Improving the Performance of Delay Based Protocol in Delivering Real Time Media via Early Retransmission

Zhiyuan Yin, Hussein Alnuweiri, A.L. Narasimha Reddy, Hasari Celebi and Khalid Qaraqe
Department of Electrical and Computer Engineering, Texas A& M University
Email: {zyin3@neo, hussein.alnuweiri@qatar, reddy@ece, hasari.celebi@qatar, khalid.qaraqe@qatar}.tamu.edu

***Abstract*—Early Retransmission (ER) has already been shown to improve TCP-SACK's performance in delivering time sensitive media. In this paper, we integrate the ER scheme into a delay-based TCP variant, namely TCP-PERT, and performs extensive *ns2* simulation. Our results show that ER can also improve PERT's performance in delivering real time media by reducing the latency caused by retransmission timeouts. Furthermore, we also explored the improvement brought by employing a fine-grained retransmission timer, and compared it with ER. We find out a) ER outperforms fine grained timer in a variety of conditions and b) the combination of the two can further improve performance.**

I. Introduction

Currently, a large portion of Internet video streaming is delivered via HTTP over TCP. Despite being the dominant transport protocol, TCP has been regarded as not suitable for delivering time sensitive media for a long time. One of the major reason for that is the latency caused by TCP's reliable transmission and in order delivery.

When a packet drop occurs, all the successfully delivered packet will be stuck at receiver's buffer until enough duplicate acknowledgements (**DUPACK**) are generated to trigger the sender to retransmit the lost packet, which is usually several Round Trip Time (**RTT**) later. In the worst case, if not enough DUPACKs is able to be generated to trigger loss recovery, TCP sender will have to rely on Retransmission Timeout (**RTO**) to retransmit the dropped packet, which will result in a even larger latency.

Researchers have proposed several approaches to address those problems.

TCP friendly congestion control protocols without packet retransmission such as [1] and [2] have been proposed to eliminate the latency induced by reliable transmission. However, the effect of packet loss in video streaming can be very severe. For instance, loss of packet that is a part of an I-frame in MPEG movie can cause a large group of surrounding frames to be un-viewable. Therefore, those scheme usually rely on FEC to overcome this problem.

Attention has also been focused on developing TCP variants that perform Active Queue Management (**AQM**) at end host to avoid packet drop and evaluating their performance in delivering real time media. [3] shows that TCP-Vegas only require the achievable bandwidth to be 1.5 times the streaming rate to achieve successful streaming. [4] find out for PERT, the requirement is 1.6 times the streaming rate.

Finally, various researchers have put effort in reducing the latency induced by TCP's inherent nature. [5] proposed an approach to reducing the latency caused by TCP's sender side buffering by dynamically tuning TCP's send buffer. [6] and [7] both target at alleviating latency caused by packet loss and modify TCP's in order delivery scheme by allowing receiver to skip holes in receive buffer to make sure application thread never be blocked due to packet loss. [6] also involves application level technique to ensure multimedia applications perform well over their modified TCP. In [8], we proposed an early retransmission scheme and integrated it into a typical TCP implementation, namely TCP-SACK. ER is shown to improve SACK's performance by considerably reducing retransmission timeouts.

In this paper, we measure and study the performance improvement that ER brings to a delay based TCP variant, namely TCP-PERT. Our *ns2* based simulation shows that ER can serve as an enhancement to further improve PERT's ability in delivering streaming media. In addition to that, we also experimented with adopting a fine grained retransmission timer to decrease the latency. Our results also show that compared to the fine grained timer, ER's is more effective, and combining the two techniques together can lead to a even bigger improvement.

The rest of the paper is organized as follows: Section II presents relevant background information for this work. Section III shows the simulation result of integrating ER into PERT. Section IV compares ER with fine grained retransmission timer. Conclusions and future directions are presented in Section V.

