ECSE 429 Final Project
Software Validation and Verification
Fall 2012 Semester

(Prepared by Dr. K. Radecka and Jason G. Tong)

Due: Wednesday, December 5th, 2012 at 11:59PM

Instructions

The final project for the Software Validation course will be executed in **teams of two students**. Since we have an even number of students, there should not be any group containing more than two members. It is *your responsibility to find a partner*. The projects due date is the last class of the semester. Each group will do one project submission into a specially created dropbox on the renovated WebCT course website.

Grading Scheme

The final project is worth **25%** of your final grade and it is compromised of the following:

40% – Promela Implementation

60% – Verification Report including the realization of objectives described in the last section of this write up.

The Verification Report will be judged by the thoroughness and effort your team putting into verifying your Promela model along with meeting the objectives of this project.

---

1 Introduction – Transport Control Protocol

For the final project, you will be working on the Transport Control Protocol (TCP), which is a part of the Internet Protocol Suite (IP) used for providing reliable communications between any two hosts. This protocol is popular among many similar protocols due to its reliable means of transmitting and receiving data while maintaining data integrity. In order to create a successful connection, both hosts must transmit a set of messages for channel setup and channel tear down, known as *the three-way handshake*.

In the first part of the write up you will find a short introduction to the protocols used by TCP/IP, which can serve as a basic specification of the project. You should treat this description as the starting point in your design, and search literature for more in-depth data.

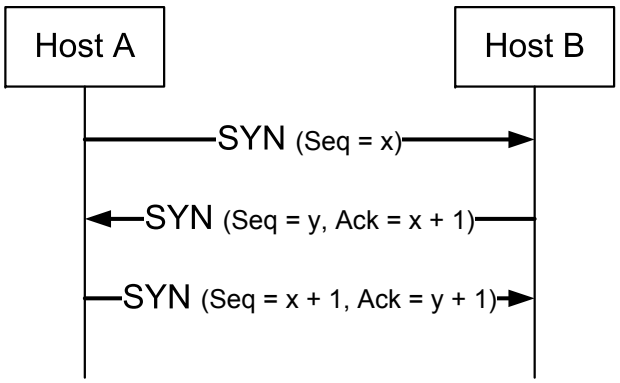

Figure 1: Three Way Handshake for Connection Setup

Establishing Connection

Shown in Figure 1 is the three-way handshake for setting up a connection between the two hosts. Host A sends a **SYN** message containing a message called (**Seq** $= x$). x represents the timestamp of operations of Host A. After Host B receives the **SYN** message, it then sends its own **SYN** message back to Host A with a different message (**Seq** $= y$, **Ack** $= x + 1$). Host A not only receives the **SYN** message from Host B , but it is also expected to receive Host B's timestamp (y) and an incremented value of its own timestamp ($x+1$). Finally, Host A sends another **SYN** message to Host B, this time accompanied with the incremented value of his timestamp ($x + 1$) and Host B's incremented timestamp ($y + 1$). This is the last step in establishing a connection.

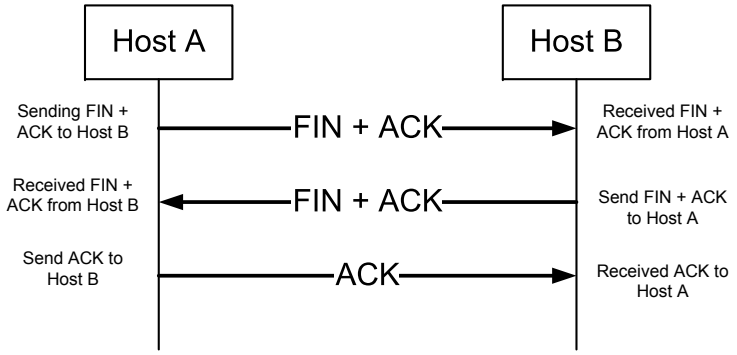

Figure 2: Three Way Handshake for Connection Teardown

Connection Tear Down

The similar three-way handshake is also applied to tear down the connection, Figure 2. The sequence of messages presented in the figure illustrates breaking the connection between Hosts A and B. To initiate the tear down, Host A will send a **FIN + ACK** message to Host B, signalling for a connection to tear down. Following this message, Host B sends back the same message to Host A notifying that the connection tear down signal has been acknowledged. Finally, Host A sends an **ACK** to Host B, reaffirming to Host B that the connection is now terminated.

