Protocol Specification

Prof Pallapa. Venkataram
Department of Electrical Communication Engineering
Indian Institute of Science
Bangalore – 560012, India
Objectives of the Chapter

To define a formal specification of protocol.

To identify the components of a protocol to be specified.

To create the formal specification of protocol by using Communicating Finite State Machines (CFSMs).

To specify the multimedia communication protocols by using FSMs.
Introduction

Protocol Specification
- Communication services
- Entities
- interfaces
- Interactions.

Specification Phase of Protocol Design
- allows the designer to prepare an abstract model of the protocol for testing and analysis.
Components of a Protocol

Peer entity of a protocol
Communication Interfaces
Interactions:
• synchronous or asynchronous
Message formats
Communication service specification

Service primitives for a conversation between the two machines

**Service Specification**

- **Service primitives**
  - Request, response, indication and confirm

- **Service primitive parameters:**
  - data size, checksum size, caller address etc.
Communication service specification

Data Transfer phase specification using FSM:

**Sender**

- `rdt_data_request(data)`
  1. compute checksum (2 bytes)
  2. make pkt (sndpkt, data, checksum)
  3. send(sndpkt) and start timer

**Wait for new data**

**WAIT FOR ACK OR NACK**

- `rdt_rcv(rcvpkt) && isNACK(rcvpkt)`
- `timer_expired`
- `send(sndpkt)`
- `wait for new data`

**Receiver**

- `rdt_data_indication(rcvpkt) && corrupt(rcvpkt)`

**Wait for data arrival**

- `rdt_send(NACK)`

- `rdt_data_indication(rcvpkt) && notcorrupt(rcvpkt)`
  1. extract(rcvpkt, data)
  2. deliver_data(data)
  3. `rdt_send(ACK)`

FSM of service specification for reliable data transfer
Communication service specification
Data Transfer phase specification using FSM:

- **rdt_data_request(data)**: compute checksum, make_pkt, send(sndpkt).

- **rdt_data_indication(rcvpkt)**:
  - receiver receives the data and sends an ACK, In case of corrupted data reception, sends NACK.

- **rdt_rcv(rcvpkt)**: (at sender)
  - performs reception of packet (ACK or NACK)
    - If ACK then send next data, if NACK then retransmit the data.

- **rdt_send(ACK)**: receiver sends the ACK.

- **rdt_send(NACK)**: receiver sends the NACK.

- **isNACK(rcvpkt)**: checks for negative acknowledgment reception

- **isACK(rcvpkt)**: checks for positive acknowledgment reception

- **corrupt(rcvpkt)**: checks the packet for corrupted data

- **Notcorrupt(rcvpkt)**: checks for non corruption

- **extract(rcvpkt,data)**: extracts the data from received packet

- **deliver data(data)**: delivers data to the upper layer
Communication service specification

Specification of behavior aspects of a protocol

- Temporal ordering of interactions
- Parameter range
- Selecting values of parameters
- Coding of PDUs
- Liveness properties
- Real-time properties
Protocol Entity Specification

Sender entity specifications

FSM of sender entity specification
Protocol Entity Specification
Sender entity specifications

- The 'rcvpkt' parameter given in the service events rdt data indication(), rdt rcv(), corrupt() and notcorrupt() denotes the ACK/NACK signal reception.
- *timeout()* terminates the timer after certain duration.
- *start_timer* starts the timer after sending the packet either 0 or 1, by using the specified duration.
- *get_data* gets the new data from its for transmission to the specified receiver
- *stop_timer* stops the timer for the sent packet (since the sender receives an acknowledgment from the receiver).
Protocol Entity Specification

Receiver entity specifications

```
rdt_data_indication(rcvpkt)
&& notcorrupt(rcvpkt)
&& has_seq0(rcvpkt)
```

1. extract(rcvpkt,data)
2. deliver_data(data)
3. compute checksum, rdt_send(ACK)
4. sndpkt=make_pkt(sndpkt,ACK,chksum)
5. send(sndpkt)

```
rdt_data_indication(rcvpkt)
&& notcorrupt(rcvpkt)
&& has_seq1(rcvpkt)
```

