J. King Saud University, Vol. 22, Comp. & Info. *Sci*., pp. 45-61,Riyadh (1431H./2010)

An Extended Review of Techniques for Enhancing TCP Performance

Mohammed A. Alnuem
Dept. of Information System, College of Computer and Information Sciences,
King Saud University
P.O. Box 51178 Riyadh 11543, Saudi Arabia
Email: malnuem@ksu.edu.sa

(Received 01/07/2009; 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 per- formance 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 in [1] divided the solutions into two general categories: 1- So- lutions 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[2]. 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 [3] divide these solutions into the fol- lowing 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 oc- cur 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 solu- tions 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 [3, 1]. 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[4], Snoop[2] and AIRMAIL[5].

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 [3][6]. 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 [6] and M-TCP [7].

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 (con- gestion and transmission) efficiently. Some of these solutions were designed for a wireless environment and some were introduced before 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 conges-tion drops. Later we will present techniques designed for congestion and trans- mission 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 [8, 9].

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 as- sumption that errors due to segment damage are rare (less than 1% of the sent segments [9]) and so it is

assumed that most of the segment loss is because of congestion [10][9]. 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. Ac- cording to Jacobson [11] 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 behaviour can be explained if we return to the combined slow start congestion avoidance algorithm explained in Stevens [10] and Allman[12]. In this algorithm Stevens [10] 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 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 [11] suggests that TCP can use the knowledge brought by duplicate acknowledgment to resend the lost packet (a fast retrans- mission) and then there is no need to enter slow start because the duplicate acknowledgments indicate that these packet have left the network and there is more space for new packets to be injected into the network [12] so no need to reduce the cwnd to one segment by entering slow start. Instead, TCP enters con- gestion 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 [10]. 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 [10].

3.3 Fast retransmission phase (TCP-NewReno)

When Stevens [10], in the RFC 2001, explained the slow start algorithm, he indi- cated that the first step in the algorithm is to initialize the slow start threshold variable ssthresh to a high value (65535 bytes). Also Allman [12], in RFC 2581, indicated that ssthresh could be set to an arbitrary high value. However, Heo [13] 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 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 [13] and this will create problems to the TCP fast retransmission mechanism proposed by Jacobson [11].

The fast retransmission algorithm [11] 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 [13].

To understand why the fast retransmission algorithm cannot recover from multiple drops, Stevens [10] indicated that the fast retransmission algorithm is terminated whenever a new acknowledgment is received. This new acknowledg- ment 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, Heo [13], suggested a change in the fast retransmission al- gorithm 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 et al. [14] call the intermediate acknowledgments the partial acknowledgments.

A new variation of TCP was proposed based on these modifications and called TCP-NewReno [14]. Also, Floyd et al. [14] has introduced two options of NewReno regarding when to reset the retransmission timeout: the first option is called slow-but-steady NewReno and the 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 [14] 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 ini- tially by Braden and Jacobson in [15]. However, detailed implementation and improvements to the idea were proposed later by Mathis et al. in [16].

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 [8]. 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 acknowl- edgment 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 [16].

Also using selective acknowledgment will allow TCP to resend all lost pack- ets without the need to do unnecessary retransmission of packets already re- ceived [17]. Using selective acknowledgment does not require the overhead of extra traffic since it is sent over normal acknowledgments [16][17].

However, a disadvantage of the implementation explained in [16] 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 requires 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 [16]. 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 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 receipt of duplicate copies of a packet by simply comparing the packet sequence number with the last acknowledged packet number [8]. 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 acknowledg- ment 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 [18] technique (CARD stands for Congestion Avoidance using Round Trip Delay) or the connection throughput in the case of the Tri-s scheme [19] and the Vegas scheme [20] 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. Following, brief explanations of some congestion predictors.

4.2 TCP-Vegas

TCP-Vegas [20, 21] is a modification to the congestion control mechanism in standard TCP [9, 11]. 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 [20], 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% [20].

We will give here an extended explanation of TCP-Vegas because of its im- portance 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 [20]. 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 [20]. 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 [20].

