Chapter 5 Implementation

The previous chapter discusses design issues for IGMP, PIM-DM, and the simulation applications. In this chapter, the discussions focus on the implementation of those designs. These include the important attributes and methods employed.

The first section concentrates on the implementation of IGMP. Thereafter the implementation of PIM-DM is presented. This is followed by the implementation of multicast simulation applications and the implementation of simulation topology. Finally a discussion on the exclusions and simplification of implementation is presented prior to the chapter summary.

5.1 Implementation of IGMP

The section below lists the implementation of each class involved in IGMP. The descriptions include the attributes and the methods implemented in a particular class.

5.1.1 IGMPv2 Class

This section lists all the important attributes and methods implemented. This includes the Group class and the IGMP Timer class. The details of the state diagram for the implementation of IGMP are listed in Appendix A while the constants used in the IGMPv2 class are included in Appendix B.

class IGMPv2 implements Serializable {
  //Important attributes/
  public SimParamGroupTable the_groupTable; //An object for displaying
  // group membership entries
  private java.util.List the_gtable; //A list of group membership
  //entries
  private java.util.List the_timerList; //A list of IGMP timers
  private int status; //A status for implementation of IGMP
  //HOST/QUERIER/NONQUERIER

86
IGMPv2(int aStatus, SimComponent aComp, JavaSim aSim);

// Important Methods  
 shl

public void igmpRouterInit( 
    Object an_outPort) {  
    // for router
    // initialization
}

private void setStatus(int a_State) {  
    // set routes' status
}

// Main interface (refer to 4.2.1)

public void igmpInput(Object [] paramlist) {  
    // IGMP Input
}

public void igmpOutput(Cell cell, Object an_outPort) {  
    // IGMP Output
}

public void groupJoin(int a_group) {  
    // handle joining event
}

public void groupLeave(int a_group) {  
    // handle leaving event
}

// IGMP timer handling (refer to 3.1.4)

public void unsolicitedReportTimeout(SimEvent e) {  
}

public void queryTimeout(SimEvent e) {  
}

public void queryResponseTimeout(SimEvent e) {  
}

public void groupMembershipTimeout(SimEvent e) {  
}

public void otherQuerierPresentTimeout(SimEvent e) {  
}

public void startupQueryTimeout(SimEvent e) {  
}

public void lastMemberQueryTimeout(SimEvent e) {  
}

private void resetTimer(int a_group, Object an_outgoingIf, int  
    an_eventType, double a_tempo, int toEventType) {  
    // reset timer
}

private void stopTimer(int a_group, int an_eventType) {  
    // stop timer
}

private void clearTimer(int a_group, int an_eventType) {  
    // clear timer
}

// IGMP notification (refer to 3.1.4)

// trigger IGMP Notification Add
private void notifyRoutingAdd(int a_group, Object an_outgoingIf) {  
    // trigger IGMP Notification Remove
private void notifyRoutingRemove(int a_group, Object an_outgoingIf) {  
}

// IGMP Message handling (refer to 3.1.4)

private void sendReport(int a_group) {  
    // to send Report
}

private void sendQuery(int a_group, Object an_outPort) {  
    // to send
    // Query
}

private void sendLeave(int a_group) {  
    // send Leave
}

private void receiveQuery(Cell aCell, Object an_inPort) {  
    // to receive
    // Query
}

private void receiveReport(Cell aCell, Object an_inPort) {  
    // to receive
    // Leave
}

private void receiveLeave(Cell aCell, Object an_inPort) {  
    // to receive
    // Leave
}

