Controlling TCP behavior over lossy links

M. Alnuem, J. Mellor and R. Fretwell

***Abstract*—Many previous attempts to improve TCP performance over lossy links especially error discriminators did not propose a comprehensive response in case of non-congestion drops. In this paper we propose a new algorithm to control TCP congestion window reaction to transmission (non-congestion) drops. This algorithm will be the first step for more comprehensive solution. The new algorithm managed to improve TCP performance in non-aggressive way by increasing the average congestion window size. On the other hand, it cuts the congestion window for both congestion and non-congestion errors which help to reduce the effect of error mismatch on the network. Simulation results shows noticeable improvement when the new technique is added to an error discriminator. Also we discuss some limitations to the new algorithm.**

***Keywords*—Error discriminators, Lossy networks, TCP**

I. Introduction

One technique to increase TCP performance over lossy networks is to use TCP error discriminators. Error discriminators are algorithms that can be used to replace standard TCP congestion window mechanism[1][2] and try to understand the cause of the packet drops and then make TCP to act differently for each type of error.

The main aim for an error discriminator is to distinguish between error types with high accuracy to avoid increasing the congestion on then network. This is because many error discriminators has been based on the idea that if we can discriminate between congestion and transmission (non-congestion) errors correctly then the action towards transmission drops can be as simple as not to cut the congestion window in case of non-congestion drops.

The same problem can be looked at from different angle so instead of only trying to increase the error discriminator accuracy we will implement an efficient action for transmission drops which should hold following properties:

- It will not increase the network congestion rate by cutting the congestion window even for transmission errors.
- It will increase TCP performance in case of transmission drops by delaying window cut decision until TCP knows how many packets are dropped.

Adaptive congestion window cut strategy and delaying congestion window cut until TCP receives the full window are the main contribution of this work.

II. New congestion window cut policy for transmission errors

Do we need to cut congestion window for transmission (non-congestion) errors? and assuming we can discriminate between error types, what is the proper action TCP should take with transmission drops? The trivial action when transmission drops occur is to resend the lost packet and avoid reducing the congestion window. This approach has been the base of most sender based end to end solutions like [3] and [4]. This is based on the following reasoning: a general formula to compute TCP throughput by using the round trip time and the window size is [5]:

$$Throughput = \frac{W_i}{RTT_i} \tag{1}$$

Where W_i is the window size in round trip time RTT_i and the average throughput can be captured as following :

$$AvgThroughput = \frac{AvgW}{AvgRTT} \tag{2}$$

So if we can keep the average window size as large as possible during the transmission drops then the throughput should be higher than the conventional TCP (where the window size is cut to half with every drop) assuming we have fixed AvgRTT (i.e. the AvgRTT does not increase with the increase in congestion window). However, we think that a trivial solution is not always the answer. Some times it is better to cut the congestion window even for transmission drops.

For example in high speed networks TCP requires a big window size in order to make use of the available link capacity. Also, If the link capacity is required by other senders, TCP will slow down by cutting the congestion window. However if TCP keeps big window all the time this will result eventually in increasing the load on the network and hence increasing the end to end packet round trip time. Moreover, if one link in the connection path suffers from transmission errors, then the link layer will be busy resending the corrupted packets and hence its goodput will decrease. This will create more delay since the link layer will be forced to buffer the packets until they correctly retransmitted or even to drop them if the buffer size is not enough. So in general it is desirable to control the increase in the congestion window in case of transmission errors for the following reasons:

- Controlling the increase of the congestion window even for transmission errors can prevent undesirable large packet drops during transient congestion phases. If a congestion happen while the congestion window is open wide, a large number of packets can be dropped which make TCP to enter a series of timeout events. These timeout events will force TCP to wait idle and also will increase exponentially with each successive timeout.
- Uncontrolled increase in the congestion window can lead to increase in the network load which will lead to increase in the RTT and any increase in the RTT has mainly two side effects on TCP throughput: the rate of increase in the RTT can be more than the rate of increase in the congestion window size and this will cancel any gain in the throughput. see equation 2. Another effect of the increase in the connection RTT is that it increases the retransmission timeout and hence increasing the period TCP should wait after errors.
- In many networks the link layer is responsible for buffering and retransmitting lost packets cased by link failure. However, if the end point sender keeps sending at high rates with no regard to transmission drops, the link layer will be forced to buffer large amounts of data or even drop some of the packets which will lead eventually to increasing the end-to-end RTT [6].

