Appendix A

```
-- The TCP state machine

MODULE main
VAR t1: tcp;

MODULE tcp
VAR

-- All the possible states in a  connection lifetime.
state: {LISTEN,  SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1,
FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, CLOSED};

-- The activity  of the TCP can be characterized as responding to events.
-- The events that occur can be cast into three categories: user calls,
-- arriving segments, and timeouts.
event: {USERCALL, SEGMENT, TIMEOUT};

-- Six usercalls are included in the model, which are OPEN-P, OPEN-A,
-- SEND, RECEIVE, CLOSE, ABORT.  STATUS is eliminated.
usercall: {OPEN-P, OPEN-A, SEND, RECEIVE, CLOSE, ABORT};

-- active_flag is used to indicate if the latest OPEN is an active OPEN
active_flag: boolean;

-- prc_flag is used to indicate  if the security  and the precedence of
-- the segment match the connection.
prc_flag: {LOW,  EQUAL, HIGH};

-- In TCP, 6 bits are used as control bits
-- urg_flag is to indicate Urgent Pointer field significant
urg_flag: boolean;

-- ack_flag is to indicate Acknowledgment field significant
ack_flag: boolean;

-- psh_flag is to Push Function
psh_flag: boolean;

-- rst_flag is to Reset the connection
rst_flag: boolean;

-- syn_flag is to Synchronize sequence numbers
syn_flag: boolean;

-- fin_flag is to indicate No more data from sender
fin_flag: boolean;

-- there are 3 kinds of timeout as defined below

timeout: {USER-TIMEOUT, RETRANSMISSION-TIMEOUT, TIMEWAIT-TIMEOUT};
-- In TCP, 32 bits are used as Acknowledgment Number and Sequence
-- Number. In this program, we use boolean to indicate if the Ack
-- Number and Seq Number are ok.

ack_ok: boolean;
seq_ok: boolean;
```

```
-- The following section is to initial all the varables

ASSIGN
init(event) := {USERCALL, SEGMENT, TIMEOUT};
next(event) := {USERCALL, SEGMENT, TIMEOUT};
init(active_flag) := {0, 1};
next(active_flag) := case
    event = USERCALL & usercall = OPEN-A: 1;
    event = USERCALL & usercall = OPEN-P: 0;
    1: active_flag;
  esac;
init(prc_flag) := {LOW, EQUAL, HIGH};
next(prc_flag) := {LOW, EQUAL, HIGH};
init(urg_flag) := {0, 1};
next(urg_flag) := {0, 1};
init(ack_flag) := {0, 1};
next(ack_flag) := {0, 1};
init(psh_flag) := {0, 1};
next(psh_flag) := {0, 1};
init(rst_flag) := {0, 1};
next(rst_flag) := {0, 1};
init(syn_flag) := {0, 1};
next(syn_flag) := {0, 1};
init(fin_flag) := {0, 1};
next(fin_flag) := {0, 1};
init(timeout) := {USER-TIMEOUT, RETRANSMISSION-TIMEOUT,
TIMEWAIT-TIMEOUT};
next(timeout) := {USER-TIMEOUT,  RETRANSMISSION-TIMEOUT,
TIMEWAIT-TIMEOUT};
init(ack_ok) := {0, 1};
next(ack_ok) := {0, 1};
init(seq_ok) := {0, 1};
next(seq_ok) := {0, 1};

init(state) := {LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1,
FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, CLOSED};
next(state) := case

-- The following section is to handle the situation when state = CLOSED

state = CLOSED :
  case
    event = USERCALL:

-- When the event is usercall, and the usercall is to open a connection
-- and at the time active_flag is on, then the next state is SYN-SENT;
-- if at the time active_flag is off, the the next state is LISTEN.

      case
        usercall = OPEN-A: SYN-SENT;
        usercall = OPEN-P: LISTEN;
        1: CLOSED;
      esac;

-- All the coming SEGMENT event  will be led to CLOSED state.

    event = SEGMENT: CLOSED;
```

```
-- When event is TIMEOUT, and it’s a USER-TIMEOUT, then the next state
-- is CLOSED.

