Providing Controlled Quality Assurance for Streaming Stored-Videos across the Internet Using VPNs *

Yingfei Dong and Zhi-Li Zhang
Dept. of Computer Science and Engineering,
University of Minnesota, Minneapolis, MN 55455.
TEL:621-626-7526, FAX:612-625-0572.
Email:{dong,zhzhang}@cs.umn.edu.

Abstract. Video Streaming across wide-area networks is one of the most important applications on the Internet. 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 through two separate flows in a staggered fashion via a VPN pipe from a central server to a proxy server. One flow containing the essential portion of the video is delivered using a novel *controlled TCP (cTCP)*, and the other flow containing the enhanced portion of the video is transmitted using a *rate-controlled* RTP/UDP (rUDP). To provide video-quality assurance in such a system, we design several application-aware flow-control and adaptation approaches to control bandwidth sharing and interactions among flows by exploiting the inherent priority structure in 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 in protecting essential data and significantly reducing the numbers of packets retransmitted/lost in transmission and the sizes of video prefixes required on proxy servers. In summary, our application-aware approach provides stable and predictable performance in streaming videos across wide-area best-effort networks. In addition, another salient feature of our approach is that it requires no changes on the client-receiving side and minimal changes on the server-sending side.

Keywords: Quality Assurance, Multimedia Content Distribution, Application-aware Flow Control, Proxy Server, Video Streaming

* This work was supported in part by the National Science Foundation under the grants EIA-9818338, ANI-9903228, ITR-0085824, and CAREER Award NCR-9734428 as well as a University of Minnesota McKnight Land-Grant Professorship. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

1. 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, 23, 22, 24, 25] 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 servers that are strategically placed at the boundaries between wide-area networks (e.g., the Internet) and local access networks are employed to assist in the delivery of stored videos from central video servers to a large population of geographically dispersed end users in a scalable manner. 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 wide-area backbone 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

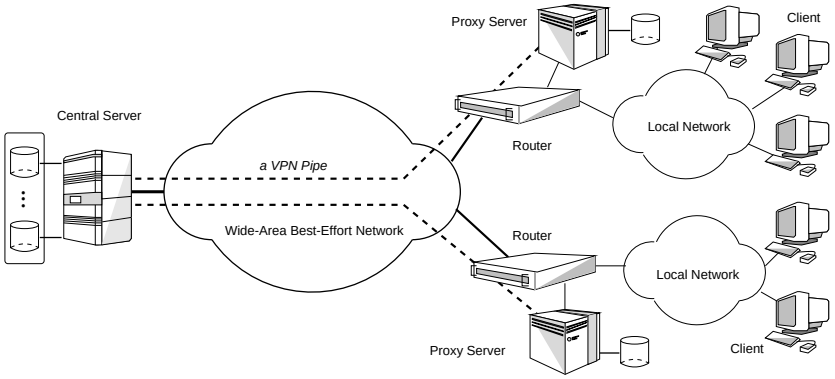


Figure 1. Common architecture of a wide-area streaming system.

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, we cannot indiscriminately pre-store or cache entire videos at proxy servers. In other words, many videos (either in their entirety or in part) have to be streamed across the wide-area best-effort network from a central video server. Because of the *stringent timing constraints* in video streaming, 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 in relatively large time-scales (e.g., hours or minutes) and no 100%-guarantee in small time scales (e.g., seconds). However, for the delivery of *individual* packets, no fine-grain delay or bandwidth guarantee is provided. This is the definition of *best-effort* in our setting. Hence a key design issue in building a wide-area video streaming system is how to provide quality assurance on best-effort networks. In this paper, we focus on the controlled quality assurance issue and develop a novel technique by utilizing overlay networks to solve the problem.

We develop a practical proxy-assisted *staggered two-flow video streaming technique* to meet the above constraints, such as the stringent timing requirements of videos, the limited caching space on proxy servers and the aggregate-only bandwidth guarantee of a VPN pipe. The proposed approach provides controlled service assurance for individual videos delivered across a wide-area best-effort network, by exploiting the inherent priority structure in videos, the disk space at proxy servers and the coarse bandwidth guarantee on an overlay network. The proposed technique is developed for stored videos that have inherent priority structures, e.g., the inter-frame dependence in MPEG. Utilizing the priority structure in a video, we partition the video into an essential part, e.g., **I** frames of a MPEG video, which is reliably prefetched via a VPN pipe and cached at a proxy server, and an enhanced part, e.g., the remaining **P/B** frames of the same video, which is unreliably transmitted in real time via the same VPN pipe. To meet the timing

constraint of the essential part, we pre-store a prefix of the essential part on a proxy server [22, 24] and prefetch its remaining on-demand using a reliable flow. The enhanced part is delivered using an unreliable flow which is adjusted based on the available bandwidth of the pipe and the requirement of the reliable flow. The two flows of a session are merged at a proxy server and then delivered to end users.

In this paper, we discuss several challenging issues that 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. This problem affects the performance of our video streaming technique. Because the reliable flow and the unreliable flow of a video both follow exactly the same path in a VPN pipe, they compete for the bandwidth along the path resulting in unnecessary packet losses even when bandwidth resources are sufficient. Such competition degrades the service of both flows and wastes network resources. To solve this problem, 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, in particular, to minimize the impact of the reliable flow transmission on the packet losses experienced by the unreliable, real time RTP/UDP flow. As a result, our video streaming technique yields more *stable* and *predictable* performance that is critical in providing consistent and controlled video quality assurance to end users.

