SELECTIVE-TCP FOR WIRED/WIRELESS NETWORKS

by

Rajashree Paul
Bachelor of Technology, University of Kalyani, 2002

PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF COMPUTING SCIENCE

In the
School
of
Computing Science

SIMON FRASER UNIVERSITY

Summer 2006

APPROVAL

Name:	**Rajashree Paul**
Degree:	**Master of Computing Science**
Title of Project:	**Selective-TCP for wired/wireless networks**

Examining Committee:

Chair: **Dr. Petra Berenbrink**
Assistant Professor of School of Computing Science

____

Dr. Funda Ergun
Senior Supervisor
Assistant Professor of School of Computing Science

____

Dr. Ljiljana Trajković
Co-Senior Supervisor
Professor of School of Engineering Science

____

Dr. Quanping Gu
Internal Examiner
Professor of School of Computing Science

Date Defended/Approved: June 05, 2006______________________________

ABSTRACT

One of the main reasons for TCP's degraded performance in wireless networks is TCP's interpretation that packet loss is caused by congestion. However, in wireless networks packet loss occurs mostly due to high bit error rate, packet corruption, or link failure. TCP performance in wired/wireless networks may be substantially improved if the cause of packet loss could be detected and appropriate rectifying measures taken dynamically. This report proposes a new end-to-end TCP protocol named Selective-TCP that distinguishes between congestion and wireless link errors (high bit error rate, packet corruption) and invokes appropriate correction mechanisms. This makes the proposed protocol better suited in a wide range of applications in mixed wired and wireless links. Selective-TCP gives up to 45% increase in goodput over NewReno in the simulation scenarios that we analyzed.

Keywords: Selective-TCP; wired/wireless networks; TCP NewReno; loss detection.

DEDICATION

To my family in India: my parents Ms. Anjali Paul and Mr. Subhrangshu Paul, my aunt Ms. Anima Pal, my uncle Mr. Provanshu Sekhar Paul, and my elder brother Mr. Rajsekhar Paul. All of them have been extremely supportive throughout my graduate studies. Without them I could not have obtained my degree.

ACKNOWLEDGEMENTS

I thank my supervisors, Professor Dr. Ljiljana Trajković and Dr. Funda Ergun, for their guidance, encouragement, and patience. I am very thankful to Dr. Ljiljana Trajković that she gave me the chance to be a member of Communication Networks Laboratory and to continue my graduate studies.

I thank all the members of Communication Networks Laboratory for their support, especially Renju Narayanan, Nikola Cacov, Modupe Omueti, and Wan Zeng.

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY

ACK	Acknowledgement
ARQ	Automatic repeat request
AWND	Advertised window
BS	Base station
CCSDS	Consultative Committee for Space Data Systems
CWND	Congestion window
DARPA	Defence Advanced Research Project Agency
ELN	Explicit loss notification
FIN	Finish
MSS	Maximum segment size
NAM	Network animator
NAK	Negative acknowledgement
NSF	National Science Foundation
OSI	Open system interconnection
RTO	Retransmission timeout
RTT	Round trip time
SACK	Selective acknowledgement
SCPS-TP	Space communication protocol standards-transport protocol
SNACK	Selective negative acknowledgement
SRTT	Smoothed round trip time

SSTHRESH	Slow start threshold
SYN	Synchronous
TCP	Transport control protocol
UDP	User datagram protocol
WLAN	Wireless local area network

1 INTRODUCTION

1.1 Motivation

The transmission control protocol (TCP) is the most extensively used transport protocol by Internet applications due to its robustness and reliable connectivity. Wireless communication technology is making immense progress and has become widely popular for access networks over past few years. These wireless access networks, such as Wireless Local Area Networks (WLAN) and cellular networks, are usually connected to a wired backbone network. Although TCP is very reliable in wired networks, its performance deteriorates in wireless environment.

Wireless networks have characteristics very different from wired networks. The host are mobile and for most wireless access networks the mobile hosts communicate with the fixed host through a base station. It is desired that the quality of service is consistent for wired and wireless networks. Typical problems for wireless communication are:

High bit error rate: Wireless links experience random packet losses due to bursty nature of the wireless traffic. Random packet loss rate ranges from 1% to 10%, which is significantly higher than wired networks.

Disconnections: There are many reasons for disconnection: such as when a mobile host moves to the region of a new base station (hand off), a mobile host moves out of the transmission area of the base station, or due physical obstructions.

Limited and variable bandwidth: In general, wireless links have lower bandwidth (~2 Mbps) than wired links and in most cases the bandwidth is shared between several mobile hosts.

Limited battery power: Mobile hosts run on battery, thus small transmission time is desirable.

Dynamic network topology: The network topology changes often due to movement of hosts.

All these reasons contribute to the random nature of packet loss in wireless networks. TCP was designed for conventional wired networks and since most of the Internet traffic is carried by reliable wired links, TCP assumes that all packet loss is due to congestion. Every packet loss is followed by reduction in transmission rate so that the congested router buffers gets time to clear the queues. As a result, in wireless networks too, every random packet loss event results in reduced transmission rate/congestion window leading to very poor utilization of available bandwidth.

The main reason for TCP's performance degradation in wireless networks is TCP's inability to distinguish congestion losses from other types of losses. In wireless links, the reasons for packet loss are high bit error rates (BER) due to bursty nature of wireless traffic, packet corruption, or link outage. However, TCP treats all these errors as congestion and initiates the congestion control mechanism. This results in low utilization of available bandwidth, unnecessary retransmissions, and, ultimately, low goodput and throughput.

To improve TCP performance in mixed wired/wireless networks, we propose an end-to-end approach, based on loss detection.

1.2 Contribution

In this report, we propose Selective-TCP, an end-to-end design that improves TCP performance in wired/wireless networks. It distinguishes packet loss due to congestion from packet loss due to transmission in wireless links. Selective-TCP is an extension to TCP NewReno [1]. Selective-TCP algorithm could also be applied to TCP Reno [2]. We used the ns-2 network simulator to implement this algorithm and to evaluate its performance in the presence of burst errors.

We have also shown a comparative study of performance evaluation for Selective-TCP, TCP NewReno, and TCP packet control algorithm [8]. TCP packet control algorithm is a link layer based approach to improve TCP's performance in mixed wired/wireless networks. It addresses two problems specific to wireless networks: one is delay variations (causing spurious fast retransmit) and the other is sudden large delays (causing spurious fast timeout). TCP packet control algorithm was developed by Wan Zeng [8]. We have also compared Selective-TCP with TCP Reno, TCP SACK, and TCP Westwood.

1.3 Outline of Report

We provide the background material on TCP and Selective Negative Acknowledgement (SNACK) in Chapter 2. A review of previous work on improving TCP performance in wireless networks through loss detection is presented in Chapter 3. In Chapter 4, we describe the proposed algorithm, named Selective-TCP. We describe the design and implementation details of the Selective-TCP algorithm in Chapter 5. In Chapter 6, we evaluate the performance of Selective-TCP and compare it with TCP NewReno, TCP packet control algorithm, TCP Reno, TCP SACK, and TCP Westwood, by using ns-2 network simulator. Conclusions and future work are given in Chapter 7.

2 BACKGROUND

In this Chapter, we present the background materials for the proposed algorithm. Selective-TCP is an extension to existing TCP NewReno [1]. First, we describe the TCP algorithms and then discuss the Selective Negative Acknowledgement (SNACK), used in Selective-TCP.

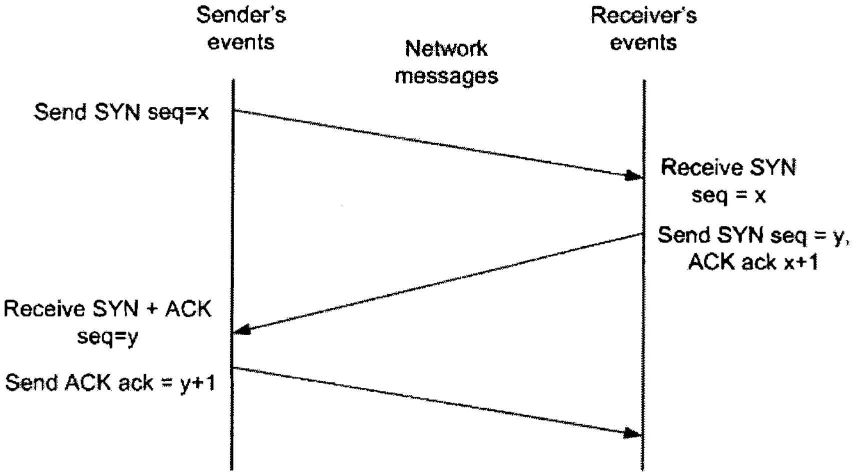
2.1 The Transmission Control Protocol

Transmission Control Protocol (TCP) is a connection oriented, point-to-point protocol. It is extensively used in the Internet.

The main characteristics of TCP are:

Reliable data transfer: In TCP, the two processes (client process and server process) that want to communicate, first handshake with each other. A three-way handshaking is used. The client first sends a special TCP segment (only the TCP header and IP header) to the server, server acknowledges and sends another special segment. Finally, the client acknowledges the special segment from the server. Fig. 1 shows the three-way handshake. The client process passes data through sockets. TCP directs these data to the send buffer. TCP takes a block of data from the send buffer. The maximum amount of the block of data is limited by the Maximum Segment Size (MSS). TCP encapsulates each block of client data with TCP header and forms a TCP segment. When TCP receives a segment, the segment's data is placed in the receive buffer of the connection. The application reads data from this buffer. A TCP connection consists of buffers, variables and a socket connection to a process in one host and another set of buffers, variables and socket connection to a process in another host. No buffers or variables are allocated to the connection in the network elements (such as routers) between two hosts. If a certain time, called timeout, has passed without acknowledgement, a new connection request is sent [3].

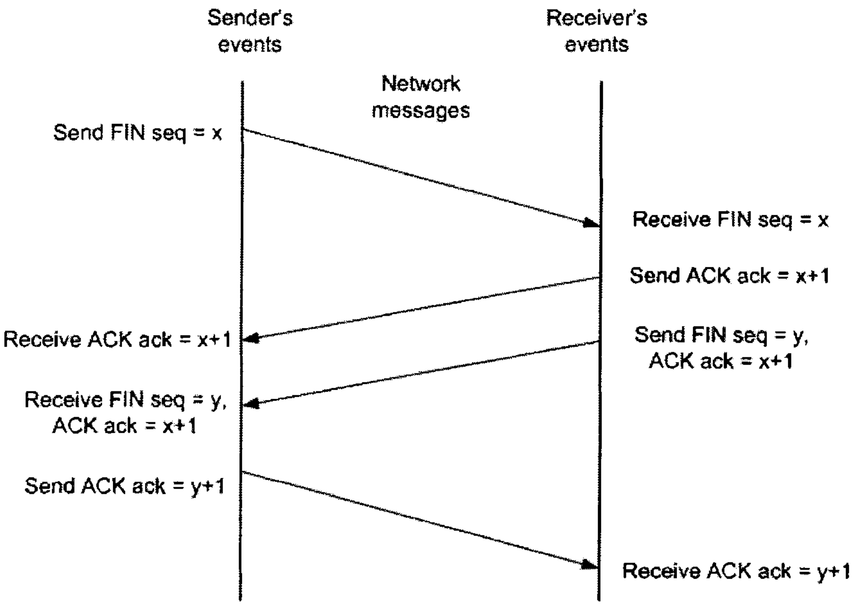
Figure 1: Three-way handshake between the TCP sender and receiver.



TCP views data as ordered stream of bytes. The sequence number of a segment is the byte-stream number of the first byte of the segment. The acknowledgement number that one host puts in its segment is the sequence number of the next byte that host is expecting from another host. TCP only acknowledges bytes up to the first missing byte in the data stream, this is known as cumulative acknowledgement. If a host receives out of order segments, TCP either discards out of order bytes or keeps them and waits for the missing bytes to fill in the gaps.

While TCP uses three segments to initiate the connection, it needs four segments to terminate the connection. When the client wants to end connection, it sends a FIN segment to the server. The server has to acknowledge the FIN segment, but can still send data to the client. This is known as half-close state of TCP connection. When the server decides to close the connection, it sends a FIN segment. Finally, when the client acknowledges the FIN segment the connection is completely closed.

Figure 2: TCP connection termination.



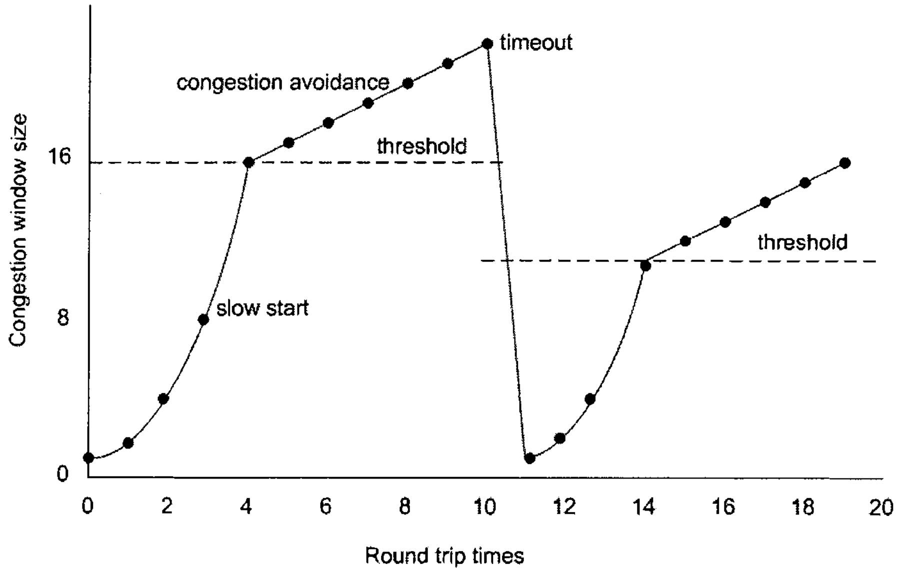
Flow control: TCP uses the sliding window mechanism, which sends as many segments as the receiving end can handle, before having to wait for acknowledgments. In this mechanism, a window of segments (certain number of segments) is transmitted at once. Each segment has a sequence number. The receiver can acknowledge more than one segment at a time by acknowledging the highest one received, meaning that all the previous segments were successfully transmitted. A field called the advertised window (AWND) in the TCP header is used to inform the sender of the receiving buffer size. The sending window is limited by the AWND, so that a fast sender does not overwhelm a slow receiver.

Connection control: If the sender is only limited by the AWND, many packets can be dropped because of a full buffer in an intermediate router. Therefore, the sending window should not be limited only by the receiver buffer, but also by the network capacity. The window size resulting from congestion control is called the congestion window (CWND). The sending window is taken as being the minimum of the AWND and the CWND. If a packet is lost, TCP retransmits it (and all the following packets) through its Automatic Repeat Request (ARQ) mechanism. This kind of ARQ is called Go-Back-N [3]. TCP's connection management has four phases: slow start, congestion avoidance, fast retransmit and fast recovery [2].

