Protocols and Finite State Machines

Dr. Yehuda Ben-Shimol
Course Overview

- Major topics
  - Formal methods for describing communication protocols - Protocol description
  - Given the description - is the protocol correct? Protocol validation
  - Given the requirements - how do we design the protocol? Protocol design (a practical approach)

These topics will be mixed throughout the lectures
Course structure

- Introduction and Finite State Machines ( FSM )
- Formal description languages (here we stop guessing or being “intuitive” in some sense)
  - CSP - rest of the course

There are also

- CCS - a similar approach to CSP by Milner
- LOTOS - a combination of CSP CCS and Z
- PROMELA
- ESTELE
- Petri Nets - old approach, extended to stochastic analysis 30 years ago, still active area of research
- ...
Why is this course?

- Helping with other courses (based on my own experience)
  - Quantitative methods, Final project
- Later in professional life - specification, design, verification, implementation, analysis of:
  - communication protocols
  - protocols for operation systems
  - parallel/concurrent and distributed algorithms
  - Distributed systems
  - Any other software/hardware based solution that interacts with environment
Note

- Communication protocols are just a private case of concurrent or distributed systems
  - Connecting remote nodes
  - Defined usually in terms of its behaviour (another word for an event driven system, or interaction with the environment).

The importance grows with the need for distributed and/or parallel (or concurrent) computation
What will be new and what will not?

- **Mostly familiar** protocols (I rely on current knowledge and understanding)
- How to "think" about protocols will be **new** and (hopefully) better organized
- The **new** material is part of any modern **software engineering** development cycle
  - Specification
  - Description
  - Validation
- And..., we will also discuss a **practical protocol design** methodology (courtesy of OPNET)
  - This methodology is used in practice for the design and implementation of ANY state dependent software (or algorithmic solution)
Lecture overview

- Introduction to protocols
- Protocol Design Methodology - A practical method
Protocol Structure

- Let's start with a small example

Assume two communicating computers

- Computer A is connected to a storage device D
  - This is the file server
- Computer B is connected to a printer P
  - This is the print server
- We want to send a file stored at D to the printer P
Protocol Structure

- To communicate, the two computers must use
  - same **physical** wires
  - compatible character **encodings**
  - transmit and scan **signals** on the wires at roughly the same speed

Is this enough for correct communication?
Protocol Structure

- computer A must be able to
  - check if the printer is available
  - adapt the sending rate of the characters to the capability of the printer
  - suspend sending if the printer is
    - out of paper
    - switched off by someone

Q: The words colored with red denote...??
Protocol Structure

- The data flows only in one direction \( A \rightarrow B \)
- The control information is two way: \( A \leftrightarrow B \)
  - The two computers need an apriory agreement on
  - the meaning of control information
  - procedures to start, stop, suspend, resume and conclude transmissions
- In case of channel errors
  - control information must be exchanged to guard the transfer of data
Protocol

All rules, formats and procedures are called collectively as a Protocol

- The protocol formalizes the interaction by standardizing the use of the communication channel.
- The protocol can contain agreements on the method used for:
  - Initiation and termination of data exchanged
  - Synchronization of senders and receivers
  - Detection and correction of transmission errors
  - Formatting and encoding of data
- The protocol can take place between two or more parties:
  - N parties $\Rightarrow$ N-peer communication and N-peer protocol
Sample levels of abstraction

- electrical signal
- bits
- symbols/characters
- message fields
- frames/packets
The five elements of a protocol

1. The **service** to be provided by the protocol
2. The **assumptions** about the environments in which the protocol is executed
3. The **vocabulary** of messages used to implement the protocol
4. The **encoding** (format) of each message in the vocabulary
5. The **procedure rules** guarding the consistency of message exchange

The fifth element is the **MOST DIFFICULT** to design and hardest to verify
Element 1 - Service specification

The purpose of the protocol is to transfer text files as sequences of characters across a telephone line