As will be discussed in Section 6, 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 of the streaming system to manage the traffic in the VPN. These control plane mechanisms help us ensure the requirement of the data plane wherein sufficient network resources are available for the essential data of admitted video sessions.

The remainder of the paper is organized as follows. We discuss the related work in Section 2, and present the problem setting and the proposed technique in Section 3. Two application-aware transport protocols, cTCP and rUDP, are described in Section 4. The performance of the data plane schemes is evaluated through simulations and experiments in Section 5. We discuss the control plane issues in Section 6, and conclude this paper in Section 7.

2. Related Work

Various studies have addressed different perspectives in the areas of proxy-assisted video streaming, application-level rate control, multi-flow streaming and TCP modeling, each of which addresses different perspectives. The distinguishing characteristic of our work is the capability of providing quality assurance and demanding no changes in current client software and little changes in server software.

In the area of proxy-assisted video streaming, Sen, et al. [22] and Zhang, et al. [24] 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, 25]. These studies mostly emphasize the adaptation issue, instead of providing controlled quality assurance that 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 that provides a coarse-grain aggregate bandwidth assurance. Rejaie, et al. [19] proposed an elegant network-layer solution that 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 due to costs. Zhang, et al. [25] 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 [25] requires high costs under the current network technologies.

One close related approach in application-level rate control is TCP pacing. TCP pacing can be classified into two types. One is pacing on the sender side, which spreads the packet injection across an 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 at the server, proportional to the

number of TCP segments (packets). As a result, it is not scalable as the server need to support a large number of video sessions. 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 a receiver or an intermediate router. 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 that are already heavy-loaded. Besides, because these routers belong to different administrative entities, the coordination is complicated and expensive.

In addition, application-level pacing seems helpful but it tends to generate large packet bursts and does not help in flow control due to the lack of kernel-level support. First, application-level pacing has to feed large data chunks to TCP because the resolutions of application timers are in tens of milliseconds. Consequently, the greedy nature in TCP flow control still results in injection bursts and causes unnecessary bandwidth competition and packet losses on a link because TCP uses packet losses as signals to control a flow. When multiple such TCP flows concurrently, the situation becomes even worse, as shown in Figure 9.b. Furthermore, the process context-switching time in an OS is another factor that affects the data pumping from an application to TCP. Even if an application can pace video segments at a desirable rate to the transport layer, we still do not have control over the TCP injection bursts. In addition, in delivery of data chunks from an application, the slow-start flow control in TCP under-utilizes the bandwidth during some periods. In summary, it is almost impossible to control these fluctuations using application-level pacing.

Another related approach in application-level rate control is Congest Manager(CM) [3]. 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.

To address the issue of multi-streaming and multi-homing, Streaming Control Transmission Protocol (SCTP) (RFC 2960) is proposed as a reliable transport protocol that ensures no packet losses and in-order delivery. However, it does not take into account application-aware rate control as our cTCP does. The major differences between SCTP and our schemes are as follows. First, SCTP has a similar flow control scheme as TCP and does not provide application-aware rate control for our VPN setting. The behavior of SCTP is similar to that of TCP shown in Section 5. Second, SCTP is not able to adjust the selective delivery of flows based on the current system resources because SCTP guarantees the delivery of packets to receivers, which is not necessary in some settings. In our setting, we require a more flexible scheme that can sacrifice partial non-essential data in order to provide sufficient bandwidth for the essential data when

bandwidth fluctuations occur in a VPN pipe. SCTP does not provide this flexibility. Lastly, using SCTP requires to completely replace both client software and server software. Our schemes requires no changes on the client side and only a small patch on the server side, which is more practical and easier to be adopted. However, combining the ideas of stream aggregation in SCTP and application-aware flow control in cTCP is a very interesting 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 be found 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 pipe.

Another major advantage of our framework is the capability of supporting the adaptation of rUDP flows based on system resources as presented in Section 6. When the bandwidth of VPN pipe is below than the contracted value or ongoing sessions generate a burst in certain period of time, our framework is able to adapt to the fluctuation. While all the previous work discussed above cannot achieve this gracefully under this setting. In addition, our approach requires no changes on the client-receiving side and minimal changes on the server-sending side, which is a more practical approach than the previous approaches that require changes in both client and server software, sometimes even router software.

3. Our Staggered Two-Flow Video Streaming Technique

In this section we present the basic idea of our approach and discuss the key issues in its development. 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. By best-effort, we mean that the wide-area network service provider does *not* provide any packet delay or loss guarantee for the delivery of videos across the 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 as shown in Figure 1. The aggregate bandwidth for the VPN pipe[1] is assured through the service-level agreement between the network service provider and the owner of the video delivering system. The proposed technique is developed for stored videos that are compressed using encoding algorithms with inter-frame dependence. We also assume

[1] Such aggregate bandwidth guarantee can be provisioned, for example, in today's networks in a variety of manners such as leased line, ATM/Frame Relay virtual circuit, MPLS, or (statically configured) weighted fair queueing mechanisms.

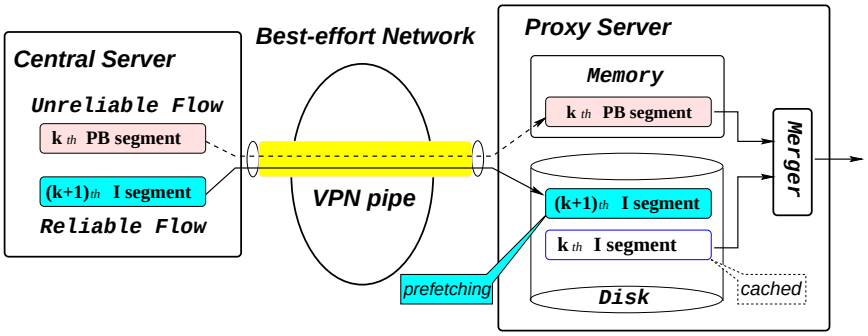


Figure 2. The illustration of our staggered two-flow video streaming.

