TCP Multiple drop action for transmission errors

M. Alnuem, J. Mellor and R. Fretwell
School of Informatics, University of Bradford
{m.alnuem, j.e.mellor, R.J.Fretwell}@bradford.ac.uk

Abstract—In this paper we present a new TCP retransmission algorithm to recover from multiple non-congestion (transmission) packet drops from the same TCP window. This algorithm will help end-to-end error discriminators to improve TCP performance over lossy and heterogeneous networks. The simulation results show that the proposed algorithm has increased TCP performance over different error rates without extra load on the network. This has been achieved by resending only number of packets equal to the dropped packets which have already left the network and by cutting the congestion window even for non-congestion drops. This algorithm will work only if there is an error discriminator in the network then even an error discriminator with medium/low accuracy will be able to increase TCP performance. However, if a single non-congestion drop is discovered then TCP will resend it and will not cut the congestion window. Moreover if multiple drops occur at the same time in a raw 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 [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11]. Hence, the main aim was to increase the accuracy of the error discriminator as much as possible.

However, other problems like action toward multiple drops caused by transmission (non-congestion) errors has not been fully addressed in the field of end-to-end error discrimination.

We will look at the same problem but from different angle so that 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 increase TCP performance in case of transmission drops by allowing TCP to recover from multiple drops and increasing TCP retransmission rate.
• It will not increase the network congestion rate because of uncontrolled congestion window action.
• It will work only if there is an error discriminator in the network.
• 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.
• If a single non-congestion drop is discovered then TCP will resend it and will not cut the congestion window.
• If more than one congestion drop are discovered then TCP will use an adaptive retransmission strategy based on the number of dropped packets per window as we will explain in this work.

Adaptive retransmission 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 but first we will start by a look at standard TCP response to drops, then we will explain our view of a proper response.

II. MICROSCOPIC LOOK AT TCP RESPONSE TO DROPS
TCP uses packet drops as an indication of reaching link limits. If a packet is dropped TCP infers that the link capacity has been reached and it stops increasing its sending window. Moreover if multiple drops occur at the same time in a raw transmission (non-congestion) errors correctly then TCP suppose there is congestion in the connection path and reduces its sending rate (by reducing its congestion window) in order to allow the congestion to be resolved. However, if the congestion continues to exist TCP will continue cutting its sending rate until it reaches its minimum (usually one segment per round trip time). The task of monitoring packet drops and reacting accordingly is done by TCP congestion control mechanism.

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 sign that the congestion window should be decreased (downward direction) and the absence of the drops as a sign that the congestion window needs to be increases (upward direction).

TCP decides the amount of change based on the direction of that change by using AIMD mechanism (Additive increase Multiplicative decrease)[13]. The reason for using multiplicative decrease in case of 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 [14] also using this mechanism will make the connections with bigger windows to cut more data (for example if TCP cut congestion window by half after each drop then a connection with window size of 100 packets will cut 50 packets while a connection with 5 packets window will cut 5 packets only) which will help to resolve the congestion faster. Additive increase is used in case of absence of drops because it helps TCP to explore the link capacity in a gentle way (compared to multiplicative increase) in order to avoid oscillation between no activity and congestion[14] which can occur if aggressive multiplicative increase is used for example (i.e. if TCP increases the rate quickly it could case congestion).

However, if a mix of congestion drops and transmission drops coexist then it is no longer possible to use packet drops to decide the link capacity unless we can discriminate error types to a good extent and hence respond to each error type in a different manner.

TCP uses two ways to detect drops, first: receiving three consecutive duplicate acknowledgments. A duplicate acknowledgment is a way TCP-receiver uses to say that a packet is missing. For example if TCP sender send six packets to a TCP receiver, assuming all packets take the same path, and packet number three has been dropped by a congested router in the connection path. TCP guarantee data delivery by making the receiver to send acknowledgment packet for each received packet. In our example when the receiver receives the first two packets it will send acknowledgment for packet one and two. However packet three is missing and packet four is received instead. When the receiver receives an out of order packet (i.e. packet four) it will not acknowledge it but instead it will resend the acknowledgment of last in-order packet which is packet two in our example. This acknowledgment is called duplicate acknowledgment since it is the second acknowledgment for packet two (First one was sent when packet two was received). The receiver will continue to sent duplicate acknowledgments with every out of order packet it receives (i.e. packets 5&6) until the sender resends the lost packet.

