WORST-CASE PERFORMANCE LIMITATION OF TCP SACK AND A FEASIBLE SOLUTION

K.N. Srijith, Lillykutty Jacob, A.L. Ananda

School of Computing, National University of Singapore, Singapore 117543, {srijith,jacobl,ananda}@comp.nus.edu.sg

Abstract - In the present implementation of Transmission Control Protocol (TCP) selective acknowledgment (SACK), every SACK block needs 8 bytes to carry information about the received packets, back to the sender. Since TCP Options field has a fixed length, there is a limit on the number of SACK block that can be carried by the acknowledgment packets. Under some error conditions, this limitation can force the TCP sender to retransmit packets that have already been received successfully by the receiver. This paper puts forward a proposal to modify the present SACK implementation, in order to prevent these unwanted retransmissions. We show that the proposed implementation of SACK mechanism increases the throughput of SACK enabled TCP connections.

Keywords – TCP SACK, Transmission Control Protocol, TCP Options, TCP SACK Limitation

I. INTRODUCTION

New Reno is the present default Transmission Control Protocol (TCP) implementation in most systems. However, when multiple packets are lost from a window of New Reno implementation, TCP may end up either re-transmitting packets that might have already been successfully received, or re-transmitting at most one dropped packet per round-trip time. To overcome this limitation, a selective acknowledgment (SACK) mechanism was defined in RFC 2018 [1]. In TCP SACK, the receiver can inform the sender about all the segments that have been received successfully, allowing the sender to retransmit only the segments that have actually been lost.

However, each SACK block needs 8 bytes to convey its information to the sender. Because of this, the number of blocks that can be carried by an acknowledgment packet is restricted to 4 (or even 3, if TCP Timestamp option [2] is used, as is usually the case). This restriction can, under certain error conditions cause unnecessary retransmission of successfully received packets. This in turn will lead to unnecessary reduction of TCP congestion window and a decrease in the efficiency of TCP connections.

In this paper, we propose a simple mechanism to overcome this limitation, thus making SACK implementation more efficient. We give a brief introduction to TCP SACK in Section 2 and elaborate on its limitation in Section 3. In Section 4, we put forward our proposal and provide a numerical example of the working. In Section 5, we show how the proposed changes improve TCP SACK performance by comparing the results obtained using simulations on the Network Simulator (NS) [3]. We conclude in Section 6.

II. TCP SACK

TCP Selective Acknowledgment (SACK) mechanism was defined in RFC 2018 [1], and later extended in RFC 2883[4]. Using TCP SACK, the receiver can inform the sender about all the segments that it has received successfully, allowing the sender to retransmit only the segments that have really been lost. This helps overcome the limitation of TCP New Reno which otherwise would have retransmitted the packets that might have already been successfully received, or retransmit at most one dropped packet per round-trip time when multiple packets are lost from a window.

SACK helps TCP survive multiple segment losses within a single window without incurring a retransmission timeout. It also provides additional information about congestion state, helping TCP recover faster.

Several studies have been conducted on the efficiency of TCP SACK. In one paper [5], Floyd and Fall analyzed the performance of TCP SACK with respect to Tahoe and New Reno using simulations, and found that SACK was able to provide better performance. In [6], Allman et al. discussed the performance of TCP SACK over satellite links.

III. LIMITATIONS OF SACK

In [7], Floyd addressed various issues related to the behavior and performance of TCP SACK. One important limitation mentioned by the author is that TCP SACK requires 64 bits (8 bytes) to represent the upper and lower bound sequence numbers of every block it selectively acknowledges. Since TCP protocol limits the maximum length of the options field to 40 bytes and SACK is usually implemented along with TCP Timestamp options, an acknowledgment packet can carry a maximum of only 3 blocks' information.