// Group membership entry handling
private void addEntry(int a_group, Object an_outgoingIf) {  
    // to add
    // new entry
}

private void increaseCount(int a_group) {  
    // to increase reference
    // count
}

private void decreaseCount(int a_group) {  
    // to decrease reference
    // count
}
private void setGroupState(int a_group, int a_state){} //to set entry
   //state
private void setFlag(int a_group){} //to set Flag
private void clearFlag(int a_group){} //to clear or reset Flag

private class Group Class {
   int state; //FREE, DELAYING or IDLE (host only)
   int groupAddress; //An address define the multicast group
   Object outLink=null; //An interface for receiving and sending
      //multicast packet
   int referenceCount; //The number of process still interested
      //in the group (host only)
   boolean lastReportFlag; //The last node send IGMP report flag
      //(host only)
}

private class IGMPTimer Class {
   int groupAddress; //An address that define the
      //multicast group, 0 for router
   int eventType; //Define the timer event type
   long timeOut; //Specify the timeout time
   Object outPort=null; //An outgoing interface
   SimEvent ole= null; //An old timer event
   IGMPTimer(int a_group, Object an_outPort){} //Constructor
   private void start(int an_eventType, double a_tempo,
      SimComponent self){} //to start a timer
   private void stop(){} //to stop a timer
   private void clear(){} //to clear a timer
}

5.1.2 SimParamGroupTable Class

This class is used to display the IGMP group membership entries.

class SimParamGroupTable extends SimParameter implements
   ActionListener, Serializable{

   private transient JComponent jcomp=null; //It is used for GUI
   private IGMPv2 the_igmp; //Called IGMPv2 object
   private SimComponent the_comp;//Called IPBTE or ATMLSR depend on
      //implementation
SimParamGroupTable(String aName, String compName, long creationTick, SimComponent a_Comp, JavaSim a_Sim, IGMPv2 a_igmp) {} 

//Action performed when button in GUI is pressed 
public void actionPerformed(ActionEvent e){}

//Format of table displayed
private class MyTableModel extends Javax.swing.table.AbstractTableModel {} 

5.1.3 Modification in IPBTE Class

Stated below are the new attributes and methods added in the IPBTE class for the implementation of IGMP. Only implementations involved IGMP are listed.

class IPBTE extends SimComponent implements Serializable {

    //Important attributes
    private IGMPv2 the_igmp=null; //Called IGMPv2 object

    // Constructor
    IPBTE(String name, int c, int t, JavaSim aSim, java.awt.Point loc) {
        Initiate an IGMPv2 Object for Host
    }

    //IGMP Event Listener
    void action(SimEvent e) {
        SWITCH event type
            CASE message receive
              pass to message handling method in IPBTE
            CASE IGMP message send out
              pass to message handling method in IPBTE
            CASE IGMP join or leave event
              pass to IGMP event handling method in IPBTE
            CASE timer
              pass to appropriate timer event handling method in IGMPv2
        END
    }
}

89
/**New method added, IGMP event handling method**
private void b_receive_igmp(SimEvent e) {
    IF join THEN
        pass to groupJoin method in IGMPv2
    END
    IF leave event THEN
        pass to groupLeave method in IGMPv2
    END
}

/** Modifications Done in Existing Methods**
/** Received message handling method**
private void b_receive(SimEvent e) {
    IF message is a sent out IGMP THEN
        Pass to output port
    END
    IF message receive from LINK THEN
        IF destination is class D address THEN
            IF destination address is ALL_SYSTEMS_MULTICAST_GROUP (224.0.0.1) THEN
                pass to igmpInput method
            END
            IF destination address larger than 224.0.0.255 THEN
                IF is IGMP packet THEN
                    pass to igmpInput method
                ELSE
                    pass to relevant multicast applications
                END
            END
        END
    END
    END
}

5.1.4 Modification in ATMLSR Class

Stated below are the new attributes and methods added in the ATMLSR class for the implementation of IGMP. Only implementations involved IGMP are listed.

class ATMLSR extends SimComponent implements Serializable {
    /////////////////////////////////////////////////////
    //Important attributes//
    /////////////////////////////////////////////////////
    private IGMPv2 the_igmp=null; //Called IGMPv2 object

    /////////////////////////////////////////////////////
    // Constructor //
    /////////////////////////////////////////////////////
    ATMLSR(String name,int c,int t,JavaSim aSim,javax.awt.Point loc) {
        Initiate an IGMPv2 Object for Router
    }
}
5.2 Implementation of PIM-DM

In this section, the implementations of each class involved PIM-DM are listed. The descriptions include the attributes and methods implemented in a particular class.