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 Internet Service Provider for service quality. Consequently, we concentrate on the delivery quality on the path from a central server to a proxy server. In this paper we use MPEG encoded video traces to illustrate how our proposed technique works.

The basic ideas behind the staggered two-flow streaming 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 that depend on the **I** frames (and possibly other **P** frames)[2]. 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[3].

When an end user requests a video at a proxy server, the first segment of the unreliable (**P**/**B** frame) flow is delivered unreliably in real-time from a 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

[2] 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) are transmitted as the unreliable flow.

[3] To simplify the discussion, we assume that only the first segment of the reliable flow is pre-stored on a proxy server. How to determine the pre-stored prefix size of a video is another interesting question, which depends on many factors such as the popularity the bandwidth requirement of a video, as well as the available storage space on a proxy server. In addition, to reduce the start-up latency, we also assume that a small prefix of the unreliable **P**/**B** frame flow is pre-stored at a proxy server [22].

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 from the central server to the proxy server, 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 $k = 1, 2, \ldots$, 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. This is one of the key features of the proposed technique that enables us to provide a minimal video quality assurance to end users. This is in contrast to the transmission of the segments of the unreliable flow, which are delivered in *real-time*[4].

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 have shown one way of partitioning a MPEG video in the above. We can also divide the same video in other manners, such as allowing the reliable flow to contain not only **I** frames but also some or all of **P** frames for a higher quality assurance, whereas the rest frames in the unreliable flow. Moreover, we can apply different partition policies on videos with various popularities.

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 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 due to competitions. This problem becomes especially acute when multiple videos are streamed through the same VPN pipe. We address this issue in Section 4, and introduce cTCP and rUDP for controlling a reliable

[4] 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 [23]. In particular this can be done when a small initial prefix of the first segment of the unreliable flow is cached/pre-stored at the proxy server [22]. In our study we assume this is the case.

flow and an unreliable flow with application bandwidth requirements, respectively, so as to avoid *blind* bandwidth competitions as in using regular TCP or UDP to deliver the two flows.

4. 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 a near Constant-Bit-Rate (CBR) stream. The main objective in designing cTCP and rUDP is to attain some controllability and predictability in 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 sessions sharing the same VPN pipe, so as to provide consistent and controlled video quality assurance to end users.

4.1. The controlled TCP (cTCP)

As discussed in Section 2, 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 a reliable flow[5]. 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, i.e., the end of the segment. However, directly applying TCP for transmitting the reliable flow has some *undesirable* effects on both flows. First, the greedy increase of the injection rate of a TCP flow causes unnecessary packet drops even when sufficient network resources are given. Recall that TCP employs a *slow-start* mechanism to explore available network bandwidth in the beginning of a delivery period, and uses the additive increase and multiplicative decrease (AIMD) algorithm for flow/congestion control. Assume the unreliable flow is an UDP CBR flow, and the reliable flow is delivered using TCP. The TCP flow under the AIMD algorithm 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 is needed, transmitting its data *in a blast*. As the result of this competition, the unreliable flow suffers more packet losses and the reliable

[5] This idea is also applicable to SCTP, which is now under development, to achieve application-level rate control.

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 fluctuated correspondingly, independent of their actual bandwidth requirements.

To address these issues, we develop cTCP to control the bandwidth shares of reliable flows. The basic idea behind cTCP is the realization that in delivering each video segment in a reliable flow, we only need the amount of bandwidth that is sufficient to transmit the video segment[6] before its deadline. More bandwidth for the reliable flow is not necessary, and may even be harmful to both flows in a session. Hence, given a sufficient amount of bandwidth[7], 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 as *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 disk. 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.

[6] 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).

[7] In Section 6 we will discuss how we ensure this condition to hold by performing application-aware admission control at either a proxy server or a central server. In Section 5 we will show the basic idea of adaptation for the situation when this condition is not met.

```
// At the end of each adjustment interval,
// p is the reference to the control block.
// Choose the constant based on loss status.
loss_ = (p->curr_pkt_loss) ? 75 : 100;

// Reset packet loss for next interval.
p->curr_pkt_loss = 0;

// Compute the target window.
p->target_win = p->target_rate * 100 *
  (p->ctcp_srtt >> cTCP_RTT_SHIFT) / (loss_ * hz);
```

Figure 3. Codes for cTCP target window adjustment in *tcp_timer.c* in our FreeBSD implementation.

We use a simple TCP throughput model [10] to estimate W_{target}. It is well known [10] that when there are small packet losses[8], 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 are no packet losses, 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 around 100ms), the adjustment interval (e.g., 8 seconds) is fairly larger, yielding the stable evolution of W_{target}. Figure 3 shows a piece of adjustment codes in our cTCP implementation on FreeBSD.

4.2. 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 flow. 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

[8] 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. Hence the basic TCP formula is sufficient for testing purposes. More sophisticated TCP models [8, 11, 15] can be plugged in to further improve the accuracy.