- while protecting against transmission errors
  - assuming that all transmission errors can in fact be detected

The protocol is defined for full-duplex file transfer

- it should allow for transfers in two directions simultaneously
- Positive and negative acknowledgments for traffic from A to B are sent on the channel from B to A, and vice versa
- Every message contains two parts: a message part, and a control part that applies to traffic on the reverse channel
Element 2 - Assumptions about the environment

Consists minimally of

- two users of the file transfer service
  - assumed to simply submit a request for file transfer and await its completion

- a transmission channel.
  - assumed to cause arbitrary message distortions, but not to lose, duplicate, insert, or reorder messages.
  - In protocol modeling one usually assumes here that a lower-level module is used to catch all distortions and change them into undistorted messages of type error.
Element 3 - The **vocabulary** of messages used to implement the protocol

- **ack** - for a message combined with a positive acknowledgment
- **nack** - for a message combined with a negative acknowledgment
- **err** - for a message with a transmission error

**Vocabulary**

\{ack, nack, err\}
Element 4 - The **encoding** (format) of each message in the vocabulary

- Control field - identifying the message type
- Data field - with the character code.

Assuming a fixed size data and control fields a C-like specification would be

```c
enum control { ack, nak, err };
struct message {
    enum control tag;
    unsigned char data;
};
```
Element 5 - Procedure Rules

1. If the previous reception was error-free, the next message on the reverse channel will carry a positive acknowledgment; if the reception was in error it will carry a negative acknowledgment.

2. If the previous reception carried a negative acknowledgment, or the previous reception was in error, retransmit the old message; otherwise fetch a new message for transmission.

How do we formalize these rules???

- State transition diagram
- A formal language
  - Final State machine
  - CSP, CCS, PROMELA, ESTEL, ...
Procedure Rules

A finite state machine (FSM) is a model of behavior composed of a finite number of states, transitions between those states, and actions. A finite state machine is an abstract model of a machine with a primitive (sometimes read-only) internal memory.
Finite State Machines

What is a finite state machine?

A finite state machine (FSM) is a model of behavior composed of a finite number of states, transitions between those states, and actions. A finite state machine is an abstract model of a machine with a primitive (sometimes read-only) internal memory.
The concept of FSMs

- **A current state** is determined by past states of the system (recording information about the past), therefore, reflects the input changes from the system start to the present moment.

- **A transition** indicates a state change and is described by a condition that would need to be fulfilled to enable the transition.

- An action is a description of an activity that is to be performed at a given moment.
Action types

- **Entry action** - performed when entering the state
  - common to all inputs

- **Exit action** - performed when exiting the state
  - common to all outputs

- **Input action** - performed depending on present state and input conditions

- **Transition action** - performed when performing a certain transition
FSM vs. Event-Driven FSM

- **Finite-state machine** - the machine is described as consuming characters or tokens.

- **Petri net** is a well known FSM that uses tokens.

- An FSM is event driven if the FSM is consuming events or messages.
  - Often these machines are implemented as threads or processes communicating with one another as part of a larger application.
  - For example, an individual car in a traffic simulation might be implemented as an event-driven finite-state machine.

The distinction between the types is not of importance in this course.
FSM representations

- A state diagram (or state transition diagram).
- State transition table (several types).
  - The most common representation is to show the combination of current state \((B)\) and condition \((Y)\) shows the next state \((C)\).
  - The complete actions information can be added only using footnotes.
**State Tables**

An FSM definition including the full actions information is possible using state tables.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Present State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>State A</td>
</tr>
<tr>
<td>Condition 1</td>
<td>...</td>
</tr>
<tr>
<td>Condition 2</td>
<td>...</td>
</tr>
<tr>
<td>Condition 3</td>
<td>...</td>
</tr>
</tbody>
</table>
Transducer states

- Transducers generate output based on a given input and/or a state using actions. Two types are distinguished
  
  - Moore machine
  - Mealy machine
**Moore Machine**

- The FSM uses only entry actions.
- The output depends only on the state
  - Advantage: simplification of the behavior
Example - An elevator door

