

Multicast Routing Modelling In OMNeT++

Generated by Doxygen 1.8.0

Sat Apr 28 2012 20:38:01

Contents

1	Multicast Routing Modelling in OMNeT++ Documentation	1
2	Directory Hierarchy	3
2.1	Directories	3
3	Class Index	5
3.1	Class List	5
4	File Index	7
4.1	File List	7
5	Directory Documentation	9
5.1	F:/ANSA/src/ansa/ Directory Reference	9
5.2	F:/ANSA/ Directory Reference	9
5.3	F:/ Directory Reference	9
5.4	F:/ANSA/src/ansa/ipv4/ Directory Reference	9
5.5	F:/ANSA/src/ansa/pim/modes/ Directory Reference	9
5.6	F:/ANSA/src/ansa/multicastRoutingTable/ Directory Reference	10
5.7	F:/ANSA/src/ansa/pim/ Directory Reference	10
5.8	F:/ANSA/src/ Directory Reference	10
5.9	F:/ANSA/src/ansa/pim/tables/ Directory Reference	11
6	Class Documentation	13
6.1	addRemoveAddr Class Reference	13
6.1.1	Detailed Description	13
6.1.2	Member Function Documentation	13
6.1.2.1	setAddr	13
6.1.2.2	setInt	14
6.1.2.3	getAddr	14
6.1.2.4	getAddrSize	14
6.1.2.5	getInt	14
6.1.3	Member Data Documentation	14
6.1.3.1	addr	14

6.1.3.2	pimInt	14
6.2	AnsaIP Class Reference	14
6.2.1	Detailed Description	15
6.2.2	Member Function Documentation	15
6.2.2.1	handlePacketFromNetwork	15
6.2.2.2	routeMulticastPacket	16
6.2.2.3	initialize	18
6.2.3	Member Data Documentation	18
6.2.3.1	mrt	19
6.2.3.2	nb	19
6.3	MulticastIPRoute Class Reference	19
6.3.1	Detailed Description	20
6.3.2	Constructor & Destructor Documentation	20
6.3.2.1	MulticastIPRoute	20
6.3.3	Member Function Documentation	20
6.3.3.1	setSource	20
6.3.3.2	setGroup	21
6.3.3.3	setRP	21
6.3.3.4	setGrt	21
6.3.3.5	setSat	21
6.3.3.6	setSrt	21
6.3.3.7	setFlags	21
6.3.3.8	isFlagSet	21
6.3.3.9	addFlag	21
6.3.3.10	removeFlag	22
6.3.3.11	setInInt	22
6.3.3.12	setInInt	22
6.3.3.13	setOutInt	22
6.3.3.14	addOutInt	22
6.3.3.15	isRpf	22
6.3.3.16	isOlistNull	22
6.3.3.17	getSource	23
6.3.3.18	getGroup	23
6.3.3.19	getRP	23
6.3.3.20	getGrt	23
6.3.3.21	getSat	23
6.3.3.22	getSrt	23
6.3.3.23	getFlags	23
6.3.3.24	getInInt	23
6.3.3.25	getInIntPtr	23

6.3.3.26	getInIntId	24
6.3.3.27	getInIntNextHop	24
6.3.3.28	getOutInt	24
6.3.3.29	getOutIntByIntId	24
6.3.3.30	getOutIdByIntId	24
6.3.4	Member Data Documentation	24
6.3.4.1	source	24
6.3.4.2	group	24
6.3.4.3	RP	24
6.3.4.4	flags	25
6.3.4.5	grt	25
6.3.4.6	sat	25
6.3.4.7	srt	25
6.3.4.8	inInt	25
6.3.4.9	outInt	25
6.4	MulticastRoutingTable Class Reference	25
6.4.1	Detailed Description	26
6.4.2	Constructor & Destructor Documentation	26
6.4.2.1	~MulticastRoutingTable	26
6.4.3	Member Function Documentation	26
6.4.3.1	routeMatches	26
6.4.3.2	updateDisplayString	27
6.4.3.3	generateShowIPMroute	27
6.4.3.4	printRoutingTable	28
6.4.3.5	getRouteFor	28
6.4.3.6	getRouteFor	29
6.4.3.7	getRoutesForSource	30
6.4.3.8	getNumRoutes	30
6.4.3.9	getRoute	31
6.4.3.10	findRoute	31
6.4.3.11	addRoute	31
6.4.3.12	deleteRoute	32
6.4.3.13	initialize	33
6.4.3.14	handleMessage	33
6.4.4	Member Data Documentation	34
6.4.4.1	multicastRoutes	34
6.4.4.2	showMRoute	34
6.4.4.3	ift	34
6.5	MulticastRoutingTableAccess Class Reference	34
6.5.1	Detailed Description	34

6.6	pimDM Class Reference	34
6.6.1	Detailed Description	35
6.6.2	Member Function Documentation	35
6.6.2.1	receiveChangeNotification	35
6.6.2.2	newMulticast	37
6.6.2.3	newMulticastAddr	38
6.6.2.4	oldMulticastAddr	39
6.6.2.5	dataOnPruned	41
6.6.2.6	dataOnNonRpf	41
6.6.2.7	dataOnRpf	42
6.6.2.8	rpfIntChange	42
6.6.2.9	processPIMTimer	43
6.6.2.10	processPruneTimer	44
6.6.2.11	processGraftRetryTimer	45
6.6.2.12	processSourceActiveTimer	46
6.6.2.13	processStateRefreshTimer	46
6.6.2.14	createPruneTimer	47
6.6.2.15	createGraftRetryTimer	47
6.6.2.16	createSourceActiveTimer	48
6.6.2.17	createStateRefreshTimer	48
6.6.2.18	processPIMPkt	49
6.6.2.19	processJoinPruneGraftPacket	50
6.6.2.20	processPrunePacket	51
6.6.2.21	processGraftPacket	52
6.6.2.22	processGraftAckPacket	53
6.6.2.23	processStateRefreshPacket	54
6.6.2.24	sendPimJoinPrune	55
6.6.2.25	sendPimGraft	56
6.6.2.26	sendPimGraftAck	56
6.6.2.27	sendPimStateRefresh	57
6.6.2.28	handleMessage	58
6.6.2.29	initialize	58
6.6.3	Member Data Documentation	59
6.6.3.1	rt	59
6.6.3.2	mrt	59
6.6.3.3	ift	59
6.6.3.4	nb	59
6.6.3.5	pimIft	59
6.6.3.6	pimNbt	59
6.7	PimInterface Class Reference	60

6.7.1	Detailed Description	60
6.7.2	Member Function Documentation	60
6.7.2.1	info	60
6.7.2.2	setInterfaceID	60
6.7.2.3	setInterfacePtr	61
6.7.2.4	setMode	61
6.7.2.5	getInterfaceID	61
6.7.2.6	getInterfacePtr	61
6.7.2.7	getMode	61
6.7.2.8	getIntMulticastAddresses	61
6.7.2.9	setIntMulticastAddresses	61
6.7.2.10	addIntMulticastAddress	61
6.7.2.11	removeIntMulticastAddress	61
6.7.2.12	isLocalIntMulticastAddress	62
6.7.2.13	deleteLocalIPs	62
6.7.3	Member Data Documentation	63
6.7.3.1	intID	63
6.7.3.2	intPtr	63
6.7.3.3	mode	63
6.7.3.4	intMulticastAddresses	63
6.8	PimInterfaceTable Class Reference	63
6.8.1	Detailed Description	64
6.8.2	Member Function Documentation	64
6.8.2.1	getInterface	64
6.8.2.2	addInterface	64
6.8.2.3	getNumInterface	64
6.8.2.4	printPimInterfaces	64
6.8.2.5	getInterfaceByIntID	65
6.8.2.6	handleMessage	65
6.8.3	Member Data Documentation	65
6.8.3.1	pimIft	65
6.9	PimInterfaceTableAccess Class Reference	65
6.9.1	Detailed Description	66
6.10	PimNeighbor Class Reference	66
6.10.1	Detailed Description	67
6.10.2	Member Function Documentation	67
6.10.2.1	info	67
6.10.2.2	setId	67
6.10.2.3	setInterfaceID	67
6.10.2.4	setInterfacePtr	67

6.10.2.5	setAddr	67
6.10.2.6	setVersion	67
6.10.2.7	setNlt	67
6.10.2.8	getId	68
6.10.2.9	getInterfaceID	68
6.10.2.10	getInterfacePtr	68
6.10.2.11	getAddr	68
6.10.2.12	getVersion	68
6.10.2.13	getNlt	68
6.10.3	Member Data Documentation	68
6.10.3.1	id	68
6.10.3.2	intID	68
6.10.3.3	intPtr	68
6.10.3.4	addr	69
6.10.3.5	ver	69
6.10.3.6	nlt	69
6.11	PimNeighborTable Class Reference	69
6.11.1	Detailed Description	70
6.11.2	Member Function Documentation	70
6.11.2.1	getNeighbor	70
6.11.2.2	addNeighbor	70
6.11.2.3	deleteNeighbor	70
6.11.2.4	getNumNeighbors	70
6.11.2.5	printPimNeighborTable	70
6.11.2.6	getNeighborsByIntID	71
6.11.2.7	getNeighborsByID	71
6.11.2.8	getIdCounter	72
6.11.2.9	isInTable	72
6.11.2.10	findNeighbor	72
6.11.2.11	getNumNeighborsOnInt	72
6.11.2.12	handleMessage	73
6.11.3	Member Data Documentation	73
6.11.3.1	id	73
6.11.3.2	nt	73
6.12	PimNeighborTableAccess Class Reference	73
6.12.1	Detailed Description	74
6.13	pimSM Class Reference	74
6.13.1	Detailed Description	74
6.14	PimSplitter Class Reference	74
6.14.1	Detailed Description	75

6.14.2	Member Function Documentation	75
6.14.2.1	processPIMPkt	75
6.14.2.2	processNLTimer	76
6.14.2.3	createHelloPkt	76
6.14.2.4	sendHelloPkt	77
6.14.2.5	processHelloPkt	77
6.14.2.6	receiveChangeNotification	78
6.14.2.7	newMulticast	79
6.14.2.8	igmpChange	80
6.14.2.9	LoadConfigFromXML	81
6.14.2.10	handleMessage	82
6.14.2.11	initialize	83
6.14.3	Member Data Documentation	84
6.14.3.1	rt	84
6.14.3.2	mrt	84
6.14.3.3	ift	84
6.14.3.4	nb	84
6.14.3.5	pimIft	85
6.14.3.6	pimNbt	85
6.14.3.7	hostname	85
7	File Documentation	87
7.1	F:/ANSA/src/ansa/ipv4/AnsaIP.cc File Reference	87
7.1.1	Detailed Description	87
7.2	AnsaIP.cc	87
7.3	F:/ANSA/src/ansa/ipv4/AnsaIP.h File Reference	90
7.3.1	Detailed Description	91
7.4	AnsaIP.h	91
7.5	F:/ANSA/src/ansa/multicastRoutingTable/MulticastIPRoute.cc File Reference	92
7.5.1	Detailed Description	92
7.6	MulticastIPRoute.cc	92
7.7	F:/ANSA/src/ansa/multicastRoutingTable/MulticastIPRoute.h File Reference	94
7.7.1	Detailed Description	94
7.7.2	Class Documentation	95
7.7.2.1	struct inInterface	95
7.7.2.2	struct outInterface	95
7.7.3	Typedef Documentation	95
7.7.3.1	InterfaceVector	95
7.7.4	Enumeration Type Documentation	95
7.7.4.1	flag	95

7.7.4.2	intState	96
7.7.4.3	AssertState	96
7.8	MulticastIPRoute.h	96
7.9	F:/ANSA/src/ansa/multicastRoutingTable/MulticastRoutingTable.cc File Reference	98
7.9.1	Detailed Description	98
7.9.2	Function Documentation	98
7.9.2.1	operator<<	98
7.10	MulticastRoutingTable.cc	98
7.11	F:/ANSA/src/ansa/multicastRoutingTable/MulticastRoutingTable.h File Reference	102
7.11.1	Detailed Description	102
7.11.2	Typedef Documentation	103
7.11.2.1	RouteVector	103
7.12	MulticastRoutingTable.h	103
7.13	F:/ANSA/src/ansa/multicastRoutingTable/MulticastRoutingTableAccess.h File Reference	104
7.13.1	Detailed Description	104
7.14	MulticastRoutingTableAccess.h	104
7.15	F:/ANSA/src/ansa/pim/modes/pimDM.cc File Reference	104
7.15.1	Detailed Description	105
7.16	pimDM.cc	105
7.17	F:/ANSA/src/ansa/pim/modes/pimDM.h File Reference	118
7.17.1	Detailed Description	119
7.17.2	Define Documentation	119
7.17.2.1	PT	119
7.17.2.2	GRT	119
7.17.2.3	SAT	119
7.17.2.4	SRT	119
7.18	pimDM.h	120
7.19	F:/ANSA/src/ansa/pim/modes/pimSM.cc File Reference	121
7.19.1	Detailed Description	121
7.20	pimSM.cc	121
7.21	F:/ANSA/src/ansa/pim/modes/pimSM.h File Reference	121
7.21.1	Detailed Description	122
7.22	pimSM.h	122
7.23	F:/ANSA/src/ansa/pim/PimSplitter.cc File Reference	122
7.23.1	Detailed Description	123
7.24	PimSplitter.cc	123
7.25	F:/ANSA/src/ansa/pim/PimSplitter.h File Reference	128
7.25.1	Detailed Description	129
7.25.2	Define Documentation	129
7.25.2.1	HT	129

7.26	PimSplitter.h	130
7.27	F:/ANSA/src/ansa/pim/tables/PimInterfaceTable.cc File Reference	130
7.27.1	Detailed Description	131
7.27.2	Function Documentation	131
7.27.2.1	operator<<	131
7.27.2.2	operator<<	131
7.28	PimInterfaceTable.cc	132
7.29	F:/ANSA/src/ansa/pim/tables/PimInterfaceTable.h File Reference	133
7.29.1	Detailed Description	134
7.29.2	Enumeration Type Documentation	134
7.29.2.1	PIMmode	134
7.30	PimInterfaceTable.h	134
7.31	F:/ANSA/src/ansa/pim/tables/PimNeighborTable.cc File Reference	135
7.31.1	Detailed Description	136
7.31.2	Function Documentation	136
7.31.2.1	operator<<	136
7.32	PimNeighborTable.cc	136
7.33	F:/ANSA/src/ansa/pim/tables/PimNeighborTable.h File Reference	138
7.33.1	Detailed Description	138
7.34	PimNeighborTable.h	138

Chapter 1

Multicast Routing Modelling in OMNeT++ Documentation

This is programming documentation to thesis Multicast Routing Modelling in OMNeT++ by Veronika Rybova. In this documentation you can see whole C++ programming part of the thesis. You do not find here description of NED files and classes which are autogenerated from files PIMTimer.msg and PIMPacket.msg. The reason is that the output of these files is big amount of classes and documentation would be overwhelmed.

The thesis is part of project ANSA, which takes place at the Faculty of Information Technology, Brno University of Technology. For more information visit: <https://nes.fit.vutbr.cz/ansa/pmwiki.php>

Chapter 2

Directory Hierarchy

2.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

F:	9
ANSA	9
src	10
ansa	9
ipv4	9
multicastRoutingTable	10
pim	10
modes	9
tables	11

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

addRemoveAddr	Class is needed by notification about new multicast addresses on interface	13
AnsaIP	Class is extension of the IP protocol implementation for multicast	14
MulticastIPRoute	Class represents one entry of MulticastRoutingTable	19
MulticastRoutingTable	Class represent multicast routing table	25
MulticastRoutingTableAccess	Class gives access to the MulticastRoutingTable	34
pimDM	Class implements PIM-DM (dense mode)	34
PimInterface	Class represents one entry of PimInterfaceTable	60
PimInterfaceTable	Class represents Pim Interface Table	63
PimInterfaceTableAccess	Class gives access to the PimInterfaceTable	65
PimNeighbor	Class represents one entry of PimNeighborTable	66
PimNeighborTable	Class represents Pim Neighbor Table	69
PimNeighborTableAccess	Class gives access to the PimNeighborTable	73
pimSM	Class implements PIM-SM (sparse mode)	74
PimSplitter	Class implements PIM Splitter, which splits PIM messages to correct PIM module	74

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

F:/ANSA/src/ansa/ipv4/ AnsaIP.cc	
File contains extension of class IP, which can work also with multicast data and multicast	87
F:/ANSA/src/ansa/ipv4/ AnsaIP.h	
File contains extension of class IP, which can work also with multicast data and multicast	91
F:/ANSA/src/ansa/multicastRoutingTable/ MulticastIPRoute.cc	
File contains implementation of multicast route	92
F:/ANSA/src/ansa/multicastRoutingTable/ MulticastIPRoute.h	
File contains implementation of multicast route	96
F:/ANSA/src/ansa/multicastRoutingTable/ MulticastRoutingTable.cc	
File contains implementation of multicast routing table	98
F:/ANSA/src/ansa/multicastRoutingTable/ MulticastRoutingTable.h	
File contains implementation of multicast routing table	103
F:/ANSA/src/ansa/multicastRoutingTable/ MulticastRoutingTableAccess.h	
File contains implementation of access class	104
F:/ANSA/src/ansa/pim/ PimSplitter.cc	
File contains implementation of PIMSplitter	123
F:/ANSA/src/ansa/pim/ PimSplitter.h	
File contains implementation of PIMSplitter	130
F:/ANSA/src/ansa/pim/modes/ pimDM.cc	
File implements PIM dense mode	105
F:/ANSA/src/ansa/pim/modes/ pimDM.h	
File implements PIM dense mode	120
F:/ANSA/src/ansa/pim/modes/ pimSM.cc	
File implements PIM sparse mode	121
F:/ANSA/src/ansa/pim/modes/ pimSM.h	
File implements PIM sparse mode	122
F:/ANSA/src/ansa/pim/tables/ PimInterfaceTable.cc	
File implements table of PIM interfaces	132
F:/ANSA/src/ansa/pim/tables/ PimInterfaceTable.h	
File implements table of PIM interfaces	134
F:/ANSA/src/ansa/pim/tables/ PimNeighborTable.cc	
File implements table of PIM neighbors	136
F:/ANSA/src/ansa/pim/tables/ PimNeighborTable.h	
File implements table of PIM neighbors	138

Chapter 5

Directory Documentation

5.1 F:/ANSA/src/ansa/ Directory Reference

Directories

- directory [ipv4](#)
- directory [multicastRoutingTable](#)
- directory [pim](#)

5.2 F:/ANSA/ Directory Reference

Directories

- directory [src](#)

5.3 F:/ Directory Reference

Directories

- directory [ANSA](#)

5.4 F:/ANSA/src/ansa/ipv4/ Directory Reference

Files

- file [AnsaIP.cc](#)
File contains extension of class IP, which can work also with multicast data and multicast.
- file [AnsaIP.h](#)
File contains extension of class IP, which can work also with multicast data and multicast.

5.5 F:/ANSA/src/ansa/pim/modes/ Directory Reference

Files

- file [pimDM.cc](#)

File implements PIM dense mode.

- file [pimDM.h](#)

File implements PIM dense mode.

- file [pimSM.cc](#)

File implements PIM sparse mode.

- file [pimSM.h](#)

File implements PIM sparse mode.

5.6 F:/ANSA/src/ansa/multicastRoutingTable/ Directory Reference

Files

- file [MulticastIPRoute.cc](#)

File contains implementation of multicast route.

- file [MulticastIPRoute.h](#)

File contains implementation of multicast route.

- file [MulticastRoutingTable.cc](#)

File contains implementation of multicast routing table.

- file [MulticastRoutingTable.h](#)

File contains implementation of multicast routing table.

- file [MulticastRoutingTableAccess.h](#)

File contains implementation of access class.

5.7 F:/ANSA/src/ansa/pim/ Directory Reference

Directories

- directory [modes](#)
- directory [tables](#)

Files

- file [PimSplitter.cc](#)

File contains implementation of PIMSplitter.

- file [PimSplitter.h](#)

File contains implementation of PIMSplitter.

5.8 F:/ANSA/src/ Directory Reference

Directories

- directory [ansa](#)

5.9 F:/ANSA/src/ansa/pim/tables/ Directory Reference

Files

- file [PimInterfaceTable.cc](#)
File implements table of PIM interfaces.
- file [PimInterfaceTable.h](#)
File implements table of PIM interfaces.
- file [PimNeighborTable.cc](#)
File implements table of PIM neighbors.
- file [PimNeighborTable.h](#)
File implements table of PIM neighbors.

Chapter 6

Class Documentation

6.1 addRemoveAddr Class Reference

Class is needed by notification about new multicast addresses on interface.

```
#include <PimInterfaceTable.h>
```

Public Member Functions

- virtual std::string **info** () const
- void **setAddr** (std::vector< IPAddress > **addr**)
- void **setInt** (PimInterface ***pimInt**)
- std::vector< IPAddress > **getAddr** ()
- int **getAddrSize** ()
- PimInterface * **getInt** ()

Protected Attributes

- std::vector< IPAddress > **addr**
- PimInterface * **pimInt**

6.1.1 Detailed Description

Class is needed by notification about new multicast addresses on interface.

If you do not use notification board, you probably do not need this class. The problem is that method fireChange-Notification needs object as the second parameter.

Definition at line 117 of file [PimInterfaceTable.h](#).

6.1.2 Member Function Documentation

6.1.2.1 void addRemoveAddr::setAddr (std::vector< IPAddress > **addr**) [inline]

Set addresses to the object.

Definition at line 134 of file [PimInterfaceTable.h](#).

6.1.2.2 `void addRemoveAddr::setInt (PimInterface * pimInt) [inline]`

Set pointer to interface to the object.

Definition at line 135 of file [PimInterfaceTable.h](#).

6.1.2.3 `std::vector<IPAddress> addRemoveAddr::getAddr () [inline]`

Get addresses from the object.

Definition at line 136 of file [PimInterfaceTable.h](#).

6.1.2.4 `int addRemoveAddr::getAddrSize () [inline]`

Returns size of addresses vector.

Definition at line 137 of file [PimInterfaceTable.h](#).

6.1.2.5 `PimInterface* addRemoveAddr::getInt () [inline]`

Get pointer to interface from the object.

Definition at line 138 of file [PimInterfaceTable.h](#).

6.1.3 Member Data Documentation

6.1.3.1 `std::vector<IPAddress> addRemoveAddr::addr [protected]`

Vector of added or removed addresses.

Definition at line 120 of file [PimInterfaceTable.h](#).

6.1.3.2 `PimInterface* addRemoveAddr::pimInt [protected]`

Pointer to interface.

Definition at line 121 of file [PimInterfaceTable.h](#).

The documentation for this class was generated from the following file:

- [F:/ANSA/src/ansa/pim/tables/PimInterfaceTable.h](#)

6.2 AnsaIP Class Reference

Class is extension of the IP protocol implementation for multicast.

```
#include <AnsaIP.h>
```

Protected Member Functions

- virtual void [handlePacketFromNetwork](#) (IPDatagram *datagram)
- virtual void [routeMulticastPacket](#) (IPDatagram *datagram, InterfaceEntry *destIE, InterfaceEntry *fromIE)
- virtual int [numInitStages](#) () const
- virtual void [initialize](#) (int stage)

Private Attributes

- [MulticastRoutingTable](#) * `mrt`
- [NotificationBoard](#) * `nb`

6.2.1 Detailed Description

Class is extension of the IP protocol implementation for multicast.

It extends class IP mainly by methods processing multicast stream.

Definition at line 42 of file [AnsaIP.h](#).

6.2.2 Member Function Documentation

6.2.2.1 `void AnsaIP::handlePacketFromNetwork (IPDatagram * datagram)` `[protected, virtual]`

HANDLE PACKET FROM NETWORK

Extension of method [handlePacketFromNetwork\(\)](#) from class IP. It stops deleting of IP Control Info, which will be needed in [routeMulticastPacket\(\)](#). All necessary info is added to IP Control Info. Datagram with protocol number 103 (IP_PROT_PIM) are sent directly to the PIM module.

All part which I added are signed by MYWORK tag.

Parameters

<i>datagram</i>	Pointer to incoming datagram.
-----------------	-------------------------------

See also

[routeMulticastPacket\(\)](#)

Definition at line 44 of file [AnsaIP.cc](#).

```
{
    //
    // "Prerouting"
    //

    // check for header biterror
    if (datagram->hasBitError())
    {
        // probability of bit error in header = size of header / size of total
        // message
        // (ignore bit error if in payload)
        double relativeHeaderLength = datagram->getHeaderLength() / (double)
        datagram->getByteLength();
        if (dblrand() <= relativeHeaderLength)
        {
            EV << "bit error found, sending ICMP_PARAMETER_PROBLEM\n";
            icmpAccess.get()->sendErrorMessage(datagram, ICMP_PARAMETER_PROBLEM
            , 0);
            return;
        }
    }

    // remove control info
    if (datagram->getTransportProtocol() != IP_PROT_DSR && datagram->
        getTransportProtocol() != IP_PROT_MANET && !datagram->getDestAddress().isMulticast() &&
        datagram->getTransportProtocol() != IP_PROT_PIM)
    {
        delete datagram->removeControlInfo();
    }
    else if (datagram->getMoreFragments())
        delete datagram->removeControlInfo(); // delete all control message
        except the last

    //MYWORK Add all necessary info to the IP Control Info for future use.
    if (datagram->getDestAddress().isMulticast() || datagram->
```

```

    getTransportProtocol() == IP_PROT_PIM)
{
    IPControlInfo *ctrl = (IPControlInfo*) (datagram->removeControlInfo());
    ctrl->setSrcAddr(datagram->getSrcAddress());
    ctrl->setDestAddr(datagram->getDestAddress());
    ctrl->setInterfaceId(getSourceInterfaceFrom(datagram)->getInterfaceId());
    datagram->setControlInfo(ctrl);
}

// hop counter decrement; FIXME but not if it will be locally delivered
datagram->setTimeToLive(datagram->getTimeToLive()-1);

// send IGMP packet to IGMP module
if (datagram->getTransportProtocol() == IP_PROT_IGMP)
{
    cPacket *packet = decapsulateIP(datagram);
    send(packet, "transportOut", mapping.getOutputGateForProtocol(
IP_PROT_IGMP));
    return;
}

//MYWORK send PIM packet to PIM module
if (datagram->getTransportProtocol() == IP_PROT_PIM)
{
    cPacket *packet = decapsulateIP(datagram);
    send(packet, "transportOut", mapping.getOutputGateForProtocol(
IP_PROT_PIM));
    return;
}

//MYWORK route packet
if (!datagram->getDestAddress().isMulticast())
    routePacket(datagram, NULL, false, NULL);
else
    routeMulticastPacket(datagram, NULL, getSourceInterfaceFrom(datagram));
}

```

6.2.2.2 void AnsalP::routeMulticastPacket (IPDatagram * *datagram*, InterfaceEntry * *destIE*, InterfaceEntry * *fromIE*)
[protected, virtual]

ROUTE MULTICAST PACKET

Extension of method [routeMulticastPacket\(\)](#) from class IP. The method checks if data come to RPF interface, if not it sends notification. Multicast data which are sent by this router and has given outgoing interface are sent directly (PIM messages). The method finds route for group. If there is no route, it will be added. Then packet is copied and sent to all outgoing interfaces in route.