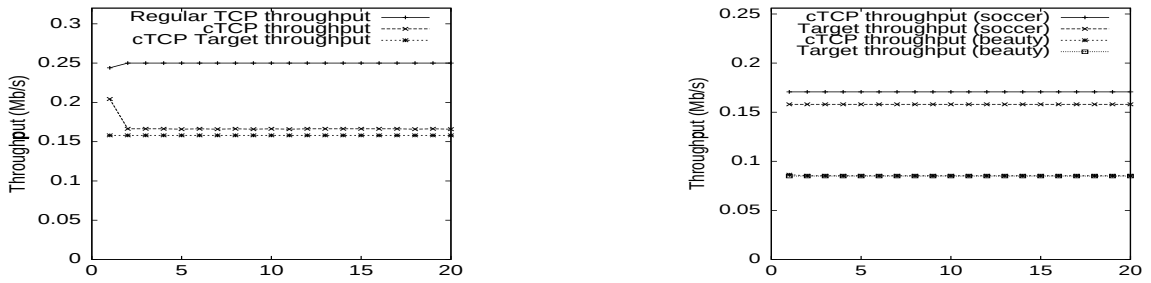
injection rate of an rUDP flow to a linear bound. The token rate of the leaky-bucket is set as the *target rate* of an rUDP flow, which is CBR. The burst size of an rUDP flow is determined by how many tokens accumulated since the last timeout. The timeout granularity and the target rate can be dynamically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

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

Combining cTCP and rUDP, 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 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 video streams currently being transmitted in Section 5. We ensure this condition through the control plane mechanisms described in Section 6.

5. Simulation and Experimental Evaluation

In this section we assume that all the control plane features are in place, i.e., sufficient network resources are always available for the essential parts of admitted video sessions. Under this assumption, we concentrate on evaluating the proposed data plane schemes. For convenience, throughout this section 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 our RTP implementation over rUDP as an rUDP flow. A video stream that has a cTCP flow and an rUDP flow is referred to as a cTCP/rUDP session, and a video stream that has a TCP flow and an rUDP flow is referred to as a TCP/rUDP session. Through this comparative study, we demonstrate that: 1) cTCP indeed provides us with the ability to control the bandwidth sharing among reliable flows; 2) combining cTCP and rUDP, we significantly reduce



a. A single cTCP/TCP flow.

b. Two cTCP flows close to target rates.

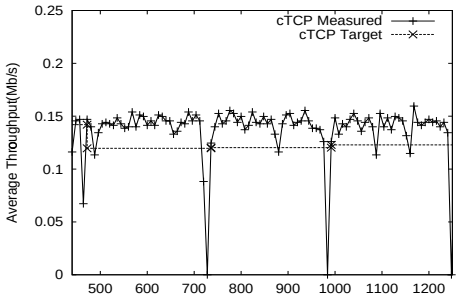
Figure 4. Throughputs of cTCP/TCP flows in simulation.

the number of packets retransmitted and dropped in both flows. Therefore, our approach yields more *stable* and *predictable* performance that is critical in providing controlled video quality assurance to end users.

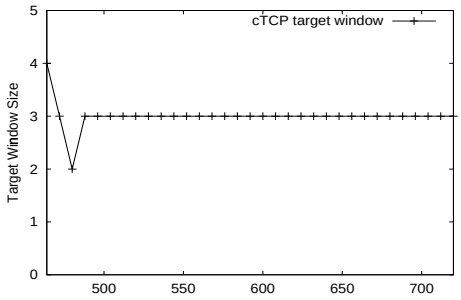
5.1. Simulation Evaluation

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 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 target-window 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 0.256 Mbps. Figure 4.a shows the rates when a cTCP or a TCP flow is used to transmit the reliable flow of a video trace *Soccer* [21] in the VPN pipe. The x-axis represents time indexed by the adjustment intervals. The reliable flow has a target rate of 0.158Mbps. The figure shows 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. Figure 4.b shows the rates when 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* [21]), 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. The results are similar to the two flows shown in Figure 9.b.



a. Near-CBR Rates (Each point represents a packet).



b. Stable Target Window.

Figure 5. The Behavior of cTCP in Simulation.

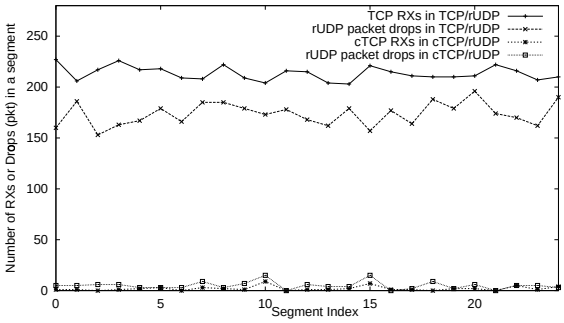


Figure 6. Comparison of packet retransmissions and losses in a cTCP/rUDP session or a TCP/rUDP session.

The above results show 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 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* [21], 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 is masked by caching a small prefix of the unreliable flow at the proxy server.

Figure 5.a shows the *measured* rate 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 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 6 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 represents the index of video segment. 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} as shown in Figure 5.b. After that, the cTCP flow injects packets into the network at a steady pace. We note that obtaining an RTT estimation from an existing connection between the server and the proxy can help a new cTCP connection skip the slow-start learning curve and directly reach a steady state. 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 due to the limit of timer resolution used 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 6. Due to the greedy nature of TCP, both the TCP flow and the rUDP flow experience rather large numbers of losses or retransmissions during the transmission of each segment.

