

Adaptive MPEG-4 Video Streaming with Bandwidth Estimation

A. Balk, D. Maggiorini, M. Gerla, and M. Y. Sanadidi

Network Research Laboratory, UCLA, Los Angeles CA 90024, USA

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 [20, 27], the unique feature of our protocol, the Video Transport Protocol (VTP), is the use of receiver side bandwidth estimation. 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 [23]. 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 a commonplace supplements to text and still image graphics.

Currently, commercial programs such as RealPlayer [19] and Windows Media Player [16] 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 [20]. 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 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 end system 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.

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 is an area 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 presents the VTP design in Section 2. Section 3 covers the VTP experiments and results. The conclusion and a brief discussion of future work follow.

2 The Video Transport Protocol

A typical video streaming server sends video data by dividing each frame into fixed size packets and adding a header containing, for example, a sequence number, the time the packet was sent, and the relative play out time of the associated video 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. In MPEG-4 video, which we use in this paper, there are dependencies between independent or “key” frames and predicted frames. Discarding a key frame can severely effect the overall frame rate as errors will propagate to all frames predicted from the key frame.

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.

- VTP is rate based; the bandwidth of the forward path is estimated and used by the sender to determine the sending rate.

2.1 Sender and Receiver Interaction

VTP follows a client/server 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.

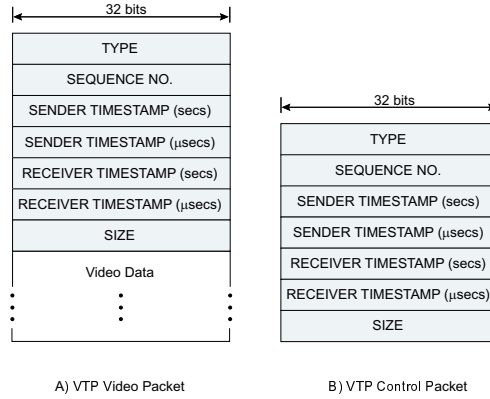


Fig. 1. VTP packet formats for a) video packets and b) control packets.

Figure 1 shows the VTP video header and acknowledgment or “control packet” formats. The symmetric design facilitates both bandwidth and RTT computation. The TYPE field 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.

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.

2.2 Bandwidth Estimation and Rate Adjustment

Bandwidth estimation is an active area of research in its own right [1, 4, 5, 13]. In this paper we provide only a brief summary following [5]. 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, \dots, t_4 , each with d_1, \dots, 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.

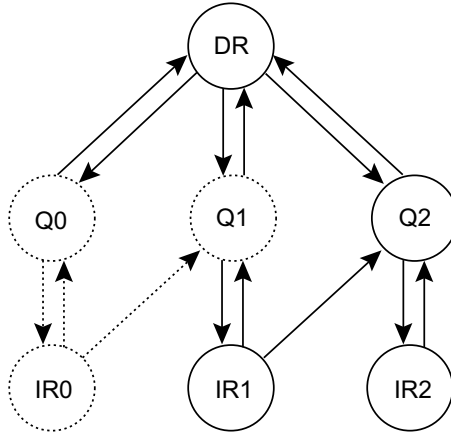


Fig. 2. VTP finite state machine with states and transitions involved in a video quality level increase represented with dashed lines.

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 send. We describe the algorithm in terms of a finite state machine (FSM), shown in Figure 2. 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 2, the states and transitions involved in a quality level increase are highlighted 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 control packet. 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. [21] 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 3.2.

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.

3 Experimental Evaluation

We implemented VTP on the Linux platform and performed extensive evaluations using the DummyNet link emulator [23]. 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. The goals of our experiments were to assess inter-protocol fairness between VTP and TCP, and to evaluate the quality of the transmitted video played by the client. In this section we cover the results of our experimental evaluation.

3.1 Transmission Schedules for Variable Bitrate Video

In a constant bitrate (CBR) video source, the compression level is 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 [6], FFmpeg 0.4.6 [10], and Microsoft MPEG-4 version 2 [16]. 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 VTP’s assumption of a similar frame pattern across the set of encodings.

Since it would be ineffective to transmit video data at uneven, bursty rates, we “smoothed” the VBR MPEG-4 video to develop a piecewise-constant sending rate profile. Figure 3 shows the results of applying a modified version of the PCRTT algorithm [15] to a 130 second sample of the movie “TRON” encoded with the DivX codec at three different levels of compression. The “QP range” in the figure represents the amount of compression applied by the codec: “QP” stands for “quantization parameters,” where higher QPs imply more compression and the lower quality. The peak rate is reduced significantly, from more than 4 Mbps to around 1.6 Mbps in the transmission plan. Similarly, Figure 4 shows the smoothing algorithm applied to a 50 second sample of a trailer for the movie “Atlantis” produced with the FFmpeg codec for three different ranges of quantization parameters. These two sets of video sources are used throughout the experimental evaluation of VTP.

