Performance of Early Retransmission Scheme In Streaming Media

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

***Abstract*—In this paper, we propose a new variant of TCP with an early retransmission scheme as an enhancement to make it more suitable for streaming media. We call this new protocol TCP-ER. We performed extensive *ns2* simulations to show that: 1) The early retransmission scheme can considerably reduce the number of retransmission timeouts and packet delay jitter in a variety of network environments. 2) TCP-ER has a higher throughput in severely congested network conditions whereas stays relatively fair with typical TCP implementations(specifically TCP-SACK) as congestion gets alleviated. 3) Under same network conditions, constrained streaming over TCP-ER has a considerably lower number of late packets than its normal TCP counterpart.**

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 which emphasize more on timeliness than reliability. Because of that, many multimedia applications turn to UDP as their underlying transport protocol. However, the majority portions of video on demand and live broadcast applications still favour TCP over UDP, due to UDP's unresponsiveness to network conditions and problems with firewalls and NATs[1].

TCP's poor performance of delivering real-time media is due to the following reasons: 1) TCP's emphasis on reliable in-order delivery causes frame jitter that interrupt media play-out[2]. 2) TCP's coarse-grained Retransmission Timeout (**RTO**) and its back-off mechanism is detrimental to any real-time based application[3].

There exist several different variants of TCP, which are distinguished by their congestion control and loss recovery mechanism. The most important of those are TCP Tahoe, Reno, NewReno, Selective Acknowledgement (SACK), and Vegas. Among those variants, SACK has been shown to have the best error recovery scheme [4] and Vegas, being the only variant that doesn't use an Additive Increase Multiplicative Decrease (**AIMD**) Congestion Avoidance (**CA**) scheme, has been touted to be a suitable protocol for streaming media. In [5], Vegas was shown to be able to achieve zero late packets when throughput is 1.5 times the playback rate with 10 seconds of pre-buffering, whereas for Reno to achieve zero late packets, throughput needs to be at least double the playback rate[6]. As is shown in [7], the good performance of Vegas is due to its early retransmission and recovery algorithm. What's more, [7] also shows that Vegas' innovative congestion avoidance mechanism has minimal or even negative impact on its throughput.

Given all the above, one very natural question will be how much performance boost we can get if we integrate Vegas' early retransmission scheme into other TCP variants? We address this question with a special emphasis on live streaming media delivery. This paper shows that by integrating this feature into typical TCP implementation (specifically, TCP-SACK), we can **a)** reduce Retransmission Timeouts to a great extent, **b)** keep packet delay jitter at a significantly lower level, **c)** achieve zero late packet under a tighter constraint when delivering real-time media.

The rest of the paper is organized as follows: Section II presents relevant background information for this work. Section III discusses some issues regarding the implementation of TCP-ER. Section IV presents the results of the simulation. Conclusions and future directions are presented in Section V.

II. 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[8]. 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.

A. 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.

Hussein Alnuweiri is with the Texas A&M University at Qatar.

This research was supported by the Qatar National Research Fund under the National Priorities Research Program, Project number NPRP 26-6-7-10, and by a research grant from Qtel.

	TCP	Vegas
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.

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

TABLE I: Difference in SS and CA scheme between Vegas and Other TCP Variants

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 [4], enables NewReno and SACK to recover from multiple packet drops in one CWND without invoking Retransmission Timeout. For the improvements brought by the introduction of Partial ACK and its importance in avoiding timeouts, please refer to [4], where insightful simulation-based comparison is made.

For the extra performance enhancement brought about by SACK, Table III and Table IV show 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 do that. We regard this feature as very desirable as the difference of one RTT (normally ranges from 50ms to 100ms) means a lot in real-time media delivery.

	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.

TABLE II: Difference in Fast Retransmission/Fast Recovery between TCP Variants

Round	ACK	Seq	Wnd	State	Note
1	6		6	CA	Pkt 9
	6		6	CA	Pkt 10
	6	7	6/2+3=6	FR	Pkt 11
	6	13	7	FR	Pkt 12
2	7	8,14	7	FR	Pkt 7
	7	15	8	FR	Pkt 13
3	14	16∼18	3	CA	Pkt 8

TABLE III: Example Recovery of TCP Newreno From Two Packet Drop: Wnd = CWND + number of DUPACKS. Packet 7 and 8 are dropped in the previous round. **Sec** column lists the new/retransmitted packet generated by the corresponding ACK. **Note** column lists the transmitted/retransmitted packet that generates the current ACK.

