

TCP CONGESTION CONTROL MECHANISM FOR WIRED/WIRELESS NETWORK

A DISSERTATION

*Submitted in partial fulfillment of the
requirements for the award of the degree*

of

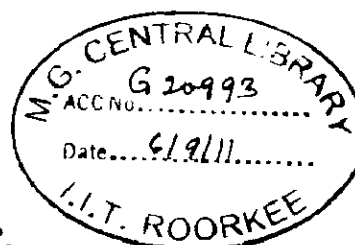
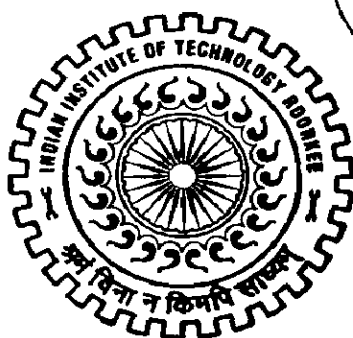
MASTER OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

SHASHI RANJAN



DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE - 247 667 (INDIA)

JUNE, 2011

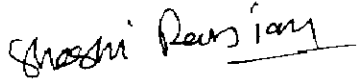
CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in the dissertation entitled “**TCP congestion control mechanism for wired/wireless network**” towards the partial fulfillment of the requirement for the award of the degree of **Master of Technology** in **Information Technology** submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee, Uttarakhand (India) is an authentic record of my own work carried out during the period from July 2010 to June 2011, under the guidance of **Dr. R. C. Joshi, Professor**, Department of Electronics and Computer Engineering, IIT Roorkee.

The matter presented in this dissertation has not been submitted by me for the award of any other degree of this or any other Institute.

Date: 30/June

Place: Roorkee



(SHASHI RANJAN)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 30/6/11

Place: Roorkee


(Dr. R. C. JOSHI)

Professor

Department of Electronics and Computer Engineering

IIT Roorkee.

ACKNOWLEDGEMENTS

First and foremost, I would like to extend my heartfelt gratitude to my guide and mentor **Dr. R. C. Joshi**, Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, for his invaluable advices, guidance, and encouragement for sharing his broad knowledge. His wisdom, knowledge and commitment to the highest standards inspired and motivated me. He has been very generous in providing the necessary resources to carry out my research. I have been greatly influenced by his tenet of bridging the gap between theory and practice, and have benefited from his detailed comments on every aspect of this thesis. He is an inspiring teacher, a great advisor, and most importantly a nice person.

I am thankful for the useful comments and suggestions from faculty members of our institute. I am greatly indebted to all my friends, who have graciously applied themselves to the task of helping me with ample moral supports and valuable suggestions.

On a personal note, I owe everything to the Almighty and my parents. The support which I enjoyed from my mother, brother, sisters and other family members provided me the mental support I needed.

Shashi Ranjan

SHASHI RANJAN

ABSTRACT

Wireless networks are going to be pervasive, and hence it is almost impossible to keep ourselves away from this technology. But still wire network has its own significance, which can not be ignored due to its reliability. The major difference between both technologies is that wireless has more error prone attitude to packet transmission with respect to wired network. Among the available transport level protocol, TCP is the most prominent due to its characteristic and long term use in application. Rather than concentrating on new transport layer protocol, I have improved the present TCP congestion control mechanism so that it can be used universally on wired and wireless technology. One of the reasons behind the degradation of TCP congestion control mechanism performance is considering packet loss due to corruption in channel as network congestion.

In this dissertation I have cast the idea of two windows for measuring the real congestion window and status of packet loss due to corruption in channel. Proposed algorithm changes real congestion window based on degree of packet loss due corruption in channel to increase in throughput of network. It has been tested on the network simulator NS3. results shows that congestion widow get affected only in case of packet loss due to high load rather than packet loss due to corruption in channel. It enhances the network throughput ratio significantly of such networks, which is susceptible to error, with respect to present TCP variant such as Tahoe, Reno and NewReno.

Table of Contents

	Page No.
Candidate's Declaration & Certificate	i
Acknowledgements	ii
Abstract	iii
Table of Contents	iv
List of Figures	vii
List of Tables	viii
CANDIDATE'S DECLARATION.....	i
CERTIFICATE.....	i
1 Introduction and Problem Statement.....	1
1.1 Introduction.....	1
1.2 Motivation.....	2
1.3 Problem Identification.....	2
1.3.1 Problem Statement.....	3
1.4 Thesis Organization.....	4
2 Background and Literature Survey.....	7
2.1 Transmission Control Protocol.....	7
2.2 Layer of congestion control mechanism.....	8
2.3 Classification of actual flow control.....	9
2.4 Congestion control in TCP.....	10
2.4.1 Window-based control.....	11
2.4.2 Acknowledgements and loss detection.....	11
2.5 TCP congestion control state.....	12
2.5.1 Slow start.....	13
2.5.2 Congestion avoidance.....	14
2.5.3 Exponential back off.....	14
2.5.4 Fast recovery.....	15
2.6 Fundamental requirements of a congestion control scheme.....	15

2.6.1 Efficiency.....	16
2.6.2 Heterogeneity.....	16
2.6.3 Stability.....	16
2.6.4 Scalability.....	17
2.6.5 Simplicity.....	17
2.7 Variants of the TCP protocol.....	17
2.7.1 TCP-Tahoe.....	18
2.7.2 TCP-Reno.....	20
2.7.3 TCP-NewReno.....	22
2.7.4 TCP-SACK.....	23
2.7.5 TCP-Vegas.....	23
2.8 Shortcomings and Research Gaps.....	25
3 Proposed TCP Congestion Control mechanism.....	27
3.1 Packet loss Rate.....	27
3.2 Modified TCP.....	28
3.3 Algorithm.....	29
3.3.1 Initial Window.....	29
3.3.2 Slow Start Algorithm.....	29
3.3.3 Congestion Avoidance Algorithm.....	30
3.3.4 Fast Retransmission and Fast Recovery Algorithm.....	31
4 Experimental Results and Discussions.....	33
4.1 Performance Metrics.....	33
4.2 Implementation in NS3.....	33
4.3 Simulation Environment.....	35
4.3 Results and Discussions.....	36
5 Conclusions.....	39
5.1 Conclusions.....	39
5.2 Scope for Future Work.....	40
References.....	41
List of Publication.....	45
Appendix.....	47

LIST OF FIGURES

<u>Figure No.</u>	<u>Page No.</u>
Figure 1.1 Packet passing through bottleneck network.....	3
Figure 2.1 The layers of the TCP/IP networking model.....	8
Figure 2.2 TCP state diagram	13
Figure 2.3 TCP Tahoe congestion window evolution	19
Figure 2.4 TCP-Reno congestion window evolutions.....	21
Figure 2.5 TCP-Vegas congestion window evolution.....	24
Figure 4.1 class shashi_universalTCP.....	33
Figure 4.2 NS3 simulation hierarchy.....	36
Figure 4.3 Congestion window plot at BER 0.0001.....	37
Figure 4.4 Congestion window plot at BER 0.00001.....	37
Figure A.1 organization of NS-3.....	43
Figure A.2 packet sending process.....	44

LIST OF TABLES

Table No.	Page No.
Table 2.1 Classification of actual flow control implementations [1]	10
Table 4.1 The protocol capability with different BER	36

1 Introduction and Problem Statement

1.1 Introduction

One of the core protocols of the Internet Protocol Suite is the Transmission Control Protocol (TCP). It is one of the two original components of the suite, complementing the Internet Protocol (IP), that's why the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered delivery of a stream of bytes from a program on one computer to another program on another computer. TCP suppose that the lost packet due to network's congestion [7], this is reasonable for wired network. But it is not suitable for the error-prone especially wireless network which has many characteristics like high bit error rate and low bandwidth etc. packet loss may happen due to weather conditions, obstacles, multipath interferences, mobility of wireless end-devices, signal attenuation and fading etc. When TCP [4] operates in wireless networks, it suffers from severe performance degradation because of the different characteristics of wireless networks and wired networks. The performance degradation is mainly caused by TCP's basic assumption that any packet loss is an indication of congestion. Although this assumption works very well in wired networks where most packets are lost due to congestion only, the assumption is not appropriate for wireless networks where most packet losses are caused by wireless transmission errors [3]. The appropriate behavior of TCP for the packet loss due to wireless transmission errors is just to retransmit the lost packet without reducing its sending rate. Unfortunately, TCP considers every packet loss as congestion signals, and unnecessarily decreases its sending rate by halving its congestion window size. To avoid such performance degradation, it is important for TCP to differentiate between wireless losses and congestion losses. So, many improved TCP congestion control mechanisms have been presented [10]. The essential of this improved scheme is to distinguish the lost packet due to congestion to those due to corruption. The improved schemes use the congestion control mechanism while the network is congestion and hold the packet sending rate while wireless link corruption.

1.2 Motivation

Over the last decade, both the Internet and mobile telephony have become parts of daily life, changing the ways we communicate and search for information. These two distinct tools are now slowly merging, both at the surface, and in the underlying communication infrastructure.

Wireless mobile Internet means that mobile gadgets are first class citizens of the Internet. Any communication task, e.g., email, web browsing, Internet radio, or peer-to-peer file sharing, that is possible with a stationary computer connected to the Internet, should be equally possible with a suitable mobile gadget.

