Transport Control Protocol Instructional Program Design

A Dissertation submitted to the University of Manchester for the degree of Master of Science in the Faculty of Engineering and Physical Sciences

2017

Student ID Number: 9918492

School of Electrical and Electronic Engineering

Contents

List of Tables

List of Figures

Abstract

Nowadays, the requirement of communications between computers in different networks is dramatically high. To ensure the efficiency and security of the communications, TCP/IP has developed to as a common protocol for modern Internet. As a reliable transmission protocol, TCP is an important content of TCP/IP. Therefore, the theory of TCP is required to be mastered by students in Communication Engineering and Computer Science. However, the conceptions of TCP are abstract and difficult to be understood. A interactive program which can show the TCP behaviour in a vivid way is pretty helpful to students. The aim of this project is to develop an interactive program to support MSc Communication Engineering students for studying TCP. The project will focus on the TCP congestion control mechanisms which is a core content of TCP. Different network topologies was simulated in this project. Firstly, a two nodes network containing one sender and one receiver was concerned. Then, a router was added to the network. After the implementation phase, the program was tested to assure it has behaved splendidly to be a qualified simulator for instructional purposes.

Declaration

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

Firstly, I am truly and sincerely obliged to my supervisor, Dr Peter Green, for his valuable advice, help and encouragement which guarantees the completion of this dissertation.

I would also express my gratitude to my parents for their financial support and continuous care and concern throughout my study period.

Finally, thank you for all my friends in Manchester for their company and help which makes me feel happy and joy over the year.

Chapter 1

Introduction

1.1 Motivation

Undoubtedly, the scale of Internet and its usage has experienced a dramatic increase in the past two decades. Actually, the Internet is the network of networks. Computers in networks wish to communicate with each other for information exchange. Just like two people should agree to use a common language when they communicate, computers also need a common "language" to transfer information. A common "language" for the computer is Transmission Control Protocol/Internet Protocol (TCP/IP). To be more specific, TCP/IP is a set of rules in terms of addressing other computers and sending data to them [24, p.1]. TCP/IP is the de facto protocol of the Internet. Any computer wants to communicate on the Internet will have to use TCP/IP [24, p.2]. TCP/IP can work effectively whether for a network connecting two hosts or an internetwork for business with 10,000 hosts [24, p.11]. In particular, as a transport protocol which is responsible for a great majority of Internet applications such as Web, file transfer, music, social software and online games, transport control protocol (TCP) plays an important role in today's Internet.

TCP consists of many mechanisms to achieve the function of connection establishment, management and reliable data transport between pairs of processes [2]. Flow control and congestion control are two important mechanisms of TCP. Flow control provides service to match the sending speed of the sender against the reading speed of the receiver while congestion control uses algorithms to deal with network congestion [3]. TCP can reduce the probability of packets loss and provide reliable data transport by this mechanism.

TCP is important for designing and optimizing Internet applications and networks. So, the principle of TCP must be mastered by students on the MSc programme in communication engineering at the University of Manchester. However, many concepts in the lectures on network protocols are abstract and difficult to be understood. An interactive program would be helpful for students to learn about TCP because they can specify the network topology and parameters by themselves and investigate the behaviour of this TCP model.

There are already some well-known network simulators which can implement TCP such as ns-2 and ns-3 [4][5]. However, they are both complex simulators that are intended for research and are time-consuming to learn and use [6]. So, they are not appropriate for instructional purposes. The requirement of the simulator for master students to use should not only be simple but also provide visual results to make users gain a deep understanding of the TCP protocol.

1.2 Aims

The aim of this project is to develop an interactive program for students who are taking the Internet and Communications Networks unit as part of the MSc in Communication Engineering. This program should be a simulator to provide simulations of TCP's congestion control and flow control mechanisms. Through it, students can understand how the protocol affects network performance. It should satisfy the following requirements.

1. Users are allowed to specify a network with associated performance parameters such as the size of node buffers, round trip time, the total number of packets, etc.
2. Algorithms should be designed to simulate packets loss.
3. The program should be a real-time system, i.e. the program state varies with time.
4. The program can respond correctly when different events happen and plot figures to show simulation results.
5. A graphical user interface (GUI) should be produced. This GUI is simple to understand and used by students on MSc Communication Engineering Programme.

1.3 Objectives

The objectives in order to achieve the aim of the project are:

1. Read and review the literature related to networks, protocols and TCP. This provides

a background about TCP implementation.

2. Review existing TCP implementations, make a comparison of the methods that were used and the assumptions were made in the code.
3. Simulation of networks and implementation of TCP's mechanisms.
4. Production of a GUI in order to select network topology, specify associated performance parameters and start transmission.
5. Testing and evaluation of the simulation to ensure that the simulator works well and all mechanisms are implemented properly.

1.4 Project Scope

This project focuses on the implementation of TCP's flow control and congestion control mechanisms. The transmission of packets is expected to be simulated. However, the processing of data in packets is not the concern of the project. Additionally, the establishment of a TCP connection between two hosts is a necessary procedure in real systems. But it is not considered in this project.

1.5 Dissertation Structure

In the remainder of this paper, chapters are organized in the following order.

Chapter 2 includes a literature review and background knowledge associated with this project. This includes the introduction to computer networks, the protocol stacks, principles of reliable data transfer and TCP mechanisms. Furthermore, several existing TCP models are reviewed.

The design architecture and details of the implementation phase are covered in Chapter 3. The explanation of the functions and assumptions are also included in this chapter.

Chapter 4 covers the test and evaluation of the TCP simulator. More specifically, different parameters are set and the simulation results are compared with theoretical results.

Finally, conclusions and limitations of this work are discussed in Chapter 5. Recommendations for future works are also provided.

Chapter 2

Background

2.1 Introduction

There are billions of computers be used all over the world.   Moreover, with the development of memory and processing power technology, users can choose to work on new kinds of applications and functions [7]. Accordingly, the pressure to communicate all these machines is irresistible. There are mainly two types of demands for connectivity: communications software and networks.

2.2 Computer Networks

There is no taxonomy that is efficient to classify all computer networks . However, transmission technology and scale are two major dimensions [8]. Transmission technology is generally divided into broadcast networks and point-to-point networks [8].

For broadcast networks, a common communication channel is used by all machines on the network. When one machine sends a packet, all others can receive it. The packet contains the information of destination address. When a packet is received, a machine checks the address information to detect whether it is the destination and choose to process or just ignore it. Broadcast systems also permit to transmit a packet to all machines on the network by using a special code in the address field [8].

In contrast, a packet in point-to-point networks may have to visit one or more intermediate machines in the process of transmitting from source to destination. Routing algorithms are important to point-to-point networks as there are often multiple routes of different lengths. Broadly speaking, smaller networks usually tend to use broadcasting, while point-to-point is often used in larger networks [8].

Another way to classify networks is to give classification according to their scale. For this kind of criterion, networks can be divided into local area networks (LANs), metropolitan area network(MAN), wide area networks (WANs). Finally, several networks are connected to form an internetwork (Internet) [9].

2.2.1 Local Area Networks

Local Area Networks belong to a private organization who is using the network to connect equipment to share resources and exchange information [7]. The key elements of LANs are their size, topology and transmission technology. LANs are restricted in size, so the bound of the transmission time is known beforehand [8]. This makes LANs different from other types of networks. Another important property of LANs is topology. Topology means the way to interconnect machines on the networks. Topology types of bus, tree, ring and star are historically common for LANs.

2.2.2 Metropolitan Area Networks

Basically, a metropolitan area network uses the similar technology as a LAN but with a larger scale. Apart from providing data and voice services, MANs can also support the local cable television. A MAN does not contain switching elements and it is not necessary to switch simplifies the design [11].

A key aspect of MAN is that all the machines on it are attached to a broadcast medium. Compared to other kinds of networks, this makes the design greatly simple [8].

2.2.3 Wide Area Networks

A wide area network generally covers a large geographical area, often a country or continent, require the crossing of public right-of-ways. A typical WAN is consisted by some switching nodes which are interconnected. These nodes route transmissions from devices to the specified destination device. The purpose of these nodes is switching, which means moving data pass through nodes to reach their destination [12].

2.2.4 Internet

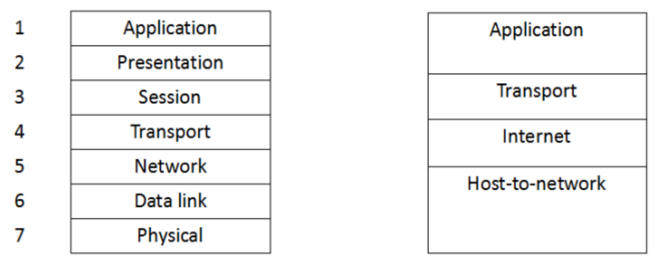
There are many networks exist in the world. People within different networks often want to communicate with others. To satisfy this requirement, networks which are different, and frequently incompatible, need to be connected. Machines called gateways are used to establish connections and make necessary translations [8]. Networks are connected to form an internetwork, or just internet [8].

In 1969, the Advanced Research Project Agency (ARPA) of the U.S. Department of Defense developed the ARPANET, which is the first operational packet-switching network [7]. The worldwide Internet evolved from it. There are hundreds of millions of hosts and billions of users today. And near 200 countries participate the Internet. The number of connected devices continues to increase rapidly [7].

2.3 Protocol Stack

Network designers organize networks as a series of layers or levels to reduce their design complexity. Layers in different networks have different named, contents and function. But of layers in all networks are designed to provide the higher layers certain services [8]. In an ideal case, if layers are defined, making changes in one layer will not require changes in other layers [7].

The same set of layered functions must exist in two systems when it takes two to communicate [7]. If layer n on a machine wishes to communicate layer n on another machine, it has to carry on a conversation. In this conversation, the peer layers must obey specific rules and conventions. These rules and conventions are known as the layer n protocol [7]. Basically, a protocol is defined as 'an agreement between the communicating parties on how communication is to proceed.' [8]



(a) (b)

Figure 2.1 (a) The OSI reference model and (b) TCP/IP reference model

Kurose and Ross [3] define protocol stack as 'the protocols of the various layers.' In a protocol stack, a layer n protocol can only interact with protocols at layer n-1 and layer n+1. There are two important protocol stack model, the seven-layer OSI reference model and the TCP/IP Reference Mode.

2.3.1 The OSI Model

Open System Interconnection (OSI) model was proposed by the International Organization for Standardization in the late 1970s. This model organizes computer networks into seven layers. After the OSI was proposed, many training and university courses started to use it for networking education [3].

Figure 2.1 (a) shows the seven-layers OSI reference model. The seven layers of it are: application layer, presentation layer, session layer, transport layer, network layer, data link layer, and physical layer [8]. Some of these layers are introduced in detail in the following part of this section.

The Physical Layer

The physical layer is designed to finish the task of transmitting raw bits through a communication channel [8]. This layer has to be deigned to make sure the receiving side receives a same bit as the sending side. For example, if a 1 bit is sent by one side, the other side will receive a 1 bit as well, not a 0 bit [8].

The Data Link Layer

The main task of Data Link Layer is to control the errors that might occur when transmitting data to the network layer [8]. This task is achieved by splitting the input data into data frames named packets. Packets are transmitted to the receiver. The receiver will send back acknowledgement frames when frames are received. Through processing these acknowledgement frames, error-detect can be done. Another task of Data Link Layer is link flow control. This is done by employing some traffic mechanism to give the transmitter information about the buffer space of the receiver [8].

The Network Layer

The network layer is responsible for controlling the operation of a subnet. A key task is to route network-layer packets between source and destination. Bottlenecks, which means packets get in each other's way, will happen if too many packets are present on the subnet at one time [8]. Network layer also provides service to control such congestion.

When transmitting a packet between different networks, many problems may occur. Different networks have different addressing. The maximum packet size networks can accept is different. The protocols of networks also various, and so on. Allowing interconnecting heterogeneous networks is a good way to address these problems [8].

The Transport Layer

Data from session layer is received by transport layer and is split into smaller if required. Then they passed to the network layer, and a special function is used to ensure all pieces arrive correctly. The whole process must be efficient [8].

