# Measurement and Performance Study of PERT for On-demand Video Streaming

Bin Qian
Dept. of ECE
Texas A&M University
Email: anson627@tamu.edu

A.L.Narasimha Reddy
Dept. of ECE
Texas A&M University
Email: reddy@ece.tamu.edu

***Abstract*—This paper demonstrates the performance of PERT, a new TCP congestion control, for video streaming. PERT stands for Probabilistic Early Response TCP. As a delay based protocol, it measures the delay at the end host and adjusts the congestion window accordingly. As our NS-2 simulation results show, the late packet of the video streams are greatly decreased when applying PERT in heterogeneous environment. Furthermore, our Linux live streaming test indicates that PERT is able to reduce the playback glitches, when high resolution video is delivered over an end-to-end link with latency and packet loss.**

## I. Introduction

Data transfers over Internet are increasingly dominated by video data transfers. Cisco forecasts that video transfers will account for 60-90% of network traffic by 2014 [15]. Recent studies indicate that a large portion of Internet streaming media in current Internet is delivered over HTTP/TCP. To achieve satisfactory quality of service (QOS), video and audio data are supposed to be delivered before playback or buffered if they arrive earlier. However, current TCP is not suitable for video streaming applications due to its insistence on reliable transmission and inability of real-time data delivery. Moreover, in today's Internet, many other services like web surfing, FTP download, as well as P2P file sharing are also competing for the limited bandwidth. This makes it more difficult for TCP to meet the demands of smooth video streaming.

Motivated by such demand on multimedia applications over the Internet, protocols for video streaming have been explored by many researchers. TFRC (TCP Friendly Rate Control) [1] and its variant [2] have been proposed to maintain long-term TCP fairness while maintaining smooth transmission rates. In [3], Wang et al. analytically studied the TCP performance for multimedia streaming. They built discrete-time Markov models for both constrained and unconstrained streaming. Smaller than MSS-sized packets have been used in CBR workloads to exploit the TCP ACK counting mechanism, and thereby reducing the TCP transport delay and its impact on congestion window variations in [4]. In [5], the authors compared Linux implementations of NEWRENO, H-TCP and CUBIC and found dynamic latency fluctuations induced by each TCP variant. They noticed that CUBIC induces larger latency than the other two when concurrent TCP flows take place. All of these studies explore the possibility of employing TCP like congestion control even for real-time video delivery.

In this paper, we explore the performance of a new TCP congestion control - PERT and compare it with other TCP variants like RENO and CUBIC for real-time video transmission. Here RENO is short for RENO-SACK. We study if the delay-based PERT mechanism can provide better support for video delivery than RENO and CUBIC. We study this problem through NS-2 based simulations and real live video transmission tests on a testbed. Both our NS-2 simulation and Linux test results show that PERT provides significant improvement on video viewing quality when compared to RENO and CUBIC.

The rest of the paper is organized as follows. In section II, we describe our previous work on PERT to make this paper self-contained. In Section III, we present our extensive NS-2 simulations and performance comparison among PERT, RENO and CUBIC. In Section IV, we explore the feasibility of applying PERT to video streaming in the real world. Finally, we conclude the paper with a discussion of possible extension of PERT in section V.

## II. Background

PERT emulates the behavior of AQM/ECN at the end host [6]. As a delay based protocol, PERT learns about network congestion by measuring delays at end host, and probabilistically reduce the congestion window as delay increases. Fig. 1 shows PERT's response probability curve, where $T_{min}$ and $T_{max}$ are two thresholds, and $P_{max}$ is the probability of response at $T_{max}$.

However, delay based protocols lose to loss-based protocols in heterogeneous environments where multiple congestion control algorithms may be employed. In current Internet, any new congestion control algorithm has to be able to coexist with prevailing TCP congestion control algorithms. To address this problem, PERT was redesigned to be adaptive to heterogeneous environments [7]. PERT increases congestion window faster than TCP at low delays to compensate for early response at higher delays, in order to equalize the bandwidth.