3.2 Fairness with TCP

The first set of experiments was designed to quantitatively measure how much bandwidth TCP and VTP attain when competing directly with each other. We streamed both the “TRON” and “Atlantis” video sources in various scenarios differing mainly in link capacity and number of competing TCP connections. The experiments were performed using a relatively simple network topology in

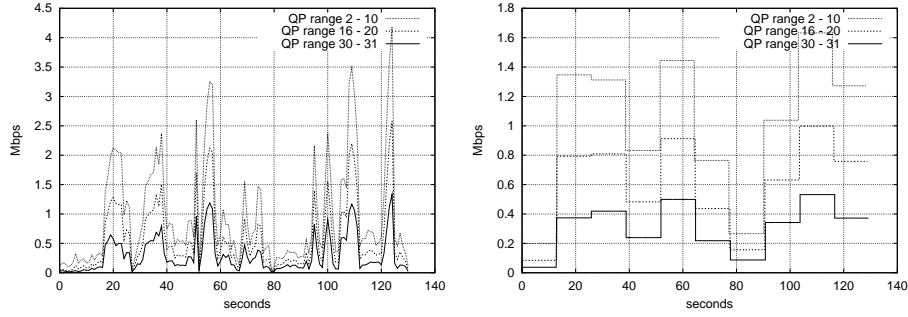


Fig. 3. Source bitrates (left) and sending rate profile (right) produced for “TRON.”

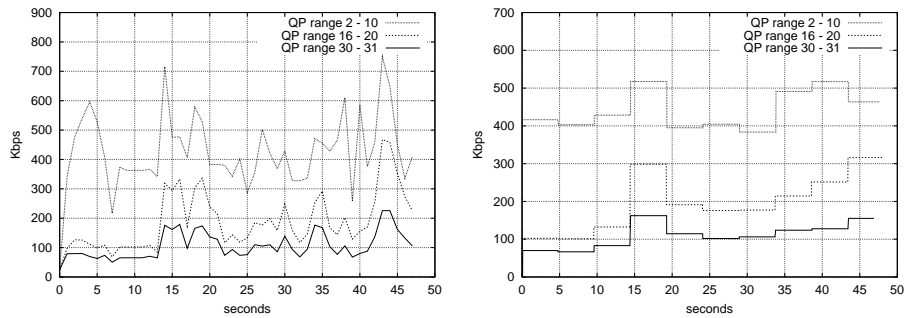


Fig. 4. Source bitrates (left) and sending rate profile (right) produced for “Atlantis.”

which two independent LANs were connected through a PC running FreeBSD acting as a gateway. The Dummynet utility and the Iperf program¹ 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 receiver as packet loss.

Figure 5 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 and several 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). The normalized throughput is computed by simply dividing the average

¹ <http://dast.nlanr.net/Projects/Iperf/>

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.

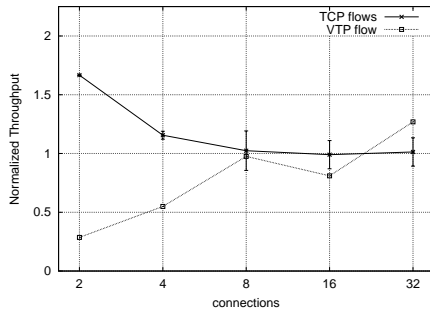


Fig. 5. Single VTP flow competing with TCP on a 3 Mbps link.

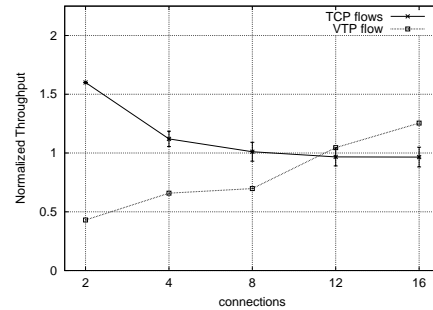


Fig. 6. “TRON” video stream transmitted using VTP sharing a 5 Mbps link with TCP connections.

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 4). 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 $3000/32 = 93.75$ Kbps, which is roughly three quarters of the rate of the lowest video quality according to Figure 4. Since VTP does not send slower than the rate of the transmission plan for the lowest video quality (about 125 Kbps according to Figure 4) 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.

In Figure 6, 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 3). Both of these differences are a result of the dissimilar bitrate profiles produced by the DivX and FFmpeg codecs, as evident in Figures 3 and 4.

Figure 6 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 allo-

cation is at least enough to stream the lowest quality of video.

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. Additionally, 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.

