The 5th International Conference on Wireless Communications, Networking and Mobile Computing

Sept. 24-26, 2009 Beijing, China

IEEE
Celebrating 125 Years of Engineering the Future

IEEE COMMUNICATIONS SOCIETY

Volumes

Click on a volume for a list of sessions

Sessions by Volume

Vol 5. Network Technologies (2)

- RFID, Bluetooth and 802.1x Technologies
- Network Protocol and Congestion Control
- QoS and Traffic Analysis
- Network Security

Main Menu
Volumes
Sessions
Authors

The 5th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2009)

Network Protocol and Congestion Control

Main Menu
Volumes
Sessions
Authors

Papers by Session

Main Menu
Volumes
Sessions
Authors

A Segment Based SACK Scheme for Wireless TCP

Lin Cui Jun Song
School of Information Technology Engineering
Tianjin University of Technology and Education
Tianjin, China

Choong Seon Hong
dept. of Comp. Eng.
Kyung Hee Univ.
Suwon, Korea

Seok Joo Koh
dept. of Comp. Science
Kyungpook National Univ.
Daegu Korea

***Abstract*—By definition of RFC 2018, each SACK block has to be indicated by two 32-bit unsigned integers which may introduce significant extra load in reverse ACK path. Moreover, due to a 40-byte length limitation, when errors occur in sudden bursts it is noted that the available option space may not be sufficient to report all sequence blocks and sometimes some unnecessary retransmission may be resulted in. This paper introduces a segment based SACK scheme for Wireless TCP. Simulation results and the theoretical analysis show that this scheme is simpler and more effective than the existing schemes.**

Keywords-TCP, SACK, Option space, Wireless

I. INTRODUCTION

Transmission Control Protocol (TCP) is a connection-oriented transport protocol, which provides a reliable data delivery by employing cumulative acknowledgements (ACKs), sequence numbers, and timers. In TCP, those out-of-sequenced data cannot be covered by acknowledgement number field of TCP header. As a result, TCP sender can only retransmit one missing segment per round-trip time. When multiple packets are lost from a data window, this forces the sender either to wait for the more round-trip times to find out all lost segments or to retransmit some unnecessarily segments due to expiration of timeout timer. Therefore, multiple segments losses from a data window can critically influence throughput performance of TCP.

In order to avoid the performance degrading in the face of multiple missing segments in a data window, TCP Selective Acknowledgment (TCP SACK) [1] scheme has been proposed. With the help of this scheme, the receiver can inform the sender of the current situation of receiving buffer. Thus TCP sender can retransmit only those missing segments selectively as quickly as possible. As defined in RFC 2018 [1], however, each sequence block of data queued at the receiver buffer needs to be defined by two 32-bit unsigned integers in TCP SACK option field, hence a SACK option field that specifies n sequence blocks will have a length of 8*n+2 bytes. Since the maximum length of TCP option field is 40 bytes and Timestamp option is expected for TCP extensions for high performance [2], which takes an additional 10 bytes (plus two bytes padding), SACK option will have the room for only three SACK blocks at most in usual cases. This restriction makes the option space scarce for the sender to convey the entire SACK information under some error burst environments that will cause unnecessary deflation of congestion window (*cwnd*) and throughput degradation of TCP.

On the other hand, the standard TCP SACK scheme [1] also introduces significant extra load in reverse ACK path, especially when a lots of TCP SACK connections are established over the same wireless link, since missing segments are often seen in wireless scenarios and each sequence block needs extra 8 bytes to represent in every duplicate ACK.

In this paper, we introduce a segment based SACK scheme to overcome the shortcoming of the existing SACK mechanism. Simulation results and the theoretical analysis show that this scheme is worthy to be further considered. The rest of this paper is organized as follows. Section 2 simply introduces some related works. Section 3 describes the proposed segment-based SACK scheme in detail. Then we compare the option spaces used by the relative schemes in section 4, show the simulation results in section 5 and conclude this paper in section 6.

II. RELATED WORKS

TCP Reno [3] performs the flow control and congestion control algorithms. The sliding window-based flow control mechanism allows the sender to advance the transmission window upon the reception of an acknowledgment (ACK) that indicates the last in-order packet has been received successfully by the receiver. When a packet loss occurs on a congested link, the sender will receive three duplicate acknowledgments or the sender's retransmission timeout timer will expire. These events will activate the sender's congestion control mechanism, by which the sender reduces the size of congestion window to relieve the link congestion.