PERT basically operates in 3 modes. When the observed delay is very low (or below the minimum threshold), it assumes that it is operating in a "high-speed" mode and increases the window fast to fill the link. In this mode, the window increase factor $\alpha$ in W = W + $\alpha$, is increased linearly until a maximum value of $\alpha_{max}$ (currently set to 32). When the observed delay is above a TCP-compete threshold (currently

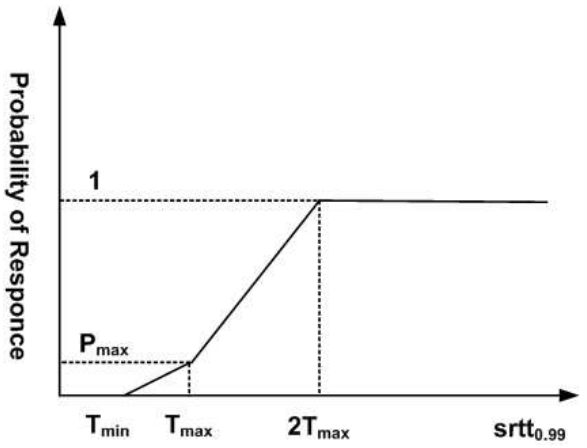



Fig. 1. Response probability vs. Smoothed RTT

set to 0.65*maximum observed queuing delay), PERT assumes it is operating in a heterogeneous environment and increases the window every RTT additively with $\alpha = 1+p'/p$, where $p'$ is the early response probability and p is the observed packet loss rate. When the observed delay is above the minimum threshold, but below the TCP-compete threshold, PERT assumes it is operating in a "safe" mode and increments window additively with $\alpha$ = 1. In addition, PERT reduces the window conservatively in the early response phase, $W = W \times (1-\beta)$, where $\beta = q'/(q'+q)$, where $q'$ is the estimated queuing delay at early response phase and $q$ is the observed maximum queueing delay. It is observed that this leads to $W = W/2$ upon a packet loss.

Simultions and real-network evalutions have shown that, (a) a single PERT flow can scale to high-speed links of up to 10Gbps, (b) PERT can compete with TCP in heterogeneous environments and (c) still benefit from near-zero packet loss rates and very low queuing delays when operating in homogeneous environments. Details of PERT design can be found in [7], [8].

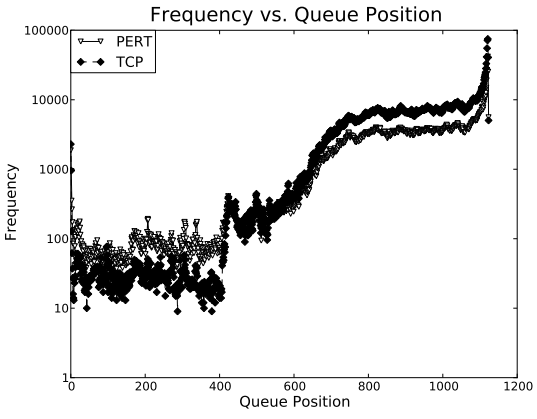



Fig. 2. Frequency vs. Queue Position

PERT sends more packets at lower delays and sends fewer packets at higher delays while being fair to TCP. This behavior is shown in Fig. 2. Fig. 2 shows the queuing frequency at certain queue length when 50 PERT and 50 RENO FTP flows are competing on a 150 Mbps link with 60ms delay. It is observed that PERT enqueues more packets earlier in the queue, and less packets later, and as a result experiences lower losses. We expect this behavior of PERT to be beneficial for video transfers and this paper investigates this issue through simulations and experiments over a testbed.

Recent work [3] has showed that TCP can be used for transmitting live video as long as the required video stream rate is a fraction of the average bandwidth achievable by a single TCP flow. Both constrained (data is available from a live stream) and unconstrained (data is available from a prerecorded or stored source) streaming were considered. Extensive simulations have shown that TCP can be adequate if the average TCP flow rate is about twice that of the required video stream rate for constrained streaming. A similar study by [14] has shown that TCP can function adequately with a 1.5 higher bandwidth than required stream rate in unconstrained streaming and that Vegas could support unconstrained streaming better than TCP NEWRENO. The question we try to answer in this paper is if a delay-based protocol such as PERT can support constrained video streaming at a lower available bandwidth than two times of the required video stream rate as required by RENO.