Slow start: The way in which TCP data transmission operates during the start of a connection is known as slow start. The slow start algorithm avoids the congestion problem by observing that the rate at which new packets should be transmitted in the network is the rate at which the acknowledgments are returned by the other end. The sender starts by transmitting one segment and waiting for its ACK. Afterwards, CWND is doubled each time an ACK is received. The main drawback to slow start is the large amount of time that is required during start up. If the data that is being sent is very small, the bandwidth efficiency will be reduced considerably.

Congestion avoidance: The slow start increases the CWND exponentially. At some point during the connection, a bottleneck in the network will be congested and will start discarding packets [3]. Therefore, above a certain threshold, an exponential increase of CWND seems inappropriate to find the right CWND value. The relation between slow start and congestion avoidance is done through a variable called slow start threshold (SSTHRESH). If CWND is smaller than SSTHRESH, the TCP sender is in slow start, otherwise it is in congestion avoidance, meaning that CWND is increased only by 1/CWND each time an ACK is received. This is an additive increase, shown in Fig. 3.

Figure 3: TCP connection: slow start, congestion avoidance, and timeout.



TCP assumes that almost all packet losses are due to congestion somewhere in the network. Therefore, it is necessary to reduce the amount of segments to be sent if a packet loss is detected. Retransmission timeout (RTO) or the reception of duplicate ACKs indicates packet loss. When congestion occurs, SSTHRESH is set to [3]:

$$\text{SSTHRESH} = \text{Max}(\text{Min}(\text{CWND}, \text{AWND}), 2).$$

The TCP behaviour is different if the congestion is detected through an RTO or three duplicate ACKs. When three duplicate ACKs are received by the TCP sender, the following TCP mechanisms can take place:

Fast Retransmit: When a RTO occurs, it implies that almost no packets could go through the network because of congestion. However, if the TCP sender receives duplicate ACKs, it means that a packet was lost, but other packets reached the receiver. In this case, TCP will retransmit the lost packets without waiting for the RTO [3].

Fast Recovery: TCP performs fast recovery immediately after fast retransmit. When the third ACK duplicate is received, TCP performs congestion avoidance instead of slow start, since it does not want to reduce the flow abruptly by going into slow start. SSTHRESH is set to one-half the current window, but CWND will be set at SSTHRESH plus three (because of the three duplicate ACKs received) [3]. Each time another duplicate ACK arrives, CWND is incremented by one. When a new ACK (acknowledging new data) arrives, TCP enters congestion avoidance phase.

TCP retransmission timeout: When data segments are not received and the RTO expires, TCP retransmits the segments, goes back to slow start, and recalculates RTO. Since the time between when a packet is sent and its ACK arrives (known as the Round Trip Time RTT) may vary depending on the network, RTO cannot have a fixed value. The RTO is calculated as [3]:

$$\text{RTO} = \text{SRTT} + 4 * \text{Deviation}$$

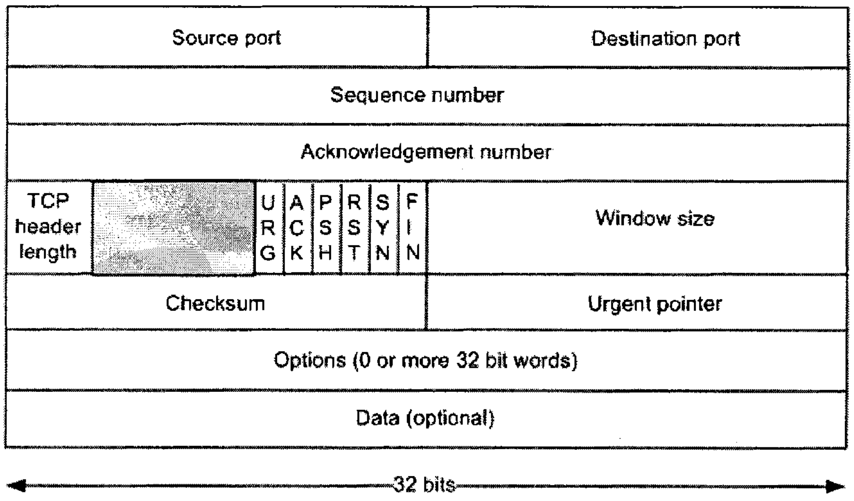
$$\text{SRTT} = 7/8 * \text{SRTT} + 1/8 * \text{SampleRTT}$$

$$\text{Deviation} = 3/4 * \text{Deviation} + 1/4 * |\text{SampleRTT} - \text{SRTT}|,$$

where SampleRTT is the last calculated RTT value, SRTT (smoothed RTT) is the moving average of RTT, and Deviation is the mean deviation of RTT. It can be seen that the RTO is dependent on the last RTT sample and on the past RTTs. When a timeout occurs, the RTO is doubled, with a maximum of 64 seconds [3].

TCP header: The TCP sender and receiver must share some information (such as, acknowledgments and receiver's buffer size). This information is sent within a header appended to each TCP segment. The standard size of this header is 20 bytes, but some protocols use 20 more bytes for TCP options as shown in Fig. 4. In order to be reliable, the two hosts using TCP must be aware of the connection and should be synchronized with each other.

Figure 4: **TCP header.**



2.2 Selective Negative Acknowledgement

The Selective Negative Acknowledgement or SNACK [4] option of the space communication protocol standards – transport protocol (SCPS-TP) [4] improves its performance in high bit error environment and increases link utilization and throughput.

SNACK is usually seen as a combination of SACK [5] and NAK (Negative Acknowledgement) [6].

The Selective Negative Acknowledgement (SNACK), as its name implies, is the process of selectively sending negative acknowledgements. The receiver of data informs the sender of the segments that it did not receive. This option may include information about multiple segments, which is suitable in presence of packet reordering. The traditional acknowledgements used could include information about only one missing segment in a window. If the TCP suffers from multiple missing segments in a window, then the receiver would have to send multiple acknowledgements to inform the sender about of all the missing segments at the receiver. This is particularly disadvantageous, especially in asymmetric channels where the upward channel has much lower bandwidth than the downward channel.

SNACK Option Operation: The SNACK option is enabled when both clients' TCPs include the SNACK capability in the TCP SYN (synchronous) segment header. If either side does not support SNACK, then the regular ACKs or delayed ACKs are used.

The receiving TCP invokes the SNACK option by sending an appropriately formed SNACK segment whenever an out-of-sequence queue forms at the receiver. The SNACK information is stored in the options area of the TCP header. SNACK usage occurs when there is disordering of packets in the network, hence it is beneficial to delay the sending of SNACK so as to give enough time for the out-of-order packets to arrive at the receiver. The receiver must not invoke a SNACK option unless it is sure of the missing packets in the network. An unnecessary SNACK can invoke unnecessary retransmission, which can degrade the system performance rather than improve it [4].

SNACK is usually used in environments prone to significant loss. Hence, it may be the case that the SNACK segment sent got lost. In order to overcome this problem, the SCPS-TP allows sending of subsequent SNACKs with the information of previous missing slots. A single SNACK can carry information about multiple missing slots in the receiver queue, so the receiver can easily include the information of the previous missing slots without much effort. A SNACK, like a SACK or NAK, does not alter the meaning of an acknowledgement; it only provides additional information to the sender about the receiver's queue. The sender, on receiving a SNACK, aggressively retransmits all the segments that are indicate as missing/empty slots (also known as holes) in the receiver queue. These aggressive retransmissions prevent retransmission timeouts and avoid link idle time, resulting in higher bandwidth utilization [4].

The SNACK Option Fields: The SNACK option is located in the TCP header options field [4]. It constitutes the following fields:

Option type: The option type field is mandatory for the usage of SNACK. It consists of one octet that is the first octet in the options field. It should contain the decimal value of 21.

Option length: The option length is also a mandatory field and occupies the second octet of the options field. It is one octet in length. The value contained in this field is the total number of octets used by the options. Hence, the length may vary depending on the optional bit-vector. If the SNACK bit-vector is not included, then the standard length is six octets. If the bit-vector is included, then the length shall be the sum of the length of the bit-vector and the standard six octets.

Hole1 offset: Hole1 offset is a mandatory field that occupies two octets (the third and the fourth octet) in the option field. This value indicates the offset at which the first hole or empty slot in the receiver queue occurs from the current acknowledgement. It is specified in terms of Maximum segment size (MSS) units and can be obtained by subtracting the ACK number from the offset sequence number and dividing the difference using integer arithmetic by the amount of user data carried in one MSS:

Hole1 offset = (offset sequence number – ACK number) /1 MSS in bytes.

If the above division results in a remainder, then it is added to the size of the hole.

Hole1 size: The Hole1 size is also a mandatory field that occupies two octets in the options area, the fifth and sixth octet. This field contains the size of the first hole that is being reported to the sender. This field gives the size of this hole in MSS units. It is obtained by dividing the size of the hole in octets by the amount of data carried in one MSS as:

Hole1 Size = Size of Hole1 (in octets) / 1 MSS in bytes.

If this division produces a non-zero remainder, then the Hole1 size is rounded up to the nearest integer.

SNACK bit-vector: The SNACK bit-vector is the only optional field of the SNACK option. It follows the Hole1 size and occupies consecutive octets. The number of octets that it occupies is implementation dependent. It contains information about the remaining holes that were detected in the receiver queue following the first hole, which is already indicated by Hole1 offset. The SNACK bit-vector maps the receiver sequence space in the form of MSS units starting one octet after the Hole1 offset. The bit-vector consists of zeros and ones. A zero indicates that an MSS sized block at that particular location is missing, and a one indicates that the particular MSS was received successfully. Since the TCP header option has to end in the proper octet boundary, zeros are added after the last occurrence of one in the bit-vector. These zeros should not be interpreted as missing data.

The SNACK Option Operation: The SNACK option is enabled when both the TCP sender and receiver include the SNACK capability in the options area of the TCP SYN segment. The receiving TCP can invoke the SNACK option by sending an appropriately formed SNACK to the TCP sender. Inclusion of the SNACK option does not alter the meaning of an ACK but only adds more information to the ACK. SNACK is invoked if an out-of-order queue forms at the receiver buffer.

Delaying SNACK: Whether the SNACK should be delayed or not depends on the kind of underlying network, as a result it is implementation dependent. The main factor that needs to be considered is the probability of packet disordering in the network. The SNACK sender has to be certain that the retransmissions that it invokes are necessary because unnecessary retransmissions may deteriorate the system rather than enhance its efficiency. It is assumed that a small number of retransmissions do not cause as much damage as an link that is idle for a long time. The SNACK is usually delayed so that the receiver can wait for the out-of-order packets to reach the receiver. Consequently, unless disordering of segments is highly unlikely in the network, it is always beneficial to delay a SNACK. There is no standard method of determining for how long the SNACK should be delayed. It is implementation-dependent.

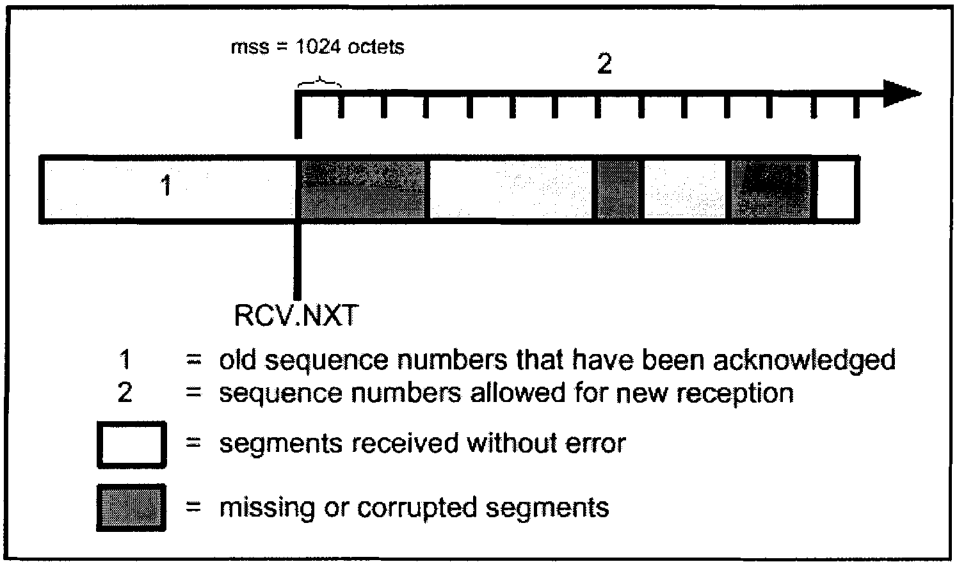
Retransmission of SNACK: If the communication environment has a high probability of losing segments, then it is desirable to send a SNACK option for a hole that has been already reported in the previous SNACK option. Thus, even if the previous SNACK gets lost, the SNACK sender can inform the SNACK receiver about all the missing segments. Again, how long retransmission should be delayed is implementation dependent. It is advisable to send a SNACK when a FIN (finish) segment is received.

SNACK Receiver: The SNACK receiver is the data sender. Upon receiving a well formed SNACK, the sender has to immediately retransmit all the segments mentioned as missing in the SNACK option. The goal is to prevent retransmission timeouts. The first retransmitted segment is the one mentioned in Hole1 of SNACK option, followed by the bit-vectors segments in the ascending order of the sequence number. The SNACK bit-vector field is left-shifted until the last "1" is found. All the "0" bits indicating the missing MSS occurring to the left of the last "1" are retransmitted.

An Example of SNACK: An example scenario of the receiver's out-of-sequence queue is shown in Fig. 5. We consider the two options of sending SNACK: with and without the SNACK bit-vector.

According to Fig. 5, there are three holes in the receive buffer queue. The first one starts immediately after the set of acknowledged segments at offset zero and is three MSS units in length. The second hole occurs at the offset eight from the already acknowledged data and comprises of just one MSS. The third hole is found at the offset eleven and is two MSS units long.

Figure 5: **Example of receivers queue.**


mss = 1024 octets
2
1
RCV.NXT
1 = old sequence numbers that have been acknowledged
2 = sequence numbers allowed for new reception
□ = segments received without error
▨ = missing or corrupted segments

Source: CCSDS Secretariat, 1999 [4], reprinted by permission.

First, we consider sending a SNACK for the above scenario where we use a SNACK bit-vector. The SNACK generated by the receiver is shown in Fig. 6.