Enabling a wireless mobile Internet is a huge task. One prerequisite is that TCP/IP, the two most important protocols on the Internet, must work satisfactorily across a heterogeneous network consisting of an assortment of stationary and mobile devices, connected by different types of wired and wireless links.

The focus of this dissertation is on the TCP protocol, the problems encountered when using TCP over error-prone network perceive packet loss due to corruption as overloading mistakenly. Due to this phenomenon unnecessarily tradition TCP congestion control mechanism reduces congestion window. So our intention has been to improve throughput of error-prone network by ignoring the packet loss due to corruption in channel. However, the key issues like Efficiency, Ability to deal with heterogeneity, Stability, Scalability and Simplicity need to be addressed while designing the TCP congestion control [23].

1.3 Problem Identification

Unfortunately present TCP mechanism could not give expected result due to packet loss due to corruption in channel. We know internet technology is changing fast and wireless network is becoming pervasive all around. And packet lost in wireless network may not imply congestion in network because it may happen due to weather conditions, obstacles, and multipath interferences, mobility of wireless end-devices, signal attenuation and fading. So our assumption upon which network congestion window has been changing in TCP congestion control mechanism is not appropriate for wireless.

The overall goal of congestion control is to optimize the performance in a communication network. This optimization means, roughly, that sending rates at the data sources should be as high as possible, without overloading the network. The primary measure of network overload is packet losses; when the arrival rate at a link exceeds capacity, the corresponding queue starts to build up, and when the queue is full, packets must be discarded. The bottleneck links in the network should be fully utilized. The requirement of a small loss rate implies that the average arrival rate at each bottleneck link should either match the link capacity exactly, or be very slightly larger.

1.3.1 Problem Statement

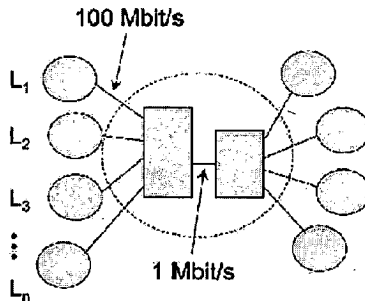


Figure 1.1 Packet is passing through bottleneck network and tends towards congestion in absence of control [2].

In the "Fig. 1.1" $L_1, L_2, L_3, \dots, L_n$ sources are sending packet through available resources, which will leads towards congestion for deteriorating the situation in absence of any control. Here sources are sending at 100Mbit/s and network capacity is 1Mbit/s.

The problem for which a control system needs to identify mechanisms which permit efficient dynamic sharing of the pool of resources (channels, buffers, and switching processors) in a packet Network can be defined mathematically [2] as follow, from "Fig. 1.1"

$$\sum_{i=1} L_i \leq C. \quad (1.1)$$

L_i = sending capacity of i^{th} node in.

C = network capacity.

There are some issues that need to be considered for maintaining the (1), considering this from “Fig. 1.1”

- Scalability: whatever be the solutions that need to be scalable for large network i.e for large N , it should take care of large number of source.
- Dynamicity: here number of node and each node capacity may change, so accordingly we have to take care of this.
i.e. N - varies , L_i -varies : more specifically N and L_i are function of time.
- Here resources are far from control system.
i.e. we need to control the flow at network node, which can be manipulated at source node only.

The main objective of the present research work can be described by the statement of the problem expressed as follows:

“To design a TCP congestion control mechanism so that it can be used universally on wired/wireless network and increase the ratio of throughput, which deviated due to corruption in network in the presence of existing TCP congestion control”.

To achieve the desired objective of increasing the throughput of error-prone network following smaller objectives can be set:

- To identify dropped packet due to corruption in channel.
- To measure the degree of corruption in packet.
- Treat congestion window as normal in case of few degree of corruption.
- To maintain two windows, one as real congestion window and other as guideline in case of corruption in channel.

1.4 Thesis Organization

This dissertation report comprises of five chapters including this chapter that introduces the topic and states the problem. The rest of the report is organized as follows:

Chapter 2 details the fundamentals and provides a literature review of the various TCP congestion control algorithm and flavor used in networks. Research gaps and shortcomings are identified and described.

Chapter 3 describes the proposed TCP congestion control algorithm which increase the throughput of error-prone network based on degree of corruption.

Chapter 4 discusses the implementation details and provides the experimental results of the proposed algorithms. In this chapter performance of the proposed algorithm is also compared with the existing one.

Chapter 5 concludes the dissertation work and gives suggestions for future work.

2 Background and Literature Survey

2.1 Transmission Control Protocol

Above the IP layer, we have the transport layer, where the Transmission Control Protocol (TCP) is the protocol of primary interest. TCP is responsible for dividing a data stream into packets, ensure reliable delivery even when the IP layer loses, reorders, or duplicates packets, and at the same time it senses the state of the network to avoid overloading it.

The dominating transport layer protocol is TCP. It is used for all kinds of data streams. A TCP connection is a bidirectional, flow controlled, and reliable stream of data between two endpoints, identified by IP address and port number. Our primary interest is in TCP connections for transfer of smaller or larger files. For this TCP usage, it is desirable to get as much data as possible through the network, while at the same time we must avoid overloading the network, and share available bandwidth in a fair way with other users. TCP uses a sliding window flow control. The window limits the amount of data that can be sent without waiting for acknowledgement (ACK) from the receiver.

When the window is constant, this results in the so called “ACK clock”; the timing of each sent packet is determined by the reception of the ACK for an earlier packet. One can think about the sliding window and the ACK clock as a peculiar inner control loop which determines the sending rate; when the roundtrip time fluctuates, the sliding window gives an average sending rate of one full window per average roundtrip time. The window size is adjusted depending on received ACKs, and it is the details of this outer loop that differ between TCP variants.

The objective of the TCP window control is to get a high throughput, close to the connection’s fair share of the available bandwidth, and at the same time avoid overloading the network. The fair share can vary due to varying amounts of competing

cross traffic, and also due to network changes such as to routing updates or wireless links with time-varying capacity.

2.2 Layer of congestion control mechanism

Congestion control is one piece of the Internet machinery. But how does it fit in the larger picture, and how does the Internet really work. Layered design of communication systems is a modularization technique, where each layer at a particular node needs to know how to communicate with the layers directly above and below at the local node, but only to the same layer at remote.

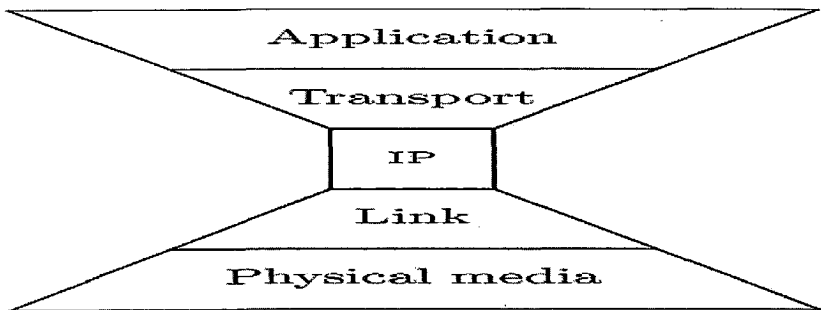


Figure 2.1 The layers of the TCP/IP networking model

One of oldest engineering concept, modularity is as old as engineering itself. Modularity is such concept which helps us to design a system by identifying its component independently and then integrate it. Just following this ideology to resolve the data network issue, there has been designed a hierarchal modularity form, which contains black box at various layer are in fact distributed black boxes. The bottom layer of the hierarchy consists of the physical communication link.

Flow control can be exercised at various levels in a packet network. The following levels are identified and discussed [1].

- 1) Hop Level: This level of flow control attempts to maintain a smooth flow of traffic between two neighboring nodes in a computer network, avoiding local

buffer congestion, throughput degradation and deadlocks. It can play the role of arbitrator between various classes of traffic competing for a common buffer pool in each node e.g. channel queue limit, buffer class, virtual circuit hop level.

- 2) **Entry-to-Exit Level:** This level of flow control is generally implemented as a protocol between the source and destination switch, and has the purpose of preventing buffer congestion at the exit switch e.g. Arpanet's RFNM (ready for next message), SNA Virtual Route Pacing Scheme, GMD Individual Flow Control
- 3) **Network Access Level:** The objective of this level is to throttle external inputs based on measurements of internal (as opposed to destination) network congestion. Congestion measures may be
 - local (buffer occupancy in the entry mode such as Input Buffer Limit Scheme)
 - global (total number of buffers available in the entire network such as isarithm scheme),
 - selective (congestion of the path leading to a given destination such as choke packet scheme)
- 4) **Transport Level:** This is the level of flow control associated with the transport protocol, i.e., the protocol which provides for the reliable delivery of packets on the "virtual" connection between two remote processes. Its main purpose is to prevent congestion of user buffers at the process level

2.3 Classification of actual flow control

In real life, however, some control structures defy the simple, hierarchical representation here proposed, and seem to combine two or more levels into hybrid solutions (Table 2.1)

	ARPA			TRANS PAC	SNA			GMDNET	
	cql	rftm	NCP	X.25	SDL C	VR pacing	Session pacing	i.c	SBP
HOP LEVEL	Y			Y	Y	Y		Y	Y
NETW. ACC.				Y			Y	Y	
ENTRY- EXIT		Y		Y		Y	Y		Y
TRANSP.			Y	NOT DEF			Y	N	N

Table 2.1 Classification of actual flow control implementations [1]

2.4 Congestion control in TCP

Computer networks form an essential substrate for a variety of distributed applications, but they are expensive to build and operate. This makes it important to optimize their performance so that users can derive the most benefit at the least cost. Though most networks perform well when lightly used, problems can appear when the network load increases. Loosely speaking, congestion refers to a loss of network performance when a network is heavily loaded. Since congestive phenomena can cause data loss, large delays in data transmission, and a large variance in these delays, controlling or avoiding congestion is a critical problem in network management and design. This dissertation presents some approaches for congestion control in wired/wireless computer networks universally.