All these factors will result eventually in increasing the per-packet delay and hence increasing the average round trip time for

M. Alnuem, J. Mellor and R. Fretwell are with School of Informatics, University of Bradford, Bradford BD7-1DP, United Kingdom. (e-mails: {m.alnuem, j.e.mellor, r.j.fretwell}@brad.ac.uk; phone:00441274233913; Fax:00441274233920)

the whole connection (AvgRTT). From that we can see that if the error discriminator does not cut the congestion window in case of transmission drops the RTT may increase in away that could cancel any benefit gained from increasing the congestion window size. For this reason the authors in [6] indicated that not cutting the congestion window for transmission drops is a bad policy and suggested to cut the congestion window by one for every transmission drop.

The algorithm:

We will call the proposed algorithm the congestion window action (CWA) and it works as following: Instead of cutting the congestion window after receiving 3 duplicate acknowledgment, see figure 1, we delay the cut decision until TCP receives all duplicate acknowledgment for current window (i.e. the packets on flight during the drop) see figure 2. Duplicate acknowledgments indicate a packet

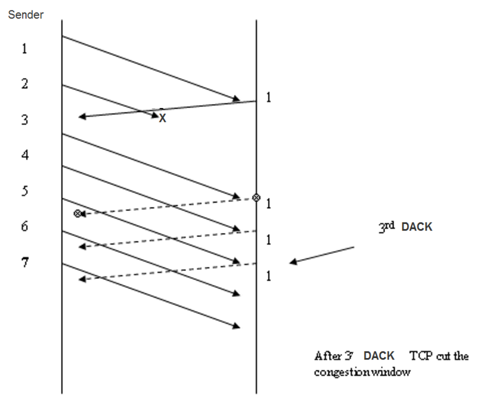

Fig. 1. TCP duplicate acknowledgment action

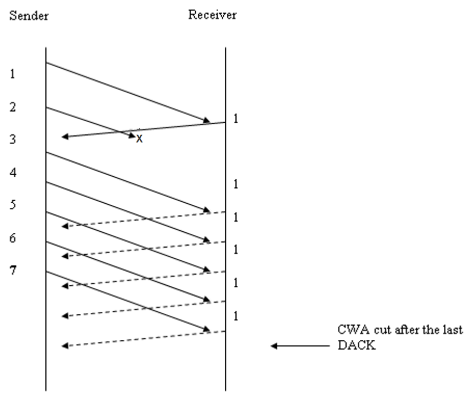

Fig. 2. CWA duplicate acknowledgment action

drop but also indicate how many packets has left the network (received by the other end). Using this information we can estimate how many packets were dropped per window ($droppedpackets = Windowsize - No.DACKs$). In case of transmission drops, instead of cutting the congestion window to half (as TCP) or not cutting it at all we cut number equals to dropped packets. This way TCP cuts the congestion window in a rate related to number of dropped packets. The benefit of this technique is that it improves the performance and avoid increasing the congestion level at the same time. The algorithm is presented in figure 3.

```
Initialization: prev_ack = -1; last_sent = -1
With every received acknowledgment $Ack_i$:
current_ack = $Ack_i$
if (current_ack == prev_ack) then                     ▷ Duplicate ack
    dackcount = dackcount+1
    if dackcount == 3 then                            ▷ Packet drop
        last_sent = $P_{max}$
        resend packet with seqNo = current_ack+1 ▷ No cut for cwnd
    end if
end if
if (current_ack > prev_ack) then                      ▷ no more DACKs
    prev_ack = current_ack
    if (last_sent > current_ack) then  ▷ Some packets still not ackowledged
        compute number of drops and reduce cwnd:
        flight_size = last_sent − current_ack
        num_drops = flight_size − dackcount
        cwnd = cwnd − num_drops
    end if
end if
if timeout==true then
    ssthresh = cwnd/2        ▷ Actually ssthresh = min(2,cwnd/2)
    cwnd = 1
end if
```

