

# A Practical Technique to Support *Controlled Quality Assurance* in Video Streaming across the Internet

Yingfei Dong, Rohit Rakshe, and Zhi-Li Zhang

Dept. of Computer Science & Engineering  
University of Minnesota, Minneapolis, MN 55455

Email: dong,zhzhzhang@cs.umn.edu, Tel: (612) 625-8568 Fax: (612) 625-0572

## Abstract

Video Streaming across wide-area networks is one of the most important applications on the Internet. However, most studies of video streaming have been done in theoretical settings instead of practical settings. In this paper, we focus on the quality assurance issue on best-effort networks and propose a practical technique, named *staggered two-flow video streaming*. We deliver a stored video into two separate flows in a staggered fashion via a VPN pipe from a central server to a proxy server. One containing its essential portion is delivered using a novel *controlled TCP*, and the other containing its enhanced portion is transmitted using a *rate-controlled RTP/UDP*. To address the issue of video quality assurance in such a system, we use application-aware approaches to control bandwidth sharing and interactions among flows by exploiting the inherent priority structure of videos, the storage space on proxy servers and the coarse-grain bandwidth assurance of VPN. Our experiments using FreeBSD and simulations on NS2 both have demonstrated the efficacy of the proposed technique.

## I. INTRODUCTION

Video streaming across wide-area networks is an important component of emerging global multimedia content distribution networks. Proxy-assisted video delivery systems have been developed in both the research community [2], [7], [13], [18], [21], [20], [22], [23] and industry (e.g., Akamai, Real Networks). Figure 1 depicts a typical proxy-assisted video delivery system over a rather simplistic internetwork. At the core of this proxy video distribution system resides a (or several) *central video server(s)* with a large video repository. A collection of *video proxy servers* is strategically placed across the internetwork, typically attached to the gateway routers connecting the wide-area backbone network and the local access networks. These proxy servers, which are simpler in their functionality, assist the central server(s) in the distribution of stored videos to a large number of end users geographically dispersed at various local access networks. These proxy servers, together with the central server(s), form a virtual video distribution network over the internetwork.

Proxy-assisted video streaming systems offer several important advantages. Proxy servers can exploit their processing and buffering capabilities to provide network-wide video streaming and media control along the distribution tree in a coordinated but distributed manner. They can also utilize their potentially large disk storage space to prefetch/cache video data, thereby significantly reducing the network resource requirements in the backbone wide-area network. Furthermore, because of their strategic positions inside an internetwork, proxy servers can take into account both the constraints of the underlying network environments as well as application-specific information in optimizing video transmission. Likewise, proxy servers can also leverage information about client end system constraints and QoS requirements to deliver video of diverse quality to clients.

Despite these important advantages, we also face some unique challenges in the design of such a proxy video streaming system. Although a proxy server provides additional disk space for caching or storing videos, it is still limited with regard to the huge volume of videos. As a result, many videos (either in their entirety or in part) have

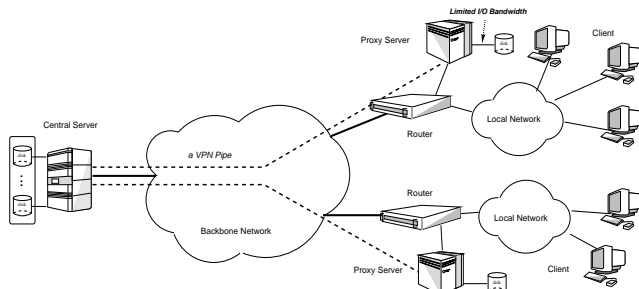


Fig. 1. The system architecture.

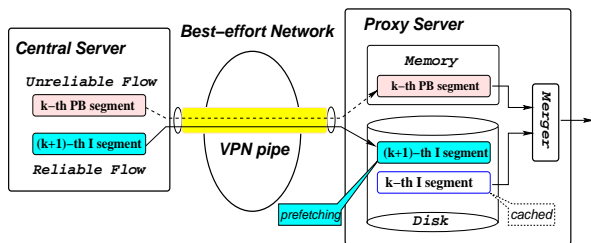


Fig. 2. Illustration of our staggered two-flow video streaming.

to be streamed across the wide-area best-effort network. Because of *stringent timing constraints*, it is important to ensure continuous playback of videos for end users so as to provide consistent and smooth quality. This problem is particularly challenging when videos are streamed across a wide-area *best-effort* network, where the availability of network resources often fluctuates. To deal with the problem, an overlay on a best-effort network referred to as a content distribution network employs a VPN pipe between a proxy server and a central server, which is built through leased lines, ATM/Frame Relay virtual circuits, or through a network-layer service agreement. In a typical VPN connection (e.g., a *hose* [6]), the *aggregate* bandwidth between the two servers is assured; however, for the delivery of *individual* videos (or packets), no fine-grain delay or bandwidth guarantee is provided due to the best-effort nature of the current Internet. Hence a key design issue in building a wide-area video streaming system is how to provide quality assurance on best-effort networks.

We develop a practical proxy-assisted *staggered two-flow video streaming technique* to meet the above constraints and provide controlled service assurance for individual videos delivered across a wide-area best-effort network. The proposed technique is developed for stored videos that have inherent priority structures (e.g., the inter-frame dependence). Utilizing the priority structure in a video, we partition the video into an essential part that is reliably prefetched via a VPN pipe and cached at a proxy server, and an enhanced part that is unreliably transmitted in real time via the same VPN pipe. To meet the timing constraint of the essential part, we pre-store its prefix on a proxy server [20]. Both flows are merged at the proxy server and then delivered to end users.