Historically, the first wide-area networks (WANs) were circuit-switched telephone networks. Since these networks carry traffic of a single type, and the traffic behavior is well known, it is possible to avoid congestion simply by reserving enough resources at the start of each call. By limiting the total number of users, each admitted call can be

guaranteed to have enough resources to achieve its performance target, and so there is no congestion. However, resources can be severely underutilized, since the resources blocked by a call, even if idle, are not available to other calls.

The objective of congestion control is to keep the load of the network close to the available capacity. The TCP protocol was developed in the late 1970s, resulting in the Internet Standard RFC 793 [4]. The principles for TCP congestion control were developed a few years later, in response to experience of “congestion collapses” in the Internet [5].

2.4.1 Window-based control

The most important concept in TCP congestion control is that of the congestion window. The window is the amount of data that has been sent, but for which no acknowledgement has yet been received. A constant congestion window means that one new packet is transmitted for each ACK that is received. The sending rate is controlled indirectly by adjusting the congestion size. The standard way of doing this is documented in RFC 2581 [7], usually referred to as TCP Reno. It is described in this section. Two other common variants are TCP NewReno [14] and TCP with selective acknowledgements (sack) [8, 9]. Before explaining the control mechanisms, we have to look into how TCP detects packet losses.

2.4.2 Acknowledgements and loss detection

At the receiving end, acknowledgement packets are sent in response to received data packets. TCP uses cumulative acknowledgements: Each acknowledgement includes a sequence number that says that all packets up to that one have been received. Equivalently, the acknowledgement identifies the next packet that the receiver expects to see. When packets are received out of order, each received packet results in an acknowledgement, but they will identify the largest sequence number such that all packets up to that number have been received. E.g., if packets 1, 2, 4, and 5 are received, four acknowledgements are generated. The first says “I got packet #1, I expect packet #2 next”, while the next three acknowledgements all say “I got packet #2, I expect packet #3

next". The last two acknowledgements are duplicate ACKs, since they are identical to some earlier ACK. On the sending side, there are several possible reasons why duplicate ACKs are received: Packets delivered by the network out of order, packets dropped by the network, and ACK packets duplicated by the network. Packet losses are detected by the sender in two ways:

- Timeout. If a packet is transmitted and no ACK for that packet is received within the retransmission timeout interval (RTO), the packet is considered lost.
- Fast retransmit. If three duplicate ACK are received, the "next expected packet" from these ACKs is considered lost. Note that this can not happen if the congestion window is smaller than four packets.

Packets that are lost, as detected by either of these mechanisms, are retransmitted. The value for RTO is not constant, but based on measured average and variation of the RTT. It is also modified by the exponential back off mechanism.

2.5 TCP congestion control state

There are four distinctive states in the TCP congestion control, illustrated in "Figure 2.2", and two state variables related to congestion control: The congestion window *cwnd* and the slow start threshold *ssthresh*. Typical initial values when TCP leaves the idle state and enters the slow start state are a *cwnd* of 2 packets, and a *ssthresh* that is the maximum value allowed by the wire protocol and by the receiving end.

We look at the operation of each of the four states in turn.

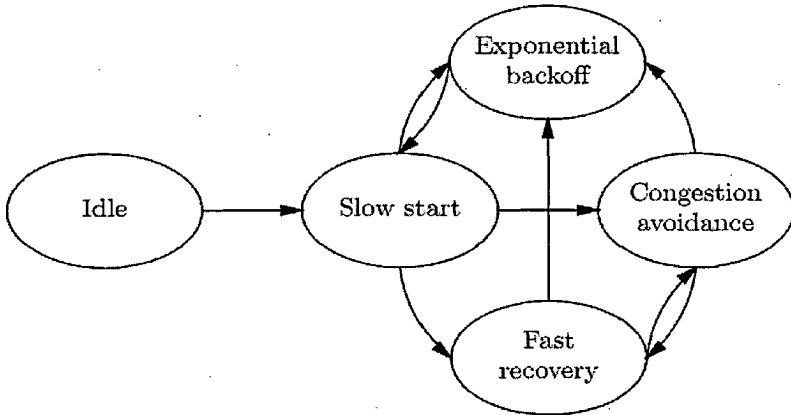


Figure 2.2 TCP state diagram. Transitions back to the idle state are omitted [21].

2.5.1 Slow start

The slow start state is the first state entered when a flow is created, or when a flow is reactivated after being idle. The slow start state can also be entered as the result of a timeout. In this state, $cwnd$ is increased by one packet for each non-duplicate ack. The effect is that for each received ACK, two new packets are transmitted. This implies that the congestion window, and also the sending rate, increases exponentially, doubling once per RTT. It may seem strange to refer to an exponential increase of the sending rate as “Slow start”; the reason is that in the early days, TCP used a large window from the start, and the introduction of the slow start mechanism did slow down connection startup. Slow start continues until either

- $cwnd > ssthresh$, in which case TCP enters the congestion avoidance state, or

- a timeout occurs, in which case TCP enters the exponential back off state, or
- Three duplicate ACKs are received, in which case TCP enters the fast recovery state.

The motivation for the slow start state is that when a new flow enters the network, and there is a bottleneck link along the path, then the old flows sharing that link need some time to react and slow down before there is room for the new flow to send at full speed.

2.5.2 Congestion avoidance

In congestion avoidance mode, $cwnd$ is increased by one packet per RTT (if $cwnd$ reaches the maximum value, it stays there). This corresponds to a linear increase in the sending rate. On timeout, TCP enters the exponential back off state, and on three duplicate ACKs, it enters the fast recovery state. The motivation for this congestion avoidance mechanism is that since TCP does not know the available capacity, it has to probe the network to see at how high a rate data can get through. Aggressive probing would make the system unstable, and a single packet increase seems to work well in practice.

2.5.3 Exponential back off

TCP enters the exponential back off mode after timeout. Several actions are taken when entering this state:

- The lost packet is transmitted.
- The state variables are updated by $ssthresh \leftarrow cwnd/2$, $cwnd \leftarrow 1$ packet.
- The RTO value is doubled.

If the retransmission timer expires again with no ACK for the retransmitted packet, the packet is repeatedly retransmitted, RTO is doubled, and $ssthresh$ is set to 1 packet [24]. The upper bound for the RTO is on the order of one or a few minutes. Exponential back off continues until an acknowledgement for the packet is received, in which case TCP enters the slow start phase, or the TCP stack or application gives up and closes the connection.

The motivation for the exponential back off mechanism is that timeouts, in particular repeated timeouts, are a sign of severe network congestion. In order to avoid congestion collapse, the load on the network must be decreased considerably and repeatedly, until it reaches a level with a reasonably small packet loss probability.

2.5.4 Fast recovery

TCP enters the fast recovery state after it detects three duplicate ACKs. When entering this mode, the first actions of TCP is to retransmit the lost packet, and set $ssthresh \leftarrow cwnd/2$. TCP then continues to send new data at approximately the same rate, one new packet of data for each received duplicate ack. In RFC 2581 [7], this is described using a fairly complex procedure that artificially inflates $cwnd$. If no ACK for the retransmitted packet is received within the rto interval, TCP enters the exponential back off state. Otherwise, when an ACK for the retransmitted packet is finally received, TCP sets $cwnd = ssthresh$, i.e., half the $cwnd$ value at the start of the recovery procedure, and enters the congestion avoidance state. If more than one packet is lost within the same window, fast recovery is limited in that it can recover only one packet per RTT. This is the main problem addressed by TCP NewReno and TCP sack.

The motivation for the fast recovery mechanism is that the reception of duplicate ACKs indicates that the network is able to deliver new data to the receiver. Hence, the network is not severely congested, and we can keep inserting new packets into the network at the same rate as packets are delivered, at least for a while. On the other hand, the loss of a packet also indicates that the network is on the border of congestion. At the end of the fast recovery procedure, $cwnd$ is halved. TCP restarts the probing of the congestion avoidance state at a lower sending rate, at which it did not get any losses. It should also be noted that halving the $cwnd$ also implies that TCP will stay silent for about half an RTT, waiting for ACKs that reduce the number of outstanding packets, until the outstanding packets match the new window size.

2.6 Fundamental requirements of a congestion control scheme

We would like a congestion control scheme to have a number of properties. These are:

- Efficiency
- Ability to deal with heterogeneity
- Stability
- Scalability
- Simplicity

We discuss these in turn

2.6.1 Efficiency

There are two aspects to efficiency. First, how much overhead does the congestion control scheme impose upon the network? This is not necessarily a binding condition: with the rapid increase in communication bandwidth, the extra load may not significantly affect network efficiency.

Second, does the control scheme lead to underutilization of critical resources in the network? Inefficient control schemes may throttle down sources even when there is no danger of congestion, leading to underutilization of resources. We would not like to operate the network suboptimal.