In the extension proposed to SACK in RFC 2883 [4], a new mechanism called DSACK was proposed wherein; the first block of SACK is used to carry information about the latest duplicate packet received. This further reduces the amount of actual SACK information that can be carried in the ACK packets. Similarly, if other TCP options are enabled, this

maximum number would decrease further. This is a very likely scenario as more and more options to TCP are being put forward and accepted, particularly in the wireless networks environment.

This rather small maximum number of SACK block information can lead to efficiency problems in the performance of TCP. Under certain packet loss scenarios, the TCP sender will end up retransmitting packets that have already been received successfully. An example of such a situation is described in [7], which is reproduced here as it forms an interesting scenario for our simulations. In this scenario, it is assumed that there is at least a congestion window of 11 packets, with at least a loss of four data and three ACK packets. Table 1 describes the scenario in more detail.

Table 1. The Error scenario

Data Packets	ACK Packets	Sender reaction
1	Normal ACK –1	Send 12
2 (lost)		
3	Dup ACK-1,3-3	No action
4	Dup ACK-1,3-4	No action
5	Dup ACK-1,3-5	Retrx pkt. 2
6	Dup ACK-1,3-6 (lost)	
7 (lost)		
8	Dup ACK-1,8-8, 3-6 (lost)	
9 (lost)		
10	Dup ACK-1,10-10,8-8, 3-6 (lost)	
11 (lost)		
12	Dup ACK-1,12-12,10-10,8-8	Retrx pkt. 6

As shown in the table, data packets 2,7,9 and 11 are lost. Also, ACKs of data packets 6,8 and 10 are lost. As a result, data packet 6 gets retransmitted unnecessarily. This is because, the duplicate ACK that was sent when packet 12 was received did not have enough space to carry the information that packet 6 had already been received (it contained only SACK blocks 12-12, 10-10 and 8-8 and not 3-6). Had this information been conveyed, the unnecessary retransmission of packet 6 could have been avoided.

IV. OUR PROPOSAL

A. *Theory*

As mentioned earlier, the main constraint that prevents TCP from sending information about more blocks that were received, is that the TCP header's option field has a size limit of 40 bytes and that 8 bytes are needed to represent every SACK block. Since we cannot make any changes to the packet structure of TCP, the aim should be to reduce the amount of data needed to represent a SACK block. The SACK option format as defined in RFC 2018 [1] is as shown in Fig. 1.

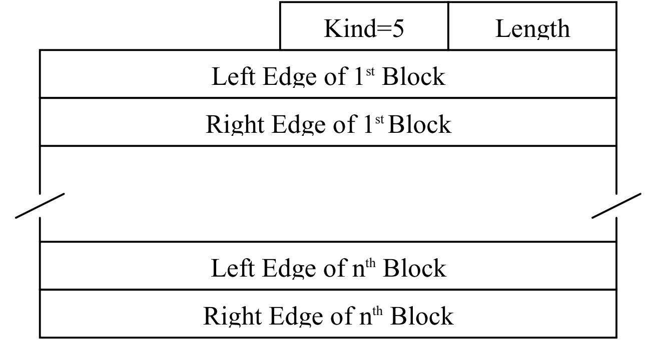

Fig. 1. Current SACK option format

Our proposal is to modify this structure as follows:

- Instead of sending the absolute 32-bit sequence number for all the edges of all the blocks, send 32-bit sequence number for only the right edge of the 1st block (let us denote it by A).

  For the rest of the edges, we consider two different alternative means of representation:

  1. In the first alternative, represent each edge as offset from edge A. We denote them by O_{12}, O_{21}, O_{22}… O_{n1}, O_{n2}, where O_{12} is the offset of the left edge of first block from A, O_{21} & O_{22} are respectively the right and left edges of the second block, and so on.
  2. The second alternative is to represent each edge as an offset from the previous edge. So for the left edge of the first block (O_{12}), compute the offset from A. For O_{21} compute it with respect to O_{12}. For O_{22} compute it with respect to O_{21}, and so on.

