Chapter 4: Protocol Specification

1. Introduction

Objective: To unambiguously describe the operation of protocols in a manner that allows us to analyze their operation and to verify that they perform all of the functions that are required.

Protocols are distributed operations.

- Parts of the protocol are executed in different places (on different machines).
- There are multiple machines that interact. For example the finite state machine for the transmitter and receiver in the ARQ protocol.
- When we model the interaction between the different machines, we must also consider the operation and properties of the physical communication channel between the transmitter and receiver. In this section the communication channel may accept a message from one of the machines and either deliver it to the other machine, or discard the message. Unlike the protocol machines, the operation of the communications channel is not deterministic. Losing messages adds a random component to the operation.

The machines can perform some operations independently, at the same time, and other operation are coordinated.

- **Concurrency**: operations can be carried out at the same time on the different machines.
- **Interaction**: coordination or synchronization - execution cannot progress without coordination with other machines

Two possible models of communicating machines are:

1. CSP: Communicating Sequential Processes:
   - When a message is to be transmitted from one machine to another machine, the transmitting machine puts the message into a FIFO queue and proceeds until it reaches a state where it must wait for a message from another machine.
   - When a machine is in a state where it expects to receive a message, it takes the first message from the queue.
     - If there are no messages in the queue, the process waits

2. Rendezvous model
   - A machine stops when it either expects to receive or transmit a message to another machine.
     - a rendezvous point
   - The protocol proceeds when a receive and transmit line up in two machines
     - Machine A would like to send a message to machine B, and machine B expects to receive a message from machine A

Progression:

1. Flow chart description of a protocol: Clear overview of the operation of a protocol
2. Pseudo Code of protocol:
   - Precise. Programs operate the same way every time. The code can be executed.
   - Very flexible but very difficult to prove that the interactions between the communicating machines are correct.
• There have been significant advances in program verification, but this type of verification has different objectives than protocol verification

• Program verification is usually concerned with the operation of an individual program. It considers both the control and data operations in the software. Especially when the data has a large number of values, there can be a very large number of possible sequences.

• Protocol verification usually only considers the interactions between processes. The control portion of a program, rather than the results that data processing can have. The number of sequences of operations is much smaller.

When continuous values of time are added to a protocol, and we are concerned with the sequence of executions or the time between executions, protocol verification becomes similar to program verification. In that there are a very large number of sequences that can occur. Work on time in protocols has been concerned with reducing an potentially infinite number of timed sequences. For instance, we may only be concerned with the order of sequences, or the result of the interaction may only differ if a first operation occurs less than T seconds before a later operation or more than T seconds before the operation. (If a following car applies his brakes less than T seconds after a lead car a crash may be avoided, but if the second car applies his brakes more than T seconds after the lead car, there will be a crash.)

3. Communicating Finite State Machines
   — Implement CSP or rendezvous model on FSM
   — Much more restricted set of operations than a general programming language
   — Possible to prove properties using graph theory

4. Extended Finite State Machines
   — Between Finite state machine and programming language
     • Variables can be transferred between communicating machines to distinguish the messages or to control the operation of the receiver. For instance, in the go back N protocol, we can have one acknowledgement message with a variable indicating the number, rather than N different acknowledgement messages.
     • The program can have parameters that affect the operation of the machine. For instance on the first N retries of a message transmission we may retransmit the message, but on the \((N+1)\)th retry we may give up and do something different. The parameter makes the state machine more compact. It also allows us to change N without changing the machine. For instance, on a noisy wireless link we may increase the number of times that we retry a transmission.
     • The parameters in an EFSM usually have a restricted set of values, instead of the large number of values that data in programs can have. Usually, the parameters are included to give the protocol a more compact graphical representation, but the EFSM can be expanded to a FSM with a larger number of states for testing.

5. Petri Nets
   — Alternative to FSM’s to model interaction and concurrency

6. Composite Finite State Machines
   — Implement coordination by considering interaction of several machines
Homework:  
Write a 1/2 to 1 page summary of reference 1 and decide which sections you would like to outline, and why.

We will go over the summaries in 1 week, and decide which sections we will outline in the second phase reading.

