



**IN THE NAME OF ALLAH,
MOST GRACIOUS, MOST MERCIFUL**

An Extended Review of Techniques for Enhancing TCP Performance

Mohammed A. Alnuem

*Department of Information System,
College of Computer and Information Sciences, King Saud University,
P. O. Box 51178, Riyadh 11543, Saudi Arabia
malnuem@ksu.edu.sa*

(Received 1/7/2009; accepted for publication 30/12/2009)

Keywords: TCP, Lossy networks, Error discriminators.

Abstract. Transmission Control Protocol (TCP) is considered one of the most important protocols in the Internet. An important mechanism in TCP is the congestion control mechanism which controls TCP sending rate and makes TCP react to congestion signals. Nowadays in heterogeneous networks, TCP may work in networks with some links that have lossy nature (wireless networks for example). TCP treats all packet loss as if they were due to congestion. Consequently, when used in networks that have lossy links, TCP reduces sending rate aggressively when there are transmission (non-congestion) errors in an uncongested network. In this paper, we present different solutions to overcome the performance degradation problem TCP faces when working over lossy links. Many solutions have been proposed, but we will concentrate on end-to-end solutions that require no help from the intermediate network.

1. Introduction

Many solutions have been proposed to overcome the problem of TCP bad performance over lossy links (like wireless networks). Some solutions were in the transport layer and some solutions were in lower layers like the link layer.

Balakrishnan *et al.* (1996) divided the solutions into two general categories: (1) Solutions that make TCP unaware of the errors that happen in the link so TCP thinks that it works on reliable links with no transmission errors, for example Snoop agents (Balakrishnan *et al.*, 1995). (2) Approaches to try to make TCP aware of the errors caused by the lossy link and make TCP avoid using congestion mechanisms for this type of errors. However, other authors like Elaarag (2002) divide these solutions into the following more specific categories: Link Layer, Split Connection and End-To-End solutions.

In the Link Layer solutions, the aim is to completely hide the errors that occur in the link so TCP will be unaware of them, and hence it will not reduce its transmission rate as a reaction to those errors. In general, link layer solutions are used for wired-wireless networks and they can be located at the base station which connects the wired network

with the wireless link just before the receiver. They monitor the packets that pass the base station from one end to another and keep record (and sometimes copies) of the packets sent and set a retransmission timeout for each packet. When the wireless link drops a packet, either a timeout will occur or duplicate acknowledgments will be received at the base station. The base station then resends the lost packet and suppresses the duplicate acknowledgment at the base station so TCP does not notice the drop, and hence will not need to reduce its transmission rate.

A good feature of this approach is that it preserves the end-to-end semantics of TCP since it does not break the connection (i.e., the connection negotiation and maintenance remains between the sender and the receiver only). However, the problem is that sometimes this method cannot completely hide errors from the TCP sender. For example, when a mechanism like Snoop resends a dropped packet but the packet is dropped again due to high error rates and then the TCP timeout for this packet occurs before Snoop has a chance to resend it again. This could happen because of the mismatch between the TCP and Snoop retransmission timeout mechanisms (RTO). In principle, the Snoop RTO should be shorter than the TCP RTO, but this is not always true

(Balakrishnan *et al.*, 1996; Elaarag, 2002). Moreover, sometimes Snoop's aggressive retransmission may cause congestion at the base station which may reduce the link utilization. Examples of link layer solutions are TULIP (Parsa and Garcia-Luna-Aceves, 2000), Snoop (Balakrishnan *et al.*, 1995) and AIRMAIL (Ayanoglu *et al.*, 1995).

In Split Connections protocols, the aim is to divide the problem into two smaller ones. This is done by separating the wired link connection from the wireless link connection. This is usually done at the base station where two connections are maintained. One standard TCP connection from the wired host to the base station and another wireless connection from the base station to the mobile host where a new protocol that can handle wireless errors is implemented. The base station plays the role of the interface between the two connections (Elaarag, 2002; Bakre and Badrinath, 1997). The TCP connection from the sender ends at the base station, and then the base station starts a new connection with the receiver.

A good feature of this method is that we do not need to do any changes at the sender because the sender does not need to deal with the errors on the wireless link. However, the sender is not now negotiating the connection with the end receiver, so the connection between the sender and the receiver is broken and the end-to-end semantics of TCP no longer hold. An example of this category is I-TCP (Bakre and Badrinath, 1997) and M-TCP (Brown and Singh, 1997).

The last category is the End-To-End solutions. Next, we will talk about them in more detail.

2. End-to-End Solutions

In general, most of the end-to-end solutions, as the name indicates, try to deal with the problem at the end point of the connection (sender and receiver) and do not expect help from the network so they look at the network as a black box. The main advantage of this approach is that it does not add overhead to the network. However, some proposed solutions from this category use some sort of indirect feedback from the network as we will see later.

Mainly the following techniques try to find ways to recover from drops (congestion and transmission) efficiently. Some of these solutions were designed for a wireless environment and some were introduced before the introduction of wireless technology. However, since the aim of all these solutions is to recover from errors efficiently, they can be used for improving TCP performance over

networks with both congestion and non-congestion (transmission) errors.

3. Congestion Drops

In the following, we will begin with techniques designed to recover from congestion drops. Later, we will present techniques designed for congestion and transmission drops.

3.1. Retransmission timeout (RTO)

In retransmission timeout, TCP attaches a timer with each sent packet, and when the timer expires before receiving acknowledgment for that packet, TCP resends the lost packet and sets the congestion window to the minimum allowed size. For more about how RTO is calculated using the RTT see Postel (1981) and Jacobson (1988).

RTO is one of the first methods provided to TCP to recover from errors. However, it is most efficient when the congestion is serious and the network needs more time to drain the congested nodes. On the other hand, if the drops are caused by transient congestion, then it is better to resend the lost packet without waiting for a timeout to occur. This idea is the base of the fast retransmission mechanism which we will talk about in the next section.

3.2. Fast retransmission (TCP-Reno)

In earlier implementations of congestion control mechanisms, there was an assumption that errors due to segment damage are rare (less than 1% of the sent segments (Jacobson, 1988)), and so it is assumed that most of the segment loss is because of congestion (Jacobson, 1988; Stevens, 1997). As a result, when there is a high packet damage rate or when the congestion loss rate exceeds 1%, the TCP performance will suffer badly. According to Jacobson (1990), TCP will lose between 50% and 75% of its throughput when the error rate reaches 1%.

This shows how congestion control mechanisms are intolerant to high error rates. This behavior can be explained if we return to the combined slow start congestion avoidance algorithm explained in Stevens (1997) and Allman, Paxson and Stevens (1999). In this algorithm, Stevens (1997) explained how TCP should react to congestion as follows: If there is a duplicate acknowledgment, then TCP should set the slow start threshold $ssthresh$ to half of the current window size (the window size when the error happened) and then enters the slow start mode when a timeout occurs. This way all drops will be recovered by entering slow-start. However, TCP performance will decrease sharply.

For this reason, Jacobson (1990) suggests that TCP can use the knowledge brought by duplicate acknowledgment to resend the lost packet (a fast retransmission) and then there is no need to enter slow start because the duplicate acknowledgments indicate that these packets have left the network and there is more space for new packets to be injected into the network (Allman, Paxson and Stevens, 1999) so no need to reduce the cwnd to one segment by entering slow start. Instead, TCP enters congestion avoidance by reducing the congestion window to the half of the current window size.

Another reason for not using slow start is given by Stevens (1997). He noted that because we know that there is still data flowing in the connection because of the duplicate acknowledgments we received, we do not want to cut this flow by entering slow start (Stevens, 1997).

3.3. Fast retransmission phase (TCP-NewReno)

When Stevens (1997), in the RFC 2001, explained the slow start algorithm, he indicated that the first step in the algorithm is to initialize the slow start threshold variable *ssthresh* to a high value (65,535 bytes). Also, Allman, Paxson and Stevens (1999), in RFC 2581, indicated that *ssthresh* could be set to an arbitrary high value. However, Hoe (1996) noted a problem in this step of the algorithm that may affect TCP performance.