In this paper, we discuss several challenging issues which arise in the development of the proposed technique. In particular, we focus on a data plane issue, the bandwidth competition between the reliable transmission of the essential data and the unreliable, real time delivery of the enhanced data. For an essential data flow, we design a novel *controlled TCP (cTCP)* scheme (a variant of TCP) to support application-level rate control. For an enhanced data flow, we also build a simple *rate-control UDP (rUDP)* protocol to regulate the unreliable delivery. Combining cTCP and rUDP, we are able to control the interactions between the two flows in a session. As a result, our video streaming technique yields more *stable* and *predictable* performance which is critical in providing consistent and controlled video quality assurance to end users.

The performance predictability of our data plane mechanisms also enables us to provide a form of *application-aware* admission control and traffic management at the control plane. As a result, these control plane mechanisms help us ensure the requirement of the data plane wherein sufficient network resources are available for admitted video sessions.

The rest of the paper is organized as follows. We introduce related work in Section II, and present the problem setting and the proposed technique in Section III. Two application-aware transport protocols, cTCP and rUDP, are described in Section IV. The performance of the data plane schemes is evaluated through simulations and experiments in Section V.

## II. RELATED WORK

Various research has been done in proxy-assisted video streaming, application-level rate control, Streaming Control Transmission Protocol (SCTP) and TCP modeling, each of which addresses different perspectives. The distinguishing characteristic of our work is the ability to provide quality assurance.

In the area of proxy-assisted video streaming, Sen, et al. [20] and Wang, et al. [22] proposed to use the storage space at proxy servers to reduce the start-up latency and the network bandwidth requirement. We expand their approaches using a VPN model [6] on the network for providing quality assurance and exploits the priority structure in a video. Several other methods have been also proposed to exploit the priority structure of videos for dynamically adapting to the current network available bandwidth [7], [9], [12], [14], [19]. These studies mostly emphasize on the adaptation issue, instead of providing controlled quality assurance which is the goal of our approach. The receiver-based layered transmission was discussed in [12], [9] for video multicasting to adapt to the heterogeneity with coarse congestion control. Merz, et al. [14] proposed to transmit a media stream in several passes based on their significance to the stream and the available network bandwidth. Feng, et al. [7] used a multi-level priority queue in conjunction with a delivery window to help smooth the video frame rate transmitted to the end user while allowing it to adapt to the change of network conditions. The methods proposed in [14], [7] are highly dependent on the estimated available bandwidth in a delivery window period, which is still an unsolved issue on a wide-area best-effort network. Our technique addresses this issue by applying the VPN model which provides a coarse-grain aggregate bandwidth assurance. Rejaie, et al. [19] proposed an elegant network-layer solution which utilizes the client buffer and the instantaneous available network bandwidth at a fine time scale. However, its fine-grain nature makes it difficult to be scalable. Zhang, et al. [23] introduced selective frame discarding algorithms based on available network bandwidth and client buffer, plus application information (such as frame types). The assumption of guaranteed bandwidth in [23] requires high costs under the current network technologies.

Two related approaches in application-level rate control are TCP pacing and Congest Manager(CM) [3]. TCP pacing can be classified into two types. One is pacing on the sender side, which spreads the packet injection across the entire RTT. However, the study in [1] has shown that this pacing approach often has significantly worse throughput than regular TCP because it is susceptible to synchronized losses and it delays congestion signals. Besides, the sender-side pacing requires a large number of fine-grain timeouts, and it is not scalable. Another type of TCP pacing is smoothing the acknowledgments on a receiver or a router on the reverse path (e.g., *Packeteer*). It requires the protocol changes on receivers and *intermediate routers*. Pacing at a receiver also requires a high timing cost as at a sender. Pacing at routers is not practical for a wide-area network because it requires per-flow information on routers. This aggravates the loads on the routers which are already heavy-loaded. Besides, these routers belong to different management entities, making the coordination complicated and expensive. In addition, application-level pacing seems helpful, but it has large error terms and does not help in flow control without kernel level support. The CM is an end-to-end framework for congestion control/management and bandwidth sharing, independent of specific transport protocols and applications. It allows different application flows to adapt to network congestion and to share the network information. Rate control or quality assurance is not considered in CM.

SCTP (RFC 2960) addresses the issue of multi-streaming and multi-homing, but does not take into account application-level rate control as our cTCP does. Combining the ideas of stream aggregation in SCTP and application-aware flow control in cTCP is a very interesting open issue to be studied. The idea of end host congestion control in CM can also applied into this new model.

Our cTCP scheme applies the basic TCP model to achieve the application-level rate control. TCP modeling work can find at the TCP-Friendly web site [10]. Our approach is TCP-friendly because it only uses the resources

in the VPN setting and does not aggressively grab other resources outside the VPN.

### III. OUR STAGGERED TWO-FLOW VIDEO STREAMING TECHNIQUE

The *staggered two-flow video streaming* technique emphasizes controlled quality assurance for videos delivered in a content distribution network across a wide-area *best-effort* network. In such a content distribution network, the videos from a central server to a proxy server are transmitted across the wide-area network through a VPN pipe. (See Figure 1.) The aggregate bandwidth for the VPN pipe is assured. We also assume that this system delivers large stored MPEG videos; a client only contacts a proxy server; and a proxy-client path is well provisioned by an ISP. Therefore, we concentrate on the delivery quality on the central-proxy server path.

The basic ideas behind our technique are illustrated schematically in Figure 2. Each MPEG video is divided into two sub-streams (referred to as flows): one flow contains the essential **I** frames that are intra-frame coded; the other flow contains the less essential **P/B** frames which depend on the **I** frames (and possibly other **P** frames)<sup>1</sup>. As we will explain shortly, the **I** frame flow is transmitted *reliably* (using cTCP) across the best-effort network, hence it is called the *reliable flow*; the **P/B** frame flow is transmitted *unreliably* in real-time (using rUDP) across the network, hence it is called the *unreliable flow*. The data in both flows are partitioned into relatively large *video segments* with equal length (measured in time, in the multitude of minutes). To take advantage of the proxy storage space, the first segment of the reliable flow is staged (i.e., pre-stored) at the proxy server<sup>2</sup>. To reduce the start-up latency, we also pre-store a small prefix of the unreliable **P/B** frame flow at a proxy server [20].