TCP SACK [1] is an extension of TCP Reno [3], in which TCP SACK [1] does not change the underlying congestion control algorithms and uses the same algorithms to increase and decrease the congestion window. The main difference between them is the behavior in the face of multiple missing segments in one data window. In particular, at the sender side the sender maintains a data structure, called scoreboard, which records acknowledgments from previous SACK options. When the sender is allowed to transmit packets, it will retransmit the packets in sequence from the list of missing reports, as long as the congestion window and the receiver's advertised window are sufficiently large. Otherwise, if there is no such missing packet, then the sender will send the new ones else.

As shown in Fig. 1, the standard SACK option format is defined in RFC 2018 [1], its limitations have been described in section I.

978-1-4244-3693-4/09/$25.00 ©2009 IEEE

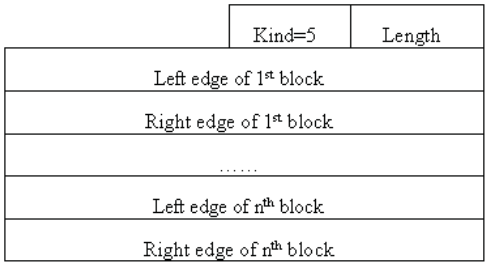

Figure 1. Standard SACK option format

Up to now, many proposals have been presented to use TCP option field to convey some useful information in order to improve TCP performance under various scenarios. For example, Timestamp is the option most frequently used in many schemes and is usually carried by option field.

In TCP HACK scheme [5], a separate checksum is also proposed in TCP option field for the header portion of each segment while the traditional TCP carries only one checksum for the whole segment. By means of this strategy in lossy environments, even if corruptions occur in the segment's data portion due to the random nature of the bit errors, as long as the header is integrated, TCP receiver will send a prompt special ACK back to the sender to indicate the corruption information at once. Nevertheless, HACK itself cannot find those irretrievable segments unless it uses TCP SACK scheme as the reinforcement leverage upon the out of order ones. In such case, both HACK and SACK schemes use the same option field to convey their information that makes the option space scarcer.

In [6], a concept of offset is presented in a revised SACK scheme for sending the more SACK blocks' information. By presentation in [6], the sender only sends one 32-bit absolute sequence number for the right edge of the 1st sequence block where the sequence number is highest (denote it by A), instead of sending all absolute ones for two edges of each sequence blocks. For the edges of the rest sequence blocks, the sender represents them as offsets with respect to A as shown in Fig. 2.

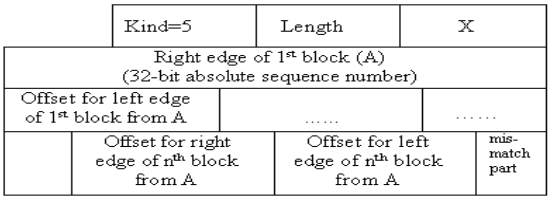

Figure 2. SACK option format proposed in [6]

It is noted that each edge of a sequence block needs an offset to represent and the length of each offset field (labeled 'X') is decided by the maximum offset (Omax) with respect to A as follows:

$$X = \lceil \log_2 O_{max} \rceil \qquad (1)$$

III. SEGMENT-BASED SACK SCHEME

Normally, TCP does not change segment size once the TCP connection is established, and all segments preceding the last one has the maximum segment size (MSS) as their size. Hence, given a offset in segment unit, if only we have an absolute sequence number as a baseline, then we can easily get any edge's absolute sequence number by computing the formula as follows:

edge's absolute SN = baseline + offset*segment size (2)

We well use this characteristic and present a very simply segment-based SACK algorithm for the case in which TCP does not change segment size. In particular, at receiver side, regarding the maximum in-order sequence number (denoted by acknowledgement number field) as baseline, the receiver represents SACK block edges by respective offsets with respect to the baseline or previous edge (relative offset) instead of absolute sequence numbers. Whereas, when received an ACK, the sender regards the acknowledgement number field as baseline and recovers the closest edge's absolute sequence number at first by applying formula (2), and then regards this edge as new baseline to recover the next one. In this way, the sender can be aware of entire situation of receiver's sequence space correctly.

Moreover, in order to minimize the size of SACK space, referring to Fig. 3, in addition to using Acknowledgement number field as the baseline, we represent every SACK block by one or two segment-based offsets. As for using one or two offsets, it depends on the size of the gap previous that SACK block. If the gap is a single segment gap, then one offset is enough; if not, two offsets are necessary. In other words, we ignore those single segment's gaps and represent each SACK block's length by a positive number less than 128 and each multiple segments' gap by a positive number larger than 128. This is because, as long as the sender extracts two positive number less than 128 in sequence from the SACK option field, the sender can infer that the receiver ignore a single gap by default since gap and SACK block are interleaved in nature. This method does not influence efficiency, but do significantly reduce SACK option space.

