous delivery rate before the next instance of errors.

Along with the results from Section 6.2, this demonstrates the adaptablity of VTP's bandwidth estimation based approach. In congestion environments, VTP conservatively yields to other traffic to promote fairness. In random loss environments, VTP maintains consistent throughput by avoiding the large drops in sending rate inherent in multiplicatively decreasing schemes.

6.5 Video Quality

An accurate, well-defined, and widely accepted standard for measuring the application-level perceived quality of network transmitted, compressed video does not exist in the literature at this time. In [21], the authors detail the problematic and subjective nature of quality assessment and discuss the shortcomings of several existing approaches. [19] suggests gathering a group of

[6] http://www.tcpdump.org

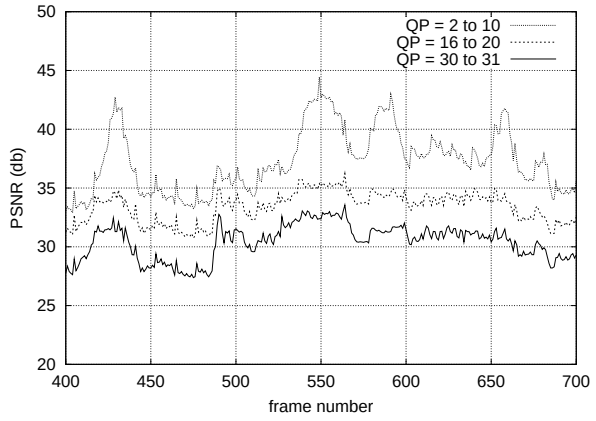

Figure 21: PSNR values for frames 400 to 700 of "Atlantis."

viewers in a room with specialized equipment to subjectively assign grades to the video they observe. Extracting a general quantitative measure from this type of assessment would be nearly impossible. The American National Standards Institute (ANSI) has produced a specification of parameters for quality degradation (blurring, distortion, tiling, etc.) [3]. However, these measures are focused entirely on quality degradation due to compression, not packet loss. Degradation due to loss is transient in nature and only affects part of the frame, whereas degradation due to compression invariably affects the whole frame. The ANSI quality parameters are insensitive to severely degraded or missing *parts* of frames, which are very noticeable to the human viewer.

Another commonly used metric for attempting to objectively measure video quality is the Peak Signal to Noise Ratio (PSNR), which is the pixel-by-pixel difference between the original and degraded images in one of the chrominance or luminance components. PSNR is defined in terms of the root mean squared error (RMSE) as PSNR = $20\log_{10}(255/\mathrm{RMSE})$. For an 8 bit image component of a degraded n by m frame f' from an original frame f,

$$\mathrm{RMSE} = \sqrt{\frac{1}{n \cdot m}\sum_{x=0}^{n-1}\sum_{y=0}^{m-1}\left[f(x,y) - f'(x,y)\right]^2} \quad (7)$$

Figure 21 shows the evolution of the PSNR of the luminance component of the "Atlantis" segment from frames 400 to 700, indicating the effect of increasing the quantization in terms of PSNR. The chart represents how PSNR can be a suitable indicator of changes in quality due to varying levels of compression. Using PSNR as a measure of perceived quality of the transmitted and client-displayed video, however, is fraught with difficulties. First, it is widely known that PSNR values do not correspond well with the characteristics of the human visual system, making it a poor gauge of perceived quality. Second, codecs that skip frames to affect compression can easily yield video that has a very high average PSNR per frame, but looks inferior in playback when compared with video with a lower average PSNR where the frame rate is held constant. Lastly, VTP draws video from the different pre-encoded streams as it progresses through its congestion control algorithm. Each received frame would have to be matched to its source stream to compute the correct PSNR values. This is also complicated by the fact that it is natural for two consecutive frames to have considerably different PSNR values, as evident from Figure 21.

For these reasons, in the experimental evaluation of the video quality delivered by VTP, we concentrate on two key parameters that are easy to interpret: the frame rate of the received video and the average values of the quantization parameters. We place a rather strict constraint on the player by configuring it to only display frames which are received completely intact, i.e., frames which have any errors due to packet loss are discarded. This bolsters the importance of the play out frame rate and magnifies the performance of VTP in terms of its key goal of providing a stable frame rate through quantization scale adjustment.