When an end user requests a video, the first segment of the unreliable (**P/B** frame) flow is delivered unreliably in real time from the central server to the proxy server. As the **P/B** frames in this segment are received, the proxy server merges them with the appropriate **I** frames that are retrieved from the cache (memory or disk) where the first segment of the reliable flow is pre-stored. The merged video stream is then delivered from the proxy server to the end user. At the same time while the first segment of the unreliable flow is being transmitted, the second segment of the reliable flow is also being delivered from the central server to the proxy server and *cached* at the proxy cache (memory or disk), as it is not needed immediately. This process continues until the entire video is delivered to the end user. We see that the video segments of the reliable and unreliable flows of a video are delivered in a *staggered* manner: for  $k = 1, 2, \dots$ , the  $k_{th}$  video segment of the unreliable flow is transmitted at the same time as the  $(k + 1)_{th}$  video segment of the reliable flow is delivered from the central server to the proxy server. Note that since the reliable flow is delivered one segment ahead of time as it is needed, this provides us with sufficient time to recover any lost packets in the reliable flow during the period through (TCP) retransmission. In this way we ensure that all **I** frames are prefetched reliably from across the best-effort wide-area network.

Given the capacity of a VPN pipe and a proxy server, the proposed technique can achieve a flexible control of quality assurance. We can provide various service assurance by applying different partition schemes for diverse user requirements. For example, we show one way of partitioning a MPEG video in the above. However, we can also divide the same video in other manners, and apply different partition policies on videos with various popularity.

The proposed technique also poses several challenging issues. Note that since both the reliable flow and the unreliable flow of a video follow the same path, they potentially compete for the bandwidth along the path. It is

<sup>1</sup>Depending on the system resource configuration, we can also divide a video in other manners. For example, the reliable flow may contain not only the **I** frames but also some or all of the **P** frames, whereas the rest of the frames (all **B** frames and maybe some **P** frames) is transmitted as the unreliable flow.

<sup>2</sup>For simplifying the discussion, we assume to pre-store only the first segment of the reliable flow on a proxy server. How to determine the pre-stored prefix size of a video is another interesting question, depending on many factors such as its popularity, its bandwidth requirement, and the available storage space on a proxy server.

therefore important to control the interaction between the two flows, in particular, to reduce the retransmissions in the reliable flow and the packet losses experienced by the unreliable flow. This problem becomes especially acute when multiple videos are streamed through the same VPN pipe. We address this issue in Section IV, and introduce cTCP and rUDP for controlling a reliable flow and an unreliable flow respectively with application knowledge to solve the *blind* bandwidth competition.

#### IV. PROPOSED CONTROLLED DATA TRANSMISSION SCHEMES

In this section, we first present the cTCP scheme with application-aware rate control for transmitting the reliable flow of a video, and then we introduce rUDP for simply regulating the delivery of an unreliable flow as near CBR. The major objective in designing cTCP and rUDP is to attain some controllability and predictability in the data transmission. In particular, we want to exert some degree of control on the interaction of the reliable flows and unreliable flows of various video streams sharing the same VPN pipe, in order to provide consistent and controlled video quality assurance to end users.

##### A. The controlled TCP (cTCP)

In the proposed technique, the reliable flow of a video is transmitted at least one video segment *ahead of time*, i.e., the **I** frames in the current segment need to be streamed to an end user in the next segment period or later. Given that the video segment length is in the order of minutes, this allows for sufficient time to recover any lost packets during the transmission of the video segment across the best-effort network. This is in contrast to the transmission of the segments of the unreliable flow, which are delivered in *real time*<sup>3</sup>.

As discussed in Section II, existing approaches (e.g., TCP pacing) are not suitable to build such a scalable high-volume video delivery system. Therefore, we extend the TCP protocol with an application-level rate control mechanism to transmit the reliable flow. (This idea is also applicable to SCTP which is now under developing, to achieve application-level rate control.) *Note that if sufficient bandwidth along the VPN pipe is available*, each segment of the reliable flow can be delivered across the best-effort wide-area network before its deadline. However, directly applying TCP for transmitting the reliable flow has some *undesirable* effects. First, the greedy increase of the injection rate of a TCP flow causes unnecessary packet drops even given sufficient network resources. Assume the unreliable flow is an UDP flow of CBR, and the reliable flow is delivered using TCP. A TCP flow uses an additive increase and multiplicative decrease (AIMD) algorithm in its flow control, which attempts to maximize its throughput by injecting as many packets as the network allows, and enters into a steady state that oscillates with *periodic packet losses*. In the context of our setting, this *greedy* behavior unfortunately has an adverse effect on both the reliable flow and the unreliable flow. Even when the available bandwidth is sufficient for transmitting both flows during a video segment period, a TCP flow grabs more bandwidth than what it needs, transmitting its data *in a blast*. As a result, the unreliable flow suffers more packet losses, and the reliable flow suffers more retransmissions. This problem is further compounded when multiple video streams share the same VPN pipe. Furthermore, TCP flows tend to share the available bandwidth more or less equally, while the bandwidth requirements of the reliable flows of various video streams are different. As new video streams are initiated or existing video streams are terminated, the bandwidth shares of the TCP flows are also varied unnecessarily, independent of their actual bandwidth requirements.

To address these issues, we develop cTCP to control the bandwidth shares of the reliable flows. In the meantime, we also develop rUDP (see Section IV-B) to regulate an unreliable flow close to CBR. Combining cTCP and rUDP,

<sup>3</sup>In general a small start-up delay can be introduced so that the proxy server can smooth its transmission of the unreliable flow using its buffer space [21]. This, in particular, can be done when a small initial prefix of the first segment of the unreliable flow is cached/pre-stored at the proxy server [20]. In our study we assume this is the case.

we are able to reduce the number of retransmissions in reliable flows and the number of packet drops in unreliable flows, and provide more predictable overall system performance in the data transmission. We show that our scheme is effective *when the available aggregate bandwidth of the VPN pipe is larger than the total bandwidth requirement of the video streams currently being transmitted* in Section V. We ensure this condition through the control plane mechanisms.