Figure 6: **SNACK options with bit vector for example receivers queue.**

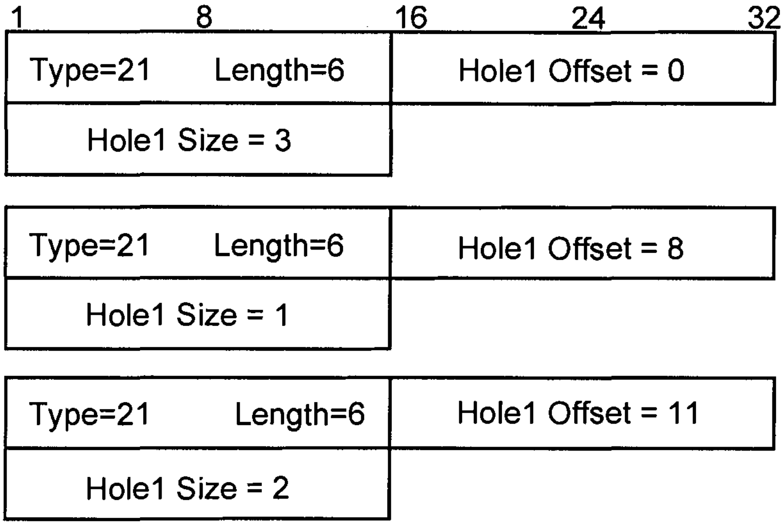
1	8	16	24	32
Type=21	Length=8	Hole1 Offset = 0		
Hole1 Size = 3		11110110	01000000	

Source: CCSDS Secretariat, 1999 [4], reprinted by permission.

The TCP option “Type” is a constant “21” for SNACK. The length field is the number of octets occupied by the SNACK option and the bit-vector. For this scenario, we have the standard six octets of the SNACK option and the two octets of bit-vector which makes the “Length” eight. The first hole is formed immediately after RCV.NXT so the “Holel Offset” is set to zero. The “Holel Size” is the size of first hole, which is three because three segments are missing in the first hole. The bit-vector shows the status of the receiver buffer queue after the first hole: The receiver has received four segments after the first hole followed by a single hole, then two segments followed by two holes and then a single segment as shown in Fig. 6. If we denote the received segments by a one and the missing segments by a zero we have four ones followed by a zero, followed by two ones, then followed by two zeros, and finally a one at the end. The remaining zeros after the last one are not counted and are used for padding. Hence, the bit-vector pattern formed is “11110110 01000000”.

The other option would be sending a SNACK without using the bit-vector, which is also allowed by the SCPS-TP specification. The three SNACK options shown in Fig. 7 may occur in the same acknowledgement or in separate acknowledgements. The SNACK options are applied for each encountered hole. These three segments fully report the state of the out-of-sequence receiver queue.

Figure 7: SNACK option without bit vector for example receivers queue.



1	8	16	24	32

Type=21 Length=6	Hole1 Offset = 0
Hole1 Size = 3	

Type=21 Length=6	Hole1 Offset = 8
Hole1 Size = 1	

Type=21 Length=6	Hole1 Offset = 11
Hole1 Size = 2	

Source: CCSDS Secretariat, 1999 [4], reprinted by permission.

3 RELATED WORK

In this Chapter, we present the previous work on different approaches of improving TCP performance in wired/wireless network by detecting types of packet losses.

3.1 Survey of Existing Techniques for Improving TCP Performance through Loss Detection

Several techniques have been proposed to mitigate the effects of non-congestion related losses on TCP's performance. They may be classified as end-to-end (TCP-Reno [2], NewReno [1], and SACK [5]), link layer (Snoop-TCP [7] and TCP packet control algorithm [8]), and split-connection (M-TCP [9] and I-TCP [10]) approaches. Comparative analysis of these approaches [11] indicates that link layer techniques are most effective in improving TCP performance in wireless networks, while split-connection based methods sometimes lead to poor end-to-end throughput due to shielding the wireless from the wired section of the network. End-to-end schemes, although less effective than link layer based techniques, are the most promising because they achieve significant performance gain without requiring expensive changes in the intermediate nodes.

3.1.1 End-to-end

End-to-end protocols are the most promising since they can achieve significant performance gain without any extensive support at the network layer, that is, in the intermediate routers and base stations; however, they are not as efficient as link layer based techniques (local recovery) in handling wireless losses.

End-to-end schemes try to improve TCP performance in wireless networks through the use of two techniques [8]. First, they use some form of Selective Acknowledgements (SACK) to allow the sender recover from multiple packet losses in a window without resorting to a coarse timeout. Second, they attempt the sender to distinguish between congestion losses and other form of losses using an Explicit Loss Notification (ELN) mechanism.

TCP Veno [12], a combination of TCP Vegas [13] and TCP Reno [2], employs an end-to-end congestion control mechanism. If a packet is lost, TCP Veno employs proactive congestion control of TCP Vegas and, thus, distinguishes between the congested and non-congested network states. TCP Veno does not address the issue of burst errors and no corrective action is taken for wireless losses.

The differentiation between congestion and random losses in wireless networks is achieved by measuring the variation of round trip delay [14], [15]. If the loss is not due to congestion, TCP congestion control is suppressed and a modified recovery strategy is implemented. Two additional detection schemes [16], [17] are based on controlling TCP AIMD algorithm. No scheme imposes corrective actions in the case of wireless losses.

TCP Westwood [18] is another end-to-end variant of TCP that improves network performance in presence of lossy links such as wireless links and satellite links. Instead of setting congestion window size and slow start threshold based on packet drop information as done in conventional TCP, TCP Westwood estimates available bandwidth from the TCP sender and sets congestion window size and slow start threshold accordingly.

TCP-Real [19] is a receiver-oriented congestion control mechanism. If the network is congested, the receiver determines data sending rate and communicates that information to the sender. TCP-Real employs two corrective mechanisms: congestion avoidance and advanced error detection.

3.1.2 Link Layer Level

There have been several proposals for link layer based protocols. These protocols are most effective in handling wireless losses but they need expensive changes to be made in the intermediate routers and base stations.

This approach hides congestion related losses from the TCP sender and therefore requires no changes to existing sender implementation. The intuitive idea behind this approach is that since the problem is local, it should be solved locally. Hence, only the link layer is involved and the transport layer need not be aware of the characteristics of individual links. All link layer protocols attempt to make lossy link appear as a higher quality link with a reduced effective bandwidth. As a result, the TCP sender cannot see most of the losses caused other than congestion.

TCP Snoop [7] implements an agent (snoop agent) in the link layer of the base station. Segments from the fixed host are received at the base station and queued there before sending to the mobile host. If a packet is lost in the wireless link, a local retransmission is performed. This means the lost segment is retransmitted from the buffer at base station without letting the fixed host know. Thus, the TCP sender at the fixed host is unaware of the packet loss event and congestion control mechanism in not invoked. The major drawback of TCP Snoop is the memory requirement (per-connection buffer) at the base station.

SNACK-Snoop [20] combines SNACK with Snoop [7]. It uses SNACK to provide explicit wireless loss notification between a base station and a mobile host. This is a link layer based scheme and requires major modification at the intermediate base station. It also introduces processing and memory overhead of the Snoop protocol at the base station.

TCP-Jersey [21] incorporates available bandwidth estimation at the sender, as is the case in TCP Westwood [18]. TCP-Jersey improves network throughput by estimating bandwidth in the case of congestion losses. It differentiates congestion from non-congestion losses with the help from intermediate routers and, thus, requires expensive changes at the routers. It does not address corrections specific to wireless losses.

3.1.3 Split Connection

Split connection approach is in between end-to-end and link layer based protocols. This scheme completely hides the wireless losses from the sender by terminating the TCP connection at the base station. They use a separate reliable connection between the base station and the destination host. To perform well over the wireless link, the second connection can use techniques such as negative acknowledgement or selective acknowledgement, rather than only regular TCP.

Since, split connection based protocols maintain two separate connections: one for the wired part and another for the wireless part of the connection path, they result in poor end-to-end performance. No split connection based approach has used loss detection to improve TCP performance in wired/wireless networks. Examples of split-connection based approaches are M-TCP [9] and I-TCP [10].

Split-connection based methods lead to poor end-to-end network throughput due to shielding the wireless link from the wired part of the network. Link layer techniques are most effective in solving wireless link errors and improving TCP performance in wired/wireless networks. However, link layer based approaches often require large buffer space at the base station. End-to-end approaches can achieve significant performance improvement and does not require any modification in the intermediate routers. This approach is also simpler to implement, involves less processing and memory overhead compared to link layer based approach. We propose an end-to-end solution to improve TCP performance in wired/wireless networks. The proposed algorithm is discussed in Chapter 4.

4 SELECTIVE-TCP

4.1 Overview

We propose an end-to-end solution to improve TCP performance in the wired/wireless networks and we named this new algorithm as Selective-TCP. Selective-TCP algorithm is based on detecting the type of losses at the TCP receiver. It is implemented as an extension to TCP NewReno. If an out-of-sequence packet is received at the sink, Selective-TCP detects the cause as either loss due to congestion or loss due to wireless transmission error. The loss detection technique [22] used in Selective-TCP is based on packet inter-arrival times at the receiver and has shown good detection accuracy [22], [23]. After detecting the type of packet loss, two corrective measures are taken to improve TCP performance.

In the case of loss due to congestion, the bandwidth is measured at the receiver and sent to the sender, unlike estimating available bandwidth at the sender only [21], [18]. The sender then adjusts its congestion window size accordingly. Thus, Selective-TCP prevents TCP's AIMD scheme from setting the sender's congestion window size lower than necessary. Selective-TCP helps the TCP NewReno sender to achieve the optimum bandwidth faster, resulting in higher bandwidth utilization.

In the case of wireless transmission losses, receiver sends SNACK instead of duplicate acknowledgements. TCP NewReno's congestion control mechanism is not invoked. As a result, the slow start threshold and congestion window size are not reset unnecessarily, resulting in better bandwidth utilization.

4.2 The Proposed Algorithm

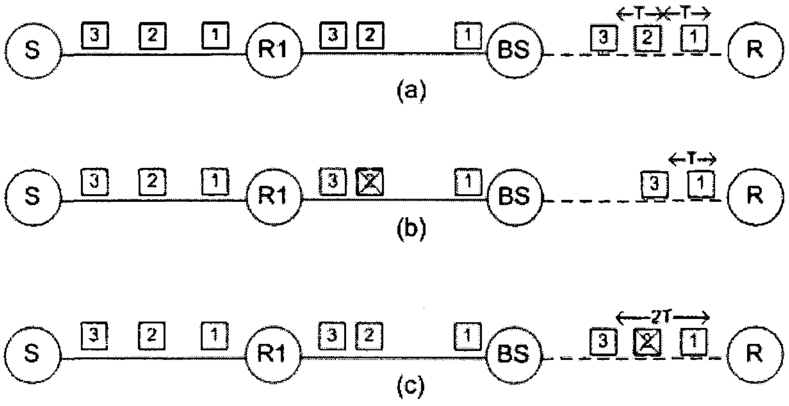
4.2.1 Loss Detection Mechanism

The loss detection technique used in Selective-TCP is based on the packet inter-arrival times at receiver. The technique assumes that the wireless link is the only network bottleneck, that it is the last hop in the connection path, and that sender performs bulk data transfer. These assumptions are often valid for wired/wireless networks such as cellular networks [22]. The wireless link is the only bottleneck in the network and, hence, packets accumulate at the base station. Therefore, most packets will be sent back-to-back over the wireless link. Fig. 8(a) shows the case of no packet loss. If T is the minimum packet inter-arrival time at the receiver, the inter-arrival time between two consecutive packets is ~T (assuming all packets are of same size). If a packet is lost in the wired link, the packet inter-arrival gap is still ~T because the packets queue at base station before being transmitted on the wireless link, as shown in Fig. 8(b). However, if a packet is lost in the wireless link, the inter-arrival gap at receiver is ~2T because the lost packet has travelled on the wireless link for some time before being lost, as shown in Fig. 8(c). Using this heuristic, the type of packet loss is detected according to Algorithm 1.

Algorithm 1. Pseudo-code of the algorithm for detecting type of packet loss [22].

```
n = number of packets lost between two packet arrivals
if ( packet loss){
            if ((n+1)T ≤ packet inter-arrival time < (n+2)T)
                  wireless loss
            else
                  congestion loss
}
```

Figure 8: **Packet inter-arrival gaps between consecutive TCP packets. (a) no packet loss, (b) packet loss in the wired link, and (c) packet loss in the wireless link. R1 is an intermediate router and BS is the base station for the wireless destination node R. Solid and dashed lines represent wired and wireless links, respectively.**



4.2.2 Module at the TCP Receiver

If an out-of-sequence packet arrives at a TCP receiver, Selective-TCP first distinguishes the type of packet loss.

In the case of congestion loss, Selective-TCP increments a counter *congestion_count.* When a threshold value k is reached, the receiver measures the bandwidth and sends it to the sender for setting the congestion window size, rather than search for the optimum congestion window size and wait for TCP to initiate congestion control. The threshold value for congestion_count, k is experimentally chosen to be equal to 10. This value is critical in deciding when the measured bandwidth is sent to the sender and when congestion window size at the sender is being set. If the congestion window size is set before TCP sends 3 duplicate acknowledgements and reduces congestion window size, it will not be helpful in terms of goodput/throughput performance. On the other hand, setting the congestion window size long after TCP AIMD algorithm has reduced it, will not be helpful as well. Experiments show any lower/ higher value than 10 deteriorates network performance, so we use k = 10 throughout the simulations.

Bandwidth is measured as: (*no. of received packets* × *size of packets in bits*)/(*inter-arrival time between previous in-sequence packet received and most recent in-sequence packet received* × 1000) in Kbps. To explain this further, bandwidth of a network is defined as the data rate supported by a network connection. TCP-Westwood [18] estimates the available bandwidth at the sender side based on the interval of returning ACKs:

$$b_k = \frac{d_k}{(t_k - t_{k-1})},$$

where, d_k is the amount of acknowledged data at time t_k and t_{k-1} is the time when previous acknowledgement was received. This sample bandwidth is smoothed further by a low-pass filtering to obtain estimated bandwidth.

TCP-Jersey [21] adopts the same idea. It employs time-sliding window (TSW) estimator at the sender and estimated available bandwidth as:

$$R_n = RTT * R_{n-1} + \frac{L_n}{(t_n - t_{n-1})} + RTT,$$

where, R_n is the estimated bandwidth when the n-th acknowledgement arrives at time t_n. t_{n-1} is the time when previous acknowledgement arrived. L_n is the size of data acknowledged by the n-th acknowledgement and RTT is the TCP's estimation of the end-to-end RTT delay at time t_n. Since duplicate acknowledgements also account for available bandwidth, both these bandwidth estimation approaches [21], [18] consider duplicate ACKs in the cases of packet loss in the network.