3.3 Video Quality

In the evaluation of video quality delivered by VTP, we concentrated on two key parameters: 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.

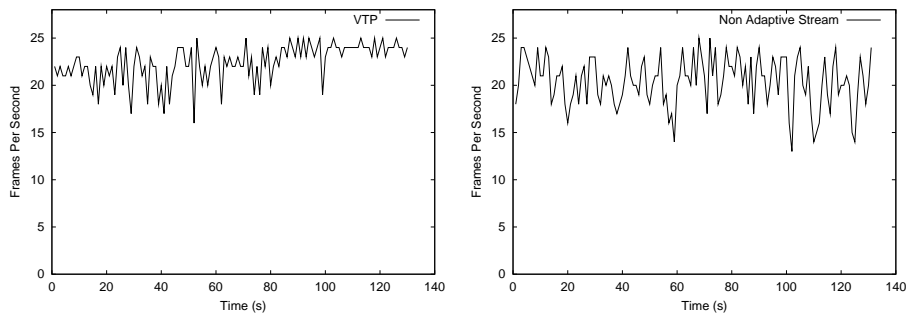


Fig. 7. Frame rate of received “TRON” stream with VTP and Non-Adaptive Streaming.

Figure 7 depicts a representative example of the advantage gained by VTP adaptivity. In this experiment, the conditions are those of the fourth case in Figure 6: 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,

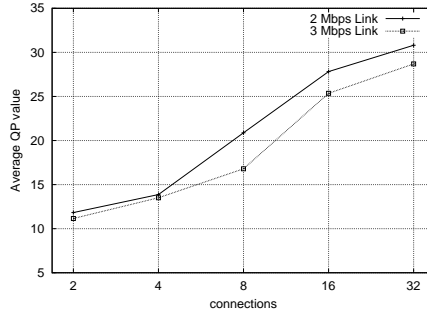


Fig. 8. Average values of quantization parameters of the delivered “Atlantis” stream.

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 8 we present the average values of the QPs of the “Atlantis” segment throughout the duration of the session. We show the 3 Mbps case from the experiment in the previous section together with the case of a 2 Mbps link. 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 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.

4 Conclusion

In this paper we designed, implemented and tested a new protocol to stream MPEG-4 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. Experimental 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.

For any streaming system to be fully useful, audio and video must be multiplexed into the data stream and synchronized during play out. A near term goal is to include the capability to adaptively stream audio and video in combination under the VTP protocol. We will also further investigate the effect of changing the k parameter, the number of packets used to compute a single bandwidth sample. We plan to implement an algorithm to dynamically adjust k during streaming to improve VTP's efficiency and fairness with TCP. Another advantage would be reducing the amount of manual user configuration required.

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. D. Bansal and H. Balakrishnan, "Binomial Congestion Control Algorithms," In *Proceedings of INFOCOMM '01*, April 2001.
4. C. Casetti, M. Gerla, S. S. Lee, S. Mascolo, and M. Sanadidi, "TCP with Faster Recovery," In *Proceedings of MILCOM '00*, October 2000.
5. 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.
6. The DivX Networks home page. <http://www.divxnetworks.com/>
7. 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.
8. N. Feamster, *Adaptive Delivery of Real-Time Streaming Video*. Masters thesis, MIT Laboratory for Computer Science, May 2001.
9. 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.
10. The FFmpeg homepage. <http://ffmpeg.sourceforge.net/>
11. S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications," In *Proceedings of ACM SIGCOMM '00*, August 2000.
12. International Organization for Standardization. *Overview of the MPEG-4 Standard*, December, 1999.

13. K. Lai and M Baker, "Measuring Link Bandwidths using a Deterministic Model of Packet Delay," In *Proceedings of ACM SIGCOMM '00*, August 2000.
14. 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.
15. 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.
16. Microsoft Windows Media Player home page. <http://www.microsoft.com/windows/windowsmedia/>
17. The MPEG home page. <http://mpeg.telecomitalialab.com/>
18. 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.
19. The RealPlayer home page. <http://www.real.com/>
20. R. Rejaie, M. Handley, and D. Estrin, "RAP: An End-to-End Rate-Based Congestion Control Mechanism for Real-time Streams in the Internet," In *Proceedings of INFOCOMM '99*, March 1999.
21. R. Rejaie, M. Handley, and D. Estrin, "Layered Quality Adaptation for Internet Video Streaming," In *Proceedings of ACM SIGCOMM '99*, September 1999.
22. 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.
23. L. Rizzo, "Dummysnet and Forward Error Correction," In *Proceedings of Freenix '98*. June 1998.
24. 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.
25. 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.
26. The xine video player home page. <http://xine.sourceforge.net>.
27. 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.