We carry out extensive NS-2 based simulations and live video transmissions on a real network testbed within the lab. We present data from both simulations and the emulations to show that PERT indeed provides better support for live video transmission than RENO and CUBIC.

## III. NS-2 Simulation

### A. Experiment Setup

To evaluate the performance of PERT and other TCP congestion control variants, we setup a dumbbell topology. In such a network environment, multiple TCP streams have sufficient bandwidth over access links separately but compete for the limited bandwidth over the bottleneck link between the two routers.

Table. I shows our experiment parameters in NS-2 simulation, we set the access links bandwidth to 10 Mbps and bottleneck link bandwidth to 25 Mbps, and CBR bit rate to 300 Kbps. Moreover, TCP packet size is set to 200 or 1,000 bytes, router buffer size to 150 packets, and video length to 7,000 seconds. We keep the above parameters constant to get rid of their impacts on TCP video streaming performance. We take HTTP and FTP flows as the background traffic, RENO and CUBIC as the control group. As a loss based protocol, RENO additively increases the congestion window by one MSS (Maximum Segment Size) every RTT (Round Trip Time), cuts down the congestion window by half on a packet loss and decreases it to one MSS on a timeout event. As for CUBIC, the congestion window growth follows a cubic function in terms of the elapsed time since the last loss event. To emulate a realistic network, we vary link delay

TABLE I
NS-2 SIMULATION EXPERIMENT SETUP

| Parameter | Value |
|---|---|
| CBR Flows # | 20 - 35 |
| CBR Senders | PERT/RENO/CUBIC |
| CBR Recvers | RENO |
| CBR Rate | 300 Kbps |
| FTP Flows # | 20 - 35 |
| FTP Senders | RENO |
| FTP Recvers | RENO |
| HTTP Flows # | 300 |
| HTTP Senders | RENO |
| HTTP Recvers | RENO |
| Video Length | 7,000 secs |
| Packet Size | 200/1,000 Bytes |
| Buffer Size | 150 Packets |
| Access Link Bandwidth | 10 Mbps |
| Access Link Delay | 5 - 15 ms |
| Bottleneck Link Bandwidth | 25 Mbps |
| Bottleneck Link Delay | 15 - 45 ms |
| Round Trip Time | 50 - 150 ms |
| Random Seed | 0 - 19 |

(RTT) by altering the access link delay and bottleneck link delay. And the RTT equals four times access link delay plus two times bottleneck link delay. We also vary the number of CBR streams and the number of FTP streams from 20-35 and keep the number of HTTP streams constant at 300 to achieve different TCP throughputs. Finally, we run the simulation 20 times with seed values of 0-19 to randomize the start time of the TCP streams, in order to statistically reduce its effect on the experiment results.

### B. Simulation Results

*1) Parameters Exploration:* In this section, we explored the experiment parameters to study the performance of PERT under different conditions. As [3] concludes, the performance of TCP generally provides good streaming performance when the T/$\mu$ is roughly 2.0, where T is the achievable TCP throughput and $\mu$ is the video bit-rate. To demonstrate PERT's performance under different T/$\mu$s, we pick sample data with certain CBR stream numbers such that T/$\mu$ falls in continual ranges of [1.0 - 1.2], [1.2 - 1.4], [1.4 - 1.6], [1.6 - 1.8], [1.8 - 2.0], as Fig. 3 shows. Under such T/$\mu$ distribution, we also plot the bandwidth allocation among CBR, FTP and HTTP streams. As Fig. 4 shows, as the number of CBR streams increases, the total bandwidth of CBR streams increases proportionally, and the rest of bandwidth is taken by FTP streams and HTTP streams.