2. A Specification Description Language - SDL

SDL, Reference 2, is a flowchart description language that was first standardized by the CCITT, Comité Consultatif International Téléphonique et Télégraphique. CCITT has been the standards organization for telecommunications since 1865, and has since adopted the name of its parent organization the ITU, International telecommunications union.

2.1 Description

— Each flowchart contains a process that is executed simultaneously with other processes.
— Each flowchart has a single entry point that is labelled as "start" or by the process name.
— The processes are specified by symbols of various shapes that are connected together by arcs.
— The six basic symbols are:
  Other symbols are being added to make it easier to describe some functions. See reference [1].

1. Statements: Assignments, such as changing the message number
2. Boolean Tests: True/False tests, such as is the current message equal to message 1 
   Evaluated without delay
3. Wait Conditions: Waiting to receive a message 
   Used for process synchronization
   Can contain a boolean expression, in which case the process waits until the boolean expression is true
4. Internal Events: Such as a timer 
   If a timeout follows a wait, it can abort the wait
5. Input: Receive a message 
   Can only follow a wait symbol "receive" in the flow chart
6. Output: Transmit a message

   — The two remaining flow chart elements, used to connect the 6 symbols are

   1. Directed Arcs

   2. Connectors: Two arcs can converge at a connector. Arcs can diverge without a connector.
      For instance at a wait we may leave the wait when we receive a message or when a timeout occurs

   Each flow chart has an implicit infinite queue where incoming messages are stored.

   2 special internal events are

   1. retrieve the next message to transmit, next:a,b, and

   2. accept the next message at the destination, accept:a,b

   Note that the messages are labelled a,b.

   The reason for two designations is that "a" is the actual sequence number for the message, and "b" is
   the sequence number that is used to identify the message when it is transmitted.

   In the alternating bit protocol, "a" is on the field of integers, and "b" is 0 or 1.
2.2 Example
Stop and Wait ARQ with both messages and acknowledgements numbered

---

sender

1. next:o

2. timeout

3. msg:o,s

4. receive

5. ack:r

6. \( r = = s \)

7. true

8. s = s+1 mod 2

9. false

receiver

10. receive r

11. a = e

12. false

13. accept:i

14. true

15. e = e+1 mod 2

---
Homework:

1. Draw the flow chart for the go-back-n protocol when the messages are numbered on the field of real numbers.

Simplifications:

A. A single timer. Retransmit from last ack’ed message if there are no new messages ack’ed for time T. We could also interpret the timeout to require that we retransmit an unacknowledged message at time T after it is transmitted, but this requires individual timers for each outstanding message.

B. The application can receive messages as quickly as they arrive. If this is not the case, we must set up a receive buffer, and stop acknowledging messages when the buffer is full. This mode of operation can be modelled with a second concurrent process at the receiver.

C. At the transmitter the process next:a gets the message a if the application has generated the message. This instruction is used to both get new messages and get previously transmitted messages from a buffer that retains unacknowledged messages.
3. Pseudo Code

- Reference 3

- Pseudo Code is a less ambiguous way of describing the operation of a protocol than English. The sample protocol is the stop-and-wait ARQ protocol.

- Usually close to a C or Pascal program.

3.1 Example

Simple Acknowledgement Protocol

- The messages are numbered mod 2
- The acknowledgements are not numbered.

We have shown that this protocol does not operate correctly.

A more detailed examination shows that the protocol operates correctly on a half-duplex channel, but not on a full duplex channel.

**XMITTER**

*Begin*

ID = 0

Frame = First packet to send

*Repeat*

Msg = ID, Frame /* Construct the message to transmit

Xmit Msg

Wait(event) /* possible events are Rcv Ack, Timeout

*If* event==Ack *then*

Frame = Next Packet

ID := ID +1 mod 2

*end*

*until* doomsday

*end*
RCVR

Begin

Expect_ID = 0

Repeat

wait(event) /* Only event is Msg Rcvd, msgs with checksum errors are discarded and do not generate an event

Xmit Ack

If Rcv_msg.ID == Expect_ID then

Send Rcv_msg.Frame to Network Layer

Expect_ID := Expect_ID +1 mod 2

end

Until doomsday

end

Homework:

1. Use pseudo-code to specify the transmitter and receiver of the "Go-Back-N" protocol when:
   A. The messages and acknowledgements are numbered on the field of real numbers, and
   B. The messages and acknowledgements have numbers on a finite field with N+1 numbers.