Round	ACK	Seq	Wnd	State	Note
1	6		6	CA	Pkt 9
	6		6	CA	Pkt 10
	6	7	3	FR	Pkt 11,pipe=3
	6	8	3	FR	Pkt 12,pipe=2
	7	13,14	3	FR	Pkt 7,pipe=1
2	12	15	3	CA	Pkt 8

TABLE IV: Example Recovery of TCP Sack In the Same Condition as III.

B. TCP-Vegas

1) Novel Congestion Avoidance Scheme: 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.

2) 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 [7], 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 $\frac{1}{4}$, 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 [7], 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 the loss recovery scheme in TCP-ER uses a different approach to increase throughput in a congested environment while staying relatively fair to TCP when congestion is not severe.

When viewing from a real-time streaming perspective, Vegas' early retransmission and recovery schemes alleviate TCP's in-order delivery and reliable transmission constraint in that they speed up the loss recovery process by avoiding retransmission timeouts, and thereby reducing delay jitter, which we will show in the later sections.

III. Implementing TCP-ER

As discussed above, in addition to the Early Retransmission feature, Vegas' loss recovery scheme also contributes to its good performance, especially in the case where bottleneck bandwidth cannot match traffic load, which results in a small CWND size. Therefore, in addition to implementing the early retransmission feature, we also modify the Loss Recovery scheme of NewReno and SACK. To achieve that, the following algorithm is used in TCP-ER:

Algorithm 1 CWND Inflation Algorithm

```
When Early/Fast Retransmission Is Triggered
if Early Retransmission then
    if cwnd < 4 then
        cwnd = 2
        Inflate cwnd by 2 packets
    else
        cwnd = cwnd/2 + Number of DUPACKS received.
    end if
else if Fast Retransmission then
    cwnd = cwnd/2 + 3
end if
```

As can be seen from Algorithm 1, in our implementation loss recovery incurred by normal fast retransmission (i.e. by 3rd DUPACK) is the same as usual. In the early retransmission case, we use 4 as a threshold value according to which we decide what operation should be performed on CWND. The reason for that is because when congestion window size is less than 4, even in a single drop case, not enough DUPACKs will be generated to trigger normal fast retransmission. Therefore, we treat the case when CWND < 4 as a signal indicating that RTO is very likely to happen, even in the case of early retransmission, as there might be packet drops during Fast Recovery.

In the case where CWND size is above the safety line, we follow the normal fast retransmission case: Cut the CWND by half, and inflate it by the number of DUPACKS received. However, in the case when CWND is below the threshold value, we set it to **2**, and then inflate it by **2** packets[1]. Following what Vegas does, we lower-bound CWND by 2 to avoid the case where no new packets will be clocked out during fast recovery, and we inflate CWND by the same number to make sure that when the ACK that pulls the sender out of FR (Recover ACK) arrived, CWND will be at least one packet higher than the number of outstanding packets. Therefore, in the case where there is even loss in Fast Recovery, at least one new packet can be generated, and RTO may still be avoided[2].

It's not hard to find out that Algorithm 1 will not make too much of a difference when CWND size is big, as it is almost the same as the fast recovery of normal SACK and NewReno[3]. However, the modification will ensure packets keep flowing during loss recovery when CWND is small. Therefore, we do expect it to make a difference in the number of timeouts when bottleneck bandwidth cannot match the traffic load, which is verified by the results shown in the later sections.

IV. Experimental Evaluation

We performed extensive *ns2* simulation to evaluate TCP-ER's performance and to compare it with its normal counterparts. Due to space constraint, we will mainly focus on the performance improvement by incorporating Early Retransmission into TCP-SACK (SACK-ER), as SACK has the best drop recovery scheme among all the TCP variants.

A. Performance Metrics

When evaluating the transport protocol with an emphasis on timely delivery, four performance metrics were measured:

- **Session Frozen Time (SFT)**
  We define SFT to be the accumulated duration of the time when not a single new ACK arrived at the sender. 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 didn't simply measure the total number of RTOs due to the exponential back-off of Retransmission Timer. 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.
- **Standard Deviation of Inter-Packet Arrival Time (IPAT)**
  Next, we measured the Standard Deviation of IPAT because it reflects the jitter of packet arrival, which has been

[1]The reason why we separate it into two steps here is to help explain the algorithm.

[2]As the DUPACK generated by this new packet will trigger another Early Retransmission.

[3]Except for the Early Retransmission. But since enough DUPACKs will be generated to trigger FR when CWND is big, ER will only make a difference when sudden huge burst of packets get injected to the network, e.g. Slow Start or HTTP Session Burst.

well recognized to have big impact on human perception of multimedia streaming[9].