To further demonstrate the advantage of cTCP over regular TCP, we examine 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 7 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

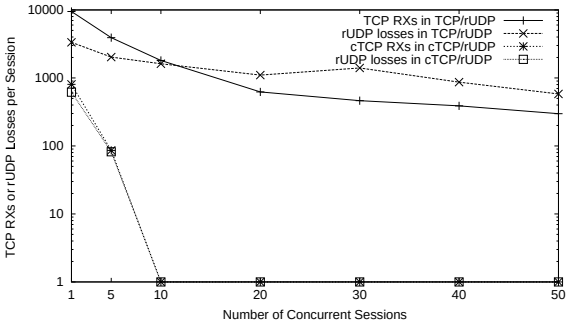


Figure 7. Comparison of packet retransmissions and losses per session in multiple cTCP/rUDP sessions or TCP/rUDP sessions.

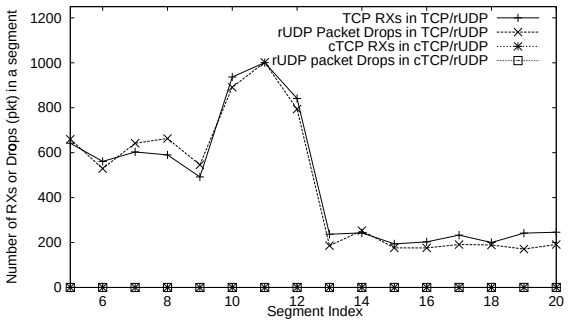


Figure 8. Fluctuations of packet retransmissions and losses in a cTCP/rUDP session or a TCP/rUDP session at session arrival/departures.

has large numbers of TCP retransmissions and rUDP packet losses[9]. Because of the controlled bandwidth sharing of the cTCP flows, there is always sufficient bandwidth for the rUDP flows to transmit their packets. Additional results from other simulations also show that, when the bottleneck link capacity of the VPN pipe is slightly higher than 1.1 times (or more) of 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.

We now investigate the potential effect of rUDP packet drops on the user perceived video quality. For this purpose, we introduce a few metrics. For a given video session, let E_f denote the number of **P**/**B** frames that are affected by packet losses (i.e., at least one packet of the frame is lost during the transmission)[10], and E_g is the number of GOPs that have at least one affected **P**/**B** frame. Moreover, we use G_{avg} and G_{max} to denote, respectively, the average number of affected **P**/**B** frames per affected GOP, and the maximum number of affected **P**/**B** frames per affected GOP, computed among all affected

[9] The packet retransmissions and losses under a small number of sessions are mainly due to the resolution of timers, which causes small bursts in transmissions. When the pipe capacity is relatively low, these bursts cause the packet retransmission and losses.

[10] Since a **P**/**B** frame is small, a packet loss basically destroys the frame.

Table I. Affected **P**/**B** frames of an rUDP flow in a TCP/rUDP session.

n	E_f	E_g	G_{max}	G_{avg}	G^{con}_{total}	G^{con}_{max}	G^{con}_{avg}
1	2862	2365	5	1.2	139	3	2.0
5	4856	3799	5	1.3	282	3	2.1
10	6501	4605	5	1.4	519	4	2.1
20	8971	5594	7	1.6	848	5	2.1
50	6196	4530	6	1.4	379	5	2.1

Table II. Affected **P**/**B** frames of an rUDP flow in a cTCP/rUDP session.

n	E_f	E_g	G_{max}	G_{avg}	G^{con}_{total}	G^{con}_{max}	G^{con}_{avg}
1	617	541	2	1.0	9	2	1.0
5	356	380	2	1.0	5	2	1.0
10	0	0	0	0	0	0	0.0
20	0	0	0	0	0	0	0.0
50	0	0	0	0	0	0	0.0

GOPs. Similarly, we use G^{con}_{avg} and G^{con}_{max} to denote, respectively, the average number of *consecutive* affected **P**/**B** frames per affected GOP, and the maximum number of *consecutive* affected **P**/**B** frames per affected GOP, again computed among all affected GOPs. G^{con}_{total} denotes the total number of *consecutive* affected **P**/**B** frames across all affected GOPs. These metrics for the regular TCP/rUDP video sessions are shown in Table I, where the results are obtained by averaging over all the TCP/rUDP sessions, where n is the number of sessions. In contrast, the corresponding results of cTCP/rUDP video sessions are shown in Table II. Clearly, cTCP/rUDP sessions outperform TCP/rUDP sessions in each category, especially in the total number of consecutive affected frames and the maximum number of consecutive affected frames which potentially cause the worst damage to playback quality. Again, given a little higher bottleneck capacity on the VPN pipe, cTCP/rUDP sessions can achieve no packet drops or retransmissions. All the entries in Table II will be zero. Under the same conditions, TCP/rUDP sessions will have similar behavior as shown in Table I.