2. Use pseudo-code to specify the transmitter and receiver of the "Selective Repeat" protocol when the messages and acknowledgments are numbered on the field of real numbers.
4. Communicating Finite State Machines

Yuang [4]

Protocol Specification using Communicating FSM models

Each process is represented by a communicating FSM

A directed, labeled graph

Nodes: States where concurrent processing is done

Edges: Transitions where interactions between the states occur

Edge Labels: Messages transmitted or received by a process

"-" message transmitted

"+" message received

Consumer-producer model

— The producer produces data and places it in a buffer, as long as a buffer is available. If there are no buffers available, the consumer waits

— The consumer takes data out of a buffer, making the buffer available for the producer.

— This protocol operates in a layer above an ARQ protocol. The messages are all received correctly in the correct order.

— Interaction

— The consumer cannot take data when the buffers are all empty

— The producer cannot create data when all of the buffers are full

— Concurrency:

• The producer and consumer operate independently when the buffers are partially full

— In ARQ protocols

• The source produces messages and consumes acknowledgement.

• The receiver consumes messages and produces acknowledgements.

• In a system with a lossy channel

— The source produces messages that are consumed by the forward channel.

— The forward channel either losses the message or produces the message for the receiver.

— The receiver produces acknowledgments consumed by the reverse channel.

— The reverse channel produces acknowledgments for the source.

• A stop-and-wait protocol mostly has interaction.

• A go-back-N protocol can have concurrency.
4.1 Example

Correctly Designed Protocol

Both processes start in state 1

In each state either P1 sends a message that P2 is expecting or P2 is sending a message that P1 is expecting

The next state is always the proper one

Incorrectly designed protocol

There are sequences of inputs and outputs that cannot be executed.

This protocol will be used for illustrative purpose in the verification section.
5. Finite State Machines

5.1 Translating the pseudo code to a finite state machine

Abstract out the important states

— Start by writing down all of the operations that are performed
— Simplify the machine by identifying the states that are transitory - states that we pass through, but do not stay in.
— The simpler we can make the machine, the easier it will be to analyze.

The operation of the transmitter is changed so that the first $M_0$ is transmitted after a timeout.

Receiver

Xmitter
5.2 Formal Definition of an FSM

Reference 5, Chap 1.4: Modeling Communication Protocols

— An automaton with a finite number of states which changes its state when an external stimulus is applied

— An FSM is a 4-tuple, < S,I,O,T > , where

  • S= the set of states in the machine
    A state is a stable condition where the FSM rests until an external stimulus, an input, is applied

  • I= the set of inputs that cause the machine to change states
    An input causes a FSM to generate an output, which may be null, and to change states - possibly to the same state.
    The initial state is the state that the FSM resides in when the power is turned on.

  • O= the set of outputs that the machine generates when it changes states
    For each state there is a set of permissible inputs I and a set of permissible outputs, O.
    O may include the null output, but I does not contain a null input.

  • T= the set of arcs in the graph that move the machine from one state to another state

    \[ T = \bigcup_i \bigcup_j < s_i, i_j > \rightarrow < n, o_j > \]

    where \( s_i, n \in S; i_j \in I, o_j \in O \)

— An FSM is fully specified if \( I_j \) is the same in every state, and for each \( i \in I \) an output is specified.

— An FSM is partially specified if some inputs are not specified in some states.

— An FSM is minimal if the specification does not have 2 or more equivalent states.

— An FSM is deterministic if its output and the next state is a function of its current state and the input - otherwise it is non-deterministic.

Homework:

1. Consider a stop-and-wait ARQ protocol that transmits messages \( M_0 \) or \( M_1 \) and receives acknowledgements \( A_0 \) and \( A_1 \). (Both messages and acknowledgements are numbered)

   A. Draw the component finite state machine (FSM) for the source.

      — The set of inputs is \( \{ T, A_0, A_1 \} \), where \( T \) is the timeout.

      — The set of outputs is \( \{ M_0, M_1, Null \} \), where Null indicates that there is no output.