The basic idea behind cTCP is the realization that in delivering each video segment in a reliable flow, only the amount of bandwidth that is sufficient to transmit the video segment<sup>4</sup> before its deadline is needed. More bandwidth for the reliable flow is not necessary, and may even be harmful to both flows. Hence, given a sufficient network bandwidth, we should limit the TCP injection rate for a video segment to what is needed. This leads to the concept of the *target rate* of a video segment: Given a video segment of length  $T$  (measured in seconds), let  $S$  be its data size (measured in bytes), then its target rate, denoted by *target*  $T_{cTCP}$ , is given by  $\frac{S}{T(1-\hat{p})}$  (bytes/sec), where  $\hat{p}$  is the cTCP retransmission threshold, e.g., 0.05. The factor  $1/(1-\hat{p})$  accounts for the potential bandwidth consumed by retransmissions in a network with a packet loss rate of at most  $\hat{p}$ . TCP uses a window-based flow control mechanism, where the number of outstanding packets that can be injected into the network is limited by a window  $W = \min(W_{cwnd}, W_{recv})$ , where  $W_{cwnd}$  is the congestion window size, and  $W_{recv}$  is the receiver window size. To limit the rate of a TCP connection, we let  $W = \min(W_{cwnd}, W_{recv}, W_{target})$ , where  $W_{target}$  is the target injection window size computed based on  $T_{cTCP}$ , RTT and packet loss information using a TCP throughput model [10]. Note that the receiver of the reliable flow is the proxy server, which is assumed to have sufficient receiving buffer space to accommodate the data of the reliable flow before writing the data into its storage. We can ignore  $W_{recv}$  from the above formula. Hence when  $W_{target} < W_{cwnd}$ , the rate of a TCP connection is limited by  $W_{target}$ , even though more packets can be injected into the network without causing congestion. (When  $W_{target} \geq W_{cwnd}$ , the rate is determined by the congestion window  $W_{cwnd}$ , as is in the regular TCP.)

We use a simple TCP throughput model [10] to estimate  $W_{target}$ . (More sophisticated TCP models [8], [11], [15] can be plugged in to further improve the accuracy.) It is well known that when there are small packet losses<sup>5</sup>, the steady state TCP rate is given approximately by the simple formula  $0.75 \cdot W \cdot MSS/RTT$ , where  $MSS$  is the TCP maximal segment size (in a packet), and  $RTT$  is the smoothed round trip time. When there is no packets lost, the TCP rate is roughly  $W \cdot MSS/RTT$ . Based on this model, we compute  $W_{target}$  from a given *target*  $T_{cTCP}$  as follows:  $W_{target} = (T_{cTCP} \cdot RTT)/(0.75 \cdot MSS)$ , if there are packet losses;  $W_{target} = (T_{cTCP} \cdot RTT)/MSS$ , otherwise. Note that in using the above model to compute  $W_{target}$ , we need to measure  $RTT$  and the packet losses. To take possible changes of  $RTT$  and packet losses into account, we adjust  $W_{target}$  periodically after each *adjustment interval*, during the transmission of a video segment of the reliable flow. Compared to  $RTT$  (usually 0.1 second or less), the adjustment interval (e.g., 8 seconds) is relatively larger, yielding more stable evolution of  $W_{target}$ . (See details in [24].)

### B. Simple Rate-controlled UDP (rUDP)

To control the delivery of an unreliable real time flow in a video session, we regulate it as a piecewise CBR traffic. In our simulation, we extend the CBR module in NS2 for building rUDP. In our experiments, we extend the standard UDP protocol on FreeBSD [16] with a simple periodical injection mechanism and a buffer to achieve this requirement. A fine-grain timer is used to periodically inject small bursts of UDP data from the buffer into the network. We combine the timer with a leaky-bucket regulator to limit the injection rate of a rUDP flow to a linear

<sup>4</sup>By *segment*, we mean a video segment which includes video frames for several hundred seconds (usually 100 KBytes or larger), different from a TCP segment which is a path MTU (1 KBytes or so).

<sup>5</sup>For quality and efficiency concern, on the control plane of the system, we manage traffic on the VPN pipe to achieve a low packet loss rate. So, the basic TCP formula is sufficient for testing purposes.

bound. The *target rate* of a rUDP flow is defined as the token rate of the leaky-bucket, which is a CBR. The burst size of a rUDP flow is determined by how many tokens accumulated after last timeout. The timeout granularity and the target rate can be dynamically adjust through the *setsockopt* system call. The buffer size is also set through the *setsockopt* system call. When the sender tries to write to a buffer which is full, it is blocked until the buffer becomes available again. We extend the IP Protocol Control Block to keep the state information of a rUDP flow.

Although rUDP is a simple scheme, our validation tests using *tcpdump* show that a rUDP flow on our FreeBSD implementation is very close to a CBR when the target rate is in the order of 10 KBps or higher. When the target rate is low as 1KBps, rUDP does not work well due to the OS scheduling and timing constraints. However, in our video streaming system, the target rate of a rUDP flow is usually in the order of 100 KBps. Therefore, the rUDP implementation is sufficient for our purpose. We also implement a RTP protocol [4] on rUDP for our streaming tests. Through this paper, a rUDP flow means a RTP flow over rUDP.