II. Background

A. TCP Retransmission Timeout

Retransmission Timeout (**RTO**) is TCP's last resort to trigger its loss recovery algorithm. When not enough DUPACKs are generated, a TCP sender won't start recovering from loss until the timer fires. Instead of using a fixed timeout value, TCP uses RTT samples and the formula

$$\text{RTO}_i = \text{SRTT}_i + 4 \times \text{SRTTVAR}_i \quad (1)$$

978-1-4577-0024-8/11/$26.00

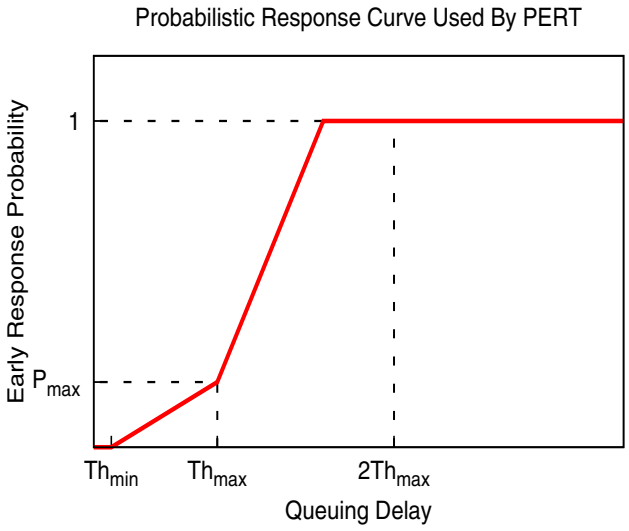


Fig. 1: PERT's Probabilistic Response Function

to calculate a dynamic RTO, where

$$\text{SRTT}_i = 7/8 \times \text{SRTT}_{i-1} + \text{RTT}_i \quad (2)$$

is the smoothed average RTT estimation and

$$\text{SRTTVAR}_i = 3/4 \times \text{SRTTVAR}_{i-1} + \text{RTTVAR}_i \quad (3)$$

is the smoothed RTT variance estimation.

In reality, the granularity of the RTO value calculated depends on the processor as well as OS implementation. What's more, most TCP implementation will clamp the minimum value of RTO to a static value. Linux, for example, has a timer granularity of 10ms and a minimum RTO value of 200ms [9]. So is the TCP implementation in *ns2*.

B. Early Retransmission

Despite the enhancement brought to its recovery algorithm by NewReno and SACK [10], TCP will invariably resort to retransmission timeout to trigger loss recovery if not enough DUPACKs are generated. [11] shows 85% of the timeouts in Internet traffic is caused by non-trigger of loss recovery algorithm.

Aimed at reducing the number of RTOs via eliminating non-triggers. ER used a similar technique in [12] and combined TCP's classic three duplicate acknowledge heuristic with a timeout/threshold mechanism, where a dynamic timer/threshold value is maintained. Each time when a DUPACK arrives, the round trip time of that DUPACK will be compared with the timer and TCP's recovery algorithm will be triggered if the round trip time is greater than the threshold, which is given by

$$\text{F-RTO} = \text{RTT'}_{max} + \text{RTT'}_{min} \quad (4)$$

, where RTT'_{max} is the estimation of the max round trip time, and is calculated through

$$\text{RTT'}_{max} = \text{SRTT}_i + 2 \times \text{SVAR}_i \quad (5)$$

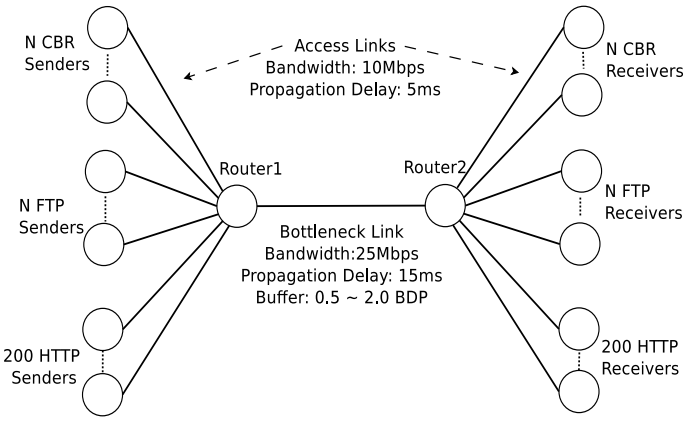


Fig. 2: Dumbbell Network Topology

where $SRTT_i$ is given by

$$\text{SRTT}_i = 15/16 \times \text{SRTT}_{i-1} + 1/16 \times \text{SRTT}_i \quad (6)$$

and SVAR_i is given by

$$\text{SVAR}_i = 3/4 \times \text{SVAR}_{i-1} + 1/4 \times \text{VAR}_i \quad (7)$$

We use the minimum round trip time experienced as the estimation for RTT_{min}.

