PERFORMANCE OF EARLY RETRANSMISSION SCHEME AND DELAY BASED PROTOCOL IN VIDEO STREAMING

A Thesis

by

ZHIYUAN YIN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2011

Major Subject: Electrical Engineering

PERFORMANCE OF EARLY RETRANSMISSION SCHEME AND DELAY BASED PROTOCOL IN VIDEO STREAMING

A Thesis

by

ZHIYUAN YIN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Co-Chairs of Committee,	Hussein Al Nuweiri
	A. L. Narasimha Reddy
Committee Members,	Dmitri Loguinov
	Srinivas Shakkottai
Head of Department,	Costas N. Georghiades

May 2011

Major Subject: Electrical Engineering

ABSTRACT

Performance of Early Retransmission Scheme

and Delay Based Protocol in Video Streaming. (May 2011)

Zhiyuan Yin, B.En., Shanghai Jiao Tong University

Co–Chairs of Advisory Committee: Dr. Hussein Al Nuweiri
Dr. A. L. Narasimha Reddy

In this paper, we propose an early retransmission scheme to improve TCP's performance in delivering time-sensitive media. Our extensive *ns2* simulations show significant improvement. When integrated into a traditional TCP variant, namely TCP-SACK, the early retransmission scheme can substantially reduce the latency caused by retransmission timeout. As a result, it can help TCP-SACK achieve a considerably higher success rate in delivering real time media. Early Retransmission also enhances the performance of a delay-based TCP variant, namely PERT. Furthermore, we also explore the improvement brought by employing a fine-grained retransmission timer, and compare it with ER. We find out that ER outperforms the fine grained timer in a variety of conditions and the combination of the two can further improve performance.

ACKNOWLEDGMENTS

I would like to thank Dr. A. L. Narasimha Reddy, Dr. Hasari Celebi and Dr. Hussein Al Nuweiri for their great help in this work.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

TCP's proven stability and scalability has made it the most widely used transport layer protocol for more than twenty years. However, as multimedia applications become ubiquitous over the Internet, TCP has been found incapable of meeting their requirements, as its flow control, congestion control and loss recovery can introduce extra latency and rate fluctuation when delivering streaming media. Because of that, many multimedia applications turn to UDP as their underlying transport protocol. However, as UDP doesn't come with any congestion control mechanism, it's now the application's responsibility to implement rate control, and many of those application level rate control algorithms have similar behaviors as TCP. Apart from that, the fact that UDP only provides best effort delivery also causes problem since nowadays media content are normally significantly compressed, which makes them very sensitive to loss [1]. Finally, study shows TCP is still being used for streaming on a large scale [2] [3]. All those make exploring ways to improve TCP's performance in delivering streaming media a meaningful topic.

This paper focuses on two inherent nature of TCP that are undesirable for media streaming. Its emphasis on reliable in-order delivery that introduces latency which interrupts media play-out[4] and its coarse-grained Retransmission Timeout (**RTO**) along with the back-off mechanism which is detrimental to any real-time based application[5].

We show that by integrating an Early Retransmission (**ER**) scheme into typical TCP implementation (specifically, TCP-SACK), we can reduce Retransmission

---

The journal model is *IEEE Transactions on Automatic Control*.

Timeouts to a great extent, keep packet delay at a significantly lower level, and lower the achieved throughput requirement by 20% to achieve zero late packet when delivering real-time media. What's more, combining ER into another delay based TCP variant, namely Probabilistic Early Response TCP (**PERT**) can help further improve its performance in low bandwidth network conditions. Finally, having a fine grained Retransmission Timer can also improve TCP's performance in small bottleneck buffer scenario and the combination of ER and fine grained RTO can give us a even better result.

The rest of the paper is organized as follows: Chapter II introduces some background information on the congestion control algorithms of all the TCP variants that are in the scope of this paper, as well as the related work to improve TCP's performance in delivering time-sensitive media. Chapter III provides the details of Early Retransmission. Chapter IV presents the simulation based performance analysis on the improvements caused by integrating ER into one typical TCP implementation, namely TCP-SACK and the delay based TCP-PERT. Chapter V shows the comparison result between ER and fine grained RTO . Conclusions and future directions are presented in Chapter VI.

CHAPTER II

BACKGROUND AND RELATED WORK

A. Background

Congestion control in TCP is achieved using four distinct algorithms, each of which plays an important part. The first of those is Slow Start (**SS**), which starts TCP slowly to avoid congestion at early stage[6]. The second is Congestion Avoidance, which aims at probing for available bandwidth by slowly increasing the number of maximum outstanding packets, i.e. Congestion Window (**CWND**) Size, while preventing congestion from happening. The other two are closely related to each other: Fast Retransmission and Fast Recovery (**FR/FR**). They help TCP achieve reliable transmission while preserving throughput. The difference between Tahoe, Reno, NewReno, and SACK lies in the different approaches taken to implement those four features.

1. TCP Tahoe, Reno, NewReno, and SACK

These four variants all share the same algorithm in SS and CA, which is shown in Table I. In short, in SS state, CWND is increased exponentially in order to acquire bandwidth quickly enough. In CA state, CWND is increased linearly to probe for available bandwidth in a conservative way.

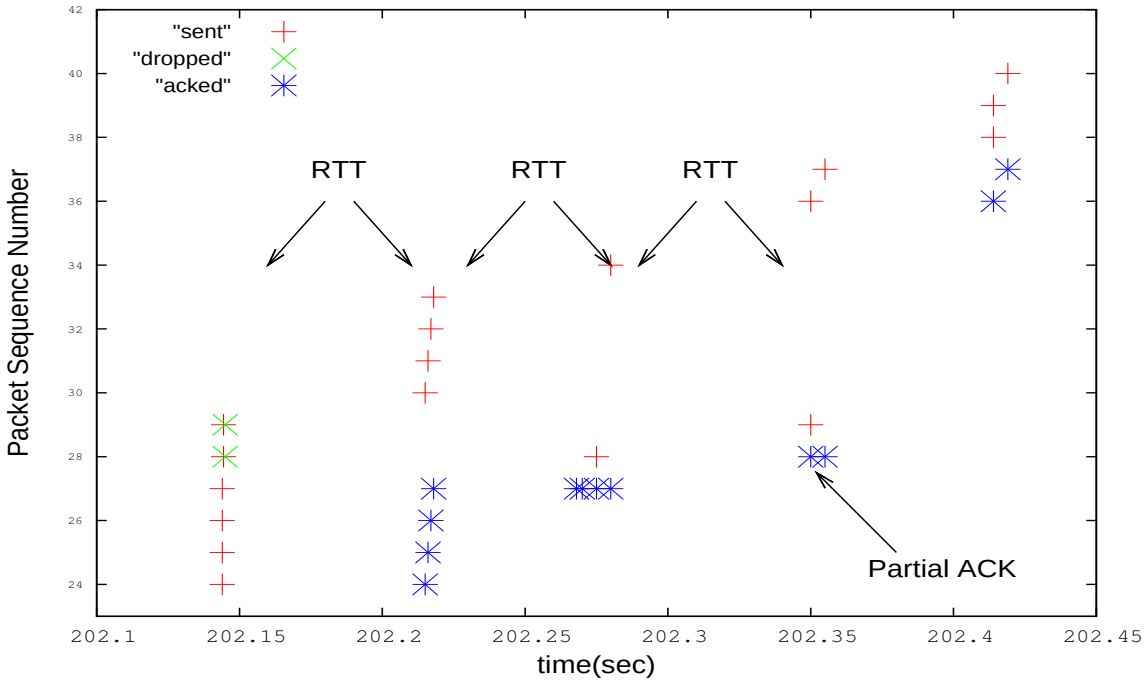
However, they do have different approaches in Fast Retransmission/Fast Recovery stage, which is summarized in Table II. To be concise, in NewReno and SACK, the new ACK that comes after retransmission will not pull the sender out of fast recovery unless it has a higher sequence number than any of the packet sent before retransmission. This scheme, which is referred to as **Partial ACK** in [7], enables NewReno and SACK to recover from multiple packet drops in one CWND without

Table I.: Difference in SS and CA Scheme between Vegas and Other TCP Variants

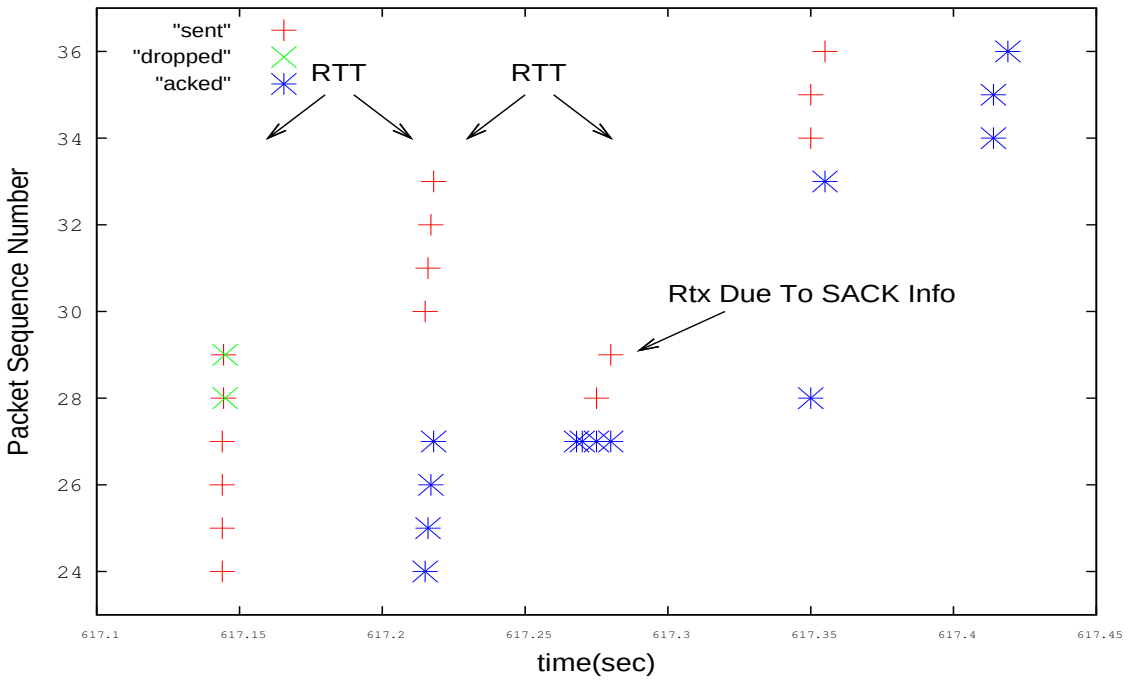
	TCP	**Vegas**	**PERT**
SS:	Init: cwnd = 1 Each New ACK: cwnd++ until *ssthresh* is reached.	Init: cwnd = 2. cwnd++ every other RTT. Switch to CA when *actual* throughput drops below *expected* throughput.	Same as TCP. No early response in Slow Start.
CA:	cwnd += 1/cwnd for each new ACK.	*actual* = cwnd/RTT *expected* = cwnd/baseRTT *diff* = *expected-actual* If *diff* < α; cwnd ++; If *diff* > β; cwnd --; Else cwnd unchanged	With p_{early}, cwnd -= β * cwnd With $1 - p_{early}$, cwnd += α/cwnd

invoking Retransmission Timeout. For the improvements brought by the introduction of Partial ACK and its importance in avoiding timeouts, please refer to [7], where insightful simulation-based comparison is made.

For the extra performance enhancement brought about by SACK, Figure 1 shows how NewReno and SACK behave differently in the same case where two packets are dropped in one CWND. Although both protocols successfully recovered from packet drop without resorting to RTO. SACK was able to fill the gap at receiver's side within one Round Trip Time (**RTT**), whereas it took NewReno two RTTs to achieve that.



(a) New Reno



(b) SACK

Fig. 1.: Comparison of NewReno and SACK

2. TCP-Vegas

a. Congestion Avoidance

As discussed above, the most different feature of TCP Vegas is its CA scheme, which is performed once per RTT according to the logic shown in Table I, where α, and β are two static threshold values.

b. Loss Recovery

- **Retransmission:** TCP Vegas' retransmission scheme also differs from that of its counterparts in that it doesn't use 3rd duplicate acknowledgement (**3rd DUPACK**) as the signal to trigger fast retransmission. Vegas sender keeps track of the RTT of every packet sent and sets a fine grained timer based on those more precise measurements of the network conditions. Once a duplicate ACK is received, the RTT for that packet is compared to the timer. If the timer has expired, the sender will determine a packet drop occurred and retransmits the packet immediately. This is the so-called early retransmission (**ER**) feature.

  After the retransmission, the RTT of the first and second non-DUPACK is also compared with the fine-grained timer, and retransmission is also triggered if their timers expire. It's easy to find out that this feature is identical to the Partial ACK mechanism in NewReno and SACK, and according to [8], it is the second most contributive feature of Vegas in improving throughput in a high traffic load environment.

- **Recovery:** Vegas' another effective feature in recovering from loss lies in its Fast Recovery algorithm. Upon detecting a dropped packet, instead of halving the CWND like the rest of the TCP variants do, a Vegas sender will decrease

its CWND by only 1/4, which has two major effects: 1) During fast recovery, Vegas sender will only wait for half as many DUPACKs as Reno to send out new packets. 2) After recovering from loss, Vegas sender's CWND will be 3/4 of what it was before, whereas it would be only half the CWND for other variants.

As mentioned by [8], this scheme contributes the most to its high throughput by allowing Vegas to "steal" bandwidth from Reno, therefore causing unfairness. In the later section, we will show that TCP-ER stays fair with TCP in a large scale, fast network condition.

3. PERT

a. Congestion Avoidance

As is summarized in Table I, PERT will also perform proactive slow down as Vegas did, but with 4 major differences:

- To avoid inaccurate measurement of RTT, which may be due to noise and busty traffic, PERT uses the Exponentially Weighted Moving Average (**EWMA**) of RTT with a weight of 0.99, which is refer to as $srtt_{0.99}$, to help predict network congestion.

- Using $srtt_{0.99}$, PERT will compute an early drop probability p_{early} with a given response function. For each new ACK, PERT will perform proactive slow down with a probability of p_{early}. The early response function can be chosen such that PERT will emulate AQM at end host. Within the scope of this thesis, we focus on PERT emulating "gentle" variant of Random Early Drop (**RED**), whose early response function is shown in Figure 2.

- Instead of decreasing **CWND** linearly during proactive slow down, PERT de-

creases multiplicativly with a fraction of β, which is computed as $\frac{\text{Current Queuing Delay}}{\text{Current Queuing Delay+Max Queuing Delay}}$. The rationale behind this is that when we perform early response when router queue is not full, it will take less time for the queue to be drained, and therefore halving congestion window by half will be overkill. Also, relating the window decrease factor with queuing delay will mitigate the impact of false alarm, and help PERT maintain high throughput in a high noise environment. Similar approach is used in TCP-Veno.

- To be able to compete with other loss based protocol, PERT will increase CWND additively by $\frac{\alpha}{\text{CWND}}$, where $\alpha = 1 + \frac{\text{Early Drop Probability}}{\text{Drop Probability}}$.

The performance of PERT in homogeneous environment has been explored in [9], and [10] addressed the problem of PERT not being able to compete with loss based protocols in a heterogeneous environment by associating the window increase factor with drop rate and early reaction rate. In later chapters, we will show that the integration of ER can further help improve PERT's performance, which is most obvious when available bandwidth is low.

B. Related Work

Researchers have proposed several approaches to improve TCP's performance in delivering real time media as well as alternatives to TCP.