- Find out the biggest number among these offsets (denote it by O_{max}). Let X be $\lceil \log_2 (O_{max}) \rceil$ (where $\lceil x \rceil$ is the smallest integer larger than x). This means that we can represent all the offsets using 'X' bits. We need to send this number 'X' to the data sender within the SACK option fields. So the proposal is to change the format of the SACK option as shown in figure 2.

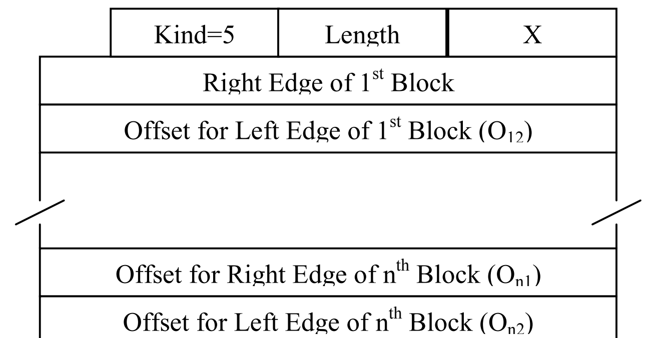

Fig. 2. Proposed SACK option format

For the rest of the paper, we use the first alternative. However, the second alternative is clearly more efficient as it saves more space in the TCP options field. From the proposed format shown in Fig. 2 note that:

- Since the sequence numbers range from 0 to 2^{32}-1, the maximum value that X can take is 32. We need 5 bits to send the value of X. To keep it simple, we allocate 1 byte for this purpose. This is the extra byte that the new format has after the 'Length' field, labeled 'X'.
- The first field after 'X' will be the right edge of the 1st block - a 32-bit sequence number.
- The next field (O_{12}) is the offset of the left edge of the 1st block with respect to the right edge. We can represent this number using X bits (not the usual 32 bits!).
- All the offsets are computed with respect to the right edge of the 1st block, as this is the only absolute 32-bit sequence number that will be sent to the data sender.

At the TCP sender side, when it receives an ACK with SACK data in it, it reads the value of X and A. Then, using blocks of X bits, it reads all the offset values and recreates the edges by subtracting the offset from the value of A.

B. *Example*

It is easier to understand the proposed changes using an example. The following is one of the examples used in RFC 2018 [1]. It is used here so that the difference can be compared. Assume the starting sequence number is 5000 and that the data transmitter sends a burst of 8 segments, each containing 500 data bytes.

In the presented scenario, every other segment (2nd, 4th, 6th etc.) is lost. Table 2 illustrates how the current implementation handles the scenario and Table 3 shows the case of the proposed implementation.

Table 2. Case with the current implementation

Triggering Segment	ACKS	First Block		Second Block		Third Block		Forth Block	
		L edge	R	L	R	L	R	L	R
5000	5500								
5500	(Lost)								
6000	5500	6000	6500						
6500	(Lost)								
7000	5500	7000	7500	6000	6500				
7500	(Lost)								
8000	5500	8000	8500	7000	7500	6000	6500		
8500	(Lost)								
9000	5500	9000	9500	8000	8500	7000	7500	6000	6500
9500	(Lost)								
10000	5500	10000	10500	9000	9500	8000	8500	7000	7500
15000	(Lost)								

Table 3. Case with the proposed implementation

Triggering Segment	ACKs	First Block		Second Block		Third Block		Forth Block		Fifth Block	
		R	L	R	L	R	L	R	L	R	L
5000	5500										
5500	(Lost)										
6000	5500	6500	500								
6500	(Lost)										
7000	5500	7500	500	1000	1500						
7500	(Lost)										
8000	5500	8500	500	1000	1500	2000	2500				
8500	(Lost)										
9000	5500	9500	500	1000	1500	2000	2500	3000	3500		
9500	(Lost)										
10000	5500	10500	500	1000	1500	2000	2500	3000	3500	4000	4500
15000	(Lost)										

We see that, in the current implementation, by the 9th segment (sequence number 9000), we would use up 34 byes (2 + 4*8) by sending information of 4 blocks. When the 11th segment is received, we are only able to send information about the latest 4 blocks, loosing information about the 6000 - 6500 segment! Now let us take a look at the proposed implementation shown in Table 3.