Tanenbaum mentions that 'The transport layer is a true end-to-end layer, from source to destination.'[8, p.32] For example, in the transport layer, the source program uses the message headers and control messages to carried on a conversation with the destination machine. However, in the lower layers' protocols, the source and destination machines are possibly separated by many routers [8, p.32].

In addition, transport layer must provide a mechanism to avoid a host sending date too fast which will overflow a slow host. Such a mechanism is known as flow control, which is very important for the transport layer.

The Application Layer

The application layer provides a place for network applications and their application-layer protocols to reside [8, p.34]. It also contains functions for transferring files. The difference between different file systems including file naming conventions, ways to represent text lines, and so on. In order to handle these, the file-transfer function is needed when transferring a file between two different file systems [8, p.34].

2.3.2 The TCP/IP Reference Model

TCP/IP came from ARPANET (which is mentioned in section 2.2.4). TCP/IP protocol suite is most widely used today, and it provides the foundation for the Internet [7]. As shown in figure 2.1 (b), there are five layers in TCP/IP reference model. Different from the OSI model, presentation layer and session layer are not presented in the TCP/IP reference model.

The TCP/IP internet layer is very similar to the OSI network layer in terms of functionality [7]. In TCP/IP, the Internet layer is designed to route IP packets and avoid congestion [7]. The TCP/IP application layer contains all the high-level protocols. The application developer can decide whether to build relevant functionality into the application according to the requirements [3].

The TCP/IP reference model does not give much information about the actions in host-to-network layer, except to make sure that the connection between the host and the network is carried on by using same protocol which allows the IP packets to be sent over it [8]. As mentioned in [8, p.38]: 'this protocol is not defined and varies from host to host and network to network.'

2.3.3 Packets[1]

Under most circumstances, data exists as rather long files. Transmitting a large block of data at once would lead to several problems. First, as shown in Figure 2.2, all computers in the network have to wait while the data is being moved [25]. Therefore, it is impossible to make timely interaction and communications when a large block is being sent. Moreover, the larger the block of data, the greater the probability of error. When an error occurs, data are required to be resent. Retransmitting large volumes of data leads to worse network traffic. To minimize these effects, large data units are divided into smaller packages which contains only a small amount of data. These small packages are called packets.

[1] Packets and segments are used interchangeably in this dissertation.

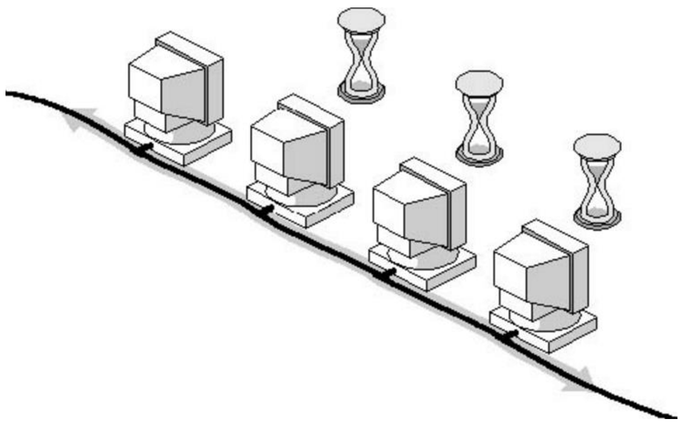


Figure 2.2 Large continuous streams of data slow down the network

https://www.microsoft.com/mspress/books/WW/sampchap/1956/F03xx05x.jpg

2.3.4 Message Formatting

Every protocol uses a special kind of formatting method to specify packets structure. Packets for different applications could be quite different. As shown in Figure 2.3, each packet generally contains three basic elements: header, data and footer [26].

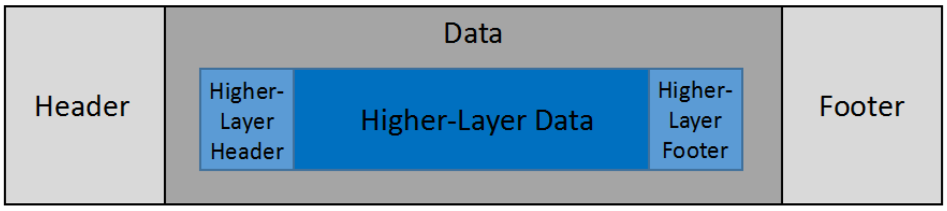


Figure 2.3: Network Message Formatting

Header

Header is the information located before the actual data. There are normally a small number of bytes of control information in the header [26]. The control information is used to support the delivery of data. It usually contains source and destination network addresses, error detection codes, and sequencing information [27].

Data

The actual data to be transmitted is often called payload. In each packet, data usually includes payload and header information from higher layers. When the header information reaches its destination, it is stripped from the packet. As a result, the destination application can only receive the payload at the end [28].

Footer

Footer, which is also called trailer sometimes, is information placed after data. Actually, there is no real difference between header and footer information, because they both generally contain control fields [26].

2.5 Transmission Control Protocol

The aim of designing transmission Control Protocol (TCP) is to provide a reliable end-to-end transmission. TCP must meet the reliability requirements from most users which are not provided by IP [8].

2.5.1 TCP acknowledgement and window strategy

The acknowledgement mechanism is the core of TCP. When data is received by the receiver, according to the requirement of the protocol, an acknowledgement of this data should be sent back. Because the bytes of data are sequentially numbered, TCP receiver only needs to track the highest byte number of arrival packets and send an acknowledgement to the sender to indicate that the data before this byte number is received correctly [29].

TCP uses window mechanism to ensure the reliability of in-order packets transmission and provides flow control. Whenever appropriate, the receiver returns a number to the sender to inform the currently available buffer size for additional data. This number of bytes, called window, defines the maximum which the sender can transmit [29]. New data cannot be transmitted until an additional window is returned by receiver [29].

2.5.2 TCP Timeout and Retransmission

Timeout and retransmission is one of the critical elements of any implementation. To avoid data packets and acknowledgements losses, TCP set a retransmission timer in the process of transmission [30]. Data will be retransmitted if it is not acknowledged before the retransmission timer expires [30]. The retransmission timeout is driven based on the estimating round-trip time and sample round-trip time [3]. A retransmission timeout (RTO) occurs when too many packets and acknowledgements are lost [31].

2.5.3 TCP segments

The TCP segment is consisted by header fields and a data field. The definition of sequence number is mentioned in [3]: ‘In a segment, the sequence number for a segment is the byte-stream number of the first byte.’ Figure 2.4 shows TCP segments are constructed out of the data stream [3]. The sequence number of the first segment is 0, the second is 1000, the thired is 2000, and so on. The information of sequence number is held in a specific filed in the header of the segment called the sequence number filed [3]. The input and output parameters of sender and receiver should content sequence number and acknowledge number to implement the transition of data. In case of missing byte, TCP can provide cumulative acknowledgements [3].

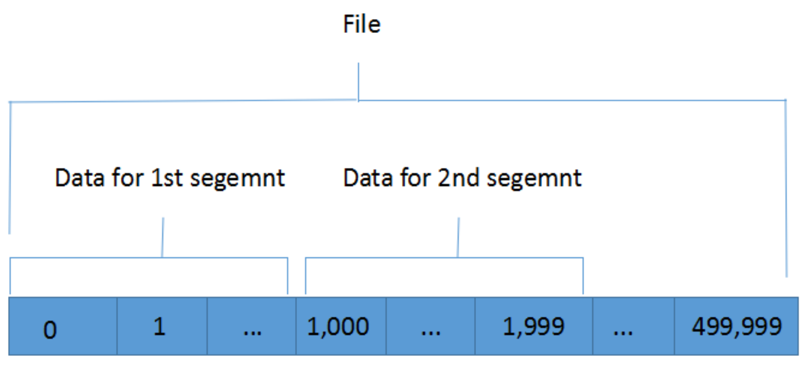


Figure 2.4 Dividing file data into TCP segments

2.5.4 TCP Flow Control

When correct and in sequence bytes are received by the TCP connection, the data are placed in the receive buffer. In most case, the application will not read the date from buffer immediately. If the sender sends too much data too quickly, the receiver buffer can be easily overflowed. To deal with this issue, TCP provides flow-control service to match the sending rate at senders against the reading rate of application. This can be done by putting a variable called receive window into the sender maintain.

If the window size is donated by RcvBuffer, the receive window, denoted rwnd, should be set to the number of spare room in the buffer [3]: `rwnd = RcvBuffer - [ LastByteRcvd - LastByteRead ]`

Define LastByteTx as the sequence number of the last byte sent and LastByteACKed as the sequence number of last byte received by the receiver and Acknowledged. To avoid overflowing the receive buffer, the sender must ensure [3]: `LastByteTx - LastByteACKed ≤ rwnd`

2.5.5 TCP Congestion Control

One celebrated TCP congestion-control algorithm has three major components including slow start, congestion avoidance and fast recovery. TCP sender limits the transmission rate by tracking of an additional variable, the congestion window, denoted cwnd [3]. To satisfy the requirement of congestion control, the sender must control the amount of unacknowledged data to be less both than cwnd and rwnd [3]: `LastByteSent - LastByteAcked ≤ min {cwnd, rwnd}`

Slow Start

When transmission begins with unknow conditions, slow start algorithm is used to avoid congesting the network with sending a large number of segments in a short time. Slow start is also used when loss detected by the retransmission timer is repaired [13].

In a slow start state, the value of cwnd begins at 1 MSS and every time a segment is first acknowledged, it increases by 1 MSS [3]. When an acknowledgement arrives, TCP sender increase cwnd by 1 MSS and sends out two maximum-sized segments [3]. In this way, the size of cwnd keep doubling until reaches ssthresh (shorthand for “ slow start threshold”). If a loss event occurs caused by a timeout, the sender sets the value of cwnd to 1 and sstresh to cwnd/2, then begins the slow start process anew [3].

Congestion Avoidance

There are two reasons lead to packets loss. The first one is packets damage and the second network congestion and the buffer is overflowed somewhere on the path. For most cases, packets loss caused by the first reason is rare (much less than 1%) so packets loss is mainly caused by congestion in the network [32]. Therefore, the loss of a packet can be regarded as the signal of congestion. Congestion avoidance is a way to deal with lost packets. Slow start state ends when cwnd reach the value that caused packets loss last time, then congestion avoidance takes over [14].

There are several algorithms to update windows size in current paper under conditions of loss-based TCP congestion control, delay-based congestion control, mixed loss-delay based TCP congestion control and explicit congestion notification [33]. Mathis *et al*. have provided detail algorithm of this kind of congestion control in terms of congestion avoidance. It uses constant p to represent random packet loss probability. The windows size then is driven [34].

A common approach to achieve congestion avoidance is for the sender to increase cwnd by MSS bytes (MSS/cwnd) every time one new acknowledgement arrives[3]. On the other hand, this algorithm performs the same behaviour as slow start state when timeout occurs.

Fast Retransmit and Fast Recovery

A TCP receiver should produce a duplicate ACK and send it to the sender immediately when an out of order arrives. Before the loss segment is acknowledged, all segments received will trigger duplicate ACKs. As re-order segment can also result in duplicate ACKs, the fast retransmit algorithm regards the case of triple duplicate ACK as an indication of segment loss [13]. Once 3 duplicate ACKs are received, a retransmission is performed before the retransmission timer expires[13].

After the stage of fast retransmit, congestion avoidance rather than slow start is performed. This algorithm is called fast recovery. It improves throughput considerably in the circumstance of congestion is moderate, especially when the size of window is large.

When out-of-order segments are received, the receiver generates duplicate ACKs and buffer these segments. This means data flowing still exists between the two ends, and an abrupt reduction of flow is not expected [14].

In order to implement fast retransmit and fast recovery, the following algorithms are provided

[15].