C. PERT

Being a delayed-based TCP variant, Probabilistic Early Response TCP (**PERT**) is designed to emulate AQM at end host. A PERT sender constantly measures queuing delay and associates it with network congestion level. When it senses the network to be congested, the sender will slow down proactively to avoid packet drop. This is achieved by performing proactive slow down with a probability that is a function of the measured queuing delay. Different early response function can be chosen depending on the algorithm one wants to achieve. Currently, the early response function shown in Figure. 1 that emulates the behavior of RED [13] is used by PERT. [14] and [15] explored PERT's performance in a homogeneous and heterogeneous environment respectively. In [4], PERT is shown to have a much better performance in delivering real time media than TCP-SACK.

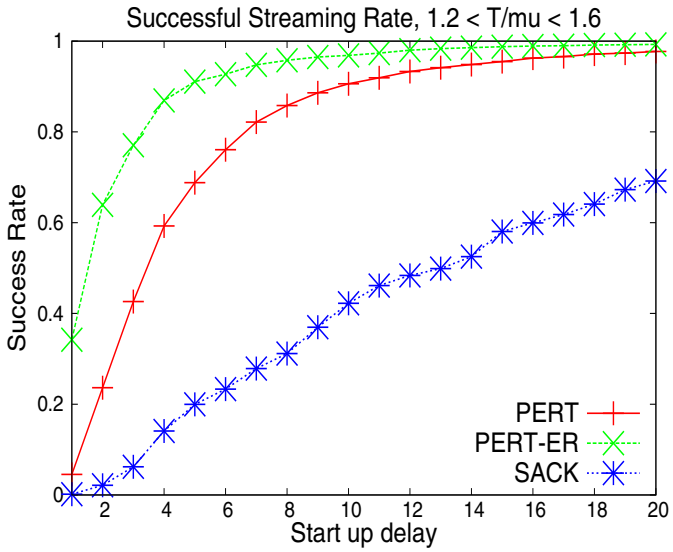
III. PERT vs. PERT-ER

In this section, we measure and study the performance improvement from integrating ER into PERT. To get a better feeling of how ER and AQM can bring improvements incrementally, the results of TCP-SACK are also collected.

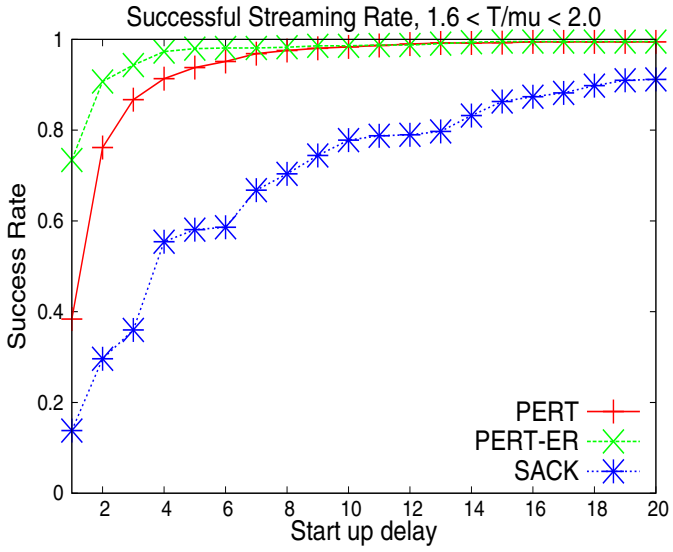
A. Simulation Methodology

Throughout this paper, a dumbbell network topology shown in Figure. 2 is used, where different TCP flows are given enough bandwidth at access link, but compete for limited bandwidth at the bottleneck link.

