Explicit Congestion Notification for Error Discrimination

A practical approach to Improve TCP performance over wireless networks

M. A. Alnuem, J. E. Mellor, R. J. Fretwell
Mobile Computing, Networks and Security Research Group
School of Informatics, University of Bradford
Bradford, UK
{M.Alnuem, J.E.Mellor, R.J.Fretwell}@bradford.ac.uk

Abstract

Explicit Congestion Notifications are used to notify TCP when imminent congestion is about to happen. In this work we use this feature to design an error discriminator that can be added to a TCP sender to differentiate between congestion drops and drops because of wireless link failure. By distinguishing between error types, TCP can avoid unnecessary cuts of congestion window when non congestion error happens and when the retransmission of lost packets is enough and there is no need to slow down. A set of experiments were conducted to validate this solution. The results show that ECN is a good predictor of imminent congestion and can be used actively in an error discriminator over networks that suffer from low and medium congestions.

Keywords: performance evaluation, error discrimination, wireless networks, Explicit Congestion Notification, TCP.

1. INTRODUCTION

TCP responds to congestions drops by reducing its transmission rate. However, over wireless links, errors may occur without congestion like the errors caused by link failure. In this case TCP does not need to reduce its transmission rate because there is no actual congestion. Since TCP respond to all errors by reducing its transmission rate, distinguishing between link errors and congestion errors is expected to improve TCP performance over error prone links like wireless links.

Dawkins et al. in [1] has proposed the use of Explicit Congestion Notification (ECN) to discriminate between congestion and link errors. Biaz [2] explains the technique as following: If a drop is detected by receiving duplicate acknowledgments, then TCP check if an ECN is received in the near past. If ECN is received before the error happened then this is a strong indication that this error is caused by congestion. This is based on the understanding that, in ECN-capable connections, ECN should always happen before congestion drops. So, if ECN came before the drop then TCP consider this drop as congestion drop and act by reducing sender’s window size in order to solve the congestion. However, if the drop happens while not preceded by ECN, it then considered as wireless error and so no need to reduce the sender’s window size [2]. However, we still need to retransmit the lost packet. The authors in [1] argue this approach will improve TCP performance over wireless networks specially those suffers from high wireless error rates.

However, Biaz in [2] studied the visibility of using ECN to distinguish between error types. He showed that the probability of ECN marked packets will come before a congestion error or a wireless errors is approximately the same. He concluded that, because ECN happens before congestion and wireless errors at the same rate, ECN can not be used to differentiate between error types and hence using ECN as described by Dawkins et al. in [1] will not improve TCP performance. However, in this paper we discuss Biaz [2] argument and propose extension to it.

2. ERROR DISCRIMINATION UNDER HIGH CONGESTION RATES

As we said before, Biaz in [2] argue that we can not use ECN to improve TCP performance over wireless networks to discriminate between errors.

However, let us discuss this argument by making some questions and try to answer them. First, if the probability of having ECN before congestion as well as before wireless errors is the same, does this means that the congestion error is high? To answer this question we will take a mechanism that uses ECN to mark packets, Random Early Detection
active queuing with forward ECN. RED with ECN capabilities uses average queue size and a threshold to mark packets with ECN flag. If the average queue size is between the minimum and maximum threshold then RED marks the packets according to probability P. However, if the average queue size exceeded the maximum threshold then RED marks all packets. Following is summary of the procedure as presented in Floyd[3]:

For each received packet:

If min-thresh<=avg< max-thresh then mark packet according to probability P

Else

If avg >= max-thresh then mark the packet.

Avg is the average queue size, min-thresh and max-thresh are the minimum and maximum thresholds respectively.

We can see from the algorithm that congestion drops will occur when the average size exceeded the max-threshold. Hence all packets will be marked during this period. If this persists then the only solution is to slow down TCP’s transmission rate. We have notice from experimental observation that with the increase in the congestion error rates the number of ECN notification increases; hence the probability of having ECN before wireless errors as well will increase. This leads us to the next question: Do we need to discriminate between errors if congestion rate is high?. We believe that under high congestion rates there is no point of discriminating between errors since the correct action to take in this case is to reduce the transmission rate in order to solve the congestion.

However, when the congestion rate is low then the probability of having ECN before wireless error will reduce gradually specially with the increase in the wireless error rates. Hence, if we use error discriminator based on ECN it will give positive results especially with the increase in the error rates.

We can conclude the following: We mainly have two cases; one case is when the wireless link operates on a network with light or no congestion. In this case we believe that using ECN to discriminate error types will improve the performance since the number of wireless errors will be much higher than generated ECNs. In this case mistaking wireless error to a congestion error will be rare and hence the unnecessary cuts of TCP window will be small.