1. When the event of “triple duplicate ACK” happens, ssthresh is set to the half of cwnd but the value must be greater than two.
2. Resend the lost segment and change the value of cwnd to ssthresh plus 3*MSS. As there are already 3 segments received and buffered by the receiver, the congestion window should be increased artificially.
3. After sending the lost segment, the sender increment cwnd by 1 MSS every time an additional duplicate ACK is received. A new segment will be transmitted if allowed by the new value of cwnd.
4. When the new ACK arrives, set cwnd equal to the value of ssthresh. This ACK should acknowledge all arrival out of order segments after the lost segment.

However, the performance of this algorithm is not very efficient if multiple packets are dropped from a single window of data [16]. To address this problem, [17] provides a set of modifications to this algorithm.

Figure 2.5 provides complete description of TCP’s behaviour in slow start, congestion avoidance, and fast recovery algorithms. Information about when to transmit new segments or retransmit lost segments is also shown in the figure. A mathematical model named Finite State Machine (FSM) is used to describe TCP congestion control in this figure. An FSM model contains a finite set of states and a specific method to map current state into a new state [35].

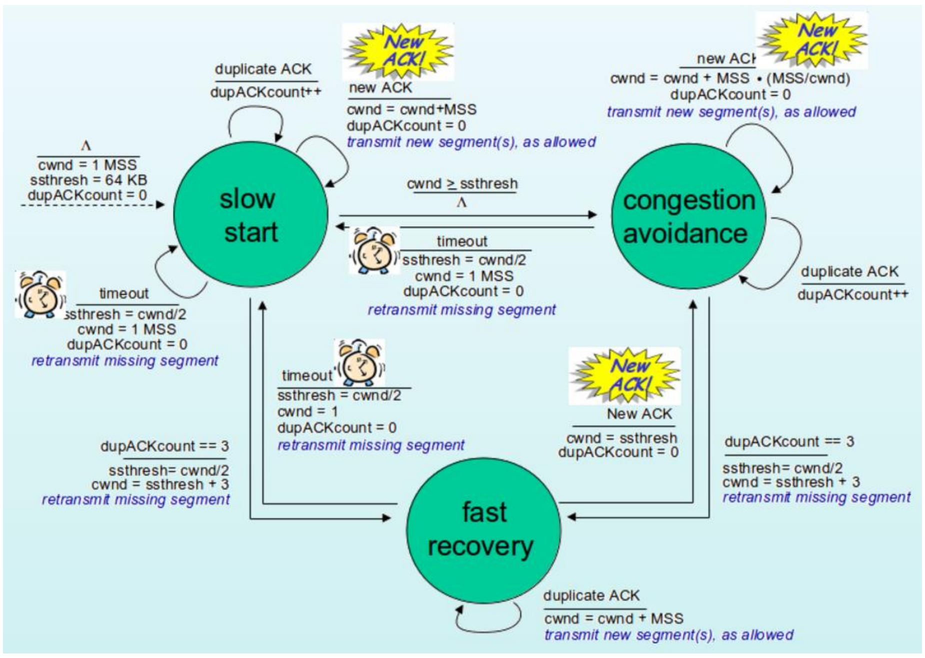


Figure 2.5 FSM description of TCP congestion control

http://imada.sdu.dk/~jamik/dm543-14/material/chapter3.html#Summary:-TCP-Congestion-Control.

2.6 Related Work

Many TCP models have been introduced in different papers in recent years. The project absorbs some design ideas in the model in [19]. Also, the results in [20] are used to evaluate the relevance of the project.

[19] introduces a TCP Reno model in which congestion avoidance, fast retransmission and timeout mechanisms are provided. However, the model in [19] does not consider slow start and fast recovery. In this model, congestions caused by timeout and three duplicate ACKs are analyzed separately. In [19], the behaviour of TCP is modelled in terms of “rounds.” At the beginning of every round, W packets are transmitted, where W equals the current congestion windows size. A round ends when the first ACK for one of these W packets is received, and the next round begins at the same time. To build this model, the author makes the following assumptions:

1. The value of RTT is a constant and it is independent of window size.

2. Only long lived connections are considered which means the sender always has enough data to send.
3. The total time to send all the packets in a window is smaller than RTT.
4. Packets loss in different rounds is independent. However, if a packet is lost in a round, all other packets in this round are assumed to be lost as well.

In [20], comparisons are made between TCP congestion avoidance simulation using FreeBSD and NS-2. Compared with NS-2, the window of FreeBSD inflates and deflates during fast recovery stage, which is shown in Figure 2.6. This may due to different implementations of fast recovery mechanism.

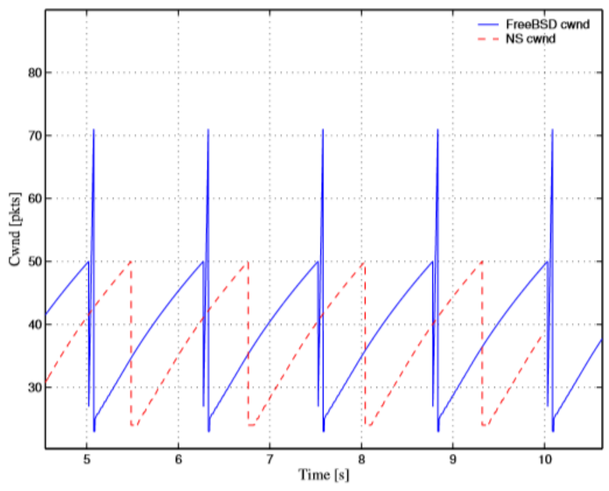


Figure 2.6 Comparison of one TCP flow during Congestion Avoidance using FreeBSD and NS [20]

2.7 Summary

This chapter has introduced the fundamental background for this project including different scale computer networks, two protocol stack reference models, the basic principles of TCP and the different mechanisms of TCP. Review of related work is presented after background. Firstly, a discrete-time model is introduced, and the assumptions used to build this model are analyzed. Then, different performances of NS-2 and FreeBSD TCP implementations are shown.

Chapter 3

Conceptual Design

To satisfy the requirements of the TCP simulator in chapter 1, several simulation tasks are given in the first section refers to the theories mentioned in chapter 2. The reasons to set these simulation tasks are also introduced in this section. Then, a series of assumptions are made for simplicity. Finally, a model architecture is proposed in order to finish these tasks.

3.1 Simulation Tasks

The performance of networks with different topologies is quite different. Therefore, it is necessary to evaluate the performance of different topology separately. In this section, different tasks are proposed to simulate different network topologies.

3.1.1 Simulation Task 1: Demonstrating Congestion Control Mechanism

The congestion control mechanism is the core of the simulator. In section 1.1, it has been mentioned that the mechanism is abstract and difficult to be understood when students first contact it. In Simulation Task 2, a large number of events may occur in a short time and the response to a specific event is hard to be observed. Due to these reasons, we decide to set this task as the first task for the purpose of basic introduction. Also, students can get familiar with MATLAB GUI through this sample simulator for Simulation Task 1. The aim of this task is to demonstrate the behaviour of TCP in terms of congestion control when a new event happens. The event is selected and sent to the program manually by users. If a program receives an event, it will change the value of relevant parameters and show the next state of congestion control. The model in this task is designed based on the FSM description in section 2.5.5.

3.1.2 Simulation Task 2: Simulating a Single Link without Router

After finishing the first simulation task, users have gained a basic understanding of congestion control mechanism. The next procedure is to put the congestion control block into a network and study its behaviour. It has been introduced in chapter 1, the simulator in this project is expected to be able to specify a network with associated parameters. Simulation Task 2 focuses on the simplest network. This is a good transition before more complex networks are researched. The network topology of this task is shown in figure 3.1. In this network, packets can only be discarded manually by users and there is no other reason leads to packets loss. The purpose to do this task is to observe the behaviour of congestion control window when a single packet is lost in one window.

sender
receiver

Figure 3.1 Network Topology of Task 1

As discussed in section 2.5.2, in general case, a retransmission occurs when too many packets lost in one window. However, multiple packets loss in a window is not the concern of this task. Therefore, in order to be able to observe the behaviour of the congestion window after a timeout event occurs. The simulator was designed to focus on two cases. The first one is the retransmission of lost packets is caused by timeout. The second one is the retransmission of lost packets can be caused by either timeout or triple duplicate ACK.

In the first case, although a triple duplicate ACK has indicated a packet loss, the retransmission does not happen. So, the retransmission timer keeps running until a timeout event occurs. In this way, the behaviour of TCP for timeout can be observed.

3.1.3 Simulation Task 3: One Sender, One receiver and a Router

Simulation Task 2 focuses on studying the TCP behaviour of a very simple network. The sender and receiver are connected by a single link directly. However, this topology is only suitable for small scale networks. According to section 2.2.3, switching nodes are needed when machines in WANs wish to communicate with each other. Therefore, Simulation Task 3 adds a switching node on the link. The difference between networks in Task 2 and Task 3 is that there is a router on the link of Task 3. The topology of the network in Task 3 is shown in figure 3.2. All packets sent by the sender are transmitted into the router first. As the router buffer of this router is finite, any packets arrive when the router buffer is full will be discarded. So, packets loss may happen in this task even if users do not discard packets manually. On the other hand, the inport and outport on the router have different throughput and can result in the performance of TCP in this task being too.

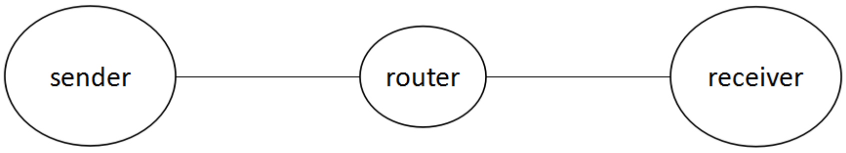


Figure 3.2 Network topology of Task 3

3.2 Assumptions

A real TCP implementation is very complex. In order to simplify the simulation and focus on the observation of the performances of TCP, the following assumptions[2] are made for this project. Also, the reasons for making these assumptions is given.

Connections

As stated in section 1.5, this project is not concerned with the procedure for connection establishment. In some cases, however, this procedure needs to be executed repeatedly, which leads to a large performance difference between real TCPs and this simulator. Therefore, the project is expected to be concerned with long-lived connections only which mean the sender always has enough packets to transmit. In the simulator, users are required to set the total number of packets to be sent in a connection. The larger the number of packets, the longer the life of the connection.

Sequence Number and Data Length

The sequence number of the first segment is assumed to be 1 and the data length of every segment is 1 byte. As introduced in section 2.5.3, the sequence number for a packet equals to

[2] Some of the assumptions are made with taking assumptions in [19] and [21] as references.

the byte-steam number of its first byte. Therefore, under this assumption, the i^{th} packet has a sequence number i, so it is easy to create all the packets to be sent before the transmission starts and easy to observe which packet is lost. Also, as the segment size of all packets equals to 1, the maximum segment size (MSS) is assumed to be 1 in the project.

Round Trip Time and Timeout Interval

The round trip time (RTT) is assumed to be a constant and independent of window size in the project. A similar assumption is made in [19]. The retransmission timeout interval is set to a value equals 2*RTT. Actually, in a complicate TCP, RTT varies with time and is sampled in every round [3, p.265]. However, the mechanism of RTT sampling is complex. Therefore, assuming RTT to be constant can simplify the simulator. Based on this assumption, the retransmission timeout interval can be assumed to be 2*RTT. According to the introduction in section 2.5.2, this interval is calculated according to estimated RTT and sample RTT in every round. A detail introduction of the method for calculation is given in [3, p.264 - p.267]. As RTT is assumed to be constant, the retransmission timeout intervals in different rounds are not different as well.

ACK packets

In the normal case, ACK packets are just consisted by headers and footers and carry no actual data to be sent [21]. Therefore, the data length of ACK packets is with short. Compared with data packets from sender, the transmission time of ACK packets is very short and is assumed to be negligible. Besides, for simplicity, we assume that the router can only discard data packets. To satisfy this requirement, the router buffer is deemed not to be occupied by ACK packets. A similar assumption is introduced in [21].

In summary, all the assumptions above are made for the purpose of simplicity. By this way, the program is easy to understand and the result is simple to be evaluated. Although the simulator built based on these assumptions is different with real TCP in terms of some mechanisms, their performance is similar. Evaluating the results produced by the simulator is helpful for handling the theories of TCP.

