A PROTOCOL TO IMPROVE THE DATA COMMUNICATION OVER WIRELESS NETWORK

S Saravanan[1] and E Karthikeyan[2]

[1]Research Scholar, Dept of Computer Science, Bharathiar University, Coimbatore, India
ssam2020@gmail.com
[2]Asst. Professor of Computer Science, Government Arts College, Udumalpet, India
e_karthi@yahoo.com

ABSTRACT

Reliable transport protocols are tuned to perform well in traditional networks where packet losses occur mostly because of congestion. However, in networks with wireless links in addition to wired segments this assumption would be insufficient, as the high wireless bit error rate could become the dominant cause of packet loss. The main reason of this poor performance for TCP is that TCP cannot distinguish between packets losses due to wireless errors from those due to congestion. Moreover, TCP sender cannot keep the size of its congestion window at optimum level and always has to retransmit packets after waiting for timeout, which significantly degrades throughput and end-to-end performance of TCP.

In this work, a novel protocol, called DLN (Data Loss Notification), is be proposed. By changing the ACK format of TCP, we could successfully distinguish packet losses incurred by congestion and channel error which will improve both throughput and delay performance of TCP in wireless environment significantly. This mechanism has been demonstrated to provide performance improvements across a range of bit-error rates.

KEYWORDS

Data Loss Notification, Wireless Error, Network Congestion, Loss Recovery, Throughput

1. INTRODUCTION

TCP is a protocol developed on wired Internet, some of the algorithm is based on characteristics of wired network. One most important aspect is that TCP host may think packet loss as network congestion. This assumption is pretty reasonable in wired Internet, because the wired transmission media has quite a small packet loss ratio; the main reason for packet loss is network congestion.

The thing is not that right in the rapid developing wireless network. The media of wireless network has totally different characteristics. The wireless channel is an error prone media; the packet transmitted through wireless link may be lost due to wireless error. The TCP host may judge the packet lost due to network congestion, and then take the action of reducing size of the transmission window or waiting for long idle time for retransmitting the lost packet. The result is degraded end-to-end performance. In addition, packet losses that occur due to use mobility cause the TCP sender to remain idle for long periods of time even after the handoff is completed, resulting in unacceptably low throughput.

Link layer protocols are an alternative for improving the poor performance of TCP over wireless link. In those methods usually forward error connection (FEC) or automatic repeat request (ARQ) are used to improve the performance. Independent timer reaction at link and transport layers that may result in unnecessary retransmission, fast retransmission interaction, and large round-trip variation are considered as major problem with link-layer approaches. Split-

DOI : 10.5121/ijwmn.2011.3508

connection protocols attempt shield the sender from the wireless link by explicitly terminating the wired connection at the base station and using a separate transport connection over the wireless link. However, they do not preserve end-to-end semantics because data may be acknowledged to the sender even before it reaches the receiver, complicate handoff procedures because they involve hard state in the network, and do not usually provide the best possible performance. Another enhancement to TCP for wireless channel reviewed here is called Snoop protocol. In this method, the base station is equipped with a module called snoop agent, which its function is to monitor the TCP packets transmitted from a fixed host to a mobile host and vice versa. The agent caches all those packets locally and in the case of receiving duplicate acknowledgements (ACKs), retransmits the packets promptly and suppresses duplicate ACKs. The Snoop protocol performs retransmission of lost packets locally (at the base station) and hence avoids lengthy fast retransmission and congestion control at the sender side. By this method, end-to-end semantics of TCP is maintained and performance of TCP is improved. The Snoop protocol is mainly used for the fixed host to mobile host direction, explicit loss notification algorithms complementing the Snoop on the mobile host to fixed host direction.

Our approach, called the Data Loss Notification (DLN) protocol, explicit optimization to improve performance. We design a mechanism that can successfully distinguish between congestion and channel error-induced packet losses to substantially enhance end-to-end performance. The DLN protocol uses only soft state at agent in the network, which is periodically refreshed upon the arrival of data segments and ACKs.

Another aspect of Internet packet we analyse in this paper is that of packet delay. Delay variation is arguably the most complex element of network behaviour to analyse with loss, for example, the packet either shows up at the receiver or it does not, while with delay there are many shades of possibility and meaning in the time required for a packet to arrive. Likewise, delay variation is potentially the richest source of information about the network.

Small TCP transmission window prevent the sender from recovering from losses without incurring expensive timeouts that keep the connection idle for long periods of time. As a result, Delay for packet from sender host to receiver host is high due to the time for waiting for timeout and retransmission. Thus, a key challenge is in enhancing TCP's loss recovery algorithms when large wireless packet loss rate leads to numerous timeouts.

Based on an extensive analysis, we show that not only are current loss recovery techniques grossly inadequate at preventing sender timeouts, but that proposed enhancements like Selective Acknowledgements (SACK) are not likely to significantly change this because typical Internet transmission window are not larger than a few segments. Based on the results of our analysis, the DLN has the function of judge the reason of packet loss, which can efficiently retransmit lost packet without unnecessarily reducing the transmission window.

2. BACKGROUND AND RELATED WORK

The purpose of this section is to give an introduction of reliable transmission protocol and the application of reliable data transmission in wireless network. We begin in section 2.1 from discuss how reliable transport protocol developed, and then give a detailed description of various transport protocol discussed in the literature. In section 2.2, we give an introduction of Transmission Control Protocol, which is the main protocol used in the Internet. In section 2.3, we discuss several end-to-end TCP protocols developed in these years.

2.1 Reliable transport Protocols

Today's Internet is a best effort transmission network. Packet is transmitted in the network without guarantee of in order and reliable transmission, this is a big difference compared with traditional telephone transmission. The transmission host sends out the packet into the network,

then the packet passes through a series of routers to its destination, these routers determined the routing of the packet. Packet may be lost due to network congestion. While this architecture of Internet ensured a simple and effective Internet protocol, which expedited the rapid grow of Internet. It sends the mission of reliable packet transmission to the higher layer of protocols. Applications like the World Wide Web [1], file transfer [2], remote terminals and electronic mail [3] that need reliable and ordered data delivery require a transport protocol to provide this functionality, freeing them of the need to achieve reliability on a per-application basis.

Several reliable transport protocols have been proposed in the literature for best-effort networks: Delta-t [4], NETBLT [5], VMTP [6], OSI/TP4 [7], XTP [8] and TCP [9]. In all these protocols, data items are identified by sequence numbers that are either byte-based or packet-based. These sequence numbers are used to detect losses, reordering and data duplication. Packet-based sequence numbers are simple to implement but are less flexible because they often constrain the sender to use fixed-size packets. Byte-based sequence numbers indicate to the receiver the exact amount of missing data and permit the sender to precisely identify and retransmit lost data.

2.2 Transmission Control Protocol (TCP)

In the Internet today, TCP is now the standard for reliable data transport. Measurements made in 1999 show that over 95% of all bytes, 90% of all packets and 75% of all flows use TCP. This section discusses its salient features and highlights its main weaknesses.

While the original formal specification of TCP is in RFC793, numerous variants of it have been developed over the past several years, such as TCP-Tahoe, TCP-Reno, Vegas etc. This section discusses the TCP-Reno variant of TCP, which is the predominantly deployed version today.

2.2.1 Cumulative Acknowledgments

TCP is an ARQ-based reliable transport protocol that uses cumulative ACKs and byte based sequence numbers for reliability. TCP provides a fully reliable, in-order; byte-stream delivery abstraction to the higher-layer application, which typical uses a socket interface to interface with the transport layer. The basic unit of transmission is called segment, which is a contiguous sequence of bytes identified by its 32-bit long start and end sequence numbers. The transmitted segments are smaller than or equal to the connection's maximum segment size (MSS), which is negotiated at the start of the connection.