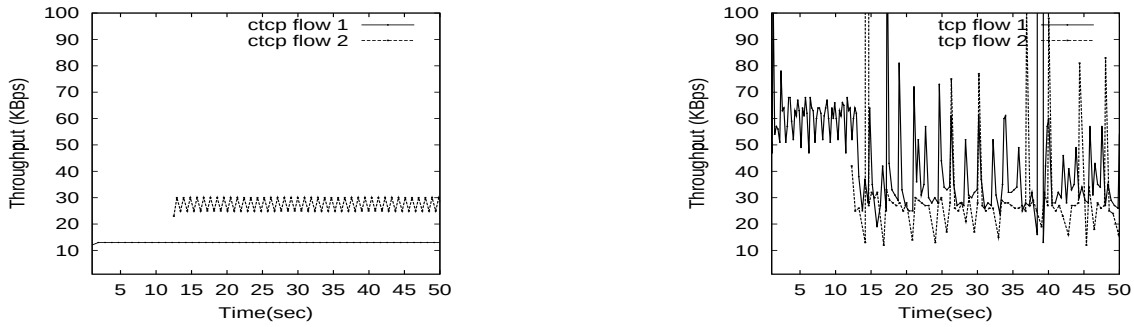
In the final set of simulations we show the effect of dynamic session arrivals and departures on a cTCP/rUDP or a TCP/rUDP session. In these simulations we set the bottleneck link capacity of the VPN pipe to 1.1 times of what is needed to carry five concurrent video sessions of *Star Wars*. At the beginning, we have four ongoing video sessions; then at video segment 10 (i.e., after 2560 seconds), we start a new video session to join the four on-going video sessions; at segment 13 (i.e., after 3328 seconds), two video sessions are terminated. Figure 8 shows the impact of the session arrivals and departures on the packet losses and retransmissions experienced by a session: the bottom two curves in Figure 8 represents for a cTCP/rUDP

session, while the upper two curves represents for a TCP/rUDP session. Clearly, the dynamic session arrivals and departures have no visible impact on the ongoing cTCP/rUDP session in this case. In contrast, they have a strong impact on both packet retransmissions and drops in the TCP/rUDP session. This is because TCP always attempts to distribute the bandwidth equally among the TCP flows of the current ongoing sessions. When a new video session joins or an existing session leaves, the available bandwidth *has to* be redistributed among the TCP flows. These fluctuations cause the packet retransmissions and drops experienced by the video sessions, and therefore 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.

5.2. Implementation and Experimental Evaluation

We have implemented cTCP and rUDP in the kernel of FreeBSD 4.1 [16]. All the changes are made in kernel files, *netinet/tcp.h, tcp_var.h, tcp_timer.h, tcp_timer.c, tcp_input.c, tcp_output.c, tcp_subr.c* and *tcp_usrreq.c*. To implement cTCP, we add a *cTCP timer* that is periodically fired to adjust the target window of a cTCP flow as the codes shown in Figure 3. We expand a regular TCP control block with several new variables. We denote *adj_interval* as the duration (e.g., 8 seconds) of an adjustment interval to fire the cTCP timer. We use a vector, *target_rate*, to hold the target rates of video segments delivered in a cTCP flow. The *target_rate(i)* is the target rate of video segment i, and is used for computing a target window at the beginning of each adjustment interval during the delivery of the video segment. We denote *target_win* as the current target window size, which is updated at the beginning of every adjustment interval. We denote *curr_pkt_loss* as the number of packet losses in the current interval, which is initial to 0 at the beginning of each interval. The *target_rate* vector and *adj_interval* are set through *setsockopt()* system call (implemented in *tcp_usrreq.c*). The *curr_pkt_loss* is updated whenever a packet retransmission is detected. In *tcp_output.c*, we select the minimum of *wnd, cwnd* and *target_win* to be the effective flow control window size, instead the smaller of *wnd* and *cwnd*. Updating *target_win* is in *tcp_timer.c* using the TCP throughput model based on *target_rate*, *curr_pkt_loss* and the smoothed RTT, i.e., the RTT estimation in regular TCP. The rUDP implementation is as described in Section 4.2. We control the injection rate of an rUDP flow by setting its target rate and timer granularity. We expand the UDP protocol with an rUDP timer and a buffer, and keep rUDP state information in an extended IP Protocol Control Block.

In our experiments we connect three Dell PCs, *oak, breeze* and *ivy*, on a dedicated 100-Mbps Ethernet switch. We change the routing tables of *oak* and *ivy* to force them to send packets to *breeze*, which runs *dummynet* [20] as a bandwidth-delay control unit. Machine *breeze* forwards packets from *oak* to *ivy*, and



a. Two cTCP flows. b. Two TCP flows.

Figure 9. Experimental comparison of the transmission rates of two cTCP flows or two TCP flows over a fixed-capacity link. (Each point is the mean rate of 10 packets.)

vice versa. We set up two dummynet pipes: pipe 1 is for the forward path from *oak* to *ivy* via *breeze*, 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 dummynet pipes are set to have a delay of 40 ms and a bandwidth capacity of 64KBps. We use *tcpdump* to capture the packet headers at *ivy*. Figure 9.a shows the rates 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 fairly stable all the time. This result is consistent with our simulation result in Figure 4.b. In contrast, Figure 9.b shows the rate 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 fairly 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, which regulates the traffic between *oak* and *ivy* with a given propagation delay, a preset link rate, and a fixed 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 each 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 proxy *ivy*, and the remaining part is delivered from the server *oak*. We observe the data transmission between the central server *oak* and the proxy server *ivy*.

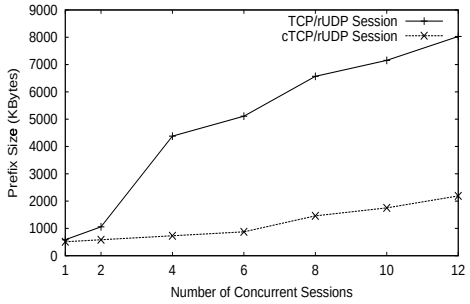


Figure 10. Comparison of the requirements on prefix size in cTCP/rUDP sessions and TCP/rUDP sessions.