5.2.1 PIMDM Class

This section lists all the important attributes and methods implemented. It includes the SG class, the SGOF class, the PIMNeighbor class and the PIMDMTimer class. Detail of
system flow for the implementation of PIM-DM is listed in section 5.4 while the constants used in the PIMDM class are included in Appendix B.

class PIMDM implements Serializable {

    //Important attributes /
    public SimParamPIMDMTable the_pimSGTable;  //An object for displaying
        // (S, G) entries
    public SimParamPIMDMNeighborTable the_pimNTable; //An object for
        //displaying information of PIM-DM neighbor
    private java.util.List the_sqTable;    //A list of (S, G) entries
    private java.util.List the_membershipTable; //A list of membership
        // entries
    private java.util.List the_neighborTable; //A list of information of
        //PIM-DM neighbor routers
    private java.util.List the_timerList;    //A list of PIM-DM timers
    private int genID;                      //A generation ID for router

    // Constructor
    PIMDM(SimComponent aComp, JavaSim aSim){}

    // Important Methods
    public void pimdmInit( Object a_port){} //initialization for router

    //Main interface (refer to 4.3.1)
    public void pimdmInput(Object [] paramlist ){} //PIM-DM Input
    public void pimdmOutput(Cell cell, Object an_outPort){} //PIM-DM
        //Output
    public java.util.List getForwardList(int a_source, int a_group,
            Object an_inPort){} //get forwarding list
    public void notRPF(int a_source, int a_group, Object an_incomingIf,
            Object an_rpfIncomingIf){} //handle no RPF packet

    //Interface handling (refer to 4.3.1)
    private void pimdmForward(int a_source, int a_group, Object
            an_outgoingIf){} //forward handling
    private void pimdmPrune(boolean isRFP, int a_source, int a_group,
            Object an_outgoingIf){} //prune handling

    //Neighbor handling (refer to 3.2.4)
    private void procGenerationID(Cell a_cell, Object
            an_inPort){} //process received GenID
    private void receiveNewGenID(int a_neighborIP, Object
            an_inPort){} //handle received new GenID

    //PIM-DM event handling (refer to3.2.4)
    public void receiveNotificationAdd(SimEvent e){} //receive IGMP
        //Notification Add
    public void receiveNotificationRemove(SimEvent e){} //receive IGMP
private void receiveNewMembership(int a_group, Object
an_outgoingIf){} // new member handling
private void receiveLeaveMembership(int a_group, Object
an_outgoingIf){} // member leave handling

// PIM-DM Message handling (refer to 3.2.4)
private void sendHello(Object an_outPort){} // to send Hello message
private void sendPrune(int a_source, int a_group, int
an_upstreamRouter, Object an_outPort){} // to send Prune message
private void sendGraft(int a_source, int a_group, Object
an_outPort){} // to send Graft message
private void sendGraftAck(Cell a_cell, Object an_outgoingIf){} // to
// send Graft-Ack message
private void receiveHello(Cell a_cell, Object an_inPort){} // to
// receive Hello message
private void receivePrune(Cell a_cell, Object an_inPort){} // to
// receive Prune message
private void receiveGraft(Cell a_cell, Object an_inPort){} // to
// receive Graft message
private void receiveGraftAck(Cell a_cell, Object an_inPort){} // to
// receive Graft-Ack message

// SG entry handling
private void addSGEntry(int a_source, int a_group, Object
an_incomingIf){} // add new SG entry
private void removeSGEntry(int a_source, int a_group, Object
an_incomingIf){} // remove SG entry

// PIM-DM entry handling
private void addNeighborEntry(int a_state, int a_source, Object
an_incomingIf, int a_genId){} // add new Neighbor entry
private void removeNeighborEntry(Object an_incomingIf){} // remove
// Neighbor entry

// PIM-DM membership handling
private void addMembership(int a_group, Object an_outgoingIf){} // add
// new Membership entry
private void removeMembership(int a_group, Object
an_outgoingIf){} // remove Membership entry