- The state machine recognizes two commands: "command_open" and "command_close" which trigger state changes.
- The entry action (E:) in state "Opening" starts a motor opening the door.
- The entry action in state "Closing" starts a motor in the other direction closing the door.
- States "Opened" and "Closed" don't perform any actions. They signal to the outside world (e.g., to other state machines) the situation: "door is open" or "door is closed" (signaling is in the sense that the state can be checked Format the outside).
Mealy Machine

- The FSM uses only input actions: output depends on input and state.
  - The use of a Mealy FSM leads often to a reduction of the number of states.
  - Mealy FSM implementation of the same behavior as in the Moore example (the behavior depends on the implemented FSM execution model and will work).
- There are two input actions (I:)
  - "start motor to close the door if command_close arrives"
  - "start motor in the other direction to open the door if command_open arrives".
- The "opening" and "closing" intermediate states are not shown.

In practice mixed models are often used
How do we design communication protocols using FSMs

- Here we model a process (without defining what a process is)
- We will present a method having several steps - the Process Modeling Methodology
- The FSM model is the closest to the software implementation
  - The FSM design can be converted automatically to a skeleton of software
  - Modern systems combine the two to get an integrated development system (OPNET)
What is the Process Modeling Methodology (PMM)?

- Systematic approach to creating process models
- Quick and efficient method of development
- Protects from some common pitfalls
- Produces consistent results
- Indispensable for larger models
Let's take one computer in a switched LAN.
We are interested with a specific layer of this computer.
The layer can be represented as a FSM

- FSM shows the logic
- The details are in software
What is behind the code of a FSM?

The following C code describes the state machine for a British traffic light, which follows the pattern

\[ \text{red} \rightarrow \text{red + yellow} \rightarrow \text{green} \rightarrow \text{yellow} \rightarrow \text{red} \]
### Important definitions - States, Events and Actions

<table>
<thead>
<tr>
<th>State</th>
<th>Event</th>
<th>Action</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>asleep</td>
<td>Alarm</td>
<td></td>
<td>awake</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Wakeup</td>
<td></td>
</tr>
</tbody>
</table>

- **Condition of system**
- **Stimulus to move system**
- **Response of system to stimulus**
- **Condition of system**

The following C code describes the state machine for a British traffic light, which follows the pattern:

- red
- red
- yellow
- green
- yellow
- red
Motivation

- PMM helps the user over many common pitfalls
  - Gets you passed the initial blank screen
    - “Where do I even begin?”
  - Reveals unexpected relationships
    - “Oh yeah... I guess that CAN happen”
  - Avoids building yourself into a corner
    - “I have to start from scratch. There goes 40 hours of my life”
Steps for designing a Process Model

- Obtain protocol specification
- Design using the methodology
- Review design
- Implementation (usually programming)
Process Modeling Methodology Stages

First five: Design

1. Context definition
2. Process level decomposition
3. Enumeration of events (per process)
4. State-level decomposition (per process)
5. State transition diagram development (per process)

Last two: Implementation

6. Specification of process actions (per process)
7. Initial state designation (per process)
Process Modeling Methodology **Example:**
The Stop-and-Wait Retransmission Protocol

- Protocol for the Data Link Layer at the sending side only
- Basic functionality: provides reliable communications over a lossy channel
- Accept frames from the upper layer and send them to the physical layer
Protocol Modeling Methodology Example: Stop-and-Wait Retransmission Protocol Requirements

- Receive frames from higher layer
- Transmit frames over physical layer
- Must receive an acknowledgment before sending the next frame
- Queue frames arriving from the higher layer if the process is waiting for acknowledgment for a previous frame
- Retransmit the frame if an acknowledgment is not received before the end of a timeout period
- When the link fails, frames may not be sent or retransmitted until the link recovers

What is common to all highlights?
Stage 1 - Context Definition:
Fit this piece into the big picture

- Step 1: Identify interdependent modules
  - equivalent to communication layers: Network, MAC, Phy

- Step 2: Select communication mechanisms with interdependent modules

- Step 3: Develop diagram of system and interdependent modules
Stage 1 - Context Definition (cont.)