2.2.2 Loss Recovery

When the TCP sender discovers that data has been lost in the network, it recovers from it by retransmitting the missing segments. TCP has two mechanisms for discovering and recovering from losses: timer-driven retransmissions and data-driven retransmissions.

Time-driven recovery: When the TCP sender does not receive a positive cumulative ACK for a segment within a certain time-out interval, it retransmits the missing data. To determine the timeout interval, it maintains a running estimate of the connection's round-trip time using an exponential weighted moving average (EWMA) formula, srtt=a*rtt+(1-a)*srtt, where 'srtt' is the smoothed round-trip time average, rtt is the current round-trip sample, and 'a' the EWMA constant set to 0.125 in the TCP specification. It also estimates the mean linear deviation, 'rttbvar', using a similar EWMA filter, with a set to 0.25. A time-out occurs if the sender does not receive an ACK for a segment within 'srtt+4*rttvar' since the arrival of the last new cumulative ACK. Furthermore, the retransmission timer is exponentially backed off after each unsuccessful retransmission. The details of the round-trip time calculations and timer management can be found in [10,11,12].

Data-driven recovery: TCP's data-driven retransmission mechanism uses a technique called Fast Retransmission. It relies on the information conveyed by cumulative ACKs and takes advantage of the receipt of later data segments after a lost one. Because ACKs are cumulative, all segments after a missing one generates duplicate cumulative ACKs that are sent to the TCP sender. The sender uses these duplicate ACKs to deduce that a segment is missing and retransmits it.

However, the sender must not retransmit a segment upon the arrival of the very first duplicate ACK. This is because the Internet service model does not preclude the reordering of packets in the network; such reordering cause's later segments to be received ahead of earlier ones, and triggers duplicate ACKs in the same way that losses do. Furthermore, the degree of packet reordering on the Internet seems to be increasing, thus, to avoid prematurely retransmitting segments, the sender waits for three duplicate ACKs, the current standard fast retransmit threshold.

This is followed by the fast recovery phase, where additional packets are transmitted after the sender is sure that at least half the current window has reached the receiver, based on a count of the number of received duplicate ACK. Fast recovery ensures that a fast retransmission is followed by congestion avoidance and not by slow start. Since the arrival of duplicate ACKs signals to the sender that data is indeed flowing between the two ends, there is no reason to suddenly throttle the sender by invoking slow start.

2.2.3 Congestion Avoidance and Control

TCP's congestion management is based largely on Jacobson's seminar paper [10]. TCP uses a window-based algorithm to manage congestion, where the window is an estimate of the number of bytes currently unacknowledged and outstanding in the network.

The TCP sender performs flow control by ensuring that the transmission window does not exceed the receiver's advertised window size. It performs congestion control by using a window-based scheme, where the sender regulates the amount of transmitted data using a congestion window. When a connection starts or resumes after an idle period of time, slow start is performed. Here, the congestion window is initialized to one segment and every new ACK increases the window by one MSS. After a certain threshold (called the slow start threshold, 'ssthresh') is reached, the connection moves into the congestion avoidance phase, in which the congestion window effectively increases by one segment for each successfully transmitted window. In response to a packet loss, the sender halves its congestion window; if a timeout occurs, the congestion window is set to one segment and the connection goes through slow start once again.

2.3 End–to-End TCP Enhancements

Over the past decade, TCP has been tuned to work well in wired networks in the face of network congestion, reacting to congestion by reducing its rate, recovering from lost segments, and probing for bandwidth in a careful way. In this section, we survey some proposed enhancements to TCP that improve its ability to recover from losses in a timely manner and/or perform better congestion control.

2.3.1 Selective Acknowledgments (SACK)

It is well known that TCP performance suffers due to coarse-grained timeouts when multiple segments are lost in a single window because it uses only cumulative ACKs. There has therefore been recent interest in adding selective acknowledgements (SACK) to the standard TCP specification to reduce the time it takes to recover from multiple losses in a window.

TCP Selective Acknowledgements can be implemented in many ways. One possible approach is to use Keshav and Morgan's SMART (Selective Mechanism to Aid Retransmission) scheme, where the receiver communicates the segment number that just arrived in addition to the cumulative ACK, whenever it sends a duplicate ACK. A second approach, described in RFC2018, is currently on track to become an Internet standard. In this scheme, the receiver reports up to three of the last received, out-of-order, maximal contiguous blocks of data, in addition to the cumulative ACK, so that the sender can accurately deduce which segments have reached the receiver.

2.3.2 NewReno

Hoes' NewReno modification to TCP-Reno reduces the number of timeouts incurred by the TCP sender when multiple losses happen in a transmission window [13]. When multiple losses occur in TCP-Reno, a fast retransmission occurs after three duplicate ACKs arrive. Later duplicate ACKs are ignored until half the window is acknowledged, after which fast recovery sends a new segment for every incoming duplicate ACK. Now, when a new ACK arrives after the successful fast retransmission, its value will be within the original window (recall that there were multiple losses in the original window). In many cases TCP-Reno would time out for this second loss if the second loss is "close" to the location of the first one, because the sender's window would now have been reduced to half its original value and the sliding window not have shifted the original window to beyond the original right edge.

In NewReno, however, the sender remains in fast recovery when this happens, when the new ACK arriving after a fast retransmission is partial. A partial ACK is defined as a cumulative ACK that does not acknowledge the original window completely. By remaining in fast recovery, the sender continues to send new segments, which would elicit more duplicate ACKs and eventually trigger another fast retransmission without incurring a timeout.

2.3.3 Forward Acknowledgments

Mathis and Mahdavi's TCP with forward Acknowledgments (FACK) [14] uses SACK information at the sender to perform better congestion management. Rather than assume that the receiver on every duplicate ACK has received a MSS-worth of data, FACK calculates this precisely using information from the SACK field. It explicitly maintains an "available window" variable, awnd, that keeps track of the number of bytes that are unacknowledged, to perform better congestion control. As long as awnd is smaller than cwnd, the sender is permitted to transmit more data.

2.3.4 NetReno

Concurrent with our work, Lin and Kung [15] discuss some mechanisms for improving TCP loss recovery and propose some modifications to TCP. They argue that their modifications are more sensitive to network conditions than current TCP is motivating the name NetReno (for "Network Sensitive Reno").

2.3.5 Snoop Protocol

Snoop protocol improves TCP performance by deploying an agent at the base station. The agent mainly performs the functions of loss detection and loss recovery via retransmission by taking advantage of the information conveyed in TCP acknowledgments (ACKs) from the receiver. For transfer of data from a fixed host to a mobile host, the snoop agent used the loss indications conveyed by duplicate TCP ACKs and locally maintained timers to retransmit loss data from the base station. The agent also suppresses duplicate ACKs corresponding to wireless losses from the TCP sender, Thereby preventing unnecessary congestion control invocations.

Snoop protocol can suppress duplicate acknowledgment for TCP segments lost and retransmitted locally, thereby avoiding unnecessary fast retransmission and congestion control invocations by the sender. Also there are some disadvantages with it: the agent must be TCP-aware, as a result, this scheme is protocol dependent and cannot work for other existing protocols or future protocols when they become available. Although a lost packet can be retransmitted locally by the base station, the generated three duplicated acknowledgment packets still reach the sender of the TCP connection and cause the sender to unnecessarily reduce it sending rate by 50%.