$X = \log_2 4500 = 13$

Bits used up = 8 (kind) + 8 (length) + 8 (X) + 32 (sequence number for A) + 9*13 (offsets) = 173 bits = 22 bytes. We have used up just 22 bytes of the available 40. The improvement is more pronounced when TCP has to use the Timestamp option along with SACK. Then SACK can convey information about only three blocks, and problem is encountered at the 9th segment itself, for the current implementation. However, if the proposed implementation is used:

$X= \log_2 3500= 12$

Bits used = 16 (timestamp) + 8 (kind) + 8 (length) + 8 (X) + 32 (sequence number for A) + 7*12 (offsets) = 156 bits (20 bytes). We are still left with 20 more bytes.

V. SIMULATION RESULTS

A. *Experiment 1*

In order to demonstrate the potential of the proposed implementation in resolving the problem faced by the current SACK implementation, we simulated the environment described in Section 3 (Table 1) using NS and observed the transmission of packets through the network. We used the "List" error model of NS for the packet corruption in order to simulate the lossy environment described in Table1.

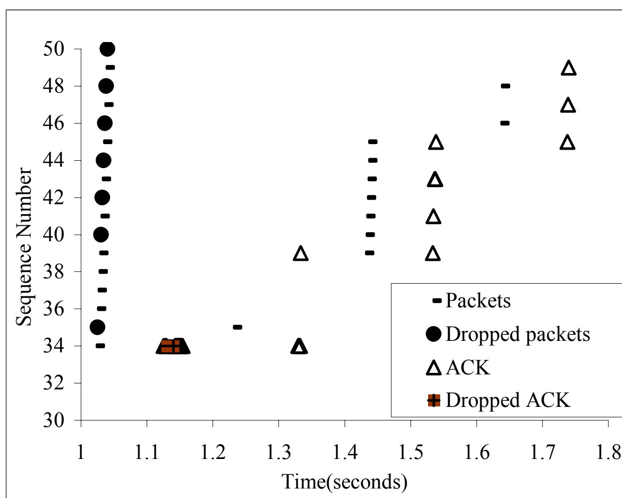

Fig. 3. Performance of original SACK

Fig. 3 shows the packets that were seen in the network when the present SACK implementation was used. Fig. 4 shows the result when we used the modified SACK implementation. In these figures, we are interested in packets whose sequence numbers range from 34 to 50.

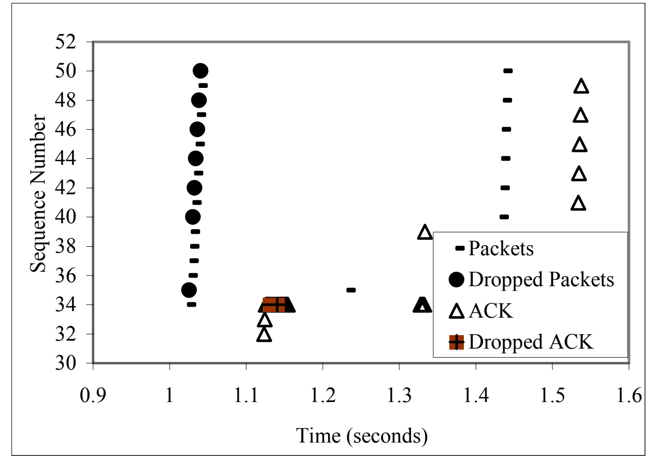

Fig. 4. Performance of modified SACK

We see in Fig. 3 that, because of the limitation of SACK, packets with sequence numbers 39,41,43 and 45 are unnecessarily retransmitted, as shown by two data packets being sent with the same sequence number at different time. However we see that, in Fig. 4, these packets are not retransmitted. This shows clearly that the modified SACK does perform better than the present implementation.