All part which I added are signed by MYWORK tag.

Parameters

<i>datagram</i>	Pointer to incoming datagram.
<i>destIE</i>	Pointer to outgoing interface.
<i>fromIE</i>	Pointer to incoming interface.

See also

[routeMulticastPacket\(\)](#)

Definition at line 124 of file [AnsalP.cc](#).

```

{
    IPAddress destAddr = datagram->getDestAddress();
    IPAddress srcAddr = datagram->getSrcAddress();
    IPControlInfo *ctrl = (IPControlInfo *) datagram->getControlInfo();
    EV << "Routing multicast datagram '" << datagram->getName() << "' with
    dest=" << destAddr << "\n";
    MulticastIPRoute *route = mrt->getRouteFor(destAddr, srcAddr);

    numMulticast++;

    // Process datagram only if sent locally or arrived on the shortest

```

```

// route (provided routing table already contains srcAddr) = RPF interface;
// otherwise discard and continue.
InterfaceEntry *rpfInt = rt->getInterfaceForDestAddr(datagram->
getSrcAddress());
if (fromIE!=NULL && rpfInt!=NULL && fromIE!=rpfInt)
{
    //MYWORK RPF interface has changed
    /*if (route != NULL && (route->getInIntId() !=
rpfInt->getInterfaceId()))
    {
        EV << "RPF interface has changed" << endl;
        nb->fireChangeNotification(NF_IPv4_RPF_CHANGE, route);
    }*/
    //MYWORK Data come to non-RPF interface
    if (!rt->isLocalMulticastAddress(destAddr) && !destAddr.
isLinkLocalMulticast())
    {
        EV << "Data on non-RPF interface" << endl;
        nb->fireChangeNotification(NF_IPv4_DATA_ON_NONRPF, ctrl);
        return;
    }
    else
    {
        // FIXME count dropped
        EV << "Packet dropped." << endl;
        delete datagram;
        return;
    }
}

//MYWORK for local traffic to given destination (PIM messages)
if (fromIE == NULL && destIE != NULL)
{
    IPDatagram *datagramCopy = (IPDatagram *) datagram->dup();
    datagramCopy->setSrcAddress(destIE->ipv4Data()->getIPAddress());
    ;
    fragmentAndSend(datagramCopy, destIE, destAddr);

    delete datagram;
    return;
}

// if received from the network...
if (fromIE!=NULL)
{
    EV << "Packet was received from the network..." << endl;
    // check for local delivery (multicast assigned to any interface)
    if (rt->isLocalMulticastAddress(destAddr))
    {
        EV << "isLocalMulticastAddress." << endl;
        IPDatagram *datagramCopy = (IPDatagram *) datagram->dup();

        // FIXME code from the MPLS model: set packet dest address to
routerId
        datagramCopy->setDestAddress(rt->getRouterId());
        reassembleAndDeliver(datagramCopy);
    }

    // don't forward if IP forwarding is off
    if (!rt->isIPForwardingEnabled())
    {
        EV << "IP forwarding is off." << endl;
        delete datagram;
        return;
    }

    // don't forward if dest address is link-scope
    // address is in the range 224.0.0.0 to 224.0.0.255
    if (destAddr.isLinkLocalMulticast())
    {
        EV << "isLinkLocalMulticast." << endl;
        delete datagram;
        return;
    }
}

//MYWORK(to the end) now: routing
EV << "AnsaIP::routeMulticastPacket - Multicast routing." << endl;

// multicast group is not in multicast routing table and has to be added
if (route == NULL)
{
    EV << "AnsaIP::routeMulticastPacket - Multicast route does not exist,
try to add." << endl;
    nb->fireChangeNotification(NF_IPv4_NEW_MULTICAST, ctrl);
    delete datagram->removeControlInfo();
    ctrl = NULL;
}

```

```

    // read new record
    route = mrt->getRouteFor(destAddr, srcAddr);
}

if (route == NULL)
{
    EV << "Still do not exist." << endl;
    delete datagram;
    return;
}

nb->fireChangeNotification(NF_IPv4_DATA_ON_RPF, route);

// data won't be sent because there is no outgoing interface and/or
// route is pruned
InterfaceVector outInt = route->getOutInt();
if (outInt.size() == 0 || route->isFlagSet(P))
{
    EV << "Route does not have any outgoing interface or it is pruned." <<
    endl;
    if(ctrl != NULL)
    {
        if (!route->isFlagSet(A))
            nb->fireChangeNotification(
            NF_IPv4_DATA_ON_PRUNED_INT, ctrl);
        delete datagram;
        return;
    }
}

// send packet to all outgoing interfaces of route (oilst)
for (unsigned int i=0; i<outInt.size(); i++)
{
    // do not send to pruned interface
    if (outInt[i].forwarding == Pruned)
        continue;

    InterfaceEntry *destIE = outInt[i].intPtr;
    IPDatagram *datagramCopy = (IPDatagram *) datagram->dup();

    // set datagram source address if not yet set
    if (datagramCopy->getSrcAddress().isUnspecified())
        datagramCopy->setSrcAddress(destIE->ipv4Data()->
        getIPAddress());

    // send
    fragmentAndSend(datagramCopy, destIE, destAddr);
}

// only copies sent, delete original datagram
delete datagram;
}

```

6.2.2.3 void AnsalP::initialize (int stage) [protected, virtual]

INITIALIZE

The method initialize ale structures (tables) which will use.

Parameters

<i>stage</i>	Stage of initialization.
--------------	--------------------------

Definition at line 23 of file [AnsalP.cc](#).

```

{
    INET_API IP::initialize();

    mrt = MulticastRoutingTableAccess().get();
    nb = NotificationBoardAccess().get();
}

```

6.2.3 Member Data Documentation

6.2.3.1 MulticastRoutingTable* AnsaIP::mrt [private]

Pointer to multicast routing table.

Definition at line 45 of file [AnsaIP.h](#).

6.2.3.2 NotificationBoard* AnsaIP::nb [private]

Pointer to notification table.

Definition at line 46 of file [AnsaIP.h](#).

The documentation for this class was generated from the following files:

- F:/ANSA/src/ansa/ipv4/[AnsaIP.h](#)
- F:/ANSA/src/ansa/ipv4/[AnsaIP.cc](#)

6.3 MulticastIPRoute Class Reference

Class represents one entry of [MulticastRoutingTable](#).

```
#include <MulticastIPRoute.h>
```

Public Member Functions

- [MulticastIPRoute](#) ()
- virtual std::string **info** () const
- virtual std::string **detailedInfo** () const
- void [setSource](#) (IPAddress [source](#))
- void [setGroup](#) (IPAddress [group](#))
- void [setRP](#) (IPAddress [RP](#))
- void [setGrt](#) (PIMgrt *[grt](#))
- void [setSat](#) (PIMsat *[sat](#))
- void [setSrt](#) (PIMsrt *[srt](#))
- void [setFlags](#) (std::vector< [flag](#) > [flags](#))
- bool [isFlagSet](#) ([flag](#) fl)
- void [addFlag](#) ([flag](#) fl)
- void [removeFlag](#) ([flag](#) fl)
- void [setInInt](#) (InterfaceEntry *[interfacePtr](#), int [intId](#), IPAddress [nextHop](#))
- void [setInInt](#) ([inInterface](#) [inInt](#))
- void [setOutInt](#) (InterfaceVector [outInt](#))
- void [addOutInt](#) (outInterface [outInt](#))
- bool [isRpf](#) (int [intId](#))
- bool [isOilistNull](#) ()
- IPAddress [getSource](#) () const
- IPAddress [getGroup](#) () const
- IPAddress [getRP](#) () const
- PIMgrt * [getGrt](#) () const
- PIMsat * [getSat](#) () const
- PIMsrt * [getSrt](#) () const
- std::vector< [flag](#) > [getFlags](#) () const
- [inInterface](#) [getInInt](#) () const
- InterfaceEntry * [getInIntPtr](#) () const
- int [getInIntId](#) () const
- IPAddress [getInIntNextHop](#) () const

- [InterfaceVector](#) `getOutInt ()` const
- [outInterface](#) `getOutIntByIntId (int intId)`
- `int` [getOutIdByIntId \(int intId\)](#)
- `simtime_t` `getInstallTime ()` const
- `void` `setInstallTime (simtime_t time)`
- `void` `setSequencenumber (int i)`
- `unsigned int` `getSequencenumber ()` const

Private Member Functions

- [MulticastIPRoute](#) & `operator=` (const [MulticastIPRoute](#) &obj)

Private Attributes

- `IPAddress` `source`
- `IPAddress` `group`
- `IPAddress` `RP`
- `std::vector< flag >` `flags`
- `PIMgrt *` `grt`
- `PIMsat *` `sat`
- `PIMsrt *` `srt`
- [inInterface](#) `inInt`
- [InterfaceVector](#) `outInt`
- `unsigned int` `sequencenumber`
- `simtime_t` `installtime`

6.3.1 Detailed Description

Class represents one entry of [MulticastRoutingTable](#).

Definition at line 83 of file [MulticastIPRoute.h](#).

6.3.2 Constructor & Destructor Documentation

6.3.2.1 [MulticastIPRoute::MulticastIPRoute \(\)](#)

Set all pointers to null

Definition at line 12 of file [MulticastIPRoute.cc](#).

```
{
    inInt.intPtr = NULL;
    grt = NULL;
    sat = NULL;
    srt = NULL;
}
```

6.3.3 Member Function Documentation

6.3.3.1 `void MulticastIPRoute::setSource (IPAddress source)` [inline]

Set multicast source IP address

Definition at line 115 of file [MulticastIPRoute.h](#).

6.3.3.2 void MulticastIPRoute::setGroup (IPAddress *group*) [inline]

Set multicast group IP address

Definition at line 116 of file [MulticastIPRoute.h](#).

6.3.3.3 void MulticastIPRoute::setRP (IPAddress *RP*) [inline]

Set RP IP address

Definition at line 117 of file [MulticastIPRoute.h](#).

6.3.3.4 void MulticastIPRoute::setGrt (PIMgrt * *grt*) [inline]

Set pointer to PimGraftTimer

Definition at line 118 of file [MulticastIPRoute.h](#).

6.3.3.5 void MulticastIPRoute::setSat (PIMsat * *sat*) [inline]

Set pointer to PimSourceActiveTimer

Definition at line 119 of file [MulticastIPRoute.h](#).

6.3.3.6 void MulticastIPRoute::setSrt (PIMsrt * *srt*) [inline]

Set pointer to PimStateRefreshTimer

Definition at line 120 of file [MulticastIPRoute.h](#).

6.3.3.7 void MulticastIPRoute::setFlags (std::vector< flag > *flags*) [inline]

Set vector of flags (flag)

Definition at line 122 of file [MulticastIPRoute.h](#).

6.3.3.8 bool MulticastIPRoute::isFlagSet (flag *f1*)

Returns if flag is set to entry or not

Definition at line 52 of file [MulticastIPRoute.cc](#).

```
{
    for(unsigned int i = 0; i < flags.size(); i++)
    {
        if (flags[i] == f1)
            return true;
    }
    return false;
}
```

6.3.3.9 void MulticastIPRoute::addFlag (flag *f1*)

Add flag to ineterface

Definition at line 62 of file [MulticastIPRoute.cc](#).

```
{
    if (!isFlagSet(f1))
        flags.push_back(f1);
}
```

6.3.3.10 void MulticastIPRoute::removeFlag (flag fl)

Remove flag from ineterface

Definition at line 68 of file [MulticastIPRoute.cc](#).

```
{
    for(unsigned int i = 0; i < flags.size(); i++)
    {
        if (flags[i] == fl)
        {
            flags.erase(flags.begin() + i);
            return;
        }
    }
}
```

6.3.3.11 void MulticastIPRoute::setInInt (InterfaceEntry * interfacePtr, int intId, IPAddress nextHop) [inline]

Set information about incoming interface

Definition at line 127 of file [MulticastIPRoute.h](#).

6.3.3.12 void MulticastIPRoute::setInInt (inInterface inInt) [inline]

Set incoming interface

Definition at line 128 of file [MulticastIPRoute.h](#).

6.3.3.13 void MulticastIPRoute::setOutInt (InterfaceVector outInt) [inline]

Set list of outgoing interfaces

Definition at line 130 of file [MulticastIPRoute.h](#).

6.3.3.14 void MulticastIPRoute::addOutInt (outInterface outInt) [inline]

Add interface to list of outgoing interfaces

Definition at line 131 of file [MulticastIPRoute.h](#).

6.3.3.15 bool MulticastIPRoute::isRpf (int intId) [inline]

Returns if given interface is RPF or not

Definition at line 133 of file [MulticastIPRoute.h](#).

6.3.3.16 bool MulticastIPRoute::isOlistNull ()

Returns true if list of outgoing interfaces is empty, otherwise false

Definition at line 91 of file [MulticastIPRoute.cc](#).

```
{
    bool olistNull = true;
    for (unsigned int i = 0; i < outInt.size(); i++)
    {
        if (outInt[i].forwarding == Forward)
        {
            olistNull = false;
            break;
        }
    }
}
```

```

    }
    return olistNull;
}

```

6.3.3.17 IPAddress MulticastIPRoute::getSource () const [inline]

Get multicast source IP address

Definition at line 137 of file [MulticastIPRoute.h](#).

6.3.3.18 IPAddress MulticastIPRoute::getGroup () const [inline]

Get multicast group IP address

Definition at line 138 of file [MulticastIPRoute.h](#).

6.3.3.19 IPAddress MulticastIPRoute::getRP () const [inline]

Get RP IP address

Definition at line 139 of file [MulticastIPRoute.h](#).

6.3.3.20 PIMgrt* MulticastIPRoute::getGrt () const [inline]

Get pointer to PimGraftTimer

Definition at line 140 of file [MulticastIPRoute.h](#).

6.3.3.21 PIMsat* MulticastIPRoute::getSat () const [inline]

Get pointer to PimSourceActiveTimer

Definition at line 141 of file [MulticastIPRoute.h](#).

6.3.3.22 PIMsrt* MulticastIPRoute::getSrt () const [inline]

Get pointer to PimStateRefreshTimer

Definition at line 142 of file [MulticastIPRoute.h](#).

6.3.3.23 std::vector<flag> MulticastIPRoute::getFlags () const [inline]

Get list of route flags

Definition at line 143 of file [MulticastIPRoute.h](#).

6.3.3.24 inInterface MulticastIPRoute::getInInt () const [inline]

Get incoming interface

Definition at line 146 of file [MulticastIPRoute.h](#).

6.3.3.25 InterfaceEntry* MulticastIPRoute::getInIntPtr () const [inline]

Get pointer to incoming interface

Definition at line 147 of file [MulticastIPRoute.h](#).

6.3.3.26 int MulticastIPRoute::getInIntId () const [inline]

Get ID of incoming interface

Definition at line 148 of file [MulticastIPRoute.h](#).

6.3.3.27 IPAddress MulticastIPRoute::getInIntNextHop () const [inline]

Get IP address of next hop for incoming interface

Definition at line 149 of file [MulticastIPRoute.h](#).

6.3.3.28 InterfaceVector MulticastIPRoute::getOutInt () const [inline]

Get list of outgoing interfaces

Definition at line 152 of file [MulticastIPRoute.h](#).

6.3.3.29 outInterface MulticastIPRoute::getOutIntByIntId (int intId)

Get outgoing interface with given interface ID

6.3.3.30 int MulticastIPRoute::getOutIdByIntId (int intId)

Get sequence number of outgoing interface with given interface ID

Definition at line 80 of file [MulticastIPRoute.cc](#).

```
{
    unsigned int i;
    for (i = 0; i < outInt.size(); i++)
    {
        if (outInt[i].intId == intId)
            break;
    }
    return i;
}
```

6.3.4 Member Data Documentation**6.3.4.1 IPAddress MulticastIPRoute::source** [private]

Source of multicast

Definition at line 86 of file [MulticastIPRoute.h](#).

6.3.4.2 IPAddress MulticastIPRoute::group [private]

Multicast group

Definition at line 87 of file [MulticastIPRoute.h](#).

6.3.4.3 IPAddress MulticastIPRoute::RP [private]

Randevous point

Definition at line 88 of file [MulticastIPRoute.h](#).

6.3.4.4 `std::vector<flag> MulticastIPRoute::flags` [private]

Route flags

Definition at line 89 of file [MulticastIPRoute.h](#).

6.3.4.5 `PIMgrt* MulticastIPRoute::grt` [private]

Pointer to Graft Retry Timer

Definition at line 91 of file [MulticastIPRoute.h](#).

6.3.4.6 `PIMsat* MulticastIPRoute::sat` [private]

Pointer to Source Active Timer

Definition at line 92 of file [MulticastIPRoute.h](#).

6.3.4.7 `PIMsrt* MulticastIPRoute::srt` [private]

Pointer to State Refresh Timer

Definition at line 93 of file [MulticastIPRoute.h](#).

6.3.4.8 `inInterface MulticastIPRoute::inInt` [private]

Incoming interface

Definition at line 95 of file [MulticastIPRoute.h](#).

6.3.4.9 `InterfaceVector MulticastIPRoute::outInt` [private]

Outgoing interface

Definition at line 96 of file [MulticastIPRoute.h](#).

The documentation for this class was generated from the following files:

- [F:/ANSA/src/ansa/multicastRoutingTable/MulticastIPRoute.h](#)
- [F:/ANSA/src/ansa/multicastRoutingTable/MulticastIPRoute.cc](#)

6.4 MulticastRoutingTable Class Reference

Class represent multicast routing table.

```
#include <MulticastRoutingTable.h>
```

Public Member Functions

- void [generateShowIPRoute](#) ()
- virtual void [printRoutingTable](#) () const
- virtual std::vector
`< MulticastIPRoute * >` [getRouteFor](#) (IPAddress group)
- virtual [MulticastIPRoute *](#) [getRouteFor](#) (IPAddress group, IPAddress source)
- virtual std::vector
`< MulticastIPRoute * >` [getRoutesForSource](#) (IPAddress source)

- virtual int [getNumRoutes](#) () const
- virtual [MulticastIPRoute](#) * [getRoute](#) (int k) const
- virtual const [MulticastIPRoute](#) * [findRoute](#) (const IPAddress &source, const IPAddress &group, const IPAddress &RP, int intId, const IPAddress &nextHop) const
- virtual void [addRoute](#) (const [MulticastIPRoute](#) *entry)
- virtual bool [deleteRoute](#) (const [MulticastIPRoute](#) *entry)
- virtual [~MulticastRoutingTable](#) ()

Protected Member Functions

- virtual bool [routeMatches](#) (const [MulticastIPRoute](#) *entry, const IPAddress &source, const IPAddress &group, const IPAddress &RP, int intId, const IPAddress &nextHop) const
- virtual void [updateDisplayString](#) ()
- virtual int [numInitStages](#) () const
- virtual void [initialize](#) (int stage)
- virtual void [handleMessage](#) (cMessage *)

Protected Attributes

- [RouteVector](#) [multicastRoutes](#)
- std::vector< std::string > [showMRoute](#)
- [InterfaceTable](#) * [ift](#)

6.4.1 Detailed Description

Class represent multicast routing table.

It contains entries = routes for each multicast source and group. There are methods to get right route from table, add new route, delete old on, etc.

Definition at line 29 of file [MulticastRoutingTable.h](#).

6.4.2 Constructor & Destructor Documentation

6.4.2.1 [MulticastRoutingTable::~~MulticastRoutingTable](#) () [virtual]

MULTICAST ROUTING TABLE DESTRUCTOR

The method deletes Multicast Routing Table. Delete all entries in table.

Definition at line 25 of file [MulticastRoutingTable.cc](#).

```
{
    for (unsigned int i=0; i<multicastRoutes.size(); i++)
        delete multicastRoutes[i];
}
```

6.4.3 Member Function Documentation

6.4.3.1 [bool MulticastRoutingTable::routeMatches](#) (const [MulticastIPRoute](#) * *entry*, const IPAddress & *source*, const IPAddress & *group*, const IPAddress & *RP*, int *intId*, const IPAddress & *nextHop*) const [protected, virtual]

ROUTE MATCHES

Finds a match between route entry and given parameters.

Parameters

<i>entry</i>	Link to route.
<i>source</i>	IP address of multicast source.
<i>group</i>	IP address of multicast group.
<i>RP</i>	IP address of RP router.
<i>intId</i>	ID of incoming (RPF) interface.
<i>nextHop</i>	IP address of RPF neighbor.

Returns

Pointer to route in multicast table.

Definition at line 152 of file [MulticastRoutingTable.cc](#).

```
{
    if (!source.isUnspecified() && !source.equals(entry->getSource()))
        return false;
    if (!group.isUnspecified() && !group.equals(entry->getGroup()))
        return false;
    if (!RP.isUnspecified() && !RP.equals(entry->getRP()))
        return false;
    if (intId!=entry->getInIntId())
        return false;
    if (!nextHop.isUnspecified() && !nextHop.equals(entry->getInIntNextHop()))
        return false;
    return true;
}
```

6.4.3.2 void MulticastRoutingTable::updateDisplayString () [protected, virtual]

UPDATE DISPLAY STRING

Update string under multicast table icon - number of multicast routes.

Definition at line 56 of file [MulticastRoutingTable.cc](#).

```
{
    if (!ev.isGUI())
        return;

    char buf[80];
    sprintf(buf, "%d routes", multicastRoutes.size());
    getDisplayString().setTagArg("t", 0, buf);
}
```

6.4.3.3 void MulticastRoutingTable::generateShowIPMroute ()

GENERATE SHOW IP MROUTE

This method should be called after each change of multicast routing table. It is output which represents state of the table. Format is same as format on Cisco routers.

Definition at line 364 of file [MulticastRoutingTable.cc](#).

```
{
    EV << "MulticastRoutingTable::generateShowIPRoute()" << endl;
    showMRoute.clear();

    int n = getNumRoutes();
    const MulticastIPRoute* ipr;

    for (int i=0; i<n; i++)
    {
        ipr = getRoute(i);
        stringstream os;
        os << "(";
        if (ipr->getSource().isUnspecified()) os << "*", "; else os <<
```

```

ipr->getSource() << ", ";
    os << ipr->getGroup() << ")", ";
    if (!ipr->getRP().isUnspecified()) os << "RP is " << ipr->getRP
()<< ", ";
    os << "flags: ";
    vector<flag> flags = ipr->getFlags();
    for (unsigned int j = 0; j < flags.size(); j++)
    {
        EV << "MulticastRoutingTable::generateShowIPRoute():
Flag = " << flags[j] << endl;
        switch(flags[j])
        {
            case D:
                os << "D";
                break;
            case S:
                os << "S";
                break;
            case C:
                os << "C";
                break;
            case P:
                os << "P";
                break;
            case A:
                os << "A";
                break;
        }
    }
    os << endl;

    os << "Incoming interface: ";
    if (ipr->getInIntPtr()) os << ipr->getInIntPtr()->getName() <<
", ";

    os << "RPF neighbor " << ipr->getInIntNextHop() << endl;
    os << "Outgoing interface list:" << endl;

    InterfaceVector all = ipr->getOutInt();
    if (all.size() == 0)
        os << "Null" << endl;
    else
        for (unsigned int k = 0; k < all.size(); k++)
        {
            os << all[k].intPtr->getName() << ", ";
            if (all[k].forwarding == Forward) os << "
Forward/"; else os << "Pruned/";
            if (all[k].mode == Densemode) os << "Dense";
            else os << "Sparse";
            os << endl;
        }
    showMRoute.push_back(os.str());
}
stringstream out;
}

```

6.4.3.4 void MulticastRoutingTable::printRoutingTable() const [virtual]

PRINT ROUTING TABLE

Can be used for debugging purposes.

Definition at line 71 of file [MulticastRoutingTable.cc](#).

```

{
    EV << "-- Multicast routing table --\n";
    for (int i=0; i<getNumRoutes(); i++)
        EV << getRoute(i)->detailedInfo() << "\n";
    EV << "\n";
}

```

6.4.3.5 vector< MulticastIPRoute * > MulticastRoutingTable::getRouteFor (IPAddress group) [virtual]

GET ROUTE FOR

The method returns all routes from multicast routing table for given multicast group.

Parameters

<i>group</i>	IP address of multicast group.
--------------	--------------------------------

Returns

Vecotr of pointers to routes in multicast table.

See also

[getRoute\(\)](#)

Definition at line 178 of file [MulticastRoutingTable.cc](#).

```
{
    Enter_Method("getMulticastRoutesFor(%x)", group.getInt()); // note:
    str().c_str() too slow here here
    EV << "MulticastRoutingTable::getRouteFor - address = " << group << endl;
    vector<MulticastIPRoute*> routes;

    // search in multicast table
    int n = multicastRoutes.size();
    for (int i = 0; i < n; i++)
    {
        MulticastIPRoute *route = getRoute(i);
        if (route->getGroup().getInt() == group.getInt())
            routes.push_back(route);
    }

    return routes;
}
```

6.4.3.6 MulticastIPRoute * MulticastRoutingTable::getRouteFor (IPAddress *group*, IPAddress *source*)

[virtual]

GET ROUTE FOR

The method returns one route from multicast routing table for given group and source IP addresses.

Parameters

<i>group</i>	IP address of multicast group.
<i>source</i>	IP address of multicast source.

Returns

Pointer to found multicast route.

See also

[getRoute\(\)](#)

Definition at line 207 of file [MulticastRoutingTable.cc](#).

```
{
    Enter_Method("getMulticastRoutesFor(%x, %x)", group.getInt(), source.getInt()); // note: str().c_str() too slow here here
    EV << "MulticastRoutingTable::getRouteFor - group = " << group << ", source = " << source << endl;

    // search in multicast routing table
    MulticastIPRoute *route = NULL;
    int n = multicastRoutes.size();
    int i;
    // go through all multicast routes
    for (i = 0; i < n; i++)
```

```

{
    route = getRoute(i);
    if (route->getGroup().getInt() == group.getInt() && route->getSource().
        getInt() == source.getInt())
        break;
}

if (i == n)
    return NULL;
return route;
}

```

6.4.3.7 `std::vector< MulticastIPRoute * > MulticastRoutingTable::getRoutesForSource (IPAddress source)` [virtual]

GET ROUTES FOR SOURCES

The method returns all routes from multicast routing table for given source.

Parameters

<code>source</code>	IP address of multicast source.
---------------------	---------------------------------

Returns

Vector of found multicast routes.

See also

`getNetwork()`

Definition at line 238 of file [MulticastRoutingTable.cc](#).

```

{
    Enter_Method("getRoutesForSource(%x)", source.getInt()); // note:
    str().c_str() too slow here here
    EV << "MulticastRoutingTable::getRoutesForSource - source = " << source
    << endl;
    vector<MulticastIPRoute*> routes;

    // search in multicast table
    int n = multicastRoutes.size();
    int i;
    for (i = 0; i < n; i++)
    {
        //FIXME works only for classfull addresses (function getNetwork)
        !!!!
        MulticastIPRoute *route = getRoute(i);
        if (route->getSource().getNetwork().getInt() == source.getInt())
    }
        routes.push_back(route);
    }
    return routes;
}

```

6.4.3.8 `int MulticastRoutingTable::getNumRoutes () const` [virtual]

GET NUMBER OF ROUTES

Returns number of entries in multicast routing table.