TCP friendly congestion control protocols without packet retransmission such as [12] 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

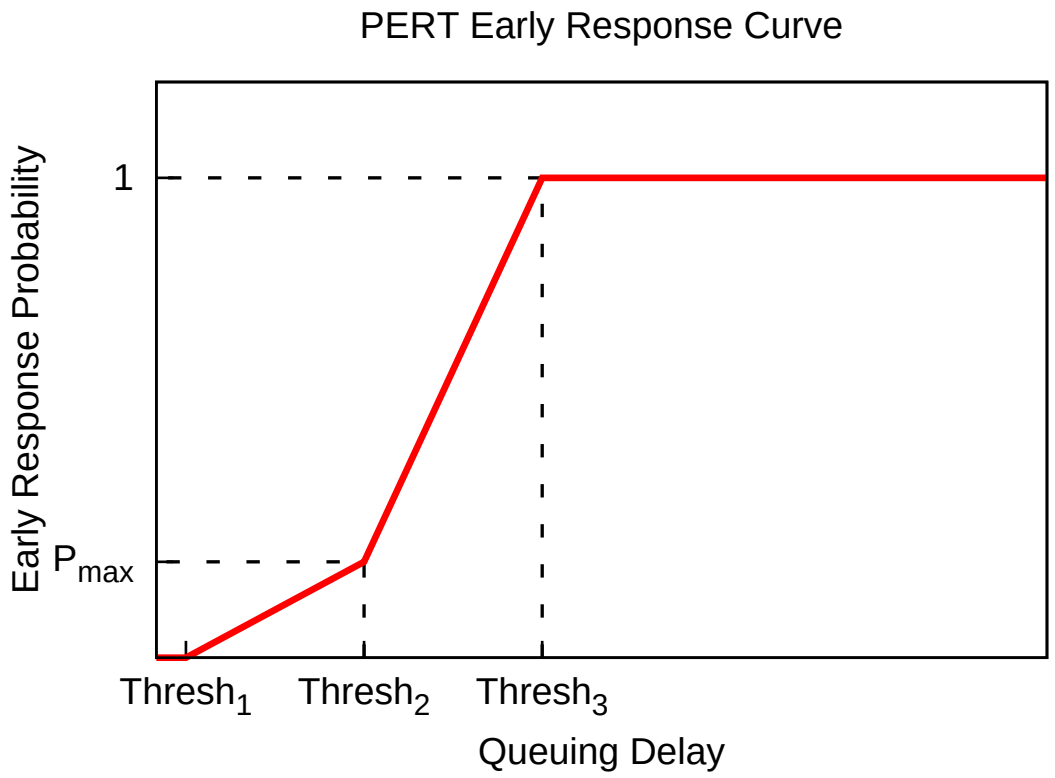


Fig. 2.: Probabilistic Response Function

to overcome this problem.

In [13], a discrete-time Markov model was developed for both live and stored video streaming over TCP. Using this model and extensive *ns2* simulations, they find out TCP requires the achievable bandwidth to be roughly twice the streaming rate to achieve successful streaming. Besides that, they also find out that in the case of live streaming, fraction of late packets increase as video length increase. In the case of stored streaming, the situation is the other way around.

Attention has also been focused on evaluating performance of delay based TCPs. It has been shown in [14] that TCP-Vegas only require the achievable bandwidth to be 1.5 times the streaming rate to achieve successful streaming. Qian *et al.* find out

for PERT, the requirement is 1.6 times the streaming rate[11].

Finally, various researchers have put effort in reducing the latency induced by TCP's inherent nature. Goel *et al.* proposed an approach to reducing the latency caused by TCP's sender side buffering by dynamically tuning TCP's send buffer[15]. Chung *et al.* 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[4]. Our approach, TCP-ER which also aims at reducing this latency by avoiding Retransmission Timeouts can therefore be classified into this category.

Table II.: Difference in Fast Retransmission/Fast Recovery between Tahoe, Reno, NewReno and SACK

	FR/FR
Tahoe:	Triggered by *Three* DUPACKs, *cwnd* = *cwnd* / 2. Go to SS.
Reno:	Triggered by *Three* DUPACKs, *cwnd* = *cwnd* / 2, *dupwnd* = 3. For every DUPACKs received, inflate *dupwnd* by one and send a new packet if permitted by *cwnd+dupwnd*. Go to CA if new ACK received.
NewReno:	Same as Reno except: When triggered, set *recov* = *maxseq*. Go to CA only if new ACK is not lower than *recov*.
SACK:	Same as NewReno except: SACK field notifies sender of the dropped packets, which will be rtxed before any new packets.

CHAPTER III

EARLY RETRANSMISSION

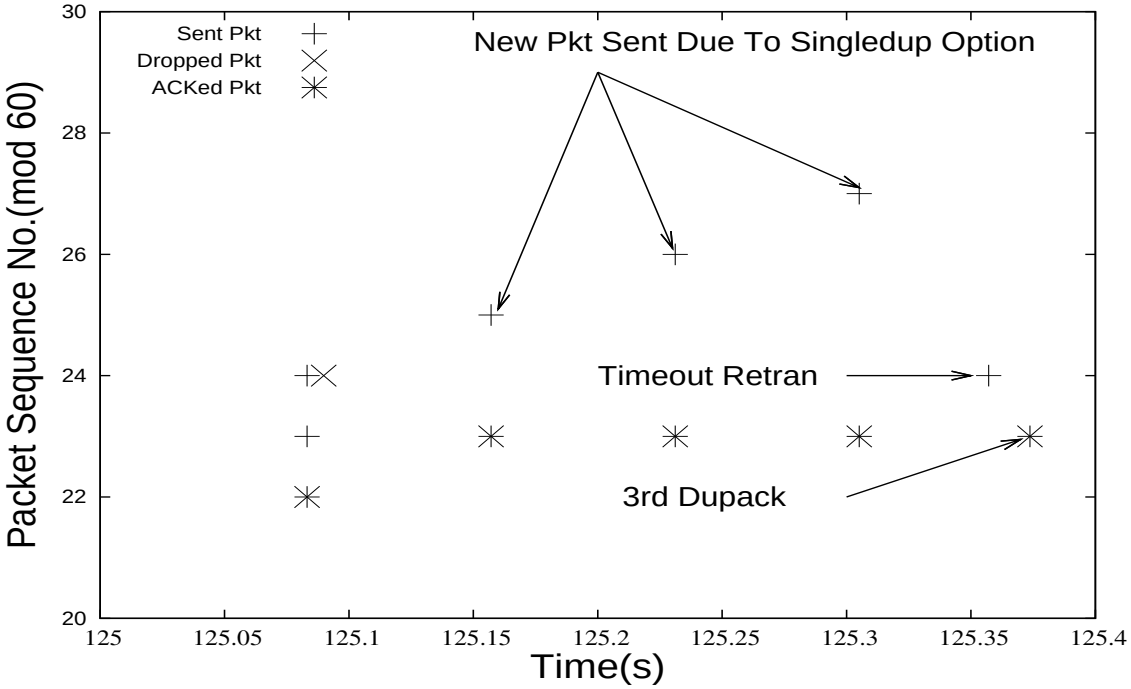
A. Problem With SACK

Despite the improvement brought about by adding SACK into TCP, there are still cases where SACK cannot prevent RTO from happening, which is when not enough DUPACKs are generated in time to trigger FR/FR, and all the improvements introduced by SACK become useless. TCP will fall back to RTO for helping. This case usually happen when multiple drops occurred in one window of packets[7] and when CWND size is small.

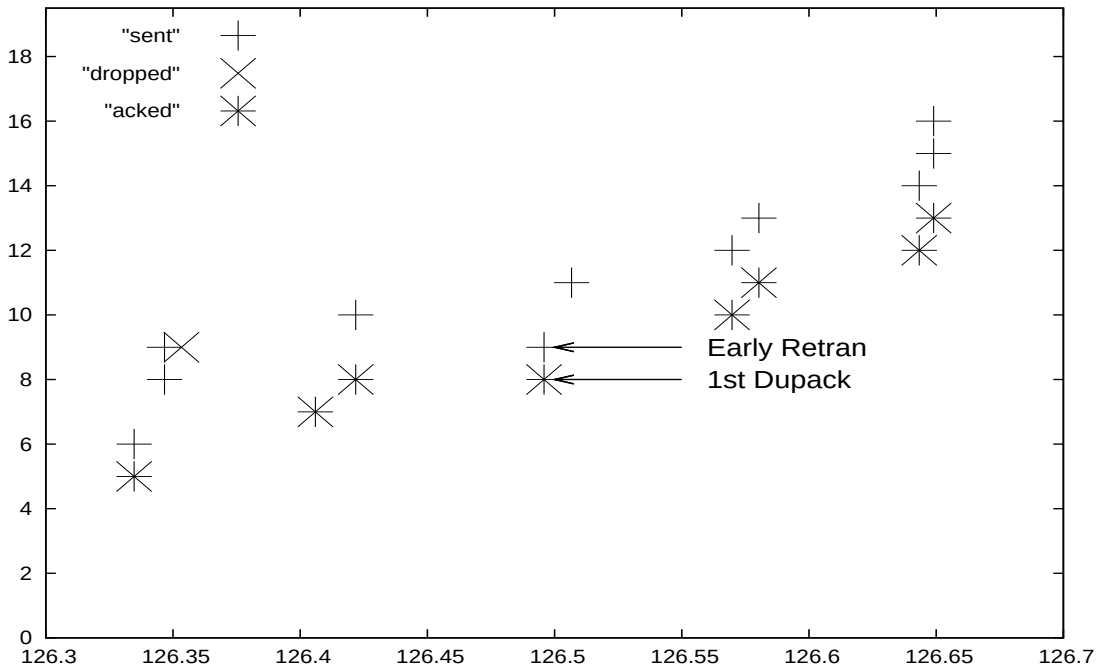
Figure 3a illustrates the problem of TCP's 3 duplicate ACK drop detection scheme when the current CWND is small. [1] It can be seen that even though the *singledup_* option is turned on, a retransmission timeout still happened before the arrival of the 3rd duplicate ACK. This also serves as a good example of the inter-packet arrival jitter caused by TCP's reliable transmission constraint[4], as all the high sequence numbered packet sent after the dropped packet will not be delivered to the application until RTO happened. The error recovery implementation of PERT is the same as that of traditional TCP, it will therefore suffer from the same problem.

On the other hand, if we allow TCP to start FR/FR without waiting for the arrival of the 3rd DUPACK, such a RTO can be avoided, as is shown in Figure 3b. Essentially, this means if we don't strictly follow the classic 3 DUPACK heuristic, the loss recovery process can be speed up by avoiding retransmission timeouts. As mentioned earlier, such an Early Retransmission (**ER**) scheme is employed by TCP-Vegas and is shown to contribute to its good performance by [14].

[1] Figure 3a and Figure 3b both use data obtained from *ns2* simulation runs



(a) SACK



(b) SACK-ER

Fig. 3.: Comparison of Early Retransmission and Normal Retransmission

B. Implementing ER

It has been shown in [16] that 91.1% of the packet reordering in the current Internet doesn't exceed a distance of 3 packet, and thus the 3 DUPACK threshold serves as a good heuristic for prevent packet reordering from having negative effect on throughput. But not sticking to the 3 duplicate ACK rules means we are at the risk of suffering from false alarm caused by packet reordering. Therefore, we need to find another heuristic to guard against that.

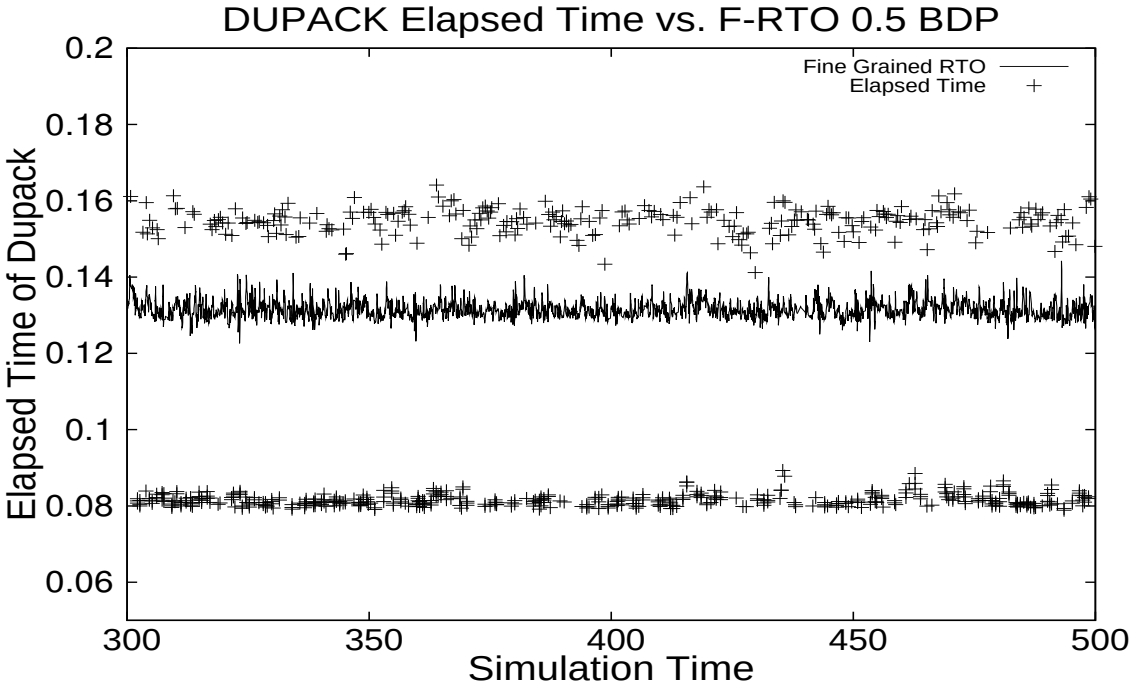
TCP-Vegas uses a fine grained timeout value as a heuristic, whose value is compared with the RTT of each DUPACK to determine whether enter FR/FR or not. We follow the same direction to implement TCP-ER with a fine grained timeout value.

Before introducing the way timeout value is obtained, let's first take a look at the arrival patterns of DUPACKs, as is shown in Figure 4. Using data obtained from a test run in the later sections, where RTT is 50ms and buffer size is 0.5 BDP and 1.0 BDP, we collect the time difference between the arrival of DUPACKs and the sending time of the packet that has the same sequence number as the DUPACK. We will refer to this time difference as **Elapsed Time** from now on. We can see two clear gathering of elapsed time in Figure 4. The lower gathering is composed of the DUPACKs generated by the packets from the same window as the dropped packet. Since drop-tail queue is used in the simulation, packet drop occurs when buffer overflows at the bottleneck. Hence, the lower gathering is around the maximum RTT experienced by the flow, which can be represented as

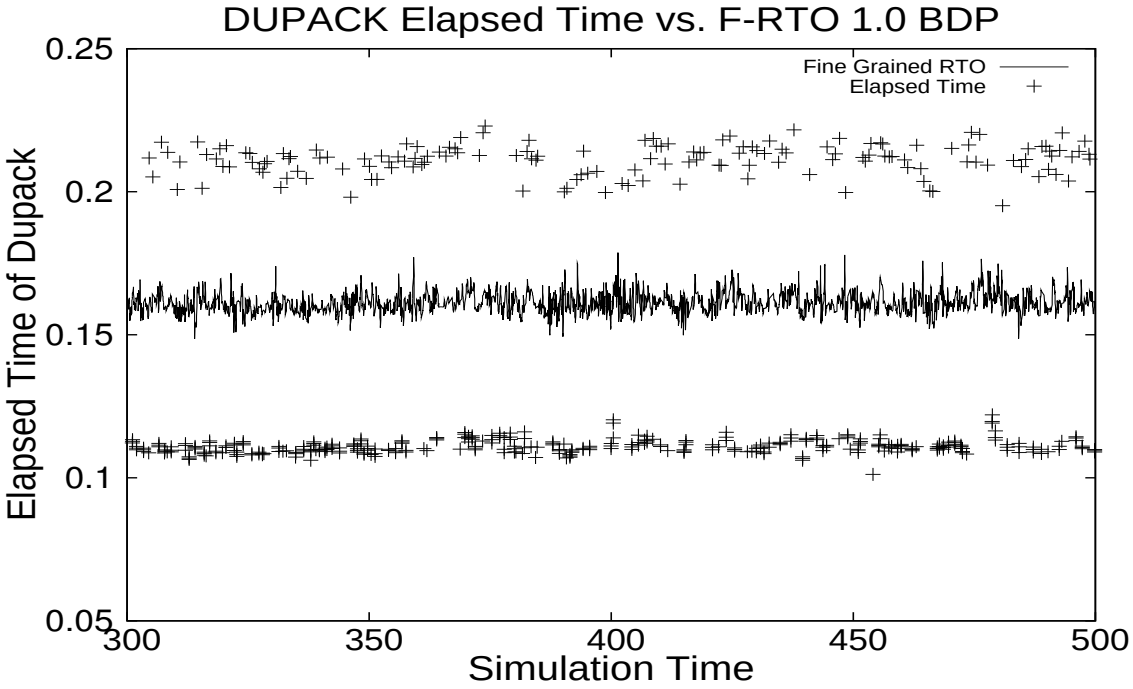
$$\mathrm{RTT}_{max} = \mathrm{Delay}_{propagation} + \mathrm{Delay}_{maxqueueing} \tag{3.1}$$

.