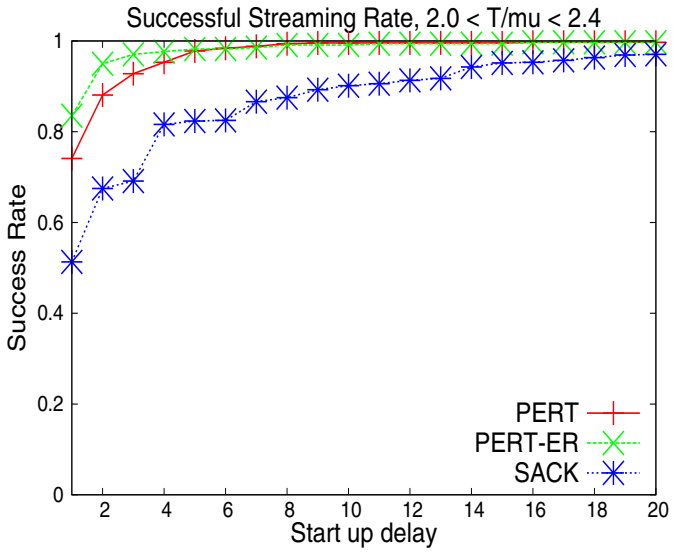
Values for various parameters of *ns2* simulation are shown in Table. I.



(a) $1.2 < T/\mu < 1.6$



(b) $1.6 < T/\mu < 2.0$

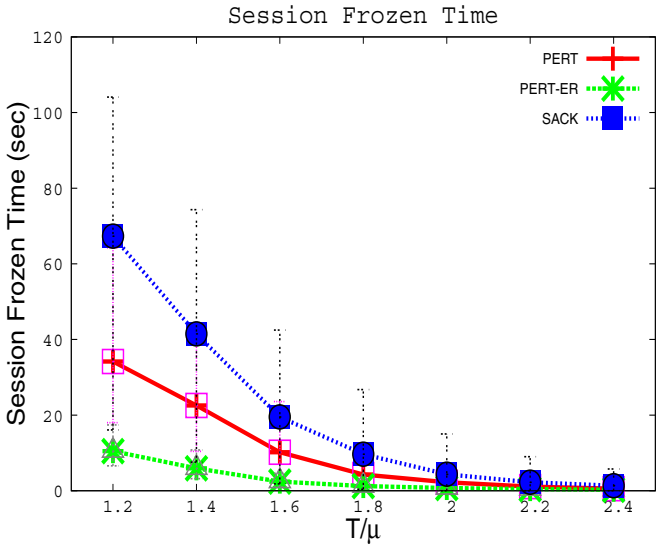


(c) $2.0 < T/\mu < 2.4$

Fig. 3: Successful Streaming Rate vs. Start Up Delay With Varying T/μ

Parameter	Value
Simulation Duration	1000s
Bottleneck Link Bandwidth	25Mbps
Access Link Bandwidth	10Mbps
Round-trip Prop. Delay	50ms
Router Buffer	0.5 BDP ∼ 2.0 BDP
CBR Flow #	16 ∼ 40
FTP Background Flow #	16 ∼ 40
HTTP Background Flows	200
Background Flow Protocol	TCP-SACK
MSS	1000Byte
Media Encoding Rate	300kbps
TCP Maximum Window Size	64KB

TABLE I: Simulation Setup



(a) $2.0 < T/\mu < 2.4$

Fig. 4: SFT Comparison of SACK, PERT and PERT-ER

To provide a good metrics for evaluating media delivery performance, we modify the **TcpSink** module in *ns2* to record the time when a packet is ready to be fed to layer-5 application and put all those timestamps into a log file. After the simulation run, different start up delay (**SUD**) is applied against those timestamps to collect the number of packets that miss play out deadline (**Late Packets**). As [16] shows that a late packets percentage of as little as 3% can affect up to 30% of the frames in MPEG-1 video, we define a video stream to be successful if less then 0.01% of its total packets sent miss their play out deadline. [17] and [3] also use identical methods.

We use a pair of Constant Bit Rate (**CBR**) traffic generator and receiver to simulate a video streaming flow. In addition to that, FTP and HTTP flows are introduce to create cross traffic and cause congestion at the bottleneck link. PERT, PERT-ER and TCP-SACK is used as the underlying transport protocol for CBR flows, and their performance compared to see how AQM and ER can improve the performance of delivering real time media. Both FTP and HTTP background flows use TCP-SACK as their transport layer protocol.