3.3 Model Architecture

As shown in Figure 3.3, the simulation model is divided into five blocks. Each block has several methods to simulate data transmission and congestion control. Through communications among these blocks, a single link network with one sender, one receiver and one router is simulated. When the program starts, these blocks can work synchronously. Besides, there are also other functions for creating GUI, setting parameters, sampling parameters and plotting figures. They will be introduced in the implementation phase.

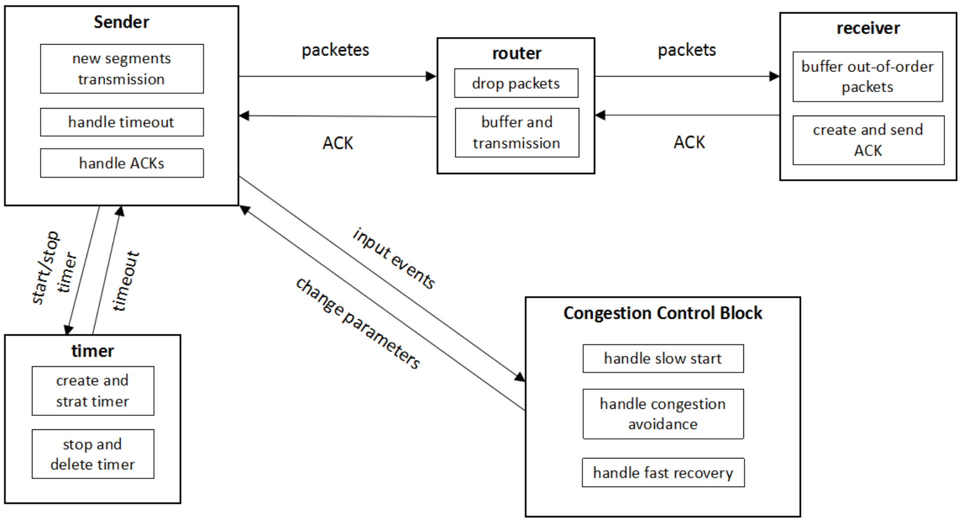


Figure 3.3 Model Architecture

3.4 Summary

The concept for designing the TCP protocol simulator has been covered in this chapter. In section 3.1, three simulation tasks were introduced. The first task focused on demonstrating the congestion control mechanism while the other two tasks were concerned with the simulation of network topologies. Section 3.2 describes the assumptions before simulation. These assumptions are made for simplicity. They have been used in other TCP implementations and proved would not highly influence the simulation. At last, the architecture of the model is presented.

Chapter 4

Implementation

This chapter describes a detailed process of implementing of the simulators. First, the environment of the implementation is explained. Then, the implementations of different blocks of the model are introduced. The way for blocks to communicate with each other is also included in this part. It is followed by an explanation of the GUI for the users to set parameters, control transmissions and observe the results.

4.1 Environment

For the implementation of the algorithms in this project, MATLAB is preferred because of its high integration and convenience. The following reasons make MATLAB an appropriate tool for building the simulator:

- MATLAB is widely used in engineering and all students in Communication Engineering are expected to be familiar with it.
- The simulator needs to plot figures to show the performance of TCP. The built-in plot function can produce these figures easily.
- MATLAB also contains built-in functionality for creating GUI which is convenient and simple.

In particular, the project was implemented in the MATLAB R2016a edition.

4.2 Implementation of the blocks in the model

In this section, the implementation of the blocks contained in the model introduced in section 3.3 is discussed. All of these blocks consist of by a set of functions. The communication between blocks can be achieved in two ways. The first is to call the specific function in another block. The second is to use the notify statement in MATLAB. This method will be used in the TCP receiver and timer.

4.1.1 TCP congestion control block

TCP congestion control is an important part of this simulation. Related mechanisms are introduced in section 2.5.5. This should be designed to take different actions according to different events. This block is expected to accept and handle the following three events:

- **new ACK**: A new acknowledgement segment is received
- **duplicate ACK**: A duplicate acknowledgement is received
- **timeout**: Retransmission Timer expires

Two additional events must also be handled by this block. They are the arrival of a third duplicate acknowledgement and the size of the congestion window exceeding the slow start threshold. These 2 events differ from the 3 above in that they are not signalled by other blocks. Every time the functions to handle "duplicate ACK" and "new ACK" is executed, they check the current number of acknowledgement segments and the congestion control window size respectively. Once the conditions are satisfied, associated response actions will be taken immediately. The following table explains the algorithm in detail.

Pseudo Code 1: Response of triple duplicate ACK and cwnd >= ssthresh

```
1: if dupACKcount = 3
2:     sstresh = cwnd/2
3:     cwnd = ssthresh+3
4:     current_state = Fast recovery
5:     retransmit the lost packet
6: end if
7: if cwnd >= ssthresh
8: current_state = congestion avoidance
9: end if
```

According to the FSM description in section 2.5.5, the TCP congestion control block has the following three states:

- **Slow start**: TCP enters to this state at the beginning of transmission and after "timeout" events happen.
- **Congestion control**: There are two possible cases that can make TCP enter this state. The first is "cwnd >= ssthresh" happens when TCP is in slow start state. The second is when a "new ACK" happens when TCP is in the fast recovery state.

- **Fast recovery**: Every time the “triple duplicate ACK” occurs, the TCP enters the fast recovery state immediately.

To handle the input event when TCP is in these states respectively, three sub-functions `handle_SS`, `handle_CA` and `handle_FR` will be called by the congestion control main function. These sub-functions provide methods to handle the event and retransmit packets when a timeout or triple duplicate ACK occurs

Every time an event is indicated by other blocks, the TCP congestion control block checks the current state and chooses a certain sub-function to handle this event. The sub-function is used to change the values of associated parameters including congestion window size, slow start threshold and the number of duplicates. Moreover, the function changes the current state of TCP at the same time.

4.1.2 TCP Sender

The functions of the TCP sender includes receiving events, starting/stopping the timer, transmitting/retransmitting segments, handling timeout and handling received acknowledgement segments. Additionally, the action of inputting events to TCP congestion control block is also taken by TCP sender. Specifically, the TCP sender contains three methods including new segments transmission, handling timeouts and handling ACKs.

New segments transmission

The main task of this method is to transmit new segments if there is available room in the congestion window. TCP uses a variable `NextSeqNum` to represent the sequence number of the next segment to be sent. If it has been checked that a transmission is allowed, this block will send a new segment to the router. This procedure will be repeated until a transmission is not permitted by the congestion control window or there are no new packets to be transmitted. When a new segment is transmitted, this block will also check whether the retransmission timer is running. If the timer is not running, it will be started.

The transmission can be interrupted by the user as well. Once a command to stop transmission is given, new packets cannot be transmitted even if transmission is permitted by congestion control. This order will also stop the retransmission timer if it is running. This is done by using a global variable `Stopbutton`. The default value of `Stopbutton` is zero. A user can change the value of `Stopbutton` to 1 to stop the transmission. Every time before sender transmitting a packet, it will check the value of `Stopbutton` first. New transmissions are only permitted when this value is zero.

Handle Timeout

This method is used to handle "timeout" events. When a timeout occurs, this function will retransmit a not-yet-acknowledged segment with the smallest sequence number immediately and start a new timer. At this stage, a timeout event is indicated to the TCP congestion control block as well.

Handle ACKs

When acknowledgement segments are received by the TCP sender, they are handled in the ACK handling block. This block accepts acknowledge segments with ACK field value y. TCP maintains a variable `sendbase` to record the sequence number of the first transmitted but unacknowledged segment. Every time an ACK is received, its ACK field value `y` is compared with `sendbase`. If `y` is larger than this variable, which means this ACK is a new ACK, a "new ACK" event will happen and the new segment transmission method is called again to start transmitting packets in a new round. Otherwise, this ACK is a duplicate ACK and a "duplicate ACK" event happens.

The "new ACK" and "duplicate ACK" events are also indicated to the TCP congestion control block. Then the action of transmission/retransmission is taken. When a "new ACK" event happens, the new segment transmission method is called again to start the transmission in a new round. When a "duplicate ACK" event is accepted by the congestion control block, different methods are used according to different value of the current state. In the case where the current state is slow start or congestion avoidance, if the duplicate ACK count reaches three, a segment with sequence number of y will be retransmitted. A different method is used when the current state is fast recovery. In this case, a new packet is transmitted every round until a “new ACK” event is indicated.

4.1.3 TCP router

TCP router receives the packets sent by senders. Before starting the program, a value `routerBuffer` is set by the user to assign the capacity of the router buffer. The structure of the router is shown in figure 4.1. If a packet arrives when the router is empty, it will be moved to the outport immediately for transmission. Then, the second arriving packet will be stored in the buffer waiting for transmission. So, the maximum number of packets can be in the router is a number of `routerBuffer` packets in the buffer plus one packet being transmitted. If a packet in the outport has been transmitted, the next packet to transmit is the oldest one received by the router.

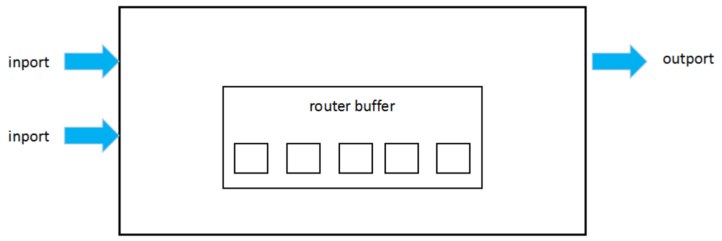


Figure 4.1 Router Structure

When the router buffer is full, any additional packets that arrive are discarded, which is known drop-tail policy. There are also other queue management policies used by TCP. Detailed research on both the drop-tail policy and other queue management policies is given in [21]. In this dissertation, only the drop-tail policy is considered.

4.1.4 TCP Receiver

The TCP receiver checks the sequence number of the arriving packets to determine whether they are in order. This is done by comparing the sequence number of the incoming segment with a value called `rcv_base`. The function of `rcv_base` is to store the sequence number which is expected to be received next. If the sequence number of the incoming segment is equal to `rcv_base`, the receiver will produce an acknowledgement packet to indicate that a new in-order packet is received. Otherwise, the receiver just sends an acknowledgement for the last packet it received correctly.

Pseudo Code 2: Algorithm for buffering out-of-order packets in receiver

```
if SeqNum = rcv_base /* the lost packet is received*/
   if rcvBuffer is not empty and rcvBuffer[1] = rcv_base+1
      while rcvBuffer contains more than one packet
         if rcvBuffer[1] != rcvBuffer[2]
/*There is a packet loss between the first two packets in buffer*/
            Break
         end if
         remove rcvBuffer[1]
      end while
      ACKbase = rcvBuffer[1]
/*acknowledge received in order packet with largest SeqNum*/
      remove rcvBuffer[1]
   end if
end if
create an acknowledgement packet with ACK field number y = ACKbase
```

To avoid the retransmission of packets received already, the TCP receiver buffers received but out-of-order packets when there is a packet loss. If the packet loss is repaired, a new ACK will acknowledge all the arrivals of these out-of-order packets. An array, `rcvBuffer`, is used to collect the sequence numbers. When the retransmitted lost packet arrives the receiver changes the value of `rcv_base` to the sequence number of the newest packet. The detailed algorithm is given in Pseudo Code 2.

This algorithm uses a loop to check whether there is a packets loss. This is achieved by checking if there is a sequence number gap in the packets array in the buffer. The existence of a sequence number gap indicates a packet loss. After the checking procedure, the program sends an ACK to acknowledge the arrival of all the packets before the packet loss.

4.1.5 Timer[3]

The timer block is used to simulate the retransmission timer in TCP. When the sender sends a start timer command to the timer block, the creation and start timer method is executed. A new timer is created and starts running. After a retransmission timeout interval, this timer

[3] The timer here refers to TCP retransmission timer. Actually, there are other kinds of timers used to sample congestion control window size and to simulate round trip time. These timers are introduced in the implementation phase.

indicates a timeout event to the sender. The sender can also give a stop timer command to the timer block before a timeout event happens. In this case, the timer block can find the retransmission timer then stop and delete it. As a result, the timeout would not be indicated to the sender.