On the other hand, DUPACKs that acknowledge the same sequence number can



(a) 0.5 BDP



(b) 1.0 BDP

Fig. 4.: DUPACK Elapsed Time and Fine Grained RTO

be generated by packets sent in different windows, which is the reason why there are two gatherings in Figure 4.[2] The elapsed time of different gatherings should therefore be differed by one round trip time, and thus the elapsed time for the higher gathering can be represented as

$$\mathrm{RTT}_{max} + \mathrm{RTT'} \tag{3.2}$$

As congestion level at bottleneck buffer may be varied after overflows happens, different RTT' may be experienced, which is the reason why the higher gathering is not as concentrated as the lower one.

Given the above, we need our heuristic to allow DUPACKs only from the higher gathering to trigger ER. By doing that, we can tolerate packet reordering of one round trip time. To implement this, we need the timer value to be able to distinguish the two gatherings. RTT_{max} can be estimated using TCP-like RTO estimators. In our implementation

$$\mathrm{RTO(t)} = \mathrm{SRTT}_i + 2 \times \mathrm{SVAR}_i \tag{3.3}$$

is used as our estimator, where

$$\mathrm{SRTT}_i = 15/16 \times \mathrm{SRTT}_{i-1} + 1/16 \times \mathrm{SRTT}_i \tag{3.4}$$

and

$$\mathrm{SVAR}_i = 3/4 \times \mathrm{SVAR}_{i-1} + 1/4 \times \mathrm{VAR}_i \tag{3.5}$$

We use the minimum round trip time experienced RTT_{min} to provide a lower bound for RTT', and our threshold value can therefore be represented as

$$\mathrm{F\text{-}RTO} = \mathrm{SRTT}_i + 2 \times \mathrm{SVAR}_i + RTT_{min} \tag{3.6}$$

[2]If ER is not employed, there may be a third, or even a fourth gathering, until the TCP RTO value is reached. This is because of the *single_dup* option to keep the TCP fly wheel running.

Its value is also plotted in Figure 4, from which we can see F-RTO provides a good threshold value to differ the two gatherings, and therefore provide good performance in deciding when to enter Early Retransmission. One thing worth mentioning is that the classic 3 DUPACK rule is still left unchanged, and DUPACKs in the lower gathering can still trigger FR/FR.

CHAPTER IV

EXPERIMENTAL EVALUATION OF EARLY RETRANSMISSION

In this chapter, we performed exhaustive *ns2* simulation to evaluate ER's improvement to TCP's ability to deliver video streaming. We mainly focus on the integration of ER to two different TCP flavours: TCP-SACK and PERT.

A. Simulation Methodology

Throughout this paper, a dumbbell network topology shown in Figure 5 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 III.

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] HTTP flows are fixed at 200. The number of flows are picked

[1] T is calculated by dividing the measured overall bandwidth by the total number of flows. μ is simply the CBR sending rate.

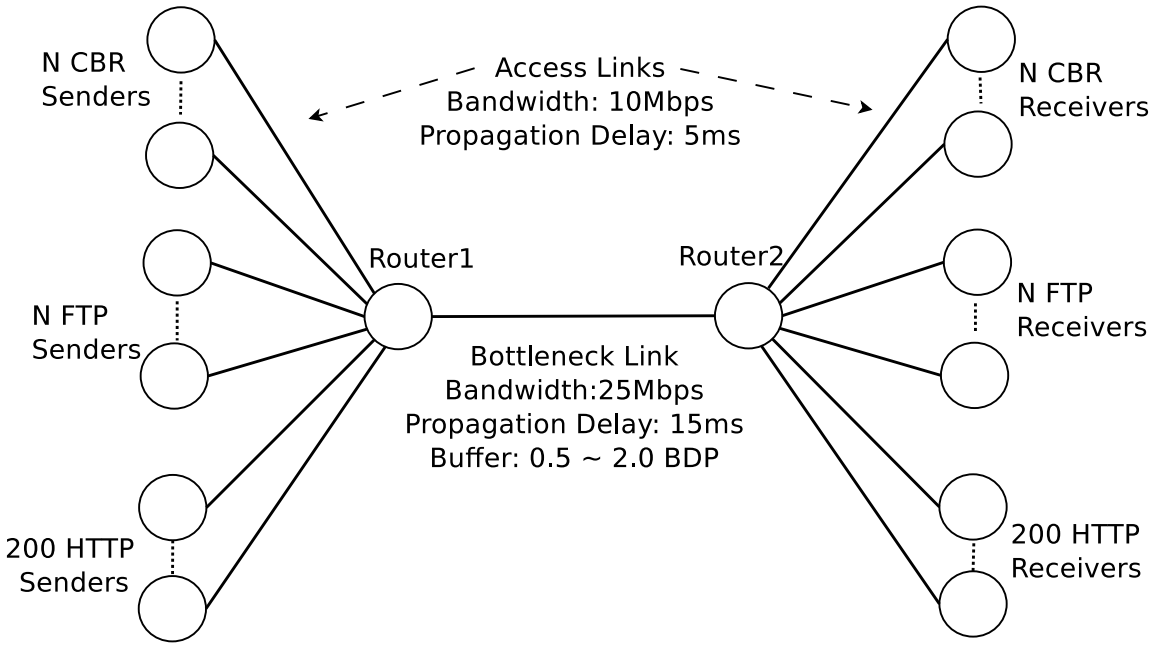


Fig. 5.: Dumbbell Network Topology

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 × BDP to 2.0 × BDP to obtain a drop rate range from 0 to 8%.

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. SACK vs. SACK-ER

1. Session Frozen Time

We define Session Frozen Time (**SFT**) to be the total time during which a TCP sender is starved with new ACK until an RTO occurs. The reason why we are interested in that is because we expect the ER feature to greatly reduce the number of timeouts and SFT is directly related to number of timeouts and loss in delivered QoS. We

Table III.: Simulation Setups

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

didn't simply measure the total number of RTOs due to the exponential back-off of Retransmission Timer. [2]

Figure 6 shows the average Session Frozen Time of the test runs as well as the error bars representing 90% confidence intervals.

It is obvious that SFT decreases as bottleneck buffer and T/μ increases. This is intuitive as increment in those two factors result in a lower packet drop rate, which is shown in Figure 7. However, the introduction of early retransmission does make a

[2]During simulation, we observed timeout value ranges from 0.2 seconds to as great as 16 seconds, and therefore, smaller number of timeouts doesn't necessarily mean good performance.

difference. Especially in the case of small buffer size and small T/μ cases. Note in all of the three cases shown, when buffer size is 80KB, SACK's average total frozen time is 80.58, 25.20, and 6.01 seconds respectively. On the other hand, if the early retransmission scheme is employed, SFT in those three extreme cases were reduced to 23.02, 4.72 and 1.22 seconds respectively, which are only 28.4%, 18.8%, and 20.3% of those without the ER scheme.

In Figure 7, we also plot the drop rate of all the test cases, which shows a very clear trend of decreasing as buffer size and T/μ increase. In addition to that, Figure 7 also shows that by doing ER, TCP'S drop rate doesn't get increased.

Essentially, Figure 6 and Figure 7 shows that under the same drop rate condition, ER can help TCP reduce the chances of relying on retransmission timeout for loss recovery.

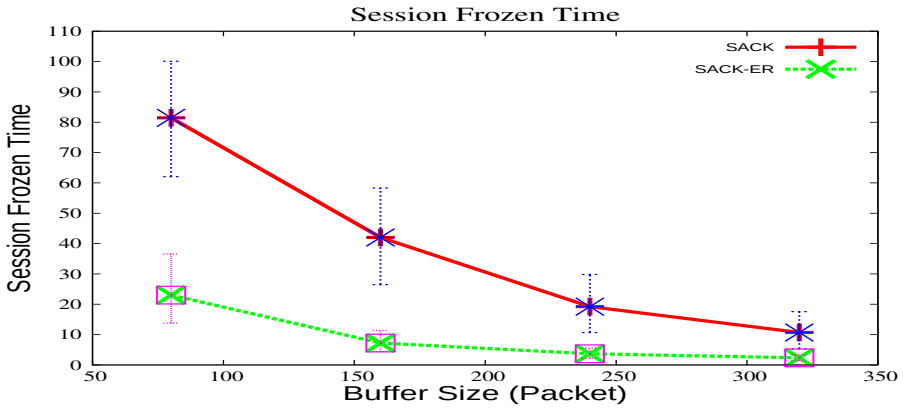
It's easy to find out the improvement is most obvious when T/μ is small and become less evident in high T/μ case. This shows the Early Retransmission and Loss Recovery scheme in SACK-ER have the most effect in avoiding timeouts under a severely congested environment. In later sections, we will show that the improvement in successful streaming rate also follows the similar pattern.

2. Standard Deviation of Inter Packet Arrival Time

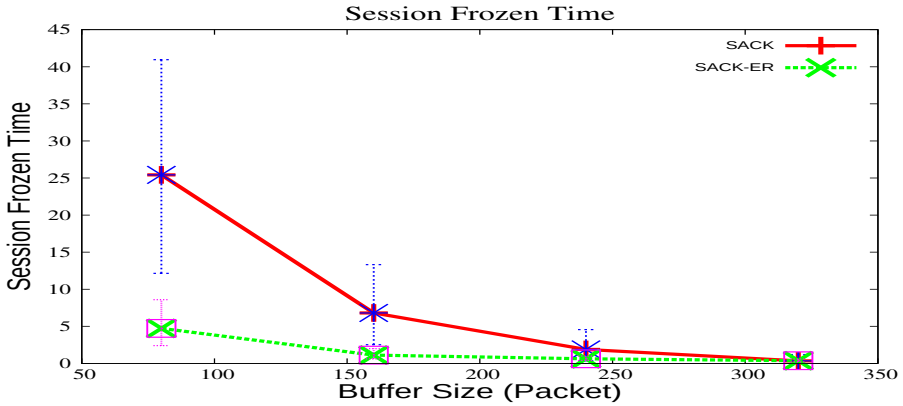
As discussed in [17], human perception is more sensitive to frame jitter than low quality video when watching streaming video. Standard deviation (**STDDEV**) of inter-packet arrival time (**IPAT**) was used here to evaluate delay jitter and was shown in Figure 8.

Figure 8 shows the average IPAT STDDEV and 90% confidence interval corresponding to the 10 test runs in the previous section.

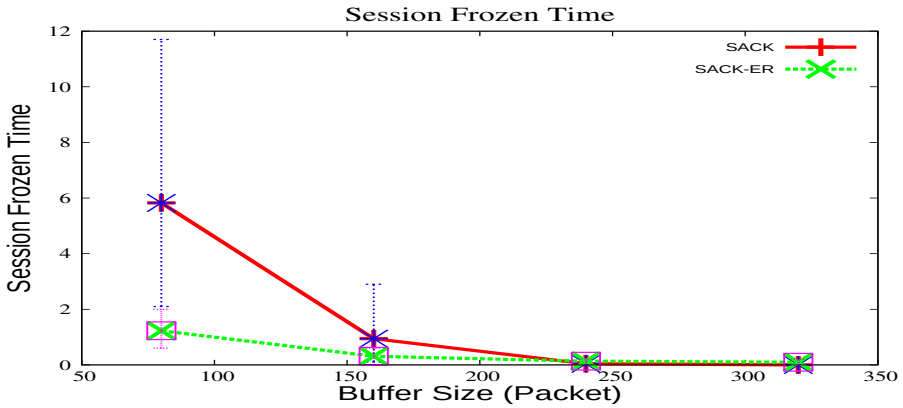
As can be seen from Figure 8, IPAT also decreases as T/μ and buffer size increase.



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

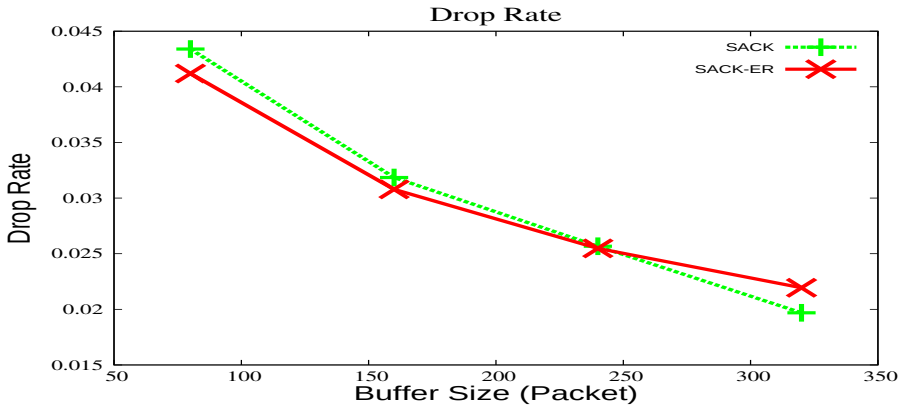


(b) $1.6 < T/\mu < 2.0$

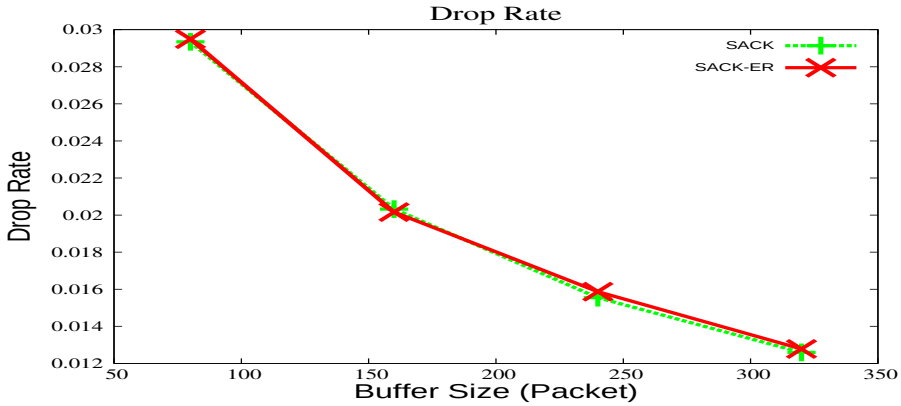


(c) $2.0 < T/\mu < 2.4$

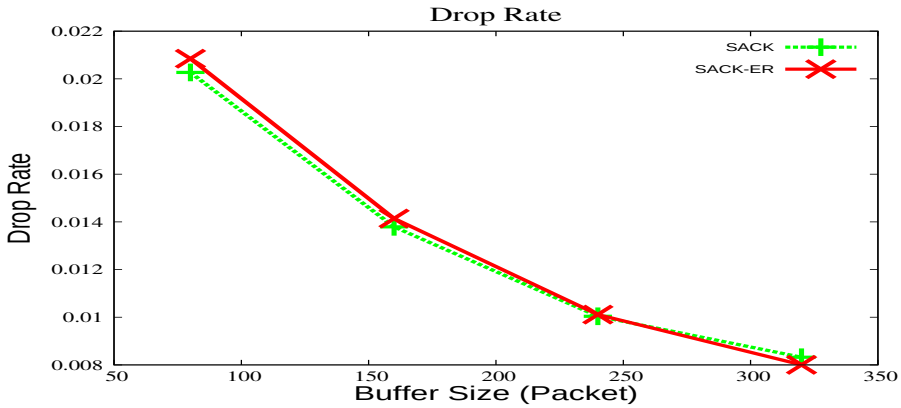
Fig. 6.: SACK vs. SACK-ER: Session Frozen Time