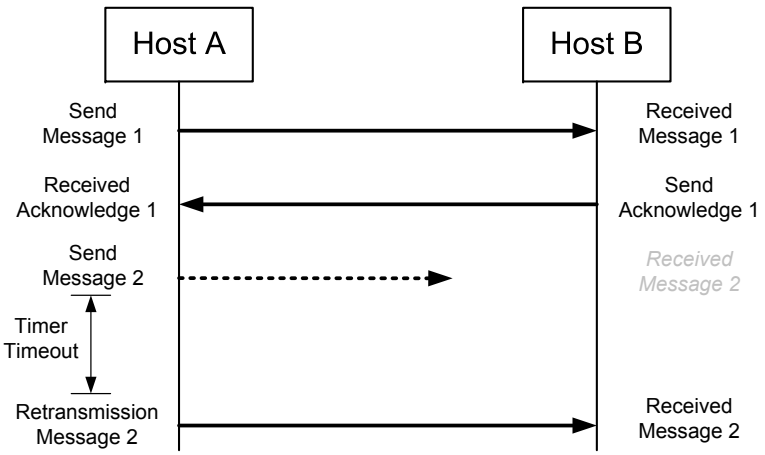

Figure 3: Packet Loss during Transmission

Re-transmission of Lost Data

Sometimes data sent from one host to the other is lost in the transmission. Therefore, it is required to have a mechanism, like in Figure 3 to retransmit the data . The illustration lists the steps required for the re-transmission of the lost data. First, consider sending *Message 1* from Host A to Host B. After establishing the connection between the two hosts as seen from Figure 1, *Message 1* is sent by Host A. At the same time the Timer starts measuring the time from the beginning of the transmission. Upon receiving *Message 1*, Host B issues to Host A an Acknowledgement signal. As the Acknowledgement signal is received by Host A before the Timer reaches the Timeout value, Message 1 is declared as being sent successfully.

Now, look what happens when *Message 2* is being sent by Host A to Host B. First, the connection has been established between the two Hosts as shown in Figure 1. Upon transmitting *Message 2*, the Timer of Host A is again reset and starts counting. When Timer reaches the value **Timeout**, and the *Acknowledgement* signal does not arrive from Host B, then *Message 2* is declared as lost, and will be retransmitted by Host A.

We note that for every data sent and received, both hosts should remain in the ESTABLISHED CONNECTION (Data Transfer) state. A connection can only be torn down whenever Host A (the sender) initiates the shutdown of the TCP channel with the sequence of messages shown in Figure 2.

FSM of Transfer Control Protocol Connection Management

Figure 4 shows the Finite State Machine of the Transfer Control Protocol Connection management. Each edge in the graph is an EVENT / ACTION pair which describes the required conditions in order for the FSM to enter into a different state of the TCP protocol. Solid lines represent the path of a client while the dashed lines represent the path of the server. Whenever the two hosts establish a connection, they enter into the data transmission state marked as ESTABLISHED CONNECTION. In this state, both hosts can transmit any kind of random data (messages, datagrams, etc.).

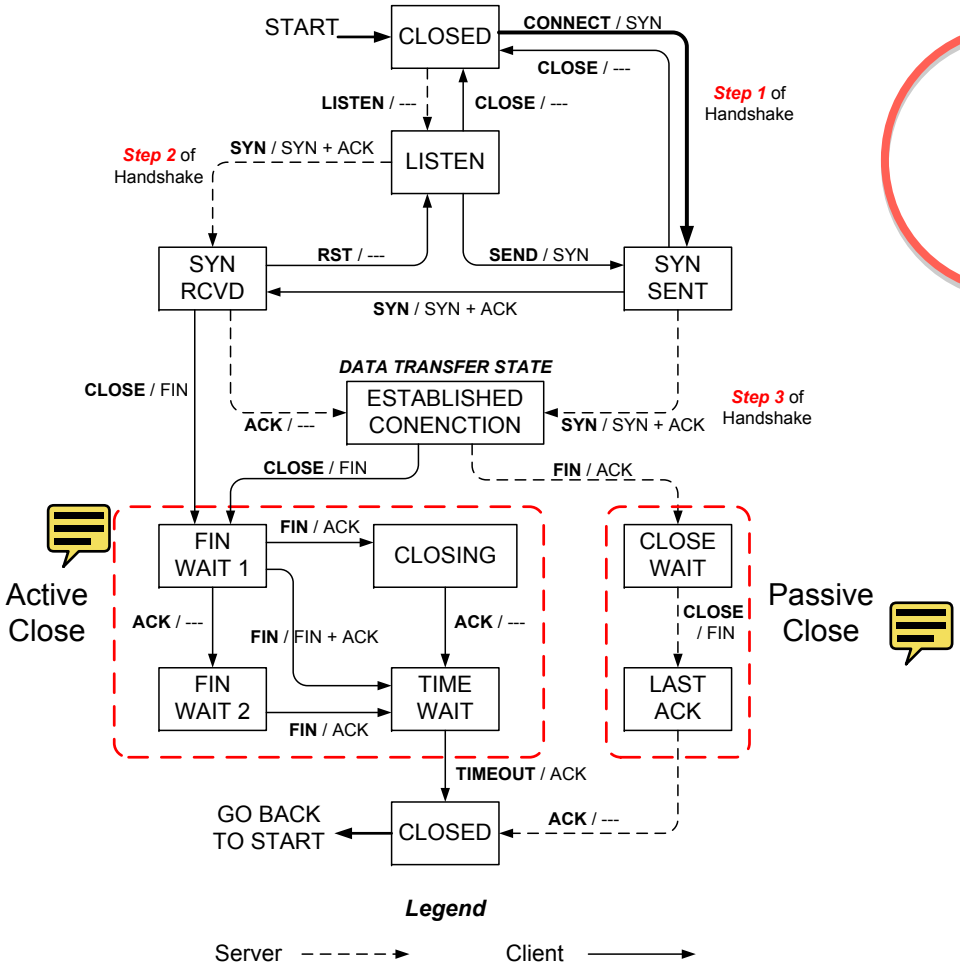

Figure 4: Finite State Machine of TCP Protocol

STATE NAME	DESCRIPTION
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED CONNECTION	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIMED WAIT	Wait for all packets to terminate
CLOSING	Both sides have tried to close the connection simulatenously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to terminate