- **TCP Fairness**
  We also put SACK-ER and Vegas in a SACK environment to evaluate how well they coexist with SACK flows.
- **Number of Late Packets**
  Last but not least, number of late packets was measured when applying different play-out rates. We are essentially simulating live streaming, where server generates media content in real time, and the transport agent is only able to deliver packets that have already been generated. Following the naming convention in [6], we call this **Constrained Streaming**, as opposed to **Unconstrained Streaming**, where the transport agent can deliver as many packets as allowed by the network bandwidth. As [10] shows that a late percentage of as little as 3% can affect up to 30% of the frames in MPEG-1 video, we focused on the conditions when 0 late packets are achieved.

B. Session Frozen Time

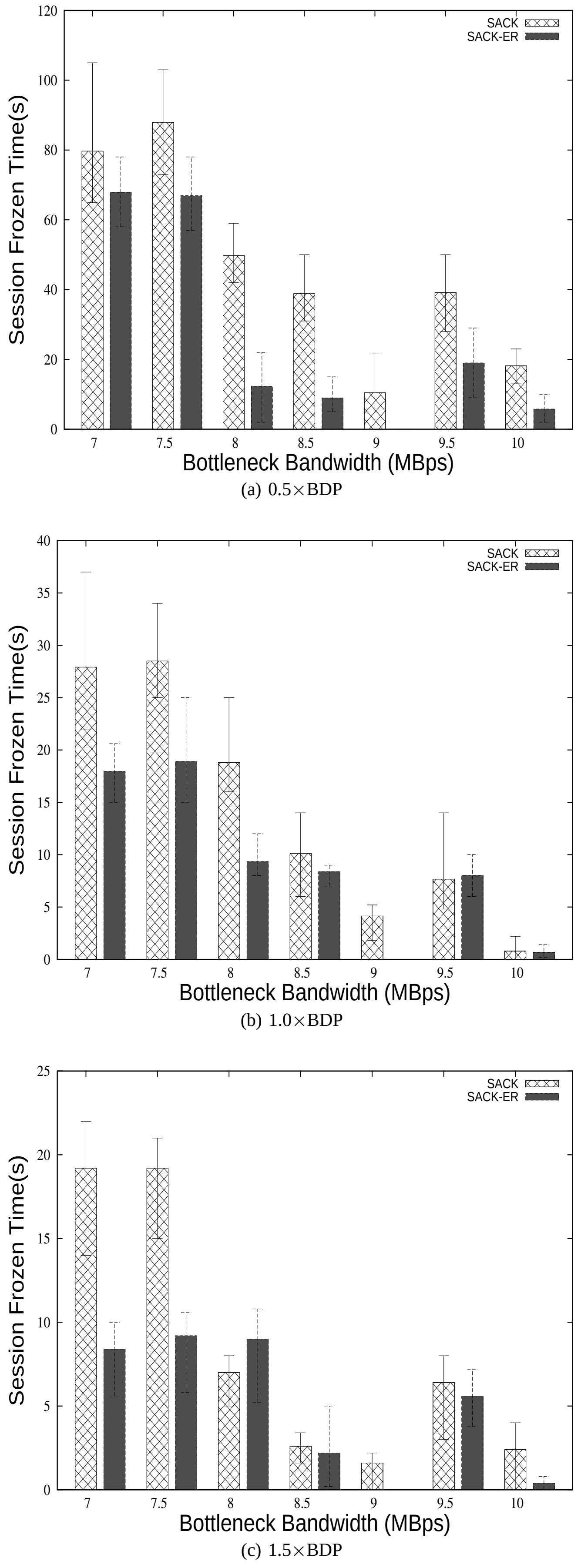
1) Simulation Setup: CBR traffic source is used in the simulation. The default factors used in the simulations (unless otherwise indicated) are shown in Table V. We fix the

Factor	Default Level
Simulation Duration	500s
Bottleneck Bandwidth	7Mbps ∼ 10Mbps
Round-trip Prop. Delay	50ms
Router Buffer	0.5 BDP ∼ 1.5 BDP
FTP Background Flows	20
HTTP Background Flows	300
Background Flow Protocol	TCP-NewReno
MSS	1000Byte
Media Encoding Rate	1Mbps
Receiver Side Buffer	64KB

TABLE V: Simulation Setup For SFT and IPAT Measurement

bottleneck buffer to be from 0.5 to 1.5 times the Bandwidth Delay Product(**BDP**) as small buffer has been highlighted to be suitable for real-time streaming[11]. Each simulation was run for 50 times with a duration of 500 seconds, during which all the FTP background flows started randomly from 0 to 50 seconds, and all of the statistics were collected 100 seconds after the simulations start to avoid taking into account of transient behavior. This setup is also used for **IPAT** measurement in the next section.