The other case is when wireless links operates with a heavily congested wired network. We believe in this case there is no point of discriminating errors (using ECN or any other method) because even if we could discriminate between errors, the priority is to solve the congestion and this can only be done by reducing TCP transmission rate. So we are going to cut the sender’s window any way.

In this work we will design a sender based TCP error discriminator that use ECN to distinguish error types. We are going to test the performance of the new protocol in presence of different wireless error rates and in network with light or no congestion.

3. Related Work

TCP-Error discriminator is the name of all methods that try to understand the cause of the packet drop/corruption and to make TCP act differently according to each type of error.

Some error discriminators deal with the network as a black box and they do not use any explicit feed back from the network in order to discriminate errors. Others use help from intermediate nodes in order to understand the cause of the error. Following we will talk about both types.

A. Network dependent error discriminators

This kind is based at the end-point of the connection but uses help from the intermediate nodes. For example TCP-Casablanca and TCP-Ifrane by Biaz et al. [4]. In TCP-Casablanca the sender marks each outgoing packet with either out or in mark according to a certain probability. Then a biased queuing mechanism is placed in the congested nodes so it will drop only the packets marked with out mark. This way when a drop occur the receiver will figure out if it is a congestion or wireless drop by looking at the pattern of marking in the received. The authors in [4] has indicated that Casablanca discriminator has achieved high improvement in TCP performance (above 100%) [4]. TCP-Ifrane [4] is a simplified version of TCP-Casablanca where the sender monitor the packet drop patterns instead of the receiver as in TCP-Casablanca. TCP-Ifrane did give slightly higher results than TCP-Casablanca when tested in [4].

Also Biaz et al. in [5] has used information provided by intermediate nods, like partial acknowledgments provided by Snoop agents[6], to help error discriminator at the end-point to distinguish errors types see [5]. Another example is the use of congestion notifications provided by the congested nodes (ECN) as the idea by [1] which was tested in [2] as well as in this paper.

B. Network independent error discriminators

This kind is based at the end hosts (or one of them) and does not require help from the intermediate nodes. These solutions use implicit indication of network congestions.
For example round trip time like in [7, 8] or throughput like in [9]. The error discrimination works by taking input from the congestion predictor about the congestion status. If a loss occurs then if the congestion predictor was predicting a coming congestion then the drop is considered as congestion loss. However if the predictor was suggesting increasing the sending rate because it does not predicts any congestion in the near future then the drop is considered to be caused by link error [10]. Also we must notice that as [10] indicated, designing accurate error predictors is important since mistakes of distinguishing transmission errors from congestion errors could case unnecessary congestion which is usually avoidable by using normal congestion control algorithms. For example if a congestion error is mistaken by the congestion predictor to be as transmission error then TCP will not decrease the window size and this will make the current congestion much worse.

4. TCP-ECN AN ERROR DISCRIMINATOR USING EXPLICATE CONGESTION NOTIFICATIONS

Dawkins [1] has proposed the use of ECN to create an error discriminator. However, in [1] they did not provide a detailed design of the idea. Here we will describe the design and testing of a sender based TCP error discriminator that uses ECN to discriminate between error types. The idea we propose is to give TCP the ability to discriminate between wireless and congestion errors by using feedback from an Active Queue management mechanism (AQM) at the congested nodes which can mark packets when congestion is happening or about to happen. When error occurs, an error discriminator at TCP sender will decide if the error is congestion error or wireless error by looking at the feedback received from the AQM mechanism. In our case we will use a queuing mechanism to mark packets when a congestion is about to occur, Random early Detection which proposed by Floyd in [3] and they call it RED.

So, if a congestion is taking place then RED will mark packets with ECN so that the error is considered a congestion error and hence, TCP reduces its transmission rate. On the other hand, if RED did not mark packets because there is no congestion and an error happened, then the error considered to be wireless error and the sender resend the lost packet without the need to reduce it’s transmission rate.

The motivation behind this mechanism is that it preserves the end-to-end semantic of TCP because the connection between the sender and receiver is not broken at any point of the path. Also, it uses already used and tested AQM mechanism (i.e. RED[3]) and with the increase deployment of RED in computer networks, this solution will not need any change in the network (changes will be needed only at the TCP sender). For this reason this solutions will need minimum changes on the existing systems and it gives a noticeable improvement of TCP performance over wireless networks as the experiment results show in the next section. Moreover, it is able also to deal with congestions, just like normal TCP, and reduces TCP transmission rate if necessary.

