TCP Protocol: Different Flavours and Implementation

Rushikesh Rajendra Dumbre (2017IS03)

Date:05/09/2017

Abstract

TCP-SACK was developed by Sally Floyd, Matt Mathis etc in 1996 published in RFC 2018. In TCP-SACK, TCP uses Selective Acknowledgement(SACK) option in the options section of TCP packet. This option allows the receiver to acknowledge unordered blocks of packets which were received correctly, as well as the acknowledgement of the last orderly recieved packet as in normal TCP. The left and the right edge of the discontinuous data is specified in the TCP options refered to as SACK block. Almost all implementations of TCP support TCP-SACK. In this report I attempt to explain 2 and 3 packet loss situations in TCP SACK.

0.1 Introduction

The evolution of TCP was as follows:

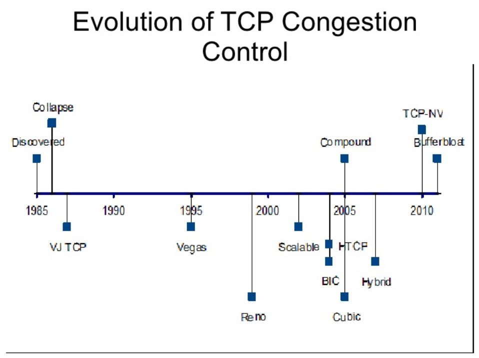

Figure 1: Evolution of TCP[2]

0.1.1 TCP Tahoe

TCP Tahoe was implemented before 1990. It included algorithms like slow start, fast retransmit and congestion avoidance. The algorithms were as follows:

1. Slow Start:- Slow Start initially sets the congestion window(cwnd) size to 1. The value of cwnd is increased by 1 per acknowledgement(ACK) recieved. It doubles the cwnd every round trip time assuming that no packet was lost. The algorithm does this till the value of cwnd reaches slow start threshold(ssthresh). Beyond that value congestion avoidance algorithm takes over.

Algorithm 0.1 Slow Start

```
if(cwnd < ssthresh){
        cwnd++ for each ACK
}
else{
        Congestion Avoidance
}
```

2. Fast Retransmit:- When sender recieves three duplicate ACK it assumes that the packet after the duplicate ACK has been lost due to network congestion. It the retransmits the packet and sets $cwnd = 1$ and the slow start begins. The Fast Retransmit is also triggered when retransmission timeout occurs when ACKs are lost.

Algorithm 0.2 Fast Retransmit

```
if(3 duplicate ACK for nth packet){
        retransmit (n+1)th packet
        cwnd = 1
        Slow Start
}
```

3. Congestion Avoidance:- When *cwnd* >= *ssthresh* the increment in cwnd is slowed down to avoid congestion. Beyond ssthresh cwnd is increased by $1/cwnd$ per ACK recieved.

Algorithm 0.3 Congestion Avoidance

```
if(cwnd >= ssthresh){
        cwnd += (1/cwnd) for each ACK
}
```

0.1.2 TCP Reno

TCP Reno included a new algorithm Fast Recovery.

1. Fast Recovery:- When a Fast Retransmit occurs due three duplicate ACKs recieved, the sender **DOES NOT** set its cwnd to 1. Instead it sets its $ssthresh = cwnd/2$ and sets its $cwnd = cwnd/2 + 3$ and enters into the Fast Recovery mode. While sender waits for new acknowledgement it increases cwnd by 1 for each subsequent duplicate ACK. When sender recieves a new ACK it sets $cwnd = ssthresh$. If a retransmit occurs due to retransmission timeout the cwnd is set to 1 and Slow Start phase begins.

Algorithm 0.4 Fast Recovery

```
if(Fast Retransmit){
        ssthresh = cwnd/2
        cwnd = cwnd/2 + 3
}
for(subsequent duplicate ACK){
        cwnd++
}
if(new ACK){
        cwnd = ssthresh
        Congestion Avoidance
}
```

0.1.3 TCP New Reno

TCP New Reno introduced the concept of partial acknowledgements. An ACK which does not acknowledge all the packets that were outstanding at the beginning of fast recovery is called as Partial ACK. The sender must remember the highest unacknowledged packet just before entering the Fast Recovery mode.