The problem is that during the start up phase of TCP connection (slow start), the sending rate grows exponentially until the congestion window reaches the *ssthresh*. So, giving *ssthresh* a high value will inject the network with a high number of segments in a short period of time.

However, the network may be unable to handle that amount of data at once and, hence, some packets may be dropped due to congestion. Moreover, due to this congestion, more than one segment may be dropped from the same window (Hoe, 1996) and this will create problems to the TCP fast retransmission mechanism proposed by Jacobson (1990).

The fast retransmission algorithm (Jacobson, 1990) can handle only one drop per window, and hence if more than one segment is dropped from the same window, only one will be resent by fast retransmission and TCP will recover from the other losses by using a retransmission timeout (RTO) which will initiate the slow start algorithm which will reduce the congestion window size to its initial value (usually one segment) and TCP performance will suffer badly (Hoe, 1996).

To understand why the retransmission algorithm cannot recover from multiple drops, Stevens (1997)

indicated that the fast retransmission algorithm is terminated whenever a new acknowledgment is received. This new acknowledgment is assumed to acknowledge all packets sent after the lost segment up to the window size. However, if multiple segments were dropped, this acknowledgment will acknowledge only the segments that have been received correctly up to the second drop. Hence, fast transmission will be terminated before resending all lost segments and TCP will enter a series of retransmission timeouts causing the performance to degrade.

As a solution, Hoe (1996) suggested a change in the fast retransmission algorithm, so it will not exit until it receives an acknowledgment for all dropped segments. This is done by ignoring the new acknowledgments that acknowledge only part of the sent segments and repeating fast retransmission until the sender receives an acknowledgment for all sent segments. This way, there is no need to wait for the retransmission timeout (RTO) to force resending the rest of the lost segments. Floyd and Henderson (1999) call the intermediate acknowledgments the partial acknowledgments.

A new variation of TCP was proposed based on these modifications and called TCP-NewReno (Floyd and Henderson, 1999). Also, Floyd and Henderson (1999) have introduced two options of NewReno regarding when to reset the retransmission timeout: The first option is called slow-but-steady NewReno, and second is called impatient NewReno. In the former, the timeout clock is initialized after each partial acknowledgment. This way TCP will stay in fast recovery mode as much as possible, but as the name indicates, the resending rate will be as low as one packet per round trip time (RTT). However, in the impatient NewReno, TCP will reset the RTO only after the first partial acknowledgment, so if there are too many packets dropped from the same window, then RTO will eventually expire before receiving a new acknowledgment and, hence, TCP will enter slow start (Floyd and Henderson, 1999) and resend all dropped packets and cut the congestion window at the same time.

3.4. Selective acknowledgment (TCP-Sack)

The original idea of using selective acknowledgments (SACK) was proposed initially by Jacobson and Braden (1988). However, detailed implementation and improvements to the idea were proposed later by Mathis *et al.* (1996).

Selective acknowledgment is a change to the way the TCP receiver reacts to receiving new packets. Usually, when the TCP receiver receives a

new packet, it sends an acknowledgment to the sender that carries the received packet sequence number which indicates to the sender that all previous packets up to this one have been received successfully at the sender. Because of that, it is called cumulative acknowledgment (Postel, 1981). This way new acknowledgments will be sent only if the packets are received in order, otherwise the acknowledgment will be sent for the last in-order packet received (duplicate acknowledgment).

However, using selective acknowledgments, the receiver will send an acknowledgment for each packet no matter in what order it has arrived. This way the sender will have a clear idea of what packets have been received successfully and this will solve the problem we described before when more than one packet is dropped from the same window (Mathis *et al.*, 1996).

Also, using selective acknowledgment will allow TCP to resend all lost packets without the need to do unnecessary retransmission of packets already received (Fall and Floyd, 1996). Using selective acknowledgment does not require the overhead of extra traffic since it is sent over normal acknowledgments (Mathis *et al.*, 1996; Fall and Floyd, 1996).

However, a disadvantage of the implementation explained in Mathis *et al.* (1996) is that it requires the use of a retransmission queue to save unacknowledged segments. Also, it requires the TCP sender to keep a record of the received acknowledgments. This may require more memory usage and perhaps more processing power for sorting and comparing sequence numbers for segments in the queue especially when TCP uses a large sending window (congestion window). Moreover, SACK is helpless when retransmission timeout occurs; all segments in the retransmission queue will be resent even if they have been sent before (Mathis *et al.*, 1996). Last but not least, applying selective acknowledgment requires changes to both sender and receiver sides which may be hard in real networks.

On the other hand, accumulative acknowledgments which were used in most TCP variations is simple and allows easy management of incoming packets with no need for extra memory or processing as in the case of selective acknowledgment. For example, it is easy, in standard TCP, to discover a receipt of duplicate copies of a packet by simply comparing the packet sequence number with the last acknowledged packet number (Postel, 1981). This way TCP does not need to keep a record of the received packets and only needs to save the last in-order received packet sequence number.

Another advantage of the standard approach is that if an acknowledgment is dropped, then it is enough to receive another acknowledgment with higher sequence number since it acknowledges all packets with lower sequence numbers.

4. Congestion and Transmission Drops

In this section, we will present techniques designed to improve TCP performance for congestion and transmission drops.

4.1. Congestion predictors

In this type of solution, TCP uses techniques such as delays on the links (round trip time) like the CARD (Jain, 1989) technique (CARD stands for Congestion Avoidance using Round Trip Delay) or the connection throughput in the case of the Tri-s scheme (Wang and Crowcroft, 1991) and the Vegas scheme (Brakmo, O'Malley and Peterson, 1994) to predict if there will be congestion and then control the inflation and the deflation of the congestion window based on this prediction. If the predictor does not see congestion happening in the near future, then it suggests increasing the congestion window. On the other hand, if the predictor notices that congestion is coming, then it suggests that TCP decrease the congestion window. As we can see, unlike TCP, drops are not used here to control the growth of the congestion window.

In theory, the perfect predictor will eliminate congestion errors since it will detect and avoid congestion before it happens. So, if an error happens, then it can be considered to be caused by the link failure (like wireless errors) rather than by congestion. We will see later how congestion predictors can be used to build error discriminators. The following are brief explanations of some congestion predictors.

4.2. TCP-Vegas

TCP-Vegas (Brakmo, O'Malley and Peterson, 1994; Brakmo and Peterson, 1995) is a modification to the congestion control mechanism in standard TCP (Jacobson, 1988; Jacobson, 1990). It aims to reduce the congestion losses and to increase TCP throughput by predicting the available capacity on the link and trying not to exceed it.

According to the Vegas authors in Brakmo, O'Malley and Peterson (1994), Vegas has increased the throughput of TCP up to 70% more than older implementations of TCP (TCP-Tahoe & TCP-Reno). Also, Vegas has reduced the losses in the link up to 50% (Brakmo, O'Malley and Peterson, 1994).

We will give here an extended explanation of TCP-Vegas because of its importance and since some other solutions are based on Vegas as we will see later.

TCP-Vegas introduces changes to TCP in four areas as follows:

Timeout computation: The authors of Vegas have noticed from experiments over the Internet that the timing mechanism used in previous implementations of TCP is not accurate, and that computing round trip time (RTT) using current timing mechanisms has given higher RTT estimations. This makes TCP take up to three times longer to recover from losses (Brakmo, O'Malley and Peterson, 1994). A new mechanism has been introduced based on using a time stamp for each packet and computing the round trip time by comparing the packet's time stamp with its acknowledgment time stamp. This way a more accurate retransmission timeout can be computed.