Higher Layer

DLC Protocol

Physical Layer

Frames for Transmission

Acknowledgements

Frames for Transmission
Stage 2: Process-Level Decomposition

1. Determine which process decomposition technique is applicable to the system
   - Single process (a single FSM) or Multiple processes (multiple FSMs)

2. If a multi-process implementation is chosen, identify the areas of responsibility assigned to each process (important)

3. For multi-process implementations, determine circumstances of process creation and which process will be the root

Use a single process for this example
Stage 3 - Enumeration of Events

1. Define **logical events** of each process
2. Select event implementation methods

---

**DLC Protocol**

- **Frame Arrival**: A frame has arrived from the higher layer.
- **Timeout**: The retransmission timer has expired. The frame needs to be retransmitted.
- **Acknowledgment Received**: An acknowledgment for the last frame sent has been received.
- **Link Down**: The link has failed. No packets may be sent or retransmitted.
- **Link Up**: The link has recovered.

**Network Protocols**
Stage 3 - Enumeration of Events (cont.)

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Event Description</th>
<th>Interrupt Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame Arrival</td>
<td>A frame has arrived from the higher layer.</td>
<td>Stream</td>
</tr>
<tr>
<td>Timeout</td>
<td>The retransmission timer has expired. The frame needs to be retransmitted</td>
<td>Self</td>
</tr>
<tr>
<td>Acknowledgment Received</td>
<td>An acknowledgment for the last frame sent has been received.</td>
<td>Stream</td>
</tr>
<tr>
<td>Link Down</td>
<td>The link has failed. No packets may be sent or retransmitted.</td>
<td>Failure</td>
</tr>
<tr>
<td>Link Up</td>
<td>The link has recovered.</td>
<td>Recovery</td>
</tr>
</tbody>
</table>
Stage 4 - State-Level Decomposition of Processes

1. Select states of the process

2. Verify that the selected set of states match the following criteria:
   2.1. Each state should represent a blocking point of the process
   2.2. A state corresponds to particular sequences of events having occurred
   2.3. Events are handled in a specific manner as a result of being located in a particular state
   2.4. Selected states are mutually exclusive and complementary
Stage 4. State-Level Decomposition of Processes (cont.)

<table>
<thead>
<tr>
<th>State Name</th>
<th>State Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Initial state. The process is waiting for frame arrival from the higher layer.</td>
</tr>
<tr>
<td>Ack Wait</td>
<td>State to handle the sending of frames to the physical layer. The process waits for an acknowledgment or a timeout.</td>
</tr>
<tr>
<td>Link Down</td>
<td>The link has failed. No frames can be sent until the link recovers.</td>
</tr>
<tr>
<td>Ack Wait and Link Down</td>
<td>The link has failed while the process is waiting for an acknowledgment. No frames can be sent or retransmitted until the link recovers.</td>
</tr>
</tbody>
</table>
Stage 5 - State Transition Diagram Development

1. Determine feasible events of each state

- **State-Event pair types:**
  - Feasible (expected)
  - Infeasible (not possible)
  - Suppressed (not desired)

Practically, this is a Cartesian (cross) product (\(\times\)) between the set of states and the set of events (covering all possibilities).

Each (state, event) pair is carefully considered for feasibility and/or suppression.

The decisions in this step must be taken by a human being, they cannot be automated.
Stage 5 - State Transition Diagram Development (cont.)

2. Construct event-response table

- Keep only feasible state-event pairs
- Determine what happens in each case
Stage 5. State Transition Diagram Development:
Step 1 - Determining Feasible Events (cont.)