    event = TIMEOUT:
      case
        timeout = USER-TIMEOUT:  CLOSED;
        1: state;
      esac;
  esac;

-- The following section is to handle the situation when state = LISTEN

state = LISTEN :
  case
    event = USERCALL:
      case
        usercall = OPEN-A: SYN-SENT;
        usercall = SEND: SYN-SENT;
        usercall = CLOSE: CLOSED;
        usercall = ABORT: CLOSED;
        1 : LISTEN;
      esac;

-- When the state is LISTEN and the event is SEGMENT, if it is not a
-- reset or an acknowledgment, rather it is a syn segment with correct
-- precedence and security level, then the next state is SYN-RECEIVED.
-- All the other segment will make the state keep unchanged.

    event = SEGMENT:
      case
        !rst_flag & !ack_flag & syn_flag & (prc_flag = EQUAL):
        SYN-RECEIVED;
        1: LISTEN;
      esac;
    event = TIMEOUT:
      case
        timeout = USER-TIMEOUT: CLOSED;
        1: state;
      esac;
  esac;

-- The following section is to handle the situation when state =
-- SYN-SENT.

state = SYN-SENT :
  case
    event = USERCALL:
      case
        usercall = SEND: SYN-SENT;
        usercall = CLOSE: CLOSED;
        usercall = ABORT: CLOSED;
        1 : SYN-SENT;
      esac;

-- In the state of SYN-SENT, we will first check if the coming segment
-- is an ack with the correct ack number: if the ack number is wrong,
-- then the following state will be SYN-SENT, else state is remain
```

```
-- unchanged;
-- secondly, if the coming segment is a reset ack with correct ack
-- number, then it will go to the state of CLOSED, else go to SYN-SENT;
-- Thirdly, if the precedence and security level is not normal, then
-- the next state must be SYN-SENT.
-- Next if the coming segment is a syn information with correct ack
-- number then the next state must be ESTABLISHED, else SYN-RECEIVED.

    event = SEGMENT:
      case
         ack_flag & !ack_ok: SYN-SENT;
         rst_flag & ack_flag & ack_ok: CLOSED;
         rst_flag & !ack_ok: SYN-SENT;
         !(prc_flag = EQUAL) : SYN-SENT;
         syn_flag & ack_ok: ESTABLISHED;
         syn_flag & !ack_ok: SYN-RECEIVED;
         1: SYN-SENT;
      esac;
    event = TIMEOUT:
      case
         timeout = USER-TIMEOUT: CLOSED;
         1: state;
      esac;
  esac;

-- The following section is to handle the situation when state =
-- SYN-RECEIVED

state = SYN-RECEIVED :
  case
    event = USERCALL:
      case
         usercall = OPEN-P | usercall = OPEN-A: SYN-RECEIVED;
         usercall = CLOSE: FIN-WAIT-1;
         usercall = ABORT: CLOSED;
     1 : SYN-RECEIVED;
      esac;

-- In the state of SYN-RECEIVED, we will first check if the coming
-- segment is with a correct sequence number, if the sequence number is
-- wrong, the state remain unchanged.
-- secondly, if the coming segment is a reset with active_flag off then
-- the next state is LISTEN; if the active_flag is on, then it will
-- goes to the state of CLOSED.
-- Thirdly, if the precedence and security level is not normal, then
-- the next state must be SYN-RECEIVED.
-- Next if the coming segment is a syn then the next state must be
-- CLOSED.
-- If the coming segment is not a ack, then it will go to SYN-RECEIVED
-- State; if the coming segment is an ack and with incorrect ack
-- number, the next state is also SYN-RECEIVED.
-- If the ack with correct ack number and the fin_flag is off, it will
-- successfully enter the state of ESTABLISHED; while if the fin_flag
-- is on, then next state is CLOSE-WAIT.