      — Label each transition with input/output to indicate the input that causes the transition and the output that results.

      — Show what happens for every possible input at every state

      — There should be NO transient states.

      — Assume that \( M_0 \) is initially transmitted before entering the first state, numbered 0, and that there are always more messages to transmitted.

   B. Draw the FSM when the source attempts to transmit a message at most 3 times before entering state "R" where it requests assistance from another protocol or the operator.

2. Reference 5 chapter 1. This is on the course WEB site as Uyar_chap.1.pdf.
A. Level 1: Write a 1/2-3/4 page summary.
B. Level 2: Write a 1-2 page outline section 4.
6. Extended Finite State Machine

Reference 5, Chap 1.4: Modeling Communication Protocols

6.1 Reason for Extending the FSM model

— Extended finite state machines are a more compact way of drawing the protocol graphs that have a large number of states that are very similar.

— This is accomplished by letting the protocol store local values and to make a transition based on the local value.

  • For instance, in practical ARQ protocols we do not try retransmitting a lost message forever, but try it $N$ times then enter a recovery process where we may test the channel or notify a human operator.

— The timeout in the transmitter is also a local variable, but it is a different type of local variable than the message count.

  The timeout can be considered a spontaneous local variable, since it can cause a transition when no other external event causes a trigger to change protocol states.

  Timeouts can also be modeled as a separate process

    - a message from the first process starts the timer or turns off the timer
    - the timer sends a message to the first process that supplies the trigger

    This model is much more complex

— The message can also carry a variable that the protocol uses to determine the action that it will take

  • For instance, in the ARQ protocol the receiver has a local variable that is the message that it expects, and the message has a variable that indicates the message number.

  • The receiver can compare the message number with the number that it expects, send an ACK in either case, and only change the message number if the received message number is the same as the number that it expects.

  • The source in the ARQ protocol can also maintain a local variable that is the message.

    — There can be only 1 message type, with the local variable as a message parameter.

    — This would result in a 1-state machine for the source.

    — This machine is much more difficult to understand because all of the information is hidden in variables.
6.2 Formal Definition of a EFSM

The formal specification of an EFSM is a 7-tuple: EFSM = <S, I, O, V, P, B, T>, where

- S is the set of states as in the FSM
- V is a set of variables that are stored at a component machine.
  For example, a variable at the receiver in the go-back-N protocol is the next message expected.
- P is a set of parameters associated with the inputs
  For example, the messages in the go-back-N protocol contain the message number
- I is the set of inputs, \( i_j \) with parameters \( p_{j,k} \), that can be received by a component machine.
  \[
  I = \bigcup_j i_j \bigcup_k p_{j,k}
  \]
  In a complicated protocol, there may be several types of messages, and each of the messages may contain parameters that are unique to each of the messages. 
  The inputs are a complete list of the messages and the parameters that they contain.
- O = the outputs, which are similar to those in the FSM, but may also contain parameters, P, that are dependent on the local variables V.
  For instance, in the go-back-N protocol the output may be a message with a specific message number as the parameter.
- B is a set of Boolean expressions (true/false expressions) that may use the local variables or parameters to determine the operation
  For instance, a boolean expression in the receiver in the go-back-N protocol is If( received message number == the next expected message number)
- The transitions T in the EFSM are similar to those in the FSM, but may be triggered spontaneously by internal variable, such as timers, or externally by received messages: \( T = ST \cup IT \), where
  - \( ST \) is a set of spontaneous transitions that occur because of V
    \[
    ST = \bigcup_i \bigcup_j <s_i, b_j, v_j> \rightarrow <n_j, o_j>
    \]
  - \( IT \) is a set of input transitions that occur because of the inputs
    \[
    IT = \bigcup_i \bigcup_j <s_i, b_j, i_j> \rightarrow <n, o_j>
    \]

There are 2 ways to include the test of a Boolean expression in the drawing of an EFSM
6.3 Example

Consider a Stop and Wait ARQ Protocol with the following characteristics

- Messages and acknowledgements are numbered on the field of natural numbers 0, 1, 2, ….

  Ack \( i \) is transmitted by the receiver in response to correctly receiving Message \( i \) from the transmitter.