1. Fast Recovery:- Just like TCP Reno, in TCP New Reno sender enters Fast Recovery Mode when it recieves 3 duplicate acknowledgements and increases cwnd by 1 as subsquent duplicate ACK arrives. When sender recieves new ACK it checks if it is a partial or complete ACK. If it is a partial ACK then sender knows that the packet just after the recently acknowledged packet is lost and it retransmits the packet. However after retransmission the sender **REMAINS** in the Fast Recovery mode and sets it *cwnd* = *ssthresh*. It only leaves the Fast Recovery mode when it recives a Complete ACK.

Algorithm 0.5 New Reno Fast Recovery

```
if(Fast Retransmit){
        ssthresh = cwnd/2
        cwnd = cwnd/2 + 3
}
for(subsequent duplicate ACK){
        cwnd++
}
if(Partial ACK){
        cwnd=ssthresh
}
if(Complete ACK){
        cwnd=ssthresh
        Congestion Avoidance
}
```

0.1.4 TCP SACK

SACK stands for selective acknowledgement. TCP SACK uses 2 types of TCP options to tell the sender which Packets were recieved and which were lost. SACK has an advantage over previous TCP versions which is that it can retransmit more than 1 lost packet per round trip time while in case of New Reno the sender had to wait for partial acknowledgements and in case of Tahoe and Reno the sender had to recieve 3 duplicate ACKs or wait till retransmission timeout.

1. "SACK Permitted" option which may be sent in a SYN segment. It must not be sent in a non-SYN segment. It is 2 byte option.

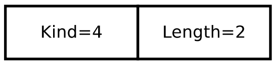

Figure 2: SACK Permitted

2. SACK option itself which consist of Left and right edges of the blocks of data that were transmitted successfully.

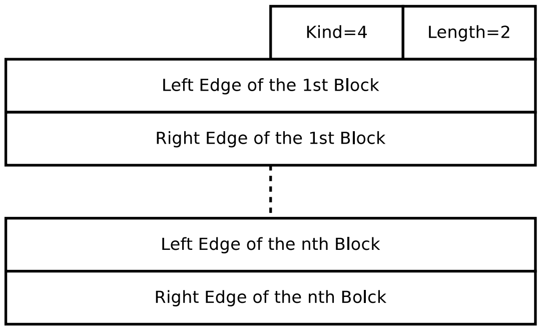
Kind=4
Length=2
Left Edge of the 1st Block
Right Edge of the 1st Block
Left Edge of the nth Block
Right Edge of the nth Bolck

Figure 3: SACK option

TCP SACK introduces a new variable called **Pipe**. Pipe gives the number of outstanding packets in the network. The number of outstanding packets is equal to the number of packets that were sent but not acknowledged during the Fast Recovery.

During Fast Recovery we send packets with respect to value of pipe and not the cwnd. The algorithm for TCP SACK is as follows:-

Algorithm 0.6 TCP SACK

```
Fast Retransmit:On third duplicate ACK
        pipe = cwnd - 3
        retransmit 1st unacknowledged
        ssthresh = cwnd/2
        cwnd = cwnd/2

Fast Recovery:After Fast Retransmit
On subsequent duplicate ACKs
        pipe - -


Transmission:
while(pipe < cwnd){
        if(UnSACKed packet){
                retransmit
                pipe++
        }
        else{
                transmit new packet
                pipe++
        }
}

Partial ACK
        pipe -= 2
        Transmission
```

0.2 TCP SACK Packet Loss Analysis

In this section we analyse the performance of TCP SACK in cases of two and three packet loss with help of examples