The other area in which TCP-Vegas provides changes is in congestion avoid- ance: TCP-Vegas has made dramatic changes to the congestion avoidance mech- anism used in TCP by making TCP to increase/decrease the sending rate, not based on packet drops as in TCP, but based on 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 decreases the window size [20]. 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 [9] 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 chance to measure the actual throughput between updating window size [20].

Hengartner et al. in [22] 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 retransmis- sion when its new timeout mechanism expires even before receiving duplicate acknowledgments.

However, the results in [22] showed that TCP-Vegas suffers from performance degradation when it coexists with versions of TCP that use the AIMD mecha- nism 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 some- times has a negative impact on the performance [22]. Also, we will see later how the authors in [23] have confirmed this fact (i.e. Vegas predictor poor perfor- mance) when we talk about using the Vegas congestion predictor in an error discriminator.

4.3 TCP-westwood

Mascolo et al. in [24] proposed a modification to the congestion avoidance al- gorithm used in TCP-Reno, which uses duplicate acknowledgment and timeout as an indicator for congestion and to update the sender window [9]. However, duplicate acknowledgments do not give indication of the type of the error (con- gestion or transmission error). For this reason Mascolo et al. [24] suggested that the TCP sender should do continuous estimation of the bandwidth and update the window size according to that estimation [24]. 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 re- flects available link capacity in the forward path [24].

Also TCP-Westwood [24] 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 [24] uses the estimated bandwidth-delay product to set the sender window according to the current congestion level [25].

The authors of TCP-Westwood [24] reported big improvements in TCP per- formance, especially over networks suffering from transmission errors like Wired- Wireless networks [24]. This improvement has been confirmed by Grieco & Mas-colo in [26]. Also the experimental results reported in both [24] and in [26] showed that TCP-Westwood has maintained fair sharing of the bandwidth and it does not lead to starvation of TCP-Reno connections.

However Biaz et al.[27] 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 estima- tion in TCP-Westwood is based on taking the average rate of received acknowl- edgments 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 under- stand 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 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 conges- tion notification through RED [28] (Random early dropping) queues. 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).

Following I will explain briefly some network dependent error discriminators.

4.6 TCP-casablanca

The key idea TCP-Casablanca introduces [27] 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” [27] 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[27].

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 [27].

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 [27]. In case of congestion error TCP cut the congestion window, otherwise it only resend the packet and does not cut the congestion window.

So, as we can see 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 [14] 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 perfor- mance [27]. 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 [27] is a sender based version of TCP-Casablanca [27]. 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 wireless error [27]. 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 [27]. However, the effect of TCP-Ifrane's accuracy was not studied by the authors in [27].

4.8 Explicit congestion notifications

Explicit congestion notification [29] was first introduced to help TCP avoid con- gestion 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 Man- agement (AQM) mechanism (e.g. RED [28]) 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. in [30] has proposed the use of ECN to improve TCP per- formance over wireless links by modifying the way TCP responds to ECN. Biaz [31] explains 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 conges- tion 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, its then considered as a wireless error and TCP does not reduce the senders window size [31]. How- ever, we still need to retransmit the lost packet. The authors in [30] argue that this approach will improve TCP performance over networks with transmission errors like wireless networks specially those suffers from high error rates.

However, Biaz in [31] studied the possibility of using ECN to distinguish between error types. He argues 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[31].

So the authors in[31] 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 congestion drop. The new protocol is called TCP-Eaglet [31] and it showed improvement over standard TCP performance.

4.9 TCP-jersey

Xu et al. [32] has suggested using the estimated bandwidth instead of errors to tell TCP when to decrease sender window size, which is an idea similar to Westwood [24] 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 [32]. So this method is a combination of TCP-Westwood and ECN error discrimina- tor [30] 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 re- verse path can delay the acknowledgments, so it will underestimate the available bandwidth.

However, improvement has been done to TCP-Jersey to overcome this prob- lem. The improved version is called TCP-New Jersey [32] and it uses acknowl- edgment timestamps [33] instead of acknowledgment arrival rate to calculate estimated bandwidth which solves the problem of delayed acknowledgments be- cause each acknowledgment has a time stamp which allows the sender to cal- culate 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 [32].

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 Ve- gas [20] or CARD [18] or Tri-s [19]. 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 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 link error [23] .

