New algorithm to control TCP behavior over lossy links

M. Alnuem, J. Mellor and R. Fretwell
School of Informatics,
University of Bradford,
UK.

Abstract

TCP (Transmission Control Protocol) is one of the most used transport protocol in the Internet. Many widely used applications use TCP for sending data like the File Transfer Protocol (FTP), Telnet and Web-HTT connections. TCP was found to be performing poorly over lossy links when transmission (non-congestion) errors existed. This is due to the fact that TCP is unable to distinguish between congestion and transmission errors, and hence reduces its sending rate for all errors, assuming a congestion exists in the connection path. One solution is to discriminate between errors and deal with each error type differently. In this paper we present a new TCP congestion window cut algorithm to be used by any end-to-end TCP error discriminator when transmission errors occur. Instead of cutting the congestion window to half (like standard TCP) we delay the cut decision until TCP receive all duplicate acknowledgments for a given window of data (packets on flight). This will give TCP a clear image about the number of drops from this window. Then congestion window size is reduced only by number of dropped packets. Using this approach, the new algorithm manged to improve TCP performance in non-aggressive way by increasing the average congestion window size. On the other hand, this algorithm cut 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 discusses some limitations of the new algorithm.

1 Introduction

One technique to increase TCP performance over lossy networks is to use TCP error discriminators. Error discriminators are algorithms that can used to replace standard TCP congestion window mechanism[11][12] 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 distinguish between error types with high accuracy to avoid increasing the congestion on the 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. See for example [5, 13, 9, 4, 8, 6, 3].

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

- It will not increase the network congestion rate because of uncontrolled congestion window action by cutting the congestion window even for transmission errors.
- It will increase TCP performance in case of transmission drops by cutting the congestion window in a rate related to number of dropped packets instead of the duplicative decreases used in TCP.
- Also It will increase TCP performance in case of transmission drops by allowing TCP to recover from multiple drops and increasing TCP retransmission rate.

The main reason it has been required to have a high accurate error discriminator is to avoid increasing the load on already congested network because of mismatches between congestion and transmission (non-congestion) errors. So, if we can maintain an action that will not increase the congestion level in the network then even an error discriminator with medium/low accuracy will be able to increase TCP performance with no harm on the network. Following we will explain a proposal to congestion window cut policy that can be added to any error discriminator. This policy will help to improve TCP performance by reducing the congestion window cut rate and will reduces the effect of congestion/transmission mismatch on the network by using conservative approach toward transmission errors. This
algorithm is intended to work with end-to-end error discriminators in order to improve their reaction toward non-congestion drops. The new design holds following properties:

- It will work only if there is an error discriminator in the TCP sender.
- It will be used only when non-congestion drops occur.
- It will change TCP reaction to drops by delaying the congestion window adjustment after any drop until the whole window is received so TCP sender have more information about number of dropped packets.

Adaptive congestion window cut strategy and delaying congestion window cut until TCP receive the full window are the main contribution of this work. Following we will explain our design.

2 TCP response to drops

When a drop occurs mainly three systems are involved in TCP: the congestion window control system, the retransmission systems and the timeout system. Although in real implementation these systems combined in one algorithm, the differences in there roles allows us to divide them in such way. Following we will explain how Congestion window control works and how drops affect it.

2.1 Congestion window control

When TCP start sending data the congestion window start to grow exponentially during the slow start phase until first packet drop occur which indicates two things: first that the link capacity has been reached, second that the congestion avoidance phase has started. At this stage TCP congestion control mechanism takes the control. TCP congestion control mechanism has mainly two different and important jobs:

- one is to decide when to increase/decrease the congestion window size (the direction of the change).
- and another is to decide the amount of the increase/decrease in the congestion window (the amount of the change).

In order to decide the direction of the change TCP uses the packets drops as a signal that the congestion window should be decreased (downward direction) and the absence of the drops, and hence receiving of new acknowledgment, as a signal that the congestion window needs to be increased (upward direction).