Following we will describe the work of the error discriminator but first we will put some assumptions:

- In order for this mechanism to work there should be no congestion drops without having ECN sent to TCP.
- In case of RED there will be no dropping, instead RED will send ECN whenever it expect that a congestion is about to occur. Having RED to mark packet instead of dropping them is one variation of RED as explained by Floyd in [3]
- If there are congestion errors as well as wireless errors then TCP should solve the congestion first by dropping transmission rate (Congestion is given higher priority).
- RED is required at the bottleneck nodes only.

The TCP implementation used is TCP-Reno. We modify TCP-Reno so it can discriminate between congestion and wireless errors based on the ECN feedback from RED. Figure 1 shows the design of the error discriminator. The changes are done to sender side only and it works as following:

![Figure 1. TCP Error Discriminator](image)
• If a packet is received by TCP sender, the discrimination module will read it first. If the received packet is a duplicate acknowledgment then we see: if the packet is marked by RED with an ECN or if we have received a marked packet recently (normally two previous marked packets is enough) then we consider it to be an indication of a congestion error. In this case we pass the control to TCP-Reno congestion control mechanism.

• If we received a duplicate acknowledgment but with no marked packets in the recent past, then it is considered as a wireless error. In this case we do not call congestion control; instead, we do retransmission of the lost packet and reset the retransmission timeout (RTO).

• In case of timeout we pass the control to the TCP-Reno congestion control.

• If a new acknowledgment is received with an ECN mark, then TCP reduces its transmission rate to prevent expected congestion.

5. MULTIPLE LOSSES FROM THE SAME WINDOW

In our work we have implemented and applied a simple version of the Idea suggested by Floyd et al. in [11] to recover from multiple packet drops. To do that TCP remain in the fast recovery until a new acknowledgment that acknowledge all outstanding packets is received. During that we increases its congestion window with every duplicate acknowledgment it receives. Also TCP resends the last acknowledge packet +1.

Using this approach we have noticed a big improvement in TCP performance particularly in high bit error rates.

In order to limit the congestion created by increasing retransmission we limited the resent of lost packets to one. This way each lost packet will be resent only once before a timeout happens. However the problem of unnecessary retransmission remains for further research.

Figure 2 shows the TCP-sender after adding the multiple packet drop support to the error discriminator.

6. PERFORMANCE EVALUATION

When a TCP sender receives a new packet, then it is one of three possibilities: New Acknowledgment, Duplicate Acknowledgment or a New/Dup Ackt packet with an ECN (The ECN is sent over Acknowledgments). Table 1 list all possible cases for an acknowledgment packet and its interpretation.

Table 1  Possible cases for an acknowledgment

<table>
<thead>
<tr>
<th>NewAck</th>
<th>DupAck</th>
<th>ECN</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Wireless Error</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Congestion Error</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Normal Ack</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Congestion Expected</td>
</tr>
</tbody>
</table>

Table 1 can be simplified as following:

- If ECN flag is 0 then either there is a wireless error or it is a normal acknowledgment and in either case TCP should not slowdown by using Reno slowdown. Instead TCP should just resend the lost packet in case of wireless error.
- If ECN is 1 then there is congestion or an expected congestion and in either case we have to slowdown.

Tables 2&3 shows the actions TCP-Reno and the modified TCP takes when Congestion or/and Wireless errors occur.
As we can notice from the tables, the only difference between Actions of TCP and modified TCP is when wireless error occurs alone. In this case there is no need to slowdown and TCP needs only to resend the lost packet.

Table 2  TCP-Reno reaction to errors

<table>
<thead>
<tr>
<th>Congestion error</th>
<th>Wireless error</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0</td>
<td>Do nothing</td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
<td>Slowdown+Resend</td>
</tr>
<tr>
<td>1 0</td>
<td>0</td>
<td>Slowdown+Resend</td>
</tr>
<tr>
<td>1 1</td>
<td>1</td>
<td>Slowdown+Resend</td>
</tr>
</tbody>
</table>

Table 3  Modified TCP reaction to errors

<table>
<thead>
<tr>
<th>Congestion error</th>
<th>Wireless error</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0</td>
<td>Do nothing</td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
<td>Resend</td>
</tr>
<tr>
<td>1 0</td>
<td>0</td>
<td>Slowdown+Resend</td>
</tr>
<tr>
<td>1 1</td>
<td>1</td>
<td>Slowdown+Resend</td>
</tr>
</tbody>
</table>

7. **END-TO-END PERFORMANCE WITH NO CONGESTION**

Here we will test the performance of the improved TCP in case of small or non congestion. The wireless error rate will vary from low to high error rates during the experiment.