As Fig. 5 shows, the fraction of late packets for PERT CBR streams becomes smaller as T/$\mu$ increases from 1.0 to 2.0. It is clear that the late packets can be reduced by giving CBR traffic more bandwidth. Fig. 5 also indicates that the fraction of late packets with PERT drops sharply when T/$\mu$ is increased from 1.0 to 1.4, and stays almost the same as T/$\mu$ ranges from 1.4 to 2.0. Moreover, as the RTT increases from 50ms to 150ms, the fraction of late packets goes up. This agrees with our intuition that it is more difficult to achieve satisfactory performance for video streaming in higher delay networks (retransmissions htbp

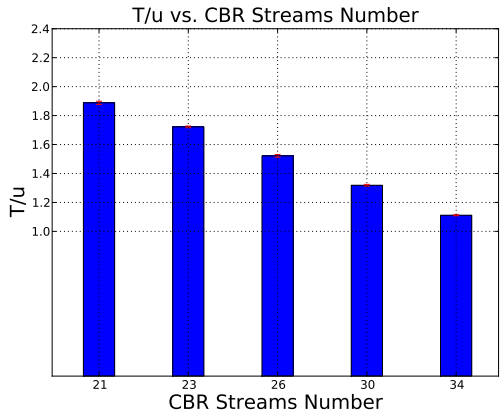



Fig. 3. T/$\mu$ Distribution vs. CBR number

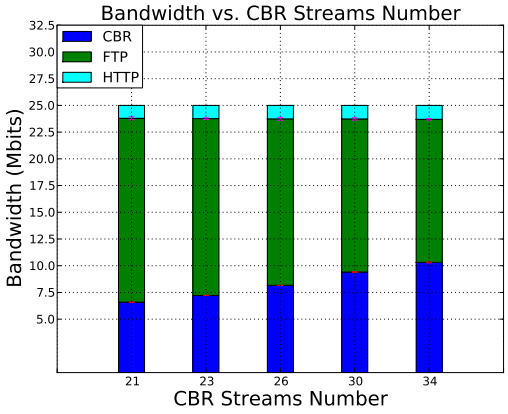



Fig. 4. CBR, FTP and HTTP bandwidth vs. CBR number

may not arrive in time, for example).

Studies show that the video viewing quality is closely related to the fraction of late packets. We define that a CBR stream is successful as long as the fraction of late packets is below $10^{-4}$, with only a few seconds of startup delay as the paper [3] did. Under such an evaluation metric, we validate PERT's performance and compare with others in the aspect of delivered video quality.

*2) Performance Comparison:* In this part, we compare the performance of PERT, RENO and CUBIC for video streaming in the same parameter space that we described in the last section. As our simulation results indicate, PERT outperforms RENO over all T/$\mu$, loss rates and start-up delays, and CUBIC over low T/$\mu$, high loss rates and strict start-up delay constraints.

First, we demonstrate the performance in terms of fraction of successful CBR streams that we defined before. Fig. 6 indicates that as T/$\mu$ increases, the fraction of successful CBR streams goes up. In the high T/$\mu$ range [1.4 - 2.0], the percentage of successful CBR streams is high and changes

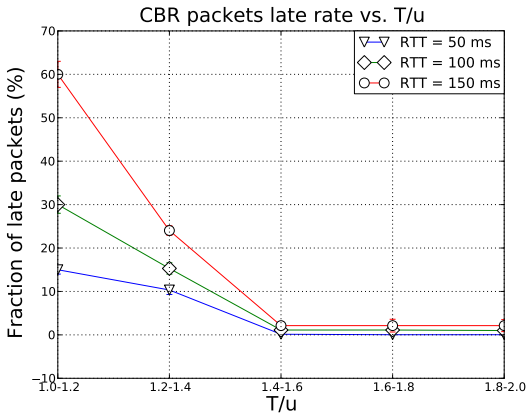



Fig. 5. Fraction of late packets vs. T/$\mu$ with PERT

slightly as T/$\mu$ increases. While in the low T/$\mu$ range [1.0 - 1.4], the performance drops drastically as T/$\mu$ decreases. In comparison, PERT and CUBIC perform better than RENO. When packet size is moderate - 1000 bytes, PERT performs better than CUBIC in the low T/$\mu$ range but has similar performance as CUBIC in the high T/$\mu$ range. Fig. 6 also shows the impact of packet size, we consider two packet sizes of 200 and 1000 bytes. It is observed that the fraction of successful streams is higher with smaller packet sizes, in all T/$\mu$ ranges.

