SACK TCP with Probing Device

Bing Liang Choong Seon Hong

School of Electronics & Information, Kyung Hee University

bing@networking.khu.ac.kr, cshong@khu.ac.kr

Abstract

This paper describes a modification to the SACK (Selective Acknowledgement) Transmission Control Protocol's (TCP), called SACK TCP with Probing Device, SACK works in conjunction with Probing Device, for improving SACK TCP performance when more than half a window of data lost that is typical in handoff as well as unreliable media. It shows that by slightly modifying the congestion control mechanism of the SACK TCP, it can be made to better performance to multiple packets lost from one window of data.

Keywords: congestion control, congestion window, retransmission timeout, error detection, error recovery.

I. Introduction

Current implementations of TCP use an acknowledgment number field that contains a cumulative acknowledgment, indicating the TCP receiver has received all of the data up to the indicated byte. A selective acknowledgment (SACK) option allows receivers to additionally report non-sequential data they have received. When coupled with a selective retransmission policy implemented in TCP senders, considerable savings can be achieved.

In this paper, we illustrate SACK TCP performs the best when less than half a window of data lost as comparison with Tahoe and Reno TCP. So we will then suggest a simple probing Device and an Immediate Recovery strategy are grafted into SACK TCP, which is responsive to the detected error conditions by alternating Slow Start and Immediate Recovery, in order to improve SACK TCP performance when more than half a window of data lost, and compare its results with the unmodified version.

This work is supported by University ITRC project of MIC.

II. Tahoe, Reno and SACK TCP

Tahoe TCP: Tahoe TCP congestion control algorithm includes Slow Start, Congestion Avoidance, and Fast Retransmit [4]. It also implements an RTT-based estimation of the retransmission timeout. In the Fast Retransmit mechanism, a number of Successive (the threshold is usually set at three), duplicate acknowledgments (dup ACKs) carrying the same sequence number triggers a retransmission without waiting for the associated time-out event to occur. The window adjustment strategy for this "early time-out" is the same as for the regular time-out: Slow Start is applied. The problem, however, is that Slow Start is not always efficient, especially if the error was purely transient or random in nature, and not persistent. In such a case the shrinkage of the congestion window is, in fact, unnecessary, and renders the protocol unable to fully utilize the available bandwidth of the communication channel during the subsequent phase of window re-expansion.

Reno TCP: The Reno TCP modified the Fast Retransmit of Tahoe TCP to include Fast Recovery. The new algorithm prevents the communication path ("pipe") from going empty after Fast Retransmit, thereby avoiding the need to Slow Start to re-fill it after

a single packet loss. Fast Recovery operates by assuming each dup ACK received represents a single packet having left the pipe. Thus, during Fast Recovery the TCP sender is able to make intelligent estimates of the amount of outstanding data.

A TCP sender enter Fast Recovery after the threshold of dup ACKs is received, the sender retransmits one packet and reduces its congestion window by one half. Instead of Slow Start, as is performed by a Tahoe TCP sender, the Reno TCP sender uses additional incoming dup ACKs to clock subsequent outgoing packets.

In Reno TCP, the sender' s *usable* window becomes *min (awin, cwnd + ndup)* where *awin* is the receiver' s advertised window, *cwnd* is the sender' s congestion window, and *ndup* is maintained at 0 until the number of dup ACKs reaches threshold, and thereafter tracks the number of duplicate ACKs. Thus, during Fast Recovery the sender “inflates” its window by the number of dup ACKs it has received, according to the observation that each dup ACK indicates some packet has been removed from the network and is now cached at the receiver. After entering Fast Recovery and retransmit a single packet, the sender effectively waits until half a window of dup ACKs have been received, and then sends a new packet for each additional dup ACK that is received. Upon receipt of an ACK for new data, the sender exits Fast Recovery by setting *ndup* to 0.

Reno TCP's Fast Recovery algorithm is optimized for the case when a single packet is dropped from a window of data. The Reno TCP sender retransmits at most one dropped packet per round-trip time. Reno TCP significantly improves upon the behavior of Tahoe TCP when a single packet is dropped from a window of data, but can suffer from performance problems when multiple packets are dropped from a window of data.

SACK TCP: The SACK option follows the format in [1]. From [1], the SACK option field contains a number of SACK blocks, where each SACK block reports a non-contiguous set of data that has been received and queued. The first block in a SACK option is required to report the data receiver's most recently received segment, and the additional SACK blocks repeat the most recently reported SACK blocks.

The congestion control algorithms in our SACK TCP are a conservative extension of Reno TCP's congestion control. The main difference between the SACK TCP and the Reno TCP is in the behavior when multiple packets are dropped from one window of data.