event = SEGMENT:
      case
         !seq_ok: SYN-RECEIVED;
```

```
        rst_flag & !active_flag: LISTEN;
        rst_flag & active_flag: CLOSED;
        !(prc_flag = EQUAL): SYN-RECEIVED;
        syn_flag: CLOSED;
        !ack_flag: SYN-RECEIVED;
        !ack_ok: SYN-RECEIVED;
        ack_ok & !fin_flag: ESTABLISHED;
        fin_flag: CLOSE-WAIT;
        1: SYN-RECEIVED;
      esac;
    event = TIMEOUT:
      case
        timeout = USER-TIMEOUT: CLOSED;
        1: state;
      esac;
  esac;

-- The following section is to handle the situation when state =
-- ESTABLISHED

state = ESTABLISHED :
  case
    event = USERCALL:
      case
        usercall = OPEN-P | usercall = OPEN-A : ESTABLISHED;
        usercall = CLOSE: FIN-WAIT-1;
        usercall = ABORT: CLOSED;
      1 : ESTABLISHED;
      esac;

-- In the state of ESTABLISHED, we will first check if the coming
-- segment is with an correct sequence number, if the sequence number
-- is wrong the state remain unchanged.
-- secondly, if the coming segment is a reset the next state will be
-- CLOSED.
-- Thirdly, if the precedence and security level is not normal, then
-- the next state must be ESTABLISHED.
-- Next if the coming segment is a syn then the next state must be
-- CLOSED.
-- If the coming segment is not an ack, then it will go to ESTABLISHED
-- State; if the coming segment is an ack and with incorrect ack
-- number, the next state is also ESTABLISHED.
-- If the ack with correct ack number and the fin_flag is on, then next
-- state is CLOSE-WAIT.

    event = SEGMENT:
      case
        !seq_ok: ESTABLISHED;
        rst_flag: CLOSED;
        !(prc_flag = EQUAL): ESTABLISHED;
        syn_flag: CLOSED;
        !ack_flag: ESTABLISHED;
        !ack_ok: ESTABLISHED;
        fin_flag: CLOSE-WAIT;
        1: ESTABLISHED;
      esac;
    event = TIMEOUT:
      case
```

```
        timeout = USER-TIMEOUT:  CLOSED;
        1: state;
      esac;
  esac;

-- The following section is to handle the situation when state =
-- FIN-WAIT-1.

state = FIN-WAIT-1 :
  case
    event = USERCALL:
      case
        usercall = ABORT: CLOSED;
        1 : FIN-WAIT-1;
      esac;

-- In the state  of FIN-WAIT-1, we will first check if the coming
-- segment is with a correct sequence number, if the sequence number is
-- wrong,the state remain unchanged.
-- Secondly, if  the coming segment is a reset then it will go to the
-- state of CLOSED.
-- Thirdly if the coming segment is a syn then the next state must be
-- CLOSED.
-- If the coming segment is not an ack, then it will go to FIN-WAIT-1
-- State; if the coming segment is an ack and with incorrect ack
-- number, the next state is also FIN-WAIT-1.
-- If the ack with correct ack number and the fin_flag is off, it will
-- either enter  the state of FIN-WAIT-1 or FIN-WAIT-2; while if  the
-- fin_flag is on, then next state is TIME-WAIT or CLOSING. To determine
-- which state to enter, we need to know if the coming ACK
acknowledgment
-- is in response to the FIN message which we have sent before which we
-- won't deal with in this model.

    event = SEGMENT:
      case
        !seq_ok: FIN-WAIT-1;
        rst_flag: CLOSED;
        syn_flag: CLOSED;
        !ack_flag: FIN-WAIT-1;
        !ack_ok: FIN-WAIT-1;
        ack_ok & !fin_flag: {FIN-WAIT-1, FIN-WAIT-2};
        fin_flag: {TIME-WAIT, CLOSING};
        1: FIN-WAIT-1 ;
      esac;
    event = TIMEOUT:
      case
        timeout = USER-TIMEOUT: CLOSED;
        1: state;
      esac;
  esac;

-- The following section is to handle the situation when state =
-- FIN-WAIT-2.

state = FIN-WAIT-2 :
  case
    event = USERCALL:
```

```
      case
        usercall = ABORT: CLOSED;
        1 : FIN-WAIT-2;
      esac;

-- In the state of FIN-WAIT-2, we will first check if the coming
-- segment is with a correct sequence number, if the sequence number is
-- wrong,the state remain unchanged.
-- Secondly, if the coming segment is a reset then it will go to the
-- state of CLOSED.