Specifically, the implementation of timer is done using the MATLAB built-in class `timer`. When an instance of the `timer` class is created, a callback function is required. When the timer starts, this callback function is executed. The callback function in this simulator is used to indicate a timeout event to the sender. Timer objects also have a property, `StartDelay`, to specify the delay between the start of the timer and the first execution of the callback function. The value of `StartDelay` is set to the retransmission timeout interval. On the other hand, if a stop timer command is given before the execution of callback function, this timer is stopped and the timeout event will not be indicated to the sender. The creation and deletion of a timer is done by the following two methods in Table 1.

Method	Description
`start_timer`	First, create a timer object with a callback function to indicate the timeout event to the sender and a start delay of retransmission timeout interval. Second, start this timer.
`stop_timer`	Find out the running timer object created by `start_timer` then stop and delete this object.

Table 4.1 Methods for creation and deletion of a timer

4.2 Implementation of Task 1

The program of Task 1 (Section 3.1.1) only contains the TCP congestion control block as no actual network topology is specified in this task. Users can set the relevant values including `cwnd`, `ssthresh`, `dupACKcount` and `current_state`. Then, an event can be chosen and input into the congestion control function. The relevant values are changed by this function and displayed to the users by the GUI.

4.3 Implementation of Task 2

In Task 2 (section 3.1.2), four blocks including sender, receiver, timer and congestion control block are used. According to the topology given in Chapter 3, the sender sends packets to the receiver directly. The following figure shows the flow chart of Task 2.

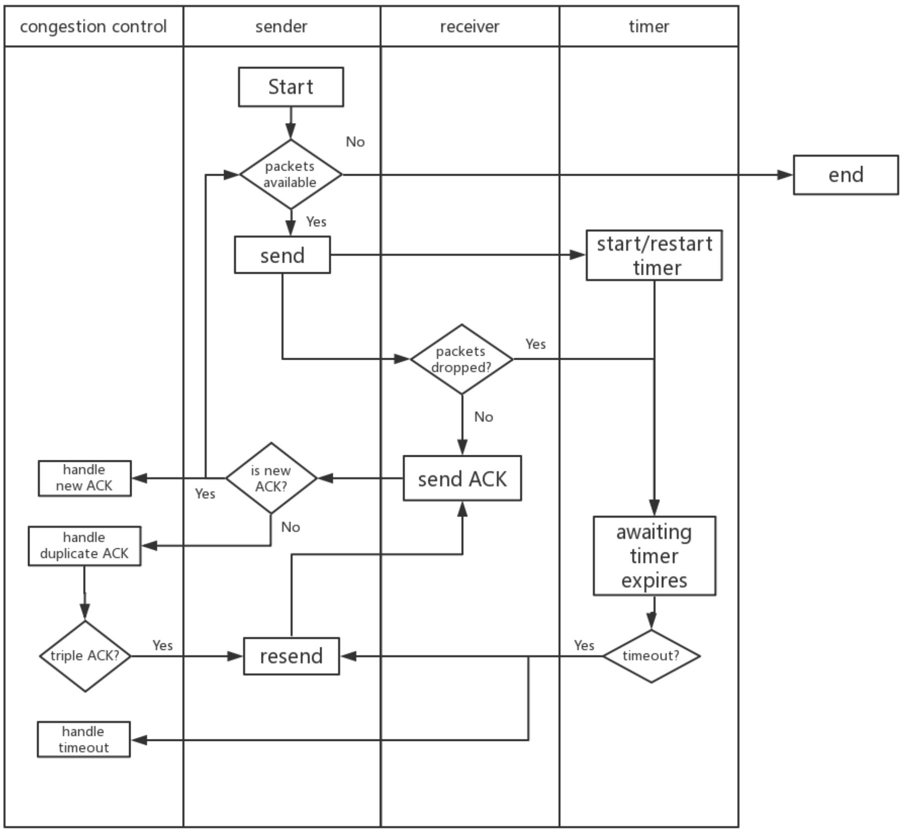


Figure 4.2 Flow chart of Task 2

It can be seen from this figure that the program of Task 2 executes iteratively. Every time the execution of a function finishes, one or more functions are called and proceed to the next procedure. All these functions in different blocks form a closed loop. Once the program has been started, the iterations will continue until all packets have been sent out and acknowledged or a stop command is sent.

As introduced in section 3.1.2, this task is divided into two parts according to whether use the fast recovery mechanism. The congestion control block uses a global variable `add_FR` to determine whether to use fast recovery mechanism or not. If the user chooses to use the fast recovery mechanism, the value of `add_FR` is set to 1. Handle methods use conditional statements to check this value and determine whether to execute the algorithms for fast recovery.

Users should decide which packet is to be dropped before starting transmission. If a packet has been dropped, its sequence number is set to zero. When the send method sends a packet with a zero sequence number, it will be ignored instead of sent to the receiver.

4.4 Implementation of Task 3

Compared with the simulator in task 2, there is a router between sender and receiver in Task 3 (Section 3.1.3). After being sent by the sender, the packets first arrive at router. The router block in this task uses an array `routerBuffer` to buffer packets. The first packet arrives is placed in `routerBuffer[1]`and the second is in `routerBuffer[2]`, and so on. The size of buffer `routerBuffersize` is determined by users before transmission. When the length of `routerBuffer` reaches `routerBuffersize`, the router buffer is full. Any packets arrive when router buffer is full will not be transmitted to the receiver, which leads to a packet loss. The transmission rates of the two ports on router are different. The situation of an overflow buffer can only occur when the outgoing link transmission rate is lower than that of inputting link.

For example, there are 12 packets arriving the router while two packets are transmitted and five packets are discarded in a round. The simulator drops five packets randomly. It is certain that the first six packets would not be dropped as they do not exceed the maximum number of packets can be in the router. Therefore, this simulator discards five packets randomly from those packets with sequence number of 7 to 12.

4.5 Implementation of Graphical User Interface (GUI)

A graphical user interface was created for the users to execute the program. The GUI that was implemented contains two pages. The first page (Figure 4.5) is used for excusing and studying Task 1. The second page is responsible for both Task 2 and Task 3.

The first page contains two pop-up menus to choose the current state of TCP and a specific event to be input. There are also three edit text boxes for inputting the values of cwnd, ssthresh, dupACKcount respectively. After handling an event, the new values of these three parameters are also shown in these edit text boxes. A static text box is created to display the current state of this simulator. The default value of the TCP state is slow start, so the static text box displays “slow start” at beginning.

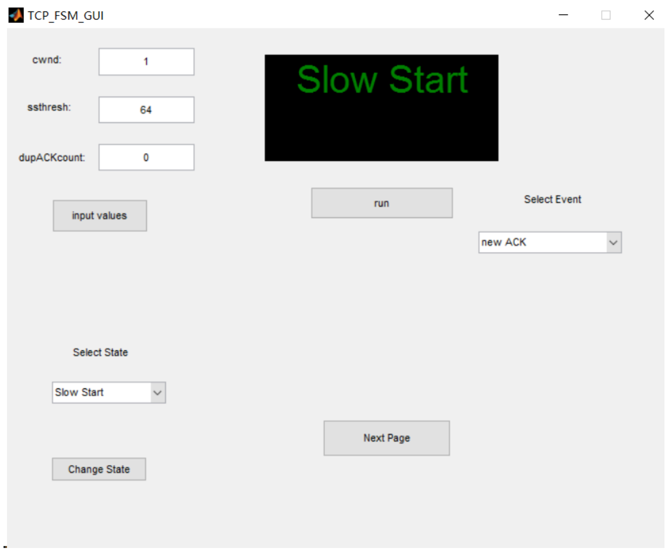


Figure 4.3 The first page of GUI

Besides the pop-up menus and buttons, there are also four buttons on the first page for different purposes.

- **Input values**: The function of this button is to read the input values in edit text boxes and assign these values to the corresponding variables in TCP blocks. It is worth mentioning that the values in the edit boxes are saved as characters and they need to be mapped to numbers before being assigned.

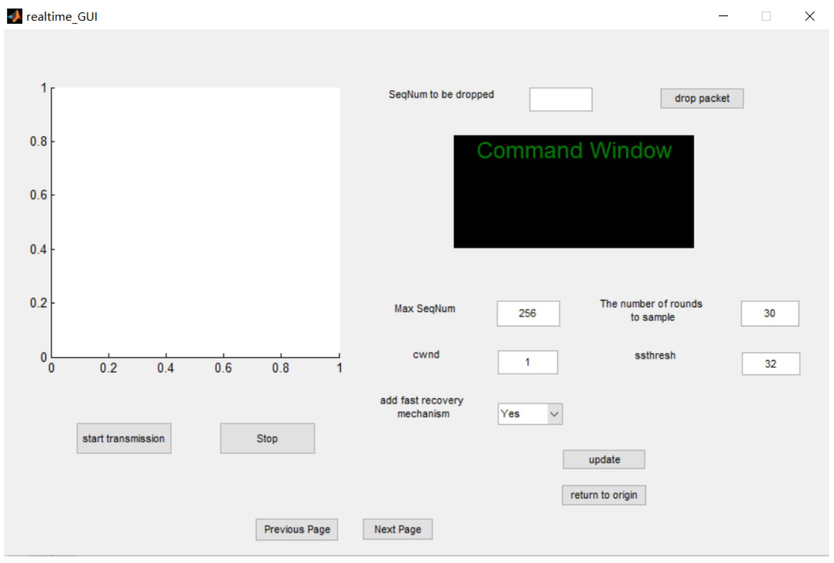
- **Change state:** This button is used to change the value of `current_state` according to the state in the pop-up menu above it.

- **Run:** This button inputs the value chosen by the select event pop-up menu to the congestion control function and changes the state being shown in the static text box. Before pressing this button, all other parameters in edit text boxes and pop-up menus should have been set.

- **Next Page:** After finishing Task 1, the user has obtained a basic understanding of TCP congestion control mechanism. By pressing Next Page button, the GUI moves forward to the second page to perform the simulation of Task 2 and Task 3.

The second page (shown in figure 4.4) requires setting the parameters to perform a complete transmission. As mentioned in section 4.3, the program uses a variable `add_FR` to determine whether to add a fast recovery mechanism. The checkbox of "add fast recovery mechanism" enables users to change the value of `add_FR`. The four edit text boxes are created for inputting associated values. The "Max SeqNum" edit text box accepts the maximum sequence number. Users can use this number to decide how many packets are going to be transmitted.

The program uses a sample timer to sample the value of cwnd every round for plotting the result figure. "The number of rounds to sample" text box requires an integer number. The *`TasksToExecute`* property of the sample timer is set equals to this number. Therefore, the number of rounds to sample is determined.



Max SeqNum 256
The number of rounds to sample 30
cwnd 1
ssthresh 32
add fast recovery mechanism Yes

Figure 4.4 The second page of GUI

There is also a text box responsible for dropping packets. The sequence number to be dropped must be entered in this box and the packet with this sequence number is dropped using the method mentioned in section 4.3. After inputting the sequence number to be dropped, the user should press the drop packet button to perform the command of dropping packets. The command window also prompts that a packet has been dropped, as seen in Figure 4.5 (a). If there are multiple packets that need to be dropped, users should input sequence numbers and press the button for every packet. It is important to note that, dropping a packet can only be done when the packets have been created and the sequence to be dropped must be made up of existing packets. For example, if a user drops a packet with sequence number of 400 when a Max SeqNum of 256 is updated, an error will occur, as seen in Figure 4.5 (b).

packet has been dropped

(a) a packet is dropped successfully

illegal operation!