1. compute checksum
2. rdt_send(NACK)
3. sndpkt=make_pkt(sndpkt,NACK,chksum)
4. send(sndpkt)

```
```
```
```
```
```
```
```
```
Protocol Entity Specification

Receiver entity specifications

- The 'rcvpkt' parameters in the service events denotes the data received from the sender.

- `has_seq1(rcvpkt)` checks for sequence number of the received packet to be 1.

- `has_seq0(rcvpkt)` checks for sequence number of the received packet to be 0.
Channel specifications

communication paths used to connect one or more FSMs of protocol processes.

lossless or lossy channel
Channel specifications

two messages (m0 and m1) transfer **channel representation** by using a FSM.

The channel has **four states**:

- 0-idle
- 1-buffering m0 (msg 0)
- 2-buffering m1 (msg 1)
- 3-buffering ack (ack)

Initially channel will be in state 0, later moves to states 1, 2 and 3 and returns back, based on certain message transitions.
Interface Specifications

**Internal interfaces:**
- mechanisms that are internal to the protocol;

**External interfaces:**
- mechanisms that make it possible for other implementations like applications, other protocols, or other services to interact with

A partial protocol implementation with internal and external interfaces
FSM of an interface of bus access protocol

1: Idle, 2: wait_for_bus, 3: get_data, 4: write-data, 5: release_bus

FSM of an interface of bus access protocol
FSM of an interface of bus access protocol

States:

- Idle (state 1) indicates that bus access protocol is not active;
- wait for bus (state 2) denotes that the protocol is waiting for bus to be idle;
- get data (state 3) gets the data to be transferred on the bus;
- write data (state 4) denotes that, it writes data on the bus; and
- release bus (state 5), releases the bus.

Transitions:

- bus req makes the FSM to transit from state 1 to state 2;
- bus idle allows the FSM to make transit from state 2 to state 3;
- data ready makes the FSM to transit from state 3 to state 4;
- data write makes the FSM to move from state 4 to state 5;
- bus release allows the FSM to transit from state 5 to state 1.
Interactions

FSM of interactions between ISDN system and a user for activation of call forwarding service
Multimedia protocol specifications

QoS (Quality of Service) requirements of multimedia streams:

- **Throughput:**
  - the data transmission rate
  - data compression
  - several Mbps (Megabits per second).

- **Transfer Delay:**
  - time between the production of data at the source and its presentation at the sink

- **Jitter:**
  - variance of the transfer delay
  - use of buffers to reduce jitter

- **Error Rates:**
  - loss of data in a continuous data transfer
Multimedia protocol specifications

FSM specifications

- Buffer requirements

20 buffers on AB and 10 buffers on BD

An example buffer requirement FSM
Multimedia protocol specifications
FSM specifications

- Synchronization

Synchronization representation FSM
Examples of Internet protocol specifications

Alternating bit window protocol

- The Sender_ABGP takes a message which is ready to be sent and transmits the message together with a sequence number via the Data Medium to the Receiver_ABGP.

- The Sender_ABGP waits for an acknowledgment from the Receiver_ABGP containing the same sequence number.
  - If the appropriate acknowledgment arrives, the Sender_ABGP performs the same procedure for the next waiting message, but this time with an inverted sequence number (i.e., \(0 ! 1; 1 ! 0\)).
  - If the appropriate acknowledgment does not arrive within a certain period of time (timeout period), the Sender_ABGP resends the same message.

- The Receiver_ABGP, when in an idle state, acknowledges all incoming messages. After receiving a message with a correct sequence number, it will acknowledge (through Ack Medium) only packets with the last correct sequence number until a Receive signal is received. After that, it will invert the sequence number, and go back to the idle state.
Examples of Internet protocol specifications

Alternating bit window protocol

Alternating bit protocol usage scenario
Examples of Internet protocol specifications

**Alternating bit window protocol**