Retransmission of lost packets: TCP-Vegas introduces a new retransmission mechanism by changing the way TCP responds to duplicate acknowledgments. TCP needs to receive three duplicate acknowledgments before it retransmits the lost packet. However, when Vegas receives the first duplicate acknowledgment for a segment, it compares the time stamp with the current time. If the difference is more than the computed timeout, then it triggers retransmission without waiting for more duplicate acknowledgments to come (Brakmo, O'Malley and Peterson, 1994). As we can see this will add overhead to the system to record a time stamp for each segment and save it until it receives an acknowledgment. However, the authors indicated that the overhead of using TCP-Vegas will not exceed 5% more than older implementations (Brakmo, O'Malley and Peterson, 1994).

The other area in which TCP-Vegas provides changes is in congestion avoidance: TCP-Vegas has made dramatic changes to the congestion avoidance mechanism used in TCP by making TCP to increase/decrease the sending rate, not based on packet drops as in TCP, but based on the prediction of available link bandwidth.

Vegas estimates an expected throughput and an actual throughput for the connection. The expected throughput is computed using the current window size and the minimum RTT seen so far. The actual throughput is computed using current window size and last RTT reading.

Then, Vegas compares the expected throughput and the actual throughput and updates the sender window according to the comparison results as

following: If the actual throughput is less than the expected one, then TCP is unable to utilize the link because there is congestion and hence it should decrease the window size (Brakmo, O'Malley and Peterson, 1994). On the other hand, if the actual throughput becomes closer to the expected throughput, then it is safe to increase the window size. The increase and decrease in the window size is linear unlike TCP which uses Jacobson's AIMD (Jacobson, 1988) mechanism (Additive increase multiplicative decrease) to update the congestion window.

The Vegas algorithm is expected to prevent congestion from occurring and, hence reduce congestion drops dramatically.

Slow start: In Vegas, the congestion predictor, explained above, is added to the slow start mechanism. Another modification Vegas makes to slow start is that the update of window size during slow start is not done every RTT; instead, it takes two RTTs before increasing the window size. This is done to give the algorithm a chance to measure the actual throughput between updating window size (Brakmo, O'Malley and Peterson, 1994).

Hengartner, Bolliger and Gross (2000) have reviewed each of the modifications Vegas did to TCP. Their results show that the new retransmission technique has improved the performance noticeably because it was able to avoid timeouts during multiple packet drops from the same window. It does this by performing retransmission when its new timeout mechanism expires even before receiving duplicate acknowledgments.

However, the results in Hengartner, Bolliger and Gross (2000) showed that TCP-Vegas suffers from performance degradation when it coexists with versions of TCP that use the AIMD mechanism like TCP-Reno. This is because the AIMD mechanism is more aggressive in grabbing the link bandwidth because it keeps increasing the window size until an error occurs while Vegas tries to prevent causing drops and hence it keeps smaller window size. This indicates that the congestion predictor in Vegas sometimes has a negative impact on the performance (Hengartner, Bolliger and Gross, 2000). Also, we will see later how Biaz and Vaidya (1998) have confirmed this fact (i.e., Vegas predictor poor performance) when we talk about using the Vegas congestion predictor in an error discriminator.

4.3. TCP-Westwood

Mascolo *et al.* (2001) proposed a modification to the congestion avoidance algorithm used in TCP-Reno, which uses duplicate acknowledgment and

timeout as an indicator for congestion and to update the sender window (Jacobson, 1988). However, duplicate acknowledgments do not give an indication of the type of the error (congestion or transmission error). For this reason, Mascolo *et al.* (2001) suggested that the TCP sender should do continuous estimation of the bandwidth and update the window size according to that estimation (Mascolo *et al.*, 2001). This way TCP will send in a rate that will occupy the available bandwidth only and hence any error could be considered safely as a transmission error. Westwood estimates the available bandwidth by monitoring incoming acknowledgments and assumes this rate reflects available link capacity in the forward path (Mascolo *et al.*, 2001).

Also, TCP-Westwood (Mascolo *et al.*, 2001) suggested that TCP does not need to halve the window size when errors happen, like TCP-Reno. TCP-Reno halves the window size whenever there is an error and hopes this action will solve the congestion and, at the same time, it increases the congestion window linearly to utilize the available bandwidth without more investigation of the link status. In contrast, after each drop TCP-Westwood (Mascolo *et al.*, 2001) uses the estimated bandwidth-delay product to set the sender window according to the current congestion level (Gerla *et al.*, 2001).

The authors of TCP-Westwood (Mascolo *et al.*, 2001) reported big improvements in TCP performance, especially over networks suffering from transmission errors like wired-wireless networks (Mascolo *et al.*, 2001). This improvement has been confirmed by Grieco and Mascolo (2004). Also, the experimental results reported in both Mascolo *et al.* (2001) and Grieco and Mascolo (2004) showed that TCP-Westwood has maintained fair sharing of the bandwidth and it does not lead to starvation of TCP-Reno connections.

However, Biaz and Vaidya (2005) did experiments on TCP-Westwood when coexisting with non-TCP traffic on the reverse link, and their results indicate that TCP-Westwood could not estimate the link capacity correctly when a non-TCP traffic exists in the reverse path. This result can be explained since bandwidth estimation in TCP-Westwood is based on taking the average rate of received acknowledgments and, since the added traffic in the reverse path could add additional delay to the received acknowledgments, TCP-Westwood will underestimate the available bandwidth.

4.4. Error discriminators

All methods that try to understand the cause of the error and to act differently to each type of error

based on that understanding are called error discriminators.

Some error discriminators deal with the network as a black box and do not need any feedback from the network in order to discriminate errors. Other types of error discriminators use help from intermediate networks in order to understand the cause of the error.

In the following, we will talk about both types and we will start with error discriminators that depend upon the network to help in distinguishing errors. As far as we know this is the first attempt to classify error discriminators.

4.5. Network dependent error

Discriminators network dependent error discriminators are actually based at the end-point of the connection but use help from the intermediate nodes. However, although they are not totally end-to-end, we mention them here for two reasons. First, all network dependent error discriminators explained in this section use already popular active queuing mechanism techniques like the use of explicit congestion notification through RED (Random early dropping) (Floyd and Jacobson, 1993) queues. The second reason is that we want to complete the picture about the error discrimination techniques.

The advantage of this approach is that both end hosts can have detailed information about the cause of drops and the network status.

However, if we want to apply this approach in a large network, we may need a wide-scale change to the network components (i.e., mainly we need to change the routers if we want notification for congestion drops and we need changes in the wireless base stations if we want wireless drop notification).

In the following, I will explain briefly some network dependent error discriminators.

4.6. TCP-Casablanca

The key idea TCP-Casablanca introduces (Biaz and Vaidya, 2005) is as follows: Congestion errors and transmission errors usually happen randomly and this is basically why it is difficult to differentiate between them. However, if we can “de-randomize” (Biaz and Vaidya, 2005) the congestion errors by making congestion errors take a non-random form, then it will be easy to discriminate the non-random congestion errors from the random wireless errors (Biaz and Vaidya, 2005).

The mechanism works as follows: The sender marks each outgoing packet with one of the marks (in/out) in a consistent pattern, for example by marking four packets with (in) and the fifth packet (out) and so on. When congestion occurs at

intermediate nodes, there should be a biased queue-management mechanism that drops only the packets marked with the (out) mark. This way, the receiver receives the packets with a consistent pattern of drops, because only packets marked with (out) are dropped, so the receiver recognizes that the errors are congestion errors.

On the other hand, if a wireless error occurs, then the drops will be random among all packets (in-marked and out-marked) and, hence, the receiver can recognize that these random errors are wireless errors (Biaz and Vaidya, 2005).

If the receiver diagnoses a wireless error, it marks the acknowledgments with an explicit loss notification (ELN). When the TCP sender receives a duplicate acknowledgment because of error, it checks if the acknowledgment contains ELN and, if so, TCP considers the loss to be wireless loss; otherwise, it considers it to be a congestion error (Biaz and Vaidya, 2005). In case of congestion error, TCP cuts the congestion window; otherwise, it only resends the packet and does not cut the congestion window.