In order to evaluate the improvement brought by ER and AQM under different conditions, we fix the number of CBR flows to be the same as the number of FTP flows, and vary the number of those two flows to create a different ratio between achievable bandwidth and CBR rate, which is 300kbps. From now on, we will refer to this ratio as T/μ . [1] The number of HTTP flows is fixed at 200. The number of flows are picked such that their T/μ fall into the range of $[1.2-2.4]$. In addition to that, we also vary the buffer size at bottleneck link from 0.5 $\times$ BDP to 2.0 $\times$ BDP to obtain a drop rate range.

Each simulation was run for 10 times with a duration of 1000 seconds, during which both the CBR and FTP flows started randomly from 0 to 50 seconds.

B. Simulation Results

In this section, we classify the sample flows into different categories according to their T/μ value. Figure. 3 shows the fraction of successful streams of all the sample CBR streams whose T/μ fall into the region of [1.2, 1.6], [1.6, 2.0] and [2.0, 2.4]. Recall that we define a stream to be successful if only 0.01% of its packet arrives later than its play out deadline. As we increase the **SUD** , the deadline of each packet is essentially postponed, therefore the curves of successful streaming rate will increase monotonically as **SUD** increases. In addition to that, it can also be observed that successful streaming rate increases as T/μ increases, this is because a higher T/μ means a higher average bandwidth available for streaming.

We can see from Figure. 3 that SACK can only roughly achieve 100% success when **SUD** is 20 seconds and T/μ is in the range [2.0, 2.4]. This matches the results in [17], where TCP-SACK is shown to provide satisfactory streaming performance only when T/μ is greater than 2.0. PERT, on the other hand, can a achieve a success rate very close to 100% when **SUD** is 10 second and T/μ is greater than 1.6, which shows PERT's AQM scheme can truly provide a very big improvement to TCP. This finding is also very identical to the ones in [4].

When ER is integrated to PERT, we can see the performance is further improved, and the improvement increases as T/μ decreases. For instance, when **SUD** is 5 seconds, ER is able to help increase successful streaming rate by 16%, 4% and 0.5% for the three T/μ respectively. This is quite intuitive as a smaller T/μ means a larger number of flows competing for the limited bandwidth, and therefore packet drop will occur more often and so is retransmission timeouts. We believe ER can reduce more retransmission timeouts in this condition. Since performance in delivering streaming media is related to the latency caused by RTO, ER should be most effective when T/μ is low. To verify this claim, we collect the total time each TCP sender spent in waiting for a retransmission timer to expire. We will refer to this as Session Frozen Time (**SFT**), because not a single packet is flowing from sender to receiver during those time and the stream is frozen. **SFT** is going to be the metric used in this paper to evaluate latency caused by RTO.

Figure. 4 shows the average Session Frozen Time of the sample streams with T/μ ranging from 1.2 to 2.4. Error bars show 90% confidence interval. From this we can also see a clear improvement from SACK to PERT and from PERT to PERT-ER. What's more, the improvement ER brings to PERT also has similar pattern as success rate. Therefore, this serves as a good proof to our previous claim that the improvement in success streaming brought about by ER is attributed to its capability in avoiding retransmission timeouts and the difference it makes is most obvious in low T/μ cases.

To summarize, the results in this section show that by integrating ER into PERT, we can further improve its performance in delivering time sensitive media.

IV. ER vs. Fine Grained Retransmission Timer

By having a finer granularity in calculating RTO and removing the clamp on minimum RTO value, we will allow TCP to consider a packet dropped with a shorter waiting time, which will help recover from loss faster. However, being less "patient" will give the bottleneck less time to drain the backlogged packet at its buffer, and therefore may incur some penalty as the retransmitted packets can also get dropped.

In this section, we set the granularity and minimum RTO value[2] in PERT to be 1ms and compare its performance improvement with ER. Besides that, we also want to explore if PERT-ER can be further improved by having a fine grained RTO timer.

A. Simulation Methodology

The same simulation settings in Table. I are used in this section. Among all the test flows, we pick those with T/μ falling into the region of [1.0, 1.5] and collect successful streaming rates of PERT, PERT-ER and those two with fine grained RTO. Apart from that, drop rate, total number of timeouts and **SFT** are also collected.