2) Simulation Results: Figure 1 shows the average Session Frozen Time of the 50 test runs as well as the error bars representing 95% confidence intervals. It can be seen that **SFT** generally decreases as bottleneck buffer and bandwidth increases. This is obvious as increment in those two factors result in lower packet drop rate. However, the introduction of early retransmission does cause a big difference. Note in all of the above three cases, when bottleneck bandwidth is 7.0Mbps, SACK's total frozen time is 80, 28, and 19 seconds respectively, which means during 16%, 5.6%, and 3.8% of the entire simulation time, the session was frozen, where not a



(a) 0.5×BDP

(b) 1.0×BDP

(c) 1.5×BDP

Fig. 1: Comparison of Early Retransmission and Normal Retransmission

single packet was transmitted to the receiver's side. However, if the early retransmission scheme is employed, the frozen time in those three extreme cases were reduced to 63, 18, and 10 seconds respectively, and the frozen period now only constitutes for 12.6%, 3.6%, and 2% of the simulation time.

It's easy to find out that the improvement is most obvious when bottleneck bandwidth is small and become less evident at higher bandwidth. This verifies our prediction that the Early Retransmission and Loss Recovery scheme in SACK-ER have the most effect in avoiding timeouts under a severely congested environment, where CWND is generally smaller.

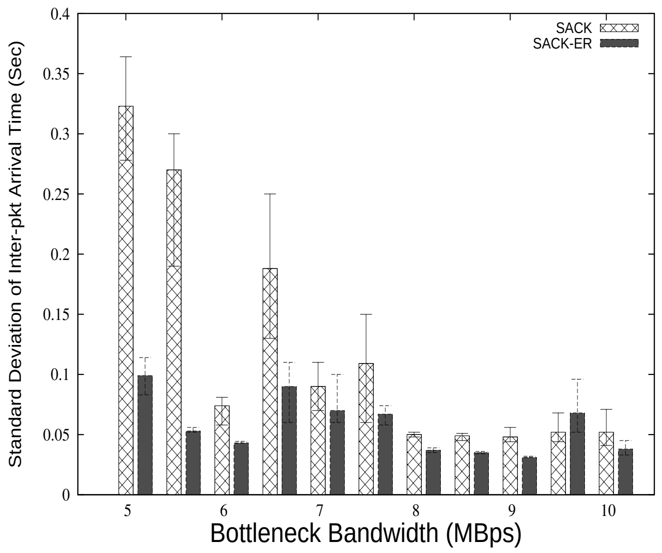
Granted are that none of the above three cases are even remotely acceptable by real-time applications, but since we want to observe the performance of the new scheme, such extreme cases help best demonstrate the improvement. In the following section, where number of late packets are measured, we will alleviate the constraints.

C. Standard Deviation of IPAT

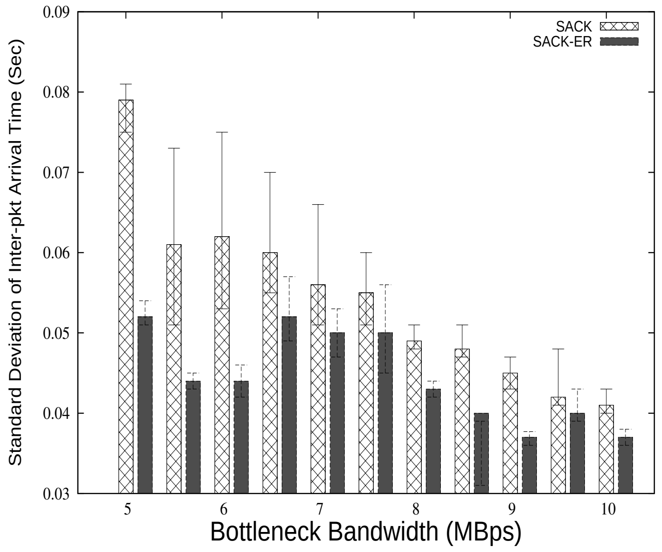
1) Simulation Results: Figure 2. shows the average standard deviation of inter-packet arrival time and 95% confidence interval corresponding to the 50 test runs in the previous section. As discussed in [9], human perception is more sensitive to frame jitter than low quality video when watching streaming video. Standard deviation of inter-packet arrival time was used here to evaluate delay jitter and was shown in Figure 2. 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. As small buffer size is used in our simulation, those two factors 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 occurred, which is usually after one or two RTTs, or, in the worst case, after a RTO fired.