```
Time_Out; d(0), Start_Timer

a(0); Stop_Timer, Sender_Ready

a(1); -

Send; d(0), Start_Timer

a(0); -

Send; d(1), Start_Timer

a(1); Stop_Timer, Sender_Ready

a(0); -

Timer_Out; d(1), Start_Timer

a(0); -

FSM of sender process in alternating bit protocol
```
Examples of Internet protocol specifications

Alternating bit window protocol

FSM of receiver process in alternating bit protocol
Examples of Internet protocol specifications

Alternating bit window protocol

FSM of data medium channel in ABP

FSM of ack medium channel in ABP
Examples of Internet protocol specifications

**HDLC protocol**

- Three types of control frames: Information, Supervisory, Unnumbered

---

**HDLC frame format**

- **Information Frame**
  - Bits: 1 3 1 3
  - Format: 0 0 type P/F Next
  - Commands: 
    - type 0: DISC, Disconnect
    - type 1: SNRM, set normal response mode
    - type 2: SABM, set async. Balance mode
    - type 3: FRMR, Frame reject

- **Supervisory Frame**
  - Bits: 1 1 2 1 3
  - Format: 0 0 type P/F Next
  - Type: 0: ACK Frame, 1: Reject Frame, 2: Receive not ready, 3: Selective Reject

- **Unnumbered Frame**
  - Bits: 1 1 2 1 3
  - Format: 1 1 type P/F Next
Examples of Internet protocol specifications

Components of HDLC protocol

- link setup
- PF control
- Source
- Sink
- Clock

HDLC protocol components

1: Direct Coupling
2: Shared Variables
3: Hierarchical Independence
Examples of Internet protocol specifications

HDLC

States: {disconnected, wait for ACK, wait for DISC ACK, connected, error}
Inputs: {SXRM, UA, DISC, error, error rectified}
Outputs: {nil}
SXRM: SNRM or SABM, DISC: disconnected request.

FSM of HDLC link setup
Examples of Internet protocol specifications

Communication interfaces

- **Direct coupling**: Execution of instructions simultaneously or sequentially at both ends happen whenever transition takes place, for example, instruction 'x' executed at P/F control, then instruction 'y' to be executed at link setup.

- **Shared variables**: Variables shared between components such as between P/F control and clock, sink and source (timing shared between P/F and clock, regular data shared between source and sink).

- **Hierarchical independence**: link setup is initiated rst, then source and sink are initiated.

- **Complete independence (in one system)**: it is locally independent, depends on local properties, e.g., frame sizes.
Examples of Internet protocol specifications

HDLC protocol

States: {remote ready, remote busy}
Inputs: {I *, RNR, RR, REJ} I * = direct coupled transition sending data

FSM of HDLC source
Examples of Internet protocol specifications

HDLC protocol

States: \{normal seq, seq exception, sent REJ exception\}
Inputs: \{I=, I , REJ\}
I = data in sequence, I = data not in sequence

FSM of HDLC sink
Examples of Internet protocol specifications

**HDLC protocol**

(a) P/F Control
SNRM Mode

- P0 : bit P = 0, not in polling mode
- P1 : bit P = 1, polling mode
- F0 : bit F = 1, finish = 0
- F1 : bit F = 1, finished transfer, go to no polling mode

(b) P/F Control
SABM Mode

- P0 : bit P = 0, not in polling mode
- P1 : bit P = 1, polling mode
- F0 : bit F = 1, finish = 0
- F1 : bit F = 1, finished transfer, go to no polling mode

FSM of HDLC P/F control
Examples of Internet protocol specifications
RSVP protocol specifications

- resource reservation protocol
- used to reserve the resources over the path connecting the server and the client

- types of messages used by RSVP:
  - PATH: data flow information from the sender to the receiver
  - RESV: reservation request from the receiver
  - PATH ERR: generated when path from sender to receiver does not exist
  - RESV ERR: indicates an error in response to the RESV message.
  - PATH TEAR: Removes the PATH state along the route.
  - RESV TEAR: Removes the reservation along the route.
RSVP Application states

Idle:
- transits to Join state at the time that the application is scheduled to join a session or terminate the current session
- transits to Data Send state when the application is going to send data

Join:
- Application receives a session Id from the Local daemon in response to a session call.
- The multicast group id is selected randomly from a uniform distribution.
- If the application is acting as a receiver it will check for the sender information in the session directory for the multicast group that it wants to join and make a receive call to the local RSVP daemon.

Arr:
- This state is activated whenever a message or a packet arrives for the application and by default returns to Idle state

Data Send:
- Creates a data packet and sends it to a selected single/multicast destination that it selects
- on default returns to the Idle state
RSVP Application states

RSVP application state diagram
RSVP Router States

RSVP router state diagram
RSVP Router States

Idle:
- Idle state transits to Arr state upon receiving a packet.

Arr:
- This state checks for the type of the packet arrived and calls the appropriate function depending on the type of message received.

Functions are described as follows:
- Pathmsg: Invoked by the Arr state when a PATH message is received.
- Ptearmsg: invoked by the Arr state when a PATH TEAR message is received
- Resvmsg: invoked by the Arr state when a RESV message is received
- Rtearmsg: invoked by the Arr state when a RESV TEAR message is received
- Rconfmsg: invoked by the Arr state when a RESV CONF message is received.
RSVP States on Hosts

RSVP host state diagram
RSVP States on Hosts

**Idle:** transits to the Arr state on packet arrival.

**Arr:** calls the appropriate functions depending on the type of message received.

Functions executed as internal events are:

- **Session:**
  - called from the Arr state whenever a Session call is received from the local application

- **Sender:**
  - called from the Arr state whenever a sender call is received from the local application

- **Reserve:**
  - called from the Arr state whenever a Reserve call is received from the local application.

- **Release:**
  - called from the Arr state whenever a Release call is received from the local application.