At the sender side, when TCP sender receives a duplicate acknowledgment it concludes there is a drop. However, since packets could be received out of order even if there was no drop and hence duplicate acknowledgments can be received in this case, TCP sender waits until it receives three consecutive duplicate acknowledgments before deciding there is a drop. The number three has been chosen by TCP congestion control designers [15] and has been accepted as a slandered. However, choosing the number of duplicate acknowledgments for deciding drops will depend heavily on the network topology and routing techniques used in the network and it is out of the scope of our paper. When the sender receives three duplicate acknowledgments it does three actions:

- First it slowdown the sending rate by reducing the congestion window size to half of its size before the drop.
- Second it resends the lost packet.
- Third it will increase the congestion window size by one segment for every duplicate acknowledgment received until a new acknowledgment is received. The logic behind this is that since a duplicate acknowledgment tells us that one packet (even if it is out of order) has been received by TCP-receiver and hence it left the network then it is safe to put another packet in the network in its place and hence TCP increases its congestion window with every duplicate acknowledgment[12].

The first two a actions are called Fast-Retransmission. Third action is called Fast-Recovery[12], [16].

However, if another drop occurred within the same window boundaries (i.e. another drop occur from the same set of packets in flight) and TCP received another three duplicate acknowledgment it will not respond the same way as explained above, instead TCP will resend the first dropped packet and then increase congestion window by one for every duplicate acknowledgment it receives. This is done to avoid unnecessary multiple retransmission of the same window of data because of unnecessary invocation of fast retransmission actions which may happen when TCP resend the whole window after a drops. This may cause multiple duplicate acknowledgment because TCP has resend packets that already been received successfully by the receiver [17]. These duplicate acknowledgment will trigger resending the same window multiple times.

However, this solution has a disadvantage that if a burst of packets were dropped then only the first packet will be treated by fast retransmission and the rest will trigger timeout which will throttle TCP sending rate severally. In our work we will advise a modification to TCP Fast retransmission in a way that allows TCP to slow down and to resend all lost packets at the same time and reducing unnecessary retransmissions as much as possible. This will help to improve TCP performance specially with transmission errors where multiple packet drop can occur for several reasons like mobile base station hand off or temporary signal fading on wireless networks.

A complete explanation of TCP response to errors can be found in [18], [12], [16].

A. Related Work

When TCP sends data it pass through Slow start phase and then Congestion avoidance phase. TCP 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. Our concern is the congestion avoidance phase because it is the most important phase in TCP life time since it compose most of the connection life.
time and it represents the equilibrium state of the connection life (The equilibrium is defined as: the state when TCP sender puts a packet into the network another packet from the same connection leaves the network at the receiver side[19]). Moreover, most of the effect of transmission errors will take place in this phase since it is the longest phase.

During congestion avoidance phase TCP tends to reduce the sending rate by increasing the congestion window linearly instead of exponentially as in slow start. TCP does this by increasing the congestion window with each new acknowledgment received according to following formula [16]:

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

This is equivalent to increasing the congestion window by one packet each round trip time. 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 (i.e. the new window size will be 1/2, 1/4, 1/8 ..etc. of the original window size before the drop since with each drop TCP multiply the current window size by 1/2).

When the first drop occurs, which indicated by receiving three duplicate acknowledgments, TCP enters Fast Retransmission and waits until it receives a new acknowledgment that acknowledge all sent data.

The problem here is that a new acknowledgment can be received which does not acknowledge all sent data but instead part of the sent data because there was multiple packet drops (This acknowledgment is called partial acknowledgment[20]). However, TCP does not recognize that and increases the window size as it does with any new acknowledgments. TCP then will wait to receive acknowledgment for the lost packets, which will not happen, and eventually will timeout.