As can be seen from Figure 2, **IPAT** generally decreases as bottleneck bandwidth increases, and SACK-ER has a consistently smaller delay jitter than SACK, which is most obvious when bandwidth is insufficient. The figure shows when bandwidth is 5Mbps, standard deviation of **IPAT** of SACK-ER is only 30%, 63%, and 80% as that of SACK in the three cases respectively. The reason for this is obvious: in extreme conditions like this, RTO occurs often in SACK, and therefore delay caused by retransmission can be as high as several seconds. [4] On the other hand, SACK-ER is able to recover the dropped packet without invoking RTO, and thus keeping delay jitter limited to several **RTT**s, which is in the scale of milliseconds. We also did similar experiments with both protocols in a homogeneous environment, where 20 SACK-ER and SACK flows were put in the same environment respectively, and similar results were obtained.

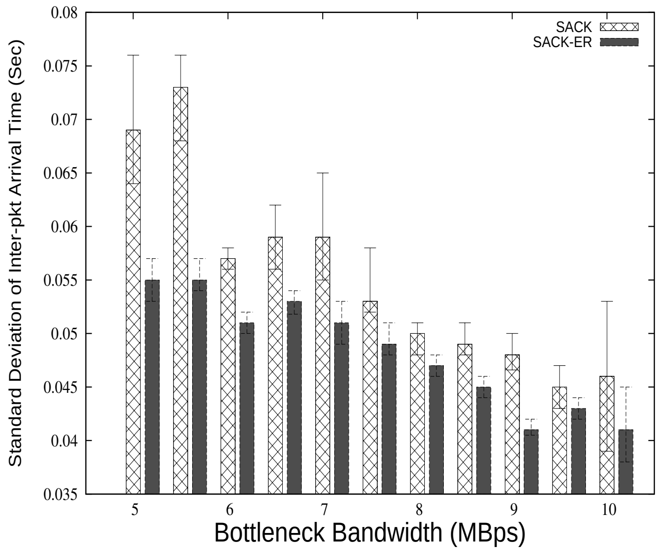
[4]This is because of Retransmission Timer's exponential back-off mechanism. The largest timeout value we observed was 16 seconds.



(a) 0.5×BDP



(b) 1.0×BDP



(c) 1.5×BDP

Fig. 2: Standard Deviation of IPAT under various conditions

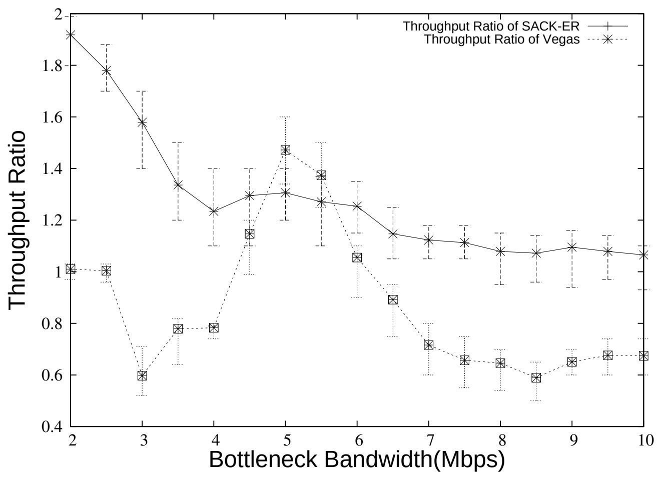


Fig. 3: Comparison of Fairness Between SACK-ER and Vegas With SACK When Bottleneck Buffer = 1.0 BDP

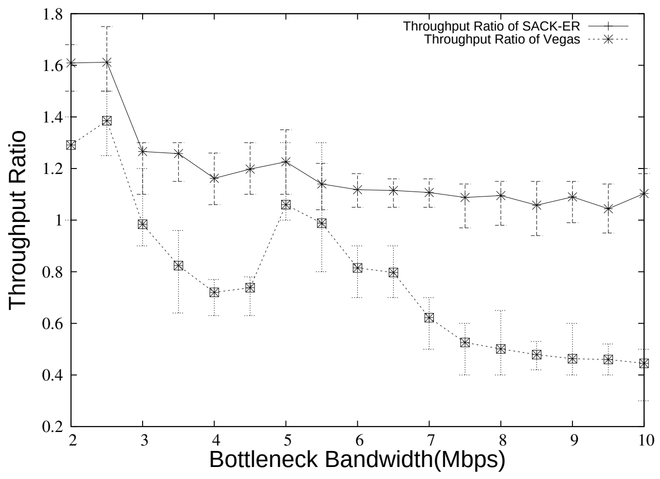


Fig. 4: Comparison of Fairness Between SACK-ER and Vegas With SACK When Bottleneck Buffer = 1.5 BDP

D. *TCP Fairness*

In order to evaluate the fairness of SACK-ER, we put it into various network conditions and measure the ratio between it and the average throughput of the background SACK flows. We also put Vegas in the same condition to compare its performance and fairness with those of SACK-ER.