## V. SIMULATION AND EXPERIMENTAL EVALUATION

Assume all the control plane features are in place—sufficient network resources are always available for admitted video sessions, we concentrate on evaluating the proposed data plane schemes in this section. For convenience, throughout this subsection we refer to a reliable flow that is transmitted using cTCP as a cTCP flow, and a reliable flow that is transmitted using regular TCP as a TCP flow, an unreliable flow that is transferred using RTP/rUDP as a rUDP flow. A video stream that has a cTCP flow and a rUDP flow is referred to as a cTCP/rUDP session, and a video stream that has a TCP flow and a rUDP flow is referred to as a TCP/rUDP session. Through this comparative study, we demonstrate that: first, cTCP indeed provides us with the ability to control the bandwidth sharing among various reliable flows; second, combining cTCP and rUDP, we significantly reduce the number of packet retransmitted and dropped in both flows. As a result, it yields more *stable* and *predictable* performance that is critical in providing controlled video quality assurance to end users.

### A. Simulation Evaluation

We set up a simple simulation setting in Network Simulator (NS2), where a central server and a proxy server are connected by a link (VPN pipe) with a capacity of  $C$ . The actual value of  $C$  depends on a specific simulation scenario. The buffer size of the bottleneck link is determined based on the link capacity in such a manner that the maximum queueing delay is 50 ms. The propagation delay of the VPN pipe is set to 40 ms. The network MTU is set to be 1500 bytes. All data packets are assumed to be the same size, with a payload of 1460 bytes. The  $W_{target}$  adjustment interval used in cTCP is 8 seconds.

In the first set of simulations, we investigate the effectiveness of our application-level rate control in cTCP. Hence in this set of simulations we consider only reliable flows. The link capacity  $C$  is set to be 0.256 Mbps. Figure 3 shows the results where the VPN pipe is used to transmit the reliable flow of a video trace *Soccer* using either cTCP or regular TCP. The reliable flow has a target rate of 0.158Mbps. From the figure we see that the cTCP flow attains a stable rate close to its target rate, whereas the TCP flow grabs almost all the available bandwidth, attaining a rate close to the link capacity. (The x-axis in the plots of Figure 3 represents time, indexed by the adjustment interval numbers.) In another test two cTCP flows share the VPN pipe, with target rates of 0.158Mbps (from video trace *Soccer*) and 0.085Mbps (from video trace *Beauty and Beast* , respectively. The sum of the target rates is less than the bottleneck link capacity. In this case, we see that each cTCP flow attains a stable rate which is close to its target rate. This is opposed to the situation when the flows are transmitted using regular TCP: the bottleneck link capacity is shared equally between both flows, regardless of their requirements. (Similar to Figure 8(b).)

From the above results, we see that when the link capacity of the VPN pipe is larger than the total target bandwidth requirement of all the reliable flows, the cTCP scheme is effective in controlling the bandwidth sharing

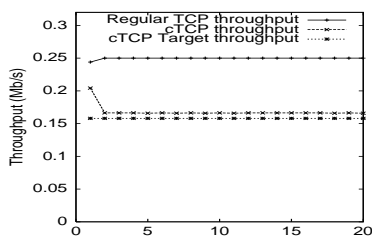


Fig. 3. Comparison of a cTCP flow and a TCP flow in simulation.

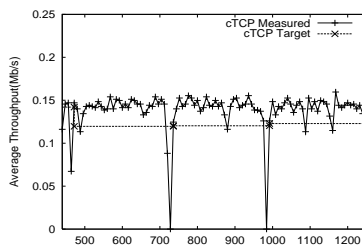


Fig. 4. Near-CBR Rates of cTCP flows when bandwidth is sufficient. (A point for each packet).

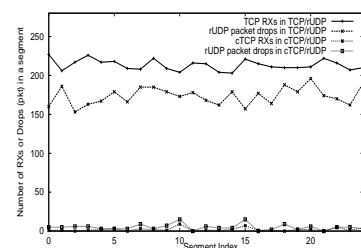


Fig. 5. Comparison of packet retransmissions and losses in a cTCP/rUDP session and a TCP/rUDP session.

among the reliable flows. We now turn our attention to the interaction between reliable flows and unreliable flows, in particular, the impact of cTCP or TCP flows on packet losses experienced by unreliable flows and packet retransmissions in the reliable flows. We still assume that *the aggregate bandwidth of the VPN pipe is sufficient to satisfy the total bandwidth requirement of all the video streams (including all reliable and unreliable flows) currently sharing the VPN pipe.*

We first show the results of delivering a single video stream across the VPN pipe using the proposed technique. The video trace used in this simulation is an approximately 100 minute long sequence from the MPEG-1 encoded *Star Wars*, with a frame rate of 24 frames/s and a GOP pattern of 12 frames (**IBBPBBPBBPBB**). The video stream is divided into a reliable flow (containing 12800 **I** frames) and an unreliable flow (containing 140800 **P/B** frames). Both flows are partitioned into segments with a length of 256 seconds. The *total* average bandwidth requirement of the two flows together is 0.495Mbps, the total maximum bandwidth requirement is 0.529 Mbps. In this simulation we assume that the bottleneck link capacity is set to 0.529 Mbps, i.e., equal to the total maximum bandwidth requirement of the two flows. We also assume that the unreliable flow starts 8 seconds (an adjustment interval) later than the reliable flow allowing cTCP to obtain the initial RTT measurement and set  $W_{target}$ . This delay in starting the rUDP flow can be masked by, for example, caching a prefix of the unreliable flow at the proxy server.

Figure V-A shows the *measured* rate (the y-axis) of the cTCP flow during the transmission of the third, fourth, and fifth segments of the video. We see that the cTCP flow meets the delivery deadline of each segment and attains a stable measured rate close to the target rate of each segment. In particular, the third, fourth, and fifth **I** segments are delivered, respectively, by the 720th, 967th, and 1248th second, all ahead of their respective deadlines (the 768th, 1024th, and 1280th second). Note that the measured cTCP rate dips at the end of each segment because the transmission of the segment is completed. The **P/B** segments of the rUDP flow are also delivered smoothly with only a few packet losses (as we will see shortly).