Unlike TCP Westwood and TCP-Jersey, Selective-TCP estimates bandwidth at the receiver and, hence, it measures the available bandwidth as the number of packets received in a given period of time. In cases of packet loss, the available bandwidth would be smaller since fewer packets will be received than in the case of no packet loss. Selective-TCP measures available bandwidth as:

$$BW = \frac{n_p * s_p * 8}{(t_n - t_{n-1}) * 1{,}000} Kbps,$$

where, t_n is the time when most recent in-sequence packet is received, t_{n-1} is the time when previous in-sequence packet was received, n_p is the number of packets received within $(t_n - t_{n-1})$, and s_p is the size of packets in bytes.

In the case of wireless loss, the receiver sends ACK with SNACK option to the sender. As a consequence, the TCP sender retransmits the missing packets indicated by SNACK. Acknowledgments with SNACK options are sent after a certain delay (*snack_delay*). As a result the chance of unnecessarily retransmitting a delayed or misordered segment is limited [24}. Since the SNACK option triggers a retransmission, there is no reliance on the Fast Retransmit algorithm to detect the loss. This independence from the Fast Retransmit algorithm is important because duplicate ACKs may never be received when operating over a highly lossy link. The pseudo-code is shown in Algorithm 2. The default value of *snack_delay* in the implementation is 50 ms. The value of this delay depends on mission requirements. We also used other values for *snack_delay* such as 35 ms and 25 ms. Nevertheless, in these cases network performance deteriorated compared to delaying SNACK for 50 ms. Hence, in simulations we used 50 ms as the *snack_delay.*

Algorithm 2: Pseudo-code of Selective-TCP at the receiver.

```
if (out-of-order packet received) {
        // check type of loss
        if ( wireless loss) {
                // before sending SNACK, wait for the snack_delay to be over
                // initial value of snack_delay = 50 ms
                if (snack_delay = 0)
                        send SNACK
                else
                        do nothing
        }
        else { // congestion loss
                1) set congestion_count = congestion_count + 1
                2) set congestion_info = current bandwidth  measured at the TCP
                receiver
                        if (congestion_count = k) {
                                1) send congestion_info to the TCP sender
                        2) reset  congestion_count
                        }
                        else
                                send ACK //as in the case of TCP sink
        }
}
else // in-sequence packet received
        send ACK //same as TCP sink
```

4.2.3 Module at the TCP Sender

When a SNACK is received, the TCP sender aggressively retransmits the packet(s) indicated as lost packet(s), without waiting for retransmission timeout to occur. Hence, congestion control mechanism and unnecessary retransmissions are avoided, leading to higher bandwidth utilization. *Congension_info* stores the bandwidth measured at the receiver when a packet loss is detected. If the *congestion_info* field in the TCP header has a non-zero value, the sender sets its congestion window size equal to *congension_info*base_rtt*, where *base_rtt* is the initial round trip time. This prevents the TCP AIMD algorithm from setting the congestion window size to be unnecessarily small. *Congestion_info* is multiplied by *base_rtt* to increase the congestion window size. The pseudo-code is shown in Algorithm 3.

Algorithm 3: Pseudo-code of Selective-TCP at the sender.

```
if (SNACK received) {
        1) retransmit packet(s) indicated as lost
        2) reset retransmission timer
        }
else if (congestion_info ≠ 0) {
        // set size of congestion window equal tothe bandwidth
// measured at receiver
        1) set cwnd_ = congestion_info * base_rtt
        // cwnd_ denotes congestion window size and base_rtt is
        // the base round trip time measured at sender
        2) reset congestion_info
        }
else // standard ACK received
        do as TCP NewReno sender
```

Congestion_info is a C++ variable of data type "double". Hence, the size of the *congestion_info* is 64 bits (8 bytes), which is equal to one octet. In actual implementation of Selective-TCP, an octet of data has to be appended to the optional data area of the TCP header. The TCP header was shown in Fig. 4.

5 IMPLEMENTATION OF SELECTIVE-TCP IN NS-2

The simulations are performed using the ns-2 network simulator version 2.27. In this Chapter, we introduce the ns-2 simulation tool along with the implementation details of the Selective-TCP algorithm.

5.1 Introduction to ns-2

Ns-2 [25] is a discrete event network simulator developed at the University of California at Berkeley (UCB). It began as a variant of the Real network simulator developed in 1989. It is currently supported through Defence Advanced Research Project Agency (DARPA) and National Science Foundation (NSF).

Ns-2 takes full advantage of the features of object-oriented programming. It is written in C++ and OTcl. Although it does not guarantee production of a faithful replica of the real world, it does try to model most of the protocol behaviour accurately and can be used to study various protocols at different levels of the OSI layers. It is focused on modelling network protocols including wired, wireless and satellite networks with transport protocols such as TCP, and UDP with both unicasting and multicasting capabilities. It models Web, Telnet, and FTP applications. It also includes the implementation of ad-hoc routing and sensor networks. It provides provisions for gathering statistics, tracing, and error modelling for the simulations carried out. Apart from the core code of the ns-2, there have been numerous contributions from other researchers. We use ns-2 to perform the network simulations. The simulation results were used to evaluate the proposed Selective-TCP algorithm. We chose ns-2 for the implementation because it is a freely distributed code and supports many interesting protocols. The architectural design of ns-2 is rather extensible.

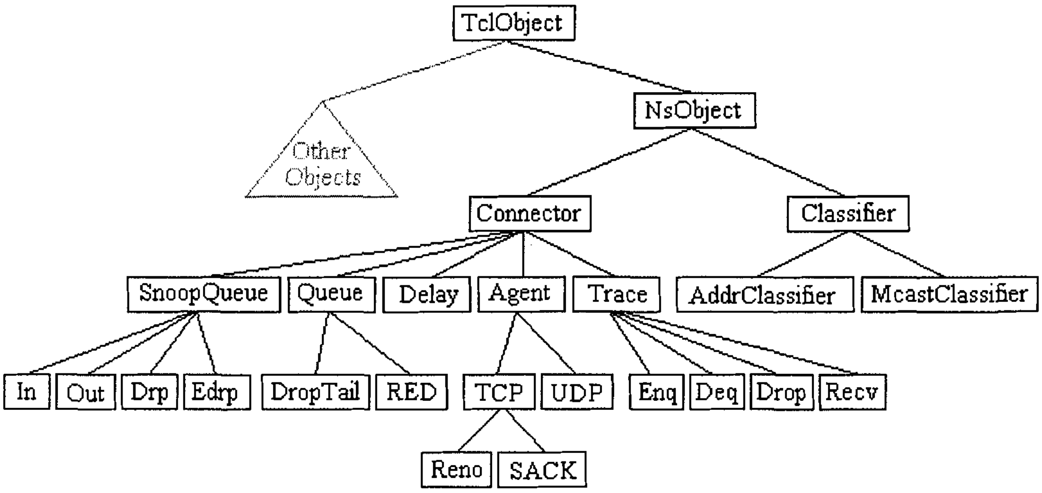
C++ and Tcl are the two languages used in ns-2. Two languages are needed to perform complex programming, coupled with the need for speed when we vary parameters/configurations to explore a large number of scenarios while studying the various protocols. C++ is fast to execute, nevertheless it is slow to change, making it suitable for the complicated protocol implementation. However, it is very slow when varying parameters and rerunning simulations. Tcl, on the other hand, is much slower but very convenient for varying simulation parameters. Consequently, C++ is used for implementing properties of the protocol while Tcl is used to implement code that needs to be changed often in order to study the protocol behaviour.

There are six most important classes when linking the C++ and OTcl code. They provide the necessary connection to interact with the code from the other language. These six classes are Class Tcl, Class TclClass, Class TclObject, Class TclCommand, Class EmbeddedTcl, and Class Instvar.

5.1.1 Class Hierarchy in ns-2

Fig. 9 shows a glimpse of the ns-2 architecture. The root of the hierarchy is the class TclObject. It is the superclass of all OTcl libray objects such as scheduler, network components, timers, and other objects (NAM). The simulator has a class hierarchy in C++ (compiled hierarchy) and a corresponding class hierarchy in OTcl (interpreted hierarchy). Both these hierarchies are closely related.

Figure 9: ns-2 class hierarchy (partial).



Source: NS by example [26], reprinted by permission.

5.1.2 Agents in ns-2

Agents, being the end points in a network, are also end points in ns-2. The class Agent is the base class, which is partly implemented in OTcl and partly in C++. There are various protocol agents in ns-2, including the basic agents such as TCP and UDP. The agents are usually created through Tcl during a simulation. In this case, the constructor for the agent in the compiled code is executed. The binding is then performed in the class.

The main tasks performed by these agents are processing the requests and responses at the sender and the receiver. They also implement a timer class if necessary. To create a new agent, we have to first decide on the inheritance structure and create appropriate class definitions, define recv() and timeout() methods, define any necessary timer classes, define the OTcl linkage functions, and write the necessary OTcl code to access the agent. The pre-existing agents of ns-2 provide an excellent base for extending various other complicated protocols.

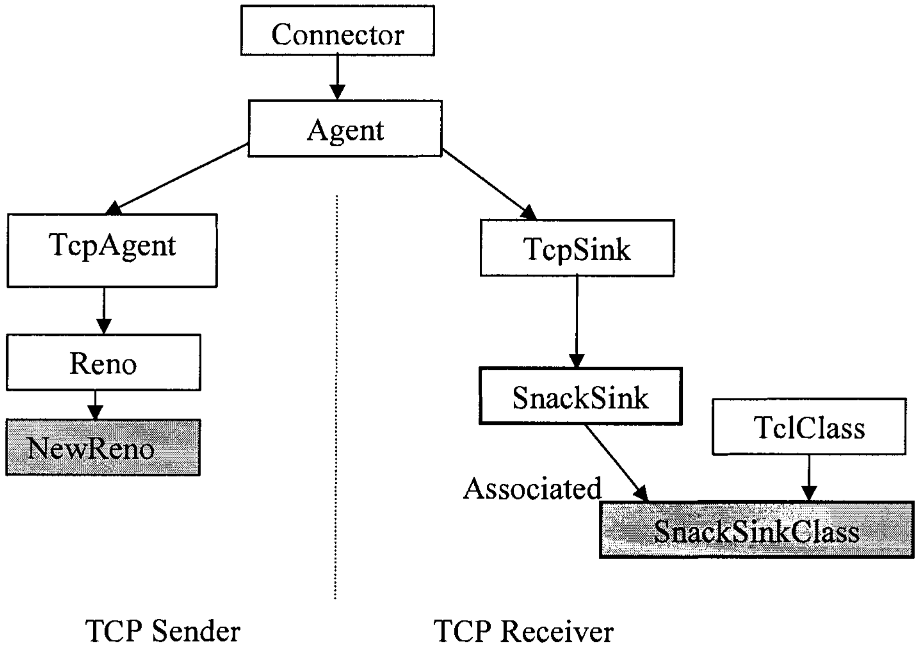
5.2 Implementation of Selective-TCP algorithm

The implementation of Selective-TCP algorithm is based on the SNACK module [27] developed for ns-2. The SNACK module provides a SNACK processing module in the TCP NewReno sender and a SNACK generating module in the TCP sink. The Selective-TCP algorithm is implemented in the SNACK module. Necessary modifications are made in the agent named class SnackSink and its recv() function, which is the main reception path for packets and provides various other necessary methods. Fig. 10 shows the implementation hierarchy and the important C++ classes for Selective-TCP implementation are shown with darker shade.

At the sender side, the class TCPNewReno is modified so that the congested/non-congested state of the network can be determined from TCP header option *congestion_info* and the congestion window size *cwnd* is set if required. Necessary modifications are made in the files tcp.h (for introducing the option field *congestion_info*) and newreno.cc.

At the receiver side, the class SnackSink is modified. The recv() function of this class is extended to introduce packet loss detection mechanism, setting the *congestion_info* of TCP header, and invoking sendSnack process if required by the algorithm. Necessary modifications are made in the files tcp-sink.h and tcp-sink.cc.

Figure 10: Selective-TCP implementation overview: C++ classes shown with dark shade are modified.



6 PERFORMANCE EVALUATION

In this Chapter, we present the performance evaluation of Selective-TCP algorithm using network simulator ns-2. We discuss the error model used to simulate the burst error in the wireless links. We also discuss the simulation topology and the simulation parameters. Next, we describe the simulation scenarios followed by the explanation of simulation results.

6.1 Error Model

We simulate realistic wireless links with burst errors [28] using a two-state Markov model (known as the Gilbert model), shown in Fig. 11. It has a *good* (error-free) and a *bad* (erroneous) *state*. In our simulations, *good state* implies no packet loss while *bad state* denotes 1 packet loss.

This model is defined by a transition probability matrix π and a steady state error rate ε ([29]). The transition probability matrix of this two-state Markov model is given by

$$\prod = \begin{bmatrix} p & 1-p \\ 1-q & q \end{bmatrix}. \qquad (1)$$

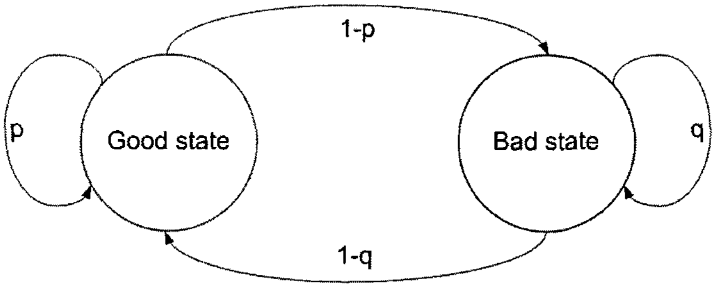
The average error rate is given by

$$\varepsilon = \frac{1-p}{2-p-q}. \qquad (2)$$

The average lengths of good state (L_{good}) and bad state (L_{bad}) are:

$$L_{good} = \frac{1}{1-p} \text{ and } L_{bad} = \frac{1}{1-q}. \qquad (3)$$

Figure 11: Two-state Markov model.
p is the probability of successfully transmitting a packet given the previous packet was successfully transmitted; 1-q is the probability of successfully transmitting a packet given that the previous packet was dropped.



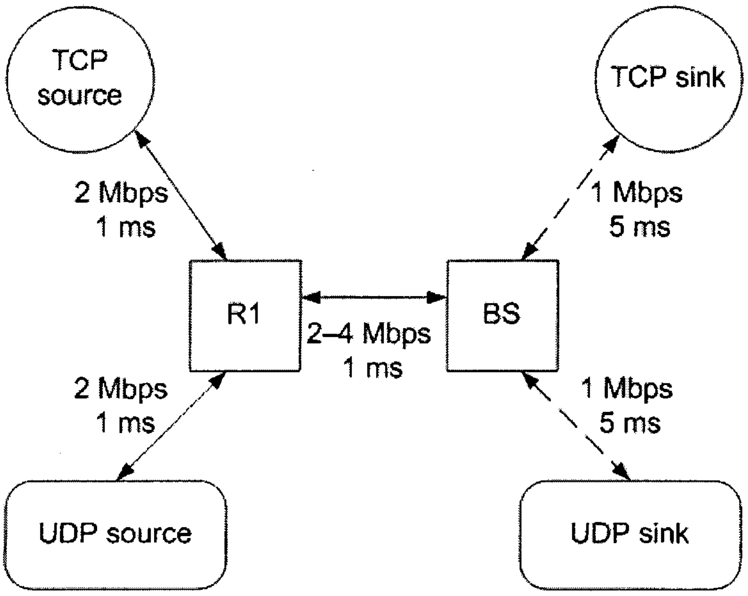
The wireless link is assumed to be in one of the two states. We assume that the wireless link is in the good state at the beginning of simulation. The transitional probabilities p = 0.9913 and q = 0.8509 model the effect of burst errors [30]. The error rate ε = 5%. These parameters present a close replication of burst errors in real wireless networks [30].