A. **Experiment Setup**

The experiment was done using ns2[12] and the topology presented in figure 3.

The bandwidth is increasing in the way from the sender to receiver so we insure that only wireless errors occur and no congestion errors. We need that to show how the new improvements increase the performance of TCP in presence of wireless errors only.

We added Random Early Detection queuing mechanism (RED) to the node before the wireless link. It will monitor the queue on that node and send ECN in case of congestion.

For simplicity, the wireless link is simulated using a wired link suffers from random errors. The Errors were added to the link using the Default *ErrorModel* class provided by ns2. The ns2 documentation has given a full explanation of how to add errors to wired and wireless links. For more information see chapter 13 of ns2 documentation edited by Fall and Varadhan[13]. We have used the TCL script provided in the ns documentation to add errors in our topology.

Following we will present the experiment results and discussion.

B. **Results and Discussion**

We will refer to the improved TCP as TCP-ECN. A set of experiments has been done on TCP-Reno and TCP-ECN using the topology shown in figure 3 and the improvement achieved by using TCP-ECN has been recorded.

Figure 4 shows the improvements in the performance of TCP-ECN over TCP-Reno for error rates from 1% to 10%. Also we can notice the improvement reaches a peak and then starts to slow down. The initial observation indicates that the reason of the degradation in the improvement with the increase of the bit error is the RTO (Retransmission timeout). With the increase in the error rate more packets are lost and hence more probability to have a timeout. Moreover, several timeouts in a raw will lead to more waiting time before resuming transmission since TCP doubles the timeout each time the RTO clock expires. Figure 4 shows the improvement in the performance at each error rate.

A suggestion that we will investigate is to make TCP-ECN to increase the RTO with the increase in the BER hence preventing TCP from having timeout. The average congestion window and ssthresh has improved also in these experiments.

With the increase in the error rate, the improvement gained by using TCP-ECN decreases until there is no improvement under 40% error rate. We can explain this behavior under high error rates as following: TCP-ECN prevents unnecessary slowdown in case of wireless error by preventing unnecessary cuts of congestion window caused by wireless error. However, if a timeout occur, cutting the congestion window is unavoidable because timeout happens only if there is no feedback (i.e. Acknowledgments) received from the receiver. With no acknowledgments being received, we can’t discriminate
between errors using ECN since ECN comes over acknowledgment.

A suggestion to overcome this problem is to allow RED to send backward ECN. However, if these backward ECN will be sent over acknowledgments also, the problem will remain. So, we suggest sending backward ECN using dedicated packets for this reason. We will study the visibility of this suggestion and its impact on the network overhead during the future work.

C. Expected Improvement

Here we will compare the performance of TCP-ECN with the results obtained by simulating a version of TCP protocol that has prior knowledge of error types and can respond according to each error type differently; [4] call it the Perfect Discriminator. We simulate the function of the perfect discriminator by using TCP-Reno that cut the congestion window for congestion errors and when retransmission timeout occurs because of wireless error. The effect of wireless error has been reduced to cover only retransmission timeout. The results are shown in Figure 5 and show that TCP-ECN has very close performance to the TCP with perfect error discriminator (i.e. TCP that does not affected by wireless errors).

8. End-To-End Performance with Congestion

Here we repeat the previous experiment after adding UDP source to create moderate congestion. Topology used is similar to one in Figure 3 but with TCP and UDP senders. The UDP sender was added to introduce congestion as shown in figure 6.

The results show that the TCP-ECN over performs TCP-Reno with presence of moderate dropping rate between 0.00001 and 0.001, see figure 7. However, unlike the case of no congestion, with presence of congestion we can notice that the improvement increases with the increase in the wireless error rate. We can explain that as following: with the increase in the error rate number of wireless losses that coexists with ECN notification becomes lower. This will lead to reduce the number of cases where the error discriminator will confuses a wireless error as a congestion error and hence the discrimination process will give better results.