From this we can see that if more than one packet were dropped from the same window TCP will resend the first drop only and then wait until a timeout occur. This timeout will trigger the resend of the rest of the dropped packets. However, this will cost TCP valuable time in two ways: first the time spent while TCP is idle without sending any data until the timeout occurred. Second after any timeout TCP reduces its congestion window to the minimum (usually 2 segments).

So the problem is that TCP resend only the first dropped packet in each window and waits for the rest to be recovered by retransmission timeout (RTO).

Multiple congestion drops can occur because of bursty behavior of TCP or because of sever congestion. To solve this problem for congestion drops the authors in [20] proposed a change on fast retransmission mechanism in TCP in order to make TCP to resend all dropped packets from the same widow one by one, they call it TCP-NewReno[20]. This method allow TCP to stay in fast retransmission until all lost packets are retransmitted. However, the problem here is that TCP only resend one per RTT because it resends one packet for each partial acknowledgment it receives. So if N packets dropped per window TCP will need N RTTs to recover that window.

Another solution proposed by [21] is to resend all outstanding data after TCP receives first partial acknowledgment, they call it TCP Bulk Repeat[21]. This is based on the assumption that if more than one packet is dropped from the same window then probably more packets were dropped also from the same window and hence it is better to resend the whole window. However, although this solution will cause TCP to recover more quickly in heavy loss networks, it will increase the resending rate largely in networks with light/medium losses specially when using large TCP windows. For example if the window size is 200 packets and only first two packets were dropped then TCP will resend the rest of the window (198 packets) unnecessarily.

B. Recovering from multiple drops

The previous solutions were proposed to recover mainly from multiple congestion drops. However, we will propose an extension to these solutions in order recover from multiple transmission drops from the same window based on the network conditions and estimated error rate. The main aim is to reduce the probability of TCP falling in idle state because of retransmission timeout. This is important specially in case of transmission errors since TCP will wait while actually there is no congestion.

In our algorithm TCP will use the set of duplicate acknowledgments which has been received after the first drop to estimate the number of dropped packets per window. Then it compute the number of dropped packets by comparing number of duplicate acknowledgments with the number of actual packets sent from that window. This will give us how many packets were dropped from this window. Then following [21][22, P540] assumption that transmission drops usually happens in bursts (set of consecutive packet dropped at once) we assume that the missing number of packets are consecutive. So after first partial acknowledgment we do not send one packet like NewReno[20] or the whole window like TCP Bulk Repeat[21], instead we send only the number of drops we calculated. This way if the packets dropped were consecutive then we may recover the whole window in one RTT with no unnecessary retransmissions. However, if there are more dropped packets in this window or the dropped packets were scattered then we will propose a new retransmission mechanism which can send them one by one as NewReno or resend the rest of the window as Bulk Repeat based on estimation of the network error rate.

So the modified fast retransmission and fast recovery algorithms will do following:

- First TCP recored the sequence number of the highest packet sent so far in a variable called lastsent[ns2 [23] implementation of TCP use variable name recover for the same purpose, we think the name lastsent is much clear).
Using variable lastsent TCP can check if this drops is the first drop in this window (current window is defined as the one between the first duplicate acknowledgment and the maximum sequence number sent when first duplicate acknowledgment is received, on other words they are the packets that have not been acknowledged yet)

When a drop occur then if this drop is the first drop in this window then we follow standard TCP action as following

1) Reduce the congestion window size to half of current size assuming the cause of the drop is a congestion and hence the logical action to take is to slowdown by reducing sender window size to half of its original size before the drop following standard TCP.
2) set slow start threshold ssthresh to half of original congestion window size.
3) resend the lost packet.
4) rest retransmission timer in order to allow more time for the retransmitted packet to reach the receiver before a timeout occur.

TCP then enters fast recovery and waits for a new acknowledgment to arrive.