2.3.6 Explicit Congestion Notification

Explicit congestion Notification is provided by Internet router for indication of incipient congestion where the notification can sometimes be through marking packets rather than dropping them. This would require an ECN field in the IP header with two bits. The ECN-capable Transport (ECT) bit would be set by the data sender to indicate that the end-points of the transport protocol are ECN-capable. The bit would be set by the router to indicate congestion in the end nodes. Routers that have a packet arriving at a full queue would drop the packet, just as they do now.

When gateway en route is congested or close due to congestion, it sets a bit in the packet header and Forward it on. A Random Early Detection (RED) gateway in marking mode is apt for this. A RED gateway detects incipient congestion by tracking the average queue size over a time window in the recent past. If the average exceeds a threshold, the gateway selects a packet at random and marks it, by setting the Explicit Congestion Notification (ECN) bit in the IP packet Header. This notification is echoed to the sender of the packet by the receiver. For TCP, This echo is piggybacked on an ACK. Now when the sender receives ACK with ECN, it reduces its congestion window, as it would if a packet loss had occurred. Thus this mechanism with explicit support from gateways allows TCP to perform proactive congestion control, over and above the reactive one triggered by packet loss [16].

The ECN algorithm can avoid depending on packet drops alone as implement congestion avoidance mechanisms.

3. PROPOSED DLN METHOD

3.1 Protocol Description

The algorithms proposed in this paper tried to improve the performance of TCP in wireless networks. But none of these algorithms actually lets TCP sender know clearly whether the packet is lost due to wireless error or network congestion. This makes the TCP sender retransmits the packet efficiently and then cannot keep the throughput high in the error prone environment.

The Snoop protocol is a good scheme to improve the performance of TCP in wireless network. But the Snoop protocol retransmits the lost packet like other link layer solutions. The Snoop protocol also suffers from not being able to completely shield the sender from the wireless losses. Based on Snoop protocol, we proposed a new protocol called Data Loss Notification (DLN) which can remedy limitations of the Snoop protocol.

In order to implement DLN protocol we modify the ACK format and the networking software at the base station and the mobile host.

3.1.1 DLN structure

We use a new form of ACK named DLN. In DLN we add the sequence number of the four most recently lost packets judged by the mobile host for each lost packet, and one bit (called

DLN bit) to indicate the reason of the lost packet. 1 indicates the packet is lost in the wired network congestion and 0 indicates the packet is lost due to wireless error. The default value of DLN bit transmitted by mobile host is 1 (assuming the corresponding packet was lost due to network congestion). The DLN bit is judged at the base station. DLN agent at the base station checked the information it stored in it to see if the packet has lost before it is arrived in the base station. If it found the packet had lost before it arrived in the base station, it remained the corresponding DLN bit to 1, or it would remain the DLN bit to 0. After the DLN is processed by DLN agent at the base station, it continues to be transmitted back to the fixed host. When the fixed host received the DLN, it would know the reason of packet loss from the DLN bit.

3.1.2 DLN agent at the base-station

The DLN agent at the base station has two main functions. One is to judge and store the packet lost information transmitted from fixed host. Like ordinary wired network, packet transmitted from fixed host to base station may be lost due to congestion. If the base station does not receive the packet in sequence, it will store the corresponding packet information in the DLN agent. Using the information stored in the agent, the base station can judge the reason of packet loss when it receive the DLN transmitted from mobile host. The second function is to judge the value of DLN. When the base station receives DLN, it will judge the lost packet based on the information it stored. If it finds the lost packet is lost before it arrived in base station, it will fill the DLN bit with 1 to indicate the packet was lost due to congestion. If the lost packet had arrived in the base station, it fills the DLN bit with 0 to indicate the packet was lost in the wireless channel.

3.1.3 Fixed host TCP sender

When the fixed host receives the DLN, it actions with the information contained in the DLN bit. If the DLN bit is 1, means the corresponding packet is lost due to wired congestion; TCP sender will proceed same as the window algorithm. If the DLN bit is 0, it means the corresponding packet is lost due to wireless error and it retransmits the packet immediately without window reduction.

3.2 Model

The DLN agent maintains a cache of TCP packets sent from the fixed host that have been forwarded to, but not yet been acknowledged by the mobile host. This is easy to do since TCP has a cumulative ACK policy. When a new packet arrives from the fixed host, the agent adds it to its cache and passes the packet on to the forwarding code, which performs the normal packet forwarding functions. The DLN agent also monitors TCP ACKs sent from the mobile host, using new ACKs to clean its cache and maintain only unacknowledged packet there.

The DLN agent has two main components: data processing and ACK processing. The flowcharts summarizing the salient features of the algorithms shown in Figure 1 and Figure 2. The remainder of this section describes the algorithm in detail.

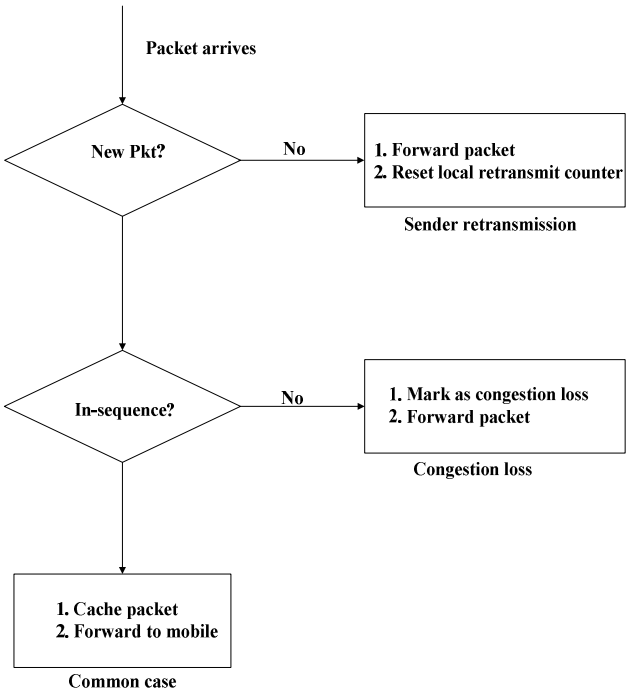

Figure 1. Flowchart for data processing in DLN

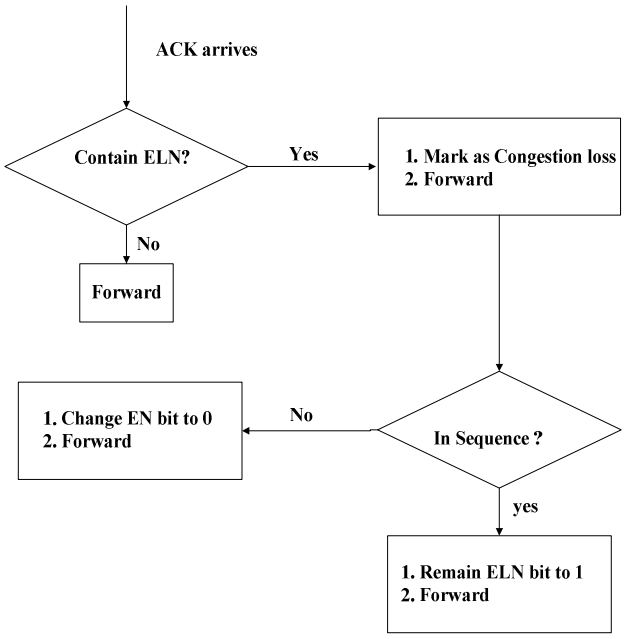

Figure 2. Flowchart for ACK processing in DLN

3.2.1 Data Processing

In this phase, the DLN agent processes incoming data segments from the fixed host. A TCP segment is identified by the sequence number of its first byte of data and its size. At the base station, the DLN agent keeps track of the last sequence number seen for the connection. At any stage in the protocol, one of the several kinds of packets can arrive at the base station from the fixed host, and the DLN agent processes them in different ways:

A new packet in the increasing TCP sequence: This is the common case, when a new packet in the normal increasing sequence arrives at the base station. In this case the packet is added to the DLN agent cache and forwarded on to the mobile host. The agent does not perform any extra copying of data while doing this.