// PIM-DM timer event handling (refer to 3.2.4)
public void helloTimeout(SimEvent e){}
public void neighborTimeout(SimEvent e){}
public void pruneTimeout(SimEvent e){}
public void dataTimeout(SimEvent e){}
public void graftAckTimeout(SimEvent e){}
public void assertTimeout(SimEvent e){}
public void unicastTableChange(SimEvent e){}
private void resetTimer(int a_state, int a_source, int a_group,
int a_eventType, double a_tempo){} // to reset timer
private void stopTimer(int aGroup, int anEventType){} // to stop timer
private void clearAllTimer(){} // to clear all timer in the timer list
private void clearTimer(int a_state, int a_source, int a_group,
int a_eventType, Object an_outgoingIf){} // to clear timer
private class SG implements Serializable {
    int state;  //FORWARD, NEGATIVE_CACHE
    int source;  //Address for sending source
    int groupAddress;  //Address define the multicast group
    Object incomingIf=null;  //An incoming interface
    java.util.List outgoingList; //A list of outgoing interfaces
SG();
}

private class SGOF implements Serializable {
    Object outgoingIf=null;  //An outgoing interface
}

private class PIMNeighbor implements Serializable {
    int ipAddress;  //An IP address of PIM-DM neighbor router
    int genId;  //A generation ID
    Object incomingIf=null;  //An incoming interface
}

private class Membership implements Serializable {
    int groupAddress;  //A group address
    Object outgoingIf;  //An outgoing interface
}

private class PIMDMTimer implements Serializable {
    int state;  //indicate the state of the timer: router(hello, assert)
    // <S,G> entry(data timeout and graft ack, neighbor>,
    // prune(individual prune if)
    //free if not used
    int groupAddress; //address define the multicast group;0 for router
    int source;  //A source address
    Object outgoingIf;  //An interface
    int eventType;  //Timer event type
    long timeOut;  //Timeout time
    SimEvent olde=null;  //An old event

    PIMDMTimer(int a_state, int a_source, int a_group, Object an_outgoingIf){} //constructor
    private void start(int aneventtype, double atempo, SimComponent self){} //to start timer
    private void stop(){} //to stop timer
    private void done(){} //a good practice to idle the expired timer
private void clear(){//to clear timer
}

5.2.2 SimParamPIMDMTable Class

This class is used to display (S, G) entries.

class SimParamPIMDMTable extends SimParameter implements ActionListener, Serializable {

    //______________________________________________________________________
    //Important attributes//
    //______________________________________________________________________
    private transient JComponent jcomp=null; //It is used for GUI
    private PIMDM the_pimdm; //Called PIMDM object
    private SimComponent the_comp; //Called ATMLSR

    //______________________________________________________________________
    // Constructor //
    //______________________________________________________________________
    SimParamPIMDMTable(String aName,String compName,long creationTick,
            SimComponent a_Comp, JavaSim a_Sim, PIMDM a_pimdm){}

    //______________________________________________________________________
    // Important Methods //
    //______________________________________________________________________
    //Action performed when button in GUI is pressed
    public void actionPerformed(ActionEvent e){}

    //Format of table displayed
    private class MyTableModel extends
            javax.swing.table.AbstractTableModel{

    }

5.2.3 SimParamPIMDMNeighbor Class

This class is used to display the information of PIM-DM neighbor routers. Basically, the functions of methods used in this class are almost similar, except the display format for the GUI part.

class SimParamPIMDMNeighborTable extends SimParameter implements ActionListener, Serializable {

    //______________________________________________________________________
    //Important attributes//
    //______________________________________________________________________
    private transient JComponent jcomp=null; //It is used for GUI
    private PIMDM the_pimdm; //Called PIMDM object

95
private SimComponent the_comp; //Called ATMLSR

SimParamPIMDMNeighborTable(String aName, String compName, long creationTick, SimComponent a_Comp, JavaSim a_Sim, PIMDM a_pimdm){}

public void actionPerformed(ActionEvent e){}

private class MyTableModel extends javax.swing.table.AbstractTableModel{
}

5.2.4 Modification in ATMLSR Class

Stated below are the new attributes and methods added in the ATMLSR class for the implementations of PIM-DM. Only implementations that involved in PIM-DM are listed.