6.2 Simulation Results

6.2.1 Network Topology

The network (dumbbell) topology is shown in Fig. 12. The TCP sender is a wired node, while the TCP receiver is a wireless node. TCP source sends file transfer protocol (FTP) traffic. The user datagram protocol (UDP) source sends constant bit rate (CBR) traffic. The UDP source is a wired node, while the UDP sink is a wireless node. The FTP traffic and the CBR traffic share a common wired link from router R1 to base station BS, as shown in Fig. 12. The TCP and UDP sender rates are 2 Mbps and 512 Kbps, respectively.

Figure 12: Simulated network topology.
R1 is an intermediate router and BS is the base station for the wireless destination nodes. Solid and dashed lines represent wired and wireless links, respectively.



Wired links have 2 Mbps bandwidth. Bandwidth of the wired link between R1 and BS is: (i) 2 Mbps when examining Selective-TCP's performance in presence of congestion (the sum of TCP and UDP data rates is 2.5 Mbps) and (ii) 4 Mbps when examining the case without congestion. Propagation delay of the wired links is 1 ms. Wireless links have 1 Mbps bandwidth and 5 ms propagation delay. Hence, wireless links are the network bottlenecks.

6.2.2 Simulation Scenarios

We compare performance of Selective-TCP and TCP NewReno in the presence and in the absence of a congested link. In both cases, the 5% burst error in wireless link has been introduced. We also study the goodput performance of Selective-TCP with no wireless error, 1% random error (random statistical error), and 5% burst error (continuous lacking of data). There is only CBR/UDP traffic for the first 100 s of simulation time. After 100 s, TCP connection starts and exists along with UDP connection. All connections end after 300 s of simulation time.

The performance measures we consider are throughput, goodput, and size of congestion window. Throughput is defined as the number of bits transmitted by the source host and it is presented in kbps. Goodput is defined as the number of bits received by the destination host, less the duplicates. Goodput can also be indicated as the maximum sequence number of packets reached at the destination. We use the later to represent goodput in the simulation results.

6.2.2.1 Presence of a Congested Link

We first investigate Selective-TCP's performance in the presence of congestion in the network. To simulate congestion, we used a 2 Mbps link as the common wired link between router R1 and base station BS. The rate of data through this link is 2.5 Mbps.

Goodput (the number of bits received at the destination host, less duplicates) in the presence of a congested link is shown in Fig. 13. We represent goodput as the maximum sequence number of packets received at the receiver. Selective-TCP achieves up to 45% improvement when compared to NewReno. Selective-TCP shows larger congestion window size than TCP NewReno, indicating better utilization of available bandwidth, as shown in Fig. 14. The slow start threshold and the average network throughput for Selective-TCP are shown in Figs. 15 and 16, respectively.

Figure 13: Goodput vs. time: goodput is represented as the maximum number of packets reached their destination.

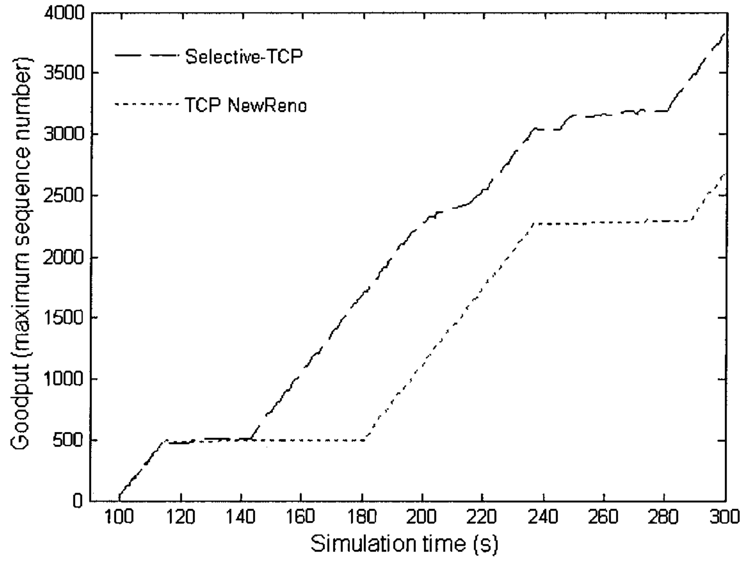


Figure 14: Congestion window size for Selective-TCP is significantly larger than that of TCP NewReno.

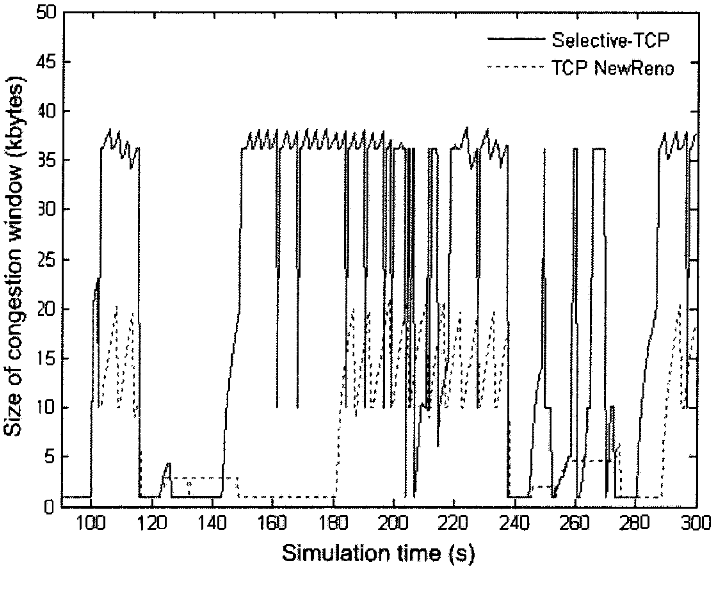


Figure 15: Compared to TCP NewReno, a Selective-TCP sender maintains a constant value of slow start threshold over a longer period of time. The initial value of the slow start threshold is equal to 20.

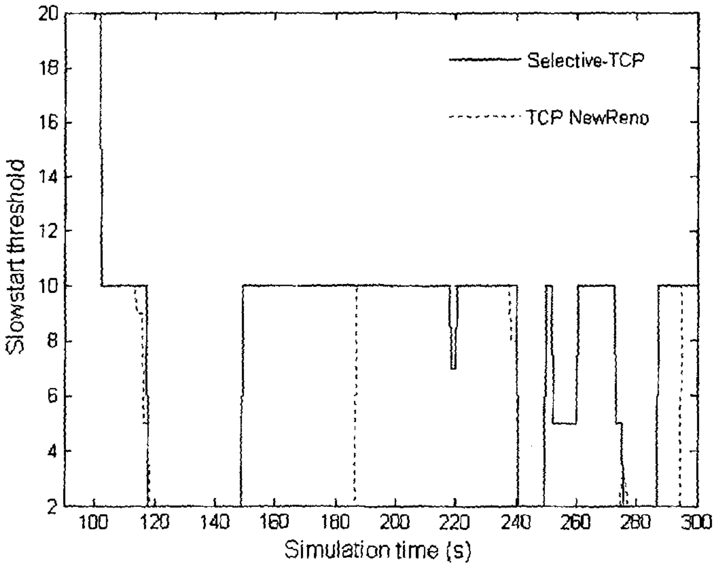
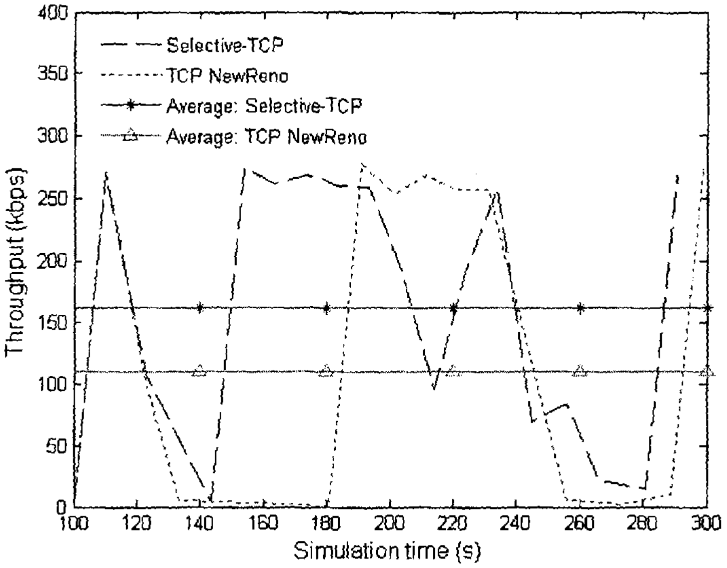
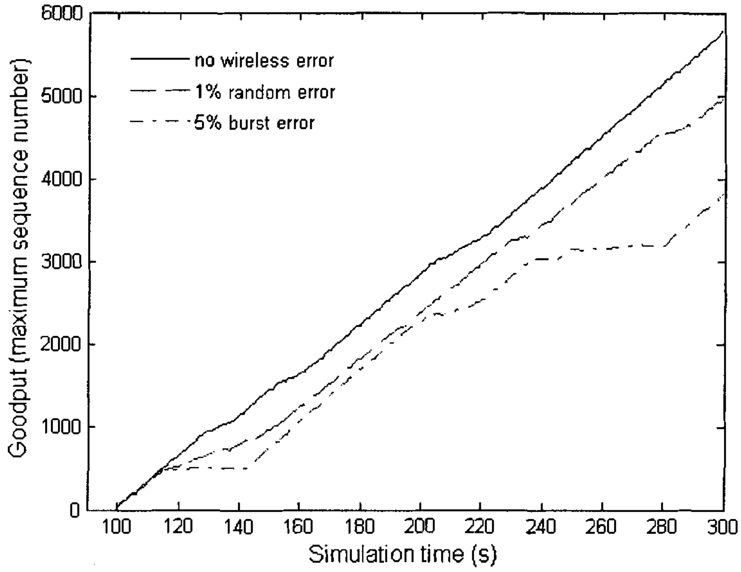


Figure 16: The average throughput of Selective-TCP (161.5 kbps) is larger than the average throughput of TCP NewReno (110.91 kbps).



A comparison of goodput performance of Selective-TCP, without wireless error, with 1% random error (random statistical error), and with 5% burst error (continuous lacking of data), is shown in Fig. 17. As expected, the goodput performance is best when there is no wireless error. It seems the performance of Selective-TCP without wireless error and TCP NewReno should be exactly same, however, they differ (maximum ~5%). This is because the loss detection mechanism [22] used in Selective-TCP is not 100% precise. Its accuracy of detection is ~95%, which is the reason that some congestion losses are detected as wireless losses and vice versa.

Figure 17: Goodput of Selective-TCP: maximum goodput is achieved when no wireless error is introduced.



6.2.2.2 Absence of a Congested Link

Next, we investigate Selective-TCP's performance without presence of congestion in the network. To simulate non-congested link, we have used a 4 Mbps link as the common wired link while the total rate of data through this link is 2.5 Mbps.

Goodput vs. simulation time without congestion in the common wired link is shown in Fig. 18. Goodput improves by 45% than TCP NewReno over 300 s of simulation. Congestion window size, slow start threshold, and throughput as functions of time are shown in Figs. 19, 20, and 21, respectively. We compare performance of Selective-TCP and TCP NewReno in a non-congested network. Similar to the case of congested network, Selective-TCP performs better than NewReno. If no error is introduced in wireless link, Selective-TCP achieves ~1.5 times the goodput in the case of 5% burst error, as shown in Fig. 22.

Figure 18: Selective-TCP shows significant increase in goodput.

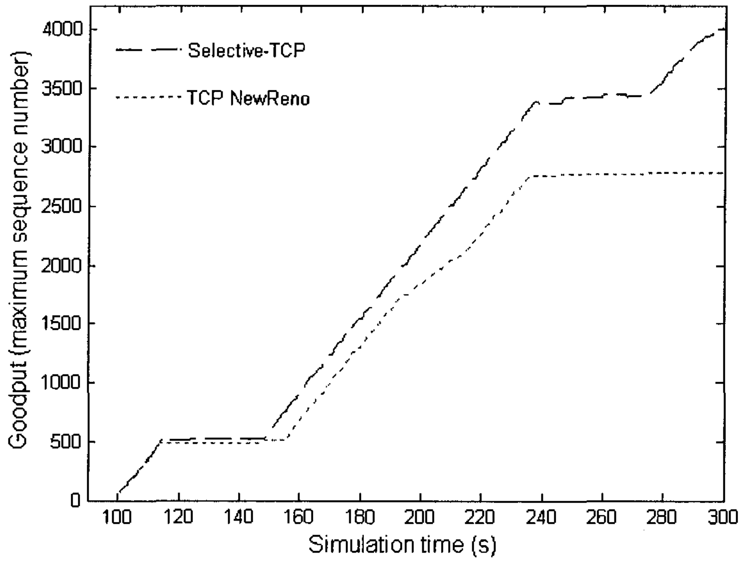


Figure 19: The size of congestion window for Selective-TCP remains larger than for TCP NewReno.

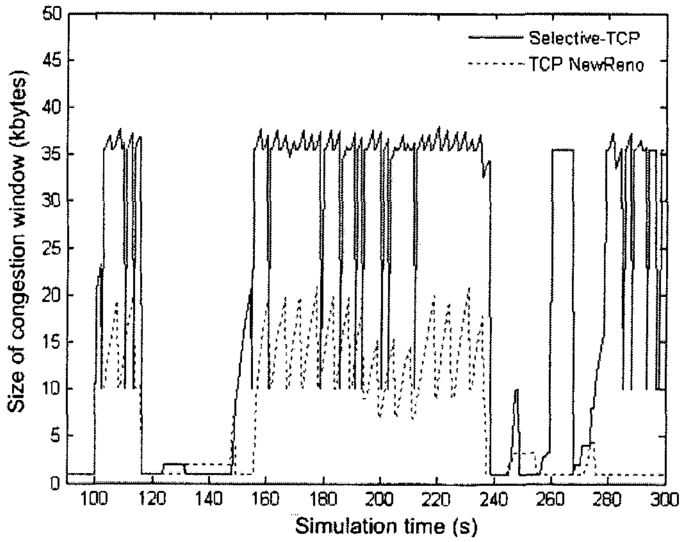


Figure 20: Slow start threshold of Selective-TCP remains constant over a longer connection period.