(b) drop an out-of-range packet

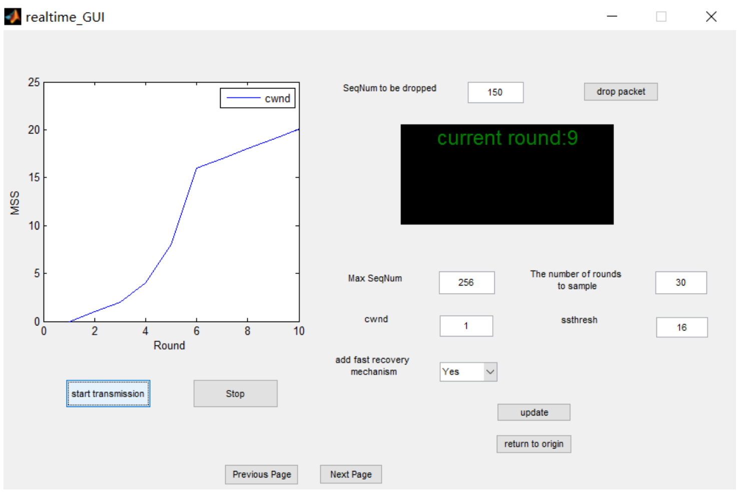
Figure 4.5 Command window for dropping a packet

There are also other buttons created in order to control transmissions and handle parameters. Detail introductions of these buttons are given below.

- **Update:** This button is used when all parameters above it have been input already. These parameters are assigned to relevant variables in the program. This process is necessary before starting a transmission.

- **Start transmission:** After updating all associated parameters and dropping some packets, users can start the transmission by pressing this button. A sample timer is created according to the updated value of “rounds to sample”. The timer function is to sample the value of cwnd and plot a figure of *cwnd vs round* on the GUI (Figure 4.6). The command window displays current round number.

- **Stop:** As introduced in section 4.1.2, the simulator allows users to interrupt the transmission. If a user presses this button, the program will stop all timers and prohibit the transmission of any new packets using the method in section 4.1.2. Once the transmission has been stopped, it cannot be continued from where it was stopped because all timers are stopped and deleted. Users can only update the parameters again and start a new transmission.



SeqNum to be dropped: 150

current round:9

Max SeqNum: 256

The number of rounds to sample: 30

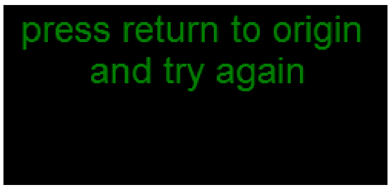
cwnd: 1

ssthresh: 16

add fast recovery mechanism: Yes

Figure 4.6 The screen shot of GUI in round 9

- **Return to origin:** Besides the parameters required by the GUI, there are also many other parameters in the program. The value of these parameters is changed during transmissions. If a user wants to start a new transmission after an old transmission has finished or has been cancelled by the stop button, s/he must press the return to origin button to restore default values. This button also erases the figure on the GUI. The command window will give a prompt if users press the update button without returning to origin, as shown in figure 4.7.



press return to origin and try again

Figure 4.7 Pressing the update button without returning to origin

There is a checkbox named “add router” with two edit text boxes near it. These two edit text boxes require the size of router buffer and the transmission rate of the outgoing port. They are only active when users choose to add router. Similarly with other edit text boxes, pressing the update button is needed to assign the values to the relevant variables.

4.6 Summary

This chapter has covered the implementation of the TCP simulator. Before explaining implementation details, the implementation environment was introduced in the first section.

Section 4.1 discusses the reason for using MATLAB for the implementation of the simulator. Then, the implementation details of the main blocks in the model architecture was proposed in section 4.2. This was followed by the methods to simulate the tasks introduced in Chapter 3. The final section discusses the implementation of GUI. The functions of text boxes, buttons and pop-up menus are introduced in this section. The next part of this project is to test and evaluate the design and implementation, which was covered in Chapter 3 and Chapter 4.

Chapter 5

Testing and Evaluation

In Chapter 3 and Chapter 4, four simulation tasks were designed and implemented. A two-page Graphical User Interface (GUI) was created for users to run these simulations. This chapter covers both testing and evaluation of the simulator. In the first part, the simulator is tested completely. The results of the four simulation tasks are compared with corresponding results in other references and related works reviewed in chapter 2. The second part concerns evaluation. At the beginning of this section, evaluation metrics for the simulator are proposed. Then, a detailed evaluation is made according to the evaluation metrics.

5.1 Test of Simulation Task 1

In this section, Simulation Task 1 is tested by inputting different values. The corresponding results are recorded and compared with expected results obtained according to the congestion control algorithms introduced in section 2.5.5. The result of testing Simulation Task 1 is described in Table 5.1. This table contains two fields including input values and test results. There is a column in the test results to check whether the results match expected value.

	Input values					Test results				
No.	dup ACK	state	ss thresh	cwnd	event	dup ACK	state	ss thresh	cwnd	Matched with expected values
1	0	SS	8	1	NACK	0	SS	8	2	YES
2	0	SS	8	2	NACK	0	SS	8	3	YES
3	0	SS	8	3	DACK	1	SS	8	3	YES
4	2	SS	8	3	DACK	3	FR	1.5	4.5	YES
5	0	SS	8	7	NACK	0	CA	8	8	YES
6	0	CA	8	8	NACK	0	CA	8	8.125	YES
7	0	CA	8	8.125	NACK	0	CA	8	8.25	YES
8	0	CA	8	8.25	TM	0	SS	4.125	1	YES
9	2	CA	8	8.25	DACK	3	FR	4.125	7.125	YES
10	3	FR	4	7	DACK	3	FR	4	8	YES
11	3	FR	4	8	NACK	0	CA	4	4	YES

Table 5.1 Simulation Task 1 Testing Summary[4]

[4] In the event column in this table, NACK, DACK and TM represent new ACK, duplicate ACK and timeout respectively while SS, CA and FR stand for slow start, congestion avoidance and fast recovery.

All the results in Table 5.1 were read from the first page of the GUI.There are 11 sets of input values in this table. It can be seen that all test results are matched with their corresponding expected values. In conclusion, Simulation Task 1 was implemented successfully and behaved as expected.

5.2 Test of Simulation Task 2

In this section, the behaviour of Simulation Task 2 is tested. As introduced in section 3.1.2, Simulation Task 2 is divided into two cases. Therefore, the testing of this task is also divided into two parts in this section. The case of using the fast retransmit mechanism (Case 1) was tested first. Then, the simulator without fast retransmit mechanism (Case 2) was tested.

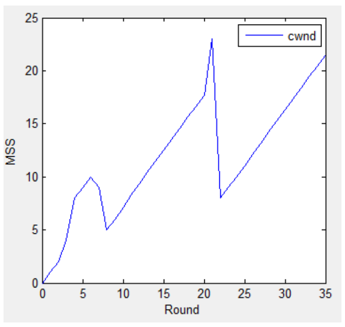
5.2.1 Test of Case 1

The parameters are set as the following table:

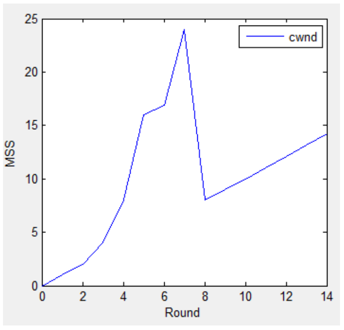
No.	cwnd	Max SeqNum	ssthresh	Sample rounds	Dropped packets
1	1	1024	8	35	30 and 180
2	1	1024	16	50	30, 180 and 360

Table 5.2 Test parameters of Case 1

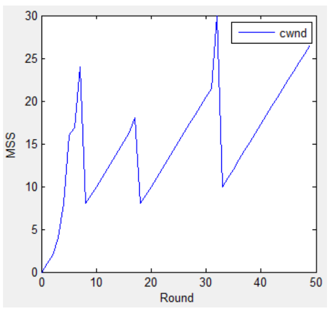
After finishing initialization, the simulation is started by pressing the *start transmission* button. The plotting is updated automatically every round. In Figure 5.1 (a) and (b) the test results of both of the two sets of data are shown. Besides, there is also a plot of the second set of data with only first ten rounds for a more detailed view, as shown in Figure 5.1 (c).



(a) The first set of data



(b) The second set of data



(c) A magnified view

Figure 5.1 Simulation Task 2 test results (Case 1)

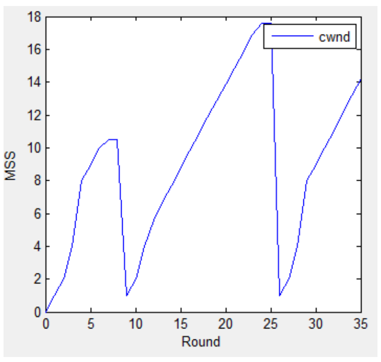
In Figure 5.1 (a), the value of congestion control window is 1 in the first round. The sender is in slow start in the first three rounds. The value of cwnd increases exponentially and the value reaches 8. Then, the state changes to congestion avoidance and cwnd increments every round. In round six, packet loss is detected by triple duplicate ACKs and the state is changed to fast recovery and the value of cwnd decreases. This lost packet is the first packet dropped in GUI whose sequence number is 30. It is dropped from senders transmission in the fifth round but the third duplicate ACK is received in the sixth round. In the seventh round, a new ACK arrives. According to the algorithms introduced in section 2.5.5, the value of cwnd is set to be equal to ssthresh which was updated in round six. Then, the sender keeps staying in congestion avoidance state until round 20 where the second packet loss is detected. In round 20, the state turns to fast recovery and the value of cwnd increments one every time a duplicate ACK is received. This behaviour also accords with the description in section 2.5.5. In round 21, a new ACK arrives and the state is changed to congestion avoidance again.

The behaviour of the second set of data is similar to the first set. Because there are three packets dropped before starting the transmission. The value of cwnd decrease three times as shown in Figure 5.2 (b). For the magnified view in Figure 5.2 (c), it can be seen that the value of cwnd keeps increasing exponentially until reaches 16. Then the sender stays in congestion avoidance until a triple ACK occurs.

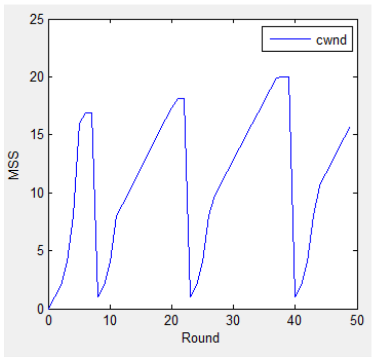
After analyzing the test result, it can be found that the behaviour of this simulator meets the algorithms introduced in the background knowledge in chapter 2. Besides, a comparison of behaviour in Simulation Task 2 and in related works is made. From analyzing the results in several related works and materials, it is found that there are two versions of TCP implementations. In the first version, the congestion window experiences inflation and deflation in the fast recovery states. TCP Reno behaviour in [22] and the behaviour of FreeBSD in [20] (the figures of [20] and [22] is cited and given in chapter 2 of this dissertation) belong to such kind of implementations. For the second version, the congestion window would not inflate and deflate in the fast recovery. The figure in [3, p.303] and NS-2 behaviour in [20] are two example implementations of the second version. As mentioned in section 2.6, the author believes that the difference between these two versions is caused by different methods of implementations. From comparison, the simulator of Simulation Task 2 is found to be similar with the first version implementation. However, the rapid inflation in the fast recovery state cannot be sampled and observed because as the value of cwnd in the simulator in this project is sampled once a round.

5.2.2 Test of Case 2

In order to test Case 2, the selected *add fast retransmit mechanism* checkbox is cancelled. The inputting parameter is same with test of Case 1. After two times simulations, the following two result figures are got.



(a) Test result of the first set of data



(b) Test result of the second set of data

Figure 5.2 Simulation Task 2 test results (Case 2)

As shown in Figure 5.2 (a), the behaviour in the first six rounds in Case 2 is same as Case 1. Also, a packet is lost in round 5. However, as there is no fast transmit mechanism, the sender needs to wait for a *timeout* event. The packet loss is detected by *timeout* in the seventh round and the retransmission starts in the eighth round. After *timeout* occurs, the state is changed to slow start the sstresh is set equals to 4. The value of cwnd increases exponentially again until it reaches 4. Another *timeout* event occurs in round 25 which is caused by the loss of No. 180 packet. The behaviour of this simulation accords with the algorithm described in section 2.5.5. The behaviour in Figure 5.2 (b) is similar to Figure 5.2 (a). In Figure 5.2 (b), *timeout* event occurs three times because three packets are dropped before starting transmission. The results of the simulations for Case 2 are also compared with corresponding TCP behaviour in related works. The behaviour of this simulator is similar with [3, p.302] and the TCP Tahoe in [20].