The bottom two lines in Figure 5 show the numbers of cTCP packet retransmissions as well as that of rUDP packet losses during the transmission of the entire session. The x-axis shows the video segment indexes. We see that the cTCP flow only experiences a small number of packet retransmissions in each video segment. The cTCP flow reaches a steady state with a stable  $W_{target}$  [24]. After that, the cTCP flow injects packets into the network at a steady pace. (Sharing the RTT from an existing connection between the server and the proxy can help the cTCP connection skip the slow-start learning curve and reach a steady state directly.) Because the cTCP flow grabs only the bandwidth it needs, the corresponding rUDP flow obtains sufficient bandwidth for its transmission. As a result, it experiences only a few packet losses (that are due to the limit of timer resolution in the RTT estimation). This is in contrast with the scenario where the reliable flow is transmitted using regular TCP, as is shown in the upper part of Figure 5. Due to the “greediness” of TCP, both the TCP flow and rUDP flow experience relatively large numbers of losses or retransmissions during the transmission of each segment.

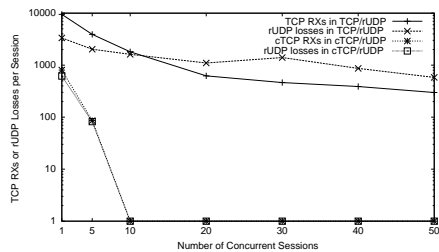


Fig. 6. Comparison of packet retransmissions and losses per session in multiple cTCP/rUDP sessions or multiple TCP/rUDP sessions.

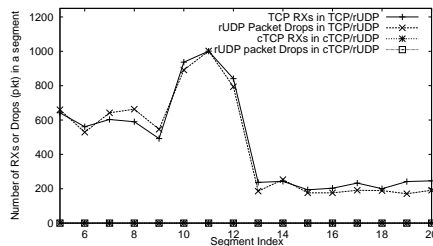


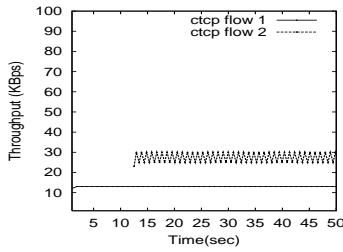
Fig. 7. Fluctuations of retransmissions and losses in a cTCP/rUDP and a TCP/rUDP on arrival/departures.

To further demonstrate the advantage of cTCP over regular TCP, we look at the packet losses experienced by both reliable flows and unreliable flows, when *multiple* video sessions of *Star Wars* are transmitted over a VPN pipe. In this set of simulations, the bottleneck link capacity is set to be the same as the total maximum bandwidth required by all concurrent sessions, and each video session starts randomly within a short period of time. Figure 6 clearly shows that, with sufficient bandwidth, a cTCP/rUDP session has no packet losses when the number of sessions is equal to or more than 10, while a TCP/rUDP session always has large numbers of TCP retransmissions and rUDP packet losses. Because of the controlled bandwidth sharing of the cTCP flows, there is always sufficient bandwidth for the rUDP flows to transmit their packets. Addition simulations (results not shown here, due to space limitation) we conducted show that, when the bottleneck link capacity of the VPN pipe is slightly higher than (e.g., 1.1 times or more) the total maximum video rate requirement, cTCP/rUDP sessions can achieve no packet drops and retransmissions in almost all cases, while TCP/rUDP sessions still have large numbers of packet drops and retransmissions.

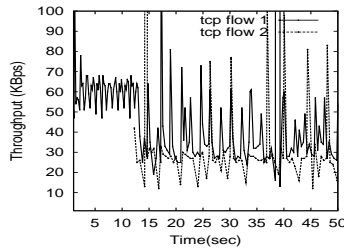
In the final set of simulations we show the impact of dynamic session arrivals and departures on a cTCP/rUDP (or TCP/rUDP) session. In these simulations we set the bottleneck link capacity of the VPN pipe to 1.1 times of what is needed to carry five concurrent video sessions of *Star Wars*. At the beginning, we have four on-going video sessions; then at segment 10 (i.e., after 2560 seconds), we start a new video session joining the four on-going video sessions; at segment 13 (i.e., after 3328 seconds), two video sessions are terminated. Figure 7 shows the impact of the session arrival and departures on the packet losses and retransmissions experienced by one of the on-going video sessions: the two curves at the bottom of Figure 7 are for a cTCP/rUDP session, while the upper two curves are for a TCP/rUDP session. We see that the dynamic session arrival and departures have no visible impact on the on-going cTCP/rUDP session in this case. In contrast, they have a strong impact on both packet retransmissions and packet drops in the TCP/rUDP session. This is because TCP always attempts to distribute the bandwidth equally among the TCP flows of the current on-going video sessions. When a new video session joins or an existing session leaves, the available bandwidth “has to” be redistributed among the TCP flows. This causes fluctuations of the packet retransmissions and losses experienced by the video sessions, which in return induce fluctuations in the video quality perceived by end users. Combining cTCP and rUDP, we are able to avoid the fluctuations, therefore providing more consistent video quality to end users.

### B. Implementation and Experimental Evaluation

We have implemented cTCP and rUDP by modifying the kernel source codes in FreeBSD 4.1 [16]. In our experiments, we use three Dell PCs on a dedicated 100-Mbps Ethernet switch: *oak*, *breeze* and *ivy*. We change the routing tables of *oak* and *ivy* to force them to send packets to *breeze* which runs *dummynet* as a bandwidth-delay control unit. *breeze* forwards the packets from *oak* to *ivy*, vice versa. We set up two dummynet pipes: pipe 1 is for



(a) Two cTCP flows.



(b) Two TCP flows.

Fig. 8. Comparison of transmission rates of two cTCP flows or two TCP flows on a fixed capacity link. (Each point is the mean rate of 10 packets.)

the path of (*oak*, *breeze*, *ivy*), and pipe 2 is for the return path.