class ATMLSR extends SimComponent implements Serializable {

private PIMDM the_pimdm=null; //Called PIMDM object

ATMLSR(String name, int c, int t, JavaSim aSim, java.awt.Point loc) {
   Initiate an IGMPv2 Object for Router
}

//PIM-DM Event Listener
void action(SimEvent e){
   SWITCH event type
      CASE message receive
         pass to message handling method in ATMLSR
      CASE PIM-DM message send out
         pass to sent out PIM-DM messages handling method in ATMLSR
      CASE timer

96
pass to appropriate timer event handling method in PIMDM
CASE IGMP notification
pass to appropriate notification handling method in PIMDM
CASE unicast table change
pass to unicast table change handling method

END

// New method added
// initialization of PIM-DM
private void sw_pimd_init(){
    FOR every interface
        do pimdInit method

}

// PIM-DM message handling method
private void sw_receive_pimd(Cell a_cell, Port a_voport){
    pass to pimdInput method
}

// PIM-DM message sent out handling method
private void sw_send_pimd(SimEvent e){
    pass to queue for sending out
}

// Multicast message handling method
private void sw_receive_ip_multicast(Cell a_cell, Port an_inPort) {
    IF multicast packet received from RPF interface THEN
        pass to getForwardList method to get forwarding interface list in PIMDM
    IF forward list is not null THEN
        pass to queue to forward out multicast packet according to the forwarding list
    END
    ELSE
        pass to notRPF method in PIMDM
    END

}

// Modifications Done in Existing Methods
private void sw_my_receive(SimEvent e) {
    IF it is a IP data packet THEN
        IF destination address is a class D address THEN
            pass to sw_receive_ip_multicast method
        ELSE
            pass to sw_receive_ip_datagram method // existing
    END

    IF it is a PIM-DM packet THEN
        pass to sw_receive_pimd method

    END
5.3 Implementation of Simulation Application

5.3.1 MulticastdataApp Class

This class is used to send multicast data. Four values have to be specified before a simulation of multicast packet sending starts. These values are multicast group address, start sending time, the data rate (in $\mu$s), and stop sending time.

When simulation starts, the object of this class checks the start and stop sending time and enqueue the multicast packet start sending and stop sending events. Once the sending event is called, this application class sends out IP multicast packet of a group at the predefined data rate. Stated below are some important attributes and methods implemented in this class.

class MulticastdataApp extends SimComponent implements Serializable {

    //////////////////////////////////////////////////////////////////////////
    // Important attributes
    //////////////////////////////////////////////////////////////////////////
    private SimParamIP the_destGroupIP=null; // A multicast group address
    private SimParamInt the_sendTime=null; // A time start sending
    //////////////////////////////////////////////////////////////////////////
    private SimParamInt the_period=null; // A sending rate in microsecond
    private SimParamInt the_stopTime=null; // A time stop sending
    //////////////////////////////////////////////////////////////////////////

    //////////////////////////////////////////////////////////////////////////
    // Constructor
    //////////////////////////////////////////////////////////////////////////
    MulticastdataApp(String name, int c, int t, JavaSim aSim, java.awt.Point loc){

    //////////////////////////////////////////////////////////////////////////
    // Important Methods
    //////////////////////////////////////////////////////////////////////////
    private void cn_ev_send_multicast(int a_time){} // Enqueue send event
    private void cn_ev_stop_multicast(int a_time){} // Enqueue stop event
    private void cn_send_multicast(){} // Send multicast packets
}

Data rate is the sending rate for the simulation of multicast application. An input of data rate in unit of microsecond is needed before the simulation starts. The MulticastdataApp object uses this input to calculate the sending rate and applies it when simulation starts. Calculation of sending rate is performed as below.

98
Let period = input data rate in microsecond

For every period send a cell

A cell consist of 53 bytes, which is equal to 53 \times 8 = 424 \text{ bits/second}

Hence to send \( x \) bits/second, period = \( \frac{1}{(x/424)} \) second

5.3.2 IPMulticast Class

This class is used to simulate the behavior of a member of a multicast group. It can perform the function of joining a group, receiving multicast packets and leaving a group. Three values must be specified before the simulation starts. They are multicast group address, joining time and duration of joining a multicast group.