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

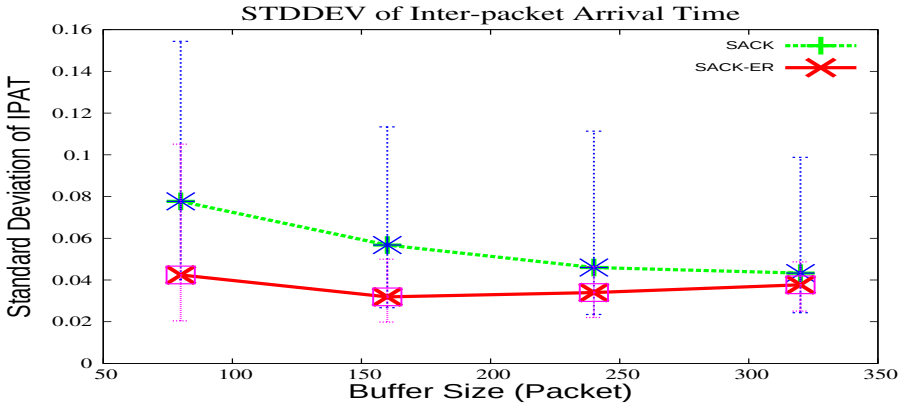


(b) $1.6 < T/\mu < 2.0$

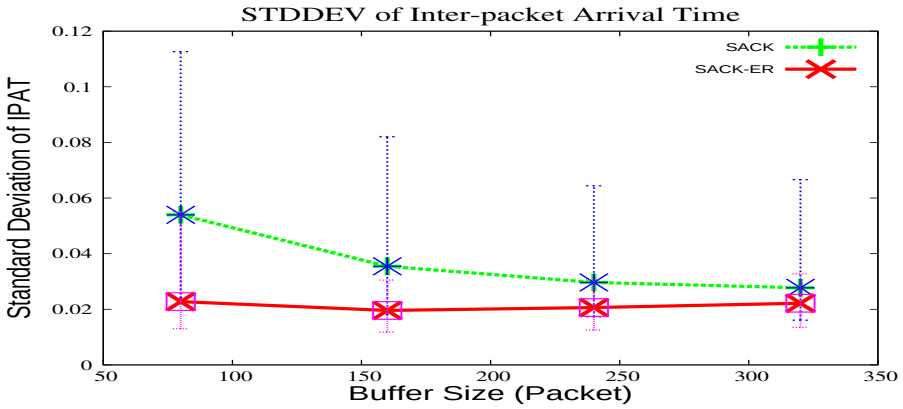


(c) $2.0 < T/\mu < 2.4$

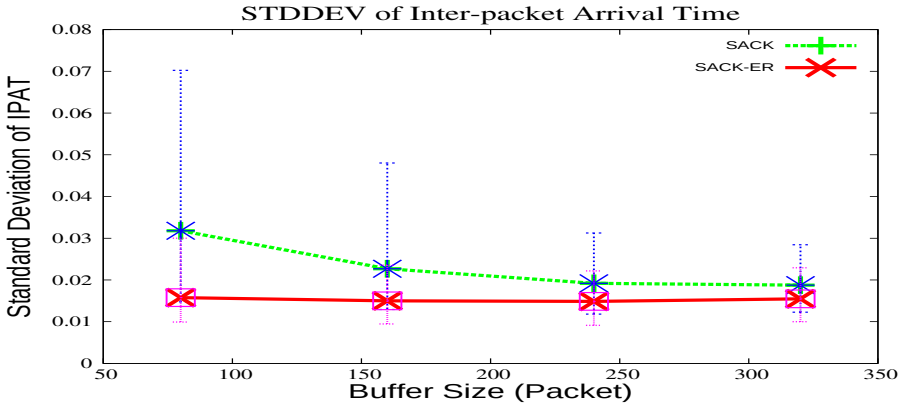
Fig. 7.: SACK vs. SACK-ER: Drop Rate



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



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



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

Fig. 8.: SACK vs. SACK-ER: IPAT STDDEV

The figure shows when bottleneck buffer size is 80KB, average standard deviation of IPAT of SACK is as high as 80ms, 58ms and 30ms in the three T/μ cases respectively. That is much bigger than the theoretical packet inter-arrival time, which is roughly 27ms for 300kbps CBR stream. The reason for this is as follows. To simulate the delay experienced by real-time applications, we didn't mark the packet as received until all the lower-sequence numbered packets arrived, thereby taking into account the delay introduced by TCP's retransmission and in order delivery. Because of that, in extreme conditions like the 80KB and [1.2, 1.6] T/μ case, RTO occurs often in SACK, and therefore delay caused by retransmission can be as high as several seconds. That contribute most to the delay jitter experienced by the application, which manifest themselves as huge spikes on the packet inter-arrival time curve. This is because packets with higher sequence numbers are stuck at the receiver's side buffer and cannot be delivered to the application until retransmission timeout occurred. What's more, after that hole is filled by a lengthy timeout retransmission, all the packets that are delayed in the receiver's buffer will all be delivered to the application at the same time, with "zero" inter-packet arrival time. Those are the reasons for the high standard deviation of IPAT , and the root cause of delay jitter introduced by TCP's reliable transmission.

On the other hand, SACK-ER has a consistently smaller delay jitter than SACK, which is especially obvious when drop rate is high. (40ms, 20ms and 15ms in the 3 T/μ cases when buffer size is 80KB, which is 40%, 50% and 50% of that of SACK) This is because ER can help TCP recover dropped packets with FR/FR and avoid RTO, and therefore limiting the delay jitter to several RTTs in most cases.

3. Successful Streaming Rate

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 [1] 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. [13] and [14] also use identical methods.

Figures 9 to 12 show the results of SACK and SACK-ER when 4 different buffer sizes are applied. It is also very obvious that both SACK and SACK-ER have a higher success rate as buffer size and T/μ increase, and SACK-ER has a higher success rate than SACK in all cases. As [13] claims, when 10 seconds of SUD is applied, TCP can roughly achieve zero late packets when T/μ is higher than 2.0. Our experiment also shows similar results, as SACK has a success rate close to 1 in all but the 80KB buffer size case, when SUD is 10 seconds. In addition to that, we also observe that when T/μ increase from [1.2, 1.6] to [1.6, 2.0], SACK has a much larger increase in success rate compared to when T/μ increase from [1.6, 2.0] to [2.0, 2.4]. This finding is also in accordance with those in [13].

When ER is integrated into SACK, we can see that close to 100% success can be achieved when T/μ is in the region of [1.2, 1.6] in all but the 80KB buffer case, which means ER can help lower TCP's requirement for success streaming by 20%. Besides that, in the small T/μ case, SACK-ER's success rate grows much faster that that of SACK's. In the worst case, SACK-ER can achieve 50% success rate when SUD is greater than 5 seconds, while SACK needs 15 seconds to achieve the same level.

The success stream results also matches the SFT results in Figure 6, where we see ER's improvement in reducing timeouts are most obvious in low T/μ and small buffer size cases. In the case of success rate, when SUD is 10 seconds and T/μ is in [1.2, 1.6], SACK-ER's success rate is 250%, 80%, 50%, and 40% higher than SACK's in the 80KB, 160KB, 240KB and 320KB buffer case respectively.

C. PERT-ER

In this chapter, we put PERT and PERT-ER into the same network environment as the one used for SACK and SACK-ER to evaluate how ER can improve the performance of a delay based protocol. To provide better comparison between all four TCP flavours, we also calculate the success rate of each protocol when different drop rates are experienced.

1. Session Frozen Time

Using the same simulation settings as in Table III, we put PERT and PERT-ER into test and perform similar measurements. The result is shown in Figure 13.

Again, we notice the clear trend of SFT decreasing as T/μ increase and buffer size increase. Also, it is obvious that PERT has a smaller SFT than SACK. When buffer size is 80KB (0.5 BDP), PERT has an average SFT of 50s, 17s and 3.2s in the three T/μ cases, which is 12s, 8s and 2.8s less than that of SACK. This is verified by the findings in [18]. Since PERT will try to reduce buffer queue size by proactively slowing down before congestion event happens, this result is quite intuitive.

When early retransmission is employed in PERT, we see that the SFT for 80KB buffer size case is further reduced to 22s, 5s and 1.5s, which is 28s, 12, and 1.7s less than that of PERT.

In addition to that, we also noticed that PERT-ER performs better than SACK-ER in low T/μ case. For instance, when T/μ is in the range of [1.2, 1.6] and buffer size is 0.5 and 1.0 BDP, SACK-ER has an average SFT of 23s and 7s respectively, whereas PERT-ER only has 16s and 5s in the above two cases. This difference also diminishes as T/μ increase, which is intuitive as drop rate will become smaller as buffer size and T/μ gets bigger, and both of them will suffer from very small number of timeouts.

2. Standard Deviation of Inter Packet Arrival Time

Figure 14 shows similar results with SACK, when bottleneck buffer is 80KB (0.5 BDP), PERT's IPAT STDDEV is 47ms, 32ms and 23ms in the three T/μ case, which is roughly 60%, 55% and 70% of SACK's IPAT STDDEV. With the help of early retransmission, we can see the standard deviation of IPAT is further reduced to 30ms, 20ms and 15ms in the three cases.

3. Successful Streaming Rate

a. Varying Buffer Size

Figures 15 to 18 shows the comparison between PERT and PERT-ER in the 0.5 BDP, 1.0 BDP, 1.5 BDP and 2.0 BDP case respectively. Unlike SACK, both PERT and PERT-ER show a slight decrease of success rate as bottleneck buffer size increases. This is not unexpected as [10] already shows PERT having problem competing with loss based protocol when big buffer size is applied.

Despite the fact that PERT's performance will be slightly affected in the face of big buffer, we see clear performance improvement from SACK to PERT. When T/μ is in [1.6, 2.0], PERT can achieve close to 100% success rate in all buffer size cases,

which is better than both SACK and SACK-ER. The reason for this improvement is out of scope, we refer the reader to [9] and [10] for more details.

When ER is applied to PERT, the improvement in performance also has similar pattern as that of SACK-ER. With the same buffer size, ER's help become less evident as T/μ increase. When T/μ is fixed, ER's improvement diminishes as buffer increases.

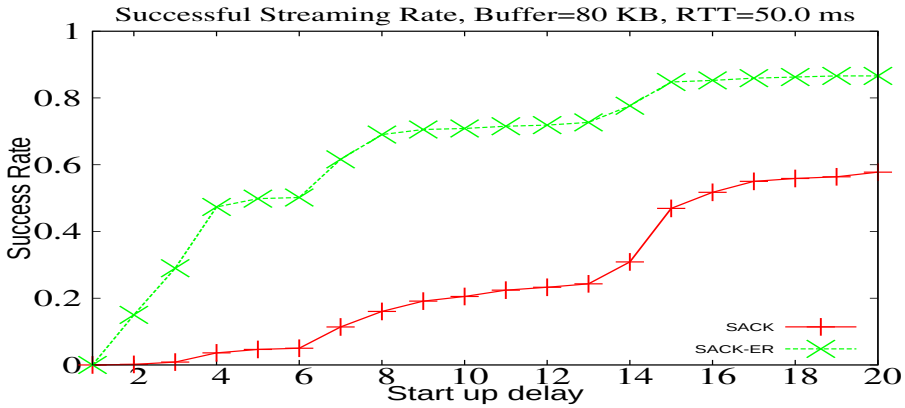
b. Varying Drop Rate

To provide a overall view of how AQM and Early Retransmission can improve TCP's performance, we collected the drop rate of all the flows running under the conditions in Table III, and calculate the success rate of the flows whose drop rate falls into the region of [0, 0.04] and [0.04, 0.08].

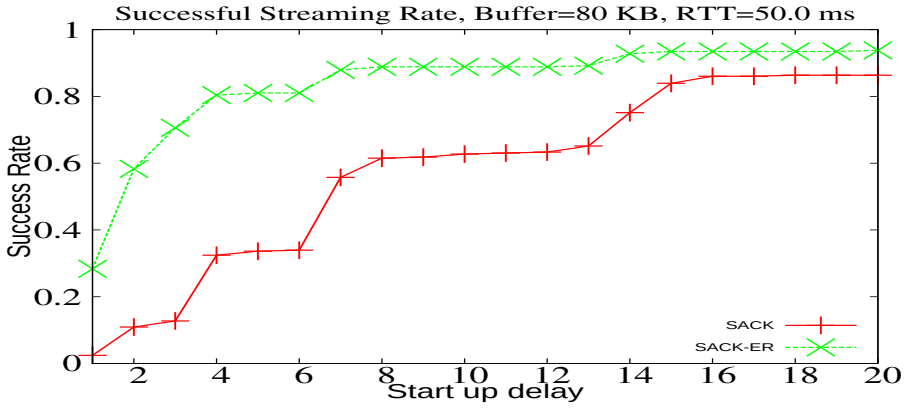
Figure 19 shows the result, from which we can see clearly that PERT's performance is not affected by drop rate as much as SACK is. Especially in the high drop rate cases, we can see that SACK's success rate never exceeds 70% when drop rate is higher than 4%. Whereas PERT is able to achieve 100% success rate if the SUD is big enough. This finding is also very similar to the ones in [11]. We speculate the reason for that is because of the reduce factor of PERT is associated with the smoothed delay estimate $srtt_{0.99}$, and therefore when drop occurred due to sudden burst of packet injection, PERT will decrease the sending rate much more gently than SACK does.

Early retransmission, on the other hand, is able to help improve both SACK and PERT's performance across both drop rate cases. This is not surprising, as we have shown in previous sections that with the same drop rate, ER is able to help prevent RTO to a great extent and therefore reducing number of late packets.

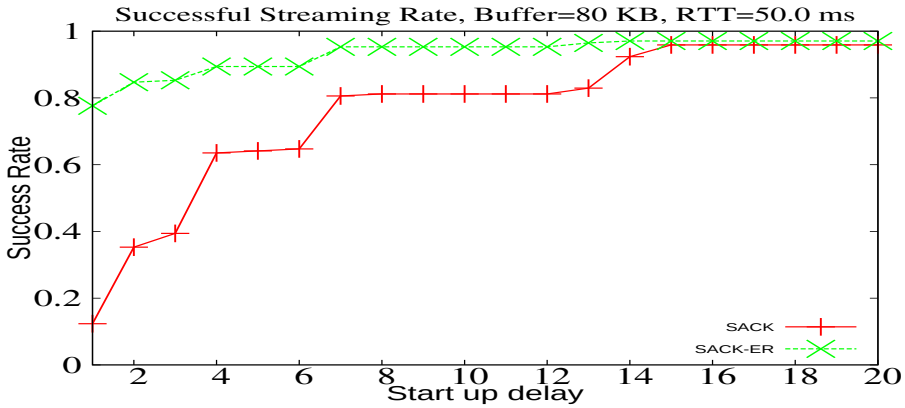
This result also assures us that ER and AQM are compatible with each other, and their effect can be combined to improve TCP's performance to a greater extent.