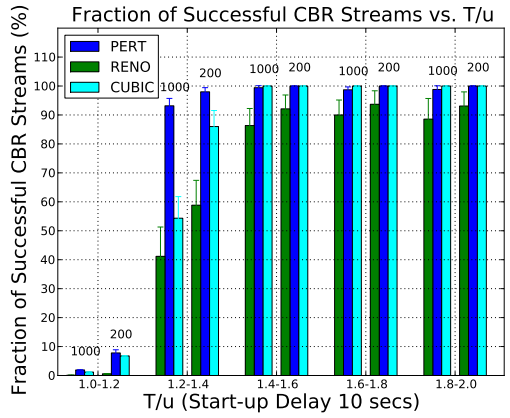



Fig. 6. Fraction of successful CBR streams vs. T/$\mu$ over different packet sizes

Fig. 7, 8 show that the percentage of successful CBR streams vs. start-up delays in different T/$\mu$ ranges when three different TCP congestion controls are employed in video transmission. As T/$\mu$ increases, the CBR streams can achieve higher throughput and lower packet loss rate. And as the start-up delay goes up and the constraint becomes loose, more CBR streams successfully are able to meet the streaming quality requirement. These observations are consistent with the earlier study [3].

In comparison, as Fig. 7 show, when the T/$\mu$ is in the low range [1.0 - 1.4], the loss rate is relatively high, PERT achieves the best performance, CUBIC is in the middle, and RENO is the worst.