0.2.1 Two Packet Loss

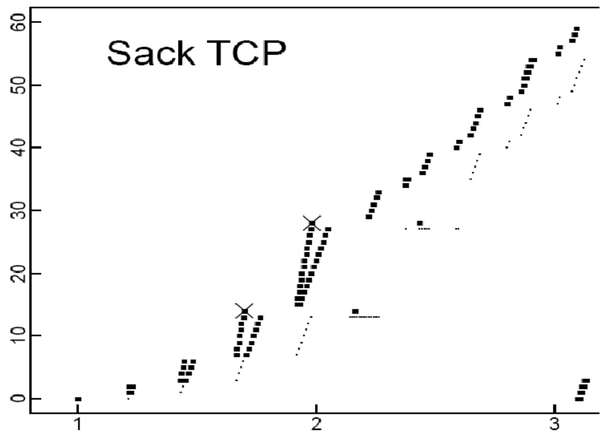

Figure 4: TCP SACK two packet loss[1]

0.2.1.1 Packet-wise Analysis

Window	cwnd	pipe	ssthresh	Event
0	1			
1-2	1+1=2			
3-6	2+2=4			
7-14	4+4=8			Packet 14 is lost
14-28	8+7=15			Packet 28 is lost
3 duplicate ACK + 10 Subsequent ACK				
In Fast Recovery Transmission depends on value of pipe				
upto 28	15/2=7	15-3-10=2	15/2=7	Retransmit 14 and send 29-33
Pipe = 7 and Partial ACK upto 27				
upto 35	7	7-2=5		2 packets can be sent as pipe-cwnd=2
34-35 are sent and 5 dup ACK are recieved, so pipe=5+2-5=2				
upto 39	7	2		28 is retransmitted and 36-39 is sent pipe=7
upto 41	7	5		2 ACK recieved so 2 Packets sent pipe=7
Complete ACK acknowledges upto 34 and sender exits Fast Recovery				
35-41	7			Congestion avoidance mode is started

Table 1: Two Packet Loss

0.2.2 Three Packet Loss

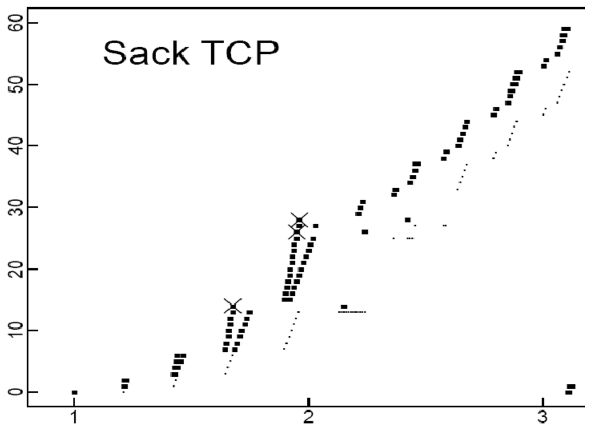

Figure 5: TCP SACK three packet loss[1]

0.2.2.1 Packet-wise Analysis

window	cwnd	pipe	ssthresh	Event
0	1			
1-2	1+1=2			
3-6	2+2=4			
7-14	4+4=8			Packet 14 is lost
14-28	8+7=15			Packets 26,28 are lost
3 duplicate ACK + 9 Subsequent ACK				
In Fast Recovery Transmission depends on value of pipe				
upto 31	15/2=7	15-3-9=3	15/2=7	Retransmit 14,26 and transmit 29-31
Pipe =7 and Partial ACK upto 25				
upto 33	7	7-2=5		transmit 32-33 Pipe = 7
3 dup ACK + 1 Partial ACK upto 27, Pipe=7-3-2=2				
upto 37	7	4+3=7		retransmit 28 as sack option in ACK of 29 indicates the loss
2 dup ACK Pipe=7-2=5				
upto 39	7	5+2=7		transmit 38-39
Complete ACK recieved and 28 is acknowledged				
33-40	7			Congestion Avoidance mode started

Table 2: Three Packet Loss

Bibliography

[1] Kevin Fall and Sally Floyd. Simulation-based comparisons of Tahoe, Reno and SACK TCP. *ACM SIGCOMM Computer Communication Review*, 26(3):5–21, 1996.

[2] Stephen Hemminger. A Baker's Dozen of TCP.