Definition at line 119 of file [MulticastRoutingTable.cc](#).

```

{
    return multicastRoutes.size();
}

```

6.4.3.9 MulticastIPRoute * MulticastRoutingTable::getRoute (int k) const [virtual]

GET ROUTE

Returns the k-th route. The returned route cannot be modified; you must delete and re-add it instead. This rule is emphasized by returning a const pointer.

Definition at line 131 of file [MulticastRoutingTable.cc](#).

```
{
    if (k < (int)mcastRoutes.size())
        return mcastRoutes[k];

    return NULL;
}
```

6.4.3.10 const MulticastIPRoute * MulticastRoutingTable::findRoute (const IPAddress & source, const IPAddress & group, const IPAddress & RP, int intId, const IPAddress & nextHop) const [virtual]

FIND ROUTE

Finds route according to given parameters (source, group, RP, ID and next hop of incoming int).

Parameters

<i>source</i>	IP address of multicast source.
<i>group</i>	IP address of multicast group.
<i>RP</i>	IP address of RP router.
<i>intId</i>	ID of incoming (RPF) interface.
<i>nextHop</i>	IP address of RPF neighbor.

Returns

Pointer to route in multicast table.

See also

[getRoute\(\)](#)
[routeMatches\(\)](#)
[getNumRoutes\(\)](#)

Definition at line 104 of file [MulticastRoutingTable.cc](#).

```
{
    int n = getNumRoutes();
    for (int i=0; i<n; i++)
        if (routeMatches(getRoute(i), source, group, RP, intId, nextHop))
            return getRoute(i);
    return NULL;
}
```

6.4.3.11 void MulticastRoutingTable::addRoute (const MulticastIPRoute * entry) [virtual]

ADD ROUTE

Function check new multicast table entry and then add new entry to multicast table.

Parameters

<i>entry</i>	New entry about new multicast group.
--------------	--------------------------------------

See also

[MulticastIPRoute](#)
[updateDisplayString\(\)](#)
[generateShowIPRoute\(\)](#)

Definition at line 267 of file [MulticastRoutingTable.cc](#).

```

{
    Enter_Method("addMulticastRoute(...)");

    // check for null multicast group address
    if (entry->getGroup().isUnspecified())
        error("addMulticastRoute(): multicast group address cannot be NULL");

    // check that group address is multicast address
    if (!entry->getGroup().isMulticast())
        error("addMulticastRoute(): group address is not multicast address");

    // check for source or RP address
    if (entry->getSource().isUnspecified() && entry->getRP().isUnspecified())
        error("addMulticastRoute(): source or RP address has to be specified");

    // check that the incoming interface exists
    if (!entry->getInIntPtr() || entry->getInIntNextHop().isUnspecified())
        error("addMulticastRoute(): incoming interface has to be specified");

    // add to tables
    multicastRoutes.push_back(const_cast<MulticastIPRoute*>(entry));

    updateDisplayString();
    generateShowIPRoute();
}

```

6.4.3.12 bool MulticastRoutingTable::deleteRoute (const MulticastIPRoute * entry) [virtual]

DELETE ROUTE

Function check new multicast table entry and then add new entry to multicast table.

Parameters

<i>entry</i>	Multicast entry which should be deleted from multicast table.
--------------	---

Returns

False if entry was not found in table. True if entry was deleted.

See also

[MulticastIPRoute](#)
[updateDisplayString\(\)](#)
[generateShowIPRoute\(\)](#)

Definition at line 305 of file [MulticastRoutingTable.cc](#).

```

{
    Enter_Method("deleteMulticastRoute(...)");

    // find entry in routing table
    vector<MulticastIPRoute*>::iterator i;
    for (i=multicastRoutes.begin(); i!=multicastRoutes.end(); ++i)
    {
        if ((*i) == entry)
            break;
    }

    // if entry was found, it can be deleted
    if (i!=multicastRoutes.end())

```

```

{
    // first delete all timers assigned to route
    if (entry->getSrt() != NULL)
    {
        cancelEvent(entry->getSrt());
        delete entry->getSrt();
    }
    if (entry->getGrt() != NULL)
    {
        cancelEvent(entry->getGrt());
        delete entry->getGrt();
    }
    if (entry->getSat())
    {
        cancelEvent(entry->getSat());
        delete entry->getSat();
    }

    // delete timers from outgoing interfaces
    InterfaceVector outInt = entry->getOutInt();
    for (unsigned int j = 0; j < outInt.size(); j++)
    {
        if (outInt[j].pruneTimer != NULL)
        {
            cancelEvent(outInt[j].pruneTimer);
            delete outInt[j].pruneTimer;
        }
    }

    // delete route
    multicastRoutes.erase(i);
    delete entry;
    updateDisplayString();
    generateShowIPMroute();
    return true;
}
return false;
}

```

6.4.3.13 void MulticastRoutingTable::initialize (int stage) [protected, virtual]

INITIALIZE

The method initializes Multicast Routing Table module. It get access to all needed objects.

Parameters

<i>stage</i>	Stage of initialization.
--------------	--------------------------

Definition at line 38 of file [MulticastRoutingTable.cc](#).

```

{
    if (stage==0)
    {
        // get a pointer to IInterfaceTable
        ift = AnsaInterfaceTableAccess().get();

        // watch multicast table
        //WATCH_PTRVECTOR(multicastRoutes);
        WATCH_VECTOR(showMRoute);
    }
}

```

6.4.3.14 void MulticastRoutingTable::handleMessage (cMessage * msg) [protected, virtual]

HANDLE MESSAGE

Module does not have any gate, it cannot get messages.

Definition at line 84 of file [MulticastRoutingTable.cc](#).

```

{
    opp_error("This module doesn't process messages");
}

```

6.4.4 Member Data Documentation

6.4.4.1 `RouteVector MulticastRoutingTable::multicastRoutes` [protected]

Multicast routing table.

Definition at line 32 of file [MulticastRoutingTable.h](#).

6.4.4.2 `std::vector<std::string> MulticastRoutingTable::showMRoute` [protected]

Output of multicast routing table, same as Cisco mroute.

Definition at line 33 of file [MulticastRoutingTable.h](#).

6.4.4.3 `InterfaceTable* MulticastRoutingTable::ift` [protected]

Pointer to interface table.

Definition at line 34 of file [MulticastRoutingTable.h](#).

The documentation for this class was generated from the following files:

- [F:/ANSA/src/ansa/multicastRoutingTable/MulticastRoutingTable.h](#)
- [F:/ANSA/src/ansa/multicastRoutingTable/MulticastRoutingTable.cc](#)

6.5 MulticastRoutingTableAccess Class Reference

Class gives access to the [MulticastRoutingTable](#).

```
#include <MulticastRoutingTableAccess.h>
```

6.5.1 Detailed Description

Class gives access to the [MulticastRoutingTable](#).

Definition at line 18 of file [MulticastRoutingTableAccess.h](#).

The documentation for this class was generated from the following file:

- [F:/ANSA/src/ansa/multicastRoutingTable/MulticastRoutingTableAccess.h](#)

6.6 pimDM Class Reference

Class implements PIM-DM (dense mode).

```
#include <pimDM.h>
```

Protected Member Functions

- virtual int **numInitStages** () const
- virtual void **handleMessage** (cMessage *msg)
- virtual void **initialize** (int stage)

Private Member Functions

- void [receiveChangeNotification](#) (int category, const cPolymorphic *details)
- void [newMulticast](#) (MulticastIPRoute *newRoute)
- void [newMulticastAddr](#) (addRemoveAddr *members)
- void [oldMulticastAddr](#) (addRemoveAddr *members)
- void [dataOnPruned](#) (IPAddress destAddr, IPAddress srcAddr)
- void [dataOnNonRpf](#) (IPAddress group, IPAddress source, int intld)
- void [dataOnRpf](#) (MulticastIPRoute *route)
- void [rpfIntChange](#) (MulticastIPRoute *route)
- void [processPIMTimer](#) (PIMTimer *timer)
- void [processPruneTimer](#) (PIMpt *timer)
- void [processGraftRetryTimer](#) (PIMgrt *timer)
- void [processSourceActiveTimer](#) (PIMsat *timer)
- void [processStateRefreshTimer](#) (PIMsrt *timer)
- PIMpt * [createPruneTimer](#) (IPAddress source, IPAddress group, int intld, int holdTime)
- PIMgrt * [createGraftRetryTimer](#) (IPAddress source, IPAddress group)
- PIMsat * [createSourceActiveTimer](#) (IPAddress source, IPAddress group)
- PIMsrt * [createStateRefreshTimer](#) (IPAddress source, IPAddress group)
- void [processPIMPkt](#) (PIMPacket *pkt)
- void [processJoinPruneGraftPacket](#) (PIMJoinPrune *pkt, PIMPacketType type)
- void [processPrunePacket](#) (MulticastIPRoute *route, int intld, int holdTime)
- void [processGraftPacket](#) (IPAddress source, IPAddress group, IPAddress sender, int intld)
- void [processGraftAckPacket](#) (MulticastIPRoute *route)
- void [processStateRefreshPacket](#) (PIMStateRefresh *pkt)
- void [sendPimJoinPrune](#) (IPAddress nextHop, IPAddress src, IPAddress grp, int intld)
- void [sendPimGraft](#) (IPAddress nextHop, IPAddress src, IPAddress grp, int intld)
- void [sendPimGraftAck](#) (PIMGraftAck *msg)
- void [sendPimStateRefresh](#) (IPAddress originator, IPAddress src, IPAddress grp, int intld, bool P)

Private Attributes

- IRoutingTable * [rt](#)
- [MulticastRoutingTable](#) * [mrt](#)
- IInterfaceTable * [ift](#)
- NotificationBoard * [nb](#)
- [PimInterfaceTable](#) * [pimIf](#)
- [PimNeighborTable](#) * [pimNbt](#)

6.6.1 Detailed Description

Class implements PIM-DM (dense mode).

Definition at line 33 of file [pimDM.h](#).

6.6.2 Member Function Documentation

6.6.2.1 void [pimDM::receiveChangeNotification](#) (int *category*, const cPolymorphic * *details*) [private]

RECEIVE CHANGE NOTIFICATION

The method from class Notification Board is used to catch its events.

Parameters

<i>category</i>	Category of notification.
<i>details</i>	Additional information for notification.

See also

[newMulticast\(\)](#)
[newMulticastAddr\(\)](#)

Definition at line 906 of file [pimDM.cc](#).

```
{
    // ignore notifications during initialize
    if (simulation.getContextType() == CTX_INITIALIZE)
        return;

    // PIM needs addition info for each notification
    if (details == NULL)
        return;

    Enter_Method_Silent();
    printNotificationBanner(category, details);
    IPControlInfo *ctrl;
    MulticastIPRoute *route;
    addRemoveAddr *members;

    // according to category of event...
    switch (category)
    {
        // new multicast data appears in router
        case NF_IPv4_NEW_MULTICAST_DENSE:
            EV << "pimDM::receiveChangeNotification - NEW
MULTICAST DENSE-" << endl;
            route = (MulticastIPRoute *) (details);
            newMulticast(route);
            break;

        // configuration of interface changed, it means some change
        from IGMP, address were added.
        case NF_IPv4_NEW_IGMP_ADDED:
            EV << "pimDM::receiveChangeNotification - IGMP change -
address were added." << endl;
            members = (addRemoveAddr *) (details);
            newMulticastAddr(members);
            break;

        // configuration of interface changed, it means some change
        from IGMP, address were removed.
        case NF_IPv4_NEW_IGMP_REMOVED:
            EV << "pimDM::receiveChangeNotification - IGMP change -
address were removed." << endl;
            members = (addRemoveAddr *) (details);
            oldMulticastAddr(members);
            break;

        case NF_IPv4_DATA_ON_PRUNED_INT:
            EV << "pimDM::receiveChangeNotification - Data appears
on pruned interface." << endl;
            ctrl = (IPControlInfo *) (details);
            dataOnPruned(ctrl->getDestAddr(), ctrl->getSrcAddr());
            break;

        // data come to non-RPF interface
        case NF_IPv4_DATA_ON_NONRPF:
            EV << "pimDM::receiveChangeNotification - Data appears
on non-RPF interface." << endl;
            ctrl = (IPControlInfo *) (details);
            dataOnNonRpf(ctrl->getDestAddr(), ctrl->getSrcAddr(),
ctrl->getInterfaceId());
            break;

        // data come to RPF interface
        case NF_IPv4_DATA_ON_RPF:
            EV << "pimDM::receiveChangeNotification - Data appears
on RPF interface." << endl;
            route = (MulticastIPRoute *) (details);
            dataOnRpf(route);
            break;

        // RPF interface has changed
        case NF_IPv4_ROUTE_ADDED:
            EV << "pimDM::receiveChangeNotification - RPF interface
has changed." << endl;
            IPRoute *entry = (IPRoute *) (details);
            vector<MulticastIPRoute*> routes = mrt->
getRoutesForSource(entry->getHost());
            for (unsigned int i = 0; i < routes.size(); i++)
                rpfIntChange(routes[i]);
            break;
    }
}
```



```

    }
}

```

6.6.2.2 void pimDM::newMulticast (MulticastIPRoute * *newRoute*) [private]

NEW MULTICAST

The method process notification about new multicast data stream. It goes through all PIM interfaces and tests them if they can be added to the list of outgoing interfaces. If there is no interface on the list at the end, the router will prune from the multicast tree.

Parameters

<i>newRoute</i>	Pointer to new entry in the multicast routing table.
-----------------	--

See also

[sendPimGraft\(\)](#)
[createGraftRetryTimer\(\)](#)
[addRemoveAddr](#)

Definition at line 1328 of file [pimDM.cc](#).

```

{
    EV << "pimDM::newMulticast" << endl;

    // only outgoing interfaces are missing
    PimInterface *rpfInt = pimIft->getInterfaceByIntID(newRoute->getIntID());
    bool pruned = true;

    // insert all PIM interfaces except rpf int
    for (int i = 0; i < pimIft->getNumInterface(); i++)
    {
        PimInterface *pimIntTemp = pimIft->getInterface(i);
        int intId = pimIntTemp->getInterfaceID();

        //check if PIM interface is not RPF interface
        if (pimIntTemp == rpfInt)
            continue;

        // create new outgoing interface
        outInterface newOutInt;
        newOutInt.intId = pimIntTemp->getInterfaceID();
        newOutInt.intPtr = pimIntTemp->getInterfacePtr();
        newOutInt.pruneTimer = NULL;

        switch (pimIntTemp->getMode())
        {
            case Dense:
                newOutInt.mode = Densemode;
                break;
            case Sparse:
                newOutInt.mode = Sparsemode;
                break;
        }

        // if there are neighbors on interface, we will forward
        if ((pimNbt->getNeighborsByIntID(intId)).size() > 0)
        {
            newOutInt.forwarding = Forward;
            pruned = false;
            newRoute->addOutInt(newOutInt);
        }
        // if there is member of group, we will forward
        else if (pimIntTemp->isLocalIntMulticastAddress(newRoute->getGroup()))
        {
            newOutInt.forwarding = Forward;
            pruned = false;
            newRoute->addFlag(C);
            newRoute->addOutInt(newOutInt);
        }
        // in any other case interface is not involved
    }
}

```

```

// directly connected to source, set State Refresh Timer
if (newRoute->isFlagSet(A))
{
    //FIXME record TTL (I do not know why???)
    PIMsrt* timerSrt = createStateRefreshTimer(newRoute->getSource(),
newRoute->getGroup());
    newRoute->setSrt(timerSrt);
}

// set Source Active Timer (liveness of route)
PIMsrt* timerSat = createSourceActiveTimer(newRoute->getSource(),
newRoute->getGroup());
newRoute->setSat(timerSat);

// if there is no outgoing interface, prune from multicast tree
if (pruned)
{
    EV << "pimDM::newMulticast: There is no outgoing interface for
multicast, send Prune msg to upstream" << endl;
    newRoute->addFlag(P);

    if (!newRoute->isFlagSet(A))
        sendPimJoinPrune(newRoute->getInIntNextHop(), newRoute
->getSource(), newRoute->getGroup(), newRoute->getInIntId());

    // FIXME set timer which I do not use
}

// add new route record to multicast routing table
mrt->addRoute(newRoute);
EV << "PimSplitter::newMulticast: New route was added to the multicast
routing table." << endl;
}

```

6.6.2.3 void pimDM::newMulticastAddr (addRemoveAddr * members) [private]

NEW MULTICAST ADDRESS

The method process notification about new multicast groups assigned to interface. For each new address it tries to find route. If there is route, it finds interface in list of outgoing interfaces. If the interface is not in the list it will be added. if the router was pruned from multicast tree, join again.

Parameters

<i>members</i>	Structure containing new multicast IP addresses.
----------------	--

See also

[sendPimGraft\(\)](#)
[createGraftRetryTimer\(\)](#)
[addRemoveAddr](#)

Definition at line 1239 of file [pimDM.cc](#).

```

{
    EV << "pimDM::newMulticastAddr" << endl;
    vector<IPAddress> newAddr = members->getAddr();
    PimInterface * pimInt = members->getInt();
    bool forward = false;

    // go through all new multicast addresses assigned to interface
    for (unsigned int i = 0; i < newAddr.size(); i++)
    {
        EV << "New multicast address: " << newAddr[i] << endl;
        vector<MulticastIPRoute*> routes = mrt->getRouteFor(newAddr[i])
;

        // there is no route for group in the table in this moment
        if (routes.size() == 0)
            continue;

        // go through all multicast routes
        for (unsigned int j = 0; j < routes.size(); j++)
        {

```

```

MulticastIPRoute *route = routes[j];
InterfaceVector outInt = route->getOutInt();
unsigned int k;

// check on RPF interface
if (route->getInIntId() == pimInt->getInterfaceID())
    continue;

// is interface in list of outgoing interfaces?
for (k = 0; k < outInt.size(); k++)
{
    if (outInt[k].intId == pimInt->getInterfaceID())
    {
        EV << "Interface is already on list of
outgoing interfaces" << endl;
        if (outInt[k].forwarding == Pruned)
            outInt[k].forwarding = Forward;
        forward = true;
        break;
    }
}

// interface is not in list of outgoing interfaces
if (k == outInt.size())
{
    EV << "Interface is not on list of outgoing
interfaces yet, it will be added" << endl;
    outInterface newInt;
    newInt.intPtr = pimInt->getInterfacePtr();
    newInt.intId = pimInt->getInterfaceID();
    newInt.mode = Densemode;
    newInt.forwarding = Forward;
    newInt.pruneTimer = NULL;
    outInt.push_back(newInt);
    forward = true;
}
route->setOutInt(outInt);
route->addFlag(C);

// route was pruned, has to be added to multicast tree
if (route->isFlagSet(P) && forward)
{
    EV << "Route was pruned -> router has to join
to multicast tree" << endl;

    // if source is not directly connected, send
    Graft to upstream
    if (!route->isFlagSet(A))
    {
        sendPimGraft(route->getInIntNextHop(),
route->getSource(), route->getGroup(), route->getInIntId());
        PIMgrt *timer = createGraftRetryTimer(
route->getSource(), route->getGroup());
        route->setGrt(timer);
    }
    else
        route->removeFlag(P);
}
}
mrt->generateShowIPMroute();
}

```

6.6.2.4 void pimDM::oldMulticastAddr(addRemoveAddr *members) [private]

OLD MULTICAST ADDRESS

The method process notification about multicast groups removed from interface. For each old address it tries to find route. If there is route, it finds interface in list of outgoing interfaces. If the interface is in the list it will be removed. If the router was not pruned and there is no outgoing interface, the router will prune from the multicast tree.

Parameters

<i>members</i>	Structure containing old multicast IP addresses.
----------------	--

See also

[sendPimJoinPrune\(\)](#)

Definition at line 1158 of file [pimDM.cc](#).

```
{
    EV << "pimDM::oldMulticastAddr" << endl;
    vector<IPAddress> oldAddr = members->getAddr();
    PimInterface * pimInt = members->getInt();
    bool connected = false;

    // go through all old multicast addresses assigned to interface
    for (unsigned int i = 0; i < oldAddr.size(); i++)
    {
        EV << "Removed multicast address: " << oldAddr[i] << endl;
        vector<MulticastIPRoute*> routes = mrt->getRouteFor(oldAddr[i]);

        // there is no route for group in the table
        if (routes.size() == 0)
            continue;

        // go through all multicast routes
        for (unsigned int j = 0; j < routes.size(); j++)
        {
            MulticastIPRoute *route = routes[j];
            InterfaceVector outInt = route->getOutInt();
            unsigned int k;

            // is interface in list of outgoing interfaces?
            for (k = 0; k < outInt.size(); k++)
            {
                if (outInt[k].intId == pimInt->getInterfaceID()
                    {
                        EV << "Interface is present, removing
it from the list of outgoing interfaces." << endl;
                        outInt.erase(outInt.begin() + k);
                    }
                    else if (outInt[k].forwarding == Forward)
                    {
                        if ((pimNbt->getNeighborsByIntID(outInt
[k].intId)).size() == 0)
                            connected = true;
                    }
                }
                route->setOutInt(outInt);

                // if there is no directly connected member of group
                if (!connected)
                    route->removeFlag(C);

                // there is no receiver of multicast, prune the router
                from the multicast tree
                if (route->isOilistNull())
                {
                    EV << "There is no receiver for the group ->
prune from the tree" << endl;
                    // if GRT is running now, do not send Prune msg
                    if (route->isFlagSet(P) && (route->getGrt() !=
NULL))
                    {
                        cancelEvent(route->getGrt());
                        delete route->getGrt();
                        route->setGrt(NULL);
                        sendPimJoinPrune(route->getInIntNextHop
(), route->getSource(), route->getGroup(), route->getInIntId());
                    }

                    // if the source is not directly connected,
                    sent Prune msg
                    if (!route->isFlagSet(A) && !route->isFlagSet(P))
                        sendPimJoinPrune(route->getInIntNextHop
(), route->getSource(), route->getGroup(), route->getInIntId());

                    route->addFlag(P);
                }
            }
        }
        mrt->generateShowIPMroute();
    }
}
```

6.6.2.5 void pimDM::dataOnPruned (IPAddress *group*, IPAddress *source*) [private]

DATA ON PRUNED

The method has to solve the problem when multicast data appears on RPF interface and route is pruned. In this case, new PIM JoinPrune has to be sent to upstream.

Parameters

<i>group</i>	Multicast group IP address.
<i>source</i>	Source IP address.

See also

[sendPimJoinPrune\(\)](#)

Definition at line 1131 of file [pimDM.cc](#).

```
{
    EV << "pimDM::dataOnPruned" << endl;
    MulticastIPRoute *route = mrt->getRouteFor(group, source);
    // if GRT is running now, do not send Prune msg
    if (route->isFlagSet(P) && (route->getGrt() != NULL))
    {
        cancelEvent(route->getGrt());
        delete route->getGrt();
        route->setGrt(NULL);
    }
    // otherwise send Prune msg to upstream router
    else if (!route->isFlagSet(A))
        sendPimJoinPrune(route->getInIntNextHop(), source, group, route
->getInIntId());
}
```

6.6.2.6 void pimDM::dataOnNonRpf (IPAddress *group*, IPAddress *source*, int *intId*) [private]

DATA ON NON-RPF INTERFACE

The method has to solve the problem when multicast data appears on non-RPF interface. It can happen when there is loop in the network. In this case, router has to prune from the neighbor, so it sends Prune message.

Parameters

<i>group</i>	Multicast group IP address.
<i>source</i>	Source IP address.
<i>intId</i>	Identificator of incoming interface.

See also

[sendPimJoinPrune\(\)](#)
[createPruneTimer\(\)](#)

Definition at line 1077 of file [pimDM.cc](#).

```
{
    EV << "pimDM::dataOnNonRpf, intID: " << intId << endl;

    // load route from mroute
    MulticastIPRoute *route = mrt->getRouteFor(group, source);
    if (route == NULL)
        return;

    // in case of p2p link, send prune
    // FIXME There should be better indicator of P2P link
    if (pimNbt->getNumNeighborsOnInt(intId) == 1)
    {
```

```

        // send Prune msg to the neighbor who sent these multicast data
        IPAddress nextHop = (pimNbt->getNeighborsByIntID(intId))[0].
getAddr();
        sendPimJoinPrune(nextHop, source, group, intId);

        // find incoming interface
        int i = route->getOutIdByIntID(intId);
        InterfaceVector outInt = route->getOutInt();

        // the incoming interface has to change its state to Pruned
        if (outInt[i].forwarding == Forward)
        {
            outInt[i].forwarding = Pruned;
            PIMpt* timer = createPruneTimer(route->getSource(),
route->getGroup(), intId, PT);
            outInt[i].pruneTimer = timer;
            route->setOutInt(outInt);

            // if there is no outgoing interface, Prune msg has to
            be sent on upstream
            if (route->isOilistNull())
            {
                EV << "pimDM::dataOnNonRpf: oilist is NULL,
send prune msg to upstream." << endl;
                route->addFlag(P);
                if (!route->isFlagSet(A))
                    sendPimJoinPrune(route->getInIntNextHop
(), route->getSource(), route->getGroup(), route->getInIntId());
                }
                mrt->generateShowIPMroute();
            }
        }

        //FIXME in case of LAN
}

```

6.6.2.7 void pimDM::dataOnRpf (MulticastIPRoute * route) [private]

DATA ON RPF INTERFACE

The method process notification about data which appears on RPF interface. It means that source is still active. The resault is resetting of Source Active Timer.

Parameters

<i>newRoute</i>	Pointer to new entry in the multicast routing table.
-----------------	--

See also

PIMsat()

Definition at line 1058 of file [pimDM.cc](#).

```

{
    cancelEvent(route->getSat());
    scheduleAt(simTime() + SAT, route->getSat());
}

```

6.6.2.8 void pimDM::rpflntChange (MulticastIPRoute * route) [private]

RPF INTERFACE CHANGE

The method process notification about interface change. Multicast routing table will be changed if RPF interface has changed. New RPF interface is set to route and is removed from outgoing interfaces. On the other hand, old RPF interface is added to outgoing interfaces. If route was not pruned, the router has to join to the multicast tree again (by different path).