2.6.2 Heterogeneity

As networks increase in scale and coverage, they span a range of hardware and software architectures. A control scheme that assumes a single packet size, a single transport layer protocol, or a single type of service cannot be successful in such an environment. Thus, we want the control scheme to be implementable on a wide range of network architectures.

2.6.3 Stability

Congestion control can be viewed as a classic negative-feedback control problem. One added complexity is that the control signals are delayed. That is, there is a finite delay between the detection of congestion and the receipt of a signal by a source. Further, the system is noisy, since observations of the system's parameters may be corrupted by transients. These complexities may introduce instabilities into the network. Thus, we would like the control scheme to be robust, and if possible, provably stable.

2.6.4 Scalability

It is the nature of a distributed system to grow with time, and we have seen an explosive growth in network sizes in the last decade. The key to success in such an environment is scalability. We would like a congestion control scheme to scale along two orthogonal axes: bandwidth and network size

2.6.5 Simplicity

Simplicity (unlike stupidity) is always an asset. Simple protocols are easier to implement, perhaps in hardware, and can handle increases in bandwidth. Also, simple protocols are more likely to be accepted as international standards. Thus, we would like an ideal congestion control mechanism to be simple to implement.

In general followings are the main functions of congestion control in a packet network [1]

- Prevention of throughput degradation and loss of efficiency due to overload
- Deadlock avoidance
- Fair allocation of resources among competing users
- Speed matching between the network and its attached user

To summarize, there are a number of issues that are affected by the choice of a congestion control scheme. In this thesis, we present the design of a set of congestion control mechanisms that substantially meet the requirements posed here.

2.7 Variants of the TCP protocol

The particular way the sources react to packet loss events is the main aspect that distinguishes the TCP flavors. The most widely used variants of the TCP protocol are known under the names of [20]

- TCP-Tahoe
- TCP-Reno
- TCP-NewReno
- TCP-SACK
- TCP-Vegas

The TCP-Tahoe flavor is the oldest variant with a simple form of ineffective loss recovery algorithm. TCP-Reno has an improved loss recovery algorithm, capable of keeping a relatively high sending rate even in the presence of occasional packet loss. TCP-NewReno improves upon the TCP-Reno loss recovery algorithm to make it more robust in case of multiple simultaneous packet losses. TCP-SACK allows the receiver to send back detailed information about which packets are missing to the sender. The TCP-SACK algorithm hence also improves the loss recovery phase in scenarios with multiple simultaneous losses. Finally, TCPVegas uses a method that observes changes in the queuing delay to discover beginning network congestion and strives to find an ideal sending rate without creating packet loss in the network.

2.7.1 TCP-Tahoe

Part of the "Tahoe" release of the BSD (Berkeley Unix) distribution in 1988, this was the first TCP to implement congestion control. It represented the state of the art in TCP until TCP Reno became widely adopted [25]. During a file transfer, TCP-Tahoe moves between the so called slow-start phase, congestion avoidance phase and timeout loss recovery phase. The evolution of the sending rate, controlled via the congestion window size, is shown in "Fig. 2.3".

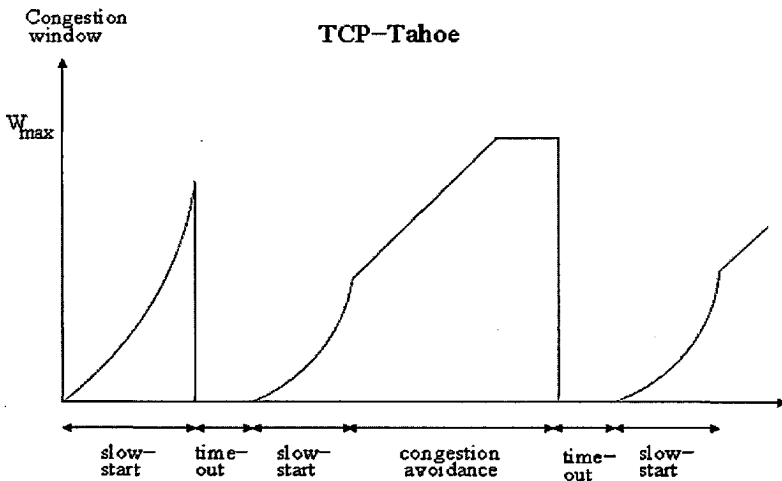


Figure 2.3. TCP Tahoe congestion window evolution

A file transfer starts in the slow-start phase with the congestion window size, W , initialized to one packet. A single packet, with sequence number 1, is sent on the link towards the receiver. Upon receiving this data packet, the receiver acknowledges it by returning a small ACK packet. The ACK states that the receiver is next expecting the packet with sequence number 2. The returning ACK also increases the congestion window size at the sender from one packet to two packets. Two new packets are then sent onto the link, and if no packet loss occurs two ACKs will return from the receiver, each increasing the congestion window by one packet, resulting in a congestion Window size of four packets after these two rounds. During the slow-start phase the congestion window size is thus doubled once every RTT. This increase of the congestion window size continues until the maximal congestion window size has been reached, or until a packet loss occurs. If the maximal congestion window size is reached, consecutive loss free rounds will not increase the congestion window size any further.

The TCP-Tahoe protocol then consecutively moves between the slow-start, congestion avoidance and timeout phases until all packets corresponding to this flow have reached the receiver and been acknowledged to the sender. Also note that a TCP receiver can implement delayed acknowledgements. Then, instead of sending an ACK for every received packet, the receiver only acknowledged every second packet. Since the returning ACKs increase the congestion window for the sender, the usage of delayed ACKs impacts TCP throughput. It is however important that delayed ACKs are not allowed to be used when packets arrive out of order. If a packet that arrives at the receiver has another sequence number than what the receiver is expecting, this immediately generates a returning ACK. The timeout/slow-start response to congestion events taken by TCP-Tahoe is restrictive and conservative. However, such a drastic action as decreasing the congestion window size to one packet at each congestion event can have significant negative impact on the throughput of the protocol.

2.7.2 TCP-Reno

TCP-Reno retains the dynamics of TCP-Tahoe in terms of operation in the congestion avoidance and the slow-start phase, as well as the significance of the congestion window, the slow-start threshold and the maximal congestion window size. The response to packet loss events has however been modified in order to maintain a high sending rate in a mildly congested network. The so called coarse-grained implementation of the TCP timeout in TCP-Tahoe leads to long idle periods, while waiting for the timeout timer to expire. During this waiting period, packet sending is discontinued which results in low throughput. In TCP-Reno, the lengthy loss recovery phase has been improved upon via the introduction of the fast retransmit loss recovery algorithm

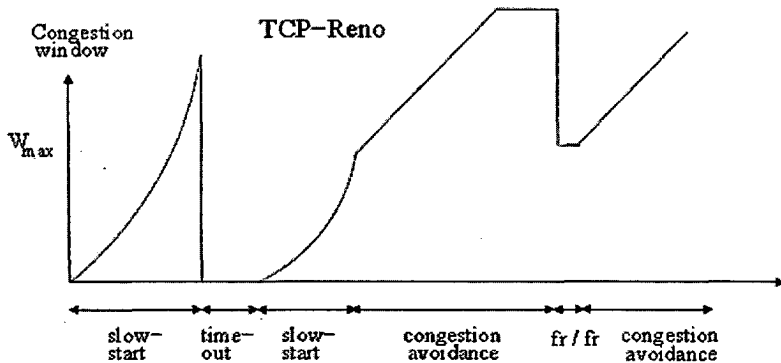


Figure 2.4 TCP-Reno congestion window evolutions

Fast retransmit is a mechanism that sometimes results in a much faster retransmission of a lost packet than what would have been possible if only expire of timeout timers was used to detect packet loss. The returning ACKs provide the sender with the information that the network does not appear to be significantly congested, since a large proportion of the sent packets are successfully received and acknowledged. Setting the congestion window size to one packet, as done by TCP Tahoe, and restarting with slow-start might therefore be too conservative. This has led to the implementation of fast retransmit in combination with the fast recovery algorithm. Fast recovery replaces slow-start after a packet loss event is discovered by triple duplicates. The effect of fast retransmit / fast recovery is in principle that if a packet loss is discovered via triple duplicates, the first lost packet will be quickly resent and the congestion window size halved. If the resulting congestion window size allows it, linear increase during congestion avoidance follows directly. This results in a more aggressive and more effective utilization of the available network capacity, resulting in high throughput for the TCP-Reno sender when only a few packets are lost at each congestion event.

The fast retransmit / fast recovery mechanism implemented in TCP-Reno is able to increase throughput significantly for a TCP source that suffers occasional packet loss.

The design goal of fast retransmit / fast recovery was to make TCP halve its congestion window size once for every congestion event. This goal is achieved if only one packet is lost from the source at each congestion event, where the fast retransmit / fast recovery instance will re-send the first lost packet and quickly resume with congestion avoidance. If multiple packets are lost from a single window, the TCP-Reno fast retransmit / fast recovery algorithm might however lead to multiple consecutive invocations, each invocation halving the congestion window size. In case of multiple packet losses from a single window, the first re-sent packet will lead to the receiver acknowledging that it expects the second lost packet. This ACK for a previously sent and lost packet could be called a partial ACK. In the TCP-Reno implementation of fast retransmit / fast recovery, the arrival of partial ACKs will initiate a new fast retransmit / fast recovery followed by window halving. These consecutive window halving will decrease the congestion window so much that TCP-Reno will ultimately not be able to send any new packets due to the congestion window size restriction on the number of packets it is allowed to have un-acknowledged on the link. Hence, multiple packet losses might finally lead to the sender having to wait for a coarse timeout timer to expire even if the re-sent packets are being correctly received and acknowledged.