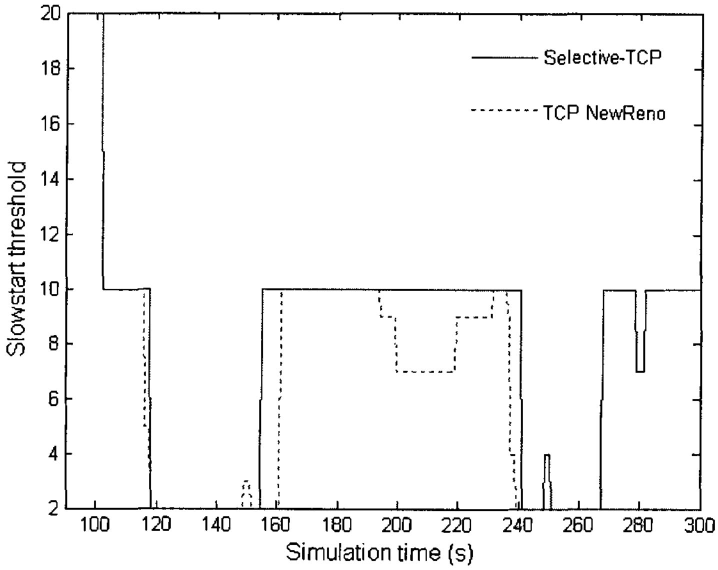


Figure 21: Average throughput of Selective-TCP (169.15 kbps) and of TCP NewReno (115.65 kbps).

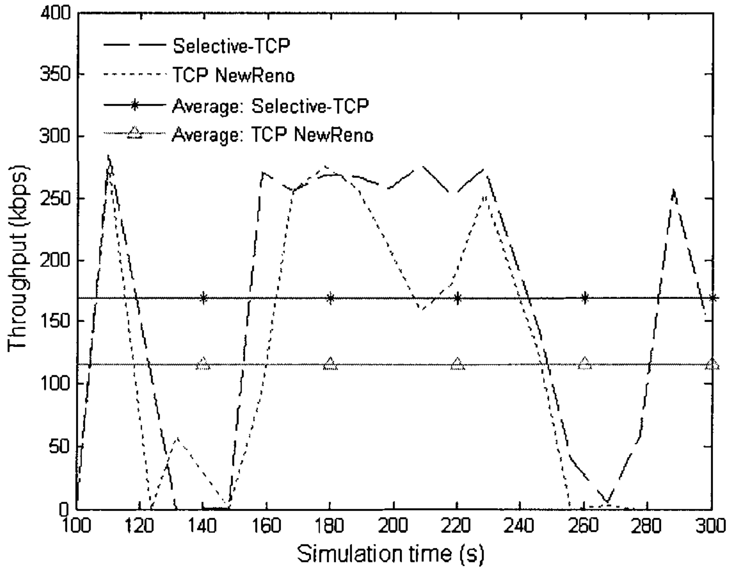
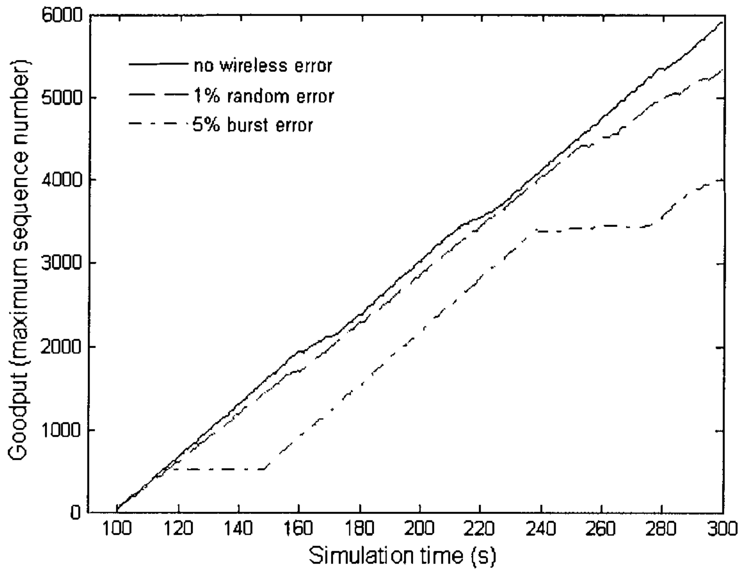


Figure 22: Effect of wireless errors: goodput of Selective-TCP in the absence of congested link.



In general, Selective-TCP shows better goodput and throughput in the presence of congested link.

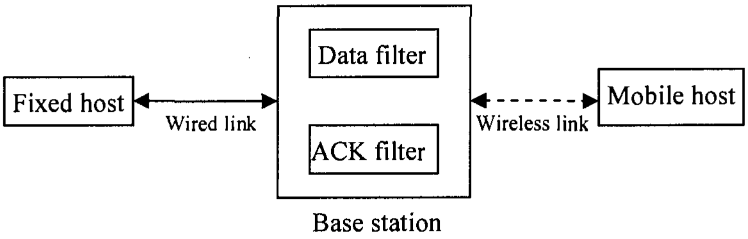
6.2.3 Comparison of Selective-TCP and TCP Packet Control Algorithm

In this Section, we present a brief description of TCP packet control algorithm [8] and compare its performance to Selective-TCP. As in the Section 6.2.2, we discuss the cases of congested link and non-congested link, each with 5% burst error, 1% random error, and no wireless error in the wireless link.

6.2.3.1 TCP Packet Control Algorithm

TCP packet control algorithm is designed as an option for TCP rather than a modification of TCP. It is a link layer based approach to improve TCP's performance in the wireless networks and, hence, requires modifications only in the base stations. It addresses two problems specific to wireless networks: delay variations (causing spurious fast retransmit) and sudden large delays (causing spurious fast timeout). This algorithm hides wireless losses from the fixed host or the TCP sender. To deal with these two problems, two filters at the base station called Data filter and ACK filter are introduced. These two filters improve TCP performance in mixed wireline/wireless networks by dealing with the wireless links with long sudden delays, delay variations and maintaining regular TCP functions. These filters keep track of TCP data and ACK packets received from the fixed host and the mobile host, respectively. They then forward packets to both client ends based on the information gathered in the base station. They do not depend on end-user TCP flavours. Packet control filters are shown in Fig. 23.

Figure 23: TCP packet control algorithm: two filters are introduced at the base station.



1. ACK Filter: Packet control algorithm reacts to ACKs received from the mobile host using the ACK filter. It drops the old ACKs and duplicate ACKs according to the duplicate ACK threshold defined by the user. It remembers the last new ACK received from the wireless receiver, called the last received ACK. When an ACK arrives, its ACK number is checked against the last received ACK. Three cases are considered:

Old ACK: The ACK is considered old if the ACK number has already been received and/or is smaller than the last received ACK. It is immediately dropped.

Duplicate ACK: If the newly received ACK number is identical to the largest ACK currently received, it is considered a duplicate ACK. Packet control algorithm keeps track of the current number of duplicate ACKs received at the BS. Based on the number of duplicate ACKs received and the user-defined duplicate ACK threshold, duplicate ACKs are evenly dropped and are not sent to the sender. The number of ACKs to be dropped is equal to the difference between the user-defined duplicate ACK thresholds at the BS and at the mobile host. For example, if the user-defined duplicate ACK threshold is 6 and TCP has defined the three duplicate ACK threshold, every second duplicate ACK is dropped.

New ACK: If the ACK number has not been received before, the ACK is considered new. The last wireless ACK is updated, the counter for the current number of duplicate ACKs is reset, and the ACK is forwarded to the sender.

The design of the ACK filter is based on the observation that a wireless link has a high number of re-ordered segments, which is the primary cause of spurious fast retransmit. By filtering some duplicate ACKs at the BS, the spurious fast retransmit may be reduced. If there is no packet loss in the network, filtering duplicate ACKs results in better TCP performance. The pseudo-code is shown in Algorithm 4.

Algorithm 4: Pseudo-code for the TCP packet control algorithm: data filter

```
if (new or unacknowledged data segment)
        forward to receiver (mobile host)
else // acknowledged data segment
        drop the segment
```

2. Data Filter: When the data filter receives a data segment from the fixed host, it passes it to the mobile host. The data filter at the base station is designed to prevent the spurious fast retransmit caused by spurious timeout. In the case of spurious timeout, retransmissions of the unacknowledged segments unnecessarily consume the scarce wireless link bandwidth and trigger additional spurious fast retransmits. Therefore, their prevention is essential in solving spurious timeout. The data filter checks whether data segments have been acknowledged or not. The sequence number is checked against the last ACK received from the receiver. Two cases are considered:

New data segment or unacknowledged segment: If the segment has not been acknowledged, it is forwarded to the receiver. The segment is either a new data segment or an unacknowledged segment. In the latter case, the system cannot distinguish whether the last transmission of the same segment has been received by the receiver or its ACK was lost. In both cases, even if the received segment is a retransmission, it is forwarded.

Acknowledged segment: This segment is a retransmission due to spurious timeout. This occurs because the ACK from the base station is lost or has not arrived at the mobile host. In both cases, the segment is dropped. We consider that a loss of ACKs could occur even though the BER and the possibility of congestion for ACKs are small in wireline networks. For every two identical retransmitted segments received, an ACK is sent from the base station to the sender. Hence, unnecessary retransmissions are eliminated and the problem of lost ACKs is resolved. The pseudo-code is shown in Algorithm 5.

Algorithm 5: Pseudo-code for the TCP packet control algorithm: ACK filter

```
if (old ACK received)
        drop the ACK
else if  (new ACK received) {
        1) update last_received_ACK
        2i) reset number of DUP_ACKs to 0
        3) forward the ACK to fixed host
}
else   { // duplicate ACK received
        1) update number of DUP_ACKs
        2) drop or forward the duplicate ACKs depending
        on user-defined DUPACK_threshold
}
```

6.2.3.2 Comparison in the Presence of a Congested Link

We compare the performance of Selective-TCP and TCP packet control algorithm with the network topology as described in Section 5.3.1. The simulation scenarios are described in Section 5.3.2. We first investigate both the algorithms in presence of congestion in the network. We again use a 2 Mbps wired link as the link between intermediate router R1 and base station BS, as shown in Fig. 12.

Figs. 24 and 25 show plots of goodput in terms of maximum number of packets received at the receiver over 300 s of simulation and 600 s of simulation time, respectively. These graphs show that the performance of Selective-TCP and the TCP packet control algorithm are comparable, Selective-TCP being better for long connections. In both cases, both algorithms perform (~ 40% – 50%) better than TCP NewReno.

Figure 24: Goodput vs. time: 300 s of simulation.

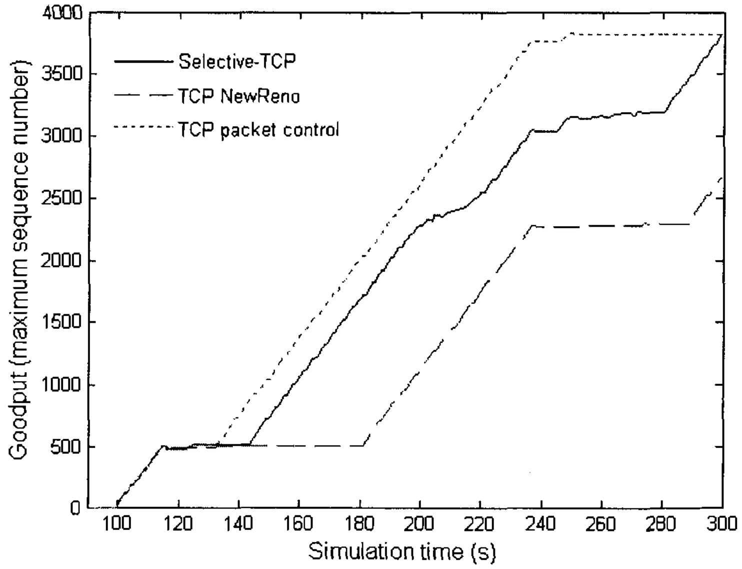
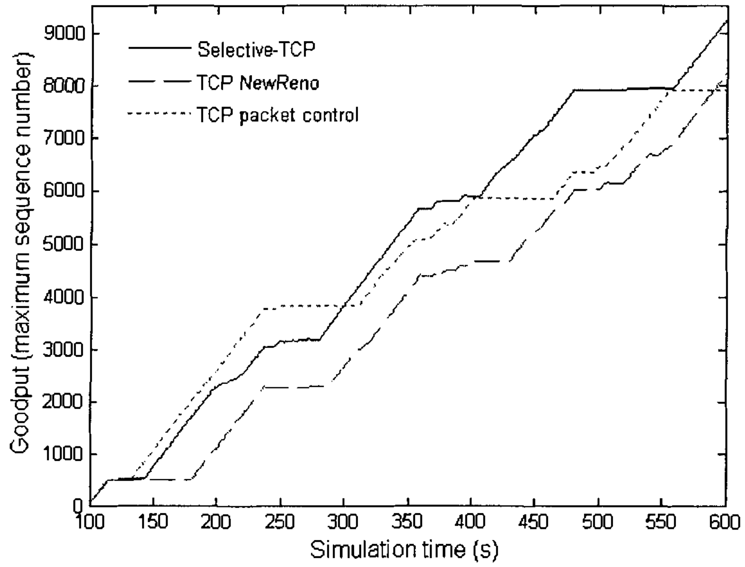


Figure 25: Goodput vs. time: 600 s of simulation.



Figs. 26 and 27 show the variation of congestion window size and slow start threshold, respectively. Again, both Selective-TCP and TCP packet control algorithm perform much better than TCP NewReno, Selective-TCP being slightly better than TCP packet control algorithm. The throughput values are compared in Fig. 28. Effect of introducing various wireless errors is shown in Fig. 29.

Figure 26: Congestion window size for Selective-TCP is larger than TCP packet control algorithm that has larger congestion window size than TCP NewReno.

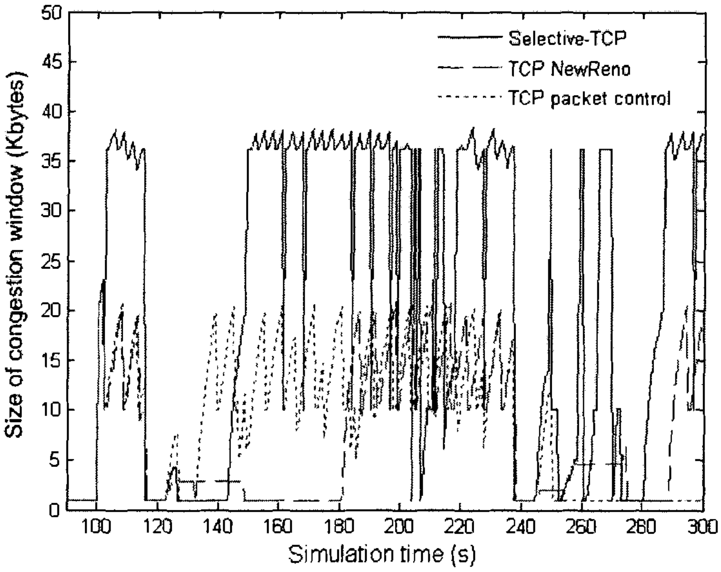


Figure 27: Slow start threshold vs. simulation time.