Parameters

<i>newRoute</i>	Pointer to new entry in the multicast routing table.
-----------------	--

See also

[sendPimGraft\(\)](#)
[createGraftRetryTimer\(\)](#)

Definition at line 991 of file [pimDM.cc](#).

```
{
    IPAddress source = route->getSource();
    IPAddress group = route->getGroup();
    InterfaceEntry *newRpf = rt->getInterfaceForDestAddr(source);
    int rpfId = newRpf->getInterfaceId();

    // is there any change?
    if (rpfId == route->getInIntId())
        return;
    EV << "New RPF int for group " << group << " source " << source << " is
" << rpfId << endl;

    // set new RPF
    inInterface oldRpf = route->getInInt();
    route->setInInt(newRpf, rpfId, pimNbt->getNeighborsByIntID(rpfId)[0].
getAddr());

    // route was not pruned, join to the multicast tree again
    if (!route->isFlagSet(P))
    {
        sendPimGraft(route->getInIntNextHop(), source, group, rpfId);
        PIMgrt* timer = createGraftRetryTimer(source, group);
        route->setGrt(timer);
    }

    // find rpf int in outgoing interfaces and delete it
    InterfaceVector outInt = route->getOutInt();
    for(unsigned int i = 0; i < outInt.size(); i++)
    {
        if (outInt[i].intId == rpfId)
        {
            if (outInt[i].pruneTimer != NULL)
            {
                cancelEvent(outInt[i].pruneTimer);
                delete outInt[i].pruneTimer;
                outInt[i].pruneTimer = NULL;
            }
            outInt.erase(outInt.begin() + i);
            break;
        }
    }

    // old RPF should be now outgoing interface if it is not down
    if (!oldRpf.intPtr->isDown())
    {
        outInterface newOutInt;
        newOutInt.intId = oldRpf.intId;
        newOutInt.intPtr = oldRpf.intPtr;
        newOutInt.pruneTimer = NULL;
        newOutInt.forwarding = Forward;
        newOutInt.mode = Densemode;
        outInt.push_back(newOutInt);
    }

    route->setOutInt(outInt);
    mrt->generateShowIPMroute();
}
```

6.6.2.9 void pimDM::processPIMTimer (PIMTimer * timer) [private]

PROCESS PIM TIMER

The method is used to process PIM timers. According to type of PIM timer, the timer is sent to appropriate method.

Parameters

<i>timer</i>	Pointer to PIM timer.
--------------	-----------------------

See also

[PIMTimer\(\)](#)
[processPruneTimer\(\)](#)
[processGraftRetryTimer\(\)](#)

Definition at line 735 of file [pimDM.cc](#).

```

{
    EV << "pimDM::processPIMTimer: ";
    switch(timer->getTimerKind())
    {
        case AssertTimer:
            EV << "AssertTimer" << endl;
            break;
        case PruneTimer:
            EV << "PruneTimer" << endl;
            processPruneTimer(check_and_cast<PIMpt *> (timer));
            break;
        case PrunePendingTimer:
            EV << "PrunePendingTimer" << endl;
            break;
        case GraftRetryTimer:
            EV << "GraftRetryTimer" << endl;
            processGraftRetryTimer(check_and_cast<PIMgrt *> (timer)
            );
            break;
        case UpstreamOverrideTimer:
            EV << "UpstreamOverrideTimer" << endl;
            break;
        case PruneLimitTimer:
            EV << "PruneLimitTimer" << endl;
            break;
        case SourceActiveTimer:
            EV << "SourceActiveTimer" << endl;
            processSourceActiveTimer(check_and_cast<PIMsat *> (
            timer));
            break;
        case StateRefreshTimer:
            EV << "StateRefreshTimer" << endl;
            processStateRefreshTimer(check_and_cast<PIMsrt *> (
            timer));
            break;
        default:
            EV << "BAD TYPE, DROPPED" << endl;
            delete timer;
    }
}

```

6.6.2.10 void pimDM::processPruneTimer (PIMpt * timer) [private]

PROCESS PRUNE TIMER

The method is used to process PIM Prune timer. It is (S,G,I) timer. When Prune timer expires, it means that outgoing interface transits back to forwarding state. If the router is pruned from multicast tree, join again.

Parameters

<i>timer</i>	Pointer to Prune timer.
--------------	-------------------------

See also

[PIMpt\(\)](#)
[sendPimGraft\(\)](#)
[createGraftRetryTimer\(\)](#)

Definition at line 600 of file [pimDM.cc](#).

```

{
    EV << "pimDM::processPruneTimer" << endl;

```



```

IPAddress source = timer->getSource();
IPAddress group = timer->getGroup();
int intId = timer->getIntId();

// find correct (S,G) route which timer belongs to
MulticastIPRoute *route = mrt->getRouteFor(group, source);
if (route == NULL)
{
    delete timer;
    return;
}

// state of interface is changed to forwarding
int i = route->getOutIdByIntId(intId);
InterfaceVector outInt = route->getOutInt();
if (i < (int) outInt.size())
{
    EV << "Nalezen out int" << endl;
    delete timer;
    outInt[i].pruneTimer = NULL;
    outInt[i].forwarding = Forward;
    route->setOutInt(outInt);

    // if the router is pruned from multicast tree, join again
    if (route->isFlagSet(P) && (route->getGrt() == NULL))
    {
        if (!route->isFlagSet(A))
        {
            EV << "Pruned cesta prejde do forwardu, posli
Graft" << endl;
            sendPimGraft(route->getInIntNextHop(), source,
group, route->getInIntId());
            PIMgrt* timer = createGraftRetryTimer(source,
group);
            route->setGrt(timer);
        }
        else
            route->removeFlag(P);
    }
    mrt->generateShowIPRoute();
}
}

```

6.6.2.11 void pimDM::processGraftRetryTimer (PIMgrt * timer) [private]

PROCESS GRAFT RETRY TIMER

The method is used to process PIM Graft Retry Timer. It is (S,G) timer. When Graft Retry timer expires, it means that the router didn't get GraftAck message from upstream router. The router has to send Prune packet again.

Parameters

<i>timer</i>	Pointer to Graft Retry timer.
--------------	-------------------------------

See also

[PIMgrt\(\)](#)
[sendPimGraft\(\)](#)
[createGraftRetryTimer\(\)](#)

Definition at line 656 of file [pimDM.cc](#).

```

{
    EV << "pimDM::processGraftRetryTimer" << endl;
    MulticastIPRoute *route = mrt->getRouteFor(timer->getGroup(), timer->
getSource());
    sendPimGraft(route->getInIntNextHop(), timer->getSource(), timer->
getGroup(), route->getInIntId());
    timer = createGraftRetryTimer(timer->getSource(), timer->getGroup());
}

```

6.6.2.12 void pimDM::processSourceActiveTimer (PIMsat * timer) [private]

PROCESS GRAFT RETRY TIMER

The method is used to process PIM Source Active Timer. It is (S,G) timer. When Source Active Timer expires, route is removed from multicast routing table.

Parameters

<i>timer</i>	Pointer to Source Active Timer.
--------------	---------------------------------

See also

PIMsat()

Definition at line 673 of file [pimDM.cc](#).

```
{
    EV << "pimDM::processSourceActiveTimer: route will be deleted" << endl;
    MulticastIPRoute *route = mrt->getRouteFor(timer->getGroup(), timer->
    getSource());

    delete timer;
    route->setSat(NULL);
    mrt->deleteRoute(route);
}
```

6.6.2.13 void pimDM::processStateRefreshTimer (PIMsrt * timer) [private]

PROCESS STATE REFRESH TIMER

The method is used to process PIM State Refresh Timer. It is (S,G) timer and it is used only on router which is connected directly to the source of multicast. When State Refresh Timer expires, the State Refresh messages are sent to all outgoing interfaces. Prune indicator in each msg is set according to state of the outgoing interface. State Refresh Timer is then reset.

Parameters

<i>timer</i>	Pointer to Source Active Timer.
--------------	---------------------------------

See also

PIMsrt()
[sendPimStateRefresh\(\)](#)
[createStateRefreshTimer\(\)](#)

Definition at line 696 of file [pimDM.cc](#).

```
{
    EV << "pimDM::processStateRefreshTimer" << endl;
    MulticastIPRoute *route = mrt->getRouteFor(timer->getGroup(), timer->
    getSource());
    InterfaceVector outInt = route->getOutInt();
    bool pruneIndicator;

    for (unsigned int i = 0; i < outInt.size(); i++)
    {
        if (outInt[i].forwarding == Pruned)
        {
            // P = true
            pruneIndicator = true;
            // reset PT
            cancelEvent(outInt[i].pruneTimer);
            scheduleAt(simTime() + PT, outInt[i].pruneTimer);
        }
        else if (outInt[i].forwarding == Forward)
    }
```

```

        {
            pruneIndicator = false;
        }
        int intId = outInt[i].intId;
        sendPimStateRefresh(ift->getInterfaceById(intId)->ipv4Data()->
getIPAddress(), timer->getSource(), timer->getGroup(), intId, pruneIndicator);
    }
    delete timer;
    route->setSrt(createStateRefreshTimer(route->getSource(), route->
getGroup()));
}

```

6.6.2.14 PIMpt * pimDM::createPruneTimer (IPAddress source, IPAddress group, int intId, int holdTime) [private]

CREATE PRUNE TIMER

The method is used to create PIMPrune timer. The timer is set when outgoing interface goes to pruned state. After expiration (usually 3min) interface goes back to forwarding state. It is set to (S,G,I).

Parameters

<i>source</i>	IP address of multicast source.
<i>group</i>	IP address of multicast group.
<i>intId</i>	ID of outgoing interface.
<i>holdTime</i>	Time of expiration (usually 3min).

Returns

Pointer to new Prune Timer.

See also

PIMpt()

Definition at line 182 of file [pimDM.cc](#).

```

{
    EV << "pimDM::createPruneTimer" << endl;
    PIMpt *timer = new PIMpt();
    timer->setName("PimPruneTimer");
    timer->setSource(source);
    timer->setGroup(group);
    timer->setIntId(intId);
    scheduleAt(simTime() + holdTime, timer);
    return timer;
}

```

6.6.2.15 PIMgrt * pimDM::createGraftRetryTimer (IPAddress source, IPAddress group) [private]

CREATE GRAFT RETRY TIMER

The method is used to create PIMGraftRetry timer. The timer is set when router wants to join to multicast tree again and send PIM Prune message to upstream. Router waits for Graft Retry Timer (3 s) for PIM PruneAck message from upstream. If timer expires, router will send PIM Prune message again. It is set to (S,G).

Parameters

<i>source</i>	IP address of multicast source.
<i>group</i>	IP address of multicast group.

Returns

Pointer to new Graft Retry Timer.

See also

PIMgrt()

Definition at line 207 of file [pimDM.cc](#).

```
{
    EV << "pimDM::createPruneTimer" << endl;
    PIMgrt *timer = new PIMgrt();
    timer->setName("PIMGraftRetryTimer");
    timer->setSource(source);
    timer->setGroup(group);
    scheduleAt(simTime() + GRT, timer);
    return timer;
}
```

6.6.2.16 PIMsat * pimDM::createSourceActiveTimer (IPAddress *source*, IPAddress *group*) [private]

CREATE SOURCE ACTIVE TIMER

The method is used to create PIMSourceActive timer. The timer is set when source of multicast is connected directly to the router. If timer expires, router will remove the route from multicast routing table. It is set to (S,G).

Parameters

<i>source</i>	IP address of multicast source.
<i>group</i>	IP address of multicast group.

Returns

Pointer to new Source Active Timer

See also

PIMsat()

Definition at line 230 of file [pimDM.cc](#).

```
{
    EV << "pimDM::createSourceActiveTimer" << endl;
    PIMsat *timer = new PIMsat();
    timer->setName("PIMSourceActiveTimer");
    timer->setSource(source);
    timer->setGroup(group);
    scheduleAt(simTime() + SAT, timer);
    return timer;
}
```

6.6.2.17 PIMsrt * pimDM::createStateRefreshTimer (IPAddress *source*, IPAddress *group*) [private]

CREATE STATE REFRESH TIMER

The method is used to create PIMStateRefresh timer. The timer is set when source of multicast is connected directly to the router. If timer expires, router will send StateRefresh message, which will propagate through all network and will reset Prune Timer. It is set to (S,G).

Parameters

<i>source</i>	IP address of multicast source.
<i>group</i>	IP address of multicast group.

Returns

Pointer to new Source Active Timer

See also

PIMsrt()

Definition at line 253 of file [pimDM.cc](#).

```
{
    EV << "pimDM::createStateRefreshTimer" << endl;
    PIMsrt *timer = new PIMsrt();
    timer->setName("PIMStateRefreshTimer");
    timer->setSource(source);
    timer->setGroup(group);
    scheduleAt(simTime() + SRT, timer);
    return timer;
}
```

6.6.2.18 void pimDM::processPIMPkt (PIMPacket * pkt) [private]**PROCESS PIM PACKET**

The method is used to process PIM packets. According to type of PIM packet, the packet is sent to appropriate method.

Parameters

<i>pkt</i>	Pointer to PIM packet.
------------	------------------------

See also

PIMPacket()
[processJoinPruneGraftPacket\(\)](#)

Definition at line 785 of file [pimDM.cc](#).

```
{
    EV << "pimDM::processPIMPkt: ";

    switch(pkt->getType())
    {
        case JoinPrune:
            EV << "JoinPrune" << endl;
            processJoinPruneGraftPacket(check_and_cast<PIMJoinPrune
*> (pkt), (PIMPacketType) pkt->getType());
            break;
        case Assert:
            EV << "Assert" << endl;
            // FIXME for future use
            break;
        case Graft:
            EV << "Graft" << endl;
            processJoinPruneGraftPacket(check_and_cast<PIMJoinPrune
*> (pkt), (PIMPacketType) pkt->getType());
            break;
        case GraftAck:
            EV << "GraftAck" << endl;
            processJoinPruneGraftPacket(check_and_cast<PIMJoinPrune
*> (pkt), (PIMPacketType) pkt->getType());
            break;
        case StateRefresh:
            EV << "StateRefresh" << endl;
    }
```

```

        processStateRefreshPacket (
check_and_cast<PIMStateRefresh *> (pkt));
        break;
    default:
        EV << "BAD TYPE, DROPPED" << endl;
        delete pkt;
    }
}

```

6.6.2.19 void pimDM::processJoinPruneGraftPacket (PIMJoinPrune * pkt, PIMPacketType type) [private]

PROCESS JOIN PRUNE GRAFT PACKET

The method is used to process PIM JoinPrune, PIMGraft, and PIMGraftAck packet. All these packets have the same structure. If packet is for this router the method goes through all multicast groups in message. There could be list of joined and pruned sources for each multicast group. Join Prune message can contains both. Graft and Graft Ack can contain only join list. According to type of packet, appropriate method is called to process info.

Parameters

<i>pkt</i>	Pointer to packet.
<i>type</i>	Type of packet.

See also

[processGraftPacket\(\)](#)
[processGraftAckPacket\(\)](#)
[processPrunePacket\(\)](#)
[sendPimGraftAck\(\)](#)

Definition at line 438 of file [pimDM.cc](#).

```

{
    EV << "pimDM::processJoinPruneGraftPacket" << endl;

    IPControlInfo *ctrl = (IPControlInfo *) pkt->getControlInfo();
    IPAddress sender = ctrl->getSrcAddr();
    InterfaceEntry * nt = rt->getInterfaceForDestAddr(sender);
    vector<PimNeighbor> neighbors = pimNbt->getNeighborsByIntID(nt->
getInterfaceId());
    IPAddress addr = nt->ipv4Data()->getIPAddress();

    // does packet belong to this router?
    if (pkt->getUpstreamNeighborAddress() != nt->ipv4Data()->getIPAddress()
&& type != GraftAck)
    {
        EV << "Paket neni urceny pro tento router" << endl;
        delete pkt;
        return;
    }

    // go through list of multicast groups
    for (unsigned int i = 0; i < pkt->getMulticastGroupsArraySize(); i++)
    {
        MulticastGroup group = pkt->getMulticastGroups(i);
        IPAddress groupAddr = group.getGroupAddress();

        // go through list of joined sources
        //EV << "JoinedSourceAddressArraySize: " <<
group.getJoinedSourceAddressArraySize() << endl;
        for (unsigned int j = 0; j < group.
getJoinedSourceAddressArraySize(); j++)
        {
            IPAddress source = group.getJoinedSourceAddress(j);
            MulticastIPRoute *route = mrt->getRouteFor (groupAddr,
source);

            if (type == JoinPrune)
            {
                //FIXME join action
                // only if there is more than one PIM neighbor
                on one interface
                // interface change to forwarding state
            }
        }
    }
}

```

```

        // cancel Prune Timer
        // send Graft to upstream
    }
    else if (type == Graft)
        processGraftPacket(source, groupAddr, sender,
nt->getInterfaceId());
    else if (type == GraftAck)
        processGraftAckPacket(route);
    }

    // go through list of pruned sources (only for PIM Join Prune
msg)
    if (type == JoinPrune)
    {
        //EV << "JoinedPrunedAddressArraySize: " <<
group.getPrunedSourceAddressArraySize() << endl;
        for(unsigned int k = 0; k < group.
getPrunedSourceAddressArraySize(); k++)
        {
            IPAddress source = group.getPrunedSourceAddress
(k);
            MulticastIPRoute *route = mrt->getRouteFor(
groupAddr, source);

            if (source != route->getSource())
                continue;

            // if there could be more than one PIM neighbor
on interface
            if (neighbors.size() > 1)
            {
                EV << "Vice sousedu na rozhrani" <<
endl;
                ; //FIXME set PPT timer
            }
            // if there is only one PIM neighbor on
interface
            else
                processPrunePacket(route, nt->
getInterfaceId(), pkt->getHoldTime());
        }
    }

    // Send GraftAck for this Graft message
    if (type == Graft)
        sendPimGraftAck((PIMGraftAck *) (pkt));
    mrt->generateShowIPMroute();
}

```

6.6.2.20 void pimDM::processPrunePacket (MulticastIPRoute * route, int intld, int holdTime) [private]

PROCESS PRUNE PACKET

The method process PIM Prune packet. First the method has to find correct outgoing interface where PIM Prune packet came to. The method also checks if there is still any forwarding outgoing interface. Forwarding interfaces, where Prune packet come to, goes to prune state. If all outgoing interfaces are pruned, the router will prune from multicast tree.

Parameters

<i>route</i>	Pointer to multicast route which GraftAck belongs to.
<i>intld</i>	

See also

[processJoinPruneGraftPacket\(\)](#)
[createPruneTimer\(\)](#)
[sendPimJoinPrune\(\)](#)
[PIMpt\(\)](#)

Definition at line 373 of file [pimDM.cc](#).

```
{
```

```

EV << "pimDM::processPrunePacket" << endl;
InterfaceVector outInt = route->getOutInt();
int i = route->getOutIdByIntId(intId);
bool change = false;

// we find correct outgoing interface
if (i < (int) outInt.size())
{
    // if interface was already pruned, restart Prune Timer
    if (outInt[i].forwarding == Pruned)
    {
        EV << "Outgoing interface is already pruned, restart
Prune Timer." << endl;
        PIMpt* timer = outInt[i].pruneTimer;
        cancelEvent(timer);
        scheduleAt(simTime() + holdTime, timer);
    }
    // if interface is forwarding, transit its state to pruned and
    set Prune timer
    else
    {
        EV << "Outgoing interfaces is forwarding now -> change
to Pruned, set the timer." << endl;
        outInt[i].forwarding = Pruned;
        PIMpt* timer = createPruneTimer(route->getSource(),
route->getGroup(), intId, holdTime);
        outInt[i].pruneTimer = timer;
        change = true;
    }
}
route->setOutInt(outInt);

// if there is no forwarding outgoing int, transit route to pruned
state
if (route->isOilistNull() && change)
{
    EV << "All interfaces are pruned, send Pruned to upstream." <<
endl;
    route->addFlag(P);

    // if GRT is running now, do not send Prune msg
    if (route->isFlagSet(P) && (route->getGrt() != NULL))
    {
        cancelEvent(route->getGrt());
        delete route->getGrt();
        route->setGrt(NULL);
    }
    else if (!route->isFlagSet(A))
        sendPimJoinPrune(route->getInIntNextHop(), route->
getSource(), route->getGroup(), route->getInIntId());
}
}

```

6.6.2.21 void pimDM::processGraftPacket (IPAddress source, IPAddress group, IPAddress sender, int intId)
[private]

PROCESS GRAFT PACKET

The method is used to process PIMGraft packet. Packet means that downstream router wants to join to multicast tree, so the packet cannot come to RPF interface. Router finds correct outgoing interface towards downstream router. Change its state to forward if it was not before and cancel Prune Timer. If route was in pruned state, router will send also Graft message to join multicast tree.

Parameters

<i>source</i>	IP address of multicast source.
<i>group</i>	IP address of multicast group.
<i>sender</i>	IP Address of Graft packet sender.
<i>intId</i>	ID of Graft packet incoming interface.

See also

[sendPimGraft\(\)](#)
[createGraftRetryTimer\(\)](#)
[PIMGraft\(\)](#)

PIMgrt()
processJoinPruneGraftPacket()

Definition at line 283 of file [pimDM.cc](#).

```
{
    EV << "pimDM::processGraftPacket" << endl;

    MulticastIPRoute *route = mrt->getRouteFor(group, source);
    bool forward = false;

    // check if message come to non-RPF interface
    if (route->isRpf(intId))
    {
        EV << "ERROR: Graft message came to RPF interface." << endl;
        return;
    }

    // find outgoing interface to neighbor
    InterfaceVector outInt = route->getOutInt();
    for (unsigned int l = 0; l < outInt.size(); l++)
    {
        if(outInt[l].intId == intId)
        {
            forward = true;
            if (outInt[l].forwarding == Pruned)
            {
                EV << "Interface " << outInt[l].intId << "
transit to forwarding state (Graft)." << endl;
                outInt[l].forwarding = Forward;

                //cancel Prune Timer
                PIMpt* timer = outInt[l].pruneTimer;
                cancelEvent(timer);
                delete timer;
                outInt[l].pruneTimer = NULL;
            }
        }
    }
    route->setOutInt(outInt);

    // if all route was pruned, remove prune flag
    // if upstream is not source, send Graft message
    if (route->isFlagSet(P) && forward && (route->getGrt() == NULL))
    {
        if (!route->isFlagSet(A))
        {
            EV << "Route is not pruned any more, send Graft to
upstream" << endl;
            sendPimGraft(route->getInIntNextHop(), source, group,
route->getInIntId());
            PIMgrt* timer = createGraftRetryTimer(source, group);
            route->setGrt(timer);
        }
        else
            route->removeFlag(P);
    }
}
```

6.6.2.22 void pimDM::processGraftAckPacket (MulticastIPRoute * route) [private]

PROCESS GRAFT ACK PACKET

The method is used to process PIMGraftAck packet. Packet means that the router was successfully joined to multicast tree. Route is now in forwarding state and Graft Retry timer was canceled.

Parameters

<i>route</i>	Pointer to multicast route which GraftAck belongs to.
--------------	---

See also

[processJoinPruneGraftPacket\(\)](#)
[PIMgrt\(\)](#)

Definition at line 345 of file [pimDM.cc](#).

```
{
    EV << "pimDM::processGraftAckPacket" << endl;
    PIMgrt *grt = route->getGrt();
    if (grt != NULL)
    {
        cancelEvent(grt);
        delete grt;
        route->setGrt(NULL);
        route->removeFlag(P);
    }
}
```

6.6.2.23 void pimDM::processStateRefreshPacket (PIMStateRefresh * pkt) [private]

PROCESS STATE REFRESH PACKET

The method is used to process PIMStateRefresh packet. The method checks if there is route in mroute and that packet has came to RPF interface. Then it goes through all outgoing interfaces. If the interface is pruned, it resets Prune Timer. For each interface State Refresh message is copied and correct prune indicator is set according to state of outgoing interface (pruned/forwarding).

State Refresh message is used to stop flooding of network each 3 minutes.

Parameters

<i>pkt</i>	Pointer to packet.
------------	--------------------

See also

[sendPimJoinPrune\(\)](#)
[sendPimStateRefresh\(\)](#)

Definition at line 528 of file [pimDM.cc](#).

```
{
    EV << "pimDM::processStateRefreshPacket" << endl;

    // FIXME actions of upstream automat according to pruned/forwarding
    state and Prune Indicator from msg

    // first check if there is route for given group address and source
    MulticastIPRoute *route = mrt->getRouteFor(pkt->getGroupAddress(), pkt
->getSourceAddress());
    if (route == NULL)
    {
        delete pkt;
        return;
    }
    InterfaceVector outInt = route->getOutInt();
    bool pruneIndicator;

    // chcheck if State Refresh msg has came to RPF interface
    IPControlInfo *ctrl = (IPControlInfo*) pkt->getControlInfo();
    if (ctrl->getInterfaceId() != route->getInIntId())
    {
        delete pkt;
        return;
    }

    // this router is pruned, but outgoing int of upstream router leading
    to this router is forwarding
    if (route->isFlagSet(P) && !pkt->getP())
    {
        // send Prune msg to upstream
        if (route->getGrt() == NULL)
```

```

        sendPimJoinPrune(route->getInIntNextHop(), route->
getSource(), route->getGroup(), route->getInIntId());
    else
    {
        cancelEvent(route->getGrT());
        delete route->getGrT();
        route->setGrT(NULL);
    }
}

// go through all outgoing interfaces, reser Prune Timer and send out
State Refresh msg
for (unsigned int i = 0; i < outInt.size(); i++)
{
    if (outInt[i].forwarding == Pruned)
    {
        // P = true
        pruneIndicator = true;
        // reset PT
        cancelEvent(outInt[i].pruneTimer);
        scheduleAt(simTime() + PT, outInt[i].pruneTimer);
    }
    else if (outInt[i].forwarding == Forward)
    {
        // P = false
        pruneIndicator = false;
    }
    sendPimStateRefresh(pkt->getOriginatorAddress(), pkt->
getSourceAddress(), pkt->getGroupAddress(), outInt[i].intId, pruneIndicator);
}
delete pkt;
}

```

6.6.2.24 void pimDM::sendPimJoinPrune (IPAddress nextHop, IPAddress src, IPAddress grp, int intId)
[private]

SEND PIM JOIN PRUNE

The method is used to create PIMJoinPrune Packet and send it to next hop router.

Parameters

<i>nextHop</i>	IP Address of receiver.
<i>src</i>	IP address of multicast source.
<i>grp</i>	IP address of multicast group.
<i>intId</i>	ID of outgoing interface.

See also

PIMJoinPrune()

Definition at line 27 of file [pimDM.cc](#).

```

{
    EV << "pimDM::sendPimJoinPrune" << endl;
    EV << "UpstreamNeighborAddress: " << nextHop << ", Source: " << src <<
", Group: " << grp << ", IntId: " << intId << endl;

    PIMJoinPrune *msg = new PIMJoinPrune();
    msg->setName("PIMJoinPrune");
    msg->setUpstreamNeighborAddress(nextHop);
    msg->setHoldTime(PT);
    msg->setMulticastGroupsArraySize(1);

    //FIXME change to add also join groups
    // we do not need it at this time

    // set multicast groups
    MulticastGroup *group = new MulticastGroup();
    group->setGroupAddress(grp);
    group->setJoinedSourceAddressArraySize(0);
    group->setPrunedSourceAddressArraySize(1);
    group->setPrunedSourceAddress(0, src);
    msg->setMulticastGroups(0, *group);
}

```

```

// set IP Control info
IPControlInfo *ctrl = new IPControlInfo();
IPAddress gal("224.0.0.13");
ctrl->setDestAddr(gal);
//ctrl->setProtocol(IP_PROT_PIM);
ctrl->setProtocol(103);
ctrl->setTimeToLive(1);
ctrl->setInterfaceId(intId);
msg->setControlInfo(ctrl);
send(msg, "spiltterOut");
}

```

6.6.2.25 `void pimDM::sendPimGraft (IPAddress nextHop, IPAddress src, IPAddress grp, int intId)` [private]

SEND PIM GRAFT

The method is used to create PIMGraft packet and send it to next hop router. Only JoinedSource part of message is used, because Graft pkt is sent when router wants to join to multicast tree again.