-- Thirdly if the coming segment is a syn then the next state must be
-- CLOSED.
-- If the coming segment is not a ack, then it will go to FIN-WAIT-2
-- State; if the coming segment is an ack and with incorrect ack
-- number, the next state is also FIN-WAIT-2.
-- If the ack with correct ack number and the fin_flag is off, it will
-- either enter the state of FIN-WAIT-2; while if the fin_flag is on,
-- then next state is TIME-WAIT.

    event = SEGMENT:
      case
        !seq_ok: FIN-WAIT-2;
        rst_flag: CLOSED;
        syn_flag: CLOSED;
        !ack_flag: FIN-WAIT-2;
        !ack_ok: FIN-WAIT-2;
        ack_ok & !fin_flag: FIN-WAIT-2;
        fin_flag: TIME-WAIT;
        1: FIN-WAIT-2 ;
      esac;
    event = TIMEOUT:
      case
        timeout = USER-TIMEOUT: CLOSED;
        1: state;
      esac;
  esac;

-- The following section is to handle the situation when state =
-- CLOSE-WAIT

state = CLOSE-WAIT :
  case
    event = USERCALL:
      case
        usercall = OPEN-P | usercall = OPEN-A : CLOSE-WAIT;
        usercall = CLOSE: LAST-ACK;
        usercall = ABORT: CLOSED;
        1 : CLOSE-WAIT;
      esac;

-- In the state of CLOSE-WAIT, we will first check if the coming
-- segment is with a correct sequence number, if the sequence number is
-- wrong,the state remain unchanged.
-- Secondly, if the coming segment is a reset then it will go to the
-- state of CLOSED.
-- Thirdly if the coming segment is a syn then the next state must be
-- CLOSED.
-- If the coming segment is not an ack, then it will go to CLOSE-WAIT
```

```
-- State; if the coming segment is an ack and with incorrect ack
-- number, the next state is also CLOSE-WAIT.
-- If the ack with correct ack number and the fin_flag is off, it will
-- either enter  the state of CLOSE-WAIT; while if the fin_flag is on,
-- then next state is CLOSE-WAIT.

    event = SEGMENT:
      case
         !seq_ok: CLOSE-WAIT;
         rst_flag: CLOSED;
         syn_flag: CLOSED;
         !ack_flag: CLOSE-WAIT;
         !ack_ok: CLOSE-WAIT;
         ack_ok & !fin_flag: CLOSE-WAIT;
         fin_flag: CLOSE-WAIT;
         1: CLOSED ;
      esac;
    event = TIMEOUT:
      case
         timeout = USER-TIMEOUT: CLOSED;
         1: state;
      esac;
  esac;

-- The following section is to handle the situation when state =
-- CLOSING.

state = CLOSING :
  case
    event = USERCALL:
      case
         usercall = ABORT: CLOSED;
         1 : CLOSING;
      esac;

-- In the state  of CLOSING, we will first check  if the coming segment
-- is with a correct sequence number, if the sequence number is wrong,
-- the state remain unchanged.
-- Secondly, if  the coming segment is a reset then it will go to the
-- state of CLOSED.
-- Thirdly if the coming segment is a syn then the next state must be
-- CLOSED.
-- If the coming segment is not an ack, then it will go to CLOSING
-- State; if the coming segment is an ack and with incorrect ack
-- number, the next state is also CLOSING.
-- If the ack with correct ack number and the fin_flag is off, it will
-- either enter  the state of CLOSING or TIME-WAIT; while if the
-- fin_flag is on, then next state is CLOSING. To determine which
-- state to enter, we need to know if the coming ACK acknowledgment
-- is in response to the FIN message which we have sent before which
-- we won't deal with in this model.

    event = SEGMENT:
      case
         !seq_ok: CLOSING;
         rst_flag: CLOSED;
         syn_flag: CLOSED;
         !ack_flag: CLOSING;
```

```
        !ack_ok: CLOSING;
        ack_ok: {CLOSING, TIME-WAIT};
        fin_flag: CLOSING;
        1: CLOSING ;
      esac;
    event = TIMEOUT:
      case
        timeout = USER-TIMEOUT: CLOSED;
        1: state;
      esac;
  esac;

-- The following section is to handle the situation when state =
-- LAST-ACK.