TCP decides the amount of change (decrease/increase) based on the direction of that change by using AIMD mechanism (Additive increase Multiplicative decrease)[11]. During congestion avoidance phase TCP tends to reduce the sending rate by increasing the congestion window linearly (Additive increase) instead of exponentially as in slow start. TCP do this by increasing the congestion window with each new acknowledgment received according to following formula [11]:

\[
\text{new\_window\_size} = \text{old\_window\_size} + \frac{1}{\text{old\_window\_size}}
\]

This is equivalent to increasing the congestion window by one packet each round trip time [11]. In other words, TCP will wait until the whole window is sent safely (i.e. without any drop) and then it increases the window size by one packet. However, if a drop occurred TCP takes this as an indication that the new window exceeded the link capacity or that a new load has been introduced to the network (for example new users start downloading ftp files). In this case TCP will reduce congestion window size to half of its size before the drop occurred. This will reduce TCP performance in a multiplicative manner since the new window size will be 1/2,1/4,1/8 ..etc. of the original window size before the drop.

The reason for using multiplicative decrease after drops is to make sources to slow down quickly when a congestion occurs in order to give congested routers enough time to clear the congestion [10] also using this mechanism will make the connections with bigger window to cut more data (for example a connection with window size of 100 packets will cut 50 packets while a connection with 10 packets window will cut 5 packets only) which will help to resolve the congestion faster. On the other hand, additive increase is used in case of absence of drops because it helps TCP to explore the link capacity in a gentle way also it maintains fairness by giving different flows the same chance to increase sending rate equally [10].

3 New congestion window cut policy for transmission errors

Our aim is to have an efficient congestion window action in case of transmission error that will increase the TCP performance in case of transmission errors and at the same time will prevent congestion from happening by cutting the congestion window. In this section we will discuss a new congestion window cut policy for transmission errors.

When a drop occur TCP wait until it receive 3 duplicate acknowledgments and then it cut the congestion widow size to half of its original size (multiplicative decrease). Our proposal is that instead of cutting the congestion window after
receiving 3 duplicate acknowledgments we delay the cut decision until TCP receive all duplicate acknowledgment for current window (All duplicate acknowledgments will be received after TCP receives the first new acknowledgment after the drop). The Duplicate acknowledgment indicates a packet drop but also indicate how many packets has left the network (i.e. received by the other end). Using this information we can estimate how many packet were dropped per window (dropped packets= Window size- Number of Dups). In case of transmission drops, instead of cutting the congestion window to half of original size TCP cut only number of equals to number of dropped packets. Then the new cwnd size is reduced by number of dropped packets only. This is done because the drops are not caused by congestion but by a link failure. However, although TCP has received indication the these packet were dropped they actually may be still in the network in the link layer buffer of the lossy link which try to retransmit these packets (An example of these link layer protocols is Snoop protocol[2]) and since all other packets were drained from the network (this is confirmed by receiving duplicate acknowledgment) then TCP reduces cwnd by the number of dropped packets only. This will allow link layer buffers to clear more quickly. Using this policy TCP does not use multiplicative decrease for transmission errors, instead it cut the congestion window in a rate related to number of dropped packets.

Moreover, since this action will be integrated in an error discriminator, we will define another measure to decrease the rate of congestion error mismatch and to increase the error discriminator accuracy. As we know, cwnd (congestion window size) and sshresh (slow start threshold) are the main two variable controlling congestion avoidance mechanism in TCP. The cwnd variable hold the current window size and controls the sending rate. The sshresh defines the boundary between the slow start phase (exponential increase of cwnd) and the congestion avoidance phase (linear increase of cwnd).

We will define another threshold we call it Transmission drops threshold (tthresh). This will define the area (in term of congestion window sizes) between the start of congestion avoidance phase (i.e. from sshresh) and to the first drop occur. Since this is the first drops between the sshresh and the cwnd then we set tthresh to cwnd. This means that any dropped packet diagnosed by the error discriminator as a transmission drop and at the same time the cwnd ≤ tthresh then it is probability a correct diagnosis.

Based on the previous discussion, following we will explain the new policy to cut the congestion window based on tthresh variable:

1. As we said before the aim of tthresh is to define the cwnd size where the first drop happened and set tthresh = cwnd. So no matter what the accuracy of the error discriminator we know that before tthresh there was no congestion that can cause drops.

2. Now if a drop happened and the error discriminator described it as a congestion drop then TCP does normal congestion control by cutting the cwnd to half. Also TCP check if cwnd ≤ tthresh and if so then TCP lower the tthresh by making tthresh = cwnd, otherwise does nothing. From that we can see that tthresh only decreases and does not increase after first initialization in step 1.