Parameters

<i>nextHop</i>	IP Address of receiver.
<i>src</i>	IP address of multicast source.
<i>grp</i>	IP address of multicast group.
<i>intId</i>	ID of outgoing interface.

See also

PIMGraft()

Definition at line 102 of file [pimDM.cc](#).

```

{
    EV << "pimDM::sendPimGraft" << endl;
    EV << "UpstreamNeighborAddress: " << nextHop << ", Source: " << src <<
    ", Group: " << grp << ", IntId: " << intId << endl;

    PIMGraft *msg = new PIMGraft();
    msg->setName("PIMGraft");
    msg->setHoldTime(0);
    msg->setUpstreamNeighborAddress(nextHop);
    msg->setMulticastGroupsArraySize(1);

    // set multicast groups
    MulticastGroup *group = new MulticastGroup();
    group->setGroupAddress(grp);
    group->setJoinedSourceAddressArraySize(1);
    group->setPrunedSourceAddressArraySize(0);
    group->setJoinedSourceAddress(0, src);
    msg->setMulticastGroups(0, *group);

    // set IP Control info
    IPControlInfo *ctrl = new IPControlInfo();
    ctrl->setDestAddr(nextHop);
    //ctrl->setProtocol(IP_PROT_PIM);
    ctrl->setProtocol(103);
    ctrl->setTimeToLive(1);
    ctrl->setInterfaceId(intId);
    msg->setControlInfo(ctrl);
    send(msg, "spiltterOut");
}

```

6.6.2.26 `void pimDM::sendPimGraftAck (PIMGraftAck * msg)` [private]

SEND PIM GRAFT ACK

The method is used to create PIMGraftAck packet and send it to next hop router. PIMGraftAck pkt is copy of received PIMGraft pkt. Only type and IP control info has to be changed.

Parameters

<i>msg</i>	Pointer to PIMGraft packet.
------------	-----------------------------

See also

PIMGraftAck()

Definition at line 71 of file [pimDM.cc](#).

```
{
    msg->setName("PIMGraftAck");
    msg->setType(GraftAck);

    // set IP Control info
    IPControlInfo *oldCtrl = (IPControlInfo*) (msg->removeControlInfo());
    IPControlInfo *ctrl = new IPControlInfo();
    ctrl->setDestAddr(oldCtrl->getSrcAddr());
    ctrl->setSrcAddr(oldCtrl->getDestAddr());
    ctrl->setProtocol(103);
    ctrl->setTimeToLive(1);
    ctrl->setInterfaceId(oldCtrl->getInterfaceId());
    delete oldCtrl;
    msg->setControlInfo(ctrl);
    send(msg, "spiltterOut");
}
```

6.6.2.27 void pimDM::sendPimStateRefresh (IPAddress *originator*, IPAddress *src*, IPAddress *grp*, int *intId*, bool *P*)
[private]

SEND PIM STATE REFRESH

The method is used to create PIMStateRefresh packet and send it to next hop router. By using the message we do not need to flood the network every 3 minutes.

Parameters

<i>originator</i>	IP Address of source router.
<i>src</i>	IP address of multicast source.
<i>grp</i>	IP address of multicast group.
<i>intId</i>	ID of outgoing interface.
<i>P</i>	Indicator of pruned outgoing interface. If interface is pruned it is set to 1, otherwise to 0.

See also

PIMStateRefresh()

Definition at line 145 of file [pimDM.cc](#).

```
{
    EV << "pimDM::sendPimStateRefresh" << endl;

    PIMStateRefresh *msg = new PIMStateRefresh();
    msg->setName("PIMStateRefresh");
    msg->setGroupAddress(grp);
    msg->setSourceAddress(src);
    msg->setOriginatorAddress(originator);
    msg->setInterval(SRT);
    msg->setP(P);

    // set IP Control info
    IPControlInfo *ctrl = new IPControlInfo();
    ctrl->setDestAddr(grp);
    //ctrl->setProtocol(IP_PROT_PIM);
    ctrl->setProtocol(103);
    ctrl->setTimeToLive(1);
    ctrl->setInterfaceId(intId);
    msg->setControlInfo(ctrl);
    send(msg, "spiltterOut");
}
```

6.6.2.28 void pimDM::handleMessage (cMessage * msg) [protected, virtual]

HANDLE MESSAGE

The method is used to handle new messages. Self messages are timer and they are sent to method which processes PIM timers. Other messages should be PIM packets, so they are sent to method which processes PIM packets.

Parameters

<i>msg</i>	Pointer to new message.
------------	-------------------------

See also

PIMPacket()
PIMTimer()
[processPIMTimer\(\)](#)
[processPIMPkt\(\)](#)

Definition at line 831 of file [pimDM.cc](#).

```
{
    EV << "PIMDM::handleMessage" << endl;

    // self message (timer)
    if (msg->isSelfMessage())
    {
        EV << "PIMDM::handleMessage:Timer" << endl;
        PIMTimer *timer = check_and_cast<PIMTimer*>(msg);
        processPIMTimer(timer);
    }
    // PIM packet from PIM neighbor
    else if (dynamic_cast<PIMPacket*>(msg))
    {
        EV << "PIMDM::handleMessage: PIM-DM packet" << endl;
        PIMPacket *pkt = check_and_cast<PIMPacket*>(msg);
        processPIMPkt(pkt);
    }
    // wrong message, mistake
    else
        EV << "PIMDM::handleMessage: Wrong message" << endl;
}
```

6.6.2.29 void pimDM::initialize (int stage) [protected, virtual]

INITIALIZE

The method initializes PIM-DM module. It get access to all needed tables and other objects. It subscribes to important notifications. If there is no PIM interface, all module can be disabled.

Parameters

<i>stage</i>	Stage of initialization.
--------------	--------------------------

Definition at line 863 of file [pimDM.cc](#).

```
{
    if (stage == 4)
    {
        EV << "pimDM::initialize" << endl;

        // Pointer to routing tables, interface tables, notification
        board
        rt = RoutingTableAccess().get();
        mrt = MulticastRoutingTableAccess().get();
        ift = AnsaInterfaceTableAccess().get();
        nb = NotificationBoardAccess().get();
        pimIft = PimInterfaceTableAccess().get();
        pimNbt = PimNeighborTableAccess().get();
    }
```

```

// is PIM enabled?
if (pimIft->getNumInterface() == 0)
{
    EV << "PIM is NOT enabled on device " << endl;
    return;
}

// subscribe for notifications
nb->subscribe(this, NF_IPv4_NEW_MULTICAST_DENSE);
nb->subscribe(this, NF_IPv4_NEW_IGMP_ADDED);
nb->subscribe(this, NF_IPv4_NEW_IGMP_REMOVED);
nb->subscribe(this, NF_IPv4_DATA_ON_PRUNED_INT);
nb->subscribe(this, NF_IPv4_DATA_ON_NONRPF);
nb->subscribe(this, NF_IPv4_DATA_ON_RPF);
//nb->subscribe(this, NF_IPv4_RPF_CHANGE);
nb->subscribe(this, NF_IPv4_ROUTE_ADDED);
}
}

```

6.6.3 Member Data Documentation

6.6.3.1 IRoutingTable* pimDM::rt [private]

Pointer to routing table.

Definition at line 36 of file [pimDM.h](#).

6.6.3.2 MulticastRoutingTable* pimDM::mrt [private]

Pointer to multicast routing table.

Definition at line 37 of file [pimDM.h](#).

6.6.3.3 IInterfaceTable* pimDM::ift [private]

Pointer to interface table.

Definition at line 38 of file [pimDM.h](#).

6.6.3.4 NotificationBoard* pimDM::nb [private]

Pointer to notification table.

Definition at line 39 of file [pimDM.h](#).

6.6.3.5 PimInterfaceTable* pimDM::pimIft [private]

Pointer to table of PIM interfaces.

Definition at line 40 of file [pimDM.h](#).

6.6.3.6 PimNeighborTable* pimDM::pimNbt [private]

Pointer to table of PIM neighbors.

Definition at line 41 of file [pimDM.h](#).

The documentation for this class was generated from the following files:

- [F:/ANSA/src/ansa/pim/modes/pimDM.h](#)
- [F:/ANSA/src/ansa/pim/modes/pimDM.cc](#)

6.7 PimInterface Class Reference

Class represents one entry of [PimInterfaceTable](#).

```
#include <PimInterfaceTable.h>
```

Public Member Functions

- virtual std::string [info](#) () const
- void [setInterfaceID](#) (int iftID)
- void [setInterfacePtr](#) (InterfaceEntry *intPtr)
- void [setMode](#) (PIMmode mode)
- int [getInterfaceID](#) () const
- InterfaceEntry * [getInterfacePtr](#) () const
- PIMmode [getMode](#) () const
- std::vector< IPAddress > [getIntMulticastAddresses](#) () const
- void [setIntMulticastAddresses](#) (std::vector< IPAddress > [intMulticastAddresses](#))
- void [addIntMulticastAddress](#) (IPAddress addr)
- void [removeIntMulticastAddress](#) (IPAddress addr)
- bool [isLocalIntMulticastAddress](#) (IPAddress addr)
- std::vector< IPAddress > [deleteLocalIPs](#) (std::vector< IPAddress > multicastAddr)

Protected Attributes

- int [intID](#)
- InterfaceEntry * [intPtr](#)
- PIMmode [mode](#)
- std::vector< IPAddress > [intMulticastAddresses](#)

6.7.1 Detailed Description

Class represents one entry of [PimInterfaceTable](#).

One entry contains interfaces ID, pointer to the interface, PIM mode and multicast addresses assigned to the interface.

Definition at line 31 of file [PimInterfaceTable.h](#).

6.7.2 Member Function Documentation

6.7.2.1 std::string PimInterface::info () const [virtual]

Actually not in use

Definition at line 50 of file [PimInterfaceTable.cc](#).

```
{
    std::stringstream out;
    out << "ID = " << intID << "; mode = " << mode;
    return out.str();
}
```

6.7.2.2 void PimInterface::setInterfaceID (int iftID) [inline]

Set identifier of interface.

Definition at line 45 of file [PimInterfaceTable.h](#).

6.7.2.3 `void PimInterface::setInterfacePtr (InterfaceEntry * intPtr)` `[inline]`

Set pointer to interface.

Definition at line 46 of file [PimInterfaceTable.h](#).

6.7.2.4 `void PimInterface::setMode (PIMmode mode)` `[inline]`

Set PIM mode configured on the interface.

Definition at line 47 of file [PimInterfaceTable.h](#).

6.7.2.5 `int PimInterface::getInterfaceID () const` `[inline]`

Get identifier of interface.

Definition at line 50 of file [PimInterfaceTable.h](#).

6.7.2.6 `InterfaceEntry* PimInterface::getInterfacePtr () const` `[inline]`

Get pointer to interface.

Definition at line 51 of file [PimInterfaceTable.h](#).

6.7.2.7 `PIMmode PimInterface::getMode () const` `[inline]`

Get PIM mode configured on the interface.

Definition at line 52 of file [PimInterfaceTable.h](#).

6.7.2.8 `std::vector<IPAddress> PimInterface::getIntMulticastAddresses () const` `[inline]`

Get list of multicast addresses assigned to the interface.

Definition at line 53 of file [PimInterfaceTable.h](#).

6.7.2.9 `void PimInterface::setIntMulticastAddresses (std::vector< IPAddress > intMulticastAddresses)`
`[inline]`

Set multicast addresses to the interface.

Definition at line 56 of file [PimInterfaceTable.h](#).

6.7.2.10 `void PimInterface::addIntMulticastAddress (IPAddress addr)` `[inline]`

Add multicast address to the interface.

Definition at line 57 of file [PimInterfaceTable.h](#).

6.7.2.11 `void PimInterface::removeIntMulticastAddress (IPAddress addr)`

REMOVE INTERFACE MULTICAST ADDRESS

The method removes given address from vector of multicast addresses.

Parameters

<i>addr</i>	IP address which should be deleted.
-------------	-------------------------------------

Definition at line 64 of file [PimInterfaceTable.cc](#).

```
{
    for(unsigned int i = 0; i < intMulticastAddresses.size(); i++)
    {
        if (intMulticastAddresses[i] == addr)
        {
            intMulticastAddresses.erase(intMulticastAddresses.begin
            () + i);
            return;
        }
    }
}
```

6.7.2.12 bool PimInterface::isLocalIntMulticastAddress (IPAddress *addr*)

IS LOCAL INETRFACE MULTICAST ADDRESS

The method finds out if IP address is assigned to interface as local multicast address.

Parameters

<i>addr</i>	Multicast IP address which we are looking for.
-------------	--

Returns

True if method finds the IP address on the list, return false otherwise.

Definition at line 116 of file [PimInterfaceTable.cc](#).

```
{
    for(unsigned int i = 0; i < intMulticastAddresses.size(); i++)
    {
        if (intMulticastAddresses[i] == addr)
            return true;
    }
    return false;
}
```

6.7.2.13 std::vector< IPAddress > PimInterface::deleteLocalIPs (std::vector< IPAddress > *multicastAddr*)

DELETE LOCAL IPs

The method removes all link local (224.0.0.0 to 224.0.0.255) multicast addresses from the list.

Parameters

<i>multicastAddr</i>	List of address which has to be checked.
----------------------	--

Returns

List of multicast address without link local IPs.

See also

[isLinkLocalMulticast\(\)](#)

Definition at line 86 of file [PimInterfaceTable.cc](#).

```

{
    EV << "PimInterface::deleteLocalIPs" << endl;

    for(int j = 0; j < (multicastAddr.size()); j++)
        EV << multicastAddr[j] << ", ";
    EV << endl;

    std::vector<IPAddress> newMulticastAddresses;
    for(unsigned int i = 0; i < multicastAddr.size(); i++)
    {
        EV << multicastAddr[i] << endl;
        if (!multicastAddr[i].isLinkLocalMulticast())
        {
            EV << "isLinkLocalMulticast" << endl;
            newMulticastAddresses.push_back(multicastAddr[i]);
        }
    }
    EV << "Velikost vysledku: " << newMulticastAddresses.size() << endl;
    return newMulticastAddresses;
}

```

6.7.3 Member Data Documentation

6.7.3.1 int PimInterface::intID [protected]

Identification of interface.

Definition at line 34 of file [PimInterfaceTable.h](#).

6.7.3.2 InterfaceEntry* PimInterface::intPtr [protected]

Pointer to interface table entry.

Definition at line 35 of file [PimInterfaceTable.h](#).

6.7.3.3 PIMmode PimInterface::mode [protected]

Type of mode.

Definition at line 36 of file [PimInterfaceTable.h](#).

6.7.3.4 std::vector<IPAddress> PimInterface::intMulticastAddresses [protected]

Multicast addresses assigned to interface.

Definition at line 37 of file [PimInterfaceTable.h](#).

The documentation for this class was generated from the following files:

- F:/ANSA/src/ansa/pim/tables/[PimInterfaceTable.h](#)
- F:/ANSA/src/ansa/pim/tables/[PimInterfaceTable.cc](#)

6.8 PimInterfaceTable Class Reference

Class represents Pim Interface Table.

```
#include <PimInterfaceTable.h>
```

Public Member Functions

- virtual [PimInterface](#) * [getInterface](#) (int k)
- virtual void [addInterface](#) (const [PimInterface](#) entry)

- virtual int [getNumInterface](#) ()
- virtual void [printPimInterfaces](#) ()
- virtual [PimInterface](#) * [getInterfaceByIntID](#) (int intID)

Protected Member Functions

- virtual void **initialize** (int stage)
- virtual void [handleMessage](#) (cMessage *)

Protected Attributes

- std::vector< [PimInterface](#) > [pimIft](#)

6.8.1 Detailed Description

Class represents Pim Interface Table.

It is vector of [PimInterface](#). Class contains methods to work with the table.

Definition at line 68 of file [PimInterfaceTable.h](#).

6.8.2 Member Function Documentation

6.8.2.1 virtual [PimInterface](#)* [PimInterfaceTable::getInterface](#) (int *k*) [inline, virtual]

Get pointer to entry of [PimInterfaceTable](#) from the object.

Definition at line 77 of file [PimInterfaceTable.h](#).

6.8.2.2 virtual void [PimInterfaceTable::addInterface](#) (const [PimInterface](#) *entry*) [inline, virtual]

Add entry to [PimInterfaceTable](#).

Definition at line 78 of file [PimInterfaceTable.h](#).

6.8.2.3 virtual int [PimInterfaceTable::getNumInterface](#) () [inline, virtual]

Returns number of entries in [PimInterfaceTable](#).

Definition at line 80 of file [PimInterfaceTable.h](#).

6.8.2.4 void [PimInterfaceTable::printPimInterfaces](#) () [virtual]

PRINT PIM INTERFACES

Actually not in use. Printout of Table of PIM interfaces

Definition at line 149 of file [PimInterfaceTable.cc](#).

```
{
    for(std::vector<PimInterface>::iterator i = pimIft.begin(); i < pimIft.
end(); i++)
    {
        EV << (*i).info() << endl;
    }
}
```

6.8.2.5 PimInterface * PimInterfaceTable::getInterfaceByIntID (int *intID*) [virtual]

Returns entry from [PimInterfaceTable](#) with given interface ID.

GET INTERFACE BY INTERFACE ID

The method finds interface in interface table by given interface ID.

Parameters

<i>intID</i>	ID of interface which is wanted.
--------------	----------------------------------

Returns

Returns link to wanted record in table.

See also

[getNumInterface\(\)](#)
[getInterface\(\)](#)

Definition at line 168 of file [PimInterfaceTable.cc](#).

```
{
    for(int i = 0; i < getNumInterface(); i++)
    {
        if(intID == getInterface(i)->getInterfaceID())
        {
            return getInterface(i);
            break;
        }
    }
    return NULL;
}
```

6.8.2.6 void PimInterfaceTable::handleMessage (cMessage * *msg*) [protected, virtual]

HANDLE MESSAGE

Module does not have any gate, it cannot get messages

Definition at line 133 of file [PimInterfaceTable.cc](#).

```
{
    opp_error("This module doesn't process messages");
}
```

6.8.3 Member Data Documentation**6.8.3.1 std::vector<PimInterface> PimInterfaceTable::pimlft [protected]**

List of PIM interfaces.

Definition at line 71 of file [PimInterfaceTable.h](#).

The documentation for this class was generated from the following files:

- F:/ANSA/src/ansa/pim/tables/[PimInterfaceTable.h](#)
- F:/ANSA/src/ansa/pim/tables/[PimInterfaceTable.cc](#)

6.9 PimInterfaceTableAccess Class Reference

Class gives access to the [PimInterfaceTable](#).

```
#include <PimInterfaceTable.h>
```

Public Member Functions

- virtual [PimInterfaceTable](#) * **getMyIfExists** ()

Private Attributes

- [PimInterfaceTable](#) * **p**

6.9.1 Detailed Description

Class gives access to the [PimInterfaceTable](#).

Definition at line 92 of file [PimInterfaceTable.h](#).

The documentation for this class was generated from the following file:

- F:/ANSA/src/ansa/pim/tables/[PimInterfaceTable.h](#)

6.10 PimNeighbor Class Reference

Class represents one entry of [PimNeighborTable](#).

```
#include <PimNeighborTable.h>
```

Public Member Functions

- virtual std::string **info** () const
- void **setId** (int **id**)
- void **setInterfaceID** (int **intID**)
- void **setInterfacePtr** (InterfaceEntry ***intPtr**)
- void **setAddr** (IPAddress **addr**)
- void **setVersion** (int **ver**)
- void **setNlt** (PIMnlt ***nlt**)
- int **getId** () const
- int **getInterfaceID** () const
- InterfaceEntry * **getInterfacePtr** () const
- IPAddress **getAddr** () const
- int **getVersion** () const
- PIMnlt * **getNlt** () const

Protected Attributes

- int **id**
- int **intID**
- InterfaceEntry * **intPtr**
- IPAddress **addr**
- int **ver**
- PIMnlt * **nlt**

6.10.1 Detailed Description

Class represents one entry of [PimNeighborTable](#).

Structure PIM neighbor with info about interface, IP address of neighbor link to Neighbor Liveness Timer and PIM version. The class contains methods to work with items of structure.

Definition at line 26 of file [PimNeighborTable.h](#).

6.10.2 Member Function Documentation

6.10.2.1 `std::string PimNeighbor::info () const` [virtual]

Printout of structure Neighbor table ([PimNeighbor](#)).

Definition at line 26 of file [PimNeighborTable.cc](#).

```
{
    std::stringstream out;
    out << id << ": ID = " << intID << "; Addr = " << addr << "; Ver = " <<
    ver;
    return out.str();
}
```

6.10.2.2 `void PimNeighbor::setId (int id)` [inline]

Set unique identifier of entry.

Definition at line 42 of file [PimNeighborTable.h](#).

6.10.2.3 `void PimNeighbor::setInterfaceID (int intID)` [inline]

Set interface ID.

Definition at line 43 of file [PimNeighborTable.h](#).

6.10.2.4 `void PimNeighbor::setInterfacePtr (InterfaceEntry * intPtr)` [inline]

Set pointer to interface.

Definition at line 44 of file [PimNeighborTable.h](#).

6.10.2.5 `void PimNeighbor::setAddr (IPAddress addr)` [inline]

Set IP address of neighbor.

Definition at line 45 of file [PimNeighborTable.h](#).

6.10.2.6 `void PimNeighbor::setVersion (int ver)` [inline]

Set PIM version (from Hello msg).

Definition at line 46 of file [PimNeighborTable.h](#).

6.10.2.7 `void PimNeighbor::setNlt (PIMnlt * nlt)` [inline]

Set pointer to NeighborLivenessTimer.

Definition at line 47 of file [PimNeighborTable.h](#).

6.10.2.8 `int PimNeighbor::getId () const` `[inline]`

Get unique identifier of entry.

Definition at line 51 of file [PimNeighborTable.h](#).

6.10.2.9 `int PimNeighbor::getInterfaceID () const` `[inline]`

Get interface ID.

Definition at line 52 of file [PimNeighborTable.h](#).

6.10.2.10 `InterfaceEntry* PimNeighbor::getInterfacePtr () const` `[inline]`

Get pointer to interface.

Definition at line 53 of file [PimNeighborTable.h](#).

6.10.2.11 `IPAddress PimNeighbor::getAddr () const` `[inline]`

Get IP address of neighbor.

Definition at line 54 of file [PimNeighborTable.h](#).

6.10.2.12 `int PimNeighbor::getVersion () const` `[inline]`

Get PIM version.

Definition at line 55 of file [PimNeighborTable.h](#).

6.10.2.13 `PIMnlt* PimNeighbor::getNlt () const` `[inline]`

Get pointer to NeighborLivenessTimer.

Definition at line 56 of file [PimNeighborTable.h](#).

6.10.3 Member Data Documentation

6.10.3.1 `int PimNeighbor::id` `[protected]`

Unique identifier of entry.

Definition at line 29 of file [PimNeighborTable.h](#).

6.10.3.2 `int PimNeighbor::intID` `[protected]`

Identification of interface.

Definition at line 30 of file [PimNeighborTable.h](#).

6.10.3.3 `InterfaceEntry* PimNeighbor::intPtr` `[protected]`

Link to interface table entry.

Definition at line 31 of file [PimNeighborTable.h](#).

6.10.3.4 IPAddress PimNeighbor::addr [protected]

IP address of neighbor.

Definition at line 32 of file [PimNeighborTable.h](#).

6.10.3.5 int PimNeighbor::ver [protected]

PIM version.

Definition at line 33 of file [PimNeighborTable.h](#).

6.10.3.6 PIMnt* PimNeighbor::nlt [protected]

Pointer to Neighbor Livness Timer.

Definition at line 34 of file [PimNeighborTable.h](#).

The documentation for this class was generated from the following files:

- F:/ANSA/src/ansa/pim/tables/[PimNeighborTable.h](#)
- F:/ANSA/src/ansa/pim/tables/[PimNeighborTable.cc](#)

6.11 PimNeighborTable Class Reference

Class represents Pim Neighbor Table.

```
#include <PimNeighborTable.h>
```

Public Member Functions

- virtual [PimNeighbor](#) * [getNeighbor](#) (int k)
- virtual void [addNeighbor](#) ([PimNeighbor](#) entry)
- virtual bool [deleteNeighbor](#) (int id)
- virtual int [getNumNeighbors](#) ()
- virtual void [printPimNeighborTable](#) ()
- virtual std::vector< [PimNeighbor](#) > [getNeighborsByIntID](#) (int intID)
- virtual [PimNeighbor](#) * [getNeighborsByID](#) (int id)
- virtual int [getIdCounter](#) ()
- virtual bool [isInTable](#) ([PimNeighbor](#) entry)
- virtual [PimNeighbor](#) * [findNeighbor](#) (int intId, IPAddress addr)
- virtual int [getNumNeighborsOnInt](#) (int intId)

Protected Member Functions

- virtual void [initialize](#) (int stage)
- virtual void [handleMessage](#) (cMessage *)

Protected Attributes

- int [id](#)
- std::vector< [PimNeighbor](#) > [nt](#)

6.11.1 Detailed Description

Class represents Pim Neighbor Table.

Table is list of [PimNeighbor](#) and class contains methods to work with them.

Definition at line 63 of file [PimNeighborTable.h](#).

6.11.2 Member Function Documentation

6.11.2.1 `virtual PimNeighbor* PimNeighborTable::getNeighbor (int k)` `[inline, virtual]`

Get k-th entry in the table

Definition at line 73 of file [PimNeighborTable.h](#).

6.11.2.2 `virtual void PimNeighborTable::addNeighbor (PimNeighbor entry)` `[inline, virtual]`

Add new entry to the table

Definition at line 74 of file [PimNeighborTable.h](#).

6.11.2.3 `bool PimNeighborTable::deleteNeighbor (int id)` `[virtual]`

DELETE NEIGHBOR

The method removes entry with given unique identifier from the table.

Parameters

<i>id</i>	Identifier of entry in the table.
-----------	-----------------------------------

Returns

True if entry was found and deleted successfully, otherwise false.

Definition at line 113 of file [PimNeighborTable.cc](#).

```
{
    for(int i = 0; i < getNumNeighbors(); i++)
    {
        if(id == getNeighbor(i)->getId())
        {
            nt.erase(nt.begin() + i);
            return true;
        }
    }
    return false;
}
```

6.11.2.4 `virtual int PimNeighborTable::getNumNeighbors ()` `[inline, virtual]`

Get number of entries in the table

Definition at line 76 of file [PimNeighborTable.h](#).

6.11.2.5 `void PimNeighborTable::printPimNeighborTable ()` `[virtual]`

PRINT PIM NEIGHBOR TABLE

Printout of Table of PIM interfaces

Definition at line 54 of file [PimNeighborTable.cc](#).

```
{
    for(std::vector<PimNeighbor>::iterator i = nt.begin(); i < nt.end(); i++)
    {
        EV << (*i).info() << endl;
    }
}
```

6.11.2.6 std::vector< PimNeighbor > PimNeighborTable::getNeighborsByIntID (int *intId*) [virtual]

GET NEIGHBORS BY INTERFACE ID

The method returns all neighbors which are connected to given router interface.

Parameters

<i>intId</i>	Identifier of interface.
--------------	--------------------------

Returns

Vector of entries from PIM neighbor table.