We first compare the injection behavior of cTCP with that of TCP using two simple experiments. The two pipes are set to have a delay of 40 ms and a capacity of 64KBps. We use *tcpdump* for capturing the packet headers at *ivy*. Figure 8(a) shows the rate curves of two cTCP flows from *oak* to *ivy*, where each point in the curves is the mean rate of every 10 samples. The first flow has a target rate 13 KBps; the second flow starts at 12-second later with a target rate 27 KBps. Clearly, both cTCP rate curves are close to their target rates and relatively stable all the time. This result is consistent with our simulation results. In contrast, Figure 8(b) shows the rate curves of two regular TCP flows without bandwidth control, which transmit the same data. We can see that, during the first 12 seconds, TCP flow 1 grabs almost all the available bandwidth; after TCP flow 2 is started, the two TCP flows share all the available bandwidth but with huge fluctuations.

We now compare cTCP/rUDP sessions with TCP/rUDP sessions in a similar setting as the above. We use *oak* as a central server, *ivy* as a proxy server, and *breeze* still as a bandwidth-delay control unit regulating the traffic between *oak* and *ivy* with given propagation delay, link rate, and queue size. We set the pipe capacity to 1.1 times of the total requirement of all concurrent sessions, and set the queue size of the pipe to be the product of the link capacity and the estimated RTT (80 ms). We divide a MPEG clip *Evita* into an **I** frame flow with a target rate 52 KBps and a **P/B** frame flow with a target rate 200 KBps. In an experiment, multiple sessions are started within a short period, delivering the video from *oak* to *ivy* using cTCP/rUDP (or TCP/rUDP). A prefix of the video is pre-stored at the receiver *ivy*, and the remaining part is delivered from the sender *oak*. We observe the data transmission between the central server *oak* and the proxy server *ivy*.

We repeatedly run the experiments to figure out what is the prefix size of the reliable flow, which can guarantee that the **I** frames of the video are always on time. We compare the requirements of cTCP/rUDP and TCP/rUDP sessions as the number of concurrent sessions (the x-axis) is increased in Figure 9. The y-axis is the required prefix size (in KBytes) to guarantee that no **I** frames are late. We can see that a cTCP/rUDP session requires a much smaller prefix to be pre-stored on a proxy server, comparing with that of a TCP/rUDP session.

Using the same setting as the above, given a fixed prefix size of **I** frames (2 MBytes), we compare the retransmissions and the packet drops in two types of video sessions in another set of experiments. The x-axis of both Figure 10 and Figure 11 is the number of concurrent video sessions. Figure 10 shows the average bytes retransmitted by a cTCP or TCP flow in a session. Figure 11 shows the average number of packet drops experienced by a rUDP flow in a session. Clearly, a cTCP/rUDP session does much better than a TCP/rUDP session in both cases. However, different from the simulation results, we do see a moderate number of rUDP packet drops and a few cTCP retransmissions in our experiments. These drops and retransmission are caused by the bursts generated in

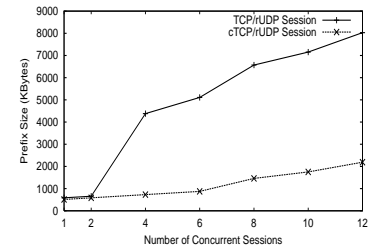


Fig. 9. Comparison of the requirements on prefix size in cTCP/rUDP sessions and TCP/rUDP sessions.

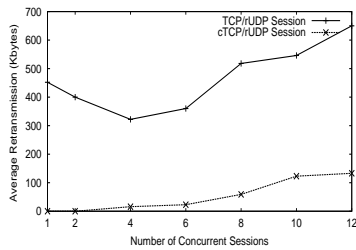


Fig. 10. Comparison of the average number of retransmissions in cTCP flows of cTCP/rUDP sessions or in TCP flows of TCP/rUDP sessions.

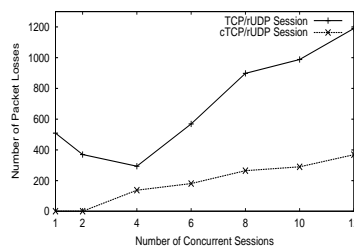


Fig. 11. Comparison of the average number of rUDP packet losses in rUDP flows of cTCP/rUDP sessions or TCP/rUDP sessions.

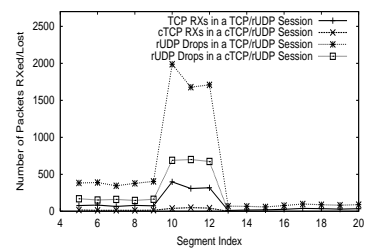


Fig. 12. Comparison of the average numbers of cTCP- (or TCP-) retransmissions and rUDP packet losses in a cTCP/rUDP session or a TCP/rUDP session on session arrival/departures.

our current rUDP implementation. We have a rUDP timer for each rUDP flow. When multiple rUDP flows exist, a timeouted rUDP flow may not get a chance to send data before its *next* timeout, due to the OS scheduling limitation (10ms in our setting). (In the simulation, no scheduling and context switching cost.) We are currently building a flow-aggregation mechanism to solve this issue. A rUDP timer is used for multiple rUDP flows. At a timeout, the data from multiple rUDP flows (which fall into a scheduling interval) is sent. This aggregation scheme will reduce the burst sizes of rUDP flows, and decrease the number of packets retransmitted and dropped in cTCP/rUDP sessions.

The last set of experiments is designed to compare the effect of session arrivals or departures on two types of sessions. Similar to the last set of simulations, the pipe capacity is set to 5.5 times of the target rate of video *Evita* (1.386MBps). We first have four sessions running, then start the fifth session, and terminate two sessions after three video segments. The X-axis of Figure 12 is the index of video segments, the Y-axis is the number of packet retransmitted or dropped in a session. The figure shows the similar trend as our simulation (See Figure 7). A cTCP/rUDP session has much smaller fluctuation compared with a TCP/rUDP session.