B. *Experiment 2*

As presented in [8], unnecessary retransmission of packets can decrease the throughput of TCP connections. Thus the problem with TCP SACK that we see graphically in Fig. 3 can lead to lower throughout. This is investigated in this experiment.

We looked at the performance gain when file transfer was performed for a specified period of time. We used the two-state Markov error model of NS shown in Fig. 5 to generate the error conditions.

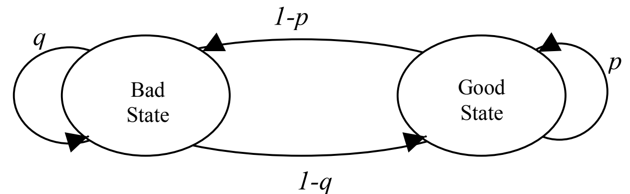

Fig. 5. Two-state Markov model for the lossy link

In this model, the link state is sampled every t_1 seconds when it is in the 'good' state and every t_2 seconds when it is in the 'bad' state. If the link is in a good state (respectively

bad state) at the current sampling instant, then with probability p (respectively q) it will continue to be in the 'good' state (respectively 'bad' state) at the next instant also and with a probability $1-p$ (resp. $1-q$) the transition to the 'bad' state (respectively 'good' state) takes place. When the link is in the 'good' state (respectively 'bad' state), each packet may be corrupted with the probability α (resp. β). The values used in the simulation are given in Table 4.

Table 4. Parameter values used in two state error model

Variable	Source to Destination	Destination to Source
t1	25.0	25.0
t2	5.0	15.0
P	0.05	0.45
q	0.55	0.45
α	0.05	0.05
β	0.25	0.75

Table 5 below shows the throughput measured for different periods of trial runs. Each value in the table is the average of measured throughputs from ten runs.

Table 5. Throughput over different trial run time periods

	50s	60s	120s	300s	600s
No-Offset (Kbps)	162.64	141.52	72.4	93.44	56.64
Offset (Kbps)	169.84	156.0	124.24	94.48	72.72

The above results clearly show that when TCP connections are established over a long period of time, under the error conditions simulated, the new modification to SACK that we have suggested performs better than the existing implementation. We see that the new implementation of SACK always gives a better throughput.

VI. CONCLUSION

Current SACK implementation has the limitation of being able to send a maximum of only 3 or 4 SACK blocks with each ACK. In this paper we propose an alternate representation for the SACK blocks in the option field of the TCP segment for TCP SACK implementation to overcome this limitation. Using examples and simulations, we showed that the modified implementation of SACK produces better TCP performance in terms of the throughput obtained, and makes the SACK mechanism more resilient to the high packet error rates often seen in wireless scenarios.

REFERENCES

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

[2] V. Jacobson, R. Braden, D. Borman, "TCP Extensions for High Performance", *RFC 1323*, IETF, May 1992.

[3] S. McCanne, S. Floyd, "ns-2 Network Simulator" - http://www.isi.edu/nsnam/ns/.

[4] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", *RFC 2883*, IETF, July 2000.

[5] K. Fall, S. Floyd, "Simulation based Comparisons of Tahoe, Reno, and SACK TCP", *Computer Communications Review*, vol.26, pp 5-21, 1996.

[6] M. Allman, D.Glover, NASA Lewis, L. Sanchez, "Enhancing TCP Over Satellite Channels Using Standard Mechanisms", *RFC 2488*, IETF, January 1999.

[7] S. Floyd, "Issues with TCP SACK", *Technical Report*, LBL Network Group, 1996.

[8] A. Romanow, S. Floyd, "Dynamics of TCP traffic over ATM Networks*", IEEE Journal on Selected Areas in Communications,* Vol. 13 No. 4, p 633-641, 1995.