As in Reno TCP, the SACK TCP enters Fast Recovery when the data sender receives *threshold* dup ACKs. The sender retransmits a packet and cuts the congestion window in half. During Fast Recovery, SACK TCP maintains a variable called pipe that represents the estimated number of packets outstanding in the path. (This differs from the mechanisms in the Reno TCP.) The sender only sends new or retransmitted data when the estimated number of packets in the path is less than the congestion window. The variable pipe is incremented by one when the sender either sends a new packet or retransmits an old packet. It is decremented by one when the sender receives a dup ACK packet with a SACK option reporting that new data has been received at the receiver.

Use of the pipe variable decouples the decision of *when* to send a packet from the decision of *which* packet to send. The sender maintains a data structure, the *scoreboard* that remembers acknowledgments from previous SACK options. When the sender is allowed to send a packet, it retransmits the next packet from the list of packets inferred to be missing at the receiver. If there are no such packets and the receiver's advertised window is sufficiently large, the sender sends a new packet.

The sender exits Fast Recovery when a recovery acknowledgment is received acknowledging all data that was outstanding when Fast Recovery was entered.

III. Comparisons of Tahoe, Reno and SACK TCP

From [2], it can be seen that when there is only one segment dropped, Tahoe TCP goes into Slow Start after retransmit the lost segments and in the process of, the *cwnd* closes all the way to its initial value. In case of Reno and SACK TCP, the *cwnd* is reduced to half of the value it had before the error, their performances are better than Tahoe TCP.

But when multiple segments are dropped from one window of data, Reno TCP has performance problems. These problems result from the need to await retransmission timer expiration before reinitiating data flow, because Reno TCP retransmits at most one

dropped packet per round-trip time. However, SACK TCP recovers without having to wait for a retransmit timeout, with the addition of selective acknowledgments and selective retransmission, a sender has a better idea of exactly which packets have been successfully delivered. Given such information, a sender can avoid unnecessary delays and retransmissions, resulting in improved throughput as compared to Reno TCP.

Therefore, from above comparison, it has been deduced now that SACK TCP will be the best option to improve TCP performance.

IV. Probing Device and Immediate Recovery

We introduced a Probing Device for error detection and an Immediate Recovery strategy that is responsive to the nature of the error detected.

Probing Device: a "Probe Cycle" [5] consists of a structured exchange of "probe" segments between the sender and the receiver that monitor network conditions. The sender enters a probe cycle after a time-out event happen. When the sender initiates a probe cycle during which data transmission is suspended and only probe segments (header without payload) are sent. A lost probe or acknowledgment will not cause TCP time-out extension, instead, reinitiates the cycle. The probe cycle terminates when network condition have improved sufficiently that the sender can make two successive round-trip time (RTT) measurements from the network. Hence suspending data transmission for the duration of the error, when the probe cycle is completed, the sender compares the measured probe RTTs and determines the level of congestion. Had congestion been the possible cause of the drop, the sender would have applied a Slow Start. The enhanced error detection mechanism allows for "Immediate Recovery" (full-window recovery) when the error is detected to be transient.

Hence, a Probing Device models two properties: (i) it inspects the network load whenever time-out event is detected and rules on the cause of that error and, (ii) it suspends data transmission till the end of probe cycle, thereby forcing the sender to adapt its data transmission rate to the actual conditions of the channel.

Immediate Recovery: That is, neither the congestion window nor the Slow Start threshold is adjusted downwards. Time-out values during probing are also not adjusted.

The logic here is that having sat out the error condition during the probe cycle and finding that network throughput is improved at the end of the probe cycle, an aggressive transmission is more clearly indicated. The Immediate Recovery avoids the Slow Start and/or the congestion avoidance phase, immediately adjusts the congestion window to the recorded value prior to the initiation of the probe cycle.

V. Proposal

We know that SACK TCP perform the best as compared to Tahoe and Reno TCP when dropped packets are less than the half of window. However we can see from [3], if more than half a window of data is lost, SACK TCP will result in a retransmit timeout, and followed by a Slow Start. This sequential events underutilizes the network over several round-trip times, has an effect on TCP throughput, which results in SACK TCP performance degradation.

Rather than passively wait for retransmit timeout followed by a Slow Start, actively distinguish network nature of error: transient error or persistent error, then implements different error recovery strategy Slow Start and Immediate Recovery respectively to transient error or persistent error for further improving SACK TCP throughout when multiple packets drop in a window of data. So we propose that SACK TCP combines with Probing Device.