1) Simulation Setup: The network environment parameters used in the simulations here are similar to those in SFT and IPAT measurement, with two differences: **a)** TCP-SACK are used as background flows; **b)** Bottleneck bandwidth varies from 2Mbps to 10Mbps. Each setup was run for 50 times, and averages were taken.

2) Results: Figure 3 and 4 show the average throughput ratio of the 50 test runs when bottleneck buffer size is 1.0×BDP and 1.5×BDP respectively. The error bars show 95% confidence intervals. The rest of the case follow the similar pattern. We can see clearly from those results that SACK-ER does have a higher throughput than SACK when network congestion is severe (90%, and 60% higher in the two cases respectively when bottleneck bandwidth is 2Mbps), and stays within 20% higher than SACK when congestion gets alleviated. This is because in those cases, congestion window size will normally be relatively bigger, and SACK-ER's recovery scheme, i.e. Lower-bounding CWND and CWND Inflation will rarely happen, and Early Retransmission alone, as mentioned by Van Jacobson will not make a big difference in improving throughput[12]. However, as we shall see later, those two features still play an important role in reducing retransmission timeouts in this kind of network environment.

TCP Vegas, on the other hand, didn't maintain a constant throughput in most cases. It totally lost to SACK when bottleneck buffer is 1.5×BDP, and had similar performance in the case when buffer equals to 1×BDP. This is confirmed by the findings in [13], where the authors show that Vegas can only compete fairly with Reno[5] when small buffer size is used.

E. *Number of Late Packets*

1) Simulation Setup: In the previous section, to better show how the early retransmission scheme can improve network level metrics, some severely congested conditions were used in the simulation. Here, we used a less congested network environment for two reasons: **a)** Severely congested network is simply not viable for streaming, and hence there is no practical reason to explore into that. **b)** We need to show the early retransmission scheme can also improve performance in a not-so-congested environment. This time, we fix the number of background FTP flows to be 50, and scale the bottleneck bandwidth according to the traffic load. 5 streaming media sessions were used and the bottleneck bandwidth is set to be $k \times (num_{bkgnd\_flow} + num_{media\_flow}) \times$ CBR rate, where the scaling factor k takes value in 0.7, 0.9, 1.0, 1.1 and 1.3. We expect the number of late packets to decrease as k gets bigger, and aim at comparing the performance of SACK and SACK-ER in delivery real-time media when bandwidth is plenty. The detailed simulation setup for this section is shown in Table VI. Each setup was run for 20 times, and averages were taken.

2) Simulation Results: The number of late packets as a fraction of total transmitted packets are shown in Figure 5, 6, 7, 8 and 9, where the average results of all the 5 media streams during 20 test runs are plotted against the ratio between media encoding rate and play-back rate when 10 seconds of pre-buffering delay is applied. Again, the error bars show the 95% confidence intervals. For each settings, we also put the average number of **RTO**s for the streaming sessions in Table VII.

It can be seen from the results that, both protocols have smaller fraction of late packets as scaling factor k increases. When the scaling factor equals to 0.7, 0.9 and 1.0, SACK-ER

[5]Not to mention competing with SACK, which can be regarded as the enhanced version of Reno.

Factor	Default Level
Simulation Duration	500s
Pre-buffering Delay	10s
FTP Background Flows	50
HTTP Background Flows	300
Background Flow Protocol	NewReno
CBR Streaming Flows	5
Media Encoding Rate	300Kbps
Bottleneck Bandwidth	k×Encoding rate ×(ftp_flows+media_flow), k = 0.7, 0.9, 1.0, 1.1 and 1.3
Round-trip Prop. Delay	50ms
Router Buffer	BDP
MSS	1000Byte
Receiver Side Buffer	64KB

TABLE VI: Simulation Setup For Late Packets Statistics Measurement

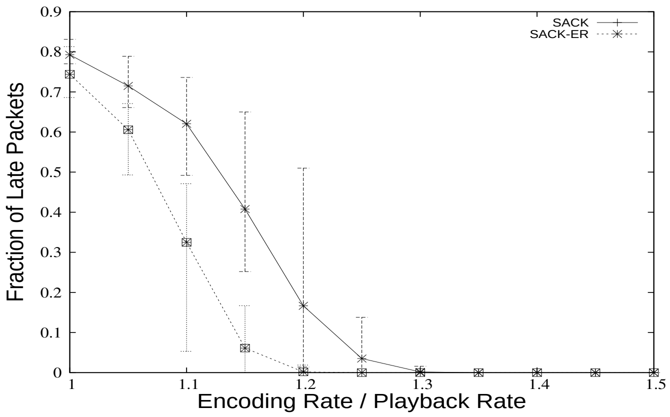


Fig. 7: Fraction of Late Packets of SACK and SACK-ER When Bandwidth Scaling Factor k = 1.0