Figure 22 contrasts the frame rate of the received "Atlantis" stream using VTP and non-adaptive streaming. By non-adaptive streaming, we mean the highest video rate is sent according to its transmission plan throughout the duration of the streaming session, regardless of network conditions. No bandwidth estimation or video quality changes are performed, and the rate changes only when dictated by the piecewise-constant transmission schedule developed for "Atlantis." The experimental scenario is the same as in the previous section where VTP is competing with 15 TCP flows on a 3 Mbps capacity link with a 10 millisecond RTT. The non-adaptive streaming flow is likewise examined under the same conditions. The overall frame rate of the encoded source video is 23.975 frames per second (fps) in both cases. At several instances, around times 7 and 15 seconds, the non-adaptive frame rate drops below 15 fps, which is widely held to be the threshold of viewable video. With VTP, these severe decreases are avoided and the frame rate is always in the range 18 to 24 fps.

Figure 23 depicts another representative example of the advantage gained by VTP adaptivity. In this experiment, the conditions are those of the fourth case in figure 18: 1 monitored flow (either VTP or non-adaptive streaming) sharing a 5 Mbps, 10 ms RTT link with 11 competing TCP connections. As the streaming session progresses, VTP discovers the fair share of available bandwidth and appropriately tunes to sending rate and video bitrate to avoid overflowing the router buffer. The resulting frame rate of the VTP stream stabilizes with time, while the frame rate of the non-adaptive stream increasingly oscillates toward the end of the segment, suffering from the effect of router packet drops.

In Figure 24 we present the average values of the QPs of the "Atlantis" segment throughout the duration of the session. Both the 2 Mbps and 3 Mbps cases are shown. The plot verifies that VTP adapts the outgoing video stream to fit the available network bandwidth. When there is little contention for the link, e.g. 2 and 4 total connections, VTP chooses video primarily from the high quality, high bitrate stream (recall lower QP values imply less compression and higher quality). As the number of competing TCP connections increases, the QP values consistently increase, indicating VTP lowering the quality of the outgoing video in response to congestion. This clearly illustrates the mechanism by which VTP attains adaptivity. VTP is also

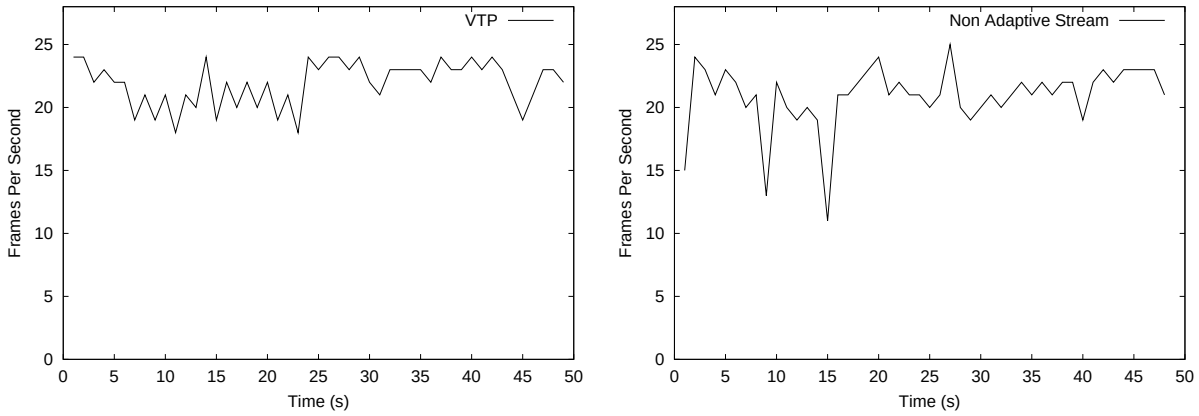

Figure 22: Frame rate of received "Atlantis" stream using VTP and Non-Adaptive Streaming.