<table>
<thead>
<tr>
<th>State Name</th>
<th>Logical Event</th>
<th>Feasibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Frame Arrival</td>
<td>Feasible</td>
</tr>
<tr>
<td></td>
<td>Timeout</td>
<td>Infeasible</td>
</tr>
<tr>
<td></td>
<td>ACK Received</td>
<td>Infeasible*</td>
</tr>
<tr>
<td></td>
<td>Link Up</td>
<td>Infeasible*</td>
</tr>
<tr>
<td></td>
<td>Link Fail</td>
<td>Feasible</td>
</tr>
</tbody>
</table>

* Possible case for suppression - in practice you may need to ask the systems engineer
Stage 5. State Transition Diagram Development:
Step 1 - Determining Feasible Events (cont.)

<table>
<thead>
<tr>
<th>State Name</th>
<th>Logical Event</th>
<th>Feasibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ack Wait</td>
<td>Frame Arrival</td>
<td>Feasible</td>
</tr>
<tr>
<td></td>
<td>Timeout</td>
<td>Feasible</td>
</tr>
<tr>
<td></td>
<td>ACK Received</td>
<td>Feasible</td>
</tr>
<tr>
<td></td>
<td>Link Up</td>
<td>Infeasible*</td>
</tr>
<tr>
<td></td>
<td>Link Fail</td>
<td>Feasible</td>
</tr>
</tbody>
</table>

* Possible case for suppression - in practice you may need to ask the systems engineer
Stage 5. State Transition Diagram Development:
Step 1 - Determining Feasible Events (cont.)

<table>
<thead>
<tr>
<th>State Name</th>
<th>Logical Event</th>
<th>Feasibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link Down</td>
<td>Frame Arrival</td>
<td>Feasible</td>
</tr>
<tr>
<td></td>
<td>Timeout</td>
<td>Infeasible</td>
</tr>
<tr>
<td></td>
<td>ACK Received</td>
<td>Infeasible*</td>
</tr>
<tr>
<td></td>
<td>Link Up</td>
<td>Feasible</td>
</tr>
<tr>
<td></td>
<td>Link Fail</td>
<td>Infeasible*</td>
</tr>
</tbody>
</table>

* Possible case for suppression - in practice you may need to ask the systems engineer
Stage 5. State Transition Diagram Development:
Step 1 - Determining Feasible Events (cont.)

<table>
<thead>
<tr>
<th>State Name</th>
<th>Logical Event</th>
<th>Feasibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ack Wait and Link Down</td>
<td>Frame Arrival</td>
<td>Feasible</td>
</tr>
<tr>
<td></td>
<td>Timeout</td>
<td>Feasible</td>
</tr>
<tr>
<td></td>
<td>ACK Received</td>
<td>Infeasible*</td>
</tr>
<tr>
<td></td>
<td>Link Up</td>
<td>Feasible</td>
</tr>
<tr>
<td></td>
<td>Link Fail</td>
<td>Infeasible*</td>
</tr>
</tbody>
</table>

* Possible case for suppression - in practice you may need to ask the systems engineer
Stage 5. State Transition Diagram Development:  
Step 1 - Keeping the Feasible Events only

<table>
<thead>
<tr>
<th>State Name</th>
<th>Logical Event</th>
<th>Feasibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Frame Arrival</td>
<td>Feasible</td>
</tr>
<tr>
<td></td>
<td>Link Fail</td>
<td>Feasible</td>
</tr>
<tr>
<td>Ack Wait</td>
<td>Frame Arrival</td>
<td>Feasible</td>
</tr>
<tr>
<td></td>
<td>Timeout</td>
<td>Feasible</td>
</tr>
<tr>
<td></td>
<td>ACK Received</td>
<td>Feasible</td>
</tr>
<tr>
<td></td>
<td>Link Fail</td>
<td>Feasible</td>
</tr>
<tr>
<td>Link Down</td>
<td>Frame Arrival</td>
<td>Feasible</td>
</tr>
<tr>
<td></td>
<td>Link Up</td>
<td>Feasible</td>
</tr>
<tr>
<td>Ack Wait and Link Down</td>
<td>Frame Arrival</td>
<td>Feasible</td>
</tr>
<tr>
<td></td>
<td>Timeout</td>
<td>Feasible</td>
</tr>
<tr>
<td></td>
<td>Link Up</td>
<td>Feasible</td>
</tr>
</tbody>
</table>