Also we must notice that as [23] indicated, designing an accurate error pre- dictor 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 [23]. 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 [23] on three different error discriminators based on congestion predictors: the CARD [18], Tri-s [19] and Vegas [20]. Unfortunately the results obtained by Biaz and Vaidya experiments in [23] show that these congestion predictors are no better than a random loss predictor. From these results, Biaz came to the conclusion that these three con- gestion 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 small part of the expected network throughput then this means that a congestion exists.

However, in [34] the authors 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 [34]. This is because usually a single connection forms a small part of the network aggregate traffic.

However, the authors in [34] also emphasized on the sensitivity of the net- work 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 [20] described earlier, Biaz and Vaidya [23] pro- posed an error discriminator that computes the difference between expected throughput (link capacity) and the actual throughput in order to predict con- gestion and use this difference to define a new variable fVegas. The difference is computed as follows: D = expected throughput - 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 [23] 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 [23] and this will affect the error discriminator ability to discover congestion errors.

This also applies to the next two error discriminators based on CARD [18] and Tri-s [19] congestion predictors.

4.12 Error discriminator based on card delay-based congestion

Predictor Congestion Avoidance Round trip Delay (CARD) [18] is an approach to up- date the TCP sender window size without the need to have any feedback from the network. It is called [18] a black-box approach since it deals with the net- work as a black box and does not require any explicit feedback from the net- work. 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. The authors in [18] call it maximum Power where the power is the ratio of throughput and delay : P ower = (T hroughput/Delay) [18]. 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 [21]. However Jain[18] did not provide a complete TCP solution like TCP-Vegas, instead, it gives a mechanism that can be used to replace Jacobson's [9] congestion avoidance mechanism in TCP.

The CARD [18] 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 [18]. This approach assumes there is a single con- nection that can utilize the whole network capacity and hence increase/decrease the network delay [18]. As we said before this assumption is not always valid in real networks.

Biaz et al. [23] designed an error discriminator based on the CARD [18] 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 is considered transmission drop.

The results presented in [23] indicate that the error discriminator based on the CARD predictor is poor in discriminating between error types [23]. Again this 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 [19] congestion predictor proposed an approach to predict conges- tions in the link based on the throughput rather than errors. Its difference than CARD [18] 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 [19].

An error discriminator based on this idea has been proposed in [23]. This is based on the assumption that if the network is free of congestion then the con- nection 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 conges- tion drop. The results presented in [23] 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 [35] applies changes to the Vegas [20] congestion predictors in order to differentiate between congestive states [35] and non congestive states [35] 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 (3 in this case) then the system enters congestive state [35]. It uses Vegas [20] 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 [35].

The authors in [35] 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 [35] 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. Following we will describe some solutions which are designed to be in the receiver side of the connection.

In [36] the authors proposed a receiver based error discriminator that uses a heuristic method to discriminate between transmission and congestion losses. In this method the authors assume that the lossy link will be always the bottle- neck 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 bot- tleneck 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 packet 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 conges- tion, then packets 1 and 3 will probably be queued in the base station because the wireless link is the bottleneck, the time between packet 1 and 3 will be less than 2T and hence the receiver can recognize that this error is due to congestion error

[36].

The problem with this method is that it requires the wireless link to be the bottleneck (the one with least bandwidth) [36]. 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)[36]. However, the simulation results in [36] 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) [37] 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 [37] after using this approach.

Another receiver based error discriminator is proposed in [38] 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 [20] 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 that the forward path by considering the available bandwidth on the forward path only [38]. Experi- mental results in [38] shows 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 [39] has introduced a modification to TCP fast retransmission [11] and fast recovery [12] 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. The authors in [39] assumes 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. The author in [39] 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 [39] presents 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

The authors in [40] did a series of experiments on UDP performance in the Internet and they noticed that most congestion drops occur during specific pe- riods related to noticeable increase in the packet trip time from the sender to receiver. They call these periods spike-train periods [40] 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 [40].

The authors in [41] used this idea to design an error discriminator which uses spike-train periods [40]. 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 [41]. 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 $B_{spikend}$ [41]. These thresholds are computed dynamically according to current relative one way trip time (ROTT) reading as follows:

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

$$B_{spikend} = ROTT_{min} + \beta(ROTT_{max} - ROTT_{min}) \quad (2)$$

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 [41] 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 [41].