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

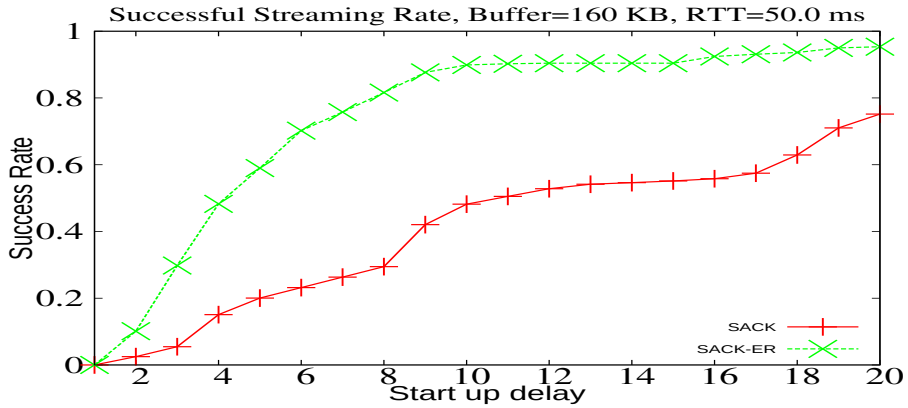


(b) $1.6 < T/\mu < 2.0$

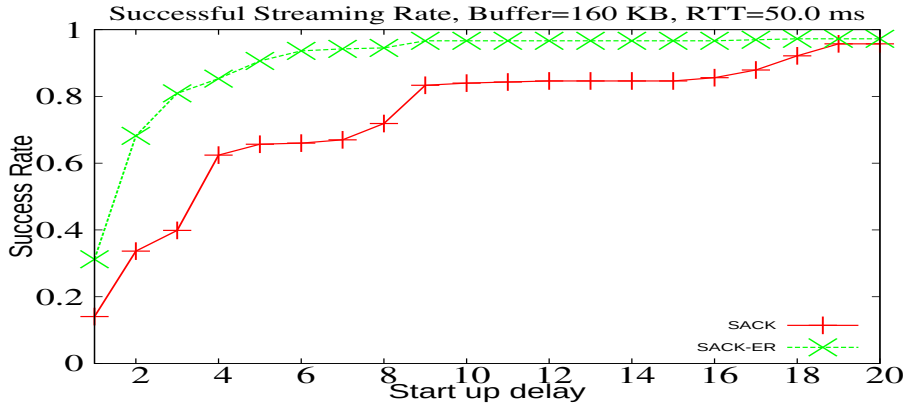


(c) $2.0 < T/\mu < 2.4$

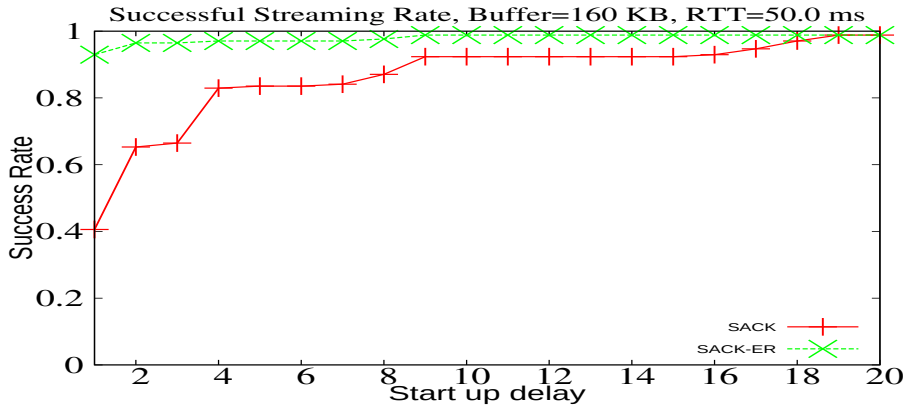
Fig. 9.: SACK vs. SACK-ER: Successful Streaming Rate, BUF=80KB



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

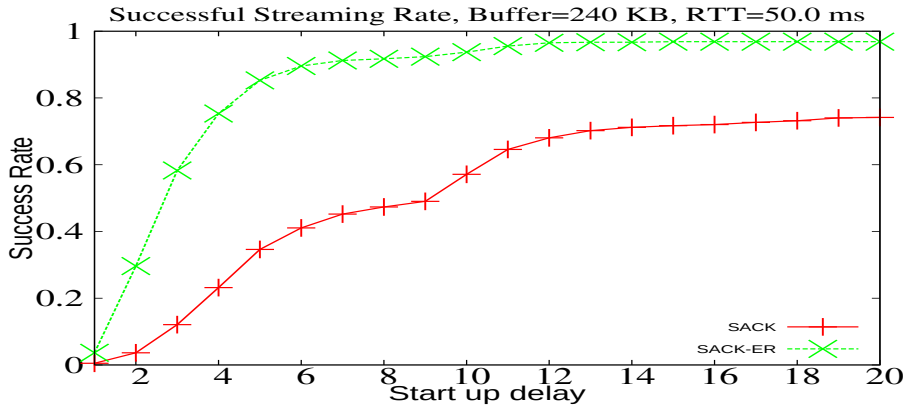


(b) $1.6 < T/\mu < 2.0$

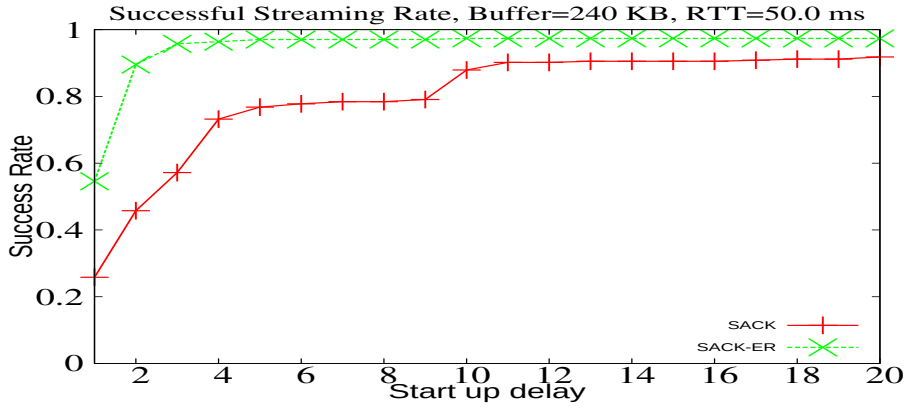


(c) $2.0 < T/\mu < 2.4$

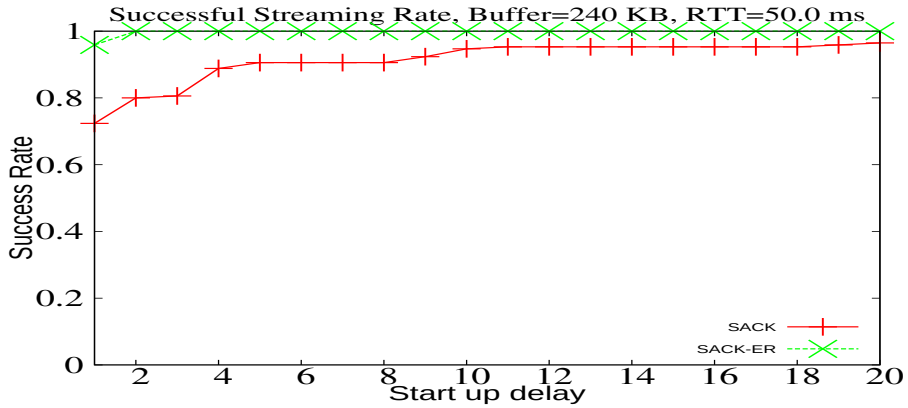
Fig. 10.: SACK vs. SACK-ER: Successful Streaming Rate, BUF=160KB



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

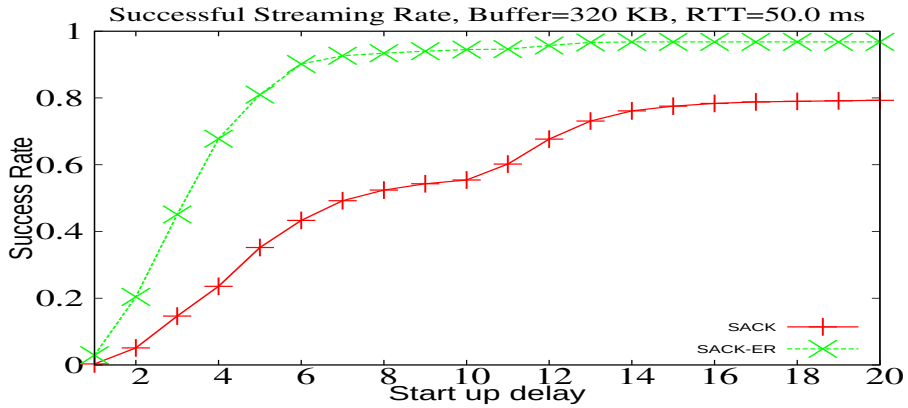


(b) $1.6 < T/\mu < 2.0$

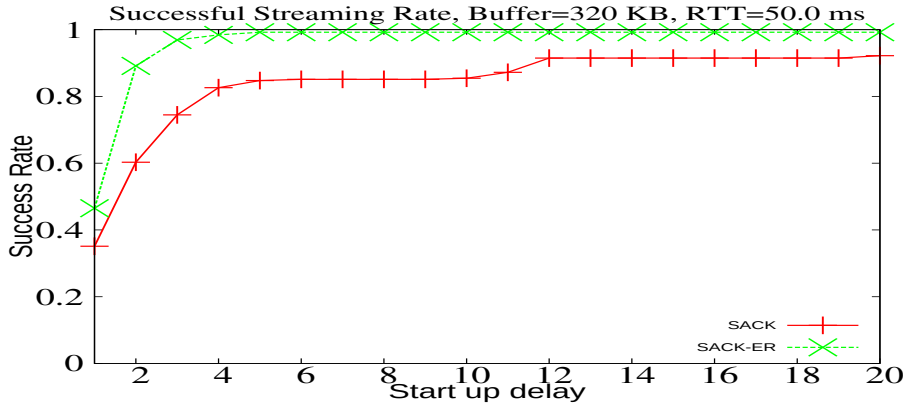


(c) $2.0 < T/\mu < 2.4$

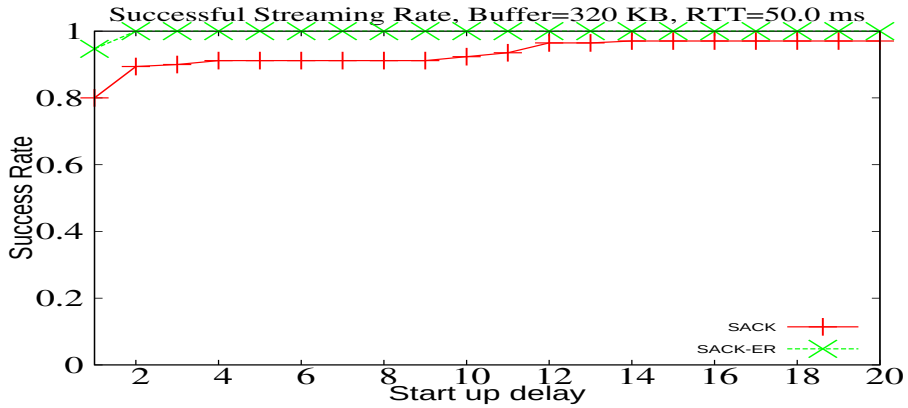
Fig. 11.: SACK vs. SACK-ER: Successful Streaming Rate, BUF=240KB



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

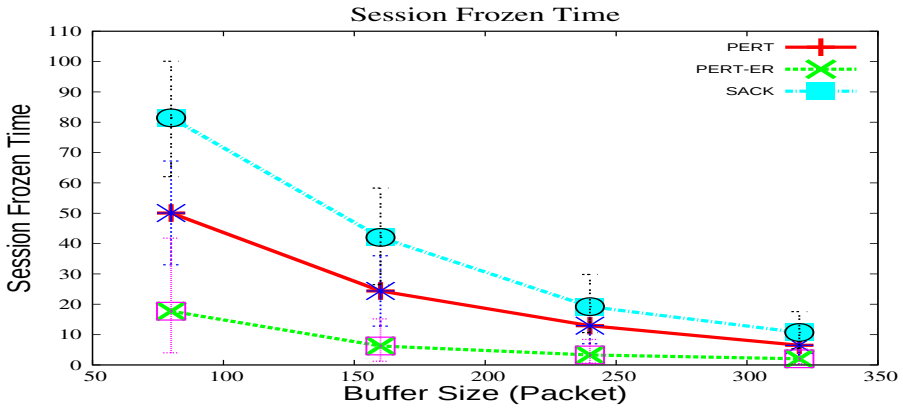


(b) $1.6 < T/\mu < 2.0$

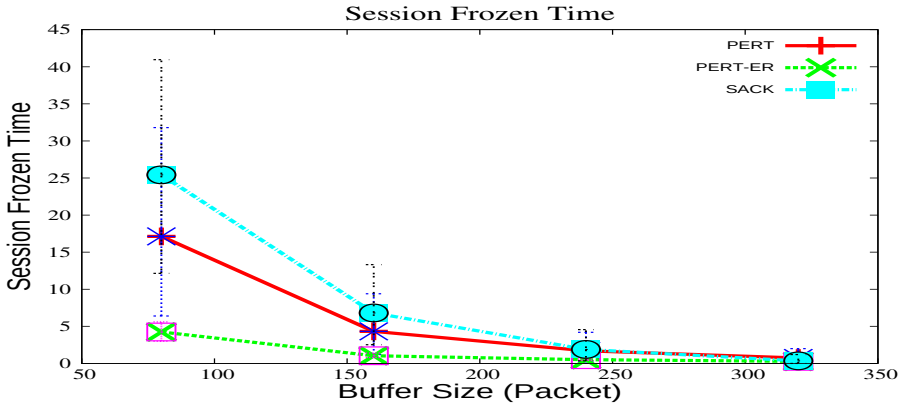


(c) $2.0 < T/\mu < 2.4$

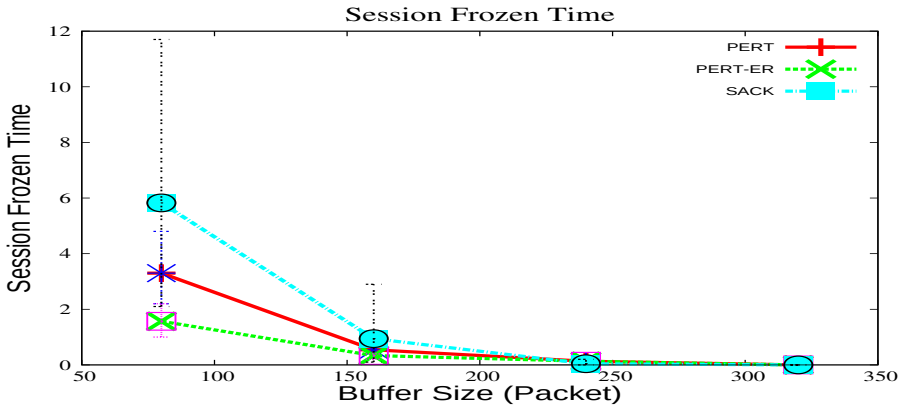
Fig. 12.: SACK vs. SACK-ER: Successful Streaming Rate, BUF=320KB



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

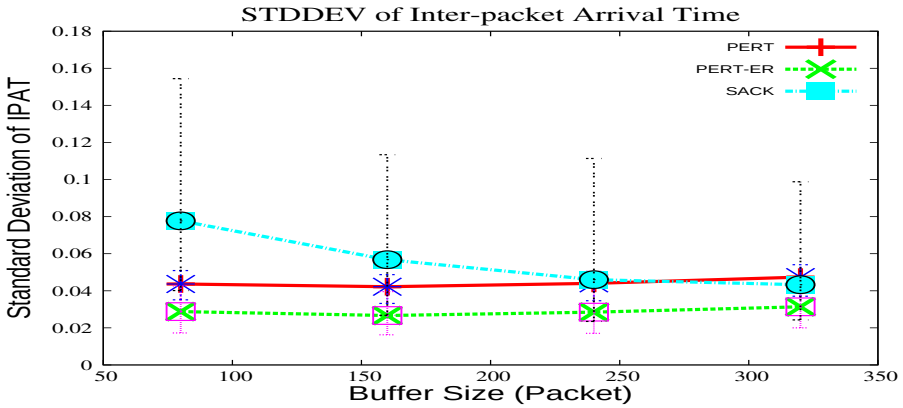


(b) $1.6 < T/\mu < 2.0$

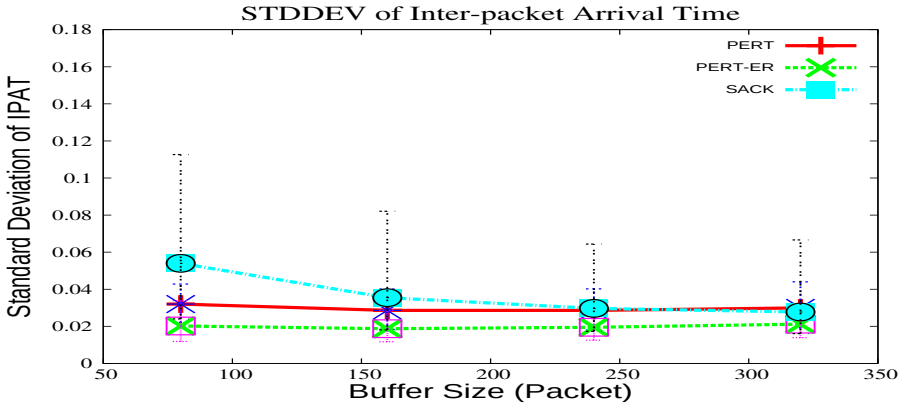


(c) $2.0 < T/\mu < 2.4$

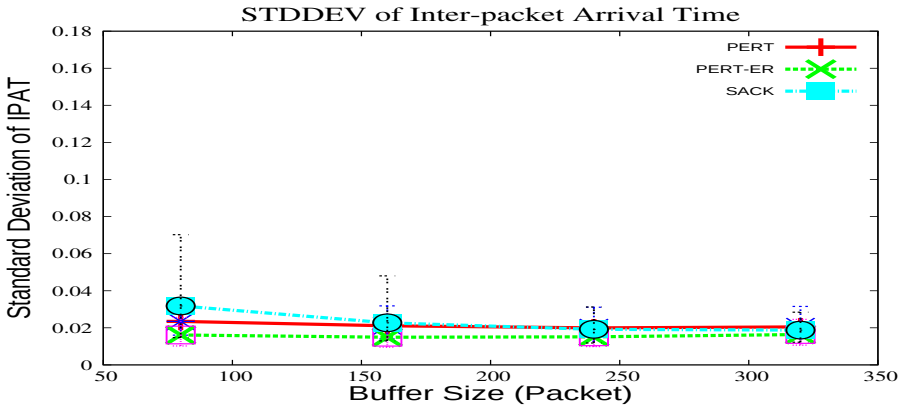
Fig. 13.: PERT vs. PERT-ER: Session Frozen Time