An out-of-sequence packet that has been cached earlier: Although this is an uncommon case, it can happen. If there have been multiple losses in a single window due to congestion on the wired path, there are times when a timeout and slow start occur after a fast retransmission. This could lead to a sender retransmission of a previously cached segment. If the sequence number is greater than the last ACK seen, it is very likely that this packet didn't reach the mobile host earlier, and so it is forwarded on. If, on the other hand, the sequence number is less than the last ACK, the mobile host has probably already received this packet. At this point, there are several possible actions the agent could take. One possibility would be to discard this packet and continue, but this is not always the best thing to do. The reason for this is that the original ACK with the same sequence number could have been lost due to congestion while going back to the fixed host. The second possibility, to facilitate the sender getting to the current state of the last ACK seen at the base station (with the source address and port corresponding to the mobile host) to the fixed host. However, the disadvantage of this is that if the information in the DLN agent's state is wrong for any reason, the correctness of the end-to-end protocol is compromised. The third option is to simply forward the packet to the mobile host, and await information from subsequent ACKs to refresh the state at the agent. This is the option the agent uses, in keeping with our soft-state philosophies. The agent also resets the number of local retransmissions to zero, and updates the transmission time of this segment to correctly estimate the round-trip time when its ACK arrives.

An out-of-sequence packet that has not been cached earlier: In this case the packet was either lost earlier due to congestion on the wired network, or has been delivered out-of-order by the network. The former is more likely, especially if the sequence number of the packet (i.e., the sequence number of its first data type) is more than one or two packets away from the last one seen so far by the DLN agent. The agent and then forward to the mobile host cache this packet. It is also marked as having been retransmitted by the sender. The DLN processing algorithm uses this information to process DLN bit contained in ACKs that from the mobile host.

3.2.2 DLN Processing

The mobile host produces DLN if there is lost packet information contained in the receiving packet sequence. Each DLN packet may contain at most three lost packets that are most near by the acknowledged packet. Because the mobile host cannot judge if the packet is lost due to congestion or wireless channel error, it always fills the DLN bit corresponding to each lost packet for 1 (assume the packet is lost due to wired network congestion). In the base station when DLN agent receive DLN packet, it can use the information stored in the cache to judge the reason of the packet loss.

The DLN agent performs various operations depending on type of ACKs it receives. These ACKs fall into one of the following categories:

A new, expected ACK: This is the common case that occurs when no recent segments have been lost, and signifies an increase in the packet sequence received at the mobile host. This

ACK initiates the cleaning of the DLN agent cache and all acknowledged segments are freed. Finally, the ACK is forwarded to the fixed host.

An ACK contain DLN information: This is an ACK that contains one or several lost packet information. The lost packet is the packet that was not received by the mobile host. When the DLN agent received the ACK, it checks the information of data transmitted from fixed host. If the lost packet contained in the DLN is also cached in the agent that means this packet was lost before it was transmitted to the base station, the DLN agent marks the DLN bit of the corresponding packet to 1. If the lost packet contained in the DLN is not cached in the agent, that means this packet was lost after the base station, then it marked the DLN bit of the corresponding packet to 0.

A spurious ACK: This is an ACK less than the last ACK seen by the DLN agent, and is a situation that rarely occurs. We forward it to the fixed host, to guard against the possibility that the internal state of the DLN agent may be incorrect.

3.3 Simulation and Performance Results

We performed several simulations with the DLN protocol and compared the resulting performance with other algorithms using well known simulator ns-2. We present the results of the experiments in this section. We first introduce the simulation topology and parameters used in the simulation in Section 3.3.1. Later we discuss a trace comparison of Reno with DLN in high wireless error environment in Section 3.3.2 and the results of detailed simulation on different TCP algorithm in different parameters in Section 3.3.3. In section 3.3.4 we give simulation results of delay and window size in the Reno and DLN procession, and an analysis of the simulation results.

3.3.1 Simulation Topology

Figure 4 shows the network used for the simulations in this paper. The circle indicates a finite-buffer drop-tail gateway, and the squares indicate sending and receiving hosts. In the simulation, some parameters can be set to indicate different network condition; the parameters are summarized as following: 1. Buffer size (B packets) in the base station, 2. Propagation delay (D msec) which includes: 1) the time between the release of a packet from the source and its arrival into the link buffer; 2) the time between the transmission of the packet on the bottleneck link and its arrival as its destination; and 3) the time between the arrival of the packet at the destination and the arrival of the corresponding acknowledgment at the source, 3. The bandwidth (U packets/msec) of bottleneck link from base station to mobile host.

Figure 4. Simulation Topology

3.3.2 Trace Comparison

Figure 5 shows the sequence traces of TCP transfers over an error-free link, as well as TCP-Reno and the DLN protocol over error-prone links. The middle curve shows the progress of a transfer using the DLN protocol, which is a curve close to the ideal, error-free case. The performance improvement in this transfer over TCP-Reno is a factor of four. This is typical of the degree of improvement we obtain in our ns-2 simulation for reliable data transfers.

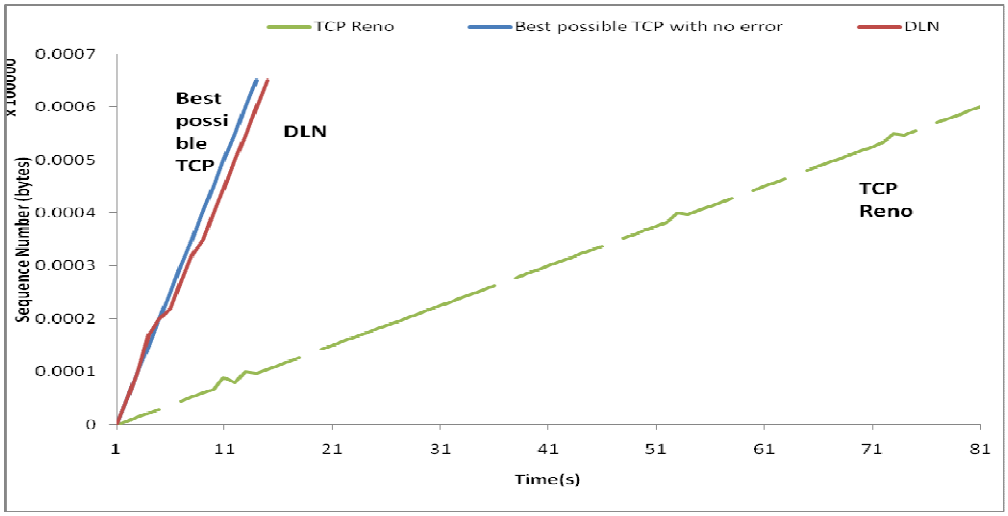

Figure 5. Sequence traces of TCP using an Ideal error-free links, as well as TCP Reno and the DLN protocol over error prone links

3.3.3 Throughput Comparison

Figure 6 to 9 show the throughput of algorithms under different wireless packet loss rate. We perform simulation with different network condition by changing the parameters such as buffer size, propagation delay and the bandwidth of bottleneck link. From simulation results we can see that throughput of TCP-Reno and TCP-Tahoe drop sharply when the error rate is increased to above 10-2. Throughput of TCP-Reno and TCP-Tahoe drop to only 10%~20% compared with no error rate in wireless link, DLN scheme can keep the throughput as high as 80%~90% of error free environment. There are significant performance benefits of using the DLN protocol. The main advantage of DLN is that it helps maintaining a large TCP congestion window when wireless error rate is high.