So, as we can see that applying this mechanism requires mainly four changes to TCP sender/receiver and the network. First, adding an error discriminator at the sender (which is called Casablanca) and acting according to the ELN signals it receives. Second, the receiver should be able to deduce when a random or non-random drop occurs and to send ELN if a random error occurs. Third, an active queuing mechanism should be implemented in the bottleneck, which will drop only packets marked with the (out) mark. Finally, the packet format should be altered to add in/out marking and ELN. The reset of the protocol is based on the NewReno (Floyd and Henderson, 1999) version of TCP.

The authors indicated that the Casablanca discriminator has achieved high accuracy in discriminating between congestion and transmission errors and, using it in TCP gave significant (above 100%) improvement in TCP performance (Biaz and Vaidya, 2005). Accuracy is a crucial component in this error discriminator since it uses an aggressive action toward non-congestion drops by not cutting the congestion window size for these drops and keeps it as big as it was before the drop.

4.7. TCP-Ifrane

TCP-Ifrane (Biaz and Vaidya, 2005) is a sender based version of TCP-Casablanca (Biaz and Vaidya, 2005). In TCP-Ifrane, changes are made at the sender only and not the receiver. When the sender sends a packet, it records whether this packet is marked as out or in. If the sender receives a duplicate

acknowledgment indicating that a packet is lost, it looks at its record and sees if that packet was marked out or in when it was sent. If the packet was marked out the error is considered to be congestion error; otherwise, it is considered as a wireless error (Biaz and Vaidya, 2005). TCP-Ifrane was found to give higher throughput than TCP-Casablanca; this is because it has less congestion accuracy and hence it slows down less than TCP-Casablanca (Biaz and Vaidya, 2005). However, the effect of TCP-Ifrane's accuracy was not studied by Biaz and Vaidya (2005).

4.8. Explicit congestion notifications

Explicit congestion notification (Floyd, 1994) was first introduced to help TCP avoid congestion by allowing intermediate nodes to set a congestion notification bit in the IP header whenever congestion is expected. The TCP sender will respond to this notification by reducing its transmission rate. An Active Queue Management (AQM) mechanism (e.g. RED (Floyd and Jacobson, 1993)) is placed at the congested nodes and becomes responsible for marking packets when congestion is expected (in case of RED the packets will be marked when the queue reaches a particular threshold).

Using ECN requires changes in both the TCP sender and receiver. Also, it requires the use of AQM at the congested nodes. However, using ECN does not require changing the TCP congestion mechanisms since TCP responds to ECN in the same way as it responds to a packet drop.

Dawkins *et al.* (2000) have proposed the use of ECN to improve TCP performance over wireless links by modifying the way TCP responds to ECN. Biaz and Wang (2004) explain the technique as follows: If a drop is detected by receiving duplicate acknowledgments, then we look if we have received an ECN 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 the ECN preceded the drop, then TCP considers this drop to be a congestion drop and acts by reducing the senders window size in order to resolve the congestion.

However, if the drop happens while not preceded by ECN, it is then considered as a wireless error and TCP does not reduce the senders window size (Biaz and Wang, 2004). However, we still need to retransmit the lost packet. Biaz and Wang (2004) argue that this approach will improve TCP performance over networks with transmission errors like wireless networks especially those that suffer from high error rates.

However, Biaz and Wang (2004) studied the possibility of using ECN to distinguish between error types. They argue that this approach is not an accurate method to differentiate between congestion and transmission errors and showed that transmission errors can be random, so that the probability that ECN will precede a congestion error is approximately the same as the probability that ECN will precede a transmission error (Biaz and Wang, 2004).

So, Biaz and Wang (2004) proposed that instead of using ECN directly to infer the type of the error, TCP should also look at the state of the sender. If the sender was in congestion avoidance phase, then the drop is probably a transmission drop. However, if the sender was in slow start phase, then the drop is considered a congestion drop. The new protocol is called TCP-Eaglet (Biaz and Wang, 2004) and it showed improvement over standard TCP performance.

4.9. TCP-Jersey

Xu, Tian and Ansari (2005) have suggested using the estimated bandwidth instead of errors to tell TCP when to decrease sender window size, which is an idea similar to Westwood (Mascolo *et al.*, 2001) but with a different implementation. The available bandwidth is estimated based on the rate of arrival acknowledgments. High acknowledgment arrival rate means packets can get to the other end fast, and hence high network capacity. Moreover, in this approach the nodes in the middle should be able to mark packets when congestion is expected in order to notify the sender (Xu, Tian and Ansari, 2005). So, this method is a combination of TCP-Westwood and ECN error discriminator (Dawkins *et al.*, 2000) except that it differs in some implementation details in both cases.

However, like TCP-Westwood, TCP-Jersey may suffer from performance degradation when coexisting with non-TCP traffic on the reverse link because it cannot estimate the link capacity correctly since the added traffic in the reverse path can delay the acknowledgments, so it will underestimate the available bandwidth.

However, improvement has been done to TCP-Jersey to overcome this problem. The improved version is called TCP-New Jersey (Xu, Tian and Ansari, 2005) and it uses acknowledgment timestamps (Jacobson, Braden and Borman, 1992) instead of acknowledgment arrival rate to calculate estimated bandwidth which solves the problem of delayed acknowledgments because each acknowledgment has a time stamp which allows the sender to calculate the forward path delay. The authors indicated that simulation results of TCP-New

Jersey gave good results and show improvement in TCP performance particularly with reverse paths that suffers from congestion and lossy links (Xu, Tian and Ansari, 2005).

4.10. Network independent error discriminators

This kind of solution implicitly infers the cause of packet drop without the need of explicit notification from the network about the cause of the drop. In this kind, the solution is based at the end hosts (or one of them). The advantage of using this approach is to keep the changes to a minimum (to the end hosts) and there is no need to make changes to the network components, which may require wide scale changes. However, an obvious limitation to this approach is that the end hosts will not have detailed information about the status of the congestion or the transmission drops and can only guess the situation using implicit signs from the network (like packet delay for example).

Some of these solutions are based on using congestion predictors like Vegas (Brakmo, O'Malley and Peterson, 1994) or CARD (Jain, 1989) or Tri-s (Wang and Crowcroft, 1991). In this approach, the discriminator works by taking input from the congestion predictor about the congestion status when a drop occurs. If the congestion predictor was predicting congestion, then the drop is considered to be a congestion loss. However, if the predictor was suggesting increasing the sending rate because it does not predict any congestion in the near future, then the drop is considered to be caused by a link error (Biaz and Vaidya, 1998).

Also, we must notice that as Biaz and Vaidya (1998) indicated, designing an accurate error predictor is important since mistakes of distinguishing transmission errors from congestion errors could cause unnecessary congestion which is usually avoidable by using normal congestion control algorithms (Biaz and Vaidya, 1998). For example, if a congestion error is mistaken to be a transmission error, then TCP will not decrease the window size and this will make the current congestion much worse.

Experiments were performed by Biaz and Vaidya (1998) on three different error discriminators based on congestion predictors: The CARD (Jain, 1989), Tri-s (Wang and Crowcroft, 1991), and Vegas (Brakmo, O'Malley and Peterson, 1994). Unfortunately, the results obtained by Biaz and Vaidya (1998) experiments show that these congestion predictors are no better than a random loss predictor. From these results, Biaz came to the conclusion that these three congestion predictors are not suitable as an accurate error discriminator.

The reason which leads to the failure of these methods to make a good error discriminator is that they assume that if one TCP increases its congestion window then the network delay will increase. So, they assume that one connection can affect the whole network. Using this assumption, if TCP is able to gain high throughput, then this is an indication that the network is not congested. On the other hand, if TCP is able to gain only a small part of the expected network throughput, then this means that a congestion exists.