3. However, if the drop is diagnosed as transmission drops by the error discriminator then TCP cut the congestion window but according to following:

   a) If the cwnd ≤ tthresh then there is a big chance that the error discriminator diagnoses is correct and the drops is actually a transmission error so we still cut the congestion window but according to new strategy. First we calculate the number of dropped packets using the duplicate acknowledgments and the window size as following:

   i. First we calculate number of packets sent from this window and we call it flight_size as following:

   \[ \text{flight}_\text{size} = \text{lastsent} - \text{lastack} \] (1)

   where lastack is a variable stores the sequence number of the last acknowledged packet in this window and lastsent is the sequence number of the last sent packet.

   ii. Then we calculate the number of dropped packets as following:

   \[ \text{num}_\text{drops} = \text{flight}_\text{size} - \text{num}_\text{dup} \] (2)

   where num_dup is the number of duplicate acknowledgment we received after the first drop is occurred.

   b) If cwnd > tthresh then TCP cut the congestion window to half as it does to congestion errors. We do that because when cwnd above tthresh there is a higher chance that the error discriminator has mismatched congestion drops for transmission drop. This action is conservative but it will prevent any unnecessary congestions.

4. After each timeout we calculate tthresh again (i.e. go to step 1)

We will call this algorithm Congestion Window Action (CWA). The algorithm will take place during the congestion
avoidance phase only. Any drop during slow start phase will be considered congestion drop. We change the way TCP respond to duplicate acknowledgments. TCP decrease the congestion window after receiving three duplicate acknowledgments. However, in this algorithm TCP waits until it receive all duplicate acknowledgment and then decide on the way to decrease the congestion window based on drop information from the duplicate acknowledgments. This will help TCP to make a cut decision more related to the network status.

Moreover, one merit of this algorithm is that it allows cwnd to be reduced even for transmission errors which will prevent congestion caused by error discrimination mismatch, at the same time it defines a new reduction policy based on the number of packets expected to be still buffered in the network for retransmission. Also it defines a two level check of error type by using a threshold (tthresh) that defines the area between error-free and first error and then an error discriminator diagnosis of the error type. This two level mechanism will reduce the effect of error mismatch because of low accuracy in error discrimination.

Limitation: Since tthresh records the congestion window size when first drop occur, the performance of the algorithm will depend on when errors occur after a timeout. If errors keep occurring early enough, the performance will be like normal TCP. This scenario could happen on high error rates as we will see in the result section later.

4 Performance of CWA

In order to measure the improvement using CWA we will measure TCP goodput (the actual amount of data received regardless of retransmissions) after and before adding CWA to TCP. Also we will measure number of average congestion window size during the connection lifetime. 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 [14]. 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[6][7][14] to simulate wireless errors.

The chart in figure 1 shows that after adding CWA TCP 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 2 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 events. Reducing number of RTO events will reduce the total time TCP stays idle and hence will increase the throughput. Figure 3 shows the improvement in number of RTO events after using CWA. The increase in conges-
Figure 4. CWA- log scale congestion window size and number of RTO

Congestion window size has reduced 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.

Also when we compare the rate of the decrease in the congestion window we found it consistent with rate of increase in number of RTO as we can see in Figure 4 which indicates the direct effect of congestion window size on the RTO events. 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 \( t\text{thresh} \) after first drops). However, since transmission errors are usually random (like wireless errors for example) the probability to have error early in the connection will depend totally on the error rate. This explains why with the increase in the error rate the performance of CWA become closer to slandered TCP performance. However, with the increase in the transmission error rate the differentiation between congestion and wireless losses will become harder for a delay based error discriminator (which we are using at the moment) since the packet delay may increase due to the high drop rate and hence it become harder to differentiate between the increase in delay because of congestion and of link failure and hence the error discriminator may give wrong results. So in this case it is better to reduce CWA performance to prevent increasing the network congestion. One solution to this problem is to delay assigning \( t\text{thresh} \) to congestion window with the increase in the transmission error rate, so instead of first drop we take second or third drop as when we assign \( t\text{thresh} \). More, solutions for this problem will be investigated in the future work. 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[1].

5 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 [9] and MDA [1] 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 5 shows the results. Moreover, the new error discrimina-
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. All can suffer from congestion errors. Cross traffic is used to create the required level of congestion drops by varying the UDP sources sending rate.

6 Conclusion and Future work

In this paper we present 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 $t_{thresh}$. This will prevent unnecessary congestions when the error discriminator mismatches errors (in our experiments both TCP and CWA had the same average bottleneck queue size). 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 events 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 [1] 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. For future work 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 towards transmission drops.

References