5.3 Test of simulation Task 3

In this section, five groups of data are simulated. The first group is selected to be the control group. Barring sample rounds, the other four groups only have one field different with it. A detailed analysis is given in the following part of this section. As mentioned in 3.2, the round trip time (RTT) in this project is assumed to be constant. Because the transmission rate of the sender equals to cwnd/RTT, the transmission rate is only related to the value of cwnd. In this section, we use the value of cwnd to describe the transmission rate.

No.	cwnd	Max SeqNum	ssthresh	Sample rounds	Dropped packets	routerBuffer size	Outport Rate
1	1	1024	32	100	none	6	15
2	1	1024	16	80	none	6	15
3	1	1024	32	50	none	6	30
4	1	1024	32	80	none	12	15
5	1	1024	32	70	20	6	15

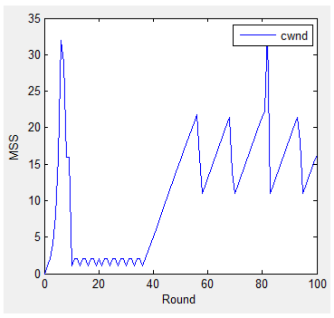
Table 5.3 Simulation Task 3 testing parameters

In Figure 5.3 (a), the transmission rate increases rapidly in the first five rounds. Then, the transmission rate is so high that a large number of packets is lost in round five which leads to a serious congestion. After that, the sender can only retransmit one packet every round and no new packet is allowed to be transmitted because there are too many sent out but unacknowledged packets. This continues until round 38 when the packets losses are repaired. Then, the transmission rate increases again and fluctuates in a within a relatively stable range around 15 until the transmission is finished. The whole process costs 100 RTTs.

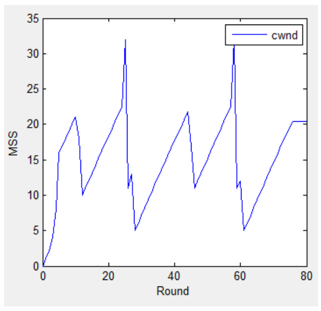
The simulation of group 2 is done next. The difference between group 1 and group 2 is that the initial value of sstresh of group 2 is lower than group 1. It can be seen from Figure 5.3 (b) that, the sender enters congestion avoidance before packets loss happens. A packet loss is indicated by *triple ACK* in round 9 and the sender enters fast recovery state. The congestion is not so serious because the transmission is low. It takes 76 RTTs to finish this transmission which is faster than group 1.

The third group has a larger router outport rate compared with group 1. Compared with group

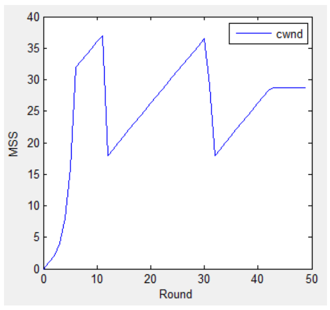
1 the transmission rate reaches a higher value before packets are dropped. However, such a high transmission rate does not result in a heavy congestion like group one. This is because the outport rate is higher and more packets can leave the router in one round. The efficiency of this group improves a lot and it takes only 42 RTTs to finish this transmission.



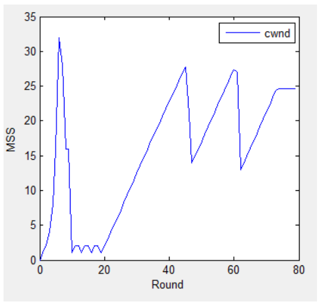
(a)



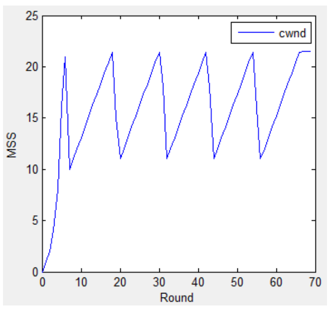
(b)



(c)



(d)



(e)

Figure 5.3 Test results of Simulation Task 3

The router buffer size in group 4 is doubled so that more packets can be buffered in one round. As shown in Figure 5.3 (d), the performance of group 4 in first 20 rounds is similar with group 1. However, the packet loss in group 4 is required faster comparing with group 1. This is because the router of group 4 can buffer more packets and a smaller number of packets are dropped. Finally, this transmission costs 70 RTTs.

In group 5, all properties of the network are same as group 1. The difference is that the packet, whose sequence number is 20, is dropped before transmission starts. This packet loss is detected in the fifth round. Then, the sender enters fast recovery state and the transmission rate increases to about 10/RTT. In the following rounds, the transmission rate floats up and down near 15 until the transmission is finished in round 65. The interesting thing is that the efficiency of the simulator is not reduced when a packet is dropped manually. On the contrary, the transmission is finished faster. In practical application, a queue management method called Random Early Detection (RED) is designed according to this phenomenon. A router with RED mechanism would drop a packet even there are some vacant memory slots. The probability to drop packets increases with the increase of the average queue size [23]. This method seems inefficient but it improves the performance of TCP significantly.

It can be found from the results of the 5 groups that, the transmission rate tends to be stable within a range around outport rate after the initial adjustment. Compared with the control group, all other study groups have a better performance. According to the result, the methods to avoid congestions on the router can be summarized. First, increase the buffer size and outgoing port rate of the buffer to avoid discarding too many packets when the incoming transmission rate is high. Second, set a reasonable slow start threshold to avoid the transmission rate of the sender increasing too fast. Finally, use queue management methods to decrease the transmission rate of the senders before congestions happen.

5.4 Evaluation

In this section, the project and simulator are evaluated in multi aspects. The evaluation metrics will be given first in order to evaluate the simulator and the accomplishment of our project. Then, detail evaluations will be given in terms of these metrics.

5.4.1 Evaluation metrics

After the implementation phase and tests of the Simulation Tasks, evaluations should be undertaken to consider the relevance of the simulator and whether it meets the requirements of a TCP simulator to support the students in MSc Communication Engineering. In particular, the criteria are given below:

- **Relevance**: The similarity between this simulator and real TCP implementations is evaluated to consider whether this simulator can show TCP theory correctly.
- **Usability**: This metric mainly concerns whether the simulator is easy to learn and use. Next, whether the results are clear and easy to understand is considered.
- **Effectiveness**: The extent to which the aims and objectives in chapter 1 are

accomplished is estimated. If there are unsuccessful aims, the difficulties are stated.

5.4.2 Relevance

In the above sections of this chapter, test results of all simulation tasks are analyzed and compared with expected values or behaviour of other successful TCP implementations. In particular, the results of Simulation Task 1 are compared with the expected value calculated according to the algorithms in Chapter 2. All test results are matched with expected values. In the test of Simulation Task 2, the behaviour of the congestion window of the simulator is analyzed and it accords with the description of the protocol in Chapter 2. Moreover, the results are also compared with some successful implementations. As explained in section 5.2, the behaviour of this simulator is similar to some implementations in related work. For simulation Task 3, the behaviours of both five groups are explained. According to the analysis, the performance of the simulation results matches the expected performance. In conclusion, the similarity between this simulator and real TCP implementations is high and this simulator demonstrates TCP theory correctly.

5.4.3 Usability

The GUI of this simulator allows users to set parameters and start simulation easily. The command window of the GUI displays clear prompts when users operate the simulator. Moreover, as mentioned in section 4.1, this simulator is developed using a language that is expected to be familiar to all MSc students in the Communication Engineering. Therefore, doing some small redesigns for implementing different functions is a possible pieced laboratory work for the students. In order to evaluate the usability of the simulator objectively, it is necessary to find some other MSc students in Communication Engineering Programme to do simulations and give feedback. However, due to time limitation, this work is not covered in the project and will be suggested in the future work.

5.4.4 Effectiveness

All the objectives of this project have been achieved successfully and a suitable TCP simulator to support MSc students who are taking the Internet and Communications Networks course has been produced. Indeed, the results are believed to be similar to real TCP. Therefore, we consider the aims are achieved and the effectiveness is high.

5.5 Summary

In this chapter, test and evaluation of the project is covered. In the first three sections, tests of the simulation tasks are done and the results are analyzed in detail. In section 5.4 the project is evaluated according to three metrics given previously. The conclusions of these evaluations are stated.

Chapter 6

Conclusion and Future Work

This chapter covers a summary of the project and some suggestions for future work. In particular, section 6.1 draws a conclusion for the project, while section 6.2 proposes some suggestions for future work.

6.1 Conclusion

The aim of this project was to implement a TCP simulator for instructional purposes. To achieve this, objectives were proposed at the beginning of the project which defined the subsequent work. Literature was reviewed and related works were discussed. The background covers knowledge about computer networks, protocol stack and an introduction to TCP. Then, some implementations were reviewed which provided some ideas for the design stage. In the stage of conceptual design, four simulation tasks were proposed in order to meet the aim of this project. The first simulation task focuses on the TCP congestion control behaviour of a single event while the others were designed to simulate particular, simple network topologies. After that, some simplifying assumptions were explained. Some of these assumptions were made by taking some related works as references. In the final part of the conceptual design stage, the architecture of the simulator was designed. There are five main blocks in the architecture while some other functions are also used to finish the simulation.

In order to implement the TCP simulator, MATLAB was chosen to be the language for development as it is familiarized by all MSc Communication Engineering students. The first stage of implementation was to produce the main blocks of the model architecture. Then, the simulation tasks were implemented by combining these blocks to build a network and dealing with the communication between blocks. After finishing these simulation tasks, a two page GUI was created allowing users to create and run simulations. The first page is used to do Simulation Task 1 while the second page is responsible for both Simulation Tasks 2 and 3. In the next stage, the simulator was tested and the results were analyzed. Conclusions were got that all the results were matched with expected results and the simulation tasks were finished successfully.

After the end of the project, it can be claimed that the interactive simulator produced in the project is an appropriate simulator which matches the aim of this project. However, it is not a complete TCP implementation, some functions of a real TCP such as establishing connections and processing data in packets are not covered in this simulator.

6.2 Future Work

The project succeeds in simulating parts of TCP and developing an interactive program. MSc students can use this program to understand TCP theories in their future studies. Nevertheless, due to the limited time of the project, the program only implements limited functions of TCP. Therefore, as a future plan, the following improvements can be done for the simulator.

- Add the mechanism of establishing TCP connections so that short-lived transmissions are able to be simulated.
- Change the RTT value from a constant to a variable and add the mechanism to sample RTT every round then update a new retransmission timeout interval.
- Focus on the action within a round rather than only sampling the parameters at the beginning of a round.
- Simulate a more complex network with multiple senders, multiple routers and multiple receivers and research the behaviour.
- Find some other students on the MSc Communications Engineering programme to evaluate this simulator and record feedback.

Reference

[1] The Linux Information Project. http://www.linfo.org/tcp.html. Accessed: 2017-08-01.

[2] Edwan, Talal A. Improved algorithms for TCP congestion control. Diss. Loughborough University, 2010.

[3] Kurose, James F., and Keith W. Ross. Computer networking: a top-down approach. Vol. 4. Boston, USA: Addison Wesley, 2009.

[4] The Network Simulator - ns-2. https://www.isi.edu/nsnam/ns/. Accessed: 2017-08-01.

[5] ns-3. https://www.nsnam.org/. Accessed: 2017-08-01.

[6] Marsic, Ivan. *"Simple TCP Protocol Simulator."* (2005).

[7] Stallings, William, and Moumita Mitra Manna. *Data and computer communications. Vol. 6.* Englewood Cliffs, NJ: Prentice hall, 2011.

[8] Andrew S. Tanenbaum. *"Computer networks. 3-rd edition."* Upper Saddle River, NJ: Prentice Hall, 1996.