- The transmitter retransmits message \( i \) if it does not receive Ack \( i \) before a timeout \( T_x \).

  If the transmitter fails to receive an acknowledgement after \( N_x \) attempts, it goes into a "Failure" state, and the protocol terminates (until the problem is corrected).

- The transmitter has an unlimited supply of messages and does not have to wait for the next message from the source.

- There is no limit on the number of messages that may be on the channel simultaneously.

  The channel may deliver messages out of order.

1. The transmitter is sending message \( i \) and receives acknowledgement \( j \). What action should the transmitter take, and why, when:
   
   a. \( j = i \)

      Increase the message number and transmit the next message. The acknowledgement is for the message that the transmitter is trying to transmit.

   b. \( j < i \)

      Discard the acknowledgement. It applies to an earlier message.

   c. \( j > i \)

      Go to the failure state. During normal operation it is not possible for the receiver to acknowledge a message that has not been transmitted.

2. Draw the EFSM of the transmitter

   i. Start in an initial state \( S_I \), where the machine transmits the first message and sets the local variables

   ii. Terminate in a failure state \( S_F \), with no output transitions

   iii. For each output transition from a state, other than \( s_I \), indicate the input signal as input = \( i_j \) that causes the machine to leave the state. (see figure A)

   iv. Express all boolean tests as a diamond with 1 input and 2 outputs, one if true and the other if false. The boolean expression should be in the diamond. (see figure B)

   v. Indicate all outputs as Output = \( o_i \). If there is no output, indicate it as Output = Null. (see figure B)

   vi. Indicate all changes in local variables
3. With reference to the EFSM of the transmitter in (2), specify
   a. The complete set of states $S = \{ s_i \}$ and state what the protocol is doing in each state.
      
      $S = \{ S_I, S_W, S_F \}$
      $S_I = \text{Initial State}$
      $S_W = \text{Wait for an Ack}$
      $S_F = \text{failure - terminating state}$
   
   b. The complete set of Inputs $I = \{ i_i \}$, and state what each $i_i$ is.
      
      $I = \{ \text{Ack(a), timeout} \}$
Ack(a) = Message from receiver with parameter a

timeout = no message from receiver before $T_x$ after transmitting a message

c. The complete set of outputs $O = \{o_i\}$, and describe what the output is.

$O = \{M(m)\}$

$M(m) =$ message from transmitter with parameter m

d. The set of variables stored at the source $V = \{v_i\}$ and describe the function of each variable.

$V = \{m, n_x\}$

$m =$ number of message being transmitted

$n_x =$ number of times that the message has been transmitted

e. The set of $P = \{p_i\}$ and describe the function of the parameter and which input the parameter is received with.

$P = \{a\}$

$a =$ received with the acknowledgement and is the message number being acknowledged

4. Can this protocol be described by a finite state machine, rather than an extended finite state machine?

No. There would be an infinite number of states corresponding to the field of natural numbers

Homework:

1. Go-back-N protocol, window size 4, acks and messages are numbered mod 5.

   A. Draw the component FSM for the source and receiver.

   B. Draw the component EFSM for the source and receiver.

      Consider different ways to use variables instead of individual states. The objective is to make the machine reasonably compact by combining similar states, but to retain the flow of the state machine, rather than reducing the machine to a program.

   C. Draw the component EFSM for the source and receiver, when the window size is n.

      — Identify the internal variables, the parameters, and the boolean expressions.
7. Petri Nets

Components:

1. Places: (circle) a state that part of the system may be in
2. Token - (dot) an indication that a system is in a place - when a system is in a place, the place has one or more tokens
3. Transition: (bar) the rule that governs how tokens move between places
4. Arc: Connections between places and transitions
5. Grammar: the rule describing each transition
6. Rule: When all of the places preceding a transition have at least one token, the transition may fire. When a transition fires, one token is taken from each of the places preceding the token and one token is transferred to each of the places following the transition.