B. Simulation Results

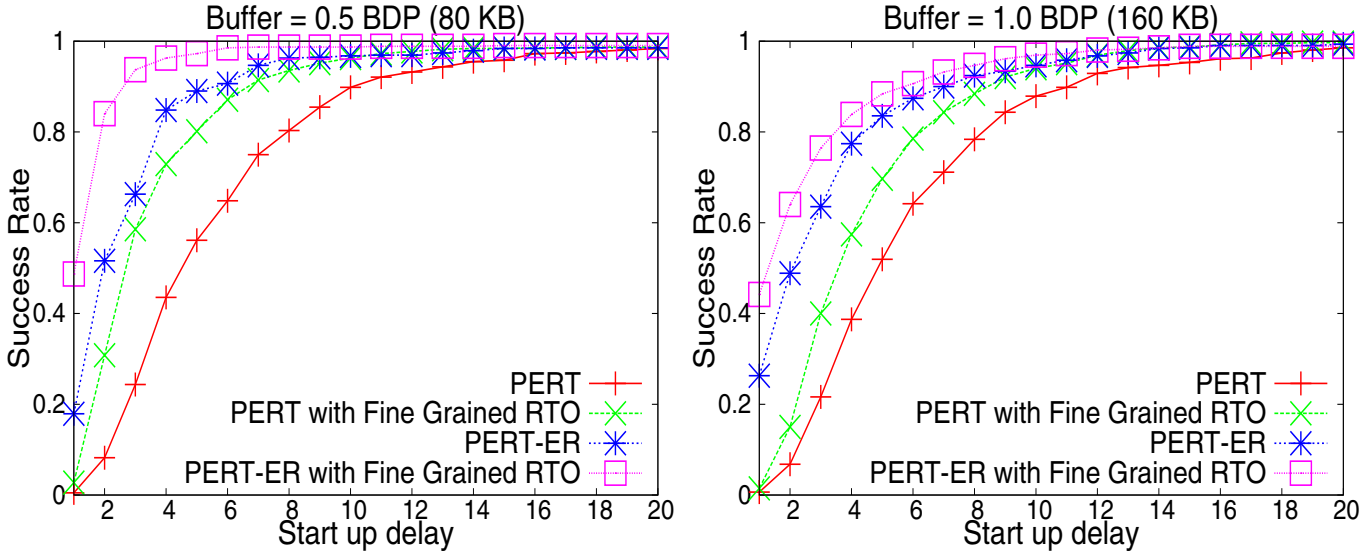
Figure. 5 and Figure. 6 show the successful streaming rate of all 4 buffer size cases. Firstly, we notice the performance of all protocols is negatively affected by the increase in buffer size. This is due to the nature of PERT and delay based protocol in general. When a big buffer is used, after overflow happens the bottleneck router won't have enough time to drain out all the backlog before packets starting to get backlogged again. Therefore, queuing delay will constantly stay at a high level, and PERT will attempt to reduce the queue size by performing early response more frequently. The loss based TCP-SACK, which is used by all the background traffic, will do the exact opposite: increasing sending rate until another buffer overflow occurs. Hence, throughput of PERT will be negatively affected. However, this doesn't prevent us from evaluating ER and fine

[1] This notation follows the naming convention in [17]. T is calculated by dividing the measured overall bandwidth of both the CBR and FTP flows by the total number of flows. μ is simply the CBR sending rate.