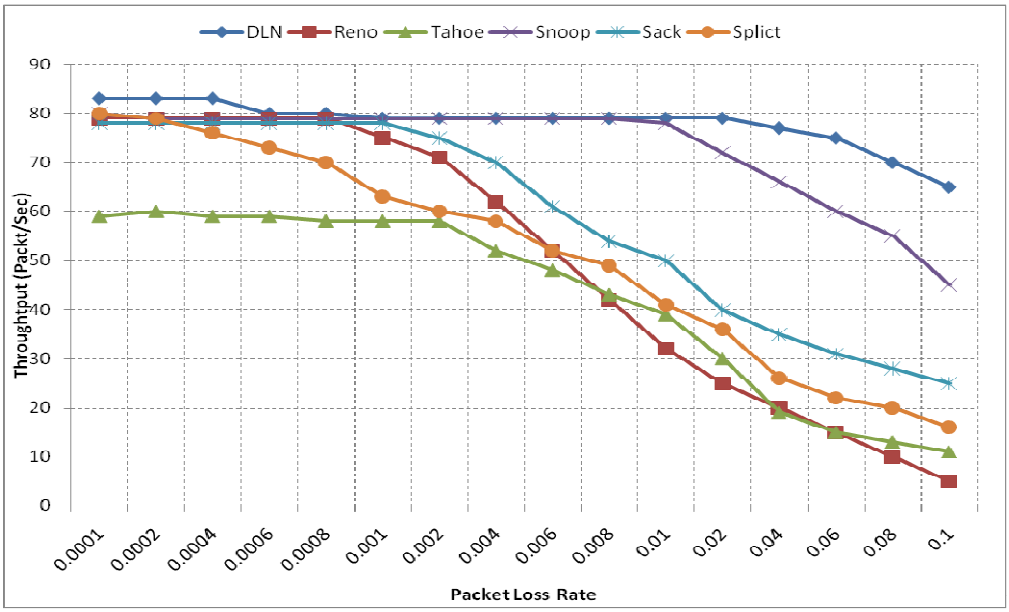

Figure 6. Simulation comparison (B=5; D=0.2; U=100)

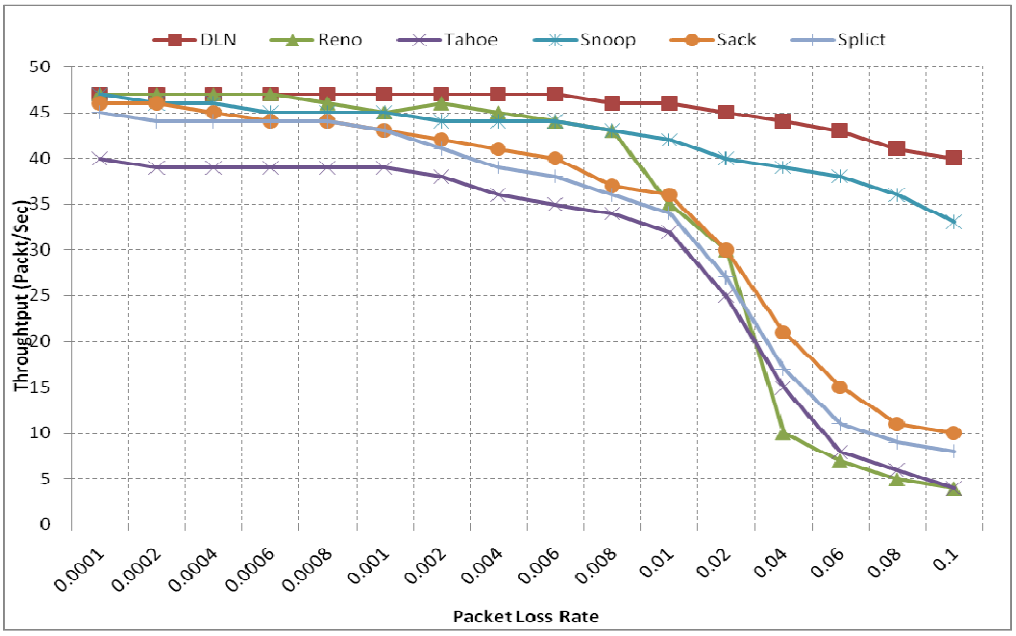

Figure 7 Simulation comparison (B=5; D=0.2; U=50)

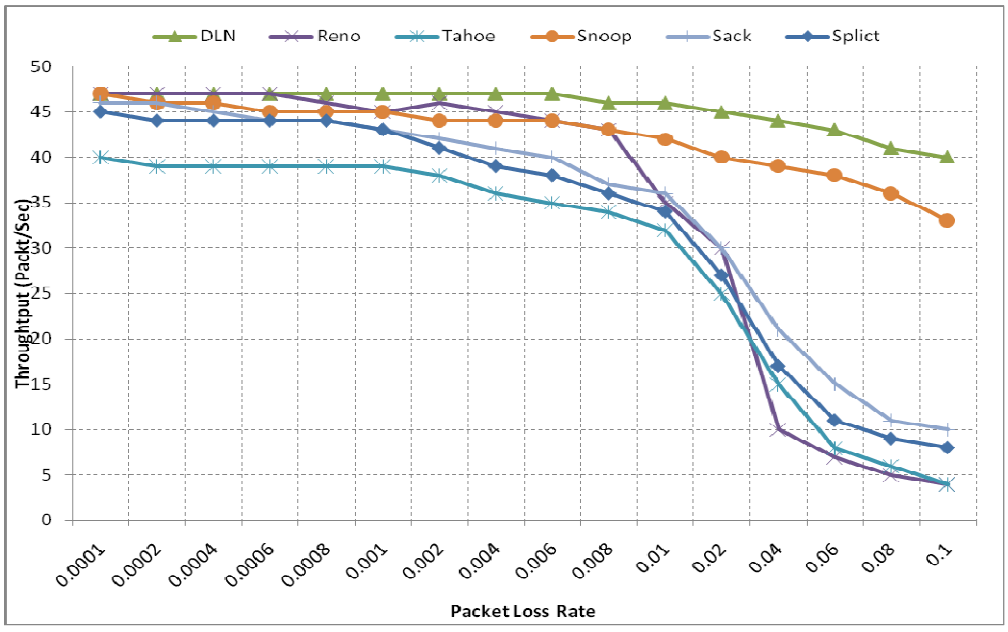

Figure 8 Simulation comparison (B=8; D=0.2; U=50)