However, Biaz and Vaidya (2003a) showed that when TCP increases its sending rate the RTT could go either way (i.e. increase/decrease). They showed that the correlation between a single connection sending rate and the RTT is weak (Biaz and Vaidya, 2003a). This is because usually a single connection forms a small part of the network aggregate traffic.

However, Biaz and Vaidya (2003a) also emphasized on the sensitivity of the network delay to the total load, which makes the measured RTT a good indication of congestion events and hence RTT can be used to build an effective error discriminator.

In the following sections, we will present briefly some error discriminators based on congestion predictors and show how they work.

4.11. Error discriminator based on Vegas congestion predictor

Based on the Vegas predictor (Brakmo, O'Malley and Peterson, 1994) described earlier, Biaz and Vaidya (1998) proposed an error discriminator that computes the difference between expected throughput (link capacity) and the actual throughput in order to predict congestion and use this difference to define a new variable f_{Vegas} . The difference is computed as follows: $D = \text{expected throughput} - \text{actual throughput}$. If $D > 0$, this means that TCP throughput is less than what it should be to utilize the link, and this indicates that congestion exists in the connection path and hence any drop is considered to be a congestion drop. On the other hand, if $D \leq 0$, this means that TCP throughput is actually able to utilize the link capacity and hence there is no congestion and any error is considered to be a transmission error.

The simulation results in Biaz and Vaidya (1998) show that the Vegas based error discriminator has achieved low to medium performance in terms of accuracy in defining error types. As we said before, this is due to the assumption that the network will respond noticeably to the changes in a single connection window. This is not always true since, in large networks, a single connection forms a small fraction of the whole traffic (Biaz and Vaidya, 1998) and this will affect the error discriminator ability to

discover congestion errors.

This also applies to the next two error discriminators based on CARD (Jain, 1989) and Tri-s (Wang and Crowcroft, 1991) congestion predictors.

4.12. Error discriminator based on card delay-based congestion

Predictor Congestion Avoidance Round trip Delay (CARD) (Jain, 1989) is an approach to update the TCP sender window size without the need to have any feedback from the network. It is called (Jain, 1989) a black-box approach since it deals with the network as a black box and does not require any explicit feedback from the network. It works by analyzing the relation between the round-trip delay and the throughput of the connection in order to predict the optimum window size that gives maximum throughput with minimum delay. Jain (1989) calls it maximum Power where the power is the ratio of throughput and delay: $\text{Power} = (\text{Throughput}/\text{Delay})$ (Jain, 1989). The aim is to have maximum Power.

Unlike TCP, this approach does not use errors to update the window size which is approach similar to TCP-Vegas (Brakmo and Peterson, 1995). However, Jain (1989) did not provide a complete TCP solution like TCP-Vegas; instead, it gives a mechanism that can be used to replace Jacobson's (1988) congestion avoidance mechanism in TCP.

The CARD (Jain, 1989) measures the change of the increase/decrease rate in the connection throughput and delay. When the network is fully utilized, then any small increase in the throughput will result in a big increase in the observed delay. This gives a good indication that the network is congested. However, when the network capacity is underutilized, then the increase in the throughput will result in a small (or none) increase in the network delay.

Using this approach will add no overhead on the network since it requires no feedback from the network (Jain, 1989). This approach assumes that there is a single connection that can utilize the whole network capacity, and hence increase/decrease the network delay (Jain, 1989). As we said before, this assumption is not always valid in real networks.

Biaz and Vaidya (1998) designed an error discriminator based on the CARD (Jain, 1989) congestion predictor. The discriminator uses the assumption used in CARD that if the network is not congested, then the rate of change in the delay will be zero. However, when the network starts building queues with the increase in the TCP window size, then the delay will change rapidly. The discriminator monitors the delay and the window size changes; if

both are increasing, then the drop is considered to be congestion drop; otherwise, the drops are considered transmission drop.

The results presented in Biaz and Vaidya (1998) indicate that the error discriminator based on the CARD predictor is poor in discriminating between error types (Biaz and Vaidya, 1998). Again, this is because of the assumption used in CARD that a single TCP window size will affect the network delay.

4.13. Error discriminator based on Tri-s throughput-based congestion predictor

The Tri-s (Wang and Crowcroft, 1991) congestion predictor proposed an approach to predict congestions in the link based on the throughput rather than errors. Its difference than CARD (Jain, 1989) approach is that Tri-s monitors only the changes in the connection throughput. Also, this approach tries to find the optimal window size only at the beginning of the connection and fix it through the rest of the connection period. Only when a major change in the connection happens, like when a new connection starts or an old connection terminates, the optimal window size is recalculated. Small changes during the connection are dealt with by buffering in the network instead of changing the sender window size (Wang and Crowcroft, 1991).

An error discriminator based on this idea has been proposed in Biaz and Vaidya (1998). This is based on the assumption that if the network is free of congestion, then the connection throughput will increase rapidly, and hence any drop will be considered to be a transmission drop. However, if there is congestion in the network, the TCP throughput will decrease and any error will be considered to be a congestion drop. The results presented in Biaz and Vaidya (1998) show a poor discrimination level and this is for the same reasons mentioned before for the Vegas and CARD based error discriminator.

4.14. TCP-Veno

TCP-Veno (Peng and Liew, 2003) applies changes to the Vegas (Brakmo, O'Malley and Peterson, 1994) congestion predictors in order to differentiate between congestive states (Peng and Liew, 2003) and non-congestive states (Peng and Liew, 2003) of the connection. If a packet drop occurs during a congestive state, then it is considered a congestion drop; otherwise, it is considered transmission drop.

TCP-Veno estimates the number of packets buffered in the network and if this number exceeds a predefined threshold (three in this case), then the

system enters congestive state (Peng and Liew, 2003). It uses Vegas (Brakmo, O'Malley and Peterson, 1994) congestion predictors to estimate buffered packets, and instead of updating the congestion window based on this information like Vegas, it uses it to differentiate between errors and uses TCP AIMD to update the congestion window.

The other change TCP-Veno proposes is to reduce the rate at which the congestion window increases during the congestive state. So, instead of increasing the congestion window every RTT, the window is increased every other RTT if the system is in the congestive state (Peng and Liew, 2003).

Peng and Liew (2003) reported noticeable improvement (up to 80%) for TCP-Veno over TCP-Reno in different scenarios. However, TCP-Veno suffers from the bad performance of Vegas predictor mentioned before which may lead to classify errors wrongly.

An important feature of TCP-Veno is that it cuts the congestion window even for transmission errors by a fixed factor of 4/5 (Peng and Liew, 2003) which may reduce the effect of poor discrimination ability. We could not find any other error discriminator that uses a special action in case of transmission errors.

4.15. Receiver based error discriminators

Most of the previous solutions are based in the sender side of the connection. In the following, we will describe some solutions which are designed to be in the receiver side of the connection.

Biaz and Vaidya (1999) proposed a receiver based error discriminator that uses a heuristic method to discriminate between transmission and congestion losses. In this method, they assume that the lossy link will be always the bottleneck of the connection. For example, a low bandwidth last hop in a wired-wireless network. Hence, in the case of congestion all packets will be queued in the bottleneck in the wireless base station. So, when the base station sends the packets, they will travel back-to-back on the wireless link. As a result, the TCP receiver can compute the inter arrival time of the packets and use it to determine the cause of the drop.

For example, if we have Packets 1, 2 and 3, then in normal cases there will be T time between consecutive packets. However, if one packet is dropped, say Packet 2, then the time between Packets 1 and 3 will be at least $2T$. From that the receiver can know that a drop in the wireless links has occurred.

However, if Packet 2 was dropped before the base station because of congestion, then Packets 1 and 3 will probably be queued in the base station because the wireless link is the bottleneck, the time

between Packets 1 and 3 will be less than $2T$, and hence the receiver can recognize that this error is due to congestion error (Biaz and Vaidya, 1999).