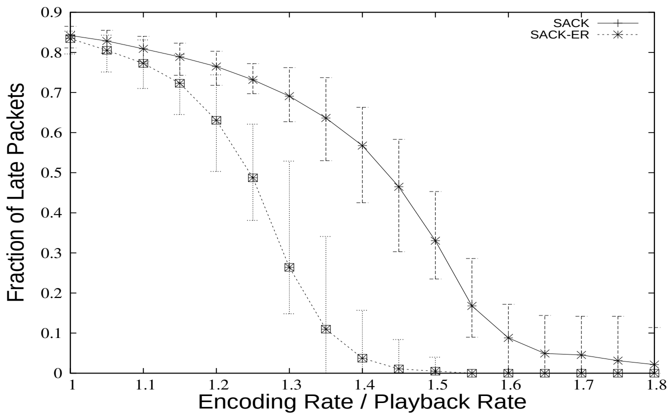


Fig. 5: Fraction of Late Packets of SACK and SACK-ER When Bandwidth Scaling Factor k = 0.7

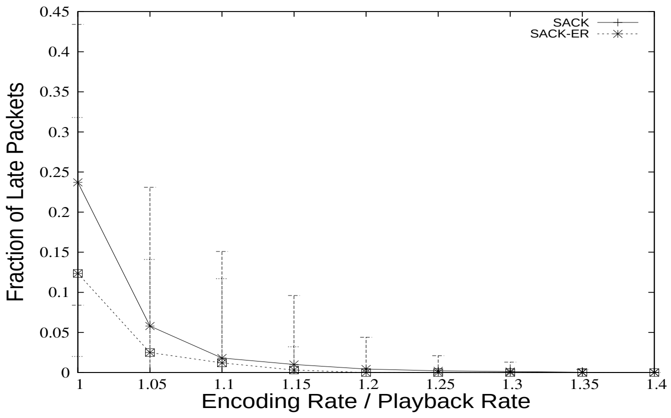


Fig. 8: Fraction of Late Packets of SACK and SACK-ER When Bandwidth Scaling Factor k = 1.1

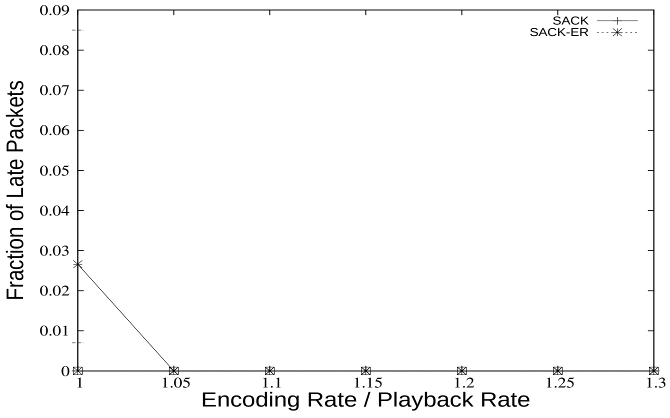


Fig. 9: Fraction of Late Packets of SACK and SACK-ER When Bandwidth Scaling Factor k = 1.3

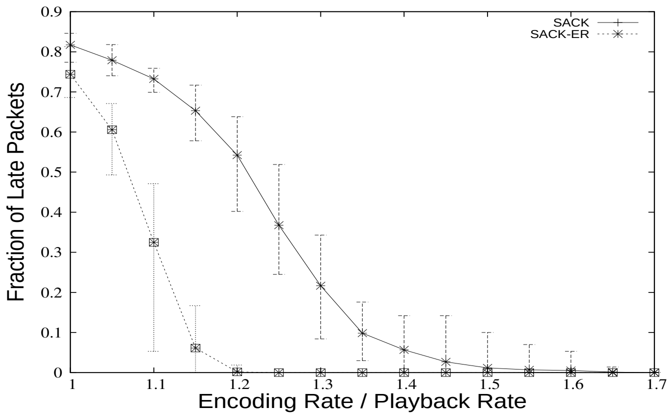


Fig. 6: Fraction of Late Packets of SACK and SACK-ER When Bandwidth Scaling Factor k = 0.9

can achieve 0 late packet when the ratio between encoding rate and play-back rate is 1.55, 1.35 and 1.25(i.e. play-back rate is 64% ,74% and 80% of encoding rate) respectively. In the other two cases, 0 late packets can be achieved when the ratio is 1.2 (play-back rate is 83% of encoding rate) and when the two rates are equal. Under the same conditions, SACK can have 0 late packet only when the ratio between encoding rate and play-back rate is higher than 1.8, 1.6, 1.3, 1.25 and 1.05.