Definition at line 70 of file [PimNeighborTable.cc](#).

```
{
    vector<PimNeighbor> nbr;

    for(int i = 0; i < getNumNeighbors(); i++)
    {
        if(intId == getNeighbor(i)->getInterfaceID())
        {
            nbr.push_back(*getNeighbor(i));
        }
    }
    return nbr;
}
```

6.11.2.7 PimNeighbor * PimNeighborTable::getNeighborsByID (int *id*) [virtual]

GET NEIGHBOR BY ID

The method returns pointer to neighbor which is registered with given unique identifier.

Parameters

<i>id</i>	Identifier of entry in the table.
-----------	-----------------------------------

Returns

Pointer to entry from PIM neighbor table.

Definition at line 92 of file [PimNeighborTable.cc](#).

```
{
    for(int i = 0; i < getNumNeighbors(); i++)
    {
        if(id == getNeighbor(i)->getId())
        {
            return getNeighbor(i);
            break;
        }
    }
    return NULL;
}
```

6.11.2.8 `virtual int PimNeighborTable::getIdCounter() [inline, virtual]`

Get counter of entry IDs

Definition at line 80 of file [PimNeighborTable.h](#).

6.11.2.9 `bool PimNeighborTable::isInTable (PimNeighbor entry) [virtual]`

IS IN TABLE

The method finds out if given entry is present in the table.

Parameters

<i>entry</i>	PIM neighbor entry.
--------------	---------------------

Returns

True if entry was found in the table, otherwise false.

Definition at line 134 of file [PimNeighborTable.cc](#).

```
{
    for(int i = 0; i < getNumNeighbors(); i++)
    {
        if((entry.getAddr() == getNeighbor(i)->getAddr()) && (entry.
getInterfaceID() == getNeighbor(i)->getInterfaceID()))
            return true;
    }
    return false;
}
```

6.11.2.10 `PimNeighbor * PimNeighborTable::findNeighbor (int intId, IPAddress addr) [virtual]`

FIND NEIGHBOR

The method finds entry in the table according given interface ID and neighbor IP address.

Parameters

<i>intId</i>	Identifier of interface.
<i>addr</i>	IP address of neighbor.

Returns

Pointer to entry if entry was found in the table, otherwise NULL.

Definition at line 153 of file [PimNeighborTable.cc](#).

```
{
    for(int i = 0; i < getNumNeighbors(); i++)
    {
        if((addr == getNeighbor(i)->getAddr()) && (intId == getNeighbor
(i)->getInterfaceID()))
            return getNeighbor(i);
    }
    return NULL;
}
```

6.11.2.11 `int PimNeighborTable::getNumNeighborsOnInt (int intId) [virtual]`

GET NUMBER OF NEIGHBORS ON INTERFACE

The method returns number of neighbors which are connected to given interface.

Parameters

<i>intId</i>	Identifier of interface.
--------------	--------------------------

Returns

Number of neighbors which are connected to given interface.

Definition at line 171 of file [PimNeighborTable.cc](#).

```
{
    std::vector<PimNeighbor> neighbors = getNeighborsByIntID(intId);
    return neighbors.size();
}
```

6.11.2.12 void PimNeighborTable::handleMessage (cMessage * msg) [protected, virtual]

HANDLE MESSAGE

Module does not have any gate, it cannot get messages

Definition at line 38 of file [PimNeighborTable.cc](#).

```
{
    opp_error("This module doesn't process messages");
}
```

6.11.3 Member Data Documentation

6.11.3.1 int PimNeighborTable::id [protected]

Counter of [PimNeighbor](#) IDs

Definition at line 66 of file [PimNeighborTable.h](#).

6.11.3.2 std::vector<PimNeighbor> PimNeighborTable::nt [protected]

List of PIM neighbors (show ip pim neighbor)

Definition at line 67 of file [PimNeighborTable.h](#).

The documentation for this class was generated from the following files:

- F:/ANSA/src/ansa/pim/tables/[PimNeighborTable.h](#)
- F:/ANSA/src/ansa/pim/tables/[PimNeighborTable.cc](#)

6.12 PimNeighborTableAccess Class Reference

Class gives access to the [PimNeighborTable](#).

```
#include <PimNeighborTable.h>
```

6.12.1 Detailed Description

Class gives access to the [PimNeighborTable](#).

Definition at line 93 of file [PimNeighborTable.h](#).

The documentation for this class was generated from the following file:

- [F:/ANSA/src/ansa/pim/tables/PimNeighborTable.h](#)

6.13 pimSM Class Reference

Class implements PIM-SM (sparse mode).

```
#include <pimSM.h>
```

Protected Member Functions

- virtual int **numInitStages** () const
- virtual void **handleMessage** (cMessage *msg)
- virtual void **initialize** (int stage)

6.13.1 Detailed Description

Class implements PIM-SM (sparse mode).

Definition at line 19 of file [pimSM.h](#).

The documentation for this class was generated from the following files:

- [F:/ANSA/src/ansa/pim/modes/pimSM.h](#)
- [F:/ANSA/src/ansa/pim/modes/pimSM.cc](#)

6.14 PimSplitter Class Reference

Class implements PIM Splitter, which splits PIM messages to correct PIM module.

```
#include <PimSplitter.h>
```

Protected Member Functions

- virtual int **numInitStages** () const
- virtual void [handleMessage](#) (cMessage *msg)
- virtual void [initialize](#) (int stage)

Private Member Functions

- void [processPIMpkt](#) (PIMPacket *pkt)
- void [processNLTimer](#) (PIMTimer *timer)
- PIMHello * [createHelloPkt](#) (int ifID)
- void [sendHelloPkt](#) ()
- void [processHelloPkt](#) (PIMPacket *pkt)
- void [receiveChangeNotification](#) (int category, const cPolymorphic *details)
- virtual void [newMulticast](#) (IPAddress destAddr, IPAddress srcAddr)

- void [igmpChange](#) (InterfaceEntry *interface)
- bool [LoadConfigFromXML](#) (const char *filename)

Private Attributes

- IRoutingTable * [rt](#)
- [MulticastRoutingTable](#) * [mrt](#)
- IInterfaceTable * [ift](#)
- NotificationBoard * [nb](#)
- [PimInterfaceTable](#) * [pimIft](#)
- [PimNeighborTable](#) * [pimNbt](#)
- const char * [hostname](#)

6.14.1 Detailed Description

Class implements PIM Splitter, which splits PIM messages to correct PIM module.

This module is needed because we cannot distinguish PIM mode on layer 3, all of them have same protocol number (103). PIM Splitter can resend PIM message to correct PIM module according to configuration saved in [PimInterfaceTable](#). Splitter also manages [PimNeighborTable](#).

Definition at line 41 of file [PimSplitter.h](#).

6.14.2 Member Function Documentation

6.14.2.1 void PimSplitter::processPIMPkt (PIMPacket * pkt) [private]

PROCESS PIM PACKET

The method processes new coming PIM packet. It has to find out where packet has to be sent to. It looks to interface from which packet is coming. According to interface ID, method findes record in PIM Interface Table and gets info about PIM mode. According to mode it send to appropriate PIM model.

Parameters

<i>pkt</i>	New coming PIM packet.
------------	------------------------

See also

[PimInterface](#)

Definition at line 168 of file [PimSplitter.cc](#).

```
{
    EV << "PIM::processPIMPkt" << endl;

    IPControlInfo *ctrl = dynamic_cast<IPControlInfo *>(pkt->getControlInfo
());
    int intID = ctrl->getInterfaceId();
    int mode = 0;

    // find information about interface where packet came from
    PimInterface *pimInt = pimIft->getInterfaceByIntID(intID);
    if (pimInt != NULL)
        mode = pimInt->getMode();

    // according to interface PIM mode send packet to appropriate PIM
    module
    switch(mode)
    {
        case Dense:
            send(pkt, "pimDMOut");
            break;
    }
}
```

```

        case Sparse:
            send(pkt, "pimSMOut");
            break;
        default:
            EV << "PIM::processPIMPkt: PIM is not enabled on
interface number: "<< intID << endl;
            delete pkt;
    }
}

```

6.14.2.2 void PimSplitter::processNLTimer (PIMTimer * timer) [private]

PROCESS NEIGHBOR LIVENESS TIMER

The method process Neighbor Liveness Timer. After its expiration neighbor is removed from [PimNeighborTable](#).

Parameters

<i>timer</i>	PIM Neighbor Liveness Timer.
--------------	------------------------------

See also

[PimNeighbor](#)
PIMnlt()

Definition at line 138 of file [PimSplitter.cc](#).

```

{
    EV << "PIM::processNLTimer"<< endl;
    PIMnlt *nlt = check_and_cast <PIMnlt *> (timer);
    int id = nlt->getNtId();
    IPAddress neighbor;

    // if neighbor exists store its IP address
    if (pimNbt->getNeighborsByID(id) != NULL)
        neighbor = pimNbt->getNeighborsByID(id)->getAddr();

    // Record in PIM Neighbor Table was found, can be deleted.
    if (pimNbt->deleteNeighbor(id))
        EV << "PIM::processNLTimer: Neighbor " << neighbor << "was
removed from PIM neighbor table." << endl;

    delete nlt;
}

```

6.14.2.3 PIMHello * PimSplitter::createHelloPkt (int iftID) [private]

CREATE HELLO PACKET

The method creates new PIM Hello Packet and sets all necessary info.

Parameters

<i>iftID</i>	ID of interface to which the packet has to be sent
--------------	--

Returns

Return PIMHello message, which is ready to be sent.

See also

PIMHello

Definition at line 27 of file [PimSplitter.cc](#).


```

{
    PIMHello *msg = new PIMHello();
    msg->setName("PIMHello");

    IPControlInfo *ctrl = new IPControlInfo();
    IPAddress gal("224.0.0.13");
    ctrl->setDestAddr(gal);
    //ctrl->setProtocol(IP_PROT_PIM);
    ctrl->setProtocol(103);
    ctrl->setTimeToLive(1);
    ctrl->setInterfaceId(iftID);
    msg->setControlInfo(ctrl);

    return msg;
}

```

6.14.2.4 void PimSplitter::sendHelloPkt() [private]

SEND HELLO PACKET

The method goes through all PIM interfaces and sends Hello packet to each of them. It also schedule next sending of Hello packets (sets Hello Timer).

See also

[createHelloPkt\(\)](#)

Definition at line 53 of file [PimSplitter.cc](#).

```

{
    EV << "PIM::sendHelloPkt" << endl;
    int intID;
    PIMHello* msg;

    // send to all PIM interfaces
    for (int i = 0; i < pimIft->getNumInterface(); i++)
    {
        intID = pimIft->getInterface(i)->getInterfaceID();
        msg = createHelloPkt(intID);
        send(msg, "transportOut");
    }

    // start Hello timer
    PIMTimer *timer = new PIMTimer("Hello");
    timer->setTimerKind(HelloTimer);
    scheduleAt(simTime() + HT, timer);
}

```

6.14.2.5 void PimSplitter::processHelloPkt (PIMPacket * msg) [private]

PROCESS HELLO PACKET

The method processes new coming Hello packet from any of its neighbor. It reads info about neighbor from the packet and tries to find neighbor in [PimNeighborTable](#). If neighbor is not in the table, method adds him and sets Neighbor Liveness Timer for the record. If neighbor is already in [PimNeighborTable](#) it refreshes Neighbor Liveness Timer

Parameters

<i>msg</i>	Pointer to incoming Hello packet
------------	----------------------------------

See also

[PimNeighbor](#)
[PIMnlt](#)

Definition at line 87 of file [PimSplitter.cc](#).

```

{
    EV << "PIM::processHelloPkt" << endl;

    IPControlInfo *ctrl = dynamic_cast<IPControlInfo *>(msg->getControlInfo
());
    PimNeighbor newEntry;
    PIMnlt *nlt;

    // get information about neighbor from Hello packet
    newEntry.setAddr(ctrl->getSrcAddr());
    newEntry.setInterfaceID(ctrl->getInterfaceId());
    newEntry.setInterfacePtr(ift->getInterfaceById(ctrl->getInterfaceId()));
;
    newEntry.setVersion(msg->getVersion());

    // new neighbor (it is not in PIM neighbor table)
    // insert new neighbor to table
    // set Neighbor Livness Timer
    if (!pimNbt->isInTable(newEntry))
    {
        nlt = new PIMnlt("NeighborLivenessTimer");
        nlt->setTimerKind(NeighborLivenessTimer);
        nlt->setNtId(pimNbt->getIdCounter());
        scheduleAt(simTime() + 3.5*HT, nlt);

        newEntry.setNlt(nlt);
        pimNbt->addNeighbor(newEntry);
        EV << "PimSplitter::New Entry was added: addr = " << newEntry.
getAddr() << ", iftID = " << newEntry.getInterfaceID() << ", ver = " <<
newEntry.getVersion() << endl;
    }
    // neighbor is already in PIM neighbor table
    // refresh Neighbor Livness Timer
    else
    {
        nlt = pimNbt->findNeighbor(ctrl->getInterfaceId(), ctrl->
getSrcAddr())->getNlt();
        cancelEvent(nlt);
        scheduleAt(simTime() + 3.5*HT, nlt);
    }

    delete msg;
}

```

6.14.2.6 void PimSplitter::receiveChangeNotification (int *category*, const cPolymorphic * *details*) [private]

RECEIVE CHANGE NOTIFICATION

The method from class Notification Board is used to catch its events.

Parameters

<i>category</i>	Category of notification.
<i>details</i>	Additional information for notification.

See also

[newMulticast\(\)](#)
[igmpChange\(\)](#)

Definition at line 306 of file [PimSplitter.cc](#).

```

{
    // ignore notifications during initialize
    if (simulation.getContextType()==CTX_INITIALIZE)
        return;

    // PIM needs details
    if (details == NULL)
        return;

    Enter_Method_Silent();
    printNotificationBanner(category, details);

    // according to category of event...

```

```

switch (category)
{
    // new multicast data appears in router
    case NF_IPv4_NEW_MULTICAST:
        EV << "PimSplitter::receiveChangeNotification - NEW
MULTICAST" << endl;
        IPControlInfo *ctrl;
        ctrl = (IPControlInfo *) (details);
        newMulticast(ctrl->getDestAddr(), ctrl->getSrcAddr());
        break;

    // configuration of interface changed, it means some change
    from IGMP
    case NF_INTERFACE_IPv4CONFIG_CHANGED:
        EV << "PimSplitter::receiveChangeNotification - IGMP
change" << endl;
        InterfaceEntry * interface = (InterfaceEntry *) (details);
};
        igmpChange(interface);
        break;
}
}

```

6.14.2.7 void PimSplitter::newMulticast (IPAddress *destAddr*, IPAddress *srcAddr*) [private, virtual]

NEW MULTICAST

The method process notification about new coming multicast data. According to IP addresses it find all necessary info to create new entry for multicast table.

Parameters

<i>destAddr</i>	Destination IP address = multicast group IP address.
<i>srcAddr</i>	Source IP address.

See also

[MulticastIPRoute](#)

Definition at line 433 of file [PimSplitter.cc](#).

```

{
    EV << "PimSplitter::newMulticast - group: " << destAddr << ", source: "
<< srcAddr << endl;

    // find RPF interface for new multicast stream
    InterfaceEntry *inInt = rt->getInterfaceForDestAddr(srcAddr);
    if (inInt == NULL)
    {
        EV << "ERROR: PimSplitter::newMulticast(): cannot find RPF
interface, routing information is missing.";
        return;
    }
    int rpfId = inInt->getInterfaceId();
    PimInterface *pimInt = pimIft->getInterfaceByID(rpfId);

    // if it is interface configured with PIM, create new route
    if (pimInt != NULL)
    {
        // create new multicast route
        MulticastIPRoute *newRoute = new MulticastIPRoute();
        newRoute->setGroup(destAddr);
        newRoute->setSource(srcAddr);

        // Directly connected routes to source does not have next hop
        // RPF neighbor is source of packet
        IPAddress rpf;
        const IPRoute *routeToSrc = rt->findBestMatchingRoute(srcAddr);
        if (routeToSrc->getSource() == IPRoute::IFACENETMASK)
        {
            newRoute->addFlag(A);
            rpf = srcAddr;
        }
        // Not directly connected, next hop address is saved in routing
        table
        else

```

```

        rpf = rt->getGatewayForDestAddr(srcAddr);

        newRoute->setInInt(inInt, inInt->getInterfaceId(), rpf);

        // notification for PIM module about new multicast route
        if (pimInt->getMode() == Dense)
            nb->fireChangeNotification(NF_IPv4_NEW_MULTICAST_DENSE,
newRoute);
    }
}

```

6.14.2.8 void PimSplitter::igmpChange (InterfaceEntry * *interface*) [private]

IGMP CHANGE

The method is used to process notification about IGMP change. Splitter will find out which IP address were added or removed from interface and will send them to appropriate PIM mode.

Parameters

<i>interface</i>	Pointer to interface where IP address changed.
------------------	--

See also

[addRemoveAddr](#)

Definition at line 348 of file [PimSplitter.cc](#).

```

{
    EV << "PimSplitter::igmpChange" << endl;
    int intId = interface->getInterfaceId();
    PimInterface * pimInt = pimIft->getInterfaceByIntID(intId);

    // save old and new set of multicast IP address assigned to interface
    vector<IPAddress> multicastAddrsOld = pimInt->getIntMulticastAddresses(
);
    vector<IPAddress> multicastAddrsNew = pimInt->deleteLocalIPs(interface
->ipv4Data()->getMulticastGroups());

    // vectors of new and removed multicast addresses
    vector<IPAddress> add;
    vector<IPAddress> remove;

    // which address was removed from interface
    for (unsigned int i = 0; i < multicastAddrsOld.size(); i++)
    {
        unsigned int j;
        for (j = 0; j < multicastAddrsNew.size(); j++)
        {
            if (multicastAddrsOld[i] == multicastAddrsNew[j])
                break;
        }
        if (j == multicastAddrsNew.size())
        {
            EV << "Multicast address " << multicastAddrsOld[i] << "
was removed from the interface " << intId << endl;
            remove.push_back(multicastAddrsOld[i]);
        }
    }

    // which address was added to interface
    for (unsigned int i = 0; i < multicastAddrsNew.size(); i++)
    {
        unsigned int j;
        for (j = 0; j < multicastAddrsOld.size(); j++)
        {
            if (multicastAddrsNew[i] == multicastAddrsOld[j])
                break;
        }
        if (j == multicastAddrsOld.size())
        {
            EV << "Multicast address " << multicastAddrsNew[i] << "
was added to the interface " << intId << endl;
            add.push_back(multicastAddrsNew[i]);
        }
    }
}

```

```

// notification about removed multicast address to PIM modules
addRemoveAddr *addr = new addRemoveAddr();
if (remove.size() > 0)
{
    // remove new address
    for(unsigned int i = 0; i < remove.size(); i++)
        pimInt->removeIntMulticastAddress(remove[i]);

    // send notification
    addr->setAddr(remove);
    addr->setInt(pimInt);
    nb->fireChangeNotification(NF_IPv4_NEW_IGMP_REMOVED, addr);
}

// notification about new multicast address to PIM modules
if (add.size() > 0)
{
    // add new address
    for(unsigned int i = 0; i < add.size(); i++)
        pimInt->addIntMulticastAddress(add[i]);

    // send notification
    addr->setAddr(add);
    addr->setInt(pimInt);
    nb->fireChangeNotification(NF_IPv4_NEW_IGMP_ADDED, addr);
}
}

```

6.14.2.9 bool PimSplitter::LoadConfigFromXML (const char * filename) [private]

LOAD CONFIG FROM XML

The method is not used now. Config is loaded by class DeviceConfigurator.

The method provides loading of configuration for protocol PIM from configuration file. The main information is if router has enabled multicast and on which interfaces which mode of PIM is set up.

```

<Routing> <Multicast enable="1"></Multicast> </Routing> <Interfaces> <Interface name="eth0"> <Pim>
<Mode>dense-mode</Mode> </Pim> </Interface> </Interfaces>

```

Parameters

<i>filename</i>	Name of configuration file.
-----------------	-----------------------------

Returns

Return true, if loading was successful, or false.

See also

DeviceConfigurator

Definition at line 500 of file [PimSplitter.cc](#).

```

{
    // file loading
    cXMLElement* asConfig = ev.getXMLDocument(filename);
    if (asConfig == NULL)
        return false;

    // first element <Router id="192.168.10.7">
    std::string routerXPath("Router[@id='");
    IPAddress routerId = rt->getRouterId();
    routerXPath += routerId.str();
    routerXPath += "']";

    cXMLElement* routerNode = asConfig->getElementByPath(routerXPath.c_str(
));
    if (routerNode == NULL)
    {
        error("No configuration for Router ID: %s", routerId.str().
c_str());
    }
}

```

```

        return false;
    }

    // Routing element
    cXMLElement* routingNode = routerNode->getElementByPath("Routing");
    if (routingNode == NULL)
        return false;

    // Multicast element
    cXMLElement* multicastNode = routingNode->getElementByPath("Multicast");
    if (multicastNode == NULL)
        return false;

    // Multicast has to be enabled
    const char* enableAtt = multicastNode->getAttribute("enable");
    if (strcmp(enableAtt, "1"))
        return false;

    // Where is PIM protocol enabled?
    // Interfaces element
    cXMLElement* iftNode = routerNode->getElementByPath("Interfaces");
    if (iftNode == NULL)
        return false;

    // list of interfaces, where PIM is enabled
    cXMLElementList childrenNodes = iftNode->getChildrenByTagName("Interface");
    //EV << "PimSplitter::Interface" << endl;
    if (childrenNodes.size() > 0)
    {
        //EV << "PimSplitter::Interface size: " << childrenNodes.size() <<
        endl;
        for (cXMLElementList::iterator node = childrenNodes.begin(); node !=
            childrenNodes.end(); node++)
        {
            cXMLElement* pimNode = (*node)->getElementByPath("Pim");
            if (pimNode == NULL)
                continue;
            //EV << "PimSplitter::PIM interface" << endl;
            // get ID of PIM interface
            InterfaceEntry *interface = iftNode->getInterfaceByName((*node)->
                getAttribute("name"));

            // create new PIM interface
            PimInterface newentry;
            newentry.setInterfaceID(interface->getInterfaceId());
            newentry.setInterfacePtr(interface);

            // register pim multicast address 224.0.0.13 on pim interface
            vector<IPAddress> intMulticastAddresses = interface->ipv4Data
            ()->getMulticastGroups();
            intMulticastAddresses.push_back("224.0.0.13");
            interface->ipv4Data()->setMulticastGroups(
                intMulticastAddresses);

            // get PIM mode for interface
            cXMLElement* pimMode = pimNode->getElementByPath("Mode");
            if (pimMode == NULL)
                return false;

            const char *mode = pimMode->getNodeValue();
            //EV << "PimSplitter::PIM interface mode = " << mode << endl;
            if (!strcmp(mode, "dense-mode"))
                newentry.setMode(Dense);
            else if (!strcmp(mode, "sparse-mode"))
                newentry.setMode(Sparse);
            else
                return false;
            pimIft->addInterface(newentry);
        }
    }
    else
        return false;
    return true;
}

```

6.14.2.10 void PimSplitter::handleMessage (cMessage * msg) [protected, virtual]

HANDLE MESSAGE

The method handles new coming message and process it according to its type. Self message is timer. Other

messages should be PIM packets.

Parameters

<i>msg</i>	Pointer to message which has came to the module.
------------	--

See also

[PIMTimer](#)
[sendHelloPkt\(\)](#)
[processNLTimer\(\)](#)
[PIMPacket](#)
[processHelloPkt\(\)](#)
[processPIMPkt\(\)](#)

Definition at line 210 of file [PimSplitter.cc](#).

```
{
    EV << "PimSplitter::handleMessage" << endl;

    // self message (timer)
    if (msg->isSelfMessage())
    {
        PIMTimer *timer = check_and_cast<PIMTimer *>(msg);
        if (timer->getTimerKind() == HelloTimer)
        {
            EV << "PIM::HelloTimer" << endl;
            sendHelloPkt();
            delete timer;
        }
        else if (timer->getTimerKind() == NeighborLivenessTimer)
        {
            EV << "PIM::NeighborLivnessTimer" << endl;
            processNLTimer(timer);
        }
    }

    // PIM packet from network layer
    else if (dynamic_cast<PIMPacket *>(msg) && (strcmp(msg->getArrivalGate()->
        getName(), "transportIn") == 0))
    {
        PIMPacket *pkt = check_and_cast<PIMPacket *>(msg);

        if (pkt->getType() == Hello)
            processHelloPkt(pkt);
        else
            processPIMPkt(pkt);
    }

    // PIM packet from PIM mode, send to network layer
    else if (dynamic_cast<PIMPacket *>(msg))
        send(msg, "transportOut");

    else
        EV << "PIM:ERROR - bad type of message" << endl;
}
```

6.14.2.11 void PimSplitter::initialize (int stage) [protected, virtual]

INITIALIZE

The method initialize ale structures (tables) which will use. It subscribes to appropriate events of Notification Board. If there is no PIM interface, PIM stops working. Otherwise it schedule Hello Timer.

Parameters

<i>stage</i>	Stage of initialization.
--------------	--------------------------

See also

[MulticastRoutingTable](#)

[PIMTimer](#)

Definition at line 261 of file [PimSplitter.cc](#).

```
{
    // in stage 2 interfaces are registered
    // in stage 3 table pimInterfaces is built
    if (stage == 4)
    {
        EV << "PimSplitter::initialize" << endl;
        hostname = par("hostname");

        // Pointer to routing tables, interface tables, notification
board
        rt = RoutingTableAccess().get();
        mrt = MulticastRoutingTableAccess().get();
        ift = AnsaInterfaceTableAccess().get();
        nb = NotificationBoardAccess().get();
        pimIft = PimInterfaceTableAccess().get();
        pimNbt = PimNeighborTableAccess().get();

        // subscription of notifications (future use)
        nb->subscribe(this, NF_IPv4_NEW_MULTICAST);
        nb->subscribe(this, NF_INTERFACE_IPv4CONFIG_CHANGED);

        // is PIM enabled?
        if (pimIft->getNumInterface() == 0)
            return;
        else
            EV << "PIM is enabled on device " << hostname << endl;

        // send Hello packets to PIM neighbors (224.0.0.13)
        PIMTimer *timer = new PIMTimer("Hello");
        timer->setTimerKind(HelloTimer);
        scheduleAt(simTime() + uniform(0,5), timer);
    }
}
```

6.14.3 Member Data Documentation

6.14.3.1 `IRoutingTable* PimSplitter::rt` [private]

Pointer to routing table.

Definition at line 44 of file [PimSplitter.h](#).

6.14.3.2 `MulticastRoutingTable* PimSplitter::mrt` [private]

Pointer to multicast routing table.

Definition at line 45 of file [PimSplitter.h](#).

6.14.3.3 `IInterfaceTable* PimSplitter::ift` [private]

Pointer to interface table.

Definition at line 46 of file [PimSplitter.h](#).

6.14.3.4 `NotificationBoard* PimSplitter::nb` [private]

Pointer to notification table.

Definition at line 47 of file [PimSplitter.h](#).

6.14.3.5 PimInterfaceTable* PimSplitter::pimlft [private]

Pointer to table of PIM interfaces.

Definition at line 48 of file [PimSplitter.h](#).

6.14.3.6 PimNeighborTable* PimSplitter::pimNbt [private]

Pointer to table of PIM neighbors.