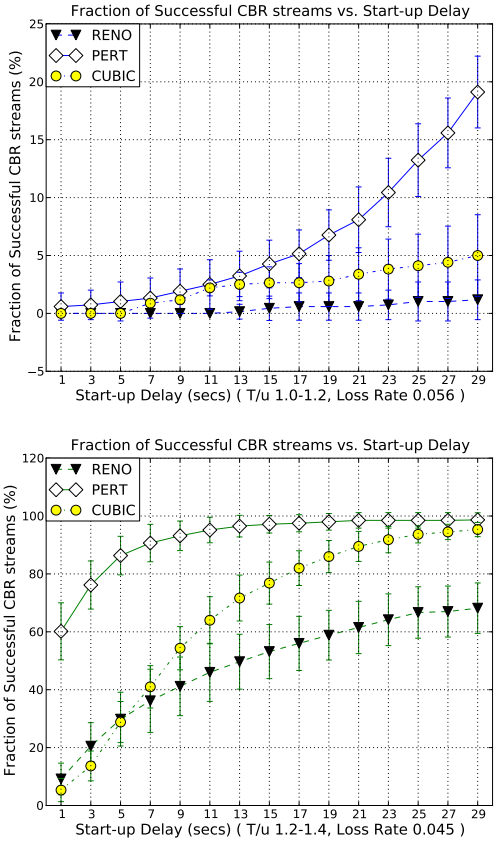


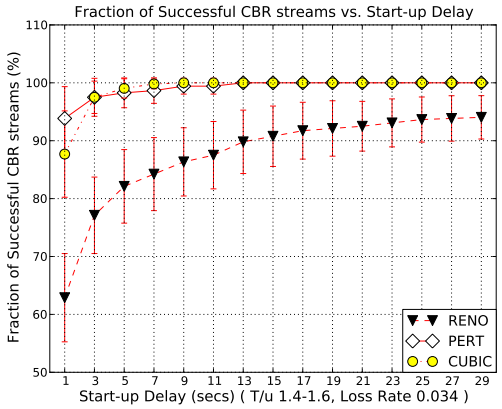

Fig. 7. Fraction of successful CBR streams over T/$\mu$ 1.0-1.4

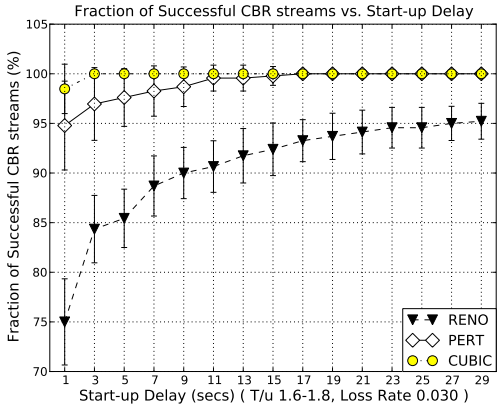
As Fig. 8 show, in the high T/$\mu$ range [1.4 - 2.0], PERT and CUBIC are both superior to RENO, and especially when the start-up delay constraint is tight (from 1 to 11 seconds). In addition, when the start-up delay is greater than 11 seconds, PERT and CUBIC achieve almost 100% success rate. In other words, the T/$\mu$ constraint for satisfactory streaming is improved from roughly 2.0 to approximately 1.4, by using PERT or CUBIC.

Fig. 9 show that PERT generally performs better than RENO and CUBIC, in the loss rate range of [0.04 - 0.06]. This is because PERT reduces the congestion window by small amounts ahead of packet loss and experiences fewer packet losses [7], which brings more stable throughput and therefore smoother video streaming.

## IV. Linux Video Streaming Test

In order to test our Linux implementation of PERT, we configured a testbed environment with the help of a network emulator. Fig. 10 displays our testbed setup. Two PCs with the same hardware and operating system, are connected to a PC configured as a network switch, through a 10/100 Mbs Ethernet link. The VLC[9] application is installed to stream



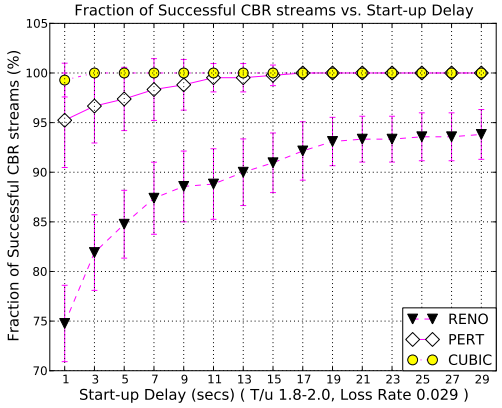
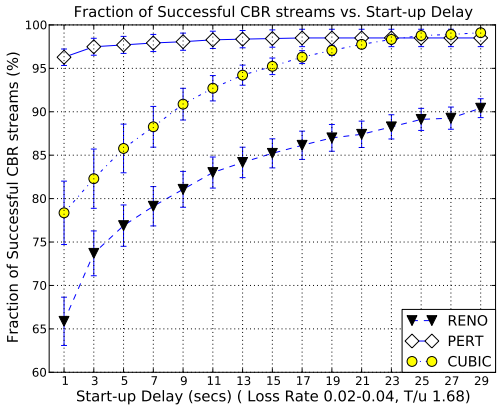