— Examples of petri nets;

\[ AB \rightarrow C \]

\[
\begin{array}{c}
\circ \\
A
\end{array}
\]

\[
\begin{array}{c}
\circ \\
B
\end{array}
\]

\[ \rightarrow \]

\[
\begin{array}{c}
\circ \\
C
\end{array}
\]

\[ A \rightarrow BC \]

\[
\begin{array}{c}
\circ \\
A
\end{array}
\]

\[
\begin{array}{c}
\circ \\
C
\end{array}
\]

\[
\begin{array}{c}
\circ \\
B
\end{array}
\]

\[
\begin{array}{c}
\circ \\
C
\end{array}
\]
Example - full duplex channel, messages numbered 0, 1, acks unnumbered
— Rules:

1. BD->AC wait for ack after xmitting msg 1 - ack received - xmit M0
2. A-> AC timeout while waiting for ack - rexmit M0
3. AD->BE wait for ack after transmitting M0 - ack received - xmit m1
4. B->BE timeout while waiting for ack - rexmit M1
5. C-> M0 lost
6. D-> ack lost
7. E-> M1 lost
8. CF->DF M0 recv while waiting for M1 - send ack and discard M0
9. EG->DG M1 rcvd while waiting for M0 - send ack and discard M1
10. CG->DF M0 rcvd while waiting for M0 - send ack, accept MO
11. EF->DG M1 rcvd while waiting for M1 - send ack, accept M1

— starting state, tokens in ACG - Message zero has just been transmitted and the receiver is waiting for it.

— Normal operation - no messages lost, no timeouts

<table>
<thead>
<tr>
<th>Tokens</th>
<th>Rule</th>
<th>operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACG</td>
<td>10) CG-&gt;DF</td>
<td>MO rcvd and accepted, ack xmitted</td>
</tr>
<tr>
<td>ADF</td>
<td>3) AD-&gt;BE</td>
<td>ack rcvd, M1 xmitted</td>
</tr>
<tr>
<td>BEF</td>
<td>11) EF-&gt;DG</td>
<td>M1 rcv and accepted, ack xmitted</td>
</tr>
<tr>
<td>BDG</td>
<td>1) BD-&gt;AC</td>
<td>ack rcvd, M0 xmitted</td>
</tr>
<tr>
<td>ACG</td>
<td>initial state</td>
<td></td>
</tr>
</tbody>
</table>

— Sequence of operations that lead to a failure

<table>
<thead>
<tr>
<th>Tokens</th>
<th>Rule</th>
<th>operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACG</td>
<td>2) A-&gt;AC</td>
<td>premature timeout - 2 M0 on the channel</td>
</tr>
<tr>
<td>ACCG</td>
<td>10) CG-&gt;DF</td>
<td>MO recd and accepted</td>
</tr>
<tr>
<td>ACDGF</td>
<td>3) AD-&gt;BE</td>
<td>ack rcvd, M1 xmitted</td>
</tr>
<tr>
<td>BCEF</td>
<td>7) E-&gt;</td>
<td>M1 lost</td>
</tr>
<tr>
<td>BCF</td>
<td>8) CF-&gt;DF</td>
<td>ack assumed lost and retransmit</td>
</tr>
<tr>
<td>BDF</td>
<td>1) BD-&gt;AC</td>
<td>ack assumed for M1 which is now irretrievable new M0 transmitted</td>
</tr>
<tr>
<td>ACF</td>
<td>8)CF-&gt;DF</td>
<td>MO rcvd and discarded, ack transmitted</td>
</tr>
<tr>
<td>ADF</td>
<td>3) AD-&gt;BE</td>
<td>ack rcvd, M1 xmitted</td>
</tr>
<tr>
<td>BEF</td>
<td>11) EF-&gt;DG</td>
<td>M1 rcvd, ack xmitted</td>
</tr>
<tr>
<td>BDG</td>
<td>1) BD-&gt;AC</td>
<td>ack rcvd, M0 xmitted</td>
</tr>
<tr>
<td>ACG</td>
<td>initial state, 2 messages lost</td>
<td></td>
</tr>
</tbody>
</table>

— The Petri net is a more compact model than composite FSM, but does not lend itself to a formal graph theoretical analysis. Instead we must create a proof system on the algebra defined by the model.

Homework:

1. Modify the Petri Net for the stop-and-wait ARQ protocol with the messages numbered to also number the acknowledgments. Is there a sequence of events that results in a protocol failure?
REFERENCES