5. Conclusions and Recommendations

Our aim in this paper is to give an overview of the efforts to improve TCP performance in 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 [42–47]. Table 1 shows some of tha 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 [48] (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 figure 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 [27] 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 detects and response to congestion and will do nothing if there are no congestion drops (and only there are transmission errors). These kind of solutions usually ap- ply techniques which by nature respond to congestion only. For example, TCP-Vegas [20] which uses expected and actual throughput to set the congestion window or TCP-Westwood [24] which uses Bandwidth-Delay product to set the congestion window size which will be affected mainly by the change in the avail- able bandwidth due to the congestion in the network. These solutions do not differentiate between error types but only respond to congestion (by increasing sending rate if there is no congestion and decrease the sending rate if there is congestion) so we call them the one action solutions.

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 [27] 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

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 availabl	s

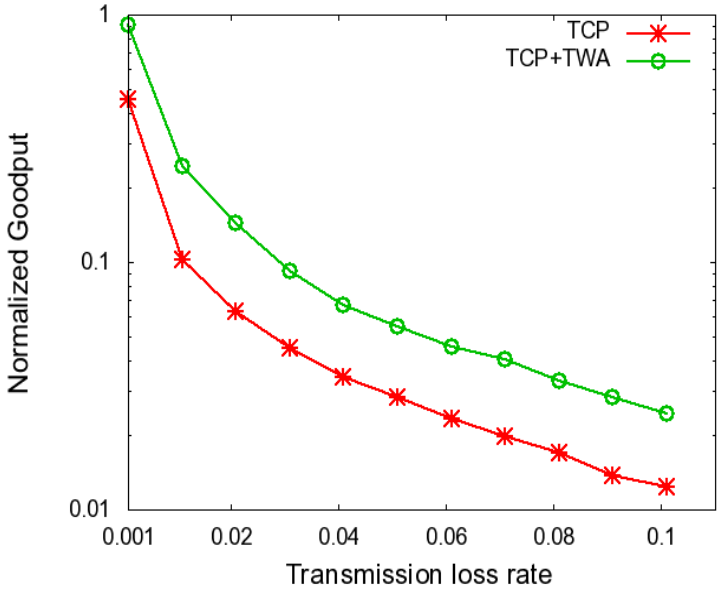

Fig. 1. TCP and TWA semi-log scale normalized goodput.

mismatches between error types can occur. Because of that some studies like [41, 27, 45] indicated that error discriminators usually increase the congestion loss rate noticeably.

Moreover, even the one action solutions can be affected by the lack of ap- propriate 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 [49] we explain our vision of how these actions should be designed to achieve the two aims mentioned above.

References

1. Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, and Randy H.Katz. A comparison of mechanisms for improving tcp performance over wirelesslinks. In Conference proceedings on Applications, technologies, architectures, and protocols for computer communications, Palo Alto, California, United States, 1996. ACM Press. pages 256-269.
2. H. Balakrishnan, S. Seshan, Amir E., and R. H. Katz. Improving TCP/IP perfor- mance over wireless networks. In Proceedings 1st ACM international conference on Mobile Computing and Networking (Mobicom), 1995.
3. H. Elaarag. Improving TCP performance over mobile networks. ACM Computing Surveys (CSUR), 34(3):357–374, 2002.
4. C. Parsa and J. J. Garcia-Luna-Aceves. Improving tcp performance over wireless networks at the link layer. Mobile Networks and Applications, 5(1):57–71, 2000.
5. E. Ayanoglu, S. Paul, T. F. LaPorta, K. K. Sabnani, and R. D. Gitlin. Airmail: a link-layer protocol for wireless networks. Wireless Networks, Kluwer Academic Publishers, 1(1):47–60, 1995.
6. A. V. Bakre and B. R. Badrinath. Implementation and performance evaluation of indirect TCP. IEEE Transactions on Computers, 46(3):260–278, 1997.
7. K. Brown and S. Singh. M-TCP: TCP for mobile cellular networks. ACM SIG- COMM Computer Communication Review, 27(5):19–43, 1997.
8. J. Postel. Transmission control protocol. RFC 793, 1981.
9. V. Jacobson. Congestion avoidance and control. In Symposium proceedings on Communications architectures and protocols, pages 314–329, Stanford, California, United States, 1988. ACM Press.
10. W. Stevens. TCP slow start, congestion avoidance, fast retransmit, and fast re- covery algorithms. RFC 2001, 1997.
11. Van Jacobson. Modified TCP congestion avoidance algorithm. email sent to end2end-interest mailing list, 1990.
12. M. Allman, V. Paxson, and W. Stevens. Tcp congestion control. RFC 2581, 1999.
13. J. C. Hoe. Improving the start-up behavior of a congestion control scheme for tcp.