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

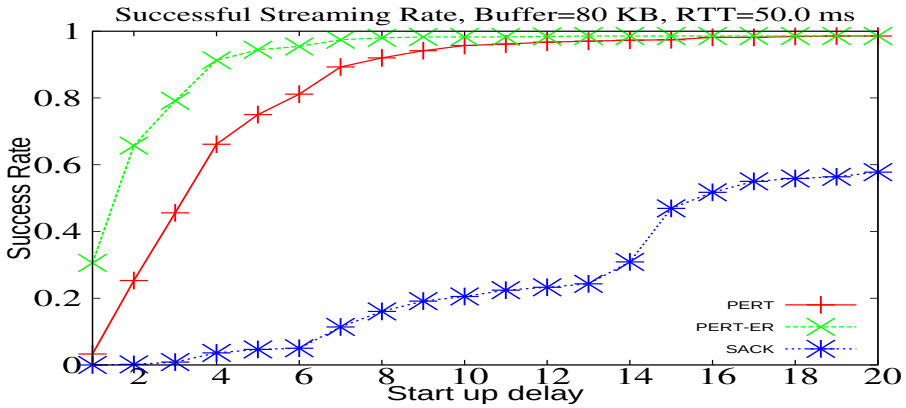


(b) $1.6 < T/\mu < 2.0$

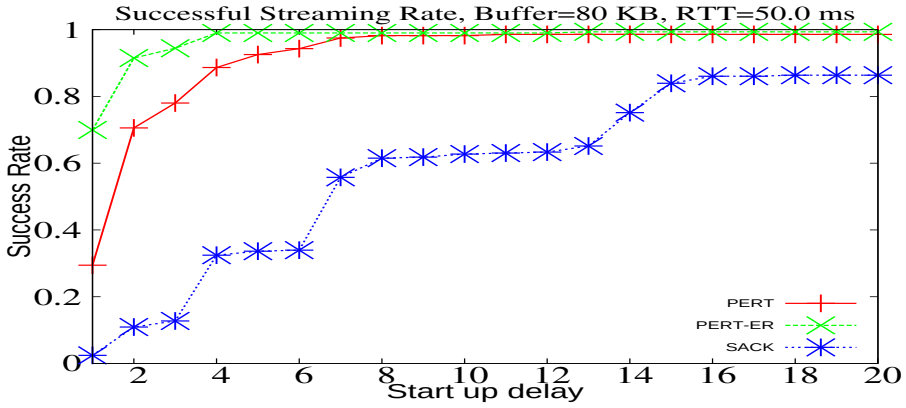


(c) $2.0 < T/\mu < 2.4$

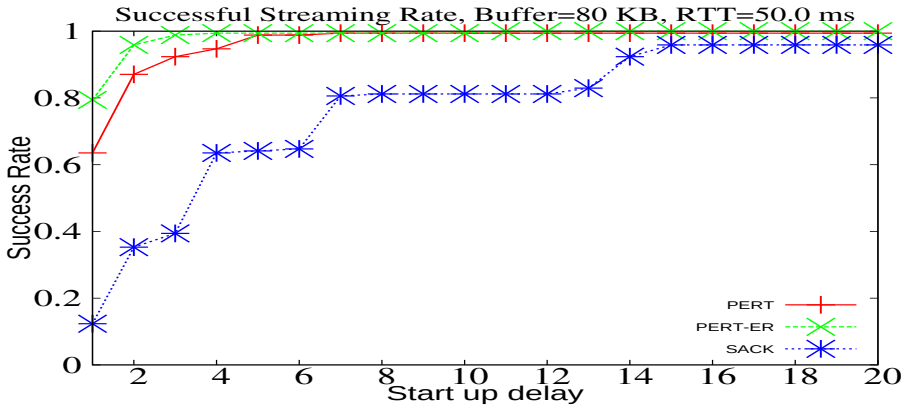
Fig. 14.: PERT vs. PERT-ER: IPAT STDDEV



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

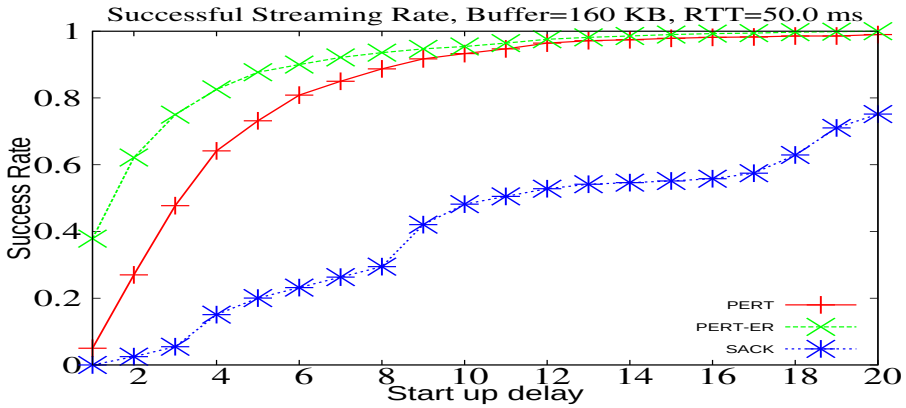


(b) $1.6 < T/\mu < 2.0$

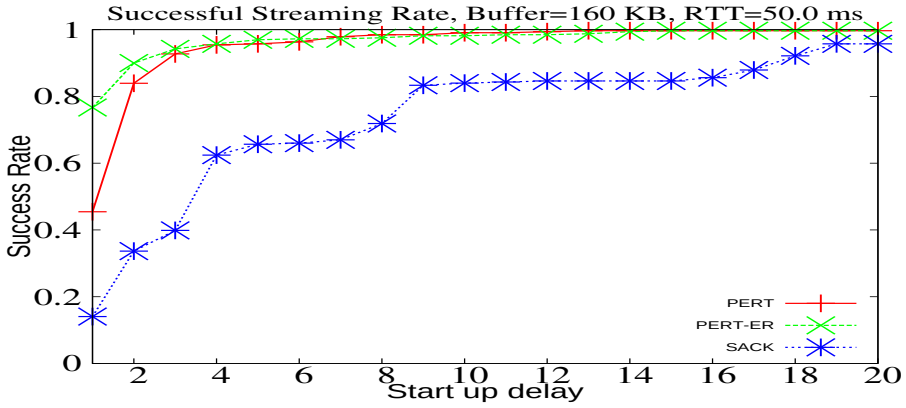


(c) $2.0 < T/\mu < 2.4$

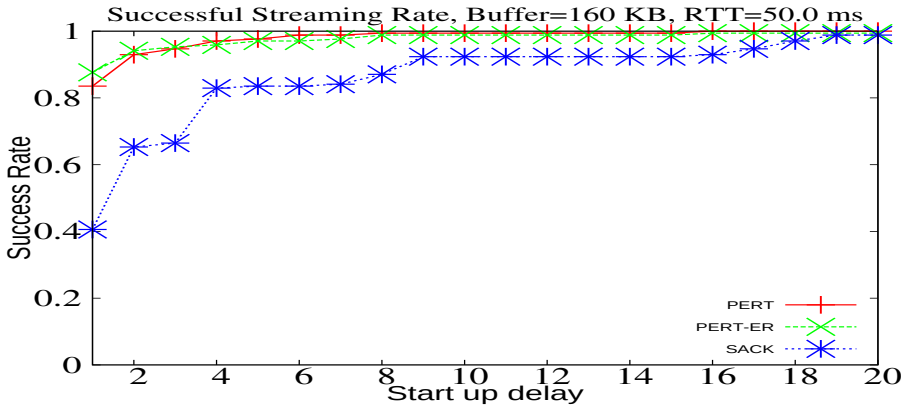
Fig. 15.: PERT vs. PERT-ER: Successful Streaming Rate, BUF=80KB



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

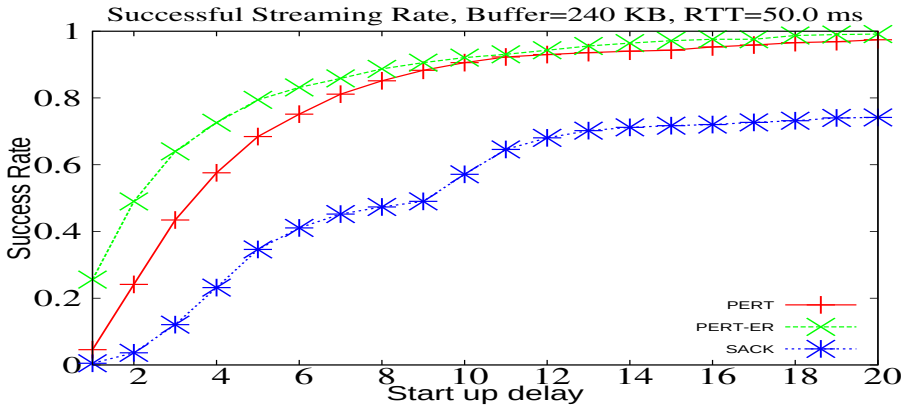


(b) $1.6 < T/\mu < 2.0$

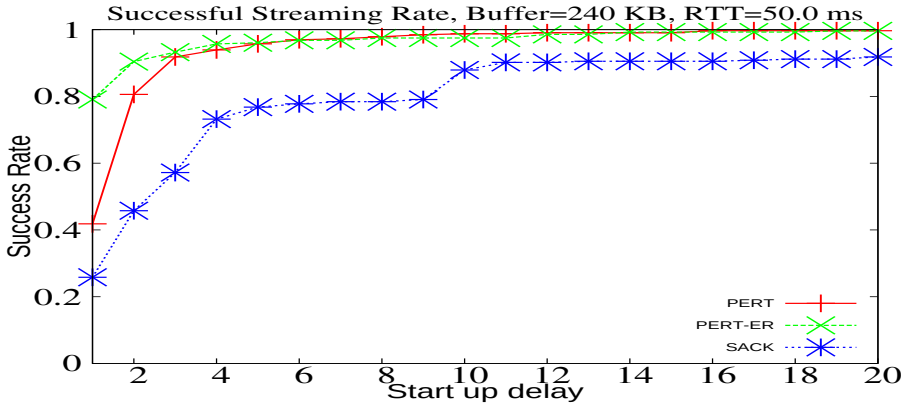


(c) $2.0 < T/\mu < 2.4$

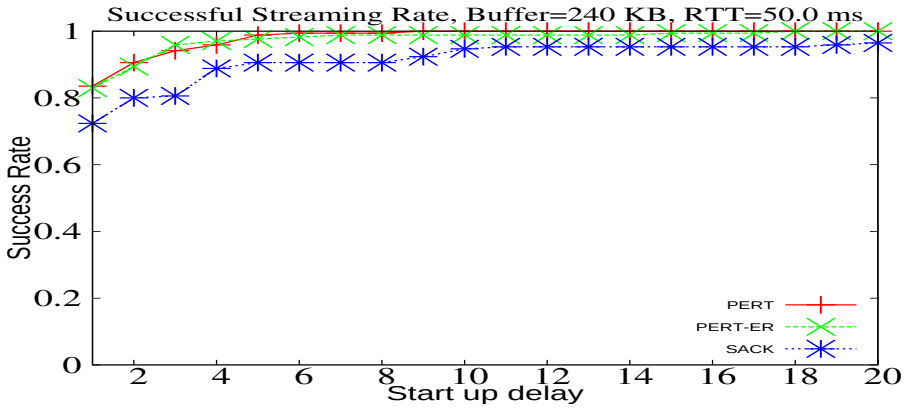
Fig. 16.: PERT vs. PERT-ER: Successful Streaming Rate, BUF=160KB



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

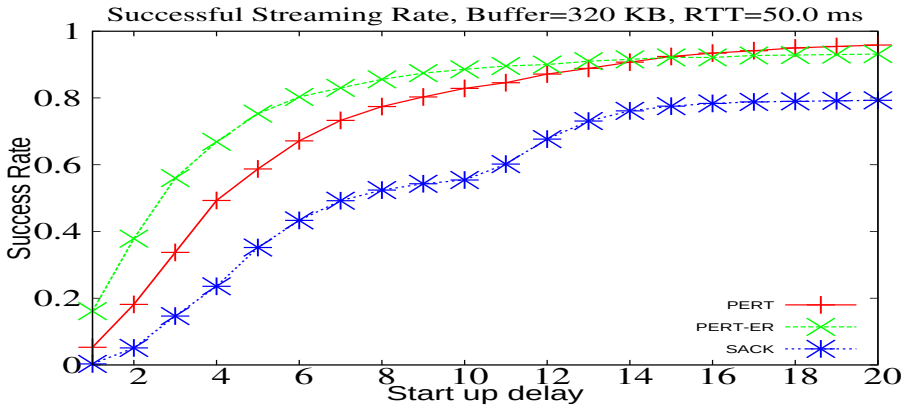


(b) $1.6 < T/\mu < 2.0$

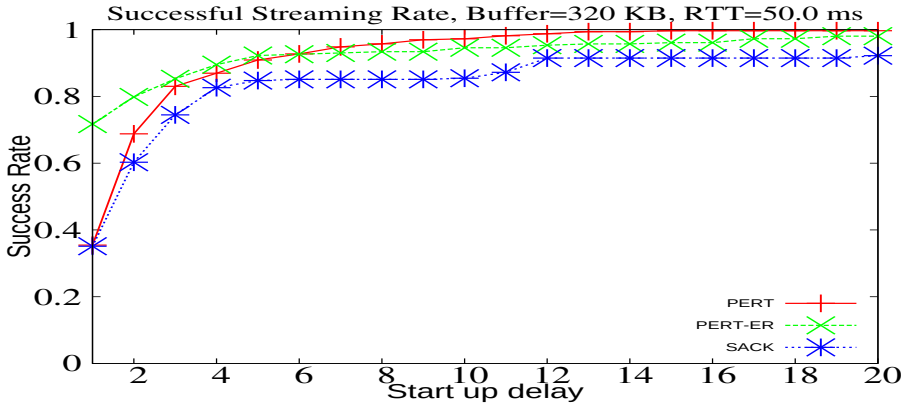


(c) $2.0 < T/\mu < 2.4$

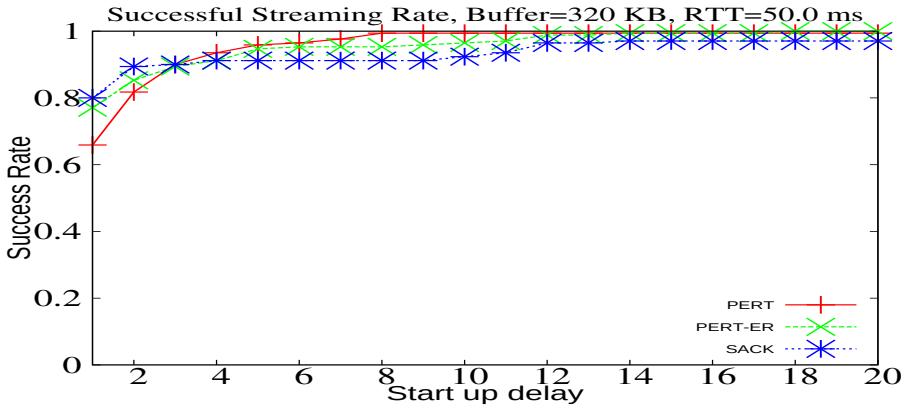
Fig. 17.: PERT vs. PERT-ER: Successful Streaming Rate, BUF=240KB