state = LAST-ACK :
  case
    event = USERCALL:
      case
        usercall = ABORT: CLOSED;
        1 : LAST-ACK;
      esac;

-- In the state of LAST-ACK, we will first check if the coming
-- segment is with a correct sequence number, if the sequence number is
-- wrong, the state remain unchanged.
-- Secondly, if the coming segment is a reset then it will go to the
-- state of CLOSED.
-- Thirdly if the coming segment is a syn then the next state must be
-- CLOSED.
-- If the coming segment is not an ack, then it will remain in LAST-ACK
-- State; if the coming segment is an ack and with incorrect ack
-- number, the next state is also LAST-ACK.
-- If the ack with correct ack number and the fin_flag is off, it will
-- either enter the state of LAST-ACK or CLOSED; while if the fin_flag
-- is on, then next state is LAST-ACK. To determine which state to
-- enter, we need to know if the coming ACK acknowledgment is in
-- response to the FIN message which we have sent before which we
-- won't deal with in this model.

    event = SEGMENT:
      case
        !seq_ok: LAST-ACK;
        rst_flag: CLOSED;
        syn_flag: CLOSED;
        !ack_flag: LAST-ACK;
        !ack_ok: LAST-ACK;
        ack_ok: {LAST-ACK, CLOSED};
        fin_flag: LAST-ACK;
        1: LAST-ACK ;
      esac;
    event = TIMEOUT:
      case
        timeout = USER-TIMEOUT: CLOSED;
        1: state;
      esac;
  esac;
```

```
-- The following section is to handle the situation when state =
-- TIME-WAIT.

state = TIME-WAIT :
  case
    event = USERCALL:
      case
        usercall = ABORT: CLOSED;
        1 : TIME-WAIT;
      esac;

-- In the state of TIME-WAIT, we will first check if the coming
-- segment is with a correct sequence number, if the sequence number is
-- wrong, the state remain unchanged.
-- Secondly, if the coming segment is a reset then it will go to the
-- state of CLOSED.
-- Thirdly if the coming segment is a syn then the next state must be
-- CLOSED.
-- If the coming segment is not  a ack, then it will go to TIME-WAIT
-- State; if the coming segment  is an ack and with incorrect ack
-- number, the next state is also TIME-WAIT.
-- If the ack with correct ack number and the fin_flag is on, it will
-- either enter  the state of TIME-WAIT.

    event = SEGMENT:
      case
        !seq_ok: TIME-WAIT;
        rst_flag: CLOSED;
        syn_flag: CLOSED;
        !ack_flag: TIME-WAIT;
        ack_flag & fin_flag: TIME-WAIT;    --start timer 2MSL
        fin_flag: TIME-WAIT;
        1: TIME-WAIT ;
      esac;
    event = TIMEOUT:
      case
        timeout = USER-TIMEOUT: CLOSED;
        timeout = TIMEWAIT-TIMEOUT: CLOSED;
        1: state;
      esac;
  esac;

DEFINE
out_rst := case
  event = SEGMENT: case
-- If a segments arrives, there are three cases for reset generation.

-- 1. If the connection does not exist (CLOSED) then a reset is sent
-- in response to any incoming segment except another reset.

    state = CLOSED  & !rst_flag: 1;

-- If the connection is in any non-synchronized state (LISTEN, SYN-SENT,
-- SYN-RECEIVED), and the incoming segment acknowledges something not
yet -- sent (the segment carries an unacceptable ACK), or if an incoming
-- segment has a security level or compartment which does not exactly
-- match the level and compartment requested for the connection, a reset
-- is sent.
```

```
  (state = LISTEN | state = SYN-SENT | state = SYN-RECEIVED) &
  ((ack_flag & !ack_ok) | !(prc_flag = EQUAL)): 1;

-- If the connection is in a synchronized state (ESTABLISHED, FIN-WAIT-
1,
-- FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), any
-- unacceptable segment (out of window sequence number or unacceptible
-- acknowledgment number or a security level or compartment which does
-- not exactly match the level and compartment requested for the
-- connection, a reset is sent.