When working under error rates higher that 10%, the improvement decreases dramatically until it reaches 0% under 40% error rate. These results are very similar to the results we have got in case of no congestion. This is because the major factor affects the performance in high error rates is the timeout.
9. VALIDATION OF THE SIMULATION RESULTS

We applied an analytical module provided by Padhye et al. in [14] in order to validate the experimental results. This mathematical module computes TCP throughput in presence of errors taking into account the factors that affects the performance. The module contains two parts one to compute the effect of congestion window cuts on the performance and another to compute the timeout effect on the performance. We use the module to compute the expected results and compare them with what we have got. Biaz et al. in [2] gave a general explanation of how to use the module to compute the performance after adding wireless errors. The formula as presented by Padhye et al. in [14] is:

\[
B(p) = \frac{1}{RTT \sqrt{\frac{2bp}{3} + T_0 \min\left(1, \frac{3bp}{8}\right)p\left(1 + 32p^2\right)}} \tag{1}
\]

The formula contains two parts. First part is:

\[
RTT \sqrt{\frac{2bp}{3}} \tag{2}
\]

In this part the effect of congestion window cuts in the performance is calculated. The \(RTT\) is the average round trip time, \(b\) is number of packets acknowledged per acknowledgment and \(p\) is the error rate.

The second part considers the effect of timeout on TCP performance:

\[
T_0 \min\left(1, \frac{3bp}{8}\right)p\left(1 + 32p^2\right) \tag{3}
\]

In formula (3) \(T_0\) is the first timeout during the connection. We use formula (1) to compute the performance of TCP after introducing the wireless and congestion errors. Biaz et al. [2] explain how to introduce both error types by considering wireless error to be \(p_w\) and the congestion error to be \(p_c\). Then the error rate \(p\) in formula (1) is computed as following \(p = p_w + p_c\).

However, TCP-ECN prevent window cuts in case of wireless errors so in formula (2) the only error rate affect the window size is \(p_c\). \(p_w\) has no effect on formula (2). The formula (2) will be as following:

\[
RTT \sqrt{\frac{2p_c}{3}} \tag{4}
\]

However, both wireless errors and congestion errors can cause timeout in TCP-ECN so we will include both error types in formula (3):

Figure 7. TCP-ECN throughput improvement for Error rates 1% to 10% and with presence of drop rate.

Figure 8. Analytical Vs Experimental Improvement
Finally the formula to compute the expected performance of TCP-ECN is:

\[ B(p) \approx \frac{1}{RTT \sqrt{\frac{2bp_c + T_{\min}}{3} + T_{\min} \left( \frac{3b(p_c + p_w)}{8}(p_c + p_w)(1 + 32(p_c + p_w)^2) \right)}} \] (6)

The formula (6) was used to compute the expected throughput of a TCP-ECN. We then use formula (1) to compute the expected throughput of TCP-Reno. The improvement as a result of applying this formula found very close to the improvement we got from using TCP-ECN in low or no congestion. In figure 8 we show the Improvement in throughput of both the TCP-ECN and the throughput calculated using formula (6) (named Analytical in the chart) over TCP-Reno and as we can see they are very close. All these experiments and analytical calculation were done assuming a little dropping rate of 0.0008 which is equivalent to 1 dropped packet every 1250. The validation for the simulation results in case of medium congestion is left for future work.

10. CONCLUSION AND FUTURE WORK

In this work we have tested the performance of a proposal to improve TCP performance over wireless networks [1]. The proposed solution suggests that TCP uses feedback from congested nodes to discriminate between congestion and wireless errors and act differently in response to each type of errors.

In our work we showed that explicit congestion notification can be used to improve TCP performance over network suffering from different rates of wireless errors. A new version of TCP, TCP-ECN, has been designed and tested using network simulator ns2[12]. TCP-ECN use an error discrimination module to discriminate between error types based on the feedback from the congested nodes. The results showed that TCP-ECN outperforms TCP-Reno in networks suffering from different wireless error rates and low or moderate congestion. The initial observations showed that discriminating between error types in case of high congestion is not helpful because in case of high congestion the priority is to resolve the congestion by reducing the sender transmission rate.

As a future work we will give RED more study to see how to improve RED to give better performance when interact with wireless environment. One proposal is to make automatic update of the minimum and maximum threshold with different number of flows. For example we want to investigate if with the increase in number of flows increasing the minimum threshold will increase the performance because less ECNs will be generated and hence less cuts to the congestion window. Another proposal is to test if we can improve RED by making it to monitor the direction of the queue and to generate ECN based on the direction. For example ECN will be generated if the queue direction is “up” meaning that the queue is building up. However, if the queue is decreasing, even if the average queue size is more than the min-threshold, then less, or none, ECN notifications are generated. Considering the queue direction will give TCP sender a clearer image of the congestion and how it is going and this will help the error discriminator.

However, in this work we consider changes in TCP (Sender) functionality only, so all work in RED is left for future work because we focused on solutions that does not require changes on the intermediate network functionality.

Testing the new protocol in heavily congested networks also left for future work.

REFERENCES