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

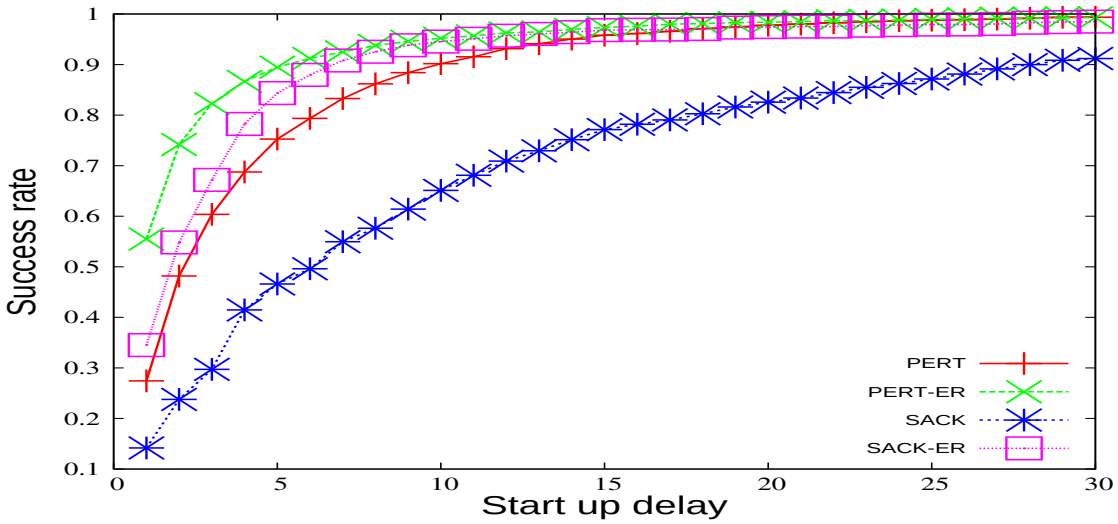


(b) $1.6 < T/\mu < 2.0$

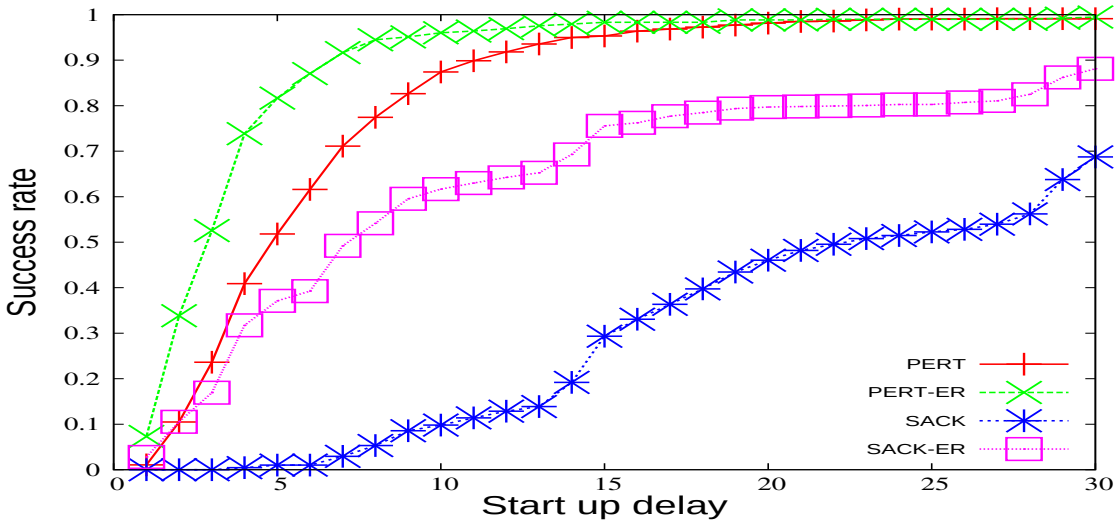


(c) $2.0 < T/\mu < 2.4$

Fig. 18.: PERT vs. PERT-ER: Successful Streaming Rate, BUF=320KB



(a) 0 < drop rate < 0.04



(b) 0.04 < drop rate < 0.08

Fig. 19.: SACK vs. SACK-ER vs. PERT vs. PERT-ER: Successful Streaming Rate With Varied Drop Rate

CHAPTER V

ER VS. FINE GRAINED RTO

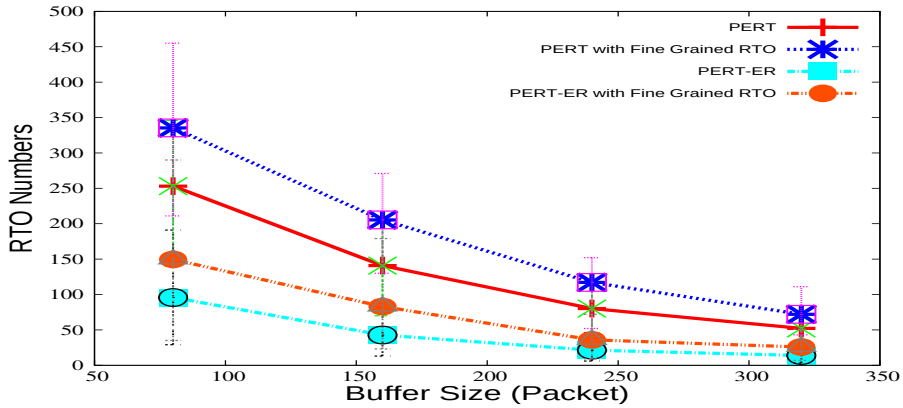
With the increase in speed of modern processors, several current OS support fine grained timer for calculating retransmission timeout value [19]. Linux, for example, now has a timer granularity of 10ms and minimum a minimum RTO value of 200ms [19]. So is the TCP implementation in *ns2*.

By having a finer granularity in calculating RTO, we essentially allow TCP to consider a packet to be dropped with a shorter waiting time, which will help TCP recover from loss faster. However, being more aggressive in this sense can also lead to a higher drop rate. In [18], TCP's performance in video streaming was shown to be improved by making both the timer granularity and minimum RTO value to 1ms.

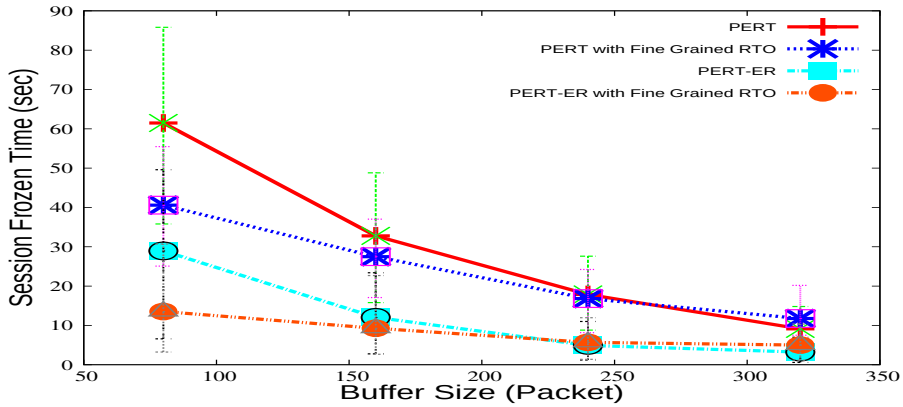
In this section, we also set the two values [1] in PERT to be 1ms and compare its performance improvement with ER. Besides that, we also want to explore if PERT-ER can be further improved by having a fine grained RTO timer.

A. Simulation Methodology

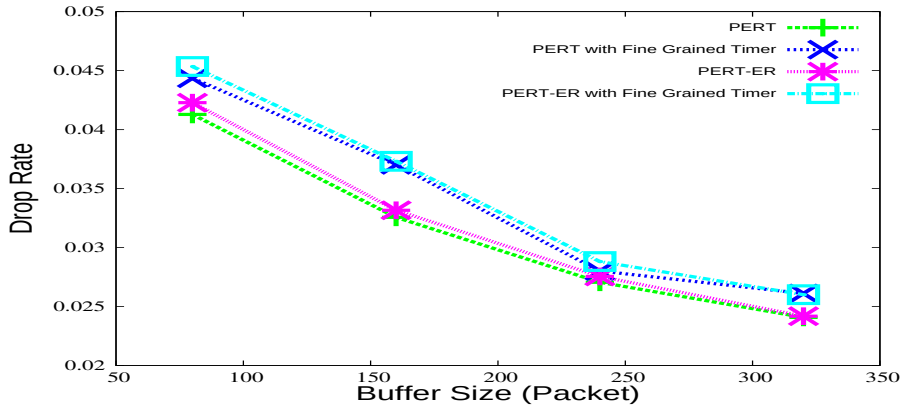
The same simulation settings in Table III are used in this chapter. Among all the test flows, we pick those with T/μ falling into the region of [1.0, 1.5] and collect total number of timeouts, SFT, drop rate and successful streaming rates of PERT, PERT-ER and those two with fine grained RTO.



(a) Number of Retransmission Timeouts



(b) Session Frozen Time



(c) Drop Rate

Fig. 20.: Fine Grained Timer vs. ER: Number of Retransmission Timeouts, SFT and Drop Rate

Table IV.: RTO Number and SFT Comparison

		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)

B. Number of Timeouts and SFT

We collect the average RTO Number and SFT of all the 1240 flows involved in the test and plot them in Figure 20a and Figure 20b respectively. Error bars show 90% confidence interval.

Figure 20a shows that when fine grained RTO is applied, PERT and PERT-ER both have a higher number of timeouts than before. This means although having a smaller RTO value will enable TCP to start recovering earlier, it also increases the possibility of a retransmitted packet also being dropped, as the bottleneck may not have enough time to drain the backlog and buffer overflow will still occur. Besides that, the extra number or RTOs induced by the fine grained RTO also decreases as

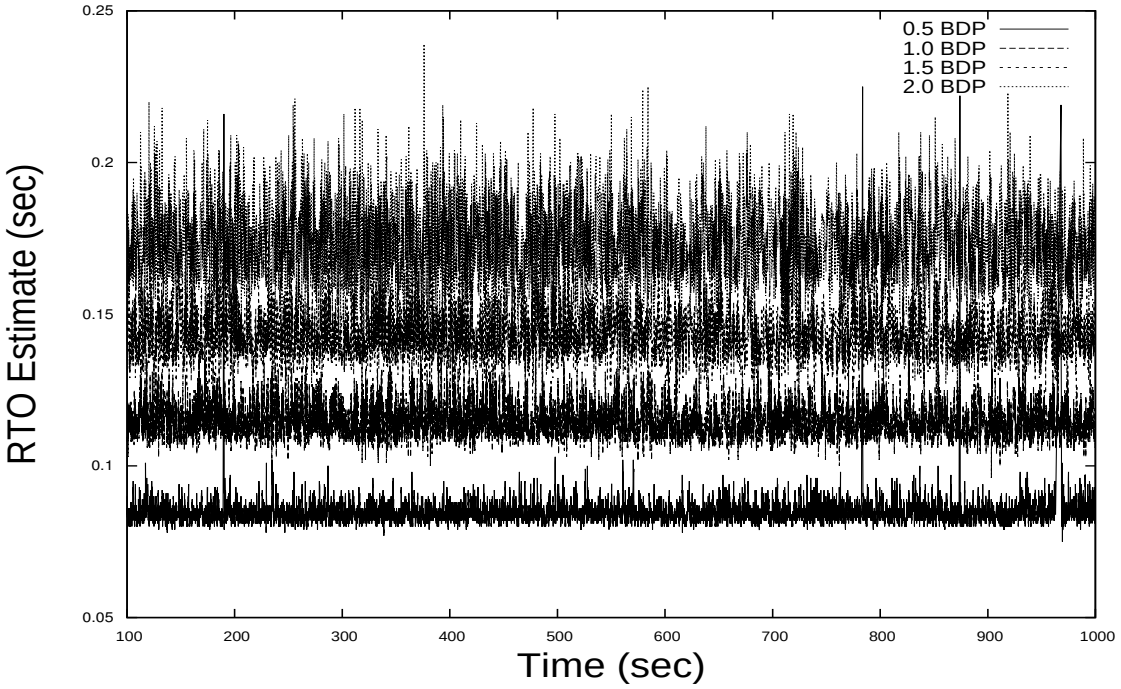
[1] *tcp_tick* and *minrto_* in *ns2*

buffer size increase, as can be seen from Table IV. We believe two reasons contribute to this. Firstly, drop rate gets lower as buffer size increases. Secondly, fine grained RTO value gets closer to the 200ms clamp.

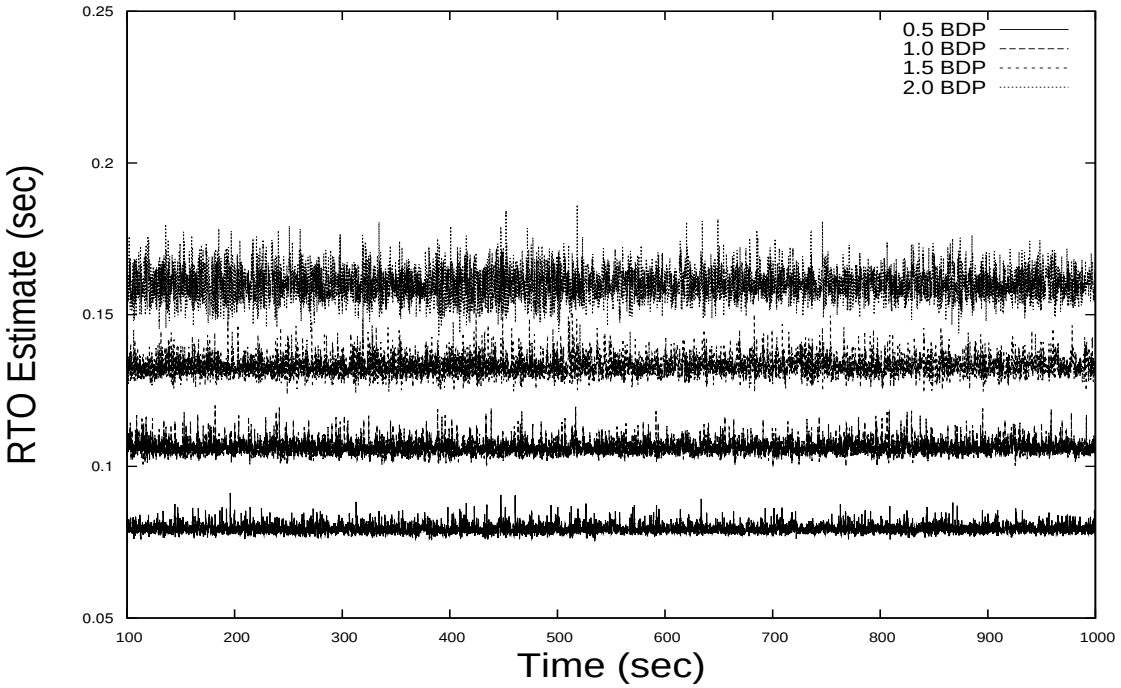
Figure 20b, shows that fine grained timer helps reduce SFT , which is the time spent in waiting for timeout to expire, especially in the case of small buffer sizes. Table IV shows that when buffer size is 80KB (0.5 BDP) and 160 KB (1.0 BDP), fine grained RTO helps reduce SFT of PERT by 21 seconds and 5 seconds respectively. In the case of PERT-ER, it helps reduce SFT by 15 seconds and 3 seconds. The is because when small buffer is applied, the fine grained RTO value computed is much smaller than the default 200ms, which can be seen in Figure 21a, where fine grained RTO is shown to be around 80ms and 120ms when buffer size is 0.5 BDP and 1.0 BDP . Therefore, even though total number of timeouts are increased, SFT is still reduced.

When compared the improvement fine grained RTO brings to PERT and PERT-ER, we find PERT benefits more from fine grained RTO . Table IV shows PERT's SFT is reduced by 21 sec, 5 sec and 2 sec when buffer size is 80KB, 160KB and 240KB, while PERT-ER has a decrease of 15 sec, 3 sec and 0 sec. We believe this is because PERT suffers from a bigger RTO number than PERT-ER, and therefore more time can be saved from employing the fine grained RTO .