The problem with this method is that it requires the wireless link to be the bottleneck (the one with the least bandwidth) (Biaz and Vaidya, 1999). Also, as we noticed from the example above, this method works only if the wireless link is the last hop in the path and directly before the TCP receiver, and also if a non-stop stream of data is being sent (bulk data) (Biaz and Vaidya, 1999). However, the simulation results in Biaz and Vaidya (1999) showed that by using this method TCP could discriminate between wireless and congestion errors, in most cases, as good as a perfect error discriminator, i.e. with accuracy around 100% of discriminating both types of errors.

A similar approach has been proposed in WTCP (Wireless Transmission Control Protocol) (Sinha *et al.*, 2002) but without the constraint that the base station should be the bottleneck. This is achieved by computing an average inter arrival time at the receiver (AvgT). When a drop occurs, instead of comparing with T we compare with average AvgT. If current inter arrival time is within a predefined threshold from AvgT, then the error is considered a transmission error; otherwise, it is considered a congestion error. A promising result has been reported in Sinha *et al.* (2002) after using this approach.

Another receiver based error discriminator is proposed in Zhang and Tsoussidis (2001) and called TCP-Real. TCP-Real uses the rate of receiving data at the receiver to detect congestion. It computes an expected receiving rate and an actual receiving rate based on the congestion window size and minimum RTT and current RTT. If the actual receiving rate is less than the expected, then the receiver signals the sender to increase its congestion window and if the expected rate is less than the actual the receiver signals the sender to reduce its congestion window (we can notice the similarity with TCP-Vegas (Brakmo, O'Malley and Peterson, 1994) which uses same concept but at the sender).

Because this method uses the receiver to calculate the congestion window size, it solves the problem when the return path is slower than the forward path by considering the available bandwidth on the forward path only (Zhang and Tsoussidis, 2001). Experimental results in Zhang and Tsoussidis (2001) show that TCP-Real improves TCP performance when compared to TCP-Reno and TCP-Tahoe specially with the increase in the error rate. However, TCP-Real does not define a clear action for transmission drops and seems to keep the congestion window open.

4.16. Fast recovery plus

Fast recovery plus (Li *et al.*, 2004) has introduced a modification to TCP fast retransmission (Jacobson, 1990) and fast recovery (Allman, Paxson and Stevens, 1999) algorithms so it can discriminate between congestion and transmission errors. The idea is simple; the TCP sender maintains a counter of how many times the fast retransmission-fast recovery module is called by duplicate acknowledgments before receiving a new acknowledgment. Li *et al.* (2004) assume that transmission errors will occur in small numbers per window of data compared to congestion errors. So, the counting of the number of fast retransmission-fast recovery events can give an indication of the error type. If this number exceeds a preset threshold, then the error is considered to be a congestion error; otherwise, it is considered a transmission error. Li *et al.* (2004) did not explain how to choose the error threshold in order to decide the error type and we assume it is a fixed one that will be chosen based on the system experimental results.

The results shown in Li *et al.* (2004) present a good improvement in TCP throughput when Fast Recovery plus is used. However, like previous error discriminators, this method does not consider an action in case of transmission errors.

4.17. Spike error discriminator

Tobe *et al.* (2000) did a series of experiments on UDP performance in the Internet and they noticed that most congestion drops occur during specific periods related to noticeable increase in the packet trip time from the sender to receiver. They call these periods spike-train periods (Tobe *et al.*, 2000) since spikes appear in the packet trip time graphs when congestions occur. These spikes were found highly correlated with congestion events and hence congestion drops (Tobe *et al.*, 2000).

Cen, Cosman and Voelker (2003) used this idea to design an error discriminator which uses spike-train periods (Tobe *et al.*, 2000). They define two states, the spike-state and, non-spike-state. In the spike state, the connection is considered in congestion state and any drop that occurs during this period is considered a congestion drop. During the non-spike-state any drop is considered a transmission drop (Cen, Cosman and Voelker, 2003). The system enters the spike-state if the packet trip time exceeded a threshold called $B_{spikestart}$ and ends when the packet trip time becomes below $B_{spikend}$ (Cen, Cosman and Voelker, 2003). These thresholds are computed dynamically according to current relative one way trip time (ROTT) reading as follows:

$$B_{\text{spikestart}} = \text{ROT } T_{\text{min}} + \alpha(\text{ROT } T_{\text{max}} - \text{ROT } T_{\text{min}}) \quad (1)$$

$$B_{\text{spikeend}} = \text{ROT } T_{\text{min}} + \beta(\text{ROT } T_{\text{max}} - \text{ROT } T_{\text{min}}) \quad (2)$$

Spike uses ROTT instead of round trip time (RTT) because it was designed for UDP applications where there is no acknowledgment, so the authors used the relative one way trip time and since the sender and receiver clock may vary the term relative is used.

The Spike (Cen, Cosman and Voelker, 2003) error discriminator performed well under different scenarios where congestion and transmission errors were present. It was able to achieve high link utilization. However, its accuracy of distinguishing between error types was moderate (around 50%) and this has led to increased congestion in several cases (Cen, Cosman and Voelker, 2003).

5. Conclusions and Recommendations

Our aim in this paper is to give an overview of the efforts to improve TCP performance in the presence of errors (congestion and transmission). Some of the main end-to-end solutions are presented here and more related solutions can be found in Biaz *et al.* (1997), Tae-eun, Songwu and Bharghavan (1999), Samaraweera (1999), Biaz and Vaidya (2003b), Wang and Kung (2001), Lim (2007), Alnuem (2009), and Alnuem, Mellor and Fretwell (2008). Table 1 shows some of the main features of the presented protocols (the meaning of the letters in 'Required changes' filed in Table 1 is as follows: s:sender, r:reciever, n:network, pf:packet format. Also, ED in the filed 'Protocol/Technique' means Error Discriminator).

Also, to show the effect of adding an error discriminator on TCP performance, in Fig. 1 we

present the performance comparison of TCP before and after adding an error discriminator named TWA (Alnuem, 2009) (Transmission Window Action) in a semi-log scale. This gives an example of the noticeable effect of adding an error discriminator on TCP performance. In Fig. 1, the Goodput is normalized by each flow fair share of the bottleneck bandwidth.

We explained several protocols in this paper, in all of these protocols the main aim was to improve TCP performance when congestion and transmission errors coexist. However, we can categorize these solutions into two categories depending on how they solve the problem. The first category tries to distinguish between congestion and transmission errors and apply different actions for each case. All error discriminators like TCP-Casablanca (Biaz and Vaidya, 2005) come under this category. We will call them two actions solutions because in concept they can apply different actions at each case (i.e., congestion or transmission drops).

On the other hand, other solutions apply one action which can only detect and response to congestion and will do nothing if there are no congestion drops (and only if there are transmission errors). This kind of solutions usually applies techniques which by nature respond to congestion only. For example, TCP-Vegas (Brakmo, O'Malley and Peterson, 1994) which uses expected and actual throughput to set the congestion window or TCP-Westwood (Mascolo *et al.*, 2001) which uses Bandwidth-Delay product to set the congestion window size which will be affected mainly by the change in the available bandwidth due to the congestion in the network. These solutions do not differentiate between error types but only respond to congestion (by increasing the sending rate if there is no congestion and decreasing the sending rate if there is congestion) so we call them the one action solutions.