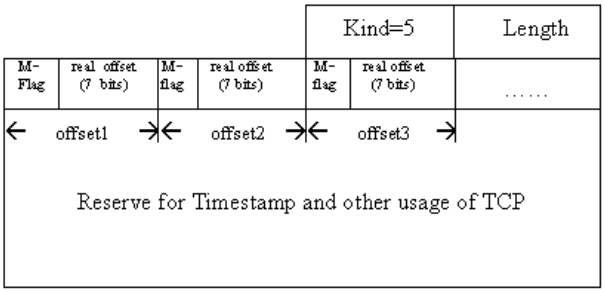

Figure 3. Proposed SACK option format

In Fig. 3, it is noted that the first bit of each offset is labeled by "M-flag", which is the flag of multiple segments' gap. This flag will be set to 1 when the offset represents a multiple segments' gap, while the other 7 bits represent real segment number of either a sequence gap or a SACK block. Since 7 bits

can represent up to 2^7-1=127 segments, a real offset larger than 127 means either there is a very large gap composed of more than 127 missing segments in receiver's sequence space or the receiver has sent more than 127 duplicate ACKs. This will directly lead TCP sender to cause either retransmission timeouts because of too long delay or fast retransmissions after receiving more than three duplicate ACKs. Hence one byte is enough for an offset. As a result, 40 bytes can be available for (40-2)/2=19 SACK blocks at least. Considering the fact that we ignore those single segment's gaps which most frequently occur in sequence space, the available maximum number of SACK blocks is far bigger than 19.

IV. AN EXAMPLE

Through comparing with RFC 2018 [1] and the proposal in [6] under the same scenario, we can understand the difference between the segment-based scheme and the others. In this sampled scenario, we assume that the starting sequence number is 5000 and each segment contains 1000 data bytes. TCP sender sends a burst of 11 segments and every other is lost.

A. Case of RFC 2018

Table I lists the SACK information of RFC 2018 with the sampled scenario. From the table, we can see that by the 9th segment (sequence number 13000-13999), 34 bytes (2 + 4*8) have been used up for sending information of 4 SACK blocks. When the 11th segment is received, only the latest 4 blocks' information can be sent, losing information about the 3rd segment (sequence number 7000-7999). Especially, if timestamp option is used, then just only 3 blocks can be sent, that is, the 5th segment's information will also be discarded (sequence number 9000-9999). While error burst occurs in reverse ACK path, TCP sender may unnecessarily retransmit 3rd and 5th segments which have actually arrived at receiver in fact.

TABLE I. CASE OF RFC 2018

Data packet	ACK Num field	First Block		Second Block		Third Block		Forth Block	
		L edge	R	L	R	L	R	L	R
5000	Normal ACK: 6000	NA	NA	NA	NA	NA	NA	NA	NA
6000 (Lost)									
7000	Dup ACK: 6000	7000	8000	NA	NA	NA	NA	NA	NA
8000 (Lost)									
9000	Dup ACK: 6000	9000	10000	7000	8000	NA	NA	NA	NA
10000 (Lost)									
11000	Dup ACK: 6000	11000	12000	9000	10000	7000	8000	NA	NA
12000 (Lost)									
13000	Dup ACK: 6000	13000	14000	11000	12000	9000	10000	7000	8000
14000 (Lost)									
15000	Dup ACK: 6000	15000	16000	13000	14000	11000	12000	9000	10000

B. Case of Proposal in [6]

It is noted that in the sampled scenario the maximum offset is equal to 9000 (16000 – 7000), thus every offset has to be allocated 14 bits according to formular (1) in the proposal of [6].

Hence, the total size of SACK option in this case can be calculated as follows:

8 (kind) + 8 (length) + 8 (define X) + 32 (absolute sequence number for A) + 9 (the number of offsets)*14 (length of each off-set field) = 182 bits = 23 bytes.

If considering timestamp option in addition to SACK scheme, the used TCP option space will be up to 33 bytes. That is to say, there is no enough spare room for other schemes (e.g. HACK+SACK, etc.). Therefore, this scheme can also increase network load of reverse ACK path. Table II shows the detailed SACK information of the proposal in [6].

