|  |
| --- |
| ECSE-429 |
| TCP Protocol |
| Simulated using Promela |
|  |
| **Julian Cooper-260246648**  **David Donna - SID** |
| **5/12/2012** |

|  |
| --- |
| [Type the abstract of the document here. The abstract is typically a short summary of the contents of the document. Type the abstract of the document here. The abstract is typically a short summary of the contents of the document.] |

Table of Contents

[Introduction 2](#_Toc342604009)

[Specification 3](#_Toc342604010)

[Opening Handshaking 3](#_Toc342604011)

[Thee Way Handshake Setup 3](#_Toc342604012)

[Simultaneous Open 4](#_Toc342604013)

[Data Transfer 4](#_Toc342604014)

[Implementation 5](#_Toc342604015)

# Introduction

In this project we will be implementing the TCP protocol using a Promela simulation. Our main basis for this implementation is the description provided in the project outline, specifically the finite state machine on page 4 (Figure 4) and the timing diagrams in figures 1-3. Furthermore, we referred to RFC-793 [1] for further clarifications of the protocol, in the case of conflicting descriptions, RFC-793 took precedence.

TCP is a protocol is a wrapper for the Internet Protocol, its functionality centers primarily around transmission reliability. This is a brief overview of some of its features and characteristics:

* Point-to-point:
  + Every connection has one sender and one receiver
* Connection-oriented:
  + Each message is encapsulated by handshaking/control messages
* Reliable, in-order data stream.
* Send and receive buffers
  + These buffers prevent unnecessary data retransmission caused by packet delay (not implemented in our model)
* Full duplex data:
  + Data can flow to-and-from sender and receiver
* Congestion management:
  + TCP will modify its timeout delay to compensate for network congestion, limiting the number of unnecessarily retransmitted packets.

Our program will focus on the handshaking and data transmission aspects of TCP. That is to say, connection setup; data transmission, timeout, and retransmission; and clean connection teardown will be accurately modeled and thoroughly verified.

# Specification

The program must simulate the creation, data-transfer, and tear-down of a TCP connection between a Client and a Server. The model must properly conduct the opening handshaking, the transmission of data after handshaking, and the tear-down of the connection after all data has been transmitted. It must also handle unexpected time-outs during data transmission.

## Opening Handshaking

TCP can open a connection in two ways. The most common way is a three way handshake, initiated by a client and sent to a host. The second way only occurs when two machines attempt to establish a connection with each-other simultaneously.

### Thee Way Handshake Setup

Firstly, the client machine, A, will send a connection request to a host machine, B. The request is a message labelled SYN contains a sequence number, ‘SEQ = *x’*, which is the starting byte of the transmission sequence. Secondly, upon receipt SYN, B will send back a SYN-ACK, with the data ‘SEQ = y’ and ACK = x + 1. Thirdly, A will send an ACK< SEQ = x+1, ACK = y+1 >. A can now send data to B.

### Simultaneous Open

It is possible, though unlikely, that two machines attempt to initiate a connection with each other, in this case Both machines A and B send a SYN request, and both machines respond with a SYN+ACK, finally both machines send the last ACK. Both clients can send data to each other.

*Simultaneous Open Diagram*

## Data Transfer

After a connection is established, Host A will data send packets to B, < SEQ = ACK­B, ACK = SEQB + 1> where ACK­B and SEQB are the last ACK and sequence numbers received from B, after sending a data packet, A will wait for an ACK< SEQ, ACK > from B. Unfortunately, it is possible for data to be lost during transmission, as such, both A and B have value of *t* seconds. If A does not receive an ACK from B after *t* seconds, A will re-transmit the oldest packet that was not ACK’d, i.e its sequence number corresponds to the last ACK value received from B. Similarly, if B does not receive a data packet within *t* seconds after sending an ACK, it will assume that its ACK packet was dropped, and B will retransmit the ACK.

## Connection Tear-down

When A is done transmitting data to B, it will send a FIN packet to

# Implementation

As previously mentioned, our project is based on the finite state machine shown in Figure 4 of the project description. In our implementation, we simplified the FSM by splitting it in to two separate machines that run concurrently; one machine for the sender, and one machine for the receiver. Each state is a defined by a label within the Promela code, and the state transitions are governed by signals sent over channel data types. The current states of both machines is also stored in a set of Boolean variables, these variables are used to verify the correctness of the model during various points of execution.

## Client FSM description

Blah blah blah

## Server FSM description

Blah blah blah

# Verification

## State assertions

## LTL-Assertions

Description of each of seems like it would be a bit intense. Maybe something about code/branch coverage

## Mutations

List of mutations/reasoning, did shit break?