This Top-down design finds all cases.
Stage 5. State Transition Diagram Development:
Step 2 - Event Response Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Logical Event</th>
<th>Condition</th>
<th>Action</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Stage 5. State Transition Diagram Development:
Step 2 - Event Response Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Logical Event</th>
<th>Condition</th>
<th>Action</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Frame Arrival</td>
<td></td>
<td>Copy the frame</td>
<td>Ack Wait</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Send copy to physical layer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Set the timer</td>
<td></td>
</tr>
<tr>
<td>Link Fail</td>
<td>None</td>
<td></td>
<td></td>
<td>Link Down</td>
</tr>
</tbody>
</table>
### Stage 5. State Transition Diagram Development: Step 2 - Event Response Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Logical Event</th>
<th>Condition</th>
<th>Action</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ack Wait</td>
<td>Frame Arrival</td>
<td>Queue frame</td>
<td>Ack Wait</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Link Fail</td>
<td>None</td>
<td>Ack Wait and Link Down</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ACK Received</td>
<td>Queu is empty</td>
<td>• Cancel timer</td>
<td>Idle</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Destroy frame</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Queue is not empty</td>
<td></td>
<td>• Cancel timer</td>
<td>Ack Wait</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Destroy frame</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Remove next frame from queue</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Copy frame</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Send copy to physical layer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Set the timer</td>
<td></td>
</tr>
<tr>
<td>Timeout</td>
<td></td>
<td></td>
<td>• Copy frame</td>
<td>Ack Wait</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Send copy to physical layer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Set the timer</td>
<td></td>
</tr>
</tbody>
</table>
Stage 5. State Transition Diagram Development:
Step 2 - Event Response Table (cont.)

<table>
<thead>
<tr>
<th>Current State</th>
<th>Logical Event</th>
<th>Condition</th>
<th>Action</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link Down</td>
<td>Frame Arrival</td>
<td>Queue is empty</td>
<td>Queue frame</td>
<td>Link Down</td>
</tr>
</tbody>
</table>
|               | Link Up       | Queue is not empty | • Remove next frame from queue  
|               |               |                 | • Copy the frame                            |            |
|               |               |                 | • Send copy to physical layer               |            |
|               |               |                 | • Set the timer                             | Ack Wait   |
|               |               |                 |                                             | Idle       |
Stage 5. State Transition Diagram Development:
Step 2 - Event Response Table (cont.)

<table>
<thead>
<tr>
<th>Current State</th>
<th>Logical Event</th>
<th>Condition</th>
<th>Action</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ack Wait and Link Down</td>
<td>Frame Arrival</td>
<td>Queue frame</td>
<td>Ack Wait and Link Down</td>
<td></td>
</tr>
<tr>
<td>Ack Wait and Link Down</td>
<td>Timeout</td>
<td>Set variable to indicate that frame must be resent</td>
<td>Ack Wait and Link Down</td>
<td></td>
</tr>
<tr>
<td>Link Up</td>
<td>Frame needs to be transmitted</td>
<td>Copy frame, Send copy to physical layer, Set the timer</td>
<td>Sending</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Frame does not need to be retransmitted</td>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Stage 6 - Specification of Process Actions

1. Review specification of the process' logical actions for completeness
2. Place logical actions within the State Transition Diagram (STD)
3. Implement STD in your specific environment (not part of this course)
4. Define or replace macros and pseudo-code (again, not part of the course)
Stage 6. Specification of Process Actions

Step 2 - Place actions in STD

Actions can be contained in three different places

- Leaving the current state - Exit Executives
- Going from one state to the next - Transition Executives
- Entering the new state - Enter Executives
### Stage 6. Specification of Process Actions