We repeat the experiment many times to identify the required 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 is increased in Figure 10. The y-axis is the required prefix size (in KBytes). The figure shows 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 the two types of video sessions in another set of experiments. As the number of concurrent video sessions increases, Figure 11 shows the average bytes retransmitted by a cTCP or TCP flow in a session, and Figure 12 shows the average number of packet drops experienced by an rUDP flow in a session. Clearly, cTCP/rUDP sessions perform much better than TCP/rUDP sessions in both cases. However, different from the simulation results, we do see a small number of rUDP packet drops and a few cTCP retransmissions in our experiments. These drops and retransmissions are caused by the bursts generated due to the OS time resolution. We have an rUDP timer for each rUDP flow. When multiple rUDP flows exists, a timeouted rUDP flow may not get a chance to send data before its *next* timeout, due the OS scheduling limitation (10ms in our setting). We build a flow-aggregation mechanism to solve this issue. An 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 reduces 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 the two type 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 in interval 10, and terminate two sessions after three video segments. Figure 13 shows the number of packet retransmitted or dropped in a session at each segment. A cTCP/rUDP session has almost no or very small fluctuations at session arrivals or departures, while a TCP/rUDP session has a large number of packet transmissions and drops. This result is consistent with our simulation as shown in Figure 8.

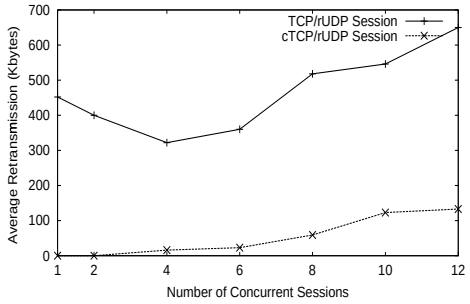


Figure 11. Comparison of the average number of retransmissions in cTCP flows of cTCP/rUDP sessions or in TCP flows of TCP/rUDP sessions.

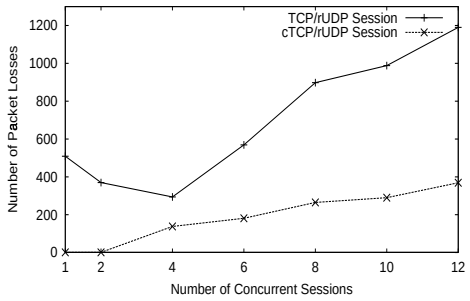


Figure 12. Comparison of the average number of rUDP packet losses in rUDP flows of cTCP/rUDP sessions or TCP/rUDP sessions.

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 an rUDP flow of a video session. We also did not address the synchronization issue 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. One lesson we learned is that a customized scheduler in the OS kernel must be built in order to support a large number of flows and enhance the scalability of the system.

One interesting observation in our experiments is that the delayed-ACK mechanism in the TCP may heavily effect the raw RTT in some cases. The delayed-ACK timer in FreeBSD4.1 is set to 100 ms. When a TCP flow is in slow-start or has a packet loss, a raw RTT may be as large as 180 ms since no data packets are sent on the return path. This large variation of raw RTTs could cause the inaccuracy of algorithms that depend on RTT estimations. We disable the delayed-ACK in our experiments to avoid this problem.

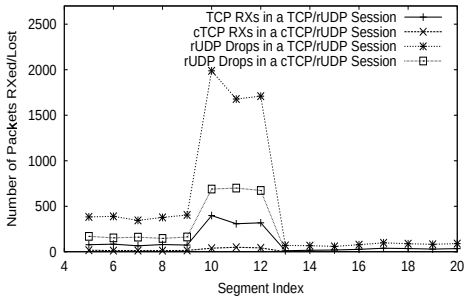


Figure 13. Comparison of the average numbers of cTCP- (or TCP-) retransmissions and rUDP packet losses in a cTCP/rUDP session or a TCP/rUDP session on session arrival/departures.

6. Control Plane Issues and Our On-going Study

In this study, we mostly address the bandwidth competition issue in the data plane. A key assumption of our proposed scheme is that a VPN pipe has sufficient bandwidth for cTCP flows to share. To ensure this condition, we propose several management schemes on the control plane.

First, we propose an *application-aware admission control* scheme to manage the load of a system [5]. Because the performance predictability of cTCP and rUDP provides us the chance of performing a form of application-aware rate control at either a proxy server or a central server, we are able to ensure that the total bandwidth requirement of all video sessions carried by a VPN pipe is less than the bottleneck capacity of the pipe. Suppose that the aggregate bandwidth of the VPN pipe is guaranteed and known via a service level agreement. The proxy server only allows a new video session to be carried over the VPN if its maximum bandwidth requirement (of both the reliable and unreliable flows) is less than the *residual* bandwidth on the VPN. Here the residual bandwidth is the difference between the capacity of the VPN pipe and the total bandwidth requirement of the ongoing video sessions. In the case where the aggregate bandwidth of the VPN pipe is not guaranteed, we resort to measurement-based techniques. We use the measured rates of both cTCP flows and rUDP flows as well as the measured packet losses of these flows as the criteria to determine whether a new video session can be admitted. If their measured rates of the flows are close to their corresponding target bandwidth requirements, *and* the measured packet losses are significantly below preset packet loss thresholds for both cTCP flows and rUDP flows (e.g., 0.05), the new video session is then admitted. Otherwise it is rejected. Our initial study of this issue is presented in [5].