TABLE II. CASE OF PROPOSAL IN [6]

Data packet	ACK Num field	X	First Block		Second Block		Third Block		Forth Block		Fifth Block	
			R	L	R	L	R	L	R	L	R	L
5000	Normal: 6000	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
6000 (Lost)												
7000	Dup ack: 6000	10	8000	1000	NA	NA	NA	NA	NA	NA	NA	NA
8000 (Lost)												
9000	Dup ack: 6000	12	10000	1000	2000	3000	NA	NA	NA	NA	NA	NA
10000 (Lost)												
11000	Dup ack: 6000	13	12000	1000	2000	3000	4000	5000	NA	NA	NA	NA
12000 (Lost)												
13000	Dup ack: 6000	13	14000	1000	2000	3000	4000	5000	6000	7000	NA	NA
14000 (Lost)												
15000	Dup ack: 6000	14	16000	1000	2000	3000	4000	5000	6000	7000	8000	9000

C. Case of Segment-based SACK Scheme

In the case of the segment-based SACK scheme as shown in Table III, we only need 7 bytes (1 (kind) + 1 (length) + 5*1 (offsets)) to convey all necessary SACK information that needs 23 bytes at least for the same effort in [6]. Thus, the proposed scheme could be more effective, especially as performing in conjunction with other mechanisms for improving TCP performance.

TABLE III. CASE OF SEGMENT BASED SACK SCHEME

Data packet	ACK Num field	offset1	offset2	offset3	offset4	Offset5
5000	Normal ACK: 6000	NA	NA	NA	NA	NA
6000 (Lost)						
7000	Dup ACK: 6000	1	NA	NA	NA	NA
8000 (Lost)						
9000	Dup ACK: 6000	1	1	NA	NA	NA
10000 (Lost)						
11000	Dup ACK: 6000	1	1	1	NA	NA
12000 (Lost)						
13000	Dup ACK: 6000	1	1	1	1	NA
14000 (Lost)						
15000	Dup ACK: 6000	1	1	1	1	1

V. SIMULATIONS

We performed the simulations for 200 seconds over a single TCP flow traversing a 2M bandwidth with 0ms delay wireless link using the ns-2 network simulator.

In the experiments, we compared SACK performance between our proposal and the standard SACK scheme. For each packet, 1000 bytes user data are contained and packet size is 1040-byte. In addition, we used two states error model for TCP connection to suffer two consecutive bad states from around 8.078 to 8.108 second. The related parameters are listed in table IV.

TABLE IV. PARAMETERS OF TWO STATE ERROR MODEL

State	Average Period	Transition Probability	Packet corruption Rate
Good	t1=0.5 s	p=0.95, (1-p)=0.05	α= 0
Bad	t2=001.5 s	(1-q)=0.55, q=0.45	β =0.25

In the experiment of RFC 2018, the burst errors directly lead to five SACK blocks in receiving buffer. In detail, the first block consists of a single segment with number 1314, the second block consists of two segments with number 1316 and 1317, the third block is also a single segment block with number 1320, the fourth block consists of three segments with number from 1322 to 1324 and the fifth block consists of two segments with number 1327 and 1328. Because the standard TCP SACK can only send up to three SACK blocks' information in ns2, the receiver has to report the last three SACK blocks and lose the first and second ones. As a result, it leads the sender to retransmit those segments with number 1314, 1316 and 1317 at 8.105 second and 8.107 second respectively which actually have been received by receiver.

On the contrary, in the experiment of segment-based SACK scheme, although five SACK blocks are also created in receiver's sequence space, the scheme can avoid those unwanted retransmissions effectively.

VI. CONCLUSION

In this paper, we propose a segment-based SACK scheme in order to save the more option space for other schemes and decrease network load of reverse ACK path. Through analysis in example, we can see that the proposed scheme waste the much less option space to send the same SACK information, compared to the standard SACK scheme and the proposed in [6]. Also, simulation results show that the proposed scheme can achieve the better performance in the face of high packet error's scenario often seen in wireless environments by means of sending the more SACK blocks' information.

ACKNOWLEDGMENT

This research was partially supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute for Information Technology Advancement)" (IITA-2008-C1090-0804-0004).

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] W. R. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC2001, Jan. 1997.

[4] Network Simulator (ns-2), available from http://www.isi.edu/nsnam/ns/.

[5] R. K. Balan, B. P. Lee, K. R. R. Kumar, L. Jacob, W. K. G. Seah, A. L. Ananda, "TCP HACK: TCP Header Checksum Option to Improve Performance over Lossy Links," in Proceedings of the IEEE INFOCOM 2001, vol. 1, pp. 309-318, April 2001.

[6] Srijith, K., Jacob, L. and A. Ananda, "Worst-case Performance Limitation of TCP SACK and a Feasible Solution," Proceedings of 8th IEEE International Conference on Communications Systems (ICCS), November 2002.