**Step 2 - Place actions in STD (cont.)**

<table>
<thead>
<tr>
<th>Exit Executive</th>
<th>Contains actions common to all outgoing transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter Executive</td>
<td>Contains actions common to all incoming transitions</td>
</tr>
<tr>
<td>Transition Executive</td>
<td>Contains actions associated with a subset of outgoing and incoming transitions</td>
</tr>
</tbody>
</table>

**Recommendation** - Start by placing all actions in transition executives. Then, actions common to all outputs or inputs can be placed in exit or enter executives.
Stage 6. Specification of Process Actions

Step 2 - Place actions in STD (cont.)

**RED** states are true states of the system.

**GREEN** states are for coding convenience.
Stage 6 - Specification of Process Actions
Step 3 - Implement STD in OPNET

<table>
<thead>
<tr>
<th>State Name</th>
<th>Enter Executives</th>
<th>Exit Executives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Initialize state variable indicating whether a frame needs to be resent</td>
<td>None</td>
</tr>
<tr>
<td>Send</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Link Down</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Sending and Link Down</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
Stage 6. Specification of Process Actions
Step 3 - Implement STD in OPNET (cont.)
Stage 6. Specification of Process Actions
Step 3 - Implement STD in OPNET (cont.)

<table>
<thead>
<tr>
<th>Current State</th>
<th>Logical Event</th>
<th>Condition</th>
<th>Action</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Frame Arrival</td>
<td></td>
<td>• Copy the frame</td>
<td>Ack Wait</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Send copy to physical layer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Set the timer</td>
<td></td>
</tr>
<tr>
<td>Link Fail</td>
<td>None</td>
<td></td>
<td></td>
<td>Link Down</td>
</tr>
</tbody>
</table>
Stage 6. Specification of Process Actions
Step 3 - Implement STD in OPNET (cont.)
### Stage 6. Specification of Process Actions

#### Step 3 - Implement STD in OPNET (cont.)

<table>
<thead>
<tr>
<th>Current State</th>
<th>Logical Event</th>
<th>Condition</th>
<th>Action</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ack Wait</td>
<td>Frame Arrival</td>
<td>Queue frame</td>
<td>Ack Wait</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Link Fail</td>
<td>None</td>
<td>Ack Wait and Link Down</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Queue is empty</td>
<td>• Cancel timer</td>
<td>Idle</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Queue is not empty</td>
<td>• Destroy frame</td>
<td>Ack Wait</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ACK Received</td>
<td>• Remove next frame from queue</td>
<td>Ack Wait</td>
<td></td>
</tr>
</tbody>
</table>
Stage 6. Specification of Process Actions
Step 3 - Implement STD in OPNET (cont.)
Stage 6. Specification of Process Actions
Step 3 – Implement STD in OPNET (cont.)
Stage 7 - Initial State Designation

In which state will the STD begin the simulation?
Complete State Transition Diagram (STD) - OPNET Implementation

Diagram:

- States: idle, ACK Wait, link_down, down

- Transitions:
  - (TIMEOUT)/RETRANSMIT_FRAME
  - (PK_ARRIVAL)/SEND_PACKET
  - (ACK & Q_EMPTY)/ACK_HANDLE
  - (PK_ARRIVAL)/ENQUEUE_PACKET
  - (LINK_FAIL)
  - (LINK_UP & Q_EMPTY)
  - (LINK_UP & 1Q_EMPTY)/SEND_PACKET_FROM_QUEUE
  - (LINK_UP)/RETRANSMIT_FRAME_IF_NECESSARY
  - (PK_ARRIVAL)/ENQUEUE_PACKET
  - (ACK)/ACK_HANDLE
  - (TIMEOUT)/SET_FRAME_TO_RETRANSMIT

- Notes:
  - Copyright © 2000 OPNET Technologies, Inc.
Alternate Solution: Using Forced States