Table 1. Comparison of some features in protocols presented in this paper

Protocol/Technique	Action-congestion	Action-transmission	Required Changes
Vegas	Additive decrease	Not available	s + r
Westwood	BWE * MinRTT (pipe size)	Not available	s
Casablanca	Multiplicative decrease	Not available	s + r + n + pf
I-frane	Multiplicative decrease	Not available	s, n, pf
ECN	Multiplicative decrease	Not available	s, n, pf
Jeresy	Delay-Bandwidth + ECN	Not available	s, n, pf
ED based on Vegas	Multiplicative decrease	Not available	s
ED based on CARD	Multiplicative decrease	Not available	s
ED based on Tri-s	Multiplicative decrease	Not available	s
Veno	Multiplicative decrease	Cut by factor (4/5)	s
Biaz (receiver)	Multiplicative decrease	Not available	r
WTCP	Based on receiver estimation of congestion level	Not available	r
TCP-Real	Based on expected and actual receiving rate	Not available	r
Spike	Multiplicative decrease	Not available	s

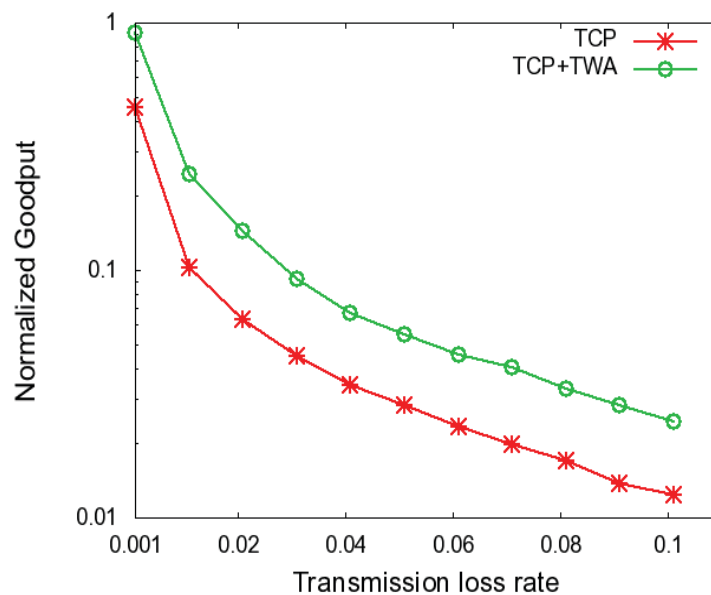


Fig. 1. TCP and TWA semi-log scale normalized Goodput.

However, in both one action and two actions solutions, the TCP reaction to transmission errors is simply not to cut the congestion window and to keep the sending rate as it was before the error. Moreover, in the two action solutions, when the protocol discovers transmission errors, it implicitly implies that it should increase the congestion window (not just do nothing).

These assumptions give rise to a question about whether the transmission action in current error discriminators is enough or not. Authors like Biaz and Vaidya (2005) indicated that the current transmission action used in error discriminators is a bad one. This is because it is simplistic and it ignores two facts. First, it is very hard to have an end-to-end error discriminator with very high accuracy. Second, even with accurate error discriminators mismatches between error types can occur. Because of that, some studies like Biaz and Vaidya (2003b, 2005), and Cen, Cosman and Voelker (2003) indicated that error discriminators usually increase the congestion loss rate noticeably.

Moreover, even the one action solutions can be affected by the lack of appropriate transmission action. This could happen when the technique used to discover congestion in the network fails to do so and hence no action is taken in case of congestion.

There should be an extended study of the effects of the lack of action in case of transmission errors on current error discriminators and on the network.

As we said before, some studies noticed increase in the congestion level in the network when using

some error discriminators which we believe related to the lack of action in the case of transmission errors. We recommend that current error discriminators should use a set of actions in the case of transmission errors where these actions should provide the following:

- These actions should be able to achieve the aim of any error discriminator which is to improve TCP performance when congestion and transmission errors coexist.
- These actions should be able to prevent increasing the congestion in the network which may occur because of the first aim.

We hope that having these addition actions in any error discriminator will create a balance between the need to improve TCP performance and the need to prevent congestion in the network.

In this work, we wanted to shed the light on the need of such actions. In other works like Alnuem, Mellor and Fretwell (2008), we explain our vision of how these actions should be designed to achieve the two aims mentioned above.

References

- Allman, M.; Paxson, V. and Stevens, W. "TCP Congestion Control." RFC 2581, (1999).
- Alnuem, M. A. "Improving TCP Performance Over Heterogeneous Networks." *PhD Thesis, University of Bradford*, (2009).
- Alnuem, M.; Mellor, J. and Fretwell, R. "TCP Multiple Drop Action for Transmission Errors." *The 9th Annual Postgraduate Symposium on the Convergence of*

- Telecommunications, Networking and Broadcasting (PGnet2008)*, (2008).
- Ayanoglu, E.; Paul, S.; LaPorta, T. F.; Sabnani, K. K. and Gitlin, R. D.** "Airmail: A Link-layer Protocol for Wireless Networks." *Wireless Networks*, Vol. 1, No. (1), (1995), 47-60.
- Bakre, A. V. and Badrinath, B. R.** "Implementation and Performance Evaluation of Indirect TCP." *IEEE Transactions on Computers*, Vol. 46, No. (3), (1997), 260-278.
- Balakrishnan, H.; Seshan, S.; Amir, E. and Katz, R. H.** "Improving TCP/IP Performance Over Wireless Networks." *Proceedings of the 1st ACM International Conference on Mobile Computing and Networking (Mobicom)*, (1995).
- Balakrishnan, H.; Padmanabhan, V. N.; Seshan, S. and Katz, R. H.** "A Comparison of Mechanisms for Improving TCP Performance Over Wireless Links." *Proceedings on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Palo Alto, California, United States, ACM Press, (1996), 256-269.
- Biaz, S. and Vaidya, N.** "Distinguishing Congestion Losses from Wireless Transmission Losses: A Negative Result." *Proceedings of the 7th International Conference on Computer Communications and Networks*, (1998), 722-731.
- Biaz, S. and Vaidya, N.** "Discriminating Congestion Losses from Wireless Losses Using Inter-arrival Times at the Receiver." *Proceedings of Application-specific Systems and Software Engineering and Technology Conference*, (1999), 10-17.
- Biaz, S. and Vaidya, N.** "Is the Round-trip Time Correlated with the Number of Packets in Flight?" *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement (IMC03)*, (2003a).
- Biaz, S. and Vaidya, N.** "Differentiated Services: A New Direction for Distinguishing Congestion Losses from Wireless Losses." Technical Report No. CSSE03-02, University of Auburn, (2003b).
- Biaz, S. and Wang, X.** "Can ECN Be Used to Differentiate Congestion Losses from Wireless Losses?" Technical Report CSSE04-04, Auburn University, (2004).
- Biaz, S. and Vaidya, N.** "De-randomizing Congestion Losses to Improve TCP Performance Over Wired-wireless Networks." *IEEE/ACM Transaction in Networking*, Vol. 13, No. (3), (2005), 596-608.
- Biaz, S.; Mehta, M.; West, S. and Vaidya, N.** "TCP Over Wireless Networks Using Multiple Acknowledgments." Technical Report No. 97-001, Texas A & M University, (1997).
- Brakmo, L. S. and Peterson, L. L.** "TCP Vegas: End-to-end Congestion Avoidance on a Global Internet." *IEEE Journal on Selected Areas in Communications*, Vol. 13, No. (8), (1995), 1465-1480.
- Brakmo, L. S.; O'Malley, S. W. and Peterson, L. L.** "TCP Vegas: New Techniques for Congestion Detection and Avoidance." *Proceedings of the Conference on Communications, Architectures, Protocols and Applications*, London, United Kingdom, ACM Press, (1994), 24-35.
- Brown, K. and Singh, S.** "M-TCP: TCP for Mobile Cellular Networks." *ACM SIGCOMM Computer Communication Review*, Vol. 27, No. (5), (1997), 19-43.
- Cen, S.; Cosman, P. and Voelker, G.** "End-to-end Differentiation of Congestion and Wireless Losses." *IEEE/ACM Transactions on Networking*, Vol. 11, No. (5), (2003), 703-717.
- Dawkins, S.; Glover, D.; Griner, J.; Tran, D.; Henderson, T.; Heidemann, J.; Touch, J.; Kruse, H.; Ostermann, S.; Scott, K. and Semke, J.** "Ongoing TCP Research Related to Satellites." RFC 2760, (2000).
- Elaarag, H.** "Improving TCP Performance Over Mobile Networks." *ACM Computing Surveys (CSUR)*, Vol. 34, No. (3), (2002), 357-374.
- Fall, K. and Floyd, S.** "Simulation-based Comparisons of Tahoe, Reno and SACK TCP." *ACM SIGCOMM Computer Communication Review*, Vol. 26, No. (3), (1996), 5-21.
- Floyd, S.** "TCP and Explicit Congestion Notification." *ACM Computer Communication Review*, Vol. 24, No. (5), (1994), 10-23.
- Floyd, S. and Henderson, T.** "The NewReno Modification to TCP's Fast Recovery Algorithm." RFC 2582, (1999).
- Floyd, S. and Jacobson, V.** "Random Early Detection Gateways for Congestion Avoidance." *IEEE/ACM Transaction on Networking*, Vol. 1, No. (4), (1993), 397-413.
- Gerla, M.; Pau, G.; Sanadidi, M. Y.; Wang, R.; Mascolo, S.; Casetti, C. and Lee, S.** "TCP Westwood: Enhanced Congestion Control for Large Leaky Pipes." *NASA Workshop*, (June 25, 2001).
- Grieco, L. and Mascolo, S.** "Performance Evaluation and Comparison of Westwood+, New Reno, and Vegas TCP Congestion Control." *ACM SIGCOMM Computer Communication Review*, Vol. 34, No. (2), (2004), 25-38.
- Hengartner, U.; Bolliger, J. and Gross, T.** "TCP Vegas Revisited." *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2000)*, Vol. 3, (2000), 1546-1555.
- Hoe, J. C.** "Improving the Start-up Behavior of a Congestion Control Scheme for TCP." *Conference Proceedings on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Palo Alto, California, United States, ACM Press, (1996), 270-280.
- Jacobson, V.** "Congestion Avoidance and Control." *Symposium Proceedings on Communications Architectures and Protocols*, Stanford, California, United States, ACM Press, (1988), 314-329.
- Jacobson, V.** "Modified TCP Congestion Avoidance Algorithm." E-mail sent to end2end-interest mailing list, (1990).
- Jacobson, V. and Braden, R. T.** "TCP Extensions for Long-delay Paths." RFC 1072, (1988).
- Jacobson, V.; Braden, R. and Borman, D.** "TCP Extensions for High Performance." RFC 1323, (1992).
- Jain, R.** "A Delay-based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks." *ACM SIGCOMM Computer Communication Review*, Vol. 19, No. (5), (1989), 56-71.
- Li, X.; Wu, J.; Cheng, S. and Ma, J.** "Performance Enhancement of Transmission Control Protocol (TCP) for Wireless Network Applications." United States Patent 6757248, (2004).
- Lim, C.** "An Adaptive End-to-end Loss Differentiation Scheme for TCP Over Wired-wireless Networks." *International Journal of Computer Science and Network Security (IJCSNS)*, Vol. 7, No. (3), (2007), 72.
- Mascolo, S.; Casetti, C.; Gerla, M.; Sanadidi, M. Y. and Wang, R.** "TCP Westwood: Bandwidth Estimation for Enhanced Transport Over Wireless Links." *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, Rome, Italy, ACM Press, (2001), 287-297.
- Mathis, M.; Mahdavi, J.; Floyd, S. and Romanow, A.** "TCP Selective Acknowledgement Options." RFC 2018, (1996).
- Parsa, C. and Garcia-Luna-Aceves, J. J.** "Improving TCP Performance Over Wireless Networks at the Link Layer."