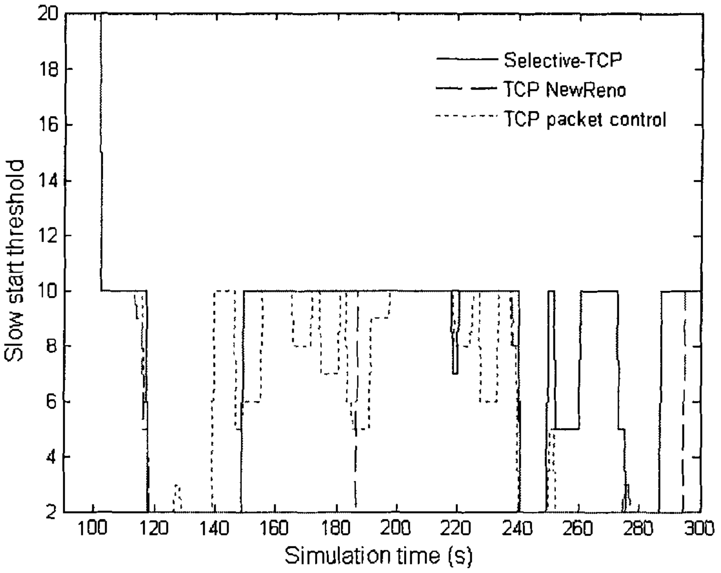


Figure 28: Average throughputs for Selective-TCP and TCP packet control algorithm are almost identical (~160 kbps).

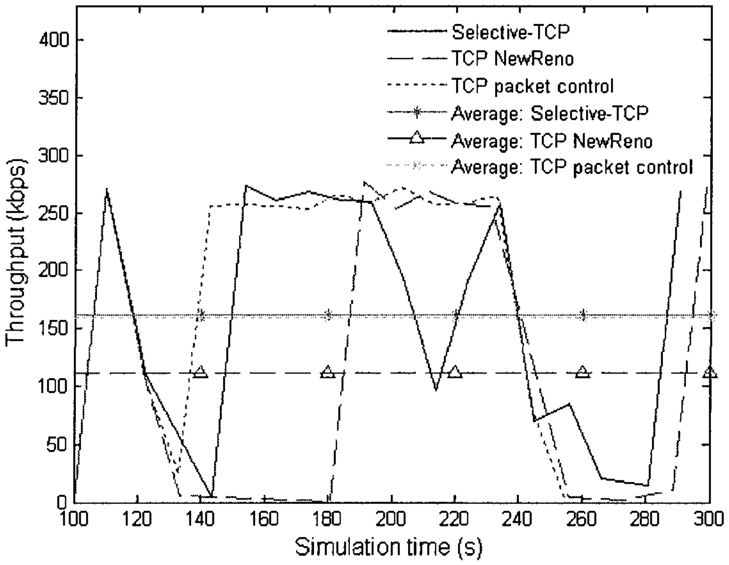
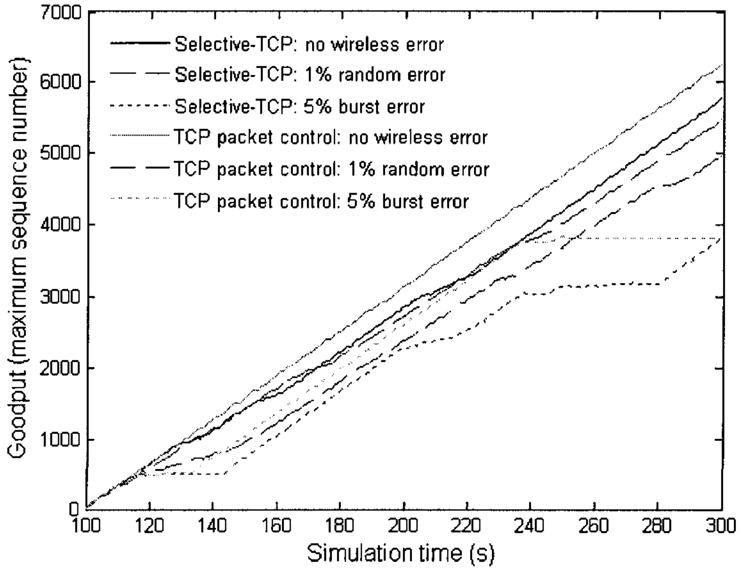


Figure 29: Effect of wireless errors on Selective-TCP and TCP packet control algorithm.



6.2.3.3 Comparison in the Absence of a Congested Link

We now compare Selective-TCP and TCP packet control algorithm without presence of congested link in the connection path between TCP sender and receiver. We again use a 4 Mbps wired link between intermediate router R1 and base station BS, as shown in Fig. 12.

In the absence of a congested link, Selective-TCP performs significantly better than TCP packet control algorithm, as shown in Fig. 30. Variation of congestion window size and slow start threshold are shown in Figs. 31 and 32, respectively. Throughput is compared in Fig. 33. In this case, average throughput for Selective-TCP is ~30% better than for TCP packet control algorithm. In contrast, in the case of congestion, both algorithms had almost identical average throughput.

Figure 30: Goodput performance in the absence of congested link: Selective-TCP performs better than TCP packet control algorithm.

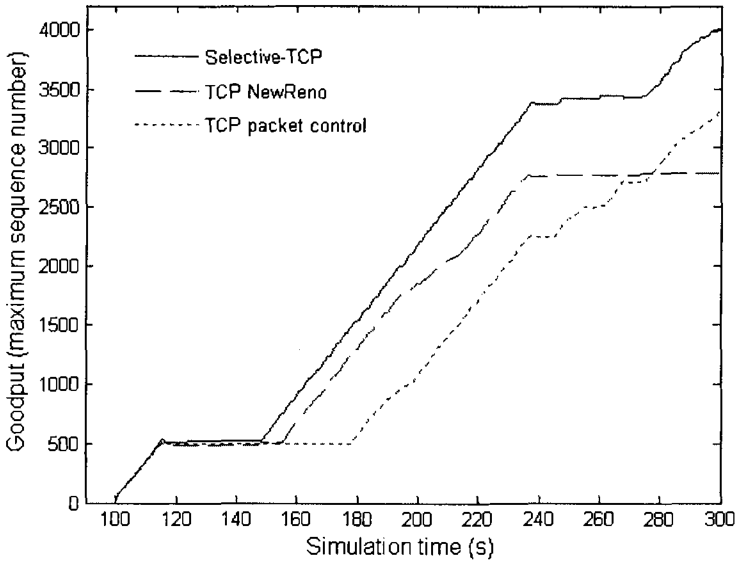


Figure 31: Congestion window vs. simulation time.

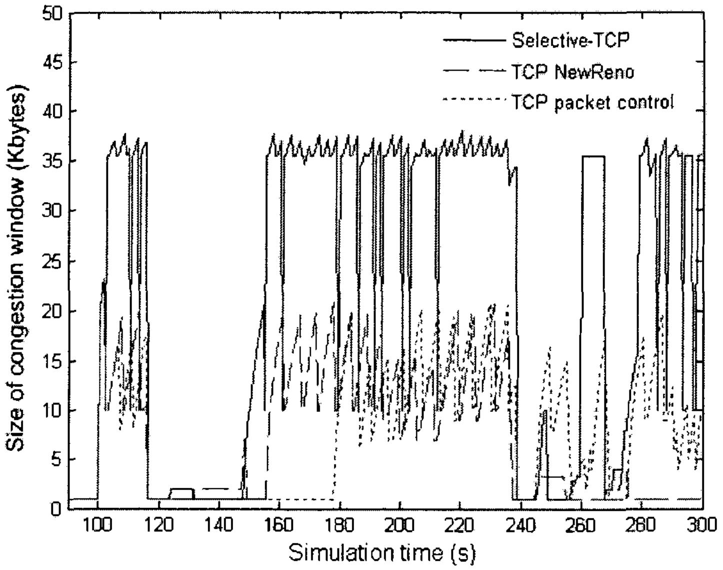


Figure 32: Slow start threshold vs. time.

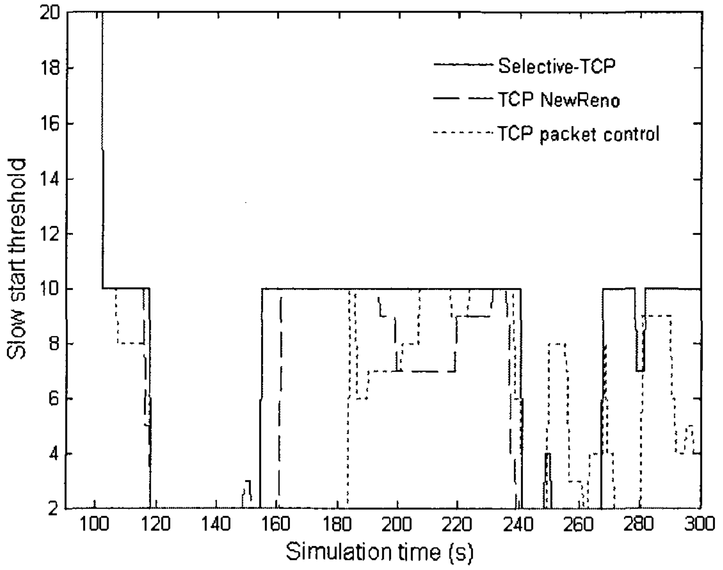
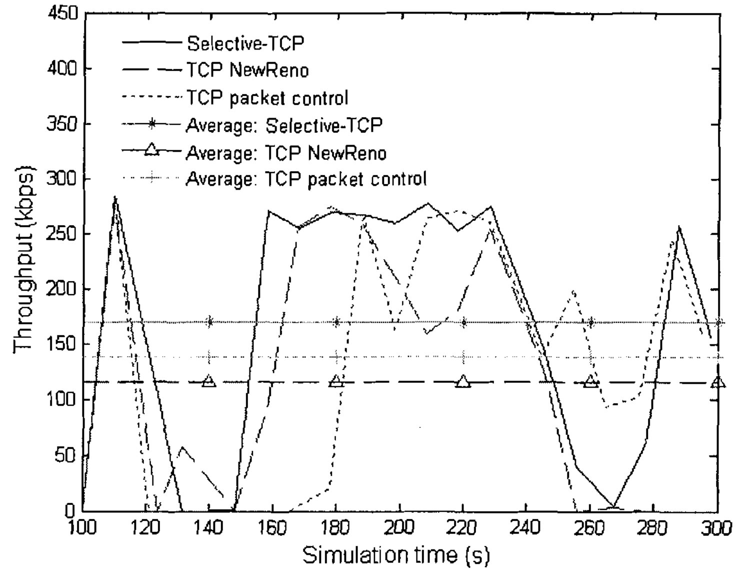
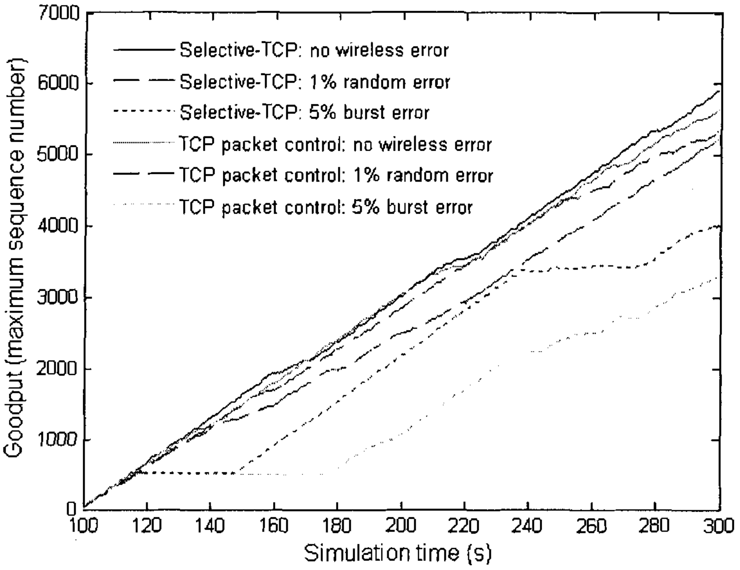


Figure 33: Throughput vs. simulation time.



The effect of varying wireless errors on goodput performance is shown in Fig. 34.

Figure 34: Effects of wireless error on goodput performance.



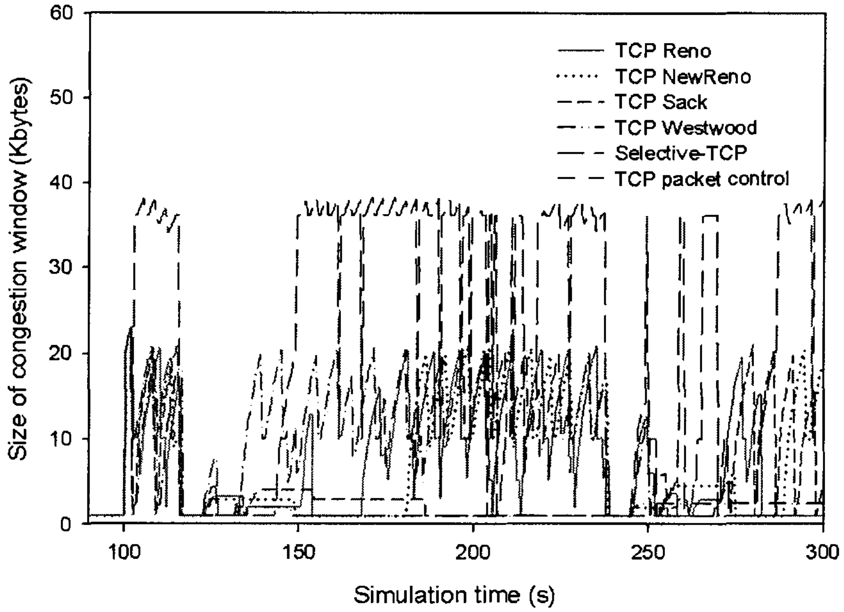
6.2.4 Comparison of Selective-TCP with Other TCP Variants

We compare Selective-TCP with TCP Reno [1], TCP SACK [5], and TCP Westwood [18] along with TCP NewReno [2] and TCP packet control algorithm [8].