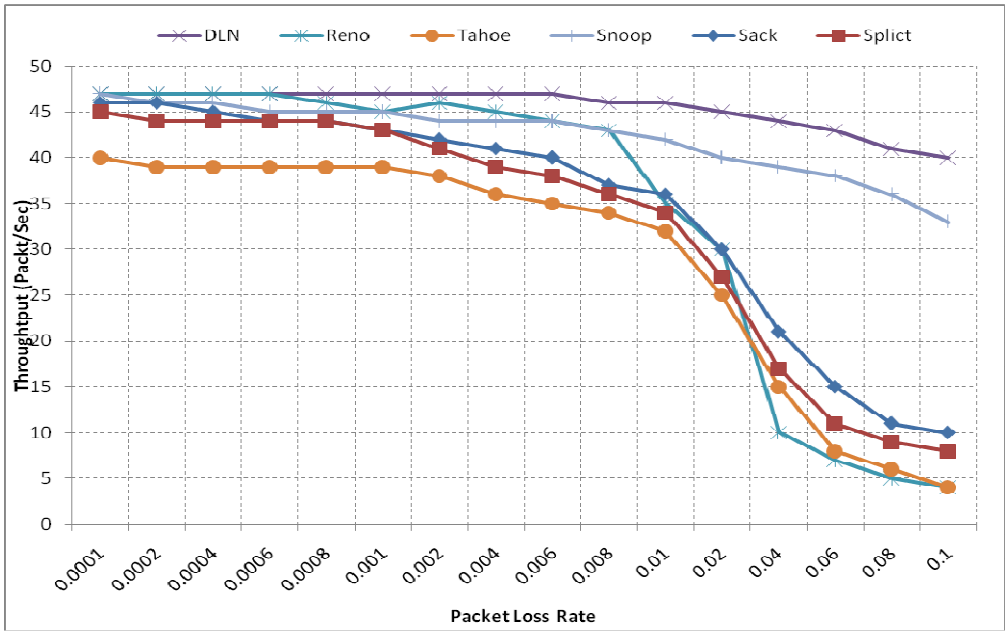

Figure 9 Simulation comparison (B=5; D=0.1; U=50)

3.3.4 Delay and Loss Recovery Analysis

In this section, we analyze the problems that arise due to small TCP transmission windows and long delay and discuss the function of DLN on solving this kind of problem [17].

TCP-Reno is the most popular TCP used in the Internet. It contains a number of algorithms aimed at controlling network congestion while maintaining good user packet is dropped for a window of data, but can suffer from performance problems when multiple packets are dropped due to the wireless error. In this section we illustrate the problem of TCP-Reno, show the end-to-end performance degradation of TCP-Reno in high packet loss wireless channel. We also compare the simulation results of the TCP-Reno algorithm with DLN to show the performance improvement of DLN.

If the packet is successfully transmitted from sender host to the receiver, the end-to-end delay is mainly determined by propagation delay, service time and queuing at the base station. But if the packet is lost due to network congestion or wireless packet loss, the TCP sender has to retransmit the lost packet by performing a loss recovery task and by waiting for time out. As a result, the end-to-end delay becomes significantly long when timeout happens.

Figure 10 to 12 show the delay of 200 packets transmission using TCP-Reno and DLN.

The TCP-Reno has quite good performance when there is no wireless error in the wireless channel. Figure 12 shows the end-to-end delay performance of TCP-Reno in error free environment. The mean end-to-end delay is about 0.15~0.2 second. There are two packets with delay around 0.5 second, this is due to network congestion, and these two packets are retransmitted by loss recovery mechanism.

In Figures 10 and 11 the packet loss rate in wireless link is 0.1, which means in 200 packets transmission there are about 20 packets lost in the wireless channel. Based on the simulation result, in TCP-Reno, the mean transmission delay is about 0.15~0.25 second and we can see that in 200 packets transmission, there are 24 packets whose transmission delay is significantly above 0.2 second. These packets are lost somewhere (because of network congestion or

wireless error) in the network. Of the 24 retransmitted packets, 11 packets have the delay around 0.4 to 0.6 second; that means these packets are retransmitted by loss recovery mechanism without invoking timeout, 13 other packets have a delay around 1 to 1.4 seconds, which means these packets are time out. The TCP sender always has to wait for time out to retransmit the lost packet, due to wireless packet loss. The DLN algorithm can efficiently avoid the timeout by retransmitting the lost packet immediately. Figure 12 we can see that the mean end-to-end delay for packet transmission is about 0.1~0.2 seconds. There are 23 packets with end-to-end delay for packet transmission is about 0.1~0.2 seconds. There are 23 packets with end-to-end delay between 0.4 to 0.6 seconds. Most of these packets are lost in wireless channel when it is first transmitted by the TCP-sender. Unlike TCP-Reno, the TCP sender knows these packets are lost due to wireless error but not network congestion. The lost packet can be retransmitted efficiently without incurring any window deduction, which avoided long idle time to wait for timeout.

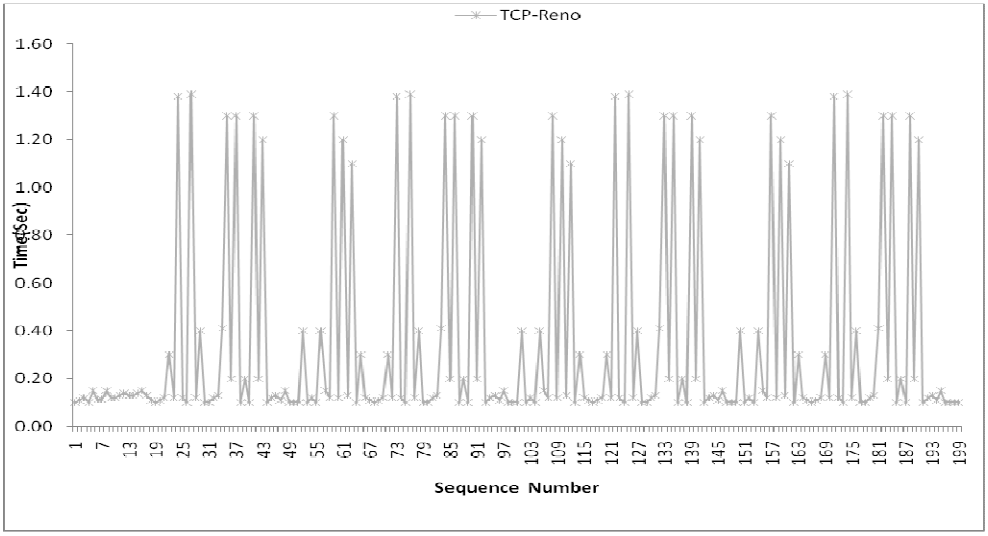

Fig 10. End-to-end delay for TCP-Reno (Wireless packet loss rate = 0.1)

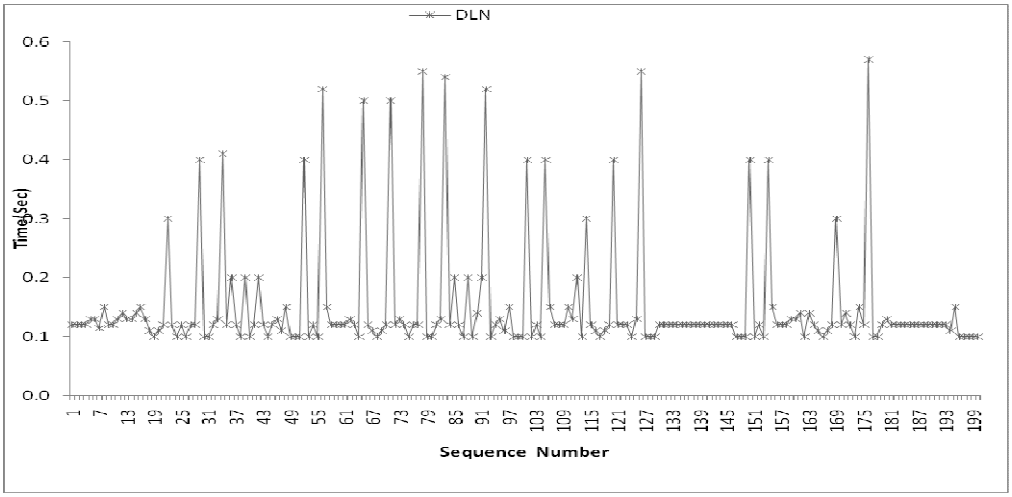

Fig 11. End-to-end delay for DLN (Wireless packet loss rate = 0.1)