- Mobile Networks and Applications*, Vol. 5, No. (1), (2000), 57-71.
- Peng, F. C. and Liew, S. C.** "TCP Veno: TCP Enhancement for Transmission Over Wireless Access Networks." *IEEE Journal on Selected Areas in Communications*, Vol. 21, No. (2), (2003), 216-228.
- Postel, J.** "Transmission Control Protocol." RFC 793, (1981).
- Samaraweera, N. K. G.** "Non-congestion Packet Loss Detection for TCP Error Recovery Using Wireless Links." *IEE Proceedings in Communications*, Vol. 146, No. (4), (1999), 222-230.
- Sinha, P.; Nandagopal, T.; Venkitaraman, N.; Sivakumar, R. and Bharghavan, V.** "WTCP: A Reliable Transport Protocol for Wireless Wide Area Networks." *Wireless Networks*, Vol. 8, No. (2/3), (2002), 301-316.
- Stevens, W.** "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms." RFC 2001, (1997).
- Tae-eun, K.; Songwu, L. and Bharghavan, V.** "Improving Congestion Control Performance through Loss Differentiation." *Proceedings of the 8th International Conference on Computer Communications and Networks*, (1999), 412-418.
- Tobe, Y.; Tamura, Y.; Molano, A.; Ghosh, S. and Tokuda, H.** "Achieving Moderate Fairness for UDP Flows by Path-status Classification." *Proceedings of the 25th Annual IEEE Conference on Local Computer Networks*, (2000), 252-261.
- Wang, S. and Kung, H.** "Use of TCP Decoupling in Improving TCP Performance Over Wireless Networks." *Wireless Networks*, Vol. 7, No. (3), (2001), 221-236.
- Wang, Z. and Crowcroft, J.** "A New Congestion Control Scheme: Slow Start and Search (Tri-s)." *ACM SIGCOMM Computer Communication Review*, Vol. 21, No. (1), (1991), 32-43.
- Xu, K.; Tian, Y. and Ansari, N.** "Improving TCP Performance in Integrated Wireless Communications Networks." *Computer Networks: The International Journal of Computer and Telecommunications Networking*, (2005).
- Zhang, C. and Tsaoussidis, V.** "TCP-Real: Improving Real-time Capabilities of TCP Over Heterogeneous Networks." *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, Port Jefferson, New York, United States, ACM Press, (2001), 189-198.

مراجعة موسعة لتقنيات تحسين بروتوكول الـ TCP

محمد عبدالله النعيم

قسم علوم الحاسب، كلية علوم الحاسب والمعلومات،
جامعة الملك سعود، ص ب ٥١١٧٨، الرياض ١١٥٤٣، المملكة العربية السعودية
malnuem@ksu.edu.sa

(قدم للنشر في ١/٧/٢٠٠٩م؛ وقبل للنشر في ٣٠/١٢/٢٠٠٩م)

ملخص البحث. يعد بروتوكول التحكم بالإرسال (TCP) واحداً من أهم بروتوكولات الاتصال في الشبكة العالمية (الإنترنت)، حيث يحتوي على عدة آليات داخلية. ومن أهم هذه الآليات آلية التحكم بالازدحام والتي تساهم في منع حدوث ازدحامات غير مرغوب بها وذلك عن طريق تغيير سرعة الإرسال بناء على حالة الازدحام في الشبكة. ومع التطور الكبير في الشبكات هذه الأيام فقد أصبح من الطبيعي أن يعمل الـ (TCP) داخل شبكات تتميز بكثرة ضياع حزم البيانات أثناء الإرسال (مثل الشبكات اللاسلكية). تحدث المشكلة حين يخلط الـ (TCP) بين الأخطاء في حالة الإرسال والأخطاء بسبب الزحام مما يترتب عليه تباطؤ غير مبرر في سرعة الإرسال. في هذه الورقة أقوم بعرض أهم الحلول لمشكلة تباطؤ أداء بروتوكول الـ (TCP) عندما يستخدم على الشبكات ذات معدل الأخطاء العالي. خلال الفترة السابقة تم اقتراح الكثير من الحلول لهذه المشكلة، وسنركز في هذه الورقة على الحلول التي تعالج المشكلة عن طريق أطراف الاتصال دون الحاجة لإحداث تغيير في برامج الشبكة.