Several practical issues need to be further studied in our current implementation. For example, we did not evaluate the synchronization cost of merging a cTCP flow and a rUDP flow of a video session; we did not discuss the synchronization of an audio stream and a video stream in a presentation at a proxy server; we have not studied the effects of the constraints of a proxy server (such as I/O bandwidth, memory and CPU resources) on our system. Limited by the memory size and the OS scheduling on our PC, we did not test a large number of sessions in our current implementation. (For scalability concern, one lesson we learned is that a customized scheduler in the kernel must be built in order to support a large number of flows.)

## VI. CONCLUSIONS AND ON-GOING WORK

In this paper we have developed the staggered two-flow video streaming technique to provide controlled quality assurance in delivering videos across a wide-area best-effort network, by taking advantage of the priority structure in videos, the disk space at proxy servers and the coarse bandwidth guarantee of a VPN pipe. We have designed cTCP and rUDP with application-level rate control for reliable and timely delivery of the essential data and the enhanced data. Through simulations and experiments, we have illustrated that the proposed technique yields a *stable* and *predictable* performance which is critical in providing consistent and controlled video quality assurance to end users.

We have addressed the bandwidth competition issue in the data plane. On the control plane, many other interesting ideas also need to be investigated. First, the performance predictability of cTCP and rUDP provides us the

chance of performing a form of *application-aware admission control* at either a proxy server or a central server. Our initial study of this issue is presented in [5]. How to improve the utilization of the VPN when the VPN is light-loaded is another interesting issue. Furthermore, other interesting issues needed to be studied, such as rate adaptation schemes when congestion happens (the VPN does not provide a hard guarantee), caching schemes to support diverse video streaming features (such as VCR, multicasting, and broadcasting), and more difficult issues in a hierarchy proxy server system.

## REFERENCES

- [1] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the Performance of TCP Pacing" in Proc. of IEEE Infocom 00, Tel-Aviv, Israel, Mar. 2000.
- [2] E. Amir, S. McCanne, and H. Zhang, "An Application Level Video Gateway," in Proc. of ACM Multimedia'95.
- [3] MIT CM: The Congestion Manager, <http://nms.lcs.mit.edu/projects/cm/>.
- [4] V. Deshmukh and E. Kumar, et al., "The Design and Implementation of RTP," Technical Report at CMNRG, Computer Science, U of Minnesota, 2000.
- [5] Y. Dong, "Building Service-Oriented Networks with Quality Assurance across the Best-Effort Internet", PhD Thesis Proposal, UMN.
- [6] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K.K. Ramakrishnan, and J. Merwe, "A Flexible Model for Resource Management in Virtual Private Networks", in Proc. of ACM SIGCOMM'99.
- [7] W. Feng, M. Liu, B. Krishnaswami, and A. Prabhudev, "A Priority-Based Technique for the Delivery of Stored Video Across Best-Effort Networks," in Proc IS&T/SPIE Multimedia Computing/ Networking'99.
- [8] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications," in Proc. of SIGCOMM'00.
- [9] X. Li, M. Ammar, and S. Paul, "Layered Video Multicast with Retransmission(LVMR): Evaluation of Hierarchical Rate Control," in Proc. of IEEE INFOCOM'98.
- [10] J. Mahdavi and S. Floyd, "TCP-friendly Unicast Rate-based Flow Control," [http://www.psc.edu/networking/papers/tcp\\_friendly.html](http://www.psc.edu/networking/papers/tcp_friendly.html).
- [11] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithms," Computer Communication Review, Vol.27(3), Jul. 1997. pp.67-82.
- [12] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven Layered Multicast," in Proc. of ACM SIGCOMM'96.
- [13] J. McManus and K. Ross, "Video-on-demand over ATM: Constant-rate Transmission and Transport," IEEE J. Selected Areas in Communications, Vol.14(6), pp.1087-1098, Aug. 1996.
- [14] M. Merz, K. Froizheim, P. Schulthess, and H. Wolf, "Iterative Transmission of Media Streams," in Proc of ACM Multimedia'97.
- [15] J. Padhye, *Model-based Approach to TCP-friendly Congestion Control*, Ph.D thesis, Univ. of Massachusetts at Amherst, 2000.
- [16] R. Rakshe, *Prototyping the Staggered Two-Flow Scheme*, MS thesis, CS, U of Minnesota.
- [17] R. Katz, "The Post-PC Era: It's All About the New Services-Enabled Internet", <http://www.cs.berkeley.edu/~randy/Talks/PostPC.ppt>.
- [18] R. Rejaie, H. Yu, M. Handely, and D. Estrin, "Multimedia Proxy Caching Mechanism for Quality Adaptive Streaming Applications in the Internet," Technical report 99-709, Computer Science Department, USC.
- [19] R. Rejaie, M. Handely, and D. Estrin, "Quality Adaptation for Congestion Controlled Video Playback over the Internet," in Proc of SIGCOMM'99.
- [20] S. Sen, J. Rexford, and D. Towsley, "Proxy Prefix Caching for Multimedia Streams," in Proc. of IEEE INFOCOM'99.
- [21] J. Salehi, Z.-L. Zhang, D. Towsley, and J. Kurose, "Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements Through Optimal Smoothing," in Proc. of ACM SIGMETRICS'96.
- [22] Y. Wang, Z.-L. Zhang, D. Du, and D. Su, "A Network Conscious Approach to End-to-End Video Delivery Over Wide Area Networks Using Proxy Servers," in Proc. of IEEE INFOCOM'98.
- [23] Z.-L. Zhang, S. Nelakuditi, R. Aggarwal, and R. Tsang, "Efficient Selective Frame Discard Algorithms for Stored Video Delivery across Resource Constrained Networks," INFOCOM'99. in Proc. of IEEE INFOCOM'99, and Journal of Real-Time Imaging, 2000.
- [24] Z.-L. Zhang and Y. Dong, "Video Streaming across Best-Effort Wide-Area Networks with Controlled Quality Assurance using Proxy Servers," Tech-Report-00-061, CS, U. of Minnesota.