We also notice that ER's ability to reduce SFT is better than that of fine grained timer, which can be proven by the fact that the SFT curve of PERT-ER stays below that of PERT with fine grained RTO across all buffer size cases.



(a) Fine Grained RTO Value



(b) Early Retransmission RTO Estimate

Fig. 21.: Fine Grained Timer vs. ER: F-RTT and ER RTO Estimate

C. Successful Streaming Rate

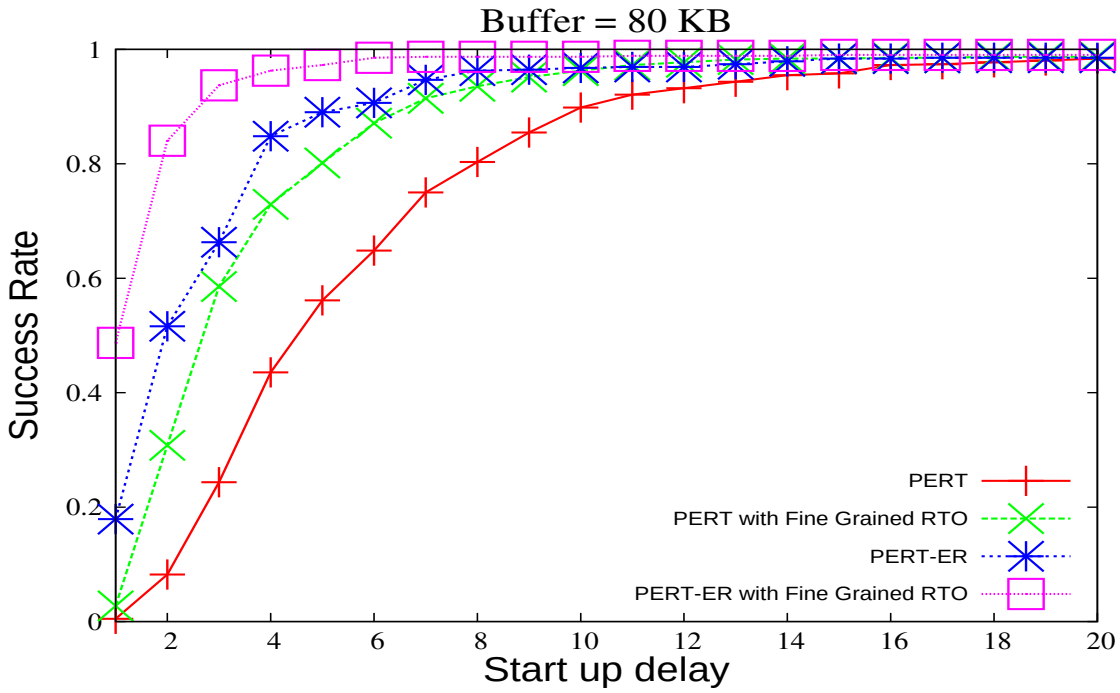
Figure 22 and Figure 23 show the successful streaming rate of all buffer size cases. Table V shows the successful streaming rate comparison when SUD is 5 seconds.

First of all, it's quite obvious that the help of fine grained RTO decreases as buffer size increases, which is clearly demonstrated in Table V. Note that when buffer size is 320KB(2.0 BDP), fine grained RTO will cause PERT-ER's success rate to decrease by 2%. This is in accordance with the slight increase in SFT in the big buffer case, which is discussed in the previous section.

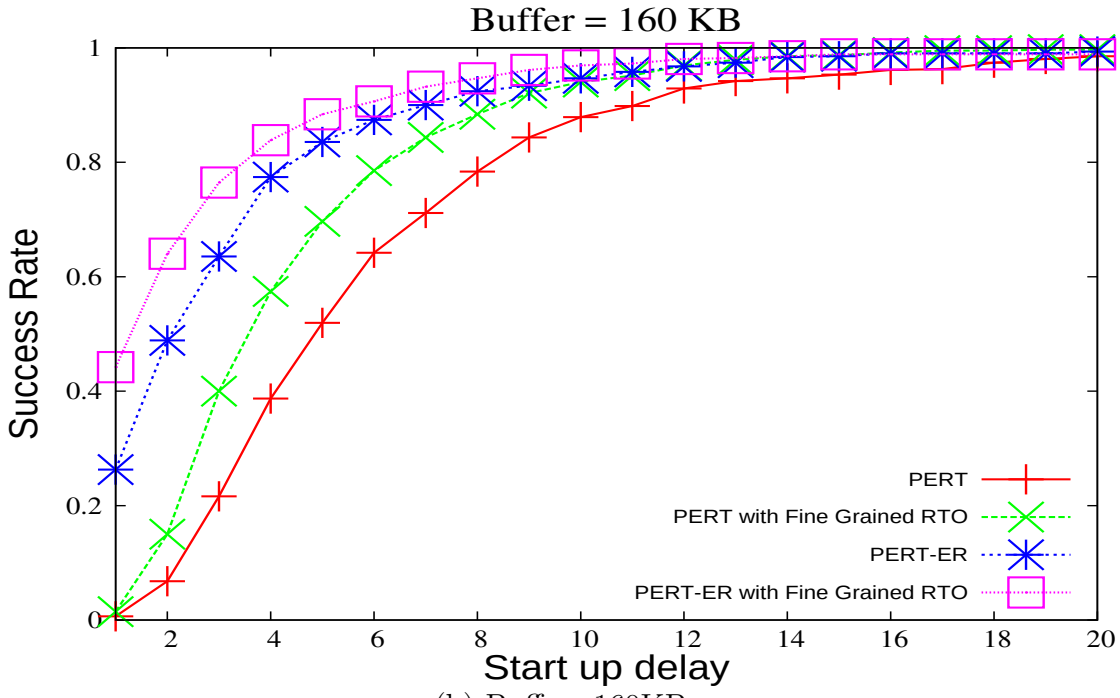
Table V.: Fine Grained Timer vs. ER: Fraction of Successful Streaming

	PERT	PERT with fine with grained RTO	PERT-ER	PERT-ER with fine with grained RTO
80KB(0.5 BDP)	56%	80% (24%)	89%	97% (8%)
160KB(1.0 BDP)	52%	70% (18%)	84%	90% (6%)
240KB(1.5 BDP)	42%	50% (8%)	64%	68% (4%)
320KB(2.0 BDP)	32%	33% (1%)	47%	45% (-2%)

Secondly, we also notice fine grained RTO help PERT more than it helps PERT-ER, which can be seen in both Figure 22, Figure 23 and Table V. This corresponds our findings in SFT , where fine grained RTO is shown to have a bigger effect on PERT than on PERT-ER. PERT-ER is also shown to perform better than PERT with fine grained RTO across all buffer cases.

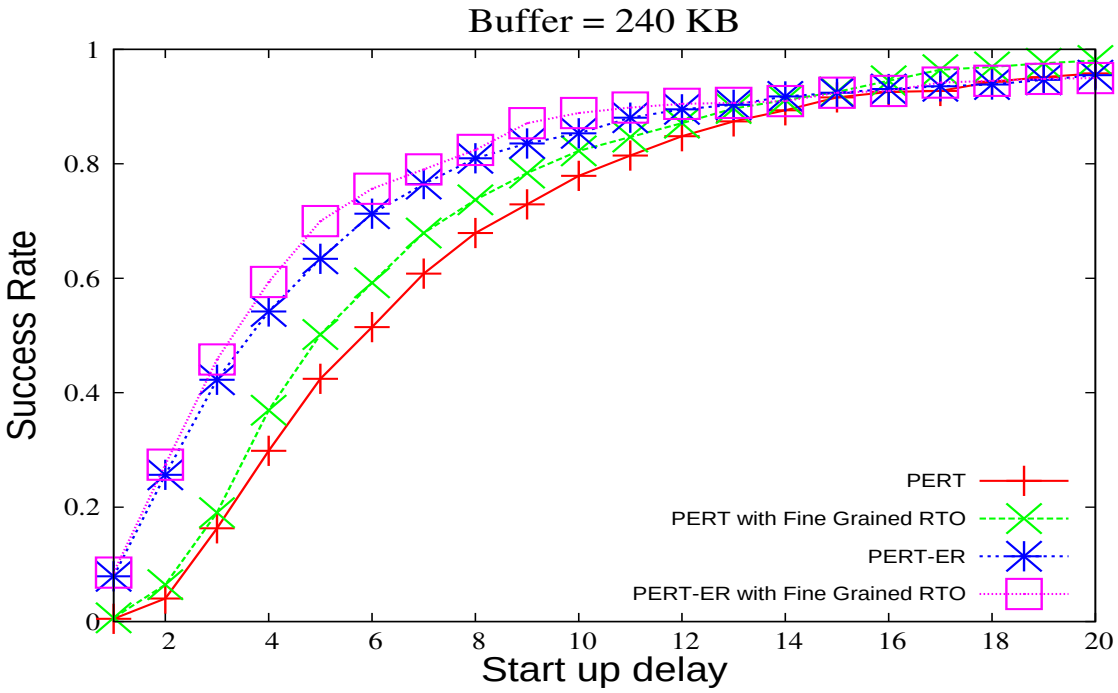


(a) Buffer=80KB

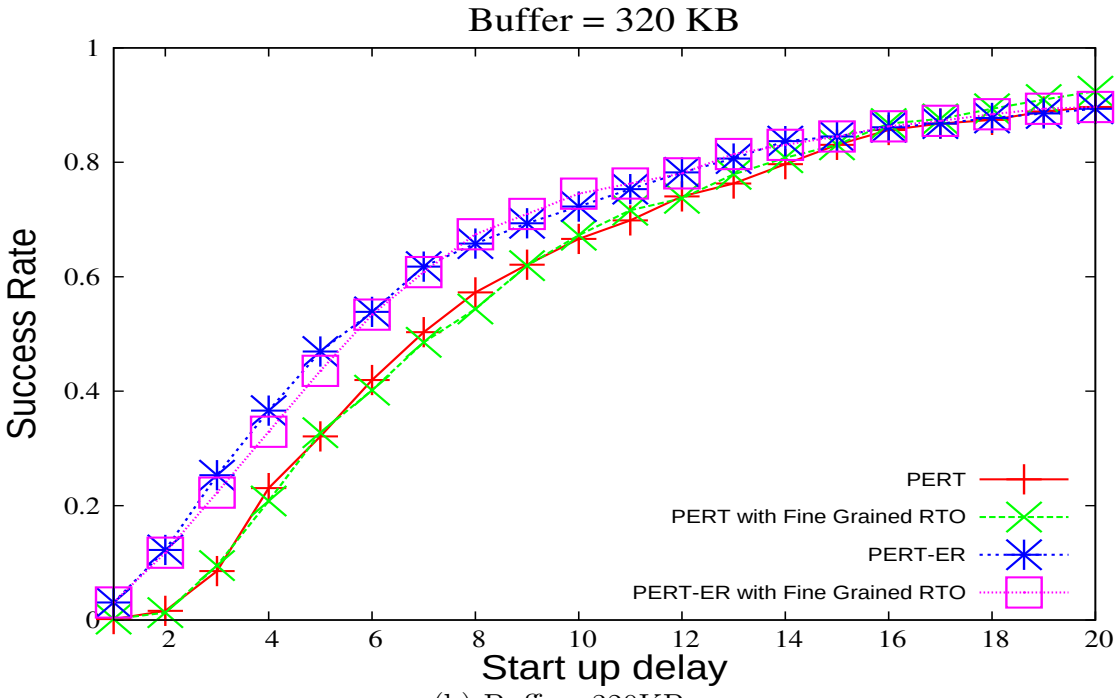


(b) Buffer=160KB

Fig. 22.: Fine Grained Timer vs. ER: Fraction of Successful Streaming, BUF=80KB, 160KB



(a) Buffer=240KB



(b) Buffer=320KB

Fig. 23.: Fine Grained Timer vs. ER: Fraction of Successful Streaming, BUF=240KB, 320KB

CHAPTER VI

CONCLUSION

In this paper, we show that early retransmission can help TCP-SACK in reducing the latency caused by RTOs, which in turn increase the rate of successful streaming in a variety of network settings. We also demonstrate that the same improvement can be produced by integrating the ER scheme into a delay based protocol, namely PERT. Our test results show a clear trend of improvement from SACK to PERT, and from PERT to PERT-ER. This means AQM and early retransmission are orthogonal to each other and the combination of the two can help improve TCP's capability in delivering time sensitive media to a greater extent. Moreover, compared to the help brought by removing the lower bound on TCP's RTO estimate and increasing its granularity, ER's improvement is more consistent and obvious. We also show the effect of fine grained RTO and ER can be combined to produce a even better performance.

REFERENCES

[1] J. Boyce and J. Gaglianello, "Packet loss effects on mpeg video sent over the public internet," in *Proc. of ACM Multimedia*, Sept. 1998, pp. 181 – 190.

[2] S. Kunwadee, M. Bruce, and Z. Hui, "An analysis of live streaming workloads on the internet," in *Proc. of the 4th ACM SIGCOMM Conference on Internet Measurement*, Oct. 2004, pp. 41–54.

[3] M. Li, M. Claypool, R. Kinicki, and J. Nichols, "Characteristics of streaming media stored on the web," *ACM Transactions on Internet Technology*, vol. 5, pp. 601–626, 2005.

[4] M. Claypool J. Chung and R. Kinichi, "Mtp a streaming friendly transport protocol," Technical Report WPI-CS-TR-05-10, Worcester Polytechnic Institute, 2005.

[5] L. Zhang, "Why tcp timers don't work well," *SIGCOMM Comput. Commun. Rev.*, vol. 16, no. 3, pp. 397–405, 1986.

[6] V. Jacobson, "Congestion avoidance and control," *SIGCOMM Comput. Commun. Rev.*, vol. 18, pp. 314–329, 1988.

[7] 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, 1996.

[8] U. Hengartner, J. Bolliger, and T. Gross, "Tcp vegas revisited," in *Proc. of INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, Mar. 2000, vol. 3, pp. 1546–1555.

[9] S. Bhandarkar, A.L.N. Reddy, Y. Zhang, and D. Loguinov, "Emulating aqm from end hosts," in *Proc. of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, Aug. 2007, pp. 349–360.

[10] K. Kotla and A.L.N. Reddy, "Making a delay-based protocol adaptive to heterogeneous environments," in *Proc. of 16th International Workshop on Quality of Service*, Jun. 2008, pp. 100–109.

[11] 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, 2000.

[12] 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, 2004.

[13] S. Boyden, A. Mahanti, and C. Williamson, "Tcp vegas performance with streaming media," in *Proc. of IEEE International Performance, Computing, and Communications Conference*, Apr. 2007, pp. 35–44.

[14] B. Qian and A.L.N Reddy, "Measurement and performance study of pert for on-demand video streaming," presented at the 8th International Workshop on Protocols for Future, Large-Scale & Diverse Network Transports, Nov. 2010.

[15] A. Goel, C. Krasic, and J. Walpole, "Low-latency adaptive streaming over tcp," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 4, pp. 20:1–20, 2008.

[16] D. Loguinov and H. Radha, "End-to-end internet video traffic dynamics: statistical study and analysis," in *Proc. of INFOCOM 2002. Twenty-First Annual*

Joint Conference of the IEEE Computer and Communications Societies, Jun. 2002, vol. 2, pp. 723 – 732.

[17] L. Zhang, L. Zheng, and K.S. Ngee, "Effect of delay and delay jitter on voice/video over ip," *Computer Communications*, vol. 25, no. 9, pp. 863 – 873, 2002.

[18] K. Garg, "Evaluation of tcp & pert for video streaming transport," Technical Report TAMU-ECE-2010-01, Texas A&M University, 2010.

[19] S. Pasi and K. Alexey, "Congestion control in linux tcp," in *Proc. of the FREENIX Track: 2002 USENIX Annual Technical Conference*, Oct. 2002, pp. 49–62.

VITA

Zhiyuan Yin received his Bachelor of Engineering degree in electrical and information engineering from Shanghai Jiao Tong University, Shanghai, China, in 2008. He joined the Electrical Engineering Department of Texas A&M University in August 2008, and received his Master of Science degree in May 2011. His research interests are in computer networks.

Mr. Yin can be reached at the Department of Electrical and Computer Engineering, Texas A&M University, 331E Wisenbaker, College Station, TX 77843- 3259. His email is zyin3@tamu.edu.