In Conference proceedings on Applications, technologies, architectures, and proto- cols for computer communications, pages 270–280, Palo Alto, California, United States, 1996. ACM Press.

14. S. Floyd and T. Henderson. The NewReno modification to TCP's fast recovery algorithm. RFC 2582, 1999.
15. V. Jacobson and R. T. Braden. TCP extensions for long-delay paths. RFC 1072,1988.
16. M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. Tcp selective acknowledgement options. RFC 2018, 1996.
17. K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno and SACK TCP. ACM SIGCOMM Computer Communication Review, 26(3):5–21, 1996.
18. R. Jain. A delay-based approach for congestion avoidance in interconnected hetero- geneous computer networks. ACM SIGCOMM Computer Communication Review,

19(5):56–71, 1989.

19. Z. Wang and J. Crowcroft. A new congestion control scheme: slow start and search

(tri-s). ACM SIGCOMM Computer Communication Review, 21(1):32–43, 1991.

20. L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: new techniques for congestion detection and avoidance. In Proceedings of the conference on com- munications architectures, protocols and applications, pages 24–35, London, United Kingdom, 1994. ACM Press.
21. L. S. Brakmo and L. L. Peterson. Tcp vegas: end to end congestion avoidance on a global internet. IEEE Journal on Selected Areas in Communications, 13(8):1465–1480, 1995.
22. U. Hengartner, J. Bolliger, and T. Gross. Tcp vegas revisited. In Proceedings of Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 2000, volume 3, pages 1546–1555, 2000.
23. S. Biaz and N. Vaidya. Distinguishing congestion losses from wireless transmission losses: a negative result. In Proceedings of the Seventh International Conference on Computer Communications and Networks, pages 722–731, 1998.
24. S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang. TCP westwood: Bandwidth estimation for enhanced transport over wireless links. In Proceedings of the 7th annual international conference on Mobile computing and networking, pages 287–297, Rome, Italy, 2001. ACM Press.