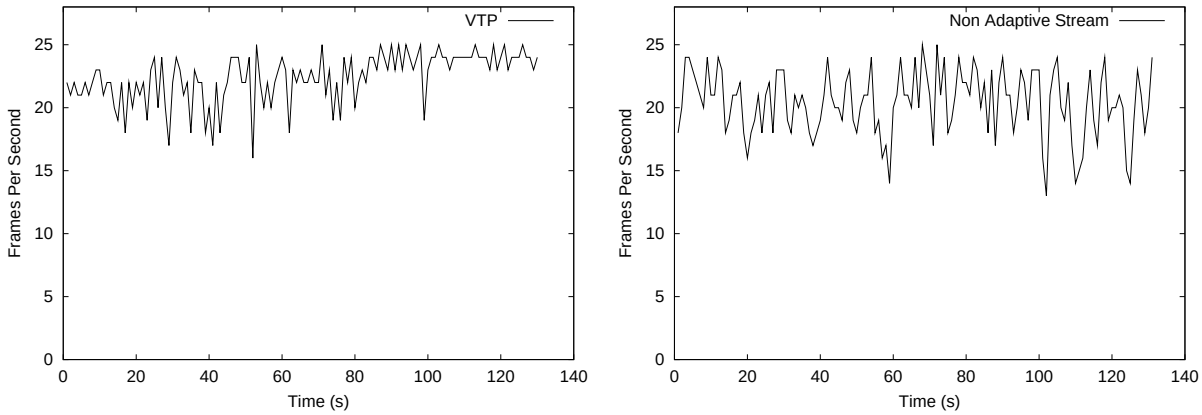

Figure 23: Frame rate of received "TRON" stream with VTP and Non-Adaptive Streaming.

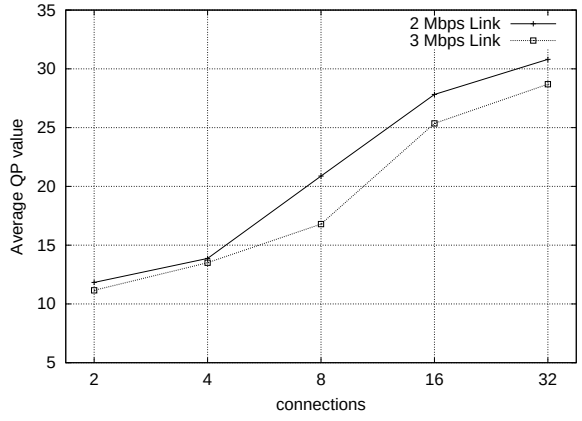

Figure 24: Average values of quantization parameters of the delivered "Atlantis" stream.

aware of the additional bandwidth afforded to it by the increase in link capacity from 2 to 3 Mbps. In the cases of 8, 16, and 32 connections, VTP carefully chooses the highest quality outgoing stream that will fit its fair share of the available bandwidth. This leads to a QP reduction of between 3 and 5, indicating higher quality video being sent when more bandwidth is available at 3 Mbps.

7 Conclusion

In this paper we designed, implemented and tested a new protocol to stream compressed video in real-time. A distinct feature of VTP is the use of bandwidth estimation to adapt the sending rate and the video encoding in response to changes in network conditions. We developed VTP in accordance with open standards for video compression and file formats, and built a plug-in for a widely used video player to serve as the VTP receiver. We have made an effort to make VTP easily extensible.

VTP was evaluated in a controlled network environment under a variety of link speeds and background traffic. Experimen-

tal results show that VTP offers considerable gains over non-adaptive streaming in effective frame rate. To a large extent, VTP behaves fairly toward TCP when both protocols compete in a congested network.

We found that VTP fairness toward TCP is vulnerable if the lowest video bitrate is higher than the average link fair share available to VTP. A priori knowledge of the general link capacity and typical network utilization can be extremely useful in the selection and configuration of the video sources for VTP. We believe this information is usually not difficult to obtain for administrators, and that a small amount of careful manual configuration is a reasonable price for the advantages of VTP.

References

[1] N. Aboobaker, D. Chanady, M. Gerla, and M. Y. Sanadidi, "Streaming Media Congestion Control using Bandwidth Estimation," In *Proceedings of MMNS '02*, October, 2002.

[2] A. Augé and J. Aspas, "TCP/IP over Wireless Links: Performance Evaluation," In *Proceedings of IEEE 48th VTC '98*, May 1998.

[3] American National Standards Institute. *American National Standard for Telecommunications - Digital Transport of One-Way Video Telephony Signals - Parameters for Objective Performance Assessment*, T1.801.03-1996.

[4] A. Balk, M. Sigler, M. Gerla, and M. Y. Sanadidi, "Investigation of MPEG-4 Video Streaming over SCTP," In *Proceedings of SCI '02*, July 2002.

[5] C. Barker, Z. Xiong, and A. Kuh, "Dynamic Programming Based Smoothing of VBR Video Traffic," In *12th International Packet Video Workshop*, April 2002.

[6] D. Bansal and H. Balakrishnan, "Binomial Congestion Control Algorithms," In *Proceedings of INFOCOMM '01*. April 2001.

[7] C. Casetti, M. Gerla, S. S. Lee, S. Mascolo, and M. Sanadidi, "TCP with Faster Recovery," In *Proceedings of MILCOM '00*, October 2000.

[8] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links," In *Proceedings of ACM MOBICOM '01*, July 2001.