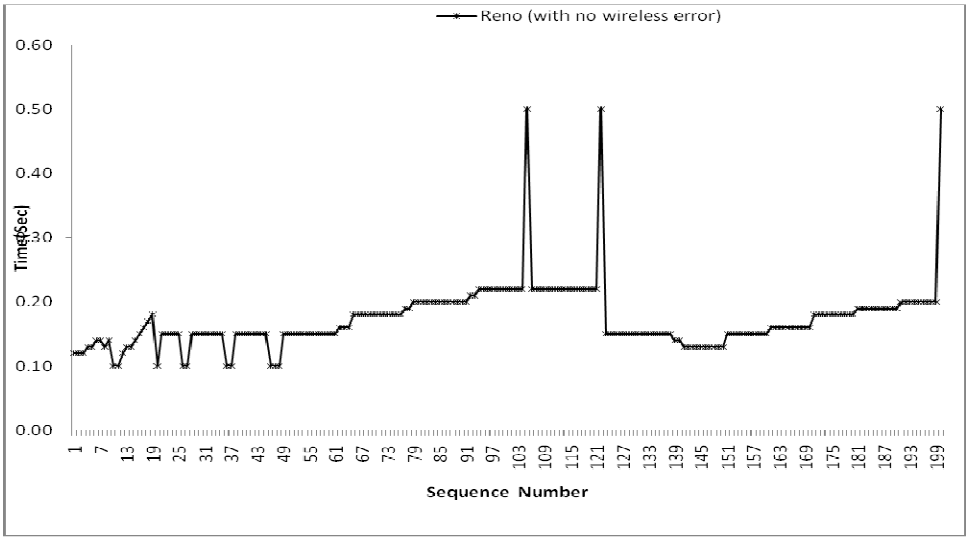

Fig 12. End-to-end delay for TCP-Reno (no wireless error)

The main reason for the occurrence of heavy timeouts in the TCP-Reno algorithm is the small congestion window, which makes TCP sender not to get enough duplicated acknowledgement in the procession and that the number of arriving duplicate ACKs is not sufficient to trigger a fast retransmission. The result is a timeout-driven transmission that keeps the link idle for long periods of time. Figure 13 to 15 show the congestion window size in the procession of transmitting 200 packets. From Figure 13 we can see the window cannot open big enough and always reduce to one due to timeout. So in the high packet loss environment, the TCP-Reno cannot efficiently transmit packet.

When there is no wireless packet loss, the only packet loss in transmission is due to network congestion. The TCP-Reno sender retransmits packet by using loss recovery algorithm. The window size is reduced to half when loss recovery happens, but no time out happens, because the congestion window (shown in Figure 15) is kept big enough and there are enough duplicated ACKs transmitted back to trigger the loss recovery.

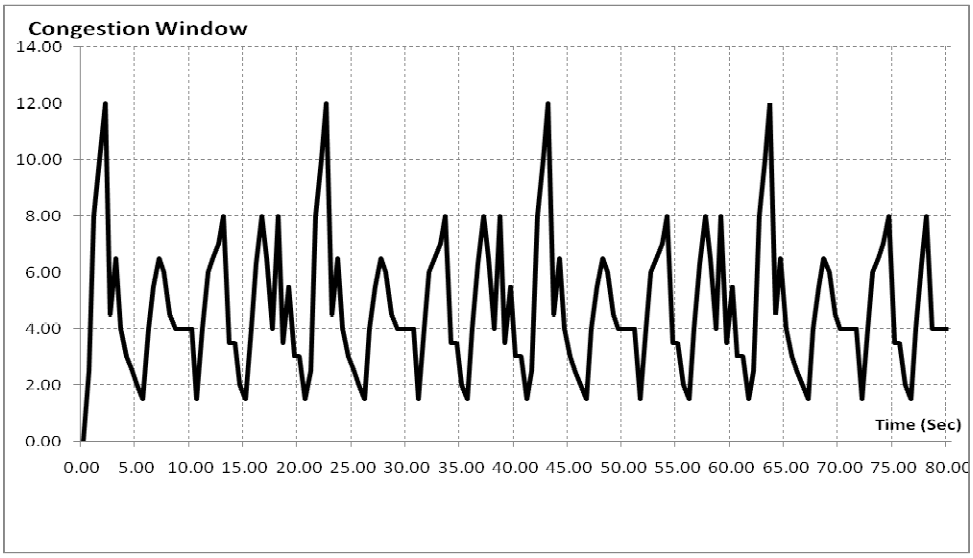

Fig 13. Window evolution for TCP-Reno (wireless packet loss rate=0.1)

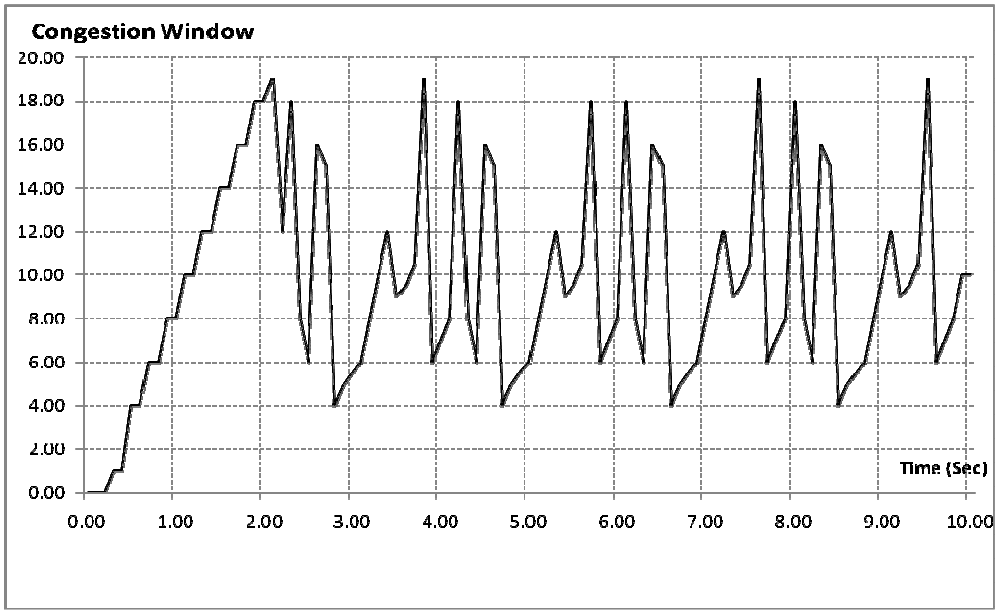

Fig 14. Window evolution for DLN (wireless packet loss rate=0.1)