2.7.3 TCP-NewReno

TCP-NewReno has refined the fast retransmit / fast recovery algorithm to make it more robust in response to multiple packet losses from one single window by only halving the congestion window once for each congestion event [26]. This is achieved by allowing the partial ACKs arriving during the fast retransmit phase to increase the congestion window, hence keeping the self-clocking in operation even during the loss recovery phase. TCP-NewReno then allows packets to be re-sent into the network as long as previously resent packets are being ACKed. For TCP-NewReno, a fast retransmit/fast recovery phase is not considered finished, and the congestion window size effectively halved, until all packets lost at the congestion event have been successfully resent. Hence, if the re-sent packets are successfully transferred, TCP-NewReno is able to recover from multiple packet losses by re-sending one lost packet per RTT. Since the ACKs from the receiver only specifies

the sequence number of the next expected packet, TCP-NewReno is however not capable of resending more than one lost packet per round.

2.7.4 TCP-SACK

The main contribution of TCP-SACK is an improved loss recovery mechanism, effective in the case of multiple packet losses from a single window. The improvement is achieved by allowing the sender and the receiver to extend more information via the ACK packets than simply the packet sequence number of the next expected packet. If the sender and the receiver both implement the TCP-SACK algorithm and the first lost packet is detected via triple duplicates, the returning ACKs will include information on which packets are missing and which packets are stored in the receiver's packet buffer. This information allows the TCP sender to re-send the correct lost packets, one or more each round in response to ACKs for previously re-sent packets, until all lost packets have been re-sent. TCP-SACK has the same dynamics as TCP-Reno during loss free periods, and also in the case when only one packet is lost. After the loss recovery phase, the congestion window is halved and congestion avoidance takes place. The TCP-SACK implementation is hence an alternative to TCP-NewReno, capable of coping with multiple lost packets from a single window. Whereas both the TCP-NewReno and the TCP-SACK variants are good at avoiding timeouts when multiple packets are lost from a window, TCP-SACK requires fewer rounds to resend the lost packets. Since the sender knows exactly which packets are lost, if the congestion window allows it, more than one lost packet can be re-sent each round. Note however that both the sender and the receiver need to implement the TCP-SACK protocol for this improved loss recovery algorithm to work.

2.7.5 TCP-Vegas

The TCP-Vegas protocol uses a rather different method, compared to the other TCP variants, to infer about and react to network congestion. TCP-Tahoe, TCP-Reno, TCP-NewReno and TCP-SACK detect congestion by stressing the network and increasing the

sending rate until the network becomes congested and packets are lost. On the contrary, the TCP-Vegas algorithm strives to *avoid* congestion and packet loss by adjusting the congestion window size in a pro-active way. The loss recovery phase used by the other TCP flavors have been complemented with a modified slow-start phase and a modified congestion avoidance phase that monitors the changing delay in the network. An increasing delay is interpreted as a beginning congestion, and an RTT close to the (estimated) propagation delay is interpreted as an under-utilization of the network. These modified congestion avoidance and slow-start phases lead to a different form of congestion window size evolution for TCP-Vegas, as shown in “Fig. 2.4”.

During both the congestion avoidance phase and the slow-start phase, TCP Vegas estimates the number, N_b , of packets that this source has back-logged in the network as

$$N_b = W/RTT (RTT - RTT_{min}) = (W/RTT_{min} - W/RTT) RTT_{min} \quad (2.1)$$

Where W is the current congestion window size, RTT is the last observed round trip time, and RTT_{min} is the smallest round trip time seen since the connection was initiated. Then $RTT - RTT_{min}$ is the total estimated queuing delay and W/RTT is the estimated current throughput; thus the formula for N_b gives the estimated number of packets that this particular source has back-logged in the queues.

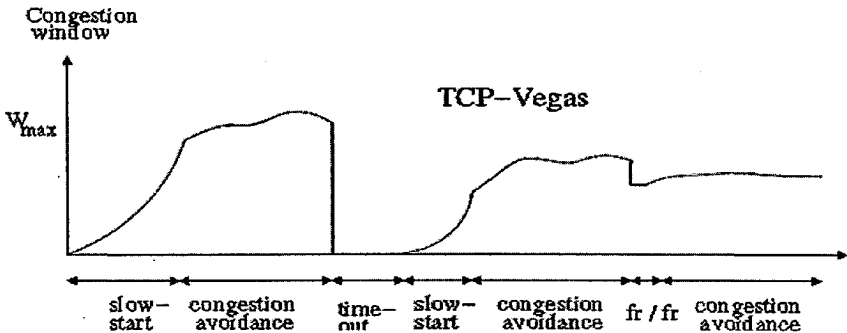
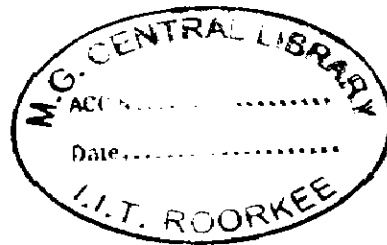


Figure 2.5 TCP-Vegas congestion window evolution



After a successful fast retransmit, TCP-Vegas reduce the window size by one quarter, while TCP-Reno reduces it by one half. Because TCPVegas adjusts the window size at the incipient stages of congestion, it does not need to make a large window size reduction upon loss events. A consequence of the behavior of TCP-Vegas during congestion avoidance is that for long file transfers in a network that is relatively stable in terms of load, given that there is enough buffer space in the network's queues, the TCP-Vegas sources are able to find their ideal sending rates. Hence, a network with TCP-Vegas sources can operate without the periodic packet loss events the other TCP flavors will always create during their search for the maximal possible sending rate. Since each source tries to keep between α and β . packets in the queues, the equilibrium sending rates for the TCP-Vegas sources will however lead to a linearly increasing queuing delay as the number of sources increases. TCP-Vegas also adapt a new algorithm during the slow-start phase. For TCP-Vegas, the congestion window size is doubled only at every second RTT. This allows the TCP-Vegas source to estimate the induced queuing delay in the network for every used congestion window size. This queuing delay is then used to calculate the number of back-logged packets N_b and decide whether to continue in slow-start or not. Particularly, if the delay in the network increases and the source has greater N_b than packets in the queues, TCP-Vegas increases the congestion window by one packet, exits the slow-start phase, and transitions to the congestion avoidance phase. In addition to the delay threshold, a maximum slow-start threshold is maintained in the same way as for the other TCP flavors.

2.8 Shortcomings and Research Gaps

Unfortunately present mechanisms do not give expected result due to packet loss due to corruption in channel in error prone network especially wireless. We know internet technology is changing fast and wireless network is becoming pervasive all around. Packet lost in wireless network may not imply congestion in network because it may happen due to weather conditions, obstacles, and multipath interferences, mobility of wireless end-devices, signal attenuation and fading. So our assumption upon which network congestion window has been changing in TCP congestion control mechanism is not appropriate for wireless.

If we summarize, how is a switch or source to detect congestion? There are several alternatives.

- The most common one is to notice that the output buffers at a switch are full, and there is no space for incoming packets. If the switch wishes to avoid packet loss, congestion avoidance steps can be taken when some fraction of the buffers are full, such as in the Fair Queue bit scheme. A time average of buffer occupancy can help smooth transient spikes in queue occupancy [27].
- A switch may monitor output line usage. It has been found that congestion occurs when trunk usage goes over a threshold (typically 90%) and so this metric can be used as a signal of impending congestion. The problem with this metric is that congestion avoidance could keep the output line underutilized, leading to possible inefficiency.
- A source may monitor round-trip delays. An increase in these delays signals an increase in queue sizes, and possible congestion [28].
- A source may probe the network state using some probing scheme [23]
- A source can keep a timer that sets off an alarm when a packet is not acknowledged 'in time'. When the alarm goes off, congestion is suspected.

Anyone of above mechanism can be tried to implement congestion control, but last one is most frequently in use. And the way it perceive ACK and decide the congestion in network may be misinterpreted as overload rather than corruption. So there is need to improve TCP congestion control, so that congestion control mechanism differentiates between overload and corruption to act accordingly.

3 Proposed TCP Congestion Control mechanism

To rid of the deteriorating situation of error-prone network, especially wireless network, different flavor of TCP congestion control have been suggested and implemented but there is need of such mechanism which can be ported upon wired/wireless network universally. To improve the situation we need to use loss discrimination mechanism i.e. packet loss happens due to high load or corruption in channel [29]. In this improved algorithm we have used two window where one window measures congestion window and another window measures degree of packet loss due corruption in channel. Before starting the algorithm, we have to define packet loss rate.

3.1 Packet loss Rate

One of the major characteristic, which differentiates between wired and wireless network is large error rates due to noise, fading, interference from other sources and mobile host movement. Before we go to design modified TCP, let's define Bit Error Rate (BER).