[9] S. Cen, J. Walpole, and C. Pu, "Flow and Congestion Control for Internet Media Streaming Applications," In *Proceedings of SPIE Multimedia Computing and Networking '98*, January 1998.

[10] L. Cheng and M. El Zarki, "The Analysis of MPEG-4 Core Profile and its System Design," In *Proceedings of MTAC '01*, November 2001.

[11] The DivX Networks home page. http://www.divxnetworks.com/

[12] N. Feamster, D. Bansal, and H. Balakrishnan, "On the Interactions Between Layered Quality Adaptation and Congestion Control for Streaming Video," In *11th International Packet Video Workshop*, April 2001.

[13] N. Feamster, *Adaptive Delivery of Real-Time Streaming Video*. Masters thesis, MIT Laboratory for Computer Science, May 2001.

[14] W. Feng and J. Rexford, "Performance Evaluation of Smoothing Algorithms for Transmitting Variable Bit Rate Video," *IEEE Trans. on Multimedia*, 1(3):302-313, September 1999.

[15] The FFmpeg homepage. http://ffmpeg.sourceforge.net/

[16] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications," In *Proceedings of ACM SIGCOMM '00*, August 2000.

[17] S. Floyd, M. Handley, and E. Kohler, "Problem Statement for DCP," IETF Internet-Draft, Feb. 2002 http://www.icir.org/floyd/papers.html

[18] International Organization for Standardization. *Overview of the MPEG-4 Standard*, December, 1999.

[19] ITU-T Recommendation P.910. *Subjective Video Quality Assessment Methods for Multimedia Applications*, International Telecommunication Union, Telecommunication Standardization Sector, 1996.

[20] K. Lai and M Baker, "Measuring Link Bandwidths using a Deterministic Model of Packet Delay," In *Proceedings of ACM SIGCOMM '00*, August 2000.

[21] X. Lu, R. Morando, and M. El Zarki, "Understanding Video Quality and its use in Encoding Control," In *12th International Packet Video Workshop*, April 2002.

[22] A. Manhanti, D. Eager, M. Vernon, and D. Sundaram-Stukel, "Scalable On-Demand Media Streaming with Packet Loss Recovery," In *Proceedings of ACM SIGCOMM '01*, August 2001.

[23] J. McManus and K. Ross, "Video-on-Demand Over ATM: Constant-Rate Transmission and Transport," *IEEE Journal on Selected Areas in Communications*, 14(6):1087-1098, August 1996.

[24] Microsoft Windows Media Player home page. http://www.microsoft.com/windows/windowsmedia/

[25] The MPEG home page. http://mpeg.telecomitalialab.com/

[26] J. Padhye, V. Firoio, D. Townsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," In *Proceedings of ACM SIGCOMM '98*, September 1998.

[27] The RealPlayer home page. http://www.real.com/

[28] R. Rejaie, M. Handley, and D. Estrin, "RAP: An End-to-End Rate-Based Congestion Control Mechanism for Realtime Streams in the Internet," In *Proceedings of INFOCOMM '99*, March 1999.

[29] R. Rejaie, M. Handley, and D. Estrin, "Layered Quality Adaptation for Internet Video Streaming," In *Proceedings of ACM SIGCOMM '99*, September 1999.

[30] R. Rejaie, M. Handley, and D. Estrin, "Architectural Considerations for Playback of Quality Adaptive Video over the Internet," In *Proceedings of IEEE Conference on Networks*, September 2000.

[31] L. Rizzo, “Dummynet and Forward Error Correction,” In *Proceedings of Freenix ’98*. June 1998.

[32] The Stream Control Transmission Protocol (SCTP), RFC 2960, http://www.ietf.org/rfc/rfc2960.txt

[33] B. Smith, *Implementation Techniques for Continuous Media Systems and Applications*. PhD thesis, Univ. of California at Berkeley, December 1994.

[34] D. Tan and A. Zahkor, “Real-time Internet Video Using Error Resilient Scalable Compression and TCP-friendly Transport Protocol,” *IEEE Trans. on Multimedia*, 1(2):172-186, May 1999.

[35] N. Wakamiya, M. Miyabayashi, M. Murata, and H. Miyahara, “MPEG-4 Video Transfer with TCP-friendly Rate Control,” In *Proceedings of MMNS ’01*. October 2001.

[36] The xine video player home page. http://xine.sourceforge.net.

[37] Q. Zhang, W. Zhe, and Y. Q. Zhang, “Resource Allocation for Multimedia Streaming Over the Internet,” *IEEE Trans. on Multimedia*, 3(3):339-355, September 2001.