Struct tcp {

U32 sacked\_out;/\*通过sack确认的包的数量，按照gso分包后计算\*/

U32 lost\_out;/\*通过sacked\_out计算出来的丢包的数量\*/

};

Tcp协议栈的变化:

Tcp\_sacktag\_write\_queue()

{

/\*tcp\_highest\_sack\_seq ,如果sacked\_out为0，则返回snd\_una;如果highest\_sack为NULL，则返回snd\_nxt;否则，返回highest\_sack的seq\*/

If(!before(start\_seq,tcp\_highest\_sack\_seq(tp)))

{

/\*tcp\_highest\_sack返回highest\_sack，highest\_ack在正常情况下为snd\_una起始的最老帧\*/

Skb = tcp\_highest\_sack(skb);

}

/\*tcp\_sacktag\_skip从skb开始向后寻找seq num大于start\_seq 的skb并返回该skb\*/

Skb = tcp\_sacktag\_skip(skb,sk,start\_seq);

/\*tcp\_sacktag\_walk，从skb开始，通过调用tcp\_match\_skb\_to\_sack来判断当前skb的seq end\_seq是否在传入的start\_seq和end\_seq内，如果在，tcp\_sacktag\_one，tcp\_sacktag\_one的主要作用一是将当前skb的sacked置为TCPCB\_SACKED\_ACKED；作用二是置当前state->flag FLAG\_DATA\_SACKED；作用三是增加sacked\_out为当前skb的pcount;作用四是增加delivered为pcount;判断如果当前skb的seq在tcp\_highest\_sack\_seq之后，此时由于sacked\_out不为零，所以tcp\_highest\_sack\_seq返回那一段的skb的seq，所以当前skb肯定在tcp\_highest\_sack\_seq之后，接着调用tcp\_advance\_highest\_sack(sk,skb)，此时将highest\_sack置为当前skb的下一个skb；如果下一个skb的seq也在传入的end\_seq之前，则进入tcp\_shift\_skb\_data流程\*/

Skb = tcp\_sacktag\_walk(skb,sk,next\_dup,state,start\_seq,end\_seq,dup\_sack)

{

}

}

/\*取当前skb的一部分数据移到前面去\*/

Tcp\_shift\_skb\_data()

{

Prev = skb\_rb\_prev(skb);

In\_sack = !after(start\_seq,TCP\_SKB\_CB(skb)->seq) &&!before(end\_seq,TCP\_SKB\_CB(skb)->end\_seq);/\*

Start\_seq <= seq < end\_seq <= end\_seq\*/

If(!in\_sack)

{

/\*说明当前skb不可继续分拆下去\*/

If(tcp\_skb\_pcount(skb)<=1)

Goto noop;

In\_sack = !after(start\_seq,TCP\_SKB\_CB(skb)->seq);/\*

In\_sack = Start\_seq <= TCP\_SKB\_CB(skb)->seq;

\*/

Len = end\_seq – TCP\_SKB\_CB(skb)->seq;/\*获取sack中的剩余长度\*/

Len mss整数倍对齐；

Tcp\_skb\_shift(prev,skb,pcount,len)

{

Return skb\_shift(to,from,shiftlen)

{

}

}

}

}