    (state = ESTABLISHED | state = FIN-WAIT-1 | state = FIN-WAIT-2 |
    state = CLOSE-WAIT | state = CLOSING | state = LAST-ACK | state =
    TIME-WAIT) & (!seq_ok | (ack_flag & !ack_ok) | !(prc_flag = EQUAL)):
    1;
   1: 0;
   esac;

-- If the connection is in the state SYN-RECEIVED or ESTABLISHED or
-- FIN-WAIT-1 or FIN-WAIT-2 or CLOSE-WAIT and there is a usercall
-- ABORT, then a reset is sent.

  event = USERCALL: case
    usercall = ABORT & (state = SYN-RECEIVED | state = ESTABLISHED |
               state = FIN-WAIT-1 | state = FIN-WAIT-2 |
               state = CLOSE-WAIT): 1;
    1:0;
    esac;
  1: 0;
```

Appendix B.

```
State 1.1:
_process_selector_ = main
out_rst = 0
state = CLOSED
event = TIMEOUT
usercall = ABORT
active_flag = 0
prc_flag = HIGH
urg_flag = 0
ack_flag = 0
psh_flag = 0
rst_flag = 0
syn_flag = 0
fin_flag = 0
timeout = TIMEWAIT-TIMEOUT
ack_ok = 0
seq_ok = 0
State 1.2:
_process_selector_ = main
out_rst = 0
state = CLOSED
event = USERCALL
usercall = OPEN-P
active_flag = 0
prc_flag = HIGH
urg_flag = 0
ack_flag = 0
psh_flag = 0
rst_flag = 0
syn_flag = 0
fin_flag = 0
timeout = TIMEWAIT-TIMEOUT
ack_ok = 0
seq_ok = 0
State 1.3:
_process_selector_ = main
out_rst = 0
state = LISTEN
event = SEGMENT
usercall = ABORT
active_flag = 0
prc_flag = EQUAL
urg_flag = 0
ack_flag = 0
psh_flag = 0
rst_flag = 0
syn_flag = 1
fin_flag = 0
timeout = TIMEWAIT-TIMEOUT
ack_ok = 0
seq_ok = 0
-- loop starts here --
State 1.4:
_process_selector_ = main
out_rst = 0
state = SYN-RECEIVED
```

```
event = TIMEOUT
usercall = ABORT
active_flag = 0
prc_flag = HIGH
urg_flag = 0
ack_flag = 0
psh_flag = 0
rst_flag = 0
syn_flag = 0
fin_flag = 0
timeout = TIMEWAIT-TIMEOUT
ack_ok = 0
seq_ok = 0
State 1.5:
_process_selector_ = main
out_rst = 0
state = SYN-RECEIVED
event = TIMEOUT
usercall = ABORT
active_flag = 0
prc_flag = HIGH
urg_flag = 0
ack_flag = 0
psh_flag = 0
rst_flag = 0
syn_flag = 0
fin_flag = 0
timeout = USER-TIMEOUT
ack_ok = 0
seq_ok = 0
State 1.6:
_process_selector_ = main
out_rst = 0
state = CLOSED
event = USERCALL
usercall = OPEN-P
active_flag = 0
prc_flag = HIGH
urg_flag = 0
ack_flag = 0
psh_flag = 0
rst_flag = 0
syn_flag = 0
fin_flag = 0
timeout = TIMEWAIT-TIMEOUT
ack_ok = 0
seq_ok = 0
State 1.7:
_process_selector_ = main
out_rst = 0
state = LISTEN
event = SEGMENT
usercall = ABORT
active_flag = 0
prc_flag = EQUAL
urg_flag = 0
ack_flag = 0
psh_flag = 0
```

```
rst_flag = 0
syn_flag = 1
fin_flag = 0
timeout = TIMEWAIT-TIMEOUT
ack_ok = 0
seq_ok = 0
State 1.8:
_process_selector_ = main
out_rst = 0
state = SYN-RECEIVED
event = TIMEOUT
usercall = ABORT
active_flag = 0
prc_flag = HIGH
urg_flag = 0
ack_flag = 0
psh_flag = 0
rst_flag = 0
syn_flag = 0
fin_flag = 0
timeout = TIMEWAIT-TIMEOUT
ack_ok = 0
seq_ok = 0
```