Fig. 8. Fraction of successful CBR streams over T/$\mu$ 1.4-1.6



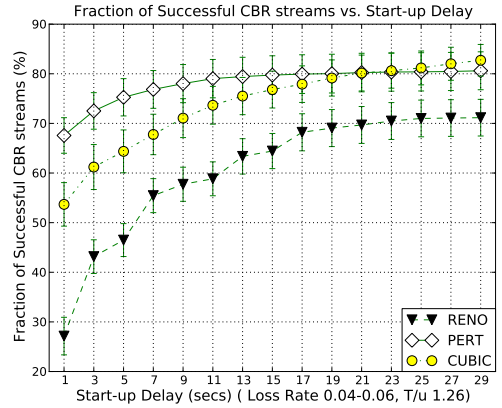




Fig. 9. Fraction of successful CBR streams over loss rate 0.02-0.06

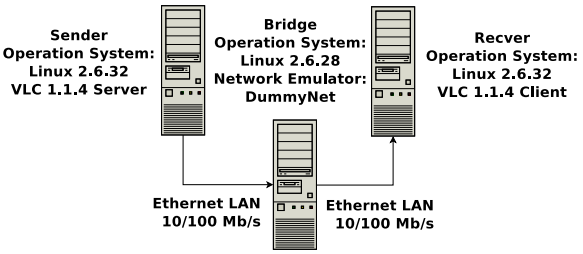



Fig. 10. Experiment computer platform end-to-end connected with network emulator

a video from the sender to the receiver. DummyNet [10] is employed as the network emulator to create a more realistic testing condition. It works on the switch to emulate buffering, queuing delays and bottleneck link bandwidth.

Based on the testbed described above, we performed the video streaming test with PERT, RENO and CUBIC. We set the link bandwidth to 15 Mbps, the link delay to 45 ms, and the queue length to 500 kbytes on the bridge. Moreover, a 1080p version of the Avatar movie trailer is chosen as the sample video, with file size of 286.5 Mb and playback duration of 3 minutes and 30 seconds. VLC 1.1.4 works on both the server and the client end as the video streaming tool. The codec of the video and audio are H-264 and MPEG 4 Audio (AAC) respectively. As a high resolution movie, the video is played at a frame rate of 24 fps and the audio at a sampling rate of 48,000 Hz. This asks for video peak bit rate of 25 mbps and audio bit rate of 192 kbps. Besides, HTTP is chosen as the application layer protocol, which takes advantage of TCP on the transport layer. We employ different versions of TCP during our experiments to measure their effectiveness at streaming.

### A. Test Results

To compare the effects of PERT, RENO and CUBIC on the perspective of video viewing quality, we analysed VLC's playback logs, which record when and how if any glitch happens during the video transmission. We play the same video with the same setting 20 times and count how many times the events of late picture skipping and audio output starving occur. When the video or audio frames are played but not found in the client's buffer, one of these events will occur. They lead to playback glitches, VLC client buffering, user waiting and therefore impair the viewing quality. As Table. II shows, the playback experienced smaller number of late picture skipping and audio output starving events when using PERT instead of RENO or CUBIC as the TCP congestion control.

TABLE II
VIDEO STREAMING PERFORMANCE COMPARISON

| TCP Congestion Control | PERT | RENO | CUBIC |
|---|---|---|---|
| Late Picture Skipping # (per playback) | 5.5 | 33.5 | 30.5 |
| Audio Output Starving # (per playback) | 3.0 | 11.0 | 7.5 |

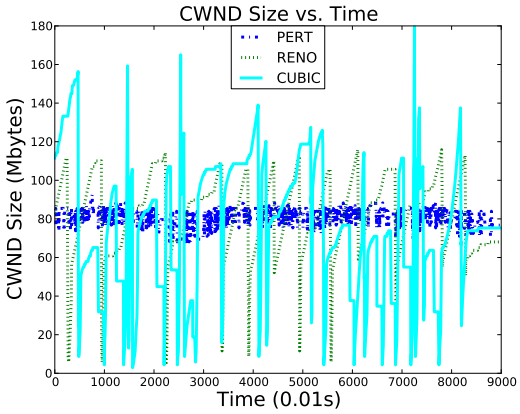



Fig. 11. Congestion window in Linux Test

To confirm our observation, we plot the TCP congestion window size with web100 [11] at the sending end during playback in Fig. 11. We choose a period of 90 seconds of playback time at the end of the movie, since the streaming is relatively stable during that time. And we track the congestion window size every 0.01 second. As Fig. 11 shows, the congestion window size of PERT has fewer fluctuations than that of RENO or CUBIC does. CUBIC increases the congestion window fast and achieves sightly better performance than RENO does, but still incurs large fluctuations. Therefore, the throughput of PERT's video streams are more steady and their data frames are more likely to arrive before playback deadline. This can explain the smaller number of late picture skipping and audio output starving events we observed, when PERT is employed as the TCP congestion control.