We also noticed large variance in some of the late packets statistics gathered from the simulations, which can be seen from the big confidence intervals at some of the data points in Figure 6, 7 and 8. But we believe they don't pose any problem to the validity of our results for two reasons: **a)** As mentioned earlier, we only focus our attention on 0 late packets case, and from Figure 5, 6, 7, 8 and 9 it can be seen that points indicating 0 late packets all have negligible variances. **b)** Despite the large variances, SACK-ER has a considerably lower number of late packets than SACK in almost all the cases.

Scaling Factor k	**Number of RTOs**	
	SACK	**SACK-ER**
0.7	337	73
0.9	240	65
1.0	234	70
1.1	161	77
1.3	113	14

TABLE VII: Comparison of RTO Number Between SACK and SACK-ER During Constrained Streaming

From Table VII, it's obvious that SACK-ER has a constantly lower number of RTOs than SACK even in the case of big scaling factor values, where bandwidth is enough for delivering streaming media. Our explanation for this is because we include HTTP sessions in our simulation, which will send out huge burst of packets during a very short period at some random time. Even if bandwidth is plenty, suddenly injecting huge burst of packets into the network will cause severe buffer overflow at the bottleneck, thereby causing multiple packets in a CWND to drop, and RTO will very likely to happen[6]. However, since SACK-ER is equipped with the features that enable it to recover from packet drops and prevent RTO from happening, timeouts caused by HTTP packets bursts are also reduced to a great extent, which is verified by the big difference in the number of RTOs between SACK-ER and SACK in the cases where k is big. The impact of HTTP packet bursts and the RTOs they incur also cause packets to miss their deadlines even when bandwidth is plenty. This can be best illustrated by Figure 9, where, if play-back rate equals to encoding rate, SACK still have around 3% late packets even when k = 1.3. On the other hand, SACK-ER manages to maintain 0 late packets under the same condition. Viewing this from a real-time perspective, by avoiding RTOs, SACK-ER's early retransmission and CWND inflation scheme allow its loss recovery process to be faster than its normal counterpart and therefore ensures timeliness as well as reliability, which ultimately enable it to provide a better **QoS** in delivering real-time content.

Note the difference between our simulation and the ones performed in [6] and [5] is that, when calculating number of late packets, we use play-back rate as a fraction of the actual media encoding rate instead of the streaming throughput, and this is what we meant "a tighter constraint" in the earlier sections. Since Constrained Streaming is used in the simulation, the throughput of the streaming session can never exceed the encoding rate. Therefore, we are using a higher standard in evaluating the protocol's ability to deliver real-time media, and even in those cases SACK-ER has a better performance than any of the results shown in [6] and [5].

[6]We believe timeouts caused by HTTP bursts become a larger portion of the total timeouts as the scaling factor increases.

V. Conclusion

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

Our simulation results show that TCP-ER was able to recover from packet lost with less Retransmission Timeout incurred and ensures lower inter-packet arrival jitter, in various settings.

Compared with TCP-Vegas, TCP-ER can achieve higher throughput in severely congested environment while stays fair to TCP in a not-so-congested environment.

Finally, at application level, constrained streaming using TCP-ER also has considerably less number of late packets when applying a tighter real-time constraint.

Our future work includes making TCP-ER more robust to packet reordering and integrating it into other delay based protocols.

References

[1] S. S. J. van der Merwe and C. Kalmanek, "Streaming video traffic: Characterization and network impact," *Proc. of the 7th International Workshop on Web Content Caching and Distribution*, 2002.

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

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

[4] 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.

[5] 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.

[6] 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.

[7] J. B. U. Hengartner and T. Gross, "Tcp vegas revisited," *In Proceedings of IEEE INFOCOM*, March 2000.

[8] V. Jacobson, "Congestion avoidance and control," *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pp. 314–329, 1988.

[9] L. Z. Liren Zhang 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.

[10] 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.

[11] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 281–292, 2004.

[12] V. Jacobson, "Problems with arizona's vegas," *end2end-tf mailing list*, March 1994.

[13] J. Mo, R. J. La, V. Anantharam, and J. Walrand, "Analysis and comparison of tcp reno and vegas," *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, pp. 1556 – 1553, 1999.