6.2.4.1 Comparison in the Presence of a Congested Link

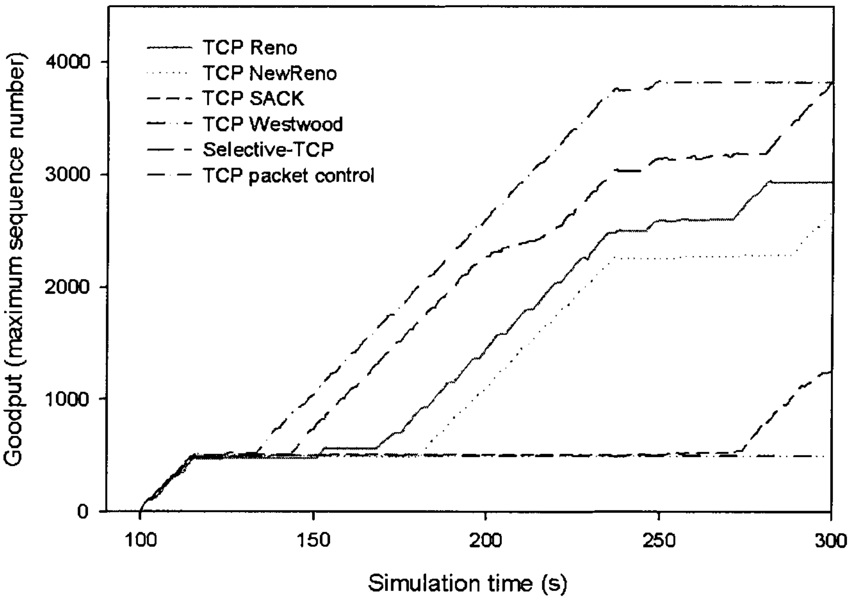
In the presence of congestion in the network, Selective-TCP achieves larger congestion window compared to other TCP variants, as shown in Fig. 35. TCP SACK and TCP Westwood show smaller bandwidth utilization. This indicates that in the presence of congested link in the network, SACK and Westwood fail to efficiently use the available bandwidth.

Figure 35: Size of congestion window vs. simulation time: congestion window size is the largest for Selective-TCP, compared to other TCP variants.



TCP packet control algorithm achieves the highest goodput followed by Selective-TCP, TCP Reno, and NewReno, as shown in Fig 36. However, performance of TCP Westwood and TCP SACK deteriorates significantly, which is in agreement with the poor bandwidth utilization of TCP SACK and Westwood, shown in Fig. 35. Among the considered TCP variants, TCP packet control is the only link layer based algorithm. All other algorithms are end-to-end approaches. This explains the reason that TCP packet control algorithm achieves highest goodput in a mixed wired/wireless network with 5% burst error in the wireless links.

Figure 36: Goodput vs. simulation time: network performance deteriorates for TCP SACK and TCP Westwood.



6.2.4.2 Comparison in the Absence of a Congested Link

In the absence of congested link, the congestion window sizes for all six TCP variants are shown in Fig. 37. Similar to the case of congested link, Selective-TCP achieves the highest congestion window when compared to other TCP variants. Selective-TCP measures the available bandwidth at the time of packet loss and sets congestion window size accordingly. This is the reason for the high congestion window size for Selective-TCP. TCP Westwood uses bandwidth estimation to set congestion window size. It should have also achieved larger congestion window size compared to TCP Reno, NewReno, SACK, and TCP packet control algorithm because none employs bandwidth measurement/estimation. However, in our simulations, somewhat unexpectedly, congestion window for TCP Westwood is similar to the other TCP variants.

Figure 37: Congestion window size for 300 s of simulation time.

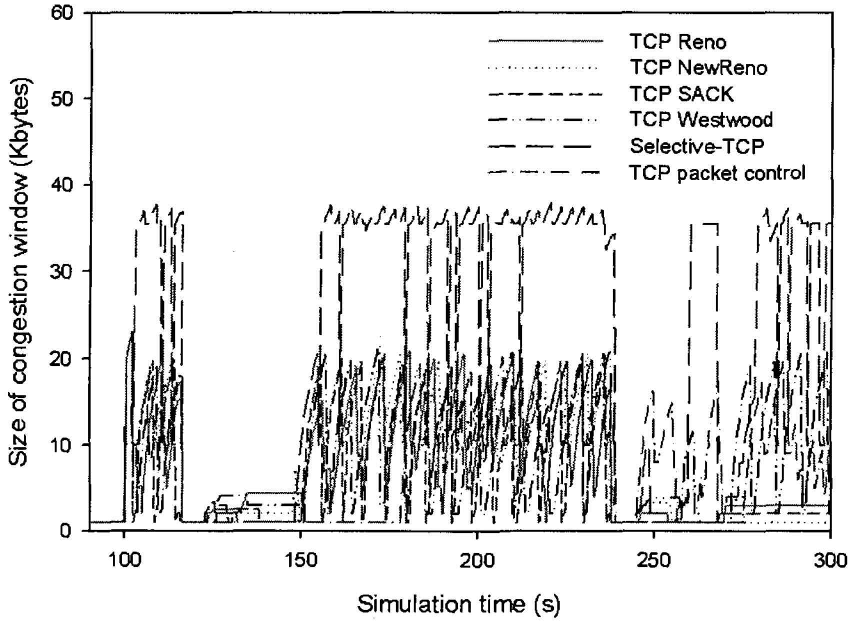
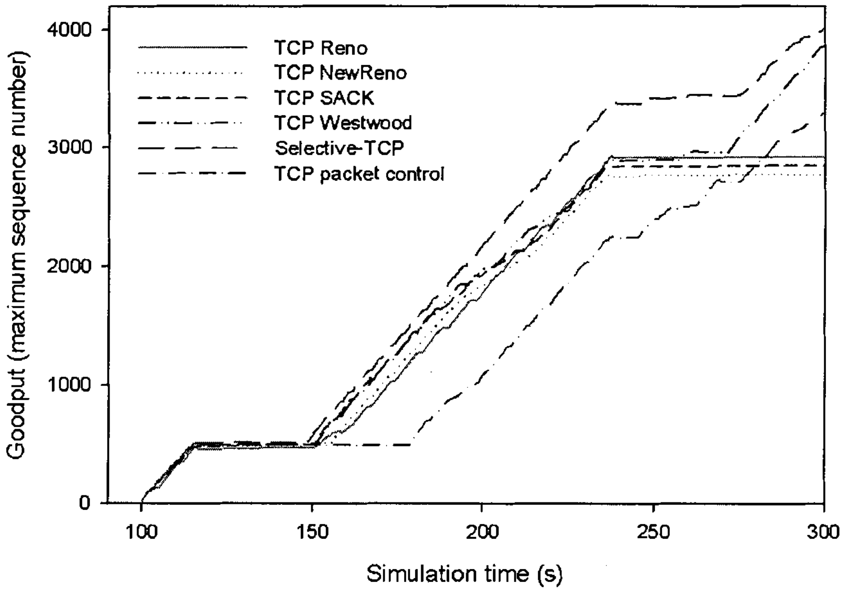


Fig. 38 shows the goodput of TCP Reno, NewReno, SACK, Westwood, Selective-TCP, and TCP packet control algorithm for 300 s of simulation time. Selective-TCP performs best, followed by TCP Westwood. The reason is that both algorithms use bandwidth measurement/estimation. TCP Reno, NewReno, and SACK perform comparably. TCP packet control algorithm, unlike in the case of congestion, achieves lower goodput. This indicates that TCP packet control algorithm performs better in the presence of congestion. Selective-TCP, on the other hand, performs well in cases of both congestion and non-congestion.

Figure 38: Goodput vs. simulation time.



7 CONCLUSIONS AND FUTURE WORK

In this report, we proposed a new end-to-end protocol called Selective-TCP to improve TCP performance in mixed wired-wireless networks, where wireless link is the network bottleneck. Selective-TCP distinguishes between congestion and wireless errors and accordingly takes corrective measures for those errors. In case of wireless errors, the receiver sends selective negative acknowledgement to the sender without sending duplicate acknowledgements and, thus, prevents congestion control that is otherwise preformed by TCP. On detection of congestion error in the network, receiver informs the sender of the measured bandwidth at receiver. The sender then sets the congestion window size accordingly, thus stopping the AIMD algorithm to set congestion window lower than necessary. Both measures taken by the Selective-TCP improve the bandwidth utilization and increase goodput up to 45% compared to TCP NewReno.

We have evaluated Selective-TCP's performance in presence of burst errors in the wireless link. To make the results relevant to deployed wired/wireless networks, we have used realistic parameters for the error model. TCP connections are simulated using the ns-2 network simulator. Selective-TCP is an extension of NewReno sender and receiver. It requires no modification on intermediate routers, making it purely an end-to-end approach.

Along with performance evaluation of Selective-TCP, we have compared its performance with the TCP packet control algorithm. TCP packet control algorithm is a link-layer based approach to improve TCP performance over wireless link. We have also compared Selective-TCP's performance to TCP Reno, NewReno, SACK, and TCP Westwood.

As future work, the Selective-TCP algorithm may be improved by employing feedback from intermediate routers, while maintaining the end-to-end semantics of the connection. The performance of Selective-TCP could be evaluated in the presence of frequent hand-offs. Furthermore, performance of Selective-TCP with protocols other than TCP NewReno could be investigated. In this report, we have studied performance of the proposed algorithm in mixed wired/wireless networks only. Evaluation of its performance in mixed satellite and wired networks would be an interesting subject to investigate.

REFERENCE LIST

[1] S. Floyd and T. Henderson, "The NewReno modification to TCP's fast recovery algorithm," *IETF RFC 2582*, Apr. 1999.

[2] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control" *IETF RFC 2581*, Apr. 1999.

[3] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The protocols*. New York: Addison-Wesley, 1994.

[4] Consultative Committee for Space Data Systems, *Space Communications Protocol Specification—Transport Protocol (SCPS-TP)*, Blue Book, issue 1, May 1999.

[5] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgement options," *IETF RFC 2018*, Apr. 1996.

[6] R. Fox, "TCP big window and NAK options," *IETF RFC1106*, Jun. 1989.

[7] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving TCP/IP performance over wireless networks," in *Proc. ACM Int. Conf. on Mobile Computing and Networking*, Berkeley, CA, Nov. 1995, pp. 2–11.

[8] W. G. Zeng and Lj. Trajkovic, "TCP packet control for wireless networks," in *Proc. IEEE Int. Conf. on Wireless and Mobile Computing, Networking and Communications (WiMob 2005)*, Montreal, Canada, Aug. 2005, pp. 196–203.

[9] K. Brown and S. Singh, "M-TCP: TCP for mobile cellular networks," *Computer Communication Review*, vol. 27, no. 5, pp. 19–43, Oct. 1997.

[10] A. Bakre and B. R. Badrinath, "I-TCP: indirect TCP for mobile hosts," in *Proc. 15th Int. Conf. on Distributed Computing Systems*, Vancouver, Canada, May 1995, pp. 136–143.

[11] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *Computer Communication Review*, vol. 26, no. 4, pp. 256–269, Aug. 1996.

[12] C. P. Fu and S. C. Liew, "TCP Veno: TCP enhancement for transmission over wireless access networks," *IEEE J. on Select. Areas Commun.*, vol. 21, no. 2, pp. 216–228, Feb. 2003.

[13] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: new techniques for congestion detection and avoidance," in *Proc. SIGCOMM*, London, U.K., Oct. 1994, pp. 24–35.

[14] N. K. G. Samaraweera, "Non-congestion packet loss detection for TCP error recovery using wireless links," *IEE Proc. Communications*, vol. 146, no. 4, pp. 222–230, Aug. 1999.

[15] D. Barman and I. Matta, "Effectiveness of loss labeling in improving TCP performance in wired/wireless networks," in *Proc. 10th IEEE Int. Conf. on Network Protocols*, Boston, MA, Nov. 2002, pp. 2–11.

[16] C. Parsa and J. J. Garcia-Luna-Aceves, "Differentiating congestion vs. random loss: a method for improving TCP performance over wireless links," in *Proc. IEEE Conf. on Wireless Communications and Networking*, Chicago, IL, Sept. 2000, vol. 1, pp. 90–93.

[17] T. Kim, S. Lu, and V. Bharghavan, "Improving congestion control performance through loss differentiation," in *Proc. Eighth International Conference on Computer Communications and Networks*, Boston, MA, Oct. 1999, pp. 412–418.

[18] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP Westwood: end-to-end congestion control for wired/wireless networks," *Wireless Networks*, vol. 8, no. 5, pp. 467–479, Sept. 2002.

[19] V. Tsaoussidis and C. Zhang, "TCP-Real: receiver-oriented congestion control," *Computer Networks*, vol. 40, no. 4, pp. 477–497, Nov. 2002.

[20] F. Sun, V. O. K. Li, and S. C. Liew, "Design of SNACK mechanism for wireless TCP with new snoop," in *Proc. IEEE Wireless Communications and Networking Conference*, Atlanta, GA, Mar. 2004, vol. 2, pp. 1051–1056.

[21] K. Xu, Y. Tian, and N. Ansari, "TCP-Jersey for wireless IP communications," *IEEE J. Select. Areas Commun.*, vol. 22, no. 4, pp. 747–756, May 2004.

[22] S. Biaz and N. H. Vaidya, "Discriminating congestion losses from wireless losses using inter-arrival times at the receiver," in *Proc. IEEE Symposium on Application-Specific Systems and Software Engineering and Technology*, Richardson, TX, Mar. 1999, pp. 10–17.

[23] S. Cen, P. C. Cosman, and G. M. Voelker, "End-to-end differentiation of congestion and wireless losses," *IEEE/ACM Trans. Networking*, vol. 11, no. 5, pp. 703–717, Oct. 2003.

[24] R. C. Durst, G. J. Miller, and E. J. Travis, "TCP extensions for space communications," in *Proc. MOBICOM*, Rye, NY, Nov. 1996, pp. 15–26.

[25] ns-2 [Online]. Available: http://www.isi.edu/nsnam/ns.

[26] J. Chung and M. Claypool, NS by example, Technical report, CS Department, Worcester Polytechnic Institute, Sep. 1999. Online at: http://perform.wpi.edu/NS.

[27] D. Anantharaman, "Performance analysis of snack in satellite networks through simulation," M.S. Thesis, Lamar University, Lamar, TX, 2004.

[28] A. Gurtov and S. Floyd, “Modeling wireless links for transport protocols,” *Computer Communication Review*, vol. 34, no. 2, pp. 85–96, Apr. 2004.

[29] J. McDougall and S. Miller, “Sensitivity of wireless network simulations to a two-state Markov model channel approximation,” in *Proc. GLOBECOM*, San Francisco, CA, Dec. 2003, pp. 697–701.

[30] A. Konrad, B. Y. Zhao, A. D. Joseph, and R. Ludwig, “A Markov-based channel model algorithm for wireless networks,” *Wireless Networks*, vol. 9, no. 3, pp. 189–199, May 2003.