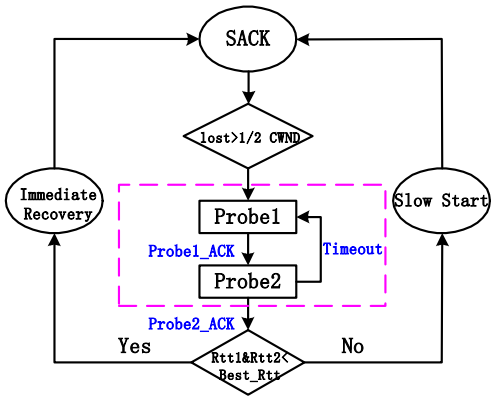

Fig 1. SACK TCP with Probing Device

VI. Simulation

Our simulations were run with OPNET Modeler, with from 10 to 11 packets dropped within a window of data. The simulated scenario is shown in Figure 2.

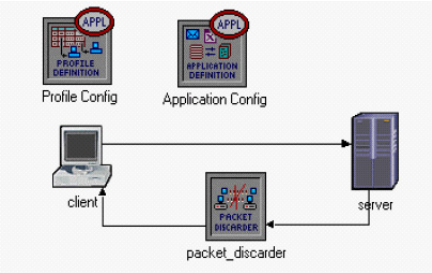

Fig 2. Simulation Scenario

It has two stations: a client and a server connected by a 1.5 Mbps line. In between the two stations is a packet discarder that discards packets going through it. The value of the number of packets to be discarded is set at the start of the simulation. We have set the packet discarder to drop 10 and 11 packets in a 0.5 sec period. We use the existing TCP model provided in OPNET. The scenario is set to simulate a 1.6 MB file transfer from server to client using FTP. The main statistics gathered were the TCP *congestion window size* for the server.

The Congestion Window comparison of SACK TCP with 10 and 11 segment drops is shown in Figure 3.

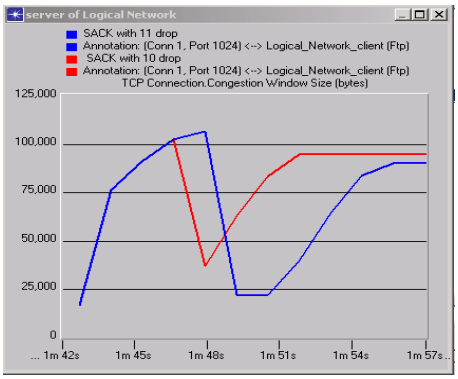

Figure 3. SACK TCP Congestion Window comparison with 10 and 11 drops

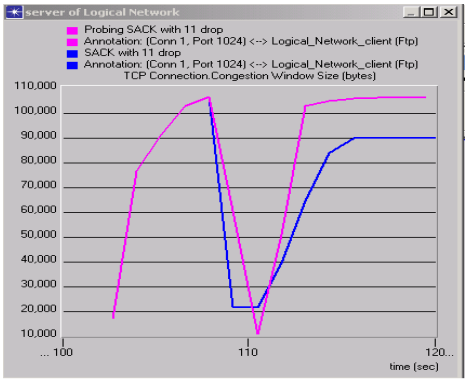

Figure 4. Congestion Window comparison with 11 drops: SACK with Probing and SACK

From Figure 3, we can observe that SACK TCP takes a considerably more time to recover from 11 (more than half a window) segments are dropped.

From Figure 4, it can be seen that SACK with Probing Device give a reduced congestion window recovery time when there is 11 segments drop as compared with original SACK. SACK with Probing Device after time-out event, paused the data transmission and entered in probing cycle. Once detected $Rtt1\&Rtt2 \leqslant Best\_Rtt$ and resumed transmission with Immediate Recovery. The congestion window was adjusted upward immediately since the error was considered to be transient. As expected, the SACK with Probing Device recovers from 11 segments drop without having to wait for a retransmit timeout.

VII. Conclusion

In this paper, we compared Tahoe, Reno and SACK TCP and showed that SACK TCP works best when multiple packets are lost from less than half a window. Then a modification was proposed to SACK TCP, SACK TCP was in conjunction with Probing Device, to improve SACK TCP performance when more than half a window of data lost. We compared its behavior to original SACK TCP, and results show that the modified version performed better as expected.

References

[1]. M.Mathis, J.Mahdavi, S.Floyd, A.Romanow, TCP selective acknowledgment options, IETF RFC 2018, 1996

[2]. Kevin Fall and Sally Floyd, Simulation-based Comparisons of Tahoe, Reno, and SACK TCP, Computer Communication Review, 1996

[3]. M. N. Akhtar, M. A. O. Barry and H. S. AI-Raweshidy, Modified Tahoe for Wireless Networks Using OPNET Simulator

[4]. Matthew Mathis and Jamshid Mahdavi, Forward Acknowledgment: Refining TCP Congestion Control, Proceedings of SIGCOMM'96, August 1996

[5]. A. Lahanas and V. Tsaoussidis, Improving the Performance of TCP in networks with Wireless Components using Probing Devices, IEEE WCNC, March 2002

[6]. A. Lahanas and V. Tsaoussidis, Behavior of TCP-Probing with Hand-offs, CSREA, June 2001