[2] *tcp_tick* and *minrto_* in *ns2*

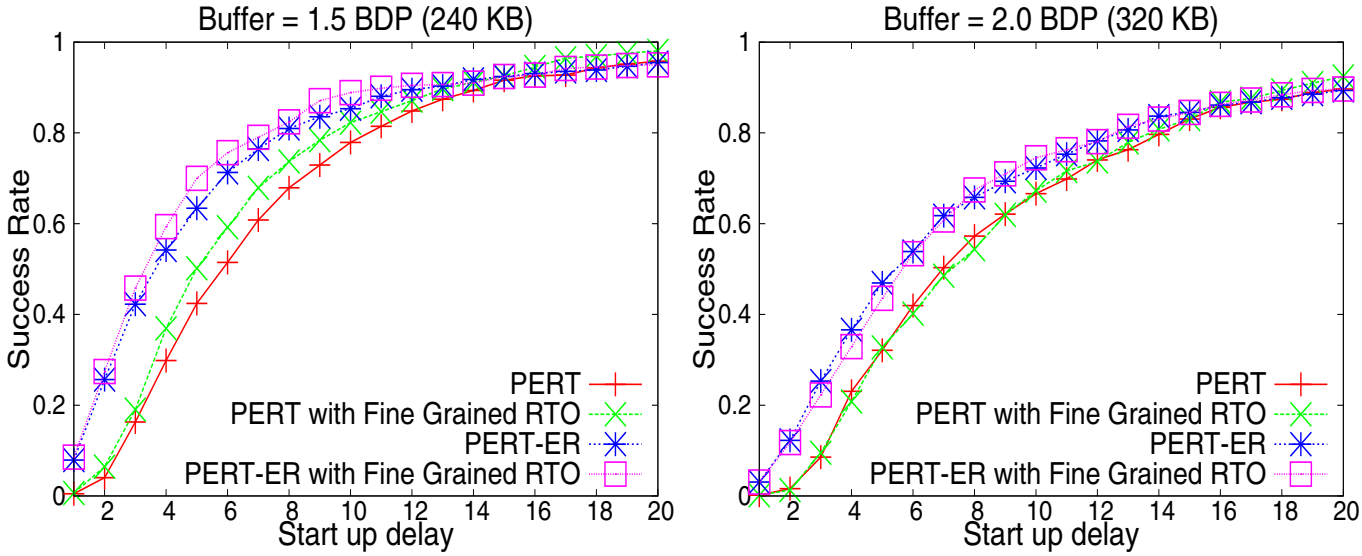
		PERT	PERT with fine grained RTO	PERT-ER	PERT-ER with fine grained RTO
80KB	**RTO Numbers**	253	335 (82)	96	149 (53)
	SFT(sec)	61	40 (-21)	29	14 (-15)
160KB	**RTO Numbers**	140	205 (65)	43	83 (40)
	SFT(sec)	33	28 (-5)	12	9 (-3)
240KB	**RTO Numbers**	80	117 (37)	21	35 (14)
	SFT(sec)	18	16 (-2)	5	5 (0)
320KB	**RTO Numbers**	52	72 (20)	14	25 (11)
	SFT(sec)	9	11 (+2)	3	5 (+2)

TABLE II: RTO Number and **SFT** Comparison



(a) Buffer=80KB (b) Buffer=160KB

Fig. 5: Fraction of Successful Stream: fine grained RTO , 80KB, 160KB



(a) Buffer=240KB (b) Buffer=320KB

Fig. 6: Fraction of Successful Stream: fine grained RTO , 240KB, 320KB

grained RTO , as what we focus on is the improvement brought to PERT by the two techniques, rather than the absolute result.

To start with, it's clear that fine grained RTO can increase the success rate of both PERT and PERT-ER. That is most obvious in small buffer cases. For instance, when **SUD** is 5 seconds and buffer is 0.5 BDP, fine grained RTO is able to improve success rate of PERT by 24% and PERT-ER by 8%, when buffer is 1.0 BDP, PERT is improved by 18% and PERT-ER is improved by 6%. On the other hand, we also notice the improvement diminishes quickly as buffer size increases.

What's more, compared to ER, fine grained RTO's help is not as effective. ER is able to increase PERT's success rate by 33%, 28%, 22% and 15% in the 4 buffer cases when **SUD** is 5 seconds, whereas fine grained RTO can improve PERT only by 24%, 18%, 8% and 1% in the same condition.

To further explore the effect of fine grained RTO , we collect the average RTO Number and SFT of all the 1240 flows involved in the test and list them in Table. II.

Table. II shows that when fine grained RTO is applied, PERT and PERT-ER both have a higher number of timeouts than before. When buffer size is 80KB, for example, fine grained RTO increase PERT's average timeout number by 82 and PERT-ER's average timeout number by 53. This verifies our concern that removing the lower bound on the timeout value and increase its granularity will incur some extra penalty. On the other hand, it is also shown that the **SFT** is reduced because of fine grained RTO . The is because when small buffer is applied, the value computed according the RTO formula is much smaller than the 200ms clamp on minimum value. Therefore, even though total number of timeouts are increased, the time saved in waiting for each RTO to expire still outweighs the penalty.

When looking at the impact of buffer size on fine grained RTO's ability in reducing latency caused by RTO, we observe a similar trend as what is found in success rate results: the larger the buffer size, the smaller the improvement. This shows that fine grained timer's improvement in success streaming rate is due to its ability in reducing the latency caused by retransmission timeouts. As for the reason why fine grained timer's performance is negatively affected by large buffer size, we believe there are two: 1) Drop rate gets lower as buffer size increases. 2) Fine grained timeout value gets closer to the 200ms clamp.