Variables:
current_ack: Sequence number of current acknowledgment.
prev_ack : Sequence number of prev acknowledgment (new acknowledgment only).
dackcount : Variable to keeps track of how many duplicate acknowledgment TCP received so far. This variable is set to 0 whenever a new acknowledgment is received.
last_sent: Variable to store sequence number of last sent packet.
P_{max}: Last sent packet.
flight_size: Number of packets sent but not acknowledged yet.
num_drops: Number of packets dropped from this flight.
cwnd: Congestion window size.
ssthresh: Slow Start threshold.
RTO: Retransmission timeout timer. Calculated based in the average RTT.

Fig. 3. CWA Pseudo code

Note that In case of transmission error we keep the ssthresh and only change the cwnd since the drop is not congestion error and probably the link capacity (indicated by ssthreah) has not changed.

The CWA should be used in case of transmission errors only. However, if the error discriminator wrongly used CWA for congestion errors as well then the congestion in the network may increase. To solve this problem we will define another threshold we call it transmission drops threshold (tthresh). It will be used to recored the congestion window size (cwnd) when the first drop occur. It will define the area between the start of congestion avoidance phase (i.e. from ssthresh) and the first drop . Since this is the first drop then we call the cwnd size up to *tthresh* the safe area.

Changes after adding *tthresh* are presented in figure 4. The main change is that after the first drop the value of cwnd is saved in tthresh. Later when another drop occur the error discriminator will check if cwnd $\leq$ tthresh and if so the drop is probably a transmission drop. Otherwise the drop is congestion drop.

```
1: Initialization: prev_ack = -1; last_sent = -1; first_drop = 1
2: With every received acknowledgment $Ack_i$:
3: current_ack = $Ack_i$
4: if (current_ack == prev_ack) then                ▷ Duplicate ack
5:     dackcount = dackcount+1
6:     if dackcount == 3 then                      ▷ Packet drop
7:         last_sent = $P_{max}$
8:         resend packet with seqNo = current_ack+1 ▷ No cut for cwnd
9:         if first_drop then
10:            tthresh = cwnd
11:            first_drop = 0
12:        end if
13:    end if
14: end if
15: if (current_ack > prev_ack) then                ▷ no more DACKs
16:    prev_ack = current_ack
17:    if (last_sent > current_ack) then   ▷ Some packets still not ackowledged
18:        compute number of drops and reduce cwnd:
19:        flight_size = last_sent − current_ack
20:        num_drops = flight_size − dackcount
21:        if cwnd < tthresh then
22:            cwnd = cwnd − num_drops
23:        else
24:            cwnd = cwnd / 2
25:            ssthresh = cwnd
26:        end if
27:    end if
28: end if
29: if timeout==true then
30:    ssthresh = cwnd/2          ▷ Actually ssthresh = min(2,cwnd/2)
31:    cwnd = 1
32:    first_drop = 1             ▷ Initialize first_drop after each timeout
33: end if
```

Variables:
first_drop: Flag to indicate that first drop in this connection has occured.

Fig. 4. CWA+tthresh Pseudo code

III. Performance of CWA

In order to measure the improvement using CWA we will measure TCP goodput (the actual amount of date received regardless of retransmissions) after and before adding CWA to TCP. Also we will measure congestion window size during the connection life time. The topology used represents TCP sender with continuance demand to send data (FTP application with big files) and suffers from transmission errors in one link of the connection path. Similar topology is used to test TCP over lossy links by other authors like [7]. We use this simple topology with one sender because we want only to test the performance with transmission errors (we will use more complex topology later when we add CWA to the error discriminator). The transmission errors are created on the last link using a two-state Markovian model. This model has been used by authors like[8][6][7] to simulate wireless errors.