Bit Error Rate (BER) and Packet Error Rate (PER) are important Quality of Service Parameters for Wireless network. Basically an error event is defined as any divergence of the decoded path at the receiver from the initially followed path in the encoder. For long codes, the error probability of block codes and convolution codes is upper-bounded in terms of error exponent. The error exponent $E(R)$ is defined as

$$E(R) = -1/N \log P_e(N, R) \quad (3.1)$$

Where $P_e(N, R)$ is the error probability of a block code of length N and rate R

With the classical approach where it is frequently supposed that errors are uniformly distributed in packets with a probability given by BER. With this hypothesis the packet

error rate (PER), which is the number of incorrectly received data packets divided by the total number of received packets. A packet is declared incorrect if at least one bit is erroneous. The expectation value of the PER, for which a data packet length of N bits can be expressed as follow [30].

$$PER = 1 - (1 - BER)^N \quad (3.2)$$

For example, normally the bit error rate of wireless BER=0.00001 and the length of data frame Length=1024bits, so the corruption loss rate PER=0.0101878; the PER=0.097336209 while BER=0.0001 and PER=0.641028521 while BER=0.001.

It is reasonable to assume that the possibility of packet (Pe) lost by corruption can be obtained approximately from $Pe = m/n$, where n is the number of total packets and m is the sum of packets lost by corruption during the period of time T.

From the simulation point of view, if the corruption loss rate Pe is higher than the certain lower limit "Pemin", the sending rate will be decreased. Many factors decide the value of corruption loss rate lower limit "Pemin", mainly include: the kind of application; the length of data frame; the bit error rate of wireless link layer; the bandwidth and the transmission delay of wireless network etc. Normally we choose Pemin =0.4.

3.2 Modified TCP

We deigned improved TCP congestion control mechanism by modifying the base design of TCP NewReno [16] and improved it by embedding additional context in TCP state "Fig. 2.2". This improved TCP congestion control mechanism is designed by changing into Slow-Start, Congestion Avoidance, and Fast Recovery part based on present corruption loss rate with respect to "Pemin". Similar to this dissertation approach, which I tried on NewReno, can be tried on other TCP variant too like Reno, Tahoe etc.

This modified TCP congestion control mechanism has been improved for error prone especially wireless network by keeping unaffected wired network. It considers the influences to TCP sender's packet sending rate by packet loss due to overload in channel and the degree of packet loss due to corruption in channel.

During transmission of packet in network we measure the degree of loss of packet due to corruption in channel and accordingly keep the congestion window as normal TCP congestion window otherwise updated to second window, which we maintain in this congestion control mechanism.

We use ewnd (error congestion window) as original congestion window (cwnd) multiplied by corruption loss rate and swnd(similar congestion window) as original congestion window subtract ewnd. It mean

$$cwnd = swnd + ewnd$$

We initialize swnd and cwnd equal to one and ewnd to zero. These three windows will change their value according to present corruption loss rate (P_e). In general if P_e is less than P_{emin} then cwnd will similar to original NewReno cwnd, ewnd is scale of corruption loss and swnd is deviation of ewnd from cwnd.

3.3 Algorithm

3.3.1 Initial Window

The IW, the initial value of cwnd, MUST be less than or equal to $2 * SMSS$ bytes and must not be more than 2 segments. It calculated as follow [7].

$$cwnd = \min(4 * SMSS, \max(2 * SMSS, 4380 \text{ bytes})) \quad (3.3)$$

3.3.2 Slow Start Algorithm

The slow start algorithm is used to start a connection of improved TCP like another TCP, and the periods after the value of retransmission timer exceed the RTO (retransmission timeout). In the start of improved TCP, the size of cwnd will be initialized to 1. The slow start algorithm describes as below

Slow start algorithm is used to start a connection and the periods after the value of retransmission timer exceed the RTO (retransmission timeout). When the ACK is received, the swnd is increased from one to two, and two segments can be sent. This provides an exponential growth. The slow start algorithm will be ended in two conditions.

First, if the congestion window size reaches the slow start threshold size (ssthresh), the slow start will be ended and then congestion avoidance takes over. Second, if there lose any packet due to congestion or high packet loss rate due to corruption, the slow start also will be ended and then fast recovery takes over.

```

if (Receive ACKs && cwnd < ssthresh)
{
cwnd++;
swnd = cwnd;
ewnd = 0;
}

```

3.3.3 Congestion Avoidance Algorithm

If the congestion window size (cwnd) is less than or equal to the slow start threshold size (ssthresh), improved TCP is in slow start; otherwise it is performing congestion avoidance. The congestion avoidance algorithm describes as below

```

if (Receive ACKs || (Receive Explicit Corruption Loss Notification && Corruption Loss Rate Pe < Pemin))

```

```

{
if (cwnd > ssthresh)
    cwnd = cwnd + 1 / cwnd;
else
    cwnd++;

m++;
if (Receive Explicit Corruption Loss Notification)
    n++;

```

$Pe = a * Pe + (1-a) * (n/m)$; [17]

```

ewnd=cwnd* Pe;
swnd=cwnd – ewnd;

if (m>cwnd)
    m=n=0;
}

```

In the algorithm, $cwnd$ denotes the congestion window size; m denotes the total number of sending packets; n denotes the number of lost packets due to wireless link corruption; P_e denotes the corruption loss rate and use parameter “ a ” to add up the old values. The idea of “ a ” has been taken from Jacobson RTT calculation [17].

3.3.4 Fast Retransmission and Fast Recovery Algorithm

The TCP sender should use the "fast retransmit" algorithm to detect and repair loss, based on incoming duplicate ACKs. The fast retransmit algorithm uses the arrival of 3 duplicate ACKs (4 identical ACKs without the arrival of any other intervening packets) as an indication that a segment has been lost. After receiving 3 duplicate ACKs, TCP performs a retransmission of what appears to be the missing segment, without waiting for the retransmission timer to expire.

"Fast Recovery procedure" begins when three duplicate ACKs are received and ends when either a retransmission timeout occurs or an ACK arrives that acknowledge all of the data up to and including the data that was outstanding when the Fast Recovery procedure began [16].

In improved TCP, the fast recovery algorithm will be taken when network congestion or heavy corruption occurs. If network congestion, set $ssthresh$ to one-half the flight size or double of MSS (maximum segment size) window. If the network has high corruption loss rate, set $ewnd$ to “ c ”. ($c < 1$) times its original size [19].

Here the network congestion means the sender receives the same ACK 3 times or retransmission timer overtime. The heavy corruption means the corruption loss rate P_e not less than P_{emin} when sender receives explicit corruption loss notification.

```

if (Congestion || Heavy Corruption)
{
if (Receive Same ACK 3 Times || Retransmission Timer Overtime) /* Congestion */
{
ssthresh = cwnd / 2;
    if (Retransmission Timer Overtime)
    {
cwnd = 1; Exit and call slow-start;
    }
    else /* Receive Same ACK 3 Time */
cwnd = cwnd / 2;
}
}

else if (Receive Explicit Corruption Loss Notification && Corruption Loss Rate
Pe >= Pemin)
{
ewnd = c * ewnd;
cwnd = swnd + ewnd;
};

m++;
if (Receive Explicit Loss Corruption Notification)
n++;

Pe = a * Pe + (1-b) * (n/m);
cwnd = cwnd * Pe;
swnd = cwnd - ewnd;
if (m > cwnd)
m = n = 0;
}

```

4 Experimental Results and Discussions

4.1 Performance Metrics

- **Throughput:** - In communication networks, throughput or network throughput is the average rate of successful message delivery over a communication channel. This data may be delivered over a physical or logical link, or pass through a certain network node. The throughput is usually measured in bits per second (bit/s or bps), and sometimes in data packets per second or data packets per time slot. [31]
- **Delay:-** sometimes we are interested in average time it takes for a block of data from an application on one system to another system .there are four component of this
 1. Transmission delay
 2. propagation delay
 3. processing delay
 4. queuing delay
- **Ratio of throughput of existing TCP with respect to improved TCP should be less than one.**

4.2 Implementation in NS3

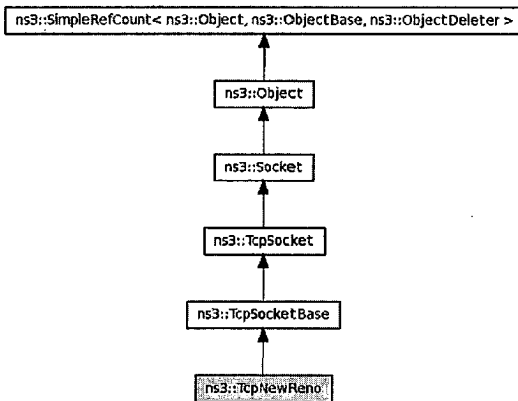


Figure 4.1 class shashi_universalTCP

We have used the scrape of TcpNewReno class, whose inheritance is given in “Fig 4.1” Just following the module creation procedure in NS-3 created new class named as shashi_universalTCP class and wrote proposed algorithm accordingly.

File which get modified are followings

src/internet-stack/shashi-universal.cc

src/internet-stack/shashi-universal.h

Methods modified:

```
/** New ACK (up to seqnum seq) received. Increase cwnd and call
TcpSocketBase::NewAck() */
```

```
NewAck (const SequenceNumber32& seq)
```

```
/** Cut cwnd and enter fast recovery mode upon triple dupack */
```

DupAck (const TcpHeader& t, uint32_t count

```
/** Retransmit timeout */
```

Retransmit (void)

We know that packet receiving and sending process pass through successive layer from application layer to physical layer and passes through shashi_univesalTCP class object to rid of the congestion in network.

To simulate the idea we have written different application on different topology in the presence of error module and tested the result.

4.3 Simulation Environment

To demonstrate effectiveness of our improved TCP congestion control mechanism, we use network simulator version 3.10 (NS3) to study the transport scheme, the proposed model has been implemented and simulated in the network simulation tool NS3. It is open source and is a relatively new simulator. NS3 provides great flexibility while simulating various scenarios [18]. Box illustrates the network topology of the simulation. Channel may be wired or wireless with error model, which adds erroneous channel.

The bandwidth of link is 5Gbps and delay is 20ms.

Length of sending data frame Length=1024bits

For corruption in channel, we are using uniform random distribution on BER 0.001, 0.0001 and 0.00001.

We are sending 1000 packets for this simulation.

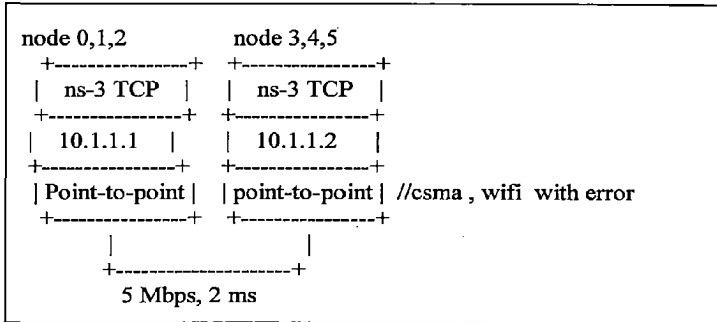


Figure 4.2. NS3 simulation hierarchy

4.3 Results and Discussions

To analyze the result we use TCPTRACE tool. TCPTRACE tool can give information related to Throughput by analyzing the Pcap file, generated by simulation. Table 1 shows the throughput of network at different BER. If we compare the BER and ratio of throughput of NewReno and Improved TCP, it shows that less the corruption more the resemblance. It means if there is no corruption both protocol will show same behavior otherwise Improved TCP shows better performance. This is also implied through “Fig. 4.1” and “Fig 4.2”. “Fig. 4.1” is illustration of congestion window at BER 0.0001. Improved TCP finishes sending packet at 6 ms due to its higher throughput and “Fig. 4.2” shows similar congestion window initially, even afterward its congestion window remains higher with respect to NewReno. In “Fig. 4.2” NewReno congestion window graph is hidden due to overlap by Improved TCP congestion control.

Protocol	Throughput		
	BER=,0010	BER=,0001	BER=,00001
Newreno	4629bps	17811bps	115990bps
Improved TCP	20099bps	37335bps	116712bps
Ratio	4.34	2.096	1.006

Table 4.1 The protocol capability with different BER

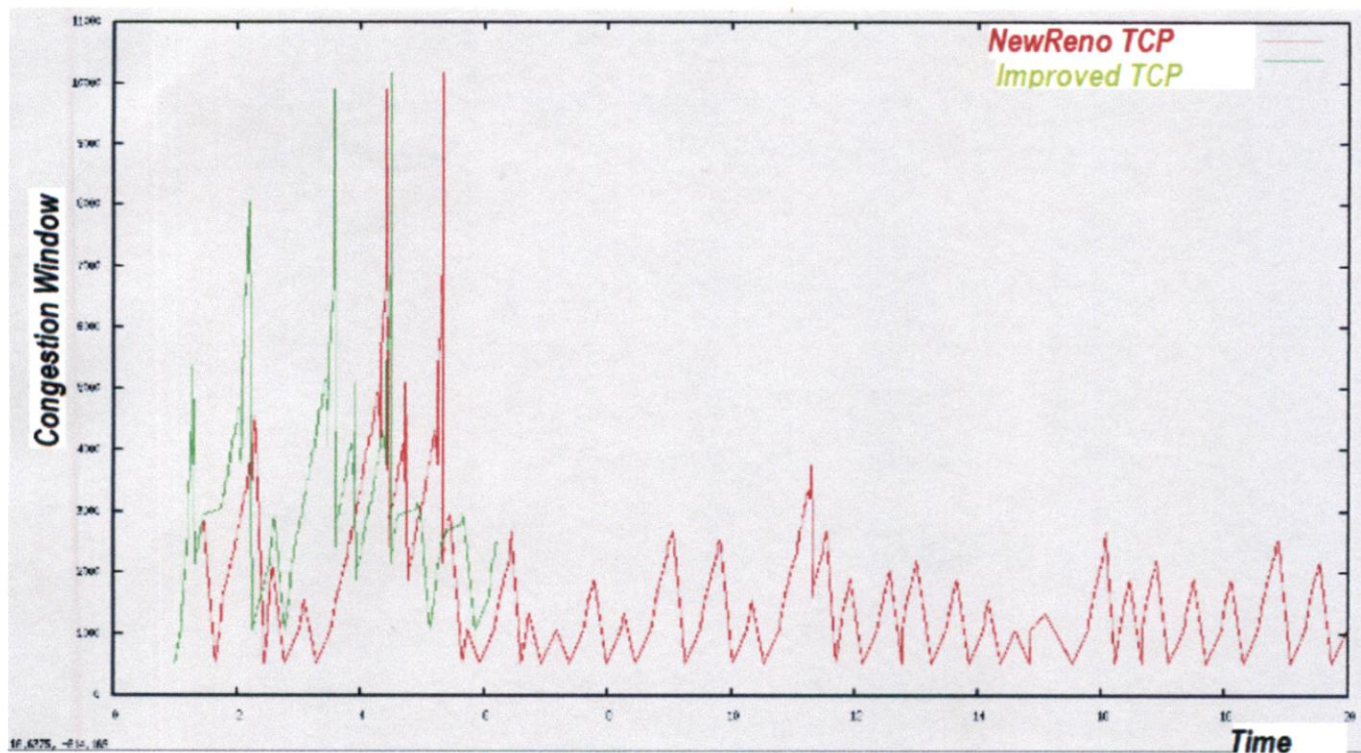


Figure 4.3 Congestion window plot at BER 0.0001

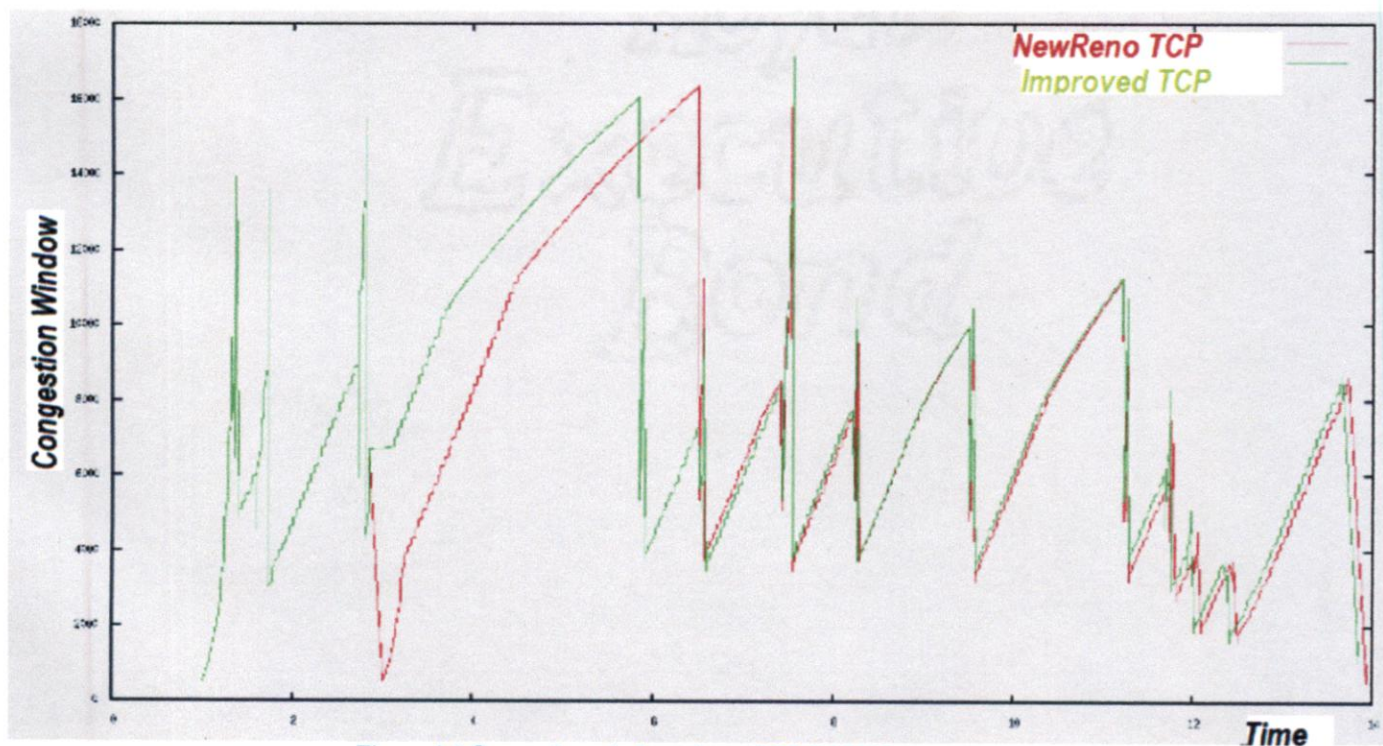


Figure 4.4 Congestion window plot at BER 0.00001

5 Conclusions

5.1 Conclusions

This dissertation presents improved TCP congestion control mechanism to implement on wired/wireless network. It considers the corruption of channel to differentiate the wired and wireless network. Corruption of packet is major hurdle of traditional TCP to implement universally on wire and wireless. Proposed algorithm is able to shows better performance by identifying the packet loss due to corruption in network rather than overload.

Following finding have been found out through this dissertation.

- TCP NewReno is better among available variant of same methodology and been compared with our Improved TCP congestion control mechanism.
- Result available in report shows improvement of improved TCP over NewReno TCP congestion control mechanism.
- Similar to NewReno TCP tahoe, Reno TCP been tested and have positive result.
- Whatever be the network, throughput of network goes to meet at same point in case of reduction in degree of corruption.
- Because of reusing of base work of available TCP variant, it is affordable to follow it.
- During the implementation I have been trying to put hand on packet pair mechanism has promising scope to try on.

5.2 Scope for Future Work

In this dissertation we have taken consideration of packet loss due to corruption in channel. Here real problem lies how to differentiate packet loss due to corruption or overload [32]. As per simulation of above proposal is concern, NS3 has its own feature to identify packet loss by overload or corruption. Simply corrupted packet will defiantly reach to end node rather than get lost in between the end nodes.

Following consideration can be thought as a future work.

- Implementation of this mechanism would as future work. One needs to recognize the reason of packet loss. For this one can use option field of TCP packet or ICMP message.
- This mechanism can be embedded in all such TCP variant, which do the congestion control by recognizing the packet loss
- Here we have considered only packet loss without packet correction; it can be tried with packet correction overload.

References

- [1] Mario Gerla and Leonard Kleinrock, "Flow Control: A Comparative Survey," *IEEE Transactions on Communications COM-28(4)*, April, 1980, pp. 553~574.
- [2] Prof. Samuel Madden, MIT, "Congestion control," [Online video] Available. <http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-033-computer-system-engineering-spring-2009/video-lectures/lecture13/>, 2008.
- [3] Dimitri P. Bertsekas and Robert G. Gallager, "Data Networks," *2nd Edition*, Prentice-Hall, India. 2004.
- [4] J. Postel (ed.), "Transmission control protocol," *RFC 793*, September 1981.DARPA Internet Program.
- [5] V. Jacobson, "Congestion avoidance and control," *ACM Computer Communication Review*, 18:314-329, 1988.
- [6] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," *RFC 1323*, May 1992.
- [7] Allman, M., Paxson V. and W. Stevens, "TCP Congestion Control", *RFC 2581*, April 1999
- [8] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, "An extension to the selective acknowledgement (SACK) option for TCP," *RFC 2883*, July 2000.
- [9] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgment options," *RFC 2018*, October 1996.
- [10] Floyd S, "A report on some recent developments in TCP congestion control," *IEEE Communication Magazine*, 2001, 39(4) : pp.84-90.
- [11] P. Karn and C. Partridge, "Improving round-trip time estimates in reliable transport protocols", *ACM Trans. Comput. Syst.*, vol. 9, 1991, pp.364 - 373.
- [12] I. Naqvi and T. Perennou, "A DCCP Congestion Control Mechanism for Wired-cum-Wireless Environments," in *IEEE Wireless Communications and Networking Conference (WCNC)*, Hong Kong, Mar. 2007.

- [13] N.K.G. Samaraweera, "Non-congestion packet loss detection for TCP error recovery using wireless links", *IEE proc.-commun.*, Vol.146, No. 4, August 1999.
- [14] Xu Chang-Biao, Long Ke-Ping and Yang Shi-Zhong, "Corruption-based TCP rate adjustment in wireless networks", *Chinese Journal of Computers*, 2002, 25(4), pp.438-444.
- [15] Ramin Khalili and Kave Salamatian, "A New Analytic Approach to Evaluation of Packet Error Rate in Wireless Networks," *In Proceedings of CNSR'2005*, pp.333-338
- [16] S.Floyd and T.Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," *RFC 2582*, Apr 1999.
- [17] Karn, Phil and Craig Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols," *ACM SIGCOMM '87*. pp. 2-7.
- [18] Ns-3 tutorials [Online]. Available: <http://www.nsnam.org/ns-3.10/tutorial>
- [19] Mukesh Kumar Dhariwal and Sanjeev Sharma, "An Improved Mechanism for Congestion Control in TCP for Ad Hoc Network," *International Journal of Computer Applications (0975 – 8887) Volume 20–No.2*, April 2011
- [20] Jorgen Olsen, "Stochastic modeling and simulation of the TCP protocol," *Uppasala dissertations in mathematics*, 2003.
- [21] Niels M Oller, "Window-based congestion control," *Doctoral Thesis Stockholm, Sweden*, 2008.
- [22] Krister Jacobsson, "Dynamic modeling of Internet congestion control," *Doctoral Thesis Stockholm, Sweden*, 2008.
- [23] S. Keshav, "Congestion Control in Computer Networks," *PhD Thesis, published as UC Berkeley TR-654*, September 1991
- [24] R. Braden (ed.), "Requirements for internet hosts — communication layers," *RFC 1122*, October 1989.
- [25] TCP Tahoe, [Online]. Available: "TCP Variants," <http://www.hep.ucl.ac.uk/~yjl/tcpip/background/Tahoe-reno.html>, 23 July, 2003.
- [26] S. Floyd, "The NewReno Modification to TCP's Fast Recovery Algorithm," *RFC 2582*, 1999.

- [27] K. K. Ramakrishnan and R. Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks," *ACM Trans. on Comp. Sys.* 8, 2 (May 1990), 158-181
- [28] R. Jain, "A Delay-based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks," *Computer Communications Review*, October 1989, 56-71
- [29] I. Naqvi and T. Perennou, "A DCCP Congestion Control Mechanism for Wired-cum-Wireless Environments," *IEEE Wireless Communications and Networking Conference (WCNC)*, Hong Kong, Mar. 2007.
- [30] Ramin Khalili and Kave Salamatian, "A New Analytic Approach to Evaluation of Packet Error Rate in Wireless Networks," *In Proceedings of CNSR, 2005*, pp.333-338
- [31] Andrew S. Tanenbaum, "Computer Networks," Fourth Edition, *Pearson Education*, 2009.
- [32] N.K.G. Samaraweera, "Non-congestion packet loss detection for TCP error recovery using wireless links," *IEE Proc.-Commun.*, Vol. 146, No. 4. August 1999

List of Publication

- [33] Shashi Ranjan and Prof R. C. Joshi, "TCP Congestion control mechanism for wired/wireless network" *International Conference on Computer Science and Information Technology (ICCSI-2011)*, Banglore, India, 24-25th July, 2011 (Accepted).

Appendix

NS-3 Simulator

The NS-3 simulator is a discrete-event network simulator. Unlike to NS-2 it requires only one scripting language rather than two. "Fig A.1" shows the general organization of NS-3 simulation software.

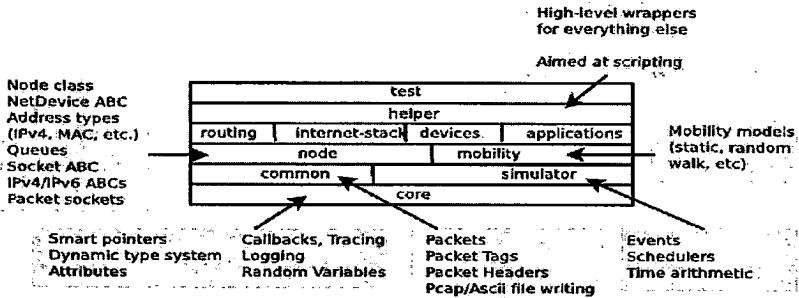


Figure A.1 organization of NS-3

Each of level is module in NS-3, whose source code is available in "src" directory.

To compile the source code it uses "waf" tools. It is very similar to "make" tools with additional simplification.

To execute the code following command works

```
./waf --run "file name"
```

Modules are written in c++ object oriented programming language, which follows OOPs concepts and can be modified code accordingly.

What differentiates this simulator with others is its tracing capability of required object.

Inbuilt number of tracing source is given and new one can be built.

How different module participate to send a packet from one node to another. It can be visualize by following way.

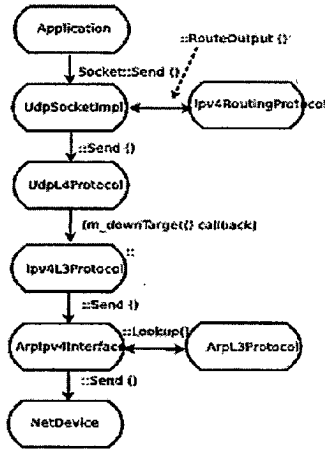


Figure A.2 packet sending process

Each of box represent object of that class, which further can be subdivided accordingly. Similarly packet receiving process will follow backward at different layer.

Tcptrace

Tcptrace is a tool for analysis of TCP dump files. It can take as input the files produced by NS-3 simulator. Tcptrace can produce several different types of output containing information on each connection seen, such as elapsed time, bytes and segments sent and received, retransmissions, round trip times, window advertisements, throughput, and more. It can also produce a number of graphs for further analysis. As of version five, minimal UDP processing has been implemented in addition to the TCP capabilities.