When comparing the ability of fine grained RTO and ER in reducing the latency caused by RTO and improving successful streaming rate, our results show that ER's performance is consistently better. We believe this is because the different approaches taken by the two techniques. Fine grained RTO simply relies on a smaller timeout value and achieves the objective by saving the time spent in each timeout, whereas ER is aimed at proactively avoiding timeouts, which is the reason why it's performance is not affected by big buffer as much.

Figure. 5, Figure. 6 and Table. II also shows that when ER and fine grained RTO are combined, latency can be further reduced and success rate can be increased greater.

V. Conclusion

This paper presents the result of incorporating an early retransmission approach into a delay based TCP variant with an aim to improve its performance in real-time streaming media delivery.

Our simulation results show that ER is able to help PERT reduce the latency caused by retransmission timeout in various settings. As a result, constrained streaming using PERT-ER can have a higher success rate.

Moreover, compared to fine grained retransmission timeout, ER's effective in reducing latency is more effective, and the combination of the two technique can produce a even better performance.

Acknowledgments

This research was supported by the Qatar National Research Fund (QNRF), under the National Priorities Research Program (NPRP) project number 26-6-7-10 and 08-152-2-043.

References

[1] H. Kohler and F. S., "Problem statement for the datagram congestion control protocol," *Internet RFC 4340*, 2006.

[2] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," *SIGCOMM Comput. Commun. Rev.*, vol. 30, pp. 43–56, August 2000. [Online]. Available: http://doi.acm.org/10.1145/347057.347397

[3] S. Boyden, A. Mahanti, and C. Williamson, "Tcp vegas performance with streaming media," in *Performance, Computing, and Communications Conference, 2007. IPCCC 2007. IEEE Internationa*, April 2007, pp. 35–44.

[4] B. Qian and A. L. N. Reddy, "Measurement and performance study of pert for on-demand video streaming," in *To appear in Proceedings of the 8th International Workshop on Protocols for Future, Large-Scale & Diverse Network Transports*, ser. PFLDnet '10, 2010.

[5] A. Goel and C. Krasic, *Low-Latency Adaptive Streaming over TCP*.

[6] S. Liang and D. Cheriton, "Tcp-rtm: Using tcp for real time multimedia applications," *in International Conference on Network Protocols*, 2002.

[7] M. C. J. Chung and R. Kinichi, "Mtp a streaming friendly transport protocol," *Technical Report*, May 2005.

[8] Z. Yin, H. Alnuweiri, and A. Reddy, "Performance of early retransmission scheme in streaming media," in *Telecommunications (ICT), 2010 IEEE 17th International Conference on*, 2010, pp. 613 –620.

[9] P. Sarolahti and A. Kuznetsov, "Congestion control in linux tcp," in *Proceedings of USENIX*. Springer, 2002, pp. 49–62.

[10] K. Fall and S. Floyd, "Simulation-based comparisons of tahoe, reno and sack tcp," *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 3, pp. 5 – 21, July 1996.

[11] D. Lin and H. Kung, "Tcp fast recovery strategies: analysis and improvements," in *INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, 1998, pp. 263 –271 vol.1.

[12] L. S. Brakmo and L. L. Peterson, "Tcp vegas: End to end congestion avoidance on a global internet," *IEEE Journal on selected Areas in communications*, vol. 13, pp. 1465–1480, 1995.

[13] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, pp. 397–413, August 1993. [Online]. Available: http://dx.doi.org/10.1109/90.251892

[14] S. Bhandarkar, A. L. N. Reddy, Y. Zhang, and D. Loguinov, "Emulating aqm from end hosts," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 349–360, 2007.

[15] K. Kotla and A. L. N. Reddy, "Making a delay-based protocol adaptive to heterogeneous environments," 2008.

[16] J. Boyce and J. Gaglianello, "Packet loss effects on mpeg video sent over the public internet," *In Proceedings of ACM Multimedia*, pp. 181 – 190, September 1998.

[17] B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "Multimedia streaming via tcp: an analytic performance study," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 406–407, June 2004.