Furthermore, rate adaptation schemes for admitted sessions are also important when temporary congestion occurs, in the case when a VPN does not provide a hard guarantee. Because a dedicated lease-line type of VPN is expensive and inefficient, most current VPN services are built at a network-layer and share the physical devices with other network services. When such a VPN is built across multiple network domains

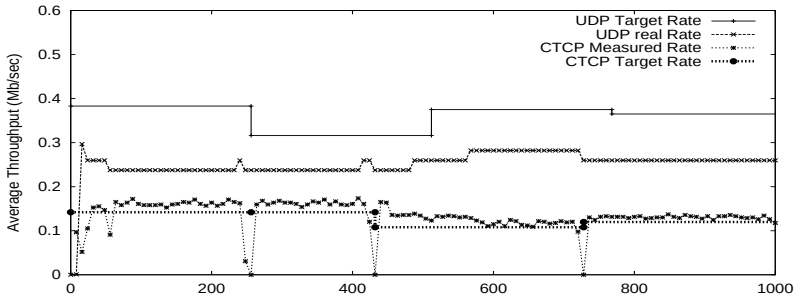


Figure 14. Illustration of the effect of rate adaptation: an rUDP flow backs off while its corresponding cTCP flow sustains close to its target rate.

belong to different administrative entities, it is extremely difficult to provide fine-grain packet delay or drop guarantee. Short term fluctuations in a VPN pipe across multiple domains can not be avoided in this case. When we stream videos through such a pipe, to avoid drastic changes in the user perceived video quality, the rates at which video flows are delivered across the pipe must also be adjusted dynamically in a proper manner. For example, taking into account application-specific information such as frame types is a practical choice. Because of the implementation issues (e.g., the drifts in timers or packet injection rates), this ability of rate adaptation may also be needed even when the aggregate bandwidth of the VPN pipe is guaranteed. We design a simple rate adaptation scheme which takes the network condition as well as the application information into consideration to adjust the unreliable flow to cede network resources to the reliable flow [24], when congestion occurs in the VPN pipe.

Figure 14 illustrates the effect of our rate adaptation scheme in a simple simulation. A cTCP/rUDP video session of *Star Wars* is carried over a VPN pipe with a bottleneck capacity C of 0.4 Mb/s. The average bandwidth requirement of the video session (including both flows) is 0.495 Mb/s, larger than C. The average rate of the cTCP flow is 0.134 Mb/s, smaller than C. The figure shows the measured rates of both the cTCP flow and the rUDP flow during the first 1000 seconds of the session, as well as their target rates. Because the rUDP flow reduces its transmission rate (lower than its target rate), the cTCP flow is able to attain its target rate. Note that the dips in measured cTCP throughput are due to the end of **I** segment transmission. Other simulations using different video traces yield the similar results.

How to improve the utilization of the VPN pipe when the pipe is light-loaded is another interesting issue. Since the VPN is provisioned for video streaming only, when the VPN is lightly loaded, it is desirable to exploit the *residual* of the available bandwidth in the pipe to further improve the quality assurance of admitted video sessions. We can utilize the buffer capacity at a proxy or a client, as well as the *residual* bandwidth to prefetch the enhanced-data flow. We can also apply error recovery schemes (e.g., FEC or selective retransmission) for unreliable flows. In addition, in order to maximize the system profit, how to

determine the long-term requirement of a VPN pipe based on video popularities is another issue to be solved.

7. Conclusions

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. Our technique is designed for stored videos that use compression schemes with inter-frame dependency. To implement such a technique, we have designed cTCP and rUDP with application-aware rate-control for the reliable and timely delivery of the essential data and the enhanced data of videos. Through simulations on NS-2 and experiments on FreeBSD 4.1, 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 mostly discuss the data transmission related issues in this paper. A major assumption in our data plane design is that a VPN pipe have sufficient bandwidth resources for cTCP flows. To ensure this condition, in the control plane we are designing a set of application-aware admission control, scheduling and adaptation schemes. As the first step, we are designing and testing a control-plane mechanism in a broadband cable access network environment, where proxy servers are naturally in place and VPN pipes are well managed. Furthermore, we will study the control plane issues on a large-scale content distribution networks which is overlayed on wide-area networks.

Acknowledgements

The authors like to thank Rohit Rakshe for his help in building the experimental system on FreeBSD.

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, Dept. of Computer Science and Engineering, U. of Minnesota.

5. Y. Dong, "Building Service-Oriented Networks with Quality Assurance across the Best-Effort Internet," PhD Thesis Proposal, Dept. of Computer Science and Engineering, U. of Minnesota.
6. N. G. Duffield, et al., "A Fexible 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.
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. Vettereli, "Receiver-driven Layered Mulitcast," 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. L. Rizzo, IP Dummynet, http://www.iet.unipi.it/~luigi/ip_dummynet/.
21. O. Rose, "Statistical Properties of MPEG Video Traffic and Their Impact on Traffic Modeling in ATM Network," TR-101, Institute of Computer Science, University of Wurzburg, Germany, Feb. 1995.
22. S. Sen, J. Rexford, and D. Towsley, "Proxy Prefix Caching for Multimedia Streams," in Proc. of IEEE INFOCOM'99.
23. 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.
24. Z.-L. Zhang, Y. Wang, D. Du, and D. Su, "Video Staging: A Proxy-Server-Based Approach to End-to-End Video Delivery over Wide-Area Networks," IEEE/ACM Transactions on Networking , Vol. 8, No. 4, August 2000, pp.429-442.
25. Z.-L. Zhang, S. Nelakuditi, R. Aggarwal, and R. Tsang, "Efficient Server Selective Frame Discard Algorithms for Stored Video Delivery over Resource Constrained Networks," in Journal of Real-Time Imaging, 2000.