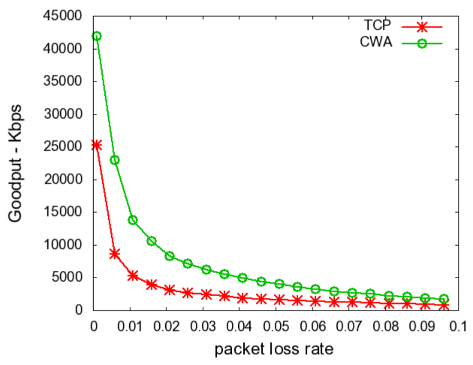

Fig. 5. TCP vs. CWA.

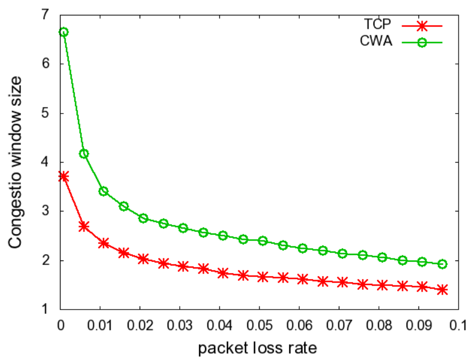

Fig. 6. TCP vs. CWA. log scale congestion window size

The chart in figure 5 shows that after adding CWA to TCP the performance has improved. This improvement is caused by CWA preventing unnecessary congestion window cuts and limiting the cuts to the number of lost packets in case of transmission drops. Figure 6 shows the average congestion window for TCP before and after adding CWA. As we can see CWA has a positive effect on the average window size of TCP (the size is measured in number of packets). The increase in congestion window size will increase TCP sending rate. Also it will help recover errors quickly and hence reducing number and length of retransmission timeout evens. Reducing number of RTO evens will reduce the the total time TCP stays idle and hence will increase the throughput. Our simulation results show improvement in number of RTO events after using CWA. The increase in congestion window size has reduces the chance to have RTO in case of CWA because with bigger window TCP gets more duplicate acknowledgment after drops. These duplicate acknowledgments will trigger lost packet retransmission and will increase the congestion window during the fast recovery phase.

However, due to the fact that the increase in the error rate will increase the timeout durations, the congestion window will not have chance to grow after a timeout event. Moreover, with the increase in error rate many packets will be dropped more than once and since TCP resends the packet only once per window more longer retransmission timeout will occur.

One limitation for CWA is that its performance depends on the timing when transmission errors occur (since CWA sets `tthresh` after first drops). However, in our work TCP should recalculate *tthresh* after each timeout event because TCP will initialize the congestion window and will start building a new window. This is done in the algorithm by using *first_drop* which will be set to one after each timeout and hence allowing *tthresh* to take a new *cwnd*.

Another problem that decreases the performance of CWA is multiple drops per window of data. Since TCP resends only one dropped packet per window the rest will be recovered through timeout. This will increase number and duration of timeout which will affect the performance of CWA. In the future work we will solve the problem by using a multiple drops action algorithm MDA[9].

IV. Performance with error discriminator

The main purpose of this paper is to present the CWA algorithm and to show how it improves TCP performance in presence of transmission errors only. However, the actual benefit of CWA will be when it is added to an error discriminator to implement TCP reaction to transmission errors (as we explained before, error discriminator usually do nothing when transmission errors occur). So in order to make the picture more complete we added CWA to an end-to-end error discriminator based on spike [10] and MDA [9] schemes.

Initial results shows that the new error discriminator, named ED+CWA, managed to outperform TCP (TCP-Reno) with increased transmission error rate and with presence of congestion drop rate between 1%-2%. Figure 7 shows the results. Moreover, the new