## V. CONCLUSION AND FUTURE WORK

In this paper, we have demonstrated the performance of PERT for on-demand video streaming. Our NS-2 simulation experiments are performed in a heterogeneous environment, where the background traffic is delivered by RENO. PERT outperforms RENO and CUBIC in successfully delivering video in constrained streaming scenarios we considered here. Moreover, the real-life video streaming test confirms PERT's ability to improve video playback quality when comparing to RENO and CUBIC. In the future, we will carry out more evaluations, including comparisons against other protocols. Further, our efforts will be devoted to deploy and measure PERT in error-prone wireless networks.

## ACKNOWLEDGMENT

This work is supported in part by NSF grants 0702012 and 0621410, a Qatar National Research Foundation grant and by Qatar Telecom. We would like to thank Zhiyuan Yin for his help with testbed building and video streaming test.

## REFERENCES

[1] S. Floyd, M. Handley, J. Pahdye, and J. Widmer. RFC 5348: TCP Friendly Rate Control (TFRC): Protocol specification. RFC 5348, Proposed Standard, September 2008.
[2] Jie Feng, Lisong Xu. Throughput-Smoothness Tradeoff in Preventing Competing TCP from Starvation. Proceedings of the 17th International Workshop on Quality of Service (IWQoS), Charleston, South Carolina, July 2009.
[3] Bing Wang, Jim Kurose, Prashant Shenoy and Don Towsley. Multimedia Streaming via TCP: An Analytic Performance Study. ACM Transactions on Multimedia Computing Communications and Applications (TOMCCAP), Vol. 4, No. 2, April 2008.
[4] Salman A. Baset, Eli Brosh, Vishal Misra, Dan Rubenstein, and Henning Schulzrinne. Understanding the behavior of TCP for real-time CBR workloads. Proceeding of International Conference On Emerging Networking Experiments And Technologies archive, Dec. 2006.
[5] Lawrence Stewart, Grenville Armitage and Alana Huebner. Collateral Damage: The Impact of Optimised TCP Variants on Real-Time Traffic Latency in Consumer Broadband Environments. IFIP International Federation for information Processing 2009.
[6] Sumitha Bhandarkar, A. L. Narasimha Reddy, Yueping Zhang, and Dmitri Loguinov. Emulating AQM from End Hosts. SIGCOMM Computer Communication Review, vol. 37, no. 4, pp. 349360, October 2007.
[7] Kiran Kotla and A. L. Narasimha Reddy Making a Delay-based Protocol Adaptive to Heterogeneous Environments. Proceedings of the 16th International Workshop on Quality of Service (IWQoS 2008), June 2008.
[8] K. Kotla Adapting a delay-based protocol to heterogeneous environments. Master's thesis, Texas A&M University, http://cegroup.ece.tamu.edu/techpubs/2008/TAMU-ECE-2008-02.pdf, Aug. 2008.
[9] The Videolan project, http://www.videolan.org.
[10] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. ACM Computer Communication Review (January) (1997). http://info.iet.unipi.it/luigi/ip_dummynet.
[11] Ajay Tirumala, Les Cottrell, and Tom Dunigan. Measuring end-to-end bandwidth with Iperf using Web100. PAM (2003).
[12] Aiman Erbad, Mahdi Tayarani Najaran and Charles Krasic. Paceline: latency management through adaptive output. Proceedings of the first annual ACM SIGMM conference on Multimedia systems, 2010.
[13] B. Wang, W. Wei, Z. Guo and D. Towsley. Multipath live streaming via TCP: Scheme, Performance, Benefits. Proceeding of International Conference On Emerging Networking Experiments And Technologies archive, Dec. 2007.
[14] S. Boyden, A. Mahanti and C. Williamson. TCP Vegas performance with streaming media. Proc. of IPCCC, 2007.
[15] Cisco Systems. Cisco Visual Networking Index: Forecast and Methodology, 2009-2014. www.cisco.com, June 2010.