25. M. Gerla, G. Pau, M. Y. Sanadidi, R. Wang, S. Mascolo, C. Casetti, and S. Lee.
TCP westwood: Enhanced congestion control for large leaky pipes. In NASA Workshop, 25 June 2001.
26. L. Grieco and S. Mascolo. Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control. ACM SIGCOMM Computer Com- munication Review, 34(2):25–38, 2004.
27. S. Biaz and N. Vaidya. De-randomizing congestion losses to improve TCP per- formance over wired-wireless networks. IEEE/ACM Transaction in Networking,
13(3):596–608, 2005.
28. S. Floyd and V. Jacobson. Random early detection gateways for congestion avoid- ance. IEEE/ACM Transaction on Networking, 1(4):397–413, 1993.
29. S. Floyd. Tcp and explicit congestion notification. ACM Computer Communication
Review, 24(5):10–23, 1994.
30. S. Dawkins, D. Glover, J. Griner, D. Tran, T. Henderson, J. Heidemann, J. Touch, H. Kruse, S. Ostermann, K. Scott, and J. Semke. Ongoing TCP research related to satellites. RFC 2760, 2000.
31. S. Biaz and X. Wang. Can ECN be used to differentiate congestion losses from wireless losses? Technical Report CSSE04-04, Auburn University, 2004.
32. K. Xu, Y. Tian, and N. Ansari. Improving TCP performance in integrated wire- less communications networks. Computer Networks: The International Journal of Computer and Telecommunications Networking, 2005.
33. V. Jacobson, R. Braden, and D. Borman. Tcp extensions for high performance.
RFC 1323, 1992.
34. S. Biaz and N. Vaidya. Is the round-trip time correlated with the number of packets in flight ? In Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement, IMC03, 2003.
35. Fu Cheng Peng and S. C. Liew. TCP Veno: TCP enhancement for transmission over wireless access networks. IEEE Journal on Selected Areas in Communications,
21(2):216–228, 2003.
36. S. Biaz and N. Vaidya. Discriminating congestion losses from wireless losses using inter-arrival times at the receiver. In Proceedings of Application-Specific Systems and Software Engineering and Technology Conference, pages 10–17, 1999.
37. P. Sinha, T. Nandagopal, N. Venkitaraman, R. Sivakumar, and V. Bharghavan.
WTCP: a reliable transport protocol for wireless wide-area networks. Wireless
Networks, 8(2/3):301–316, 2002.
38. C. Zhang and V. Tsaoussidis. TCP-real: improving real-time capabilities of tcp over heterogeneous networks. In Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video, pages 189–198, Port Jefferson, New York, United States, 2001. ACM Press.
39. X. Li, J. Wu, S. Cheng, and J. Ma. Performance enhancement of transmission control protocol (TCP) for wireless network applications. United States Patent
6757248, 2004.
40. Y. Tobe, Y. Tamura, A. Molano, S. Ghosh, and H. Tokuda. Achieving moderate fairness for udp flows by path-status classification. In Proceedings of the 25th Annual IEEE Conference on Local Computer Networks, pages 252–261, 2000.
41. S. Cen, P. Cosman, and G. Voelker. End-to-end differentiation of congestion and wireless losses. IEEE/ACM Transactions on Networking, 11(5):703–717, 2003.
42. S. Biaz, M. Mehta, S. West, and N. Vaidya. TCP over wireless networks using mul- tiple acknowledgments. Technical report, Texas A & M University, 1997. Report No. : 97-001.
43. K. Tae-eun, L. Songwu, and V. Bharghavan. Improving congestion control per- formance through loss differentiation. In Proceedings of the Eight International Conference on computer Communications and Networks, pages 412–418, 1999.
44. N. K. G. Samaraweera. Non-congestion packet loss detection for TCP error re- covery using wireless links. IEE Proceedings in Communications, 146(4):222–230,
1999.
45. S. Biaz and N. Vaidya. Differentiated services: A new direction for distinguishing congestion losses from wireless losses. Technical report, University of Auburn,
2003. Report No. : CSSE03-02.
46. S. Wang and H. Kung. Use of TCP decoupling in improving tcp performance over wireless networks. Wireless Networks, 7(3):221–236, 2001.
47. C. Lim. An adaptive End-to-End loss differentiation scheme for TCP over wired/wireless networks. IJCSNS,International Journal of Computer Science and Network Security, 7(3):72, 2007.
48. Mohammed A. Alnuem. Improving TCP Performance over Heterogeneous Net- works. PhD thesis, University of Bradford, 2009.
49. M. Alnuem, J. Mellor, and R. Fretwell. Tcp multiple drop action for transmission errors. In PGnet2008 The 9th Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting, 2008.

مراجعة موسعة لتقنيات تحسين بروتوكول الTCP

محمد عبدالله النعيم

قسم علوم الحاسب، كلية علوم الحاسب والمعلومات

جامعة الملك سعود، ص.ب: ٥١١٧٨، الرياض ١١٥٤٣، المملكة العربية السعودية

malnuem@ksu.edu.a

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

ملخص البحث. يعد برتوكول التحكم بالإرسال (TCP) واحدا من أهم بروتوكولات الإتصال في الشبكة العالمية (الإنترنت). حيث يحتوي على عدة آليات داخلية ومن أهم هذه الآليات آلية التحكم بالإزدحام والتي تساهم في منع حدوث ازدحامات غير مرغوب بها وذلك عن طريق تغيير سرعة الارسال بناء على حالة الازدحام في الشبكة. ومع التطور الكبير في الشبكات هذه الأيام فقد أصبح من الطبيعي أن يعمل ال (TCP) داخل شبكات تتميز بكثرة ضياع حزم البيانات أثناء الإرسال (مثل الشبكات اللاسلكية) . المشكلة تحدث حين يخلط ال (TCP) بين الأخطاء في 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 ال (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.