Definition at line 49 of file [PimSplitter.h](#).

6.14.3.7 const char* PimSplitter::hostname [private]

Router hostname.

Definition at line 50 of file [PimSplitter.h](#).

The documentation for this class was generated from the following files:

- [F:/ANSA/src/ansa/pim/PimSplitter.h](#)
- [F:/ANSA/src/ansa/pim/PimSplitter.cc](#)

Chapter 7

File Documentation

7.1 F:/ANSA/src/ansa/ipv4/AnsaIP.cc File Reference

File contains extension of class IP, which can work also with multicast data and multicast.

```
#include <omnetpp.h>
#include "AnsaIP.h"
#include "PimSplitter.h"
```

Functions

- **Define_Module** ([AnsaIP](#))

7.1.1 Detailed Description

File contains extension of class IP, which can work also with multicast data and multicast.

Author

Veronika Rybova

Date

10.10.2011

Definition in file [AnsaIP.cc](#).

7.2 AnsaIP.cc

```
00001
00009 #include <omnetpp.h>
00010 #include "AnsaIP.h"
00011 #include "PimSplitter.h"
00012
00013 Define_Module (AnsaIP);
00014
00015
00023 void AnsaIP::initialize(int stage)
00024 {
00025     INET_API IP::initialize();
00026
00027     mrt = MulticastRoutingTableAccess().get();
00028     nb = NotificationBoardAccess().get();
```

```

00029
00030 }
00031
00044 void AnsaIP::handlePacketFromNetwork(IPDatagram *datagram)
00045 {
00046     //
00047     // "Prerouting"
00048     //
00049
00050     // check for header biterror
00051     if (datagram->hasBitError())
00052     {
00053         // probability of bit error in header = size of header / size of total
00054         message // (ignore bit error if in payload)
00055         double relativeHeaderLength = datagram->getHeaderLength() / (double)
00056         datagram->getByteLength();
00057         if (dblrand() <= relativeHeaderLength)
00058         {
00059             EV << "bit error found, sending ICMP_PARAMETER_PROBLEM\n";
00059             icmpAccess.get()->sendErrorMessage(datagram, ICMP_PARAMETER_PROBLEM
, 0);
00060             return;
00061         }
00062     }
00063
00064     // remove control info
00065     if (datagram->getTransportProtocol()!=IP_PROT_DSR && datagram->
getTransportProtocol()!=IP_PROT_MANET && !datagram->getDestAddress().isMulticast() &&
datagram->getTransportProtocol()!=IP_PROT_PIM)
00066     {
00067         delete datagram->removeControlInfo();
00068     }
00069     else if (datagram->getMoreFragments())
00070     delete datagram->removeControlInfo(); // delete all control message
except the last
00071
00072     //MYWORK Add all neccessary info to the IP Control Info for future use.
00073     if (datagram->getDestAddress().isMulticast() || datagram->
getTransportProtocol() == IP_PROT_PIM)
00074     {
00075         IPControlInfo *ctrl = (IPControlInfo*)(datagram->removeControlInfo());
00076         ctrl->setSrcAddr(datagram->getSrcAddress());
00077         ctrl->setDestAddr(datagram->getDestAddress());
00078         ctrl->setInterfaceId(getSourceInterfaceFrom(datagram)->getInterfaceId()
);
00079         datagram->setControlInfo(ctrl);
00080     }
00081
00082     // hop counter decrement; FIXME but not if it will be locally delivered
00083     datagram->setTimeToLive(datagram->getTimeToLive()-1);
00084
00085     // send IGMP packet to IGMP module
00086     if (datagram->getTransportProtocol() == IP_PROT_IGMP)
00087     {
00088         cPacket *packet = decapsulateIP(datagram);
00089         send(packet, "transportOut", mapping.getOutputGateForProtocol(
IP_PROT_IGMP));
00090         return;
00091     }
00092
00093     //MYWORK send PIM packet to PIM module
00094     if (datagram->getTransportProtocol() == IP_PROT_PIM)
00095     {
00096         cPacket *packet = decapsulateIP(datagram);
00097         send(packet, "transportOut", mapping.getOutputGateForProtocol(
IP_PROT_PIM));
00098         return;
00099     }
00100
00101     //MYWORK route packet
00102     if (!datagram->getDestAddress().isMulticast())
00103         routePacket(datagram, NULL, false, NULL);
00104     else
00105         routeMulticastPacket(datagram, NULL, getSourceInterfaceFrom(datagram));
00106 }
00107
00124 void AnsaIP::routeMulticastPacket(IPDatagram *datagram, InterfaceEntry *destIE,
InterfaceEntry *fromIE)
00125 {
00126     IPAddress destAddr = datagram->getDestAddress();
00127     IPAddress srcAddr = datagram->getSrcAddress();
00128     IPControlInfo *ctrl = (IPControlInfo *) datagram->getControlInfo();
00129     EV << "Routing multicast datagram '" << datagram->getName() << "' with
dest=" << destAddr << "\n";
00130     MulticastIPRoute *route = mrt->getRouteFor(destAddr, srcAddr);
00131

```

```

00132     numMulticast++;
00133
00134     // Process datagram only if sent locally or arrived on the shortest
00135     // route (provided routing table already contains srcAddr) = RPF interface;
00136     // otherwise discard and continue.
00137     InterfaceEntry *rpfInt = rt->getInterfaceForDestAddr(datagram->
getSrcAddress());
00138     if (fromIE!=NULL && rpfInt!=NULL && fromIE!=rpfInt)
00139     {
00140         //MYWORK RPF interface has changed
00141         //if (route != NULL && (route->getInIntId() !=
rpfInt->getInterfaceId()))
00142         {
00143             EV << "RPF interface has changed" << endl;
00144             nb->fireChangeNotification(NF_IPv4_RPF_CHANGE, route);
00145         }
00146         //MYWORK Data come to non-RPF interface
00147         if (!rt->isLocalMulticastAddress(destAddr) && !destAddr.
isLinkLocalMulticast())
00148         {
00149             EV << "Data on non-RPF interface" << endl;
00150             nb->fireChangeNotification(NF_IPv4_DATA_ON_NONRPF, ctrl);
00151             return;
00152         }
00153         else
00154         {
00155             // FIXME count dropped
00156             EV << "Packet dropped." << endl;
00157             delete datagram;
00158             return;
00159         }
00160     }
00161
00162     //MYWORK for local traffic to given destination (PIM messages)
00163     if (fromIE == NULL && destIE != NULL)
00164     {
00165         IPDatagram *datagramCopy = (IPDatagram *) datagram->dup();
00166         datagramCopy->setSrcAddress(destIE->ipv4Data()->getIPAddress());
00167     };
00168     fragmentAndSend(datagramCopy, destIE, destAddr);
00169     delete datagram;
00170     return;
00171 }
00172
00173 // if received from the network...
00174 if (fromIE!=NULL)
00175 {
00176     EV << "Packet was received from the network..." << endl;
00177     // check for local delivery (multicast assigned to any interface)
00178     if (rt->isLocalMulticastAddress(destAddr))
00179     {
00180         EV << "isLocalMulticastAddress." << endl;
00181         IPDatagram *datagramCopy = (IPDatagram *) datagram->dup();
00182
00183         // FIXME code from the MPLS model: set packet dest address to
routerId
00184         datagramCopy->setDestAddress(rt->getRouterId());
00185         reassembleAndDeliver(datagramCopy);
00186     }
00187
00188     // don't forward if IP forwarding is off
00189     if (!rt->isIPForwardingEnabled())
00190     {
00191         EV << "IP forwarding is off." << endl;
00192         delete datagram;
00193         return;
00194     }
00195
00196     // don't forward if dest address is link-scope
00197     // address is in the range 224.0.0.0 to 224.0.0.255
00198     if (destAddr.isLinkLocalMulticast())
00199     {
00200         EV << "isLinkLocalMulticast." << endl;
00201         delete datagram;
00202         return;
00203     }
00204 }
00205
00206 //MYWORK(to the end) now: routing
00207 EV << "AnsaIP::routeMulticastPacket - Multicast routing." << endl;
00208
00209 // multicast group is not in multicast routing table and has to be added
00210 if (route == NULL)
00211 {
00212     EV << "AnsaIP::routeMulticastPacket - Multicast route does not exist,
try to add." << endl;

```

```

00213         nb->fireChangeNotification(NF_IPv4_NEW_MULTICAST, ctrl);
00214         delete datagram->removeControlInfo();
00215         ctrl = NULL;
00216         // read new record
00217         route = mrt->getRouteFor(destAddr, srcAddr);
00218     }
00219
00220     if (route == NULL)
00221     {
00222         EV << "Still do not exist." << endl;
00223         delete datagram;
00224         return;
00225     }
00226
00227     nb->fireChangeNotification(NF_IPv4_DATA_ON_RPF, route);
00228
00229     // data won't be sent because there is no outgoing interface and/or
    route is pruned
00230     InterfaceVector outInt = route->getOutInt();
00231     if (outInt.size() == 0 || route->isFlagSet(P))
00232     {
00233         EV << "Route does not have any outgoing interface or it is pruned." <<
endl;
00234         if (ctrl != NULL)
00235         {
00236             if (!route->isFlagSet(A))
00237                 nb->fireChangeNotification(
NF_IPv4_DATA_ON_PRUNED_INT, ctrl);
00238         }
00239         delete datagram;
00240         return;
00241     }
00242
00243     // send packet to all outgoing interfaces of route (oilst)
00244     for (unsigned int i=0; i<outInt.size(); i++)
00245     {
00246         // do not send to pruned interface
00247         if (outInt[i].forwarding == Pruned)
00248             continue;
00249
00250         InterfaceEntry *destIE = outInt[i].intPtr;
00251         IPDatagram *datagramCopy = (IPDatagram *) datagram->dup();
00252
00253         // set datagram source address if not yet set
00254         if (datagramCopy->getSrcAddress().isUnspecified())
00255             datagramCopy->setSrcAddress(destIE->ipv4Data()->
getIPAddress());
00256
00257         // send
00258         fragmentAndSend(datagramCopy, destIE, destAddr);
00259     }
00260
00261     // only copies sent, delete original datagram
00262     delete datagram;
00263 }
00264

```

7.3 F:/ANSA/src/ansa/ipv4/AnsaIP.h File Reference

File contains extension of class IP, which can work also with multicast data and multicast.

```
#include "QueueBase.h"
#include "InterfaceTableAccess.h"
#include "IRoutingTable.h"
#include "ICMPAccess.h"
#include "IPControlInfo.h"
#include "IPDatagram.h"
#include "IPFragBuf.h"
#include "ProtocolMap.h"
#include "ControlManetRouting_m.h"
#include "ICMPMessage_m.h"
#include "IPv4InterfaceData.h"
#include "ARPPacket_m.h"
#include "IP.h"
#include "PimSplitter.h"
#include "MulticastRoutingTableAccess.h"
```

Classes

- class [AnsaIP](#)
Class is extension of the IP protocol implementation for multicast.

Enumerations

- enum [AnsaIPProtocolId](#) { [IP_PROT_PIM](#) = 103 }

7.3.1 Detailed Description

File contains extension of class IP, which can work also with multicast data and multicast.

Author

Veronika Rybova

Date

10.10.2011

Definition in file [AnsaIP.h](#).

7.4 AnsaIP.h

```
00001
00008 #ifndef __INET_ANSAIP_H
00009 #define __INET_ANSAIP_H
00010
00011 #include "QueueBase.h"
00012 #include "InterfaceTableAccess.h"
00013 #include "IRoutingTable.h"
00014 #include "ICMPAccess.h"
00015 #include "IPControlInfo.h"
00016 #include "IPDatagram.h"
00017 #include "IPFragBuf.h"
00018 #include "ProtocolMap.h"
00019 #include "ControlManetRouting_m.h"
00020 #include "ICMPMessage_m.h"
00021 #include "IPv4InterfaceData.h"
00022 #include "ARPPacket_m.h"
00023 #include "IP.h"
00024 #include "PimSplitter.h"
```

```

00025 #include "MulticastRoutingTableAccess.h"
00026
00027
00028 class ARPPacket;
00029 class ICMPMessage;
00030
00031 //FIXME it shouldn't be there, but somewhere more globally
00032 enum AnsaIPProtocolId
00033 {
00034     IP_PROT_PIM = 103
00035 };
00036
00037
00042 class INET_API AnsaIP : public IP
00043 {
00044     private:
00045         MulticastRoutingTable *mrt;
00046         NotificationBoard *nb;
00047     protected:
00050         virtual void handlePacketFromNetwork(IPDatagram *datagram);
00051         virtual void routeMulticastPacket(IPDatagram *datagram,
InterfaceEntry *destIE, InterfaceEntry *fromIE);
00052
00053     public:
00054         AnsaIP() {}
00055
00056     protected:
00057
00058         virtual int numInitStages() const {return 5;}
00059         virtual void initialize(int stage);
00060 };
00061
00062 #endif
00063

```

7.5 F:/ANSA/src/ansa/multicastRoutingTable/MulticastIPRoute.cc File Reference

File contains implementation of multicast route.

```
#include "MulticastIPRoute.h"
```

7.5.1 Detailed Description

File contains implementation of multicast route.

Author

Veronika Rybova

Date

10.10.2011

Definition in file [MulticastIPRoute.cc](#).

7.6 MulticastIPRoute.cc

```

00001
00008 #include "MulticastIPRoute.h"
00009
00010 using namespace std;
00011
00012 MulticastIPRoute::MulticastIPRoute()
00013 {
00014     inInt.intPtr = NULL;
00015     grt = NULL;
00016     sat = NULL;
00017     srt = NULL;

```



```

00018 }
00019
00020 string MulticastIPRoute::info() const
00021 {
00022     stringstream out;
00023     out << "(";
00024     if (source.isUnspecified()) out << "* "; else out << source << ", ";
00025     if (group.isUnspecified()) out << "* "; else out << group << "), ";
00026     if (RP.isUnspecified()) out << "0.0.0.0" << endl; else out << "RP is " << RP
    << endl;
00027     out << "Incoming interface: ";
00028     if (inInt.intPtr != NULL)
00029     {
00030         if (inInt.intPtr) out << inInt.intPtr->getName() << ", ";
00031         out << "RPF neighbor " << inInt.nextHop << endl;
00032         out << "Outgoing interface list:" << endl;
00033     }
00034
00035     for (InterfaceVector::const_iterator i = outInt.begin(); i < outInt.end();
    i++)
00036     {
00037         if ((*i).intPtr) out << (*i).intPtr->getName() << ", ";
00038         if (i->forwarding == Forward) out << "Forward/"; else out << "Pruned/";
00039         if (i->mode == Densemode) out << "Dense"; else out << "Sparse";
00040         out << endl;
00041     }
00042
00043     return out.str();
00044 }
00045
00046 string MulticastIPRoute::detailedInfo() const
00047 {
00048     return string();
00049 }
00050
00051
00052 bool MulticastIPRoute::isFlagSet(flag fl)
00053 {
00054     for (unsigned int i = 0; i < flags.size(); i++)
00055     {
00056         if (flags[i] == fl)
00057             return true;
00058     }
00059     return false;
00060 }
00061
00062 void MulticastIPRoute::addFlag(flag fl)
00063 {
00064     if (!isFlagSet(fl))
00065         flags.push_back(fl);
00066 }
00067
00068 void MulticastIPRoute::removeFlag(flag fl)
00069 {
00070     for (unsigned int i = 0; i < flags.size(); i++)
00071     {
00072         if (flags[i] == fl)
00073         {
00074             flags.erase(flags.begin() + i);
00075             return;
00076         }
00077     }
00078 }
00079
00080 int MulticastIPRoute::getOutIdByIntId(int intId)
00081 {
00082     unsigned int i;
00083     for (i = 0; i < outInt.size(); i++)
00084     {
00085         if (outInt[i].intId == intId)
00086             break;
00087     }
00088     return i;
00089 }
00090
00091 bool MulticastIPRoute::isOlistNull()
00092 {
00093     bool olistNull = true;
00094     for (unsigned int i = 0; i < outInt.size(); i++)
00095     {
00096         if (outInt[i].forwarding == Forward)
00097         {
00098             olistNull = false;
00099             break;
00100         }
00101     }
00102     return olistNull;

```

```
00103 }  
00104
```

7.7 F:/ANSA/src/ansa/multicastRoutingTable/MulticastIPRoute.h File Reference

File contains implementation of multicast route.

```
#include <omnetpp.h>  
#include "IPAddress.h"  
#include "InterfaceEntry.h"  
#include "PIMTimer_m.h"
```

Classes

- struct [inInterface](#)
Structure of incoming interface. [More...](#)
- struct [outInterface](#)
Structure of outgoing interface. [More...](#)
- class [MulticastIPRoute](#)
Class represents one entry of [MulticastRoutingTable](#).

Typedefs

- typedef std::vector< [outInterface](#) > [InterfaceVector](#)

Enumerations

- enum [flag](#) {
 [D](#), [S](#), [C](#), [P](#),
 [A](#) }
- enum [intState](#) { [Densemode](#) = 1, [Sparsemode](#) = 2, [Forward](#), [Pruned](#) }
- enum [AssertState](#) { [NoInfo](#) = 0, [Winner](#) = 1, [Loser](#) = 2 }

7.7.1 Detailed Description

File contains implementation of multicast route.

Author

Veronika Rybova

Date

10.10.2011

Definition in file [MulticastIPRoute.h](#).

7.7.2 Class Documentation

7.7.2.1 struct inInterface

Structure of incoming interface.

E.g.: GigabitEthernet1/4, RPF nbr 10.10.51.145

Definition at line 53 of file [MulticastIPRoute.h](#).

Class Members

InterfaceEntry *	intPtr	Pointer to interface
int	intId	Interface ID
IPAddress	nextHop	RF neighbor

7.7.2.2 struct outInterface

Structure of outgoing interface.

E.g.: Ethernet0, Forward/Sparse, 5:29:15/0:02:57

Definition at line 64 of file [MulticastIPRoute.h](#).

Class Members

InterfaceEntry *	intPtr	Pointer to interface
int	intId	Interface ID
intState	forwarding	Forward or Pruned
intState	mode	Dense, Sparse, ...
PIMpt *	pruneTimer	Pointer to PIM Prune Timer
AssertState	assert	Assert state.

7.7.3 Typedef Documentation

7.7.3.1 typedef std::vector<outInterface> InterfaceVector

Vector of outgoing interfaces.

Definition at line 77 of file [MulticastIPRoute.h](#).

7.7.4 Enumeration Type Documentation

7.7.4.1 enum flag

Route flags. Added to each route.

Enumerator:

- D** Dense
- S** Sparse
- C** Connected
- P** Pruned
- A** Source is directly connected

Definition at line 19 of file [MulticastIPRoute.h](#).

```
{
    D,
    S,
    C,
    P,
    A
};
```

7.7.4.2 enum intState

States of each outgoing interface. E.g.: Forward/Dense.

Definition at line 31 of file [MulticastIPRoute.h](#).

```
{
    Densemode = 1,
    Sparsemode = 2,
    Forward,
    Pruned
};
```

7.7.4.3 enum AssertState

Assert States of each outgoing interface.

Definition at line 42 of file [MulticastIPRoute.h](#).

```
{
    NoInfo = 0,
    Winner = 1,
    Loser = 2
};
```

7.8 MulticastIPRoute.h

```
00001
00008 #ifndef MULTICASTIPROUTE_H_
00009 #define MULTICASTIPROUTE_H_
00010
00011 #include <omnetpp.h>
00012 #include "IPAddress.h"
00013 #include "InterfaceEntry.h"
00014 #include "PIMTimer_m.h"
00015
00019 enum flag
00020 {
00021     D,
00022     S,
00023     C,
00024     P,
00025     A
00026 };
00027
00031 enum intState
00032 {
00033     Densemode = 1,
00034     Sparsemode = 2,
00035     Forward,
00036     Pruned
00037 };
00038
00042 enum AssertState
00043 {
00044     NoInfo = 0,
00045     Winner = 1,
00046     Loser = 2
00047 };
00048
00053 struct inInterface
00054 {
00055     InterfaceEntry *intPtr;
00056     int intId;
```

```

00057         IPAddress                                     nextHop;
00058     };
00059
00064     struct outInterface
00065     {
00066         InterfaceEntry                                     *intPtr;
00067         int                                             intId;
00068         intState                                       forwarding;
00069         intState                                       mode;
00070         PIMpt                                           *pruneTimer;
00071         AssertState                                   assert;
00072     };
00073
00077     typedef std::vector<outInterface> InterfaceVector;
00078
00079
00083     class INET_API MulticastIPRoute : public cPolymorphic
00084     {
00085     private:
00086         IPAddress                                     source;
00087         IPAddress                                     group;
00088         IPAddress                                     RP;
00089         std::vector<flag>                             flags;
00090         // timers
00091         PIMgrt                                           *grt;
00092         PIMsat                                           *sat;
00093         PIMsrt                                           *srt;
00094         // interfaces
00095         inInterface                                       inInt;
00096         InterfaceVector                                   outInt;
00100         //Originated from destination.Ensures loop freeness.
00101         unsigned int sequencenumber;
00102         //Time of routing table entry creation
00103         simtime_t installtime;
00104
00105     private:
00107         MulticastIPRoute& operator=(const MulticastIPRoute& obj);
00108
00109     public:
00110         MulticastIPRoute();
00111         virtual ~MulticastIPRoute() {}
00112         virtual std::string info() const;
00113         virtual std::string detailedInfo() const;
00114
00115         void setSource(IPAddress source) {this->source = source;}
00116         void setGroup(IPAddress group) {this->group = group;}
00117         void setRP(IPAddress RP) {this->RP = RP;}
00118         void setGrt (PIMgrt *grt) {this->grt = grt;}
00119         void setSat (PIMsat *sat) {this->srt = sat;}
00120         void setSrt (PIMsrt *srt) {this->srt = srt;}
00122         void setFlags(std::vector<flag> flags) {this->flags = flags;}
00123         bool isFlagSet(flag fl);
00124         void addFlag(flag fl);
00125
00125         void removeFlag(flag fl);
00127         void setInInt(InterfaceEntry *interfacePtr, int intId, IPAddress nextHop) {
00128             this->inInt.intPtr = interfacePtr; this->inInt.intId = intId; this->inInt.
00129             nextHop = nextHop;}
00128         void setInInt(inInterface inInt) {this->inInt = inInt;}
00130         void setOutInt(InterfaceVector outInt) {EV << "MulticastIPRoute: New OutInt
00131         " << endl; this->outInt = outInt;}
00131         void addOutInt(outInterface outInt) {this->outInt.push_back(outInt);}
00133         bool isRpf(int intId){if (intId == inInt.intId) return true; else return
00134         false;}
00134         bool isOlistNull();
00137         IPAddress getSource() const {return source;}
00138         IPAddress getGroup() const {return group;}
00139         IPAddress getRP() const {return RP;}
00140         PIMgrt* getGrt() const {return grt;}
00141         PIMsat* getSat() const {return sat;}
00142         PIMsrt* getSrt() const {return srt;}
00143         std::vector<flag> getFlags() const {return flags;}
00145         // get incoming interface
00146         inInterface getInInt() const {return inInt;}
00147         InterfaceEntry* getInIntPtr() const {return inInt.intPtr;}
00148         int getInIntId() const {return inInt.intId;}
00149         IPAddress getInIntNextHop() const {return inInt.nextHop;}
00151         // get outgoing interface
00152         InterfaceVector getOutInt() const {return outInt;}
00153         outInterface getOutIntByIntId(int intId);
00154         int getOutIdByIntId(int intId);
00157         simtime_t getInstallTime() const {return installtime;}
00158         void setInstallTime(simtime_t time) {installtime = time;}
00159         void setSequencenumber(int i){sequencenumber =i;}
00160         unsigned int getSequencenumber() const {return sequencenumber;}
00161

```

```

00162 };
00163
00164
00165 #endif /* MULTICASTIPROUTE_H_ */

```

7.9 F:/ANSA/src/ansa/multicastRoutingTable/MulticastRoutingTable.cc File Reference

File contains implementation of multicast routing table.

```
#include "MulticastRoutingTable.h"
```

Functions

- **Define_Module** ([MulticastRoutingTable](#))
- `std::ostream & operator<<` (`std::ostream &os`, `const MulticastIPRoute &e`)

7.9.1 Detailed Description

File contains implementation of multicast routing table.

Date

10.3.2012

Author

Haczek

Definition in file [MulticastRoutingTable.cc](#).

7.9.2 Function Documentation

7.9.2.1 `std::ostream& operator<<` (`std::ostream &os`, `const MulticastIPRoute &e`)

Printout of structure [MulticastIPRoute](#) (one route in table).