When simulation begins, the object of this class enqueues 2 events. The first event is a group joining event, enqueued to occur at joining time. The second event is the group leaving event, enqueued to occur at leaving time. The leaving is the total of joining time and duration of joining.

When a group joining event is triggered, the object will trigger another event to the IPBTE object to start joining a multicast group. On the other hand, when a group leaving event is triggered, a leave event will be sent to the IPBTE object to leave that particular multicast group address.

Once joining a multicast group, this object is able to receive multicast packet from the multicast group. Stated below are important attributes and methods implemented in this class.

Class IPMulticastApp extends SimComponent implements Serializable {

    // Important attributes/

    private SimParamIP the_groupIP=null; //An address of a multicast group
    private SimParamInt the_joinTime=null; //A time start join
    private SimParamInt the_duration=null; //A duration of joining a group

    // Constructor /

}
IMulticastApp(String name, int c, int t, JavaSim aSim, java.awt.Point loc)
{

// Important Methods

private void cn_join_leave_group(int a_group, int a_join_time, int a_duration)

private void cn_receive(SimEvent e)

To simulate a real group joining and leaving event for multicast, a random joining time is implemented. In fact, the input value for joining time is the mean value. The input value acts like a seed for generating a random joining time. For example if the value is 144s, and there are 3 IMulticastApp objects running in the simulation, the join time for these applications may be 105s, 135s and 155s respectively.

Similarly, the input joining duration is also used as the mean value to randomly generate a new duration of joining process. Then, the leaving time for the multicast application is equal to the randomly generated joining duration plus the randomly generated joining time. For example, if the above example uses 360s as an input joining duration, the joining duration for those applications may be 341s, 361s, and 369s respectively. Hence, the leaving time for the multicast applications in the simulation are 446s, 496s and 524s respectively.

The name assigned to the IMulticastApp object is also used as the random seed, similar name of application and input value will have similar randomly generated joining and leaving time.

5.4 System Flow for Implementation

The state diagram for the implementation of the IGMPv2 class is shown in the Appendix A while the system flow diagram for the implementation of the PIMDM class is shown in Figure 5.1. The implementations are referred to these two diagrams.
Figure 5.1 System Flow for Implementation of PIMDM class
5.5 Simulation Topology

In a real multi-access medium network, some of the interconnected routers may be connecting to hosts and routers to form a subnet. Every host may be hosting one or multiple multicast applications that would be receiving or sending multicast data. It is shown in Figure 5.2.

Every host has an IGMP module, which will handle the joining and leaving of multicast group required by each multicast application. When an application desires to join or leave a multicast group, IGMP in that host will be acknowledged. Then, the IGMP modules in the host will communicate with the IGMP module in the router within the same subnet. This will subsequently notify the PIM-DM module in the routers to construct a multicast tree among routers.

![Diagram showing a multi-access network with routers, hosts, and multicast applications](image)

*Figure 5.2 A Multi-access Network*

However, for this study, UMJaNetSim could only provide point-to-point link network communications but not multi-access link network. Hence, modification on simulation process and topology has been done so that a simulation on IP Multicast through PIM-DM could be carried out as planned.

Classes created in the previous sections are used in the simulation. The components of the simulation environment are shown in Table 5.1. Each of the components is
implemented as a SimComponent. Figure 5.3 shows the connection of each simulation component in a network simulation environment.

<table>
<thead>
<tr>
<th>Table 5.1 Simulation Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
</tr>
<tr>
<td>ATMLSR</td>
</tr>
<tr>
<td>GenericLink</td>
</tr>
<tr>
<td>IPBTE</td>
</tr>
<tr>
<td>MulticastdataApp</td>
</tr>
<tr>
<td>IPMulticastApp</td>
</tr>
</tbody>
</table>

In this network simulation environment, the ATMLSR simulates a router while the GenericLink simulates the behavior of physical link. The combination of the IPBTE, the MulticastdataApp and the IPMulticastApp simulates the behavior of a subnet.

In fact, the router simulated in UMJaNetSim is an Asynchronous Transfer Mode (ATM) Switch. Hence, IP over ATM is implemented. IP packets are sliced into ATM cells before transported to their destinations. Since multicast routing protocol is a layer-3 protocol, implementation of IP over ATM in this project will not affect the results of simulation.