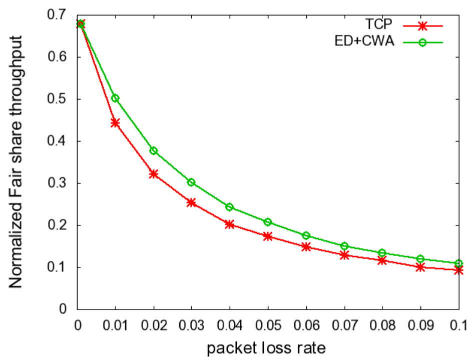

Fig. 7. TCP vs. ED+CWA normalized fair share throughput

error discriminator is able to maintain the same average queue size (around 34 KB) at the bottleneck buffer which indicates that CWA has successfully prevented increasing the congestion level caused by error discriminator mismatches. In our experiments we use a simple yet sufficient topology which contains TCP and UDP sources. The UDP sources are used to create cross traffic. TCP sources are mixture of standard TCP and the new (ED+CWA) protocol we want to test. We have two destinations D1 and D2. The link to the destinations pass through two intermediate routers R1 and R2. ED+CWA pass through the path which suffers from transmission errors at the last hop. Cross traffic is used to create the required level of congestion drops by varying the UDP sources sending rate.

V. Conclusion and Future work

In this paper we presented a new TCP congestion window action (CWA) for transmission(non-congestion) errors based on delaying the congestion window cut decision until TCP has a complete picture of number of dropped packets per window. The algorithm reduces the congestion window size when transmission drops happens using this number (i.e. calculated number of dropped packets). The CWA has been added to TCP and results show improvement in average goodput over TCP. The merit of the CWA is that it defines a TCP congestion window cut policy which able to improve TCP performance and cuts the congestion window even for transmission errors which will prevent side effects caused by error discrimination mismatch between error types. It will reduce the effect of error mismatches by allowing second level check for the error type. First level is done by the error discriminator and second level check is done by the CWA algorithm using a new congestion window threshold called `tthresh`. This will prevent unnecessary congestions when the error discriminator mismatches errors. Also, CWA showed that it is able to keep same level of bottleneck load as TCP.

However, CWA performance depends on when transmission errors occur. Solutions for this problem will be discussed in the future work. Another problem is that due to the increase in the number of RTO evens and RTO durations the new algorithm performance decreases with the increase in the error rate. We solve the problem by adding a multiple drop action MDA [9] which reduces both number of RTO events and RTO durations. An important feature of CWA algorithm is that it does not require any change in the network or in the receiver (the client). Only TCP on the sender (the server) need to be changed. In present we are working on a complete set of algorithms which are combined will form a complete mechanism to govern TCP end-to-end error discriminators reaction for transmission drops.

References

[1] V. Jacobson. Congestion avoidance and control. In *Symposium proceedings on Communications architectures and protocols*, pages 314–329, Stanford, California, United States, 1988. ACM Press. Conference Paper.

[2] Van Jacobson. Modified tcp congestion avoidance algorithm. Technical Report 30, 1990.

[3] Saad Biaz and Xia Wang. Can ecn be used to differentiate congestion losses from wireless losses? *Technical Report CSSE04-04,Auburn University*, 2004.

[4] C. Lim. An adaptive end-to-end loss differentiation scheme for tcp over wired/wireless networks. *IJCSNS*, 7(3):72, 2007.

[5] Mahbub Hassan and Raj Jain. *High Performance TCP/IP Networking Concepts,Issues and Solutions*. Prentice Hall, 2005.

[6] Saad Biaz and Nitin H. Vaidya. De-randomizing congestion losses to improve tcp performance over wired-wireless networks. *IEEE/ACM Trans. Netw.*, 13(3):596–608, 2005.

[7] Guang Yang, Ren Wang, Mario Gerla, and M. Y. Sanadidi. Tcp bulk repeat. *Computer Communications*, 28(5):507–518, 2005.

[8] S. Biaz and N. H. Vaidya. Differentiated services: A new direction for distinguishing congestion losses from wireless losses. Technical report, University of Auburn, 2003. Report No. : CSSE03-02.

[9] M. Alnuem, J. Mellor, and R. Fretwell. Tcp multiple drop action for transmission errors. In *PGnet2008 The 9th Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting*, 2008.

[10] Song Cen, Pamela C. Cosman, and Geoffrey M. Voelker. End-to-end differentiation of congestion and wireless losses. *IEEE/ACM Trans. Netw.*, 11(5):703–717, 2003.