Definition at line 15 of file [MulticastRoutingTable.cc](#).

```

{
    return os;
};

```

7.10 MulticastRoutingTable.cc

```

00001
00008 #include "MulticastRoutingTable.h"
00009
00010 Define_Module (MulticastRoutingTable);
00011
00012 using namespace std;
00013
00015 std::ostream& operator<<(std::ostream& os, const MulticastIPRoute& e)
00016 {
00017     return os;
00018 };
00019
00025 MulticastRoutingTable::~MulticastRoutingTable()
00026 {
00027     for (unsigned int i=0; i<multicastRoutes.size(); i++)

```

```

00028         delete multicastRoutes[i];
00029     }
00030
00038 void MulticastRoutingTable::initialize(int stage)
00039 {
00040     if (stage==0)
00041     {
00042         // get a pointer to IInterfaceTable
00043         ift = AnsaInterfaceTableAccess().get();
00044
00045         // watch multicast table
00046         //WATCH_PTRVECTOR(multicastRoutes);
00047         WATCH_VECTOR(showMRoute);
00048     }
00049 }
00050
00056 void MulticastRoutingTable::updateDisplayString()
00057 {
00058     if (!ev.isGUI())
00059         return;
00060
00061     char buf[80];
00062     sprintf(buf, "%d routes", multicastRoutes.size());
00063     getDisplayString().setTagArg("t",0,buf);
00064 }
00065
00071 void MulticastRoutingTable::printRoutingTable() const
00072 {
00073     EV << "-- Multicast routing table --\n";
00074     for (int i=0; i<getNumRoutes(); i++)
00075         EV << getRoute(i)->detailedInfo() << "\n";
00076     EV << "\n";
00077 }
00078
00084 void MulticastRoutingTable::handleMessage(cMessage *msg)
00085 {
00086     opp_error("This module doesn't process messages");
00087 }
00088
00104 const MulticastIPRoute *MulticastRoutingTable::findRoute(const IPAddress&
source, const IPAddress& group,
00105 const IPAddress& RP, int intId, const IPAddress& nextHop) const
00106 {
00107     int n = getNumRoutes();
00108     for (int i=0; i<n; i++)
00109         if (routeMatches(getRoute(i), source, group, RP, intId, nextHop))
00110             return getRoute(i);
00111     return NULL;
00112 }
00113
00119 int MulticastRoutingTable::getNumRoutes() const
00120 {
00121     return multicastRoutes.size();
00122 }
00123
00131 MulticastIPRoute *MulticastRoutingTable::getRoute(int k) const
00132 {
00133     if (k < (int)multicastRoutes.size())
00134         return multicastRoutes[k];
00135
00136     return NULL;
00137 }
00138
00152 bool MulticastRoutingTable::routeMatches(const MulticastIPRoute *entry,
00153 const IPAddress& source, const IPAddress& group,
00154 const IPAddress& RP, int intId, const IPAddress& nextHop) const
00155 {
00156     if (!source.isUnspecified() && !source.equals(entry->getSource()))
00157         return false;
00158     if (!group.isUnspecified() && !group.equals(entry->getGroup()))
00159         return false;
00160     if (!RP.isUnspecified() && !RP.equals(entry->getRP()))
00161         return false;
00162     if (intId!=entry->getIntId())
00163         return false;
00164     if (!nextHop.isUnspecified() && !nextHop.equals(entry->getIntNextHop()))
00165         return false;
00166     return true;
00167 }
00168
00178 vector<MulticastIPRoute> MulticastRoutingTable::getRouteFor(IPAddress group)
00179 {
00180     Enter_Method("getMulticastRoutesFor(%x)", group.getInt()); // note:
str().c_str() too slow here here
00181     EV << "MulticastRoutingTable::getRouteFor - address = " << group << endl;
00182     vector<MulticastIPRoute> routes;
00183 }

```

```

00184     // search in multicast table
00185     int n = multicastRoutes.size();
00186     for (int i = 0; i < n; i++)
00187     {
00188         MulticastIPRoute *route = getRoute(i);
00189         if (route->getGroup().getInt() == group.getInt())
00190             routes.push_back(route);
00191     }
00192
00193     return routes;
00194 }
00195
00196
00207 MulticastIPRoute *MulticastRoutingTable::getRouteFor(IPAddress group, IPAddress
    source)
00208 {
00209     Enter_Method("getMulticastRoutesFor(%x, %x)", group.getInt(), source.getInt
    ()); // note: str().c_str() too slow here here
00210     EV << "MulticastRoutingTable::getRouteFor - group = " << group << ", source
    = " << source << endl;
00211
00212     // search in multicast routing table
00213     MulticastIPRoute *route = NULL;
00214     int n = multicastRoutes.size();
00215     int i;
00216     // go through all multicast routes
00217     for (i = 0; i < n; i++)
00218     {
00219         route = getRoute(i);
00220         if (route->getGroup().getInt() == group.getInt() && route->getSource().
    getInt() == source.getInt())
00221             break;
00222     }
00223
00224     if (i == n)
00225         return NULL;
00226     return route;
00227 }
00228
00238 std::vector<MulticastIPRoute*> MulticastRoutingTable::getRoutesForSource(
    IPAddress source)
00239 {
00240     Enter_Method("getRoutesForSource(%x)", source.getInt()); // note:
    str().c_str() too slow here here
00241     EV << "MulticastRoutingTable::getRoutesForSource - source = " << source
    << endl;
00242     vector<MulticastIPRoute*> routes;
00243
00244     // search in multicast table
00245     int n = multicastRoutes.size();
00246     int i;
00247     for (i = 0; i < n; i++)
00248     {
00249         //FIXME works only for classfull addresses (function getNetwork)
00250
00251         MulticastIPRoute *route = getRoute(i);
00252         if (route->getSource().getNetwork().getInt() == source.getInt())
00253             routes.push_back(route);
00254     }
00255     return routes;
00256 }
00267 void MulticastRoutingTable::addRoute(const MulticastIPRoute *entry)
00268 {
00269     Enter_Method("addMulticastRoute(...)");
00270
00271     // check for null multicast group address
00272     if (entry->getGroup().isUnspecified())
00273         error("addMulticastRoute(): multicast group address cannot be NULL");
00274
00275     // check that group address is multicast address
00276     if (!entry->getGroup().isMulticast())
00277         error("addMulticastRoute(): group address is not multicast
    address");
00278
00279     // check for source or RP address
00280     if (entry->getSource().isUnspecified() && entry->getRP().isUnspecified())
00281         error("addMulticastRoute(): source or RP address has to be specified");
00282
00283     // check that the incoming interface exists
00284     if (!entry->getInIntPtr() || entry->getInIntNextHop().isUnspecified())
00285         error("addMulticastRoute(): incoming interface has to be specified");
00286
00287     // add to tables
00288     multicastRoutes.push_back(const_cast<MulticastIPRoute*>(entry));
00289

```



```

00290     updateDisplayString();
00291     generateShowIPRoute();
00292 }
00293
00305 bool MulticastRoutingTable::deleteRoute(const MulticastIPRoute *entry)
00306 {
00307     Enter_Method("deleteMulticastRoute(...)");
00308
00309     // find entry in routing table
00310     vector<MulticastIPRoute*>::iterator i;
00311     for (i=multicastRoutes.begin(); i!=multicastRoutes.end(); ++i)
00312     {
00313         if ((*i) == entry)
00314             break;
00315     }
00316
00317     // if entry was found, it can be deleted
00318     if (i!=multicastRoutes.end())
00319     {
00320         // first delete all timers assigned to route
00321         if (entry->getSrt() != NULL)
00322         {
00323             cancelEvent(entry->getSrt());
00324             delete entry->getSrt();
00325         }
00326         if (entry->getGrt() != NULL)
00327         {
00328             cancelEvent(entry->getGrt());
00329             delete entry->getGrt();
00330         }
00331         if (entry->getSat())
00332         {
00333             cancelEvent(entry->getSat());
00334             delete entry->getSat();
00335         }
00336
00337         // delete timers from outgoing interfaces
00338         InterfaceVector outInt = entry->getOutInt();
00339         for (unsigned int j = 0; j < outInt.size(); j++)
00340         {
00341             if (outInt[j].pruneTimer != NULL)
00342             {
00343                 cancelEvent(outInt[j].pruneTimer);
00344                 delete outInt[j].pruneTimer;
00345             }
00346         }
00347
00348         // delete route
00349         multicastRoutes.erase(i);
00350         delete entry;
00351         updateDisplayString();
00352         generateShowIPRoute();
00353         return true;
00354     }
00355     return false;
00356 }
00357
00364 void MulticastRoutingTable::generateShowIPRoute()
00365 {
00366     EV << "MulticastRoutingTable::generateShowIPRoute()" << endl;
00367     showMRoute.clear();
00368
00369     int n = getNumRoutes();
00370     const MulticastIPRoute* ipr;
00371
00372     for (int i=0; i<n; i++)
00373     {
00374         ipr = getRoute(i);
00375         stringstream os;
00376         os << " ";
00377         if (ipr->getSource().isUnspecified()) os << "*", "; else os <<
ipr->getSource() << " ";
00378         os << ipr->getGroup() << " ";
00379         if (!ipr->getRP().isUnspecified()) os << "RP is " << ipr->getRP
() << " ";
00380         os << "flags: ";
00381         vector<flag> flags = ipr->getFlags();
00382         for (unsigned int j = 0; j < flags.size(); j++)
00383         {
00384             EV << "MulticastRoutingTable::generateShowIPRoute():"
Flag = " << flags[j] << endl;
00385             switch(flags[j])
00386             {
00387                 case D:
00388                     os << "D";
00389                     break;
00390                 case S:

```

```

00391                                     os << "S";
00392                                     break;
00393             case C:
00394                 os << "C";
00395                 break;
00396             case P:
00397                 os << "P";
00398                 break;
00399             case A:
00400                 os << "A";
00401                 break;
00402         }
00403     }
00404     os << endl;
00405
00406     os << "Incoming interface: ";
00407     if (ipr->getInIntPtr()) os << ipr->getInIntPtr()->getName() <<
", ";
00408     os << "RPF neighbor " << ipr->getInIntNextHop() << endl;
00409     os << "Outgoing interface list:" << endl;
00410
00411     InterfaceVector all = ipr->getOutInt();
00412     if (all.size() == 0)
00413         os << "Null" << endl;
00414     else
00415         for (unsigned int k = 0; k < all.size(); k++)
00416         {
00417             os << all[k].intPtr->getName() << ", ";
00418             if (all[k].forwarding == Forward) os << "
Forward/"; else os << "Pruned/";
00419             if (all[k].mode == Densemode) os << "Dense";
00420             else os << "Sparse";
00421             os << endl;
00422         }
00423     showMRoute.push_back(os.str());
00424     stringstream out;
00425 }

```

7.11 F:/ANSA/src/ansa/multicastRoutingTable/MulticastRoutingTable.h File Reference

File contains implementation of multicast routing table.

```

#include <omnetpp.h>
#include "MulticastIPRoute.h"
#include "IPAddress.h"
#include "NotificationBoard.h"
#include "AnsaInterfaceTableAccess.h"
#include "AnsaInterfaceTable.h"

```

Classes

- class [MulticastRoutingTable](#)
Class represent multicast routing table.

Typedefs

- typedef std::vector
< [MulticastIPRoute](#) * > [RouteVector](#)

7.11.1 Detailed Description

File contains implementation of multicast routing table.

Date

10.3.2012

Author

Haczek

Definition in file [MulticastRoutingTable.h](#).

7.11.2 Typedef Documentation

7.11.2.1 typedef std::vector<MulticastIPRoute *> RouteVector

RouteVector represents multicast table. It is list of multicast routes.

Definition at line 21 of file [MulticastRoutingTable.h](#).

7.12 MulticastRoutingTable.h

```

00001
00008 #ifndef __MULTICASTROUTINGTABLE_H
00009 #define __MULTICASTROUTINGTABLE_H
00010
00011 #include <omnetpp.h>
00012 #include "MulticastIPRoute.h"
00013 #include "IPAddress.h"
00014 #include "NotificationBoard.h"
00015 #include "AnsaInterfaceTableAccess.h"
00016 #include "AnsaInterfaceTable.h"
00017
00021 typedef std::vector<MulticastIPRoute *> RouteVector;
00022
00029 class INET_API MulticastRoutingTable: public cSimpleModule
00030 {
00031     protected:
00032         RouteVector multicastRoutes;
00033         std::vector<std::string> showMRoute;
00034         IInterfaceTable *ift;
00036     protected:
00037         virtual bool routeMatches(const MulticastIPRoute *entry,
00038             const IPAddress& source, const IPAddress& group,
00039             const IPAddress& RP, int intId, const IPAddress& nextHop)
00040     const;
00041         virtual void updateDisplayString();
00042
00043     public:
00044         // print multicast routing table
00045         void generateShowIPMRoute();
00046         virtual void printRoutingTable() const;
00047
00048         // Returns routes for a multicast address.
00049         virtual std::vector<MulticastIPRoute*> getRouteFor(IPAddress
group);
00050         virtual MulticastIPRoute *getRouteFor(IPAddress group,
IPAddress source);
00051         virtual std::vector<MulticastIPRoute*> getRoutesForSource(
IPAddress source);
00052
00053         // for manipulation with the table
00054         virtual int getNumRoutes() const;
00055         virtual MulticastIPRoute *getRoute(int k) const;
00056         virtual const MulticastIPRoute *findRoute(const IPAddress&
source, const IPAddress& group,
00057             const IPAddress& RP, int intId, const IPAddress
& nextHop) const;
00058
00059         // the most important !!!
00060         virtual void addRoute(const MulticastIPRoute *entry);
00061         virtual bool deleteRoute(const MulticastIPRoute *entry);
00062
00063     public:
00064         MulticastRoutingTable();
00065         virtual ~MulticastRoutingTable();

```

```

00066
00067     protected:
00068         virtual int numInitStages() const {return 4;}
00069         virtual void initialize(int stage);
00070         virtual void handleMessage(cMessage *);
00071 };
00072
00073
00074 #endif /* MULTICASTROUTINGTABLE_H_ */

```

7.13 F:/ANSA/src/ansa/multicastRoutingTable/MulticastRoutingTableAccess.h File Reference

File contains implementation of access class.

```

#include <omnetpp.h>
#include "ModuleAccess.h"
#include "MulticastRoutingTable.h"

```

Classes

- class [MulticastRoutingTableAccess](#)
Class gives access to the [MulticastRoutingTable](#).

7.13.1 Detailed Description

File contains implementation of access class. 17.3.2012

Author

Veronika Rybova

Definition in file [MulticastRoutingTableAccess.h](#).

7.14 MulticastRoutingTableAccess.h

```

00001
00008 #ifndef MULTICASTROUTINGTABLEACCESS_H_
00009 #define MULTICASTROUTINGTABLEACCESS_H_
00010
00011 #include <omnetpp.h>
00012 #include "ModuleAccess.h"
00013 #include "MulticastRoutingTable.h"
00014
00018 class INET_API MulticastRoutingTableAccess : public ModuleAccess<MulticastRoutingTable>
00019 {
00020     public:
00021         MulticastRoutingTableAccess() :
00022             ModuleAccess<MulticastRoutingTable>("multicastRoutingTable") {}
00023 };
00024 #endif /* MULTICASTROUTINGTABLEACCESS_H_ */

```

7.15 F:/ANSA/src/ansa/pim/modes/pimDM.cc File Reference

File implements PIM dense mode.

```

#include "pimDM.h"

```

Functions

- **Define_Module** ([pimDM](#))

7.15.1 Detailed Description

File implements PIM dense mode.

Date

29.10.2011

Author

: Veronika Rybova

Implementation according to RFC3973.

Definition in file [pimDM.cc](#).

7.16 pimDM.cc

```

00001
00009 #include "pimDM.h"
00010
00011
00012 Define_Module(pimDM);
00013
00014 using namespace std;
00015
00027 void pimDM::sendPimJoinPrune(IPAddress nextHop, IPAddress src, IPAddress grp,
    int intId)
00028 {
00029     EV << "pimDM::sendPimJoinPrune" << endl;
00030     EV << "UpstreamNeighborAddress: " << nextHop << ", Source: " << src <<
    ", Group: " << grp << ", IntId: " << intId << endl;
00031
00032     PIMJoinPrune *msg = new PIMJoinPrune();
00033     msg->setName("PIMJoinPrune");
00034     msg->setUpstreamNeighborAddress(nextHop);
00035     msg->setHoldTime(PT);
00036     msg->setMulticastGroupsArraySize(1);
00037
00038     //FIXME change to add also join groups
00039     // we do not need it at this time
00040
00041     // set multicast groups
00042     MulticastGroup *group = new MulticastGroup();
00043     group->setGroupAddress(grp);
00044     group->setJoinedSourceAddressArraySize(0);
00045     group->setPrunedSourceAddressArraySize(1);
00046     group->setPrunedSourceAddress(0, src);
00047     msg->setMulticastGroups(0, *group);
00048
00049     // set IP Control info
00050     IPControlInfo *ctrl = new IPControlInfo();
00051     IPAddress gal("224.0.0.13");
00052     ctrl->setDestAddr(gal);
00053     //ctrl->setProtocol(IP_PROT_PIM);
00054     ctrl->setProtocol(103);
00055     ctrl->setTimeToLive(1);
00056     ctrl->setInterfaceId(intId);
00057     msg->setControlInfo(ctrl);
00058     send(msg, "spiltterOut");
00059 }
00060
00071 void pimDM::sendPimGraftAck(PIMGraftAck *msg)
00072 {
00073     msg->setName("PIMGraftAck");
00074     msg->setType(GraftAck);
00075
00076     // set IP Control info
00077     IPControlInfo *oldCtrl = (IPControlInfo*) (msg->removeControlInfo());
00078     IPControlInfo *ctrl = new IPControlInfo();

```

```

00079         ctrl->setDestAddr(oldCtrl->getSrcAddr());
00080         ctrl->setSrcAddr(oldCtrl->getDestAddr());
00081         ctrl->setProtocol(103);
00082         ctrl->setTimeToLive(1);
00083         ctrl->setInterfaceId(oldCtrl->getInterfaceId());
00084         delete oldCtrl;
00085         msg->setControlInfo(ctrl);
00086         send(msg, "spiltterOut");
00087     }
00088
00102 void pimDM::sendPimGraft(IPAddress nextHop, IPAddress src, IPAddress grp, int
intId)
00103 {
00104     EV << "pimDM::sendPimGraft" << endl;
00105     EV << "UpstreamNeighborAddress: " << nextHop << ", Source: " << src <<
", Group: " << grp << ", IntId: " << intId << endl;
00106
00107     PIMGraft *msg = new PIMGraft();
00108     msg->setName("PIMGraft");
00109     msg->setHoldTime(0);
00110     msg->setUpstreamNeighborAddress(nextHop);
00111     msg->setMulticastGroupsArraySize(1);
00112
00113     // set multicast groups
00114     MulticastGroup *group = new MulticastGroup();
00115     group->setGroupAddress(grp);
00116     group->setJoinedSourceAddressArraySize(1);
00117     group->setPrunedSourceAddressArraySize(0);
00118     group->setJoinedSourceAddress(0, src);
00119     msg->setMulticastGroups(0, *group);
00120
00121     // set IP Control info
00122     IPControlInfo *ctrl = new IPControlInfo();
00123     ctrl->setDestAddr(nextHop);
00124     //ctrl->setProtocol(IP_PROT_PIM);
00125     ctrl->setProtocol(103);
00126     ctrl->setTimeToLive(1);
00127     ctrl->setInterfaceId(intId);
00128     msg->setControlInfo(ctrl);
00129     send(msg, "spiltterOut");
00130 }
00131
00145 void pimDM::sendPimStateRefresh(IPAddress originator, IPAddress src, IPAddress
grp, int intId, bool P)
00146 {
00147     EV << "pimDM::sendPimStateRefresh" << endl;
00148
00149     PIMStateRefresh *msg = new PIMStateRefresh();
00150     msg->setName("PIMStateRefresh");
00151     msg->setGroupAddress(grp);
00152     msg->setSourceAddress(src);
00153     msg->setOriginatorAddress(originator);
00154     msg->setInterval(SRT);
00155     msg->setP(P);
00156
00157     // set IP Control info
00158     IPControlInfo *ctrl = new IPControlInfo();
00159     ctrl->setDestAddr(grp);
00160     //ctrl->setProtocol(IP_PROT_PIM);
00161     ctrl->setProtocol(103);
00162     ctrl->setTimeToLive(1);
00163     ctrl->setInterfaceId(intId);
00164     msg->setControlInfo(ctrl);
00165     send(msg, "spiltterOut");
00166 }
00167
00182 PIMpt* pimDM::createPruneTimer(IPAddress source, IPAddress group, int intId,
int holdTime)
00183 {
00184     EV << "pimDM::createPruneTimer" << endl;
00185     PIMpt *timer = new PIMpt();
00186     timer->setName("PimPruneTimer");
00187     timer->setSource(source);
00188     timer->setGroup(group);
00189     timer->setIntId(intId);
00190     scheduleAt(simTime() + holdTime, timer);
00191     return timer;
00192 }
00193
00207 PIMgrt* pimDM::createGraftRetryTimer(IPAddress source, IPAddress group)
00208 {
00209     EV << "pimDM::createPruneTimer" << endl;
00210     PIMgrt *timer = new PIMgrt();
00211     timer->setName("PIMGraftRetryTimer");
00212     timer->setSource(source);
00213     timer->setGroup(group);
00214     scheduleAt(simTime() + GRT, timer);

```

```

00215         return timer;
00216     }
00217
00230 PIMsat* pimDM::createSourceActiveTimer(IPAddress source, IPAddress group)
00231 {
00232     EV << "pimDM::createSourceActiveTimer" << endl;
00233     PIMsat *timer = new PIMsat();
00234     timer->setName("PIMSourceActiveTimer");
00235     timer->setSource(source);
00236     timer->setGroup(group);
00237     scheduleAt(simTime() + SAT, timer);
00238     return timer;
00239 }
00240
00253 PIMsrt* pimDM::createStateRefreshTimer(IPAddress source, IPAddress group)
00254 {
00255     EV << "pimDM::createStateRefreshTimer" << endl;
00256     PIMsrt *timer = new PIMsrt();
00257     timer->setName("PIMStateRefreshTimer");
00258     timer->setSource(source);
00259     timer->setGroup(group);
00260     scheduleAt(simTime() + SRT, timer);
00261     return timer;
00262 }
00263
00264
00283 void pimDM::processGraftPacket(IPAddress source, IPAddress group, IPAddress
sender, int intId)
00284 {
00285     EV << "pimDM::processGraftPacket" << endl;
00286
00287     MulticastIPRoute *route = mrt->getRouteFor(group, source);
00288     bool forward = false;
00289
00290     // check if message come to non-RPF interface
00291     if (route->isRpf(intId))
00292     {
00293         EV << "ERROR: Graft message came to RPF interface." << endl;
00294         return;
00295     }
00296
00297     // find outgoing interface to neighbor
00298     InterfaceVector outInt = route->getOutInt();
00299     for (unsigned int l = 0; l < outInt.size(); l++)
00300     {
00301         if(outInt[l].intId == intId)
00302         {
00303             forward = true;
00304             if (outInt[l].forwarding == Pruned)
00305             {
00306                 EV << "Interface " << outInt[l].intId << "
transit to forwarding state (Graft)." << endl;
00307                 outInt[l].forwarding = Forward;
00308
00309                 //cancel Prune Timer
00310                 PIMpt* timer = outInt[l].pruneTimer;
00311                 cancelEvent(timer);
00312                 delete timer;
00313                 outInt[l].pruneTimer = NULL;
00314             }
00315         }
00316     }
00317     route->setOutInt(outInt);
00318
00319     // if all route was pruned, remove prune flag
00320     // if upstream is not source, send Graft message
00321     if (route->isFlagSet(P) && forward && (route->getGrt() == NULL))
00322     {
00323         if (!route->isFlagSet(A))
00324         {
00325             EV << "Route is not pruned any more, send Graft to
upstream" << endl;
00326             sendPimGraft(route->getInIntNextHop(), source, group,
route->getInIntId());
00327             PIMgrt* timer = createGraftRetryTimer(source, group);
00328             route->setGrt(timer);
00329         }
00330         else
00331             route->removeFlag(P);
00332     }
00333 }
00334
00345 void pimDM::processGraftAckPacket(MulticastIPRoute *route)
00346 {
00347     EV << "pimDM::processGraftAckPacket" << endl;
00348     PIMgrt *grt = route->getGrt();
00349     if (grt != NULL)

```

```

00350     {
00351         cancelEvent(grt);
00352         delete grt;
00353         route->setGrt(NULL);
00354         route->removeFlag(P);
00355     }
00356 }
00357
00373 void pimDM::processPrunePacket(MulticastIPRoute *route, int intId, int holdTime)
00374 {
00375     EV << "pimDM::processPrunePacket" << endl;
00376     InterfaceVector outInt = route->getOutInt();
00377     int i = route->getOutIdByIntId(intId);
00378     bool change = false;
00379
00380     // we find correct outgoing interface
00381     if (i < (int) outInt.size())
00382     {
00383         // if interface was already pruned, restart Prune Timer
00384         if (outInt[i].forwarding == Pruned)
00385         {
00386             EV << "Outgoing interface is already pruned, restart
Prune Timer." << endl;
00387             PIMpt* timer = outInt[i].pruneTimer;
00388             cancelEvent(timer);
00389             scheduleAt(simTime() + holdTime, timer);
00390         }
00391         // if interface is forwarding, transit its state to pruned and
set Prune timer
00392         else
00393         {
00394             EV << "Outgoing interfaces is forwarding now -> change
to Pruned, set the timer." << endl;
00395             outInt[i].forwarding = Pruned;
00396             PIMpt* timer = createPruneTimer(route->getSource(),
route->getGroup(), intId, holdTime);
00397             outInt[i].pruneTimer = timer;
00398             change = true;
00399         }
00400     }
00401     route->setOutInt(outInt);
00402
00403     // if there is no forwarding outgoing int, transit route to pruned
state
00404     if (route->isOilistNull() && change)
00405     {
00406         EV << "All interfaces are pruned, send Pruned to upstream." <<
endl;
00407         route->addFlag(P);
00408
00409         // if GRT is running now, do not send Prune msg
00410         if (route->isFlagSet(P) && (route->getGrt() != NULL))
00411         {
00412             cancelEvent(route->getGrt());
00413             delete route->getGrt();
00414             route->setGrt(NULL);
00415         }
00416         else if (!route->isFlagSet(A))
00417             sendPimJoinPrune(route->getInIntNextHop(), route->
getSource(), route->getGroup(), route->getInIntId());
00418     }
00419 }
00420
00421
00438 void pimDM::processJoinPruneGraftPacket(PIMJoinPrune *pkt, PIMPacketType type)
00439 {
00440     EV << "pimDM::processJoinePruneGraftPacket" << endl;
00441
00442     IPControlInfo *ctrl = (IPControlInfo *) pkt->getControlInfo();
00443     IPAddress sender = ctrl->getSrcAddr();
00444     InterfaceEntry * nt = rt->getInterfaceForDestAddr(sender);
00445     vector<PimNeighbor> neighbors = pimNbt->getNeighborsByIntID(nt->
getInterfaceId());
00446     IPAddress addr = nt->ipv4Data()->getIPAddress();
00447
00448     // does packet belong to this router?
00449     if (pkt->getUpstreamNeighborAddress() != nt->ipv4Data()->getIPAddress()
&& type != GraftAck)
00450     {
00451         EV << "Paket neni urceny pro tento router" << endl;
00452         delete pkt;
00453         return;
00454     }
00455
00456     // go through list of multicast groups
00457     for (unsigned int i = 0; i < pkt->getMulticastGroupsArraySize(); i++)

```



```

00458     {
00459         MulticastGroup group = pkt->getMulticastGroups(i);
00460         IPAddress groupAddr = group.getGroupAddress();
00461
00462         // go through list of joined sources
00463         //EV << "JoinedSourceAddressArraySize: " <<
00464         group.getJoinedSourceAddressArraySize() << endl;
00465         for (unsigned int j = 0; j < group.
getJoinedSourceAddressArraySize(); j++)
00466         {
00467             IPAddress source = group.getJoinedSourceAddress(j);
00468             MulticastIPRoute *route = mrt->getRouteFor(groupAddr,
source);
00469
00470             if (type == JoinPrune)
00471             {
00472                 //FIXME join action
00473                 // only if there is more than one PIM neighbor
00474                 on one interface
00475                 // interface change to forwarding state
00476                 // cancel Prune Timer
00477                 // send Graft to upstream
00478             }
00479             else if (type == Graft)
00480                 processGraftPacket(source, groupAddr, sender,
nt->getInterfaceId());
00481             else if (type == GraftAck)
00482                 processGraftAckPacket(route);
00483
00484             // go through list of pruned sources (only for PIM Join Prune
msg)
00485             if (type == JoinPrune)
00486             {
00487                 //EV << "JoinedPrunedAddressArraySize: " <<
group.getPrunedSourceAddressArraySize() << endl;
00488                 for(unsigned int k = 0; k < group.
getPrunedSourceAddressArraySize(); k++)
00489                 {
00490                     IPAddress source = group.getPrunedSourceAddress
(k);
00491                     MulticastIPRoute *route = mrt->getRouteFor(
groupAddr, source);
00492                     if (source != route->getSource())
00493                         continue;
00494
00495                     // if there could be more than one PIM neighbor
00496                     on interface
00497                     if (neighbors.size() > 1)
00498                     {
00499                         EV << "Vice sousedu na rozhrani" <<
endl;
00500                         ; //FIXME set PPT timer
00501                     }
00502                     // if there is only one PIM neighbor on
interface
00503                     else
00504                         processPrunePacket(route, nt->
getInterfaceId(), pkt->getHoldTime());
00505