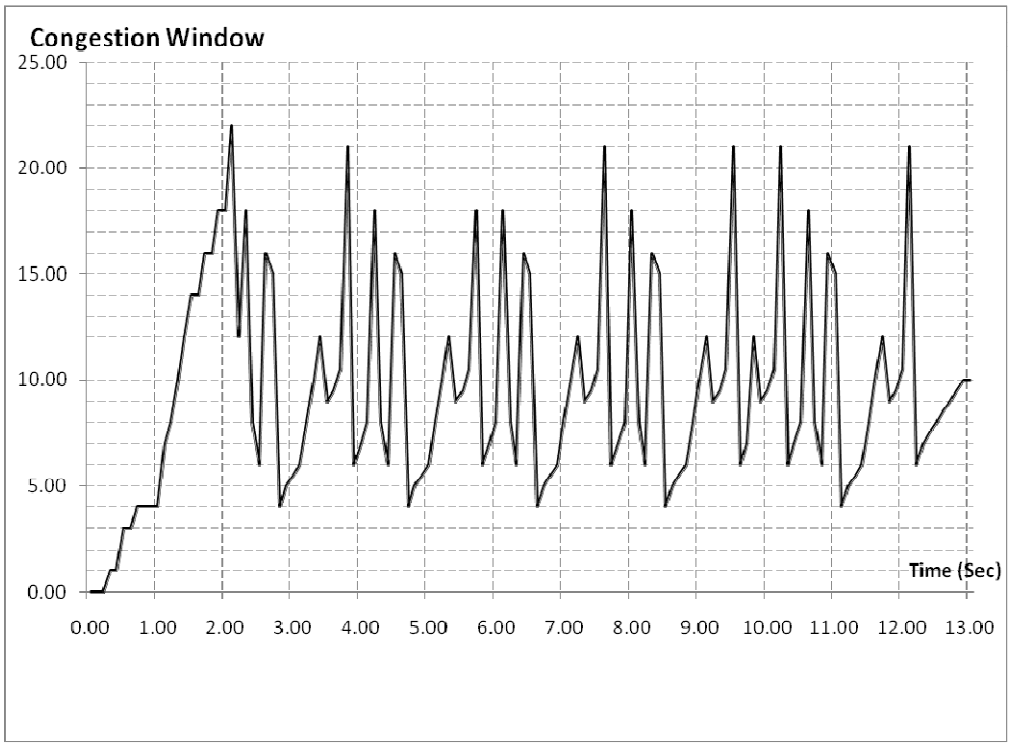

Fig 15. Window evolution for TCP-Reno (no wireless error)

4. CONCLUSION

In this paper, we identified fundamental challenges to improve the performance of TCP in wireless networks. TCP performance in many wireless networks suffers because bit-error-induced packet losses, which occur in burst because of the nature of the wireless channel, are misinterpreted by the TCP sender. TCP attributes these losses to network congestion because of the implicit assumptions made by its congestion control algorithms today. This causes TCP to reduce its transmission window in response and often cause long timeouts during loss recovery that keep the connection idle. Thus bit-error losses lead to degraded throughput and end-to-end delay performance.

While TCP adapts well to network congestion, it does not adequately handle the above problems in wireless media. This paper analysed the problems posed by the above challenges, and solved them by using modifications and enhancements to TCP at the sender and receiver.

The DLN protocol is a general framework by which receiver, base station, or other elements in the network can inform the TCP sender of losses that occur for reason other than congestion. When combined with algorithms to distinguish congestion losses from corruption, this framework provides a powerful way by which TCP senders can separate congestion control from loss recovery and recover from non-congestion-related losses without invoking congestion control. Simulation results and mathematical analysis indicate that the DLN protocol can keep the window open big enough without the affect of random wireless error.

It is easy to find the main advantage of DLN compared with other algorithm is its ability to judge the reason of packet loss by using the information contained in the acknowledgement transmitted back from the TCP receiver, thereby avoiding long idle waiting time, this algorithm can significantly improve the end-to-end delay performance in packet transmission.

ACKNOWLEDGEMENTS

I would like to begin by thanking my supervisor, Dr. E. Karthikeyan, Assistant Professor. He has been a wonderful advisor, providing me with support, encouragement, and an endless source of ideas. His breath of knowledge and his enthusiasm for research amazes and inspires me. I thank him for the countless hours he has spent with me, discussing everything from research to career choices, reading my papers, and critiquing my talks. His assistance during the travel of this journal has been absolutely invaluable-my life has been enriched professionally, intellectually, and personally by working with Dr. E. Karthikeyan, this work would have been impossible with his support and assistance. I also thank to anonymous reviewers for their many helpful comments and valuable suggestions.

REFERENCES

[1] T.Berners-Lee et al, (Aug 1994) "The World Wide Web," Communication of the ACM.

[2] J.B.Postel and J.Reynolds, (Oct 1985) "File Transfer Protocol(FTP)," Information Sciences Institute, Marina del Rey, CA,. RFC-821.

[3] J.B.Postel and J.Reynolds, (August 1982) "Simple Mail Transfer Protocol," Information Sciences Institute, Marina del Rey, CA,. RFC-821.

[4] R.W.Watson and S.A.Mamrak, (May 1987) "Gaining Efficiency in Transport Services by Appropriate Design and Implementation Choices," ACM Transactions on Computer Systems, 5(2):97-120.

[5] Tom Kelly, (2003) "Scalable TCP: improving performance in highspeed wide area networks," SIGCOMM

[6] MC.Chan and R.Ramjee, (April 2008) "Improving TCP/IP Performance over Third Generation Wireless Networks", IEEE Transactions on Mobile Computing, 7(4).

[7] OSI Transport Protocol Specification, 1986. Standard ISO-8703

[8] W.T.Strayer, B.J.Dempsey, and A.C.Weaver, (1992) "Xtp: The Xpress Transfer Protocol," Addison-Wesley, Reading, MA.

[9] J. B. Postel, (September 1981) "Transmission Control Protocol," Information Sciences Institute, Marina del Rey, CA.

[10] D-M. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks,"

[11] W.R.Stevens. (November 1994) "TCP/IP Illustrated", Volume 1. Addison-Wesley, Reading, MA.

[12] P.Karn and C.Partridge, (November 1991) "Improving Round-Trip Time Estimates in Reliable Transport Protocols," ACM Transactions on Computer Systems, 9(4):364-373.

[13] J.C Hoe, (August 1996) "Improving the Start-up Behavior of a Congestion Control Scheme for TCP", In Proc. ACM SIGCOMM'96.

[14] M.Mathis and J.Mahadavi, (August 1996) "Forward Acknowledgement Refining TCP Congestion Control," In Proc. ACM SIGCOMM.

[15] D.Lin and H.Kung, (March 1998) "TCP Fast Recovery Strategies: Analysis and Improvements," In Proc. INFOCOM.

[16] Ji-Hoon Yun, (2009) "Cross-Layer Explicit Link Status Notification to Improve TCP Performance in Wireless Networks," EURASIP Journal on Wireless Communications and Networking, vol. 2009, Article ID 617818, 15 pages, 2009. doi:10.1155/2009/617818

[17] W.Ding and A.Jamalipour, (2001) "Delay Performance of the New Explicit Loss Notification TCP Technique for Wireless Networks," GLOBECOM, San Antonio, Texas, Nov.25-29, 2001.

Authors

S.Saravanan received the master degree in Computer applications and also the M. Phil degree in Computer Science and Applications. He has over 10 years' industrial experience and 7 years in teaching experience. He was working as Senior Lecturer in one of the leading institution and also currently he is working as IT Manager. His research is specialized in Wireless networks & networking monitoring.

Dr.E.Karthikeyan received master degree from Bharathiar University and M.Phil in Bharathiar University. He also completed Ph.D. from Gandhigram Rural University, Dindigul, India. Presently he is working in Government Arts College, Udumalpet, India. He is active member in CSI, ISCA, IACST etc. His area of interest includes Network Security and Cryptography, Mobile Computing, Routing Algorithm and Video Streaming. He published a book entitled "Text Book for C: Fundamentals, Data structures and Programs" by PHI. He participated many Conferences, delivered lectures and chaired the sessions. He published more than 20 papers in various journals and conferences.