When a new acknowledgment arrives TCP compares the received acknowledgment sequence number with the value saved in lastsent, If the current acknowledgment \( \geq \) lastsent then this acknowledgment acknowledges all outstanding data so we can exit fast recovery safely.

However if the received acknowledgment < lastsent then it is a partial acknowledgment so TCP does following

1) First TCP calculate the number of packets sent from this window and we call it flight size as following :

\[
flight\_size = lastsent - lastack
\]  
(1)

where lastack is the sequence number of the last acknowledged packet before the partial acknowledgment arrived.
2) Then we calculate the number of dropped packets as following:

\[
num\_drops = flight\_size - num\_dup
\]  
(2)

where num_dup is the number of duplicate acknowledgment we received after the first drop has occurred.
3) TCP then resend number of packets equal to num_drops
4) If TCP received a new acknowledgment that acknowledge all outstanding data then it can exit fast recovery.
5) If the received acknowledgment is a partial acknowledgment then we have two options, either resent one packet and wait a full RTT for each subsequent drop to be received or we can send the rest of the window. In the next section we will use an adaptive method to choose between the two method based on error rate estimation.

The main benefit of this algorithm is that it increase TCP performance when the error is diagnosed as transmission error by increasing the resending rate of lost packets. Also it allow TCP to resend all lost packet from the same window and this will prevent timeouts specially when a burst of packets were dropped on the lossy link.

This algorithm will allow error discriminator even with low accuracy to gain improvement and to prevent monopolizing the network at the same time because it slowdown for congestion and for transmission errors (previous solutions slow down for congestion errors only).

We will call this algorithm multiple drops action for TCP (MDA).

C. MDA action

1) Motivation: The aim of the multiple drop action (MDA) is to allow TCP to resend multiple dropped packets from the same window of data. TCP sends only the first dropped packet and leave the rest to be recovered by retransmission timeout mechanism after a waiting period of time. When a retransmission timeout occur TCP cut the congestion window to its initial value (usually one/two segments) and resends all the rest of dropped packets (Mass retransmission).

The problem with this method is that when a multiple transmission drops occur TCP will be forced to be idle for long periods of time before the whole window is recovered. In case of congestion drops this behavior is acceptable because in case of sever congestion this waiting time will allow the network buffers to be drained before TCP start sending again. However, in case of transmission error there is no congestion to drain from the network and hence no need to delay packet retransmission. However, in some cases the transmission errors are persistent like the in case of long link failure and there is no point of keeping retransmit the lost packet so we need to wait event for transmission errors, for this reason we retransmit the lost packets only once per window and then we allow retransmission timeout to occur between windows

2) Results: The MDA action is concern about reducing number of timeouts TCP fall into by resending all dropped packets from the same window.

MDA does this by using information from the the duplicate acknowledgment it receives per window. Any duplicate acknowledgment is an indication that a packet has been dropped.
However, if TCP knows how many packets has been sent and compare it with the number of duplicate acknowledgment it can know how many packets were dropped in this window. Using these information we advice a new method to resend multiple packet drops by first resending only number of packets equal to number of dropped packets. This will help to recover the window very quickly (1 RTT) in case if the dropped packets were in sequence (burst drops) as we explained before. However, we also cover the case when the packets dropped are not in sequence and are scattered among the window. In this case we have two options, either to follow TCP-Newreno [20] and to resend the dropped packets one by one which will take long time if we have too many dropped packets but it will prevent multiple retransmission of already received packets. Other method is to resend all lost packets as Bulk Repeat [21] which is good if we have too many drops per window but if the error rate is low it will cause a lot of unnecessary retransmission of packets. So we will apply a method that make use of the number of dropped packets to compute dropping rate per window and then choose between the two previous methods based on that. We will use a threshold \( \alpha \) which represent the error rate, so that if drops rate per window is lower than \( \alpha \) then we use slower but more conservative method of resending packet per RTT since the number of errors is low. However, if the number of errors per window exceeded \( \alpha \) then TCP can resends all window at once without having problem of many unnecessary retransmissions since the number of drops is high. Choosing \( \alpha \) depends on the network operator needs and the state of the lossy link. If the connection has high rate of errors then it is better to choose lower values for alpha to allow TCP to do more retransmission. However, if the drops are rate then a higher value for \( \alpha \) is preferable to prevent TCP from retransmitting unnecessary packets. In our simulation we set \( \alpha \) to 5% because it is considered around high error rate as in [5] so TCP use the bulk repeat only in high error rates. In order to measure the improvement using MDA we will measure TCP Goodput (the actual throughput) after and before adding MDA. Also we will measure number of timeout events during the connection life time for the two connections.