Table 1: Description of each state

2 Project Objectives

Your team is responsible for designing, implementing and verifying your Promela model of the TCP protocol. It is important to note that for a successful completion of this project, the list of the following objectives must be met:

1. Design and implement a Promela model of the TCP protocol
2. Verify the correctness of the model by incorporating immediate and LTL assertions
3. Perform mutation testing by injecting structural (code) faults into your Promela model

Objective 1: Design Specification

Using as a starting point the description of the TCP/IP protocols provided in the first part of this project, derive your own specification of the protocol, which you are going to implement and test in the remaining part of this project. Your specification should be written up to the standards introduced in this course. You can use any means of describing your specification including algorithms, pseudo-codes, diagrams, flow charts, FSMs and plain English. Your specification must be readable to any user.

Objective 2: Design and Implementation

Based on your specification of TCP implement the TCP protocol in Promela. Your Promela model should resemble a connection setup as similar to Figures 1 through 2. The behaviour between any two hosts, such as the connection setup and tear down, should follow the Finite State Machine given in Figure 4.

During the data transmission, both hosts must adhere to the message (or data) passing mechanism shown in Figure 3. Additionally, both hosts should be capable of retransmitting a message if either fails to receive it during the time out window.

You can add more elements to the original specification in order to refine the specification. All the additions, as well as your final design must be well documented.

Objective 3: Verification of Equivalence of Specification and Implementation

Build models of your specification and implementation of TCP protocol, and verify that they are equivalent. Towards that goal you can use any appropriate methods introduced in class.

Objective 4: Assertion Writing and Verification

In this part of the project you are asked to verify the correctness of your implementation of the Promela model, using immediate and LTL assertions and Spin's model checking capabilities. The written LTL assertions should thoroughly explore the different behaviours of the TCP protocol as given from the Finite State Machine. Additionally, LTL assertions should verify the data transmission behaviour between the two hosts.

Objective 5: Mutation Testing

Using the written assertions from Objective 4, inject a set of at least 10 mutants (structural faults) into your Promela model. These faults should create the faulty behaviour of the TCP such that your LTL and immediate assertions must catch. A few examples of mutants would be to deliberately put incorrect condition guards, a process jumping into the incorrect state due to an incorrect assigned state variable, etc.

3 Deliverables and Report Outline

- Promela Model of the TCP along with immediate assertions
- A separate file containing LTL assertions used for verifying the TCP
- A fully detailed Verification Report

The list shown below is the suggested outline that your Verification Report must incorporate.

Introduction

- Brief description of the Project

Desription of your Implementation

- The approach that you used to implement TCP Connection Management Protocol
- How you implemented the retransmission mechanism in the Connection Established / Data Tranmission State of the TCP
- Any additional mechanisms that you added

Verification with Spin

- List of Immediate and LTL Assertions and Descriptions. For each assertion you incorporate, it must be accompanied by a description of what you are trying to verify. E.g., what behaviour of the TCP / retransmission mechanism you are trying to check the correctness?
- Label each assertion as `LTL_XX` or `IMM_XX` (where `XX` is an integer) for LTL and immediate assertions respectively

Mutation Testing

- For each mutant / fault:
  - Label it as `MUTANT_XX` (where `XX` is an integer)
  - Describe the intented incorrect behaviour that your TCP protocol should generate, and indicate the location (in terms of line number) of the mutant

    * Which assertion *should* catch the incorrect behaviour? Indicate the assertion based on the label number (E.g., "...`MUTANT_05` was caught by assertion `LTL_47`...")
    * Did the assertion (immediate or LTL) was able to detect the incorrect behaviour? If not, can you suggest another immediate or LTL assertion to add?

Conclusions

- Closing remarks on the implementation of your Promela model and the approach you taken to verify the TCP protocol

Project Submission Instructions

Each team will only need to submit one project and it should contain the following:

- Verification report in a **PDF** document only
- Your finalized Promela model along with :
  - Detailed comments describing the model
  - Commented out mutants / faults that you injected along with the associated label number
  - The inserted immediate assertions labelled appropriately
  - A separate file containing the LTL assertions labelled appropriately

Package the entire project as the file name:

`[LAST_NAME_OF_TEAM_MEMBERS]_final_project_2012.zip`

Good luck and Have Phun!