![Figure 5.3 Network Simulation Environments](image)

Figure 5.3 Network Simulation Environments
Since UMJaNetSim is only designed to simulate point-to-point link network, an \textit{IPBTE} cannot be simply connected to multiple hosts in a simulation. The number of hosts is not an important factor in this simulation. \textit{IPBTE} represents a subnet with only one interface to a router. Here, neither the \textit{MulticastdataApp} nor the \textit{IPMulticastApp}, which is connected to an \textit{IPBTE}, is representing a host in a subnet. In fact, the combination of an \textit{IPBTE}, a \textit{MulticastdataApp} and a \textit{IPMulticastApp} provide a scenario of IP multicasting activities and behaviors in a subnet. This means that the simulation of the subnet is emphasized on the behavior of multicast application but not the behavior of hosts.

All members of multicast applications in a subnet share a common \textit{IGMPv2} module in an \textit{IPBTE}. As mentioned before, in a real network, the IGMP module resides in every host to handle multicast group joining and leaving for that host. Here, for the simulation purposes, a common \textit{IGMPv2} module resides in the \textit{IPBTE}. It is used to handle all multicast group joining and leaving for the subnet. The \textit{IGMPv2} in an \textit{IPBTE} will communicate with the \textit{IGMPv2} module in an \textit{ATMLSR} to pass the joining and leaving information to the \textit{PIMDM} module for the construction of multicast path.

For a host, the IGMP is only used to join and leave a multicast group, so it is still valid to use a common IGMP module in the simulation environment as long as the router could receive join and leave membership correctly.

### 5.6 Exclusions and Simplifications in Implementation

For the implementations, simplifications and assumptions are made in order to limit the simulation scope. Hence, a few features in both IGMP and PIM-DM are not implemented in the simulation.

For the implementation of IGMP, all routers in the network simulation environment are assumed to be IGMP version 2 enabled, therefore Version 1 Router Present Timer is not implemented.
Due to the reason that the simulation topology implements only point-to-point link network, features that involves multi-access network in PIM-DM are also excluded from the implementations. Assert message is a method used to solve the problem of multiple path to destination in multi-access network, so the sending, receiving and processing of Assert message is not needed here, and excluded from implementation of PIM-DM. Meanwhile, selection of designated router (querier and non-querier) is also not implemented. This is because the selection of designated router is only required in multi-access network.

Existing UMJaNetSim implements OSPF as a unicast routing protocol. However, dynamic OSPF routing is not implemented in the simulation. All simulation process involves non-dynamic unicast routing. Hence, unicast table change handling is also not implemented in this project.

On the other hand, there is a simplification in implementing the PIM-DM neighbor routers handling. In fact, there is one field purposely reserved in PIM-DM Hello message or IP address of PIM-DM neighbor router. However, to simplify the implementation, a unique router identification number (hereafter it is referred to router ID) is assigned to this field. Every router has a unique router ID. Since two routers are connected in a point-to-point link and every router has a unique router ID, router ID could act as and therefore replace the IP address of an interface of a router. Thus, router ID is used in the IP address field in PIM-DM Hello message. This simplification has no impact to the implementation of PIM-DM.

Since packet errors are not simulated in this simulation environment, there is no need to implement checksum for messages in IGMP and PIM-DM.
5.7 Summary

In this chapter, implementations of IGMP, PIM-DM, simulation application and topology are discussed. The important attributes and methods used in each class are presented. There are some exclusions and simplifications in the implementation of IGMP and PIM-DM. Simulation involves IGMP version 2 routers only. Besides, the simulation of PIM-DM is only implemented in a point-to-point network simulation environment. Hence, Assert message is not included in the simulation. In addition, a method of handling of a unicast table change in the PIMDM is also not implemented. Moreover, there is a simplification in PIM-DM neighbor handling and all packets involved in the simulation are assumed to be error free.

In the next chapter, the discussion is focused on the network simulation of IP Multicast using PIM-DM. Testing and simulation results is presented along with analysis and descriptions.