The topology used in presented in figure 1. The chart in figure 2 shows that after adding MDA TCP performance has improved. This improvement is caused by MDA preventing multiple drops from causing unnecessary timeout in case of transmission drops. However, due to the fact that the retransmission process itself will consume some of the bandwidth the MDA throughput decreases with the increase of the error rate. Moreover, even with multiple drops retransmission timeouts are some time expected to occur specially with higher error rate where the same packet is dropped more than once and as we explain before we resend the packet only once per window to prevent unnecessarily retransmission in case if the connection will be dead for long time. Figure 3 shows number of timeouts occurred during the connection life time of TCP and MDA. As we can see the number of timeouts in case of MDA is much less than in TCP. However, we can also see that the number of timeouts decreases in case of TCP. The decrease of number of timeout in case of TCP is natural because with the increase in the error rate the connection dries quickly and the number of sent packets decreases and hence number of timeouts. On the other hand in case of MDA number of timeouts increase until it reaches 2% and start to decrease with the increase of the error rate. This shows that MDA actually works well specially with low error rates since it decreases the number of timeouts by resending dropped packets before timeout occur which reduces the number of timeouts in case of 1%. The same thing happens in case of 2% error rate but this time the number of timeouts increased because of the increase...
in the number of retransmitted packets (around 8000 packets were retransmitted under 1% error rate while 10410 packets in case of 2%) these extra packets can trigger more timeout events. However, the increase in error rate will cause number of timeouts to decrease again because of the fact that we resend each packets only once per window. Moreover, the average duration of each timeout increases with the increase in the error rate (due to TCP back off policy) and this explains why MDA performance decrease with higher error rates. Figure 4 shows a comparison between number of timeout events and duration of each event and as we can see the duration increases with the increase in the error rate. In all of our experiment we repeat the experiment until we reach 99% confidence interval with maximum ratio of 5% from the average value. So in all graphs only the average value is plotted.

D. Conclusion and Future work

Most of the Internet content is delivered using TCP [24] and with the advances in wireless and mobile networks, improving TCP performance over heterogeneous networks like wired-cum-wireless networks has been an important issue. With these advances, the end user expects the same level of advance in the performance or at least the same level of service provided by a fixed machine. However, Transmission Control Protocol (TCP) was found to perform badly over networks with wireless links especially when wireless errors are high.

One technique to increase TCP performance over lossy networks is to use TCP error discriminators. Error discriminators are techniques that can be impeded in TCP and try to understand the cause of the packet drops and then make TCP to act differently for each type of error. In case of congestion error, TCP slows down by cutting congestion window size. In our research we want to study the best action TCP can take in case of non-congestion errors. When TCP deals with non congestion errors, there should be a balance between increasing the congestion window and the load on the network. Uncontrolled increase in the congestion window could lead to undesirable congestions on the network. This algorithm is one of three algorithm that we are developing in a work to create a proper action toward transmission errors.

In this paper we have implemented an adaptive approach to recover from multiple transmission drops from the same window. Experiment results shows that this approach has increase the performance of TCP without extra load on the network. We achieved that by resending only number of packets equal to the dropped packets which have already left the network. Also we cut the congestion window even for non-congestion drops. This also will be part of 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.

All these actions will help even error discriminators with low/medium accuracy to improve TCP performance with no harm to the network.

References