[9] Comer, Douglas E. *Computer networks and internets.* Prentice Hall Press, 2008.

[10] Keiser, Gerd. *Local area networks.* New York: McGraw-Hill, 1989.

[11] Pellegrini, Marco, Enrico Gregori, and Luciano Lenzini. *Metropolitan Area Networks*. Springer Science & Business Media, 2012.

[12] Cowley, John. "Wide Area Networks." *Communications and Networking: An Introduction (2007): 63-78.*

[13] Allman, Mark, Vern Paxson, and Ethan Blanton. TCP congestion control. No. *RFC 5681.* 2009.

[14] Stevens, W. Richard. "TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms." No. *RFC 2001.* 1997.

[15] Paxon, V., M. Allman, and W. Stevens. *" TCP's congestion control."* No.RFC 2581. 1999.

[16] Fall, Kevin, and Sally Floyd. "Simulation-based comparisons of Tahoe, Reno and SACK TCP." *ACM SIGCOMM Computer Communication Review 26.3 (1996): 5-21.*

[17] Floyd, Sally, Andrei Gurtov, and Tom Henderson. *"The NewReno modification to TCP's fast recovery algorithm." (2004).*

[18] Postel, Jon. "Transmission control protocol specification." *RFC 793* (1981).

[19] J. Padhye, V. Firoiu, and D. F. Towsley, “Modeling TCP Reno performance: a simple

model and its empirical validation,” *IEEE/ACM Trans. Networking, vol. 8*, no. 2, pp. 133-145, Apr. 2000.

[20] Zhang, Hui. *A discrete-time model of TCP with Active Queue Management*. Diss. Applied Sciences: School of Engineering Science, 2004.

[21] Athuraliya, Sanjeewa, et al. "REM: Active queue management." *IEEE network 15.3 (2001): 48-53.*

[22] Hasegawa, Go, Masayuki Murata, and Hideo Miyahara. "Fairness and stability of congestion control mechanisms of TCP." *Telecommunication Systems 15.1-2 (2000): 167-184.*

[23] Sally Floyd, Ramakrishna Gummadi, and Scott Shenker, Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management, August 2001.

[24] Blank, Andrew G. TCP/IP Foundations. ISBN: 0-7821-4370-9, 2006.

[25] Microsoft. Lesson 2: How Networks Send Data. https://www.microsoft.com/mspress/books/WW/sampchap/1956b.aspx Accessed: 2017-08-05.

[26] Charles M. Kozierok. Message Formatting: Headers, Payloads and Footers. The TCP/IP Guide.
http://www.tcpipguide.com/free/t_MessageFormattingHeadersPayloadsandFooters.htm
Accessed: 2017-08-06.

[27] Stallings, William. *Business Data Communication (4 ed.)*. Upper Saddle River, New Jersey, USA: Prentice-Hall, 2001.

[28] TechTerms. Internet Terms: Payload Definition. https://techterms.com/definition/payload Accessed: 2017-08-06.

[29] Clark, David D. "Window and acknowledgement strategy in TCP." *RFC 813* .1982.

[30] Fall, Kevin R., and W. Richard Stevens. *TCP/IP Illustrated, Volume 1: The protocols.* Addison-Wesley, 2011.

[31] ExtraHop Networks. TCP RTOs: Retransmission Timeouts & Application Performance Degradation.
https://www.extrahop.com/company/blog/2016/retransmission-timeouts-rtos-application-performance-degradation/. Accessed:2017-08-07.

[32] Jacobson, Van. "Congestion avoidance and control." *ACM SIGCOMM computer communication review. Vol. 18. No. 4. ACM*, 1988.

[33] Lar, Saleem-ullah, Xiaofeng Liao, and Songtao Guo. "Modeling TCP NewReno slow start and congestion-avoidance using simulation approach." *International Journal of Computer Science and Network Security 11.1 (2011): 117-124.*

[34] Mathis, Matthew, et al. "The macroscopic behaviour of the TCP congestion avoidance

algorithm." *ACM SIGCOMM Computer Communication Review 27.3 (1997): 67-82.*

[35] Das, Sreerupa, and Michael C. Mozer. "A unified gradient-descent/clustering architecture for finite state machine induction." *Advances in neural information processing systems.* 1994.

[36] Brakmo, Lawrence S., and Larry L. Peterson. "TCP Vegas: End to end congestion avoidance on a global Internet." *IEEE Journal on selected Areas in communications 13.8 (1995): 1465-1480.*

Appendix A

General Risk Assessment Form

General Risk Assessment Form

Date: (1) **15/06/2017**	**Assessed by:** (2) **Yao Da**	**Checked / Validated* by:** (3)	**Location:** (4) **Barns Wallis Student Hub**	**Assessment ref no** (5)	**Review date:** (6)
Task / premises: (7) Use PC for UI design and simulation					

Activity (8)	**Hazard** (9)	**Who might be harmed and how** (10)	**Existing measures to control risk** (11)	**Risk rating** (12)	**Result** (13)
Programming	Finger injured	Yao Da	Taking rest after long time work	Low	T

Activity (8)	**Hazard** (9)	**Who might be harmed and how** (10)	**Existing measures to control risk** (11)	**Risk rating** (12)	**Result** (13)
Reading Reference	Harm to Eye	Yao Da	Use Eyedrop	Low	T

Action plan (14)

Ref No	Further action required	Action by whom	Action by when	Done

Appendix B

Project Outline

I Scope

This project involves developing simulation programs in terms of TCP which allows students of Internet and Communications Systems to set simulation scenarios and view results. The simulation of protocol and design of GUI are performed by Matlab.

II Aims

The aim of this project is provide TCP flow control and congestion control mechanisms. The user can provide parameters and traffic of the network either by internal and external sources. It should be simple to set up congestion situations so that the behavior of TCP's congestion control can be investigated.

III objective

It is expected to gain a good understanding of protocol and better programming skills of Matlab through this project.   The ability of report writing will also be improved.

IV Plan

A rough timetable is given below. Specific plan may be changed according to situations of completing tasks.

Duration	**Task**	**Detail Plan**
June 12nd----July 2nd	Background Study and Design GUI	Read relevant reference given by supervisor. Gain basic knowledge about TCP and Transport Layer. Use guide tools of Matlab to design an user interface.
July 3rd---- August 6th	Protocol simulation	Use Matlab to perform simulation of TCP with can be used by MSc students.
August 7th ---- September 3rd	Report Writing	According to the works done in first two parts, write a report which include detail processes, figures and codes.

Appendix C

MATLAB Script to simulate TCP

Appendix C covers all methods to construct the simulator and GUI.

```
function congestion_control(e)
global currentstate
switch(currentstate)
    case('Slow start')
        handleSS(e);
    case('Congestion avoidance')
        handleCA(e);
    case('Fast recovery')
        handleFR(e);
end
end

function handleSS(e)
global cwnd  dupACKcount ssthresh currentstate ACKbase add_FR
if strcmp(e,'new ACK')
    cwnd=cwnd+1;
    dupACKcount=0;
    sender.send;
    currentstate='Slow start';
else
    if strcmp(e,'duplicate ACK')
        dupACKcount=dupACKcount+1;
        currentstate='Slow start';

    end
end
   if strcmp(e,'timeout')
            ssthresh=(cwnd/2);
            cwnd=1;
            dupACKcount=0;
            %retranmit missing segment
            currentstate='Slow start';
    end
if cwnd>=ssthresh
    currentstate='Congestion avoidance';
```

```
end

if dupACKcount==3 && add_FR
    stop_timer;
    ssthresh=fix(cwnd/2);
    cwnd=ssthresh+3;
    fprintf('Sequence %d has been sent to router\n',ACKbase);
    send_to_IP(ACKbase);
    currentstate='Fast recovery';
end
end

function handleFR(e)
global cwnd  dupACKcount ssthresh currentstate
if strcmp(e,'new ACK')
    cwnd=ssthresh;
    dupACKcount=0;
    currentstate='Congestion avoidance';
    sender.send;
else
    if strcmp(e,'duplicate ACK')
    cwnd=cwnd+1;
    %transmit new segment
    currentstate='Fast recovery';
    else
        if strcmp(e,'timeout')
    ssthresh=fix(cwnd/2);
    cwnd=1;
    dupACKcount=0;
    sender.send;
    currentstate='Slow start';
        end
    end
end
end

classdef sender < handle
    events
        TimerTimeOut
        ReceiveACK
    end

    methods(Static=true)
        function send
```

```
            global  NextSeqNum cwnd sendbase i packets maxseqnum
stopbutton
                if ~isnan(NextSeqNum) && ~stopbutton
                    if isempty(timerfind)
                        start_timer();
                    end
                    while NextSeqNum-sendbase<cwnd
                        if NextSeqNum
                         fprintf('Sequence %d has been sent to
router\n',i);
                        send_to_IP(NextSeqNum);
                        else
                          fprintf('Sequence %d has been
dropped\n',i);
                        end
                        i=i+1;
                        if i<=maxseqnum
                        NextSeqNum=packets(i);
                        else
                            NextSeqNum=nan;

                        end
                    end
                end
            end
        end

    methods

        function handletimeout(obj,src,evtdata)
            global sendbase stopbutton%packetsloss
            disp('time out happens...')
            stop_timer;
            %retransmit
            if ~stopbutton
            start_timer;
            fprintf('Sequence %d has been sent to
router\n',sendbase);
            send_to_IP(sendbase);
            congestion_control('timeout');
            end
        end

    %ACK received, with ACK field value of SN
    function ACK_received(obj,src,y)
        fprintf('ACK received, with ACK field value of %d \n',y.SN);
        global sendbase NextSeqNum stopbutton
        if ~stopbutton
            if y.SN+1>sendbase
                sendbase=y.SN+1;
                stop_timer;
                if sendbase<NextSeqNum
                    start_timer;
                end
                congestion_control('new ACK');
            else
                %'a duplicate ACK is received'
                congestion_control('duplicate ACK');
            end
        end
    end
    end
end

function start_timer()
global s1
t=timer;
t.Name='timeout Timer';
t.BusyMode='queue';
t.StartDelay=6;
t.TimerFcn=@(~,~) notify(s1,'TimerTimeOut');%send timeout event
start(t);

function stop_timer()
out=timerfind('Name','timeout Timer');
if ~isempty(out)
    stop(out);
    delete(out);
end
clear out

function Receiver(SeqNum)
global ACKbase s1  rcvBuffer r
r=[r SeqNum];
    t=timer('ObjectVisibility','off');
    t.Name='RTT Simulate2';
    t.StopFcn=@TimeOutStop;
    t.StartDelay=1.5;
```

```
 if ACKbase==SeqNum
     if rcvBuffer
         rcvBuffer = sort(rcvBuffer);
         if rcvBuffer(1) == ACKbase+1
         while length(rcvBuffer)>1
             if rcvBuffer(1)+1~=rcvBuffer(2)
                 break;
             end
             rcvBuffer(1)=[];
         end
         ACKbase = rcvBuffer(1);
         rcvBuffer(1)=[];
         end
     end
     y=ACKbase;
     t.TimerFcn=@(pTimer,~)
notify(s1,'ReceiveACK',ACKEventData(y));
     ACKbase=ACKbase+1;
 else if SeqNum>ACKbase
     y=ACKbase;
     t.TimerFcn=@(pTimer,~)
notify(s1,'ReceiveACK',ACKEventData(y-1));
     rcvBuffer=[rcvBuffer SeqNum];
     end
 end
    start(t)
end

function samplecwnd(nTimer,~,handles)
global cwnd cwnddict round
 cwnddict=[cwnddict cwnd];
 x=0:length(cwnddict)-1;
 plot(handles.axes1,x,cwnddict);
 legend(handles.axes1,'cwnd');
 xlabel(handles.axes1,'Round');
 ylabel(handles.axes1,'MSS');
%  axis(handles.axes1,[0 max(cwnddict)+5 0 round+3]);
 set(handles.text7,'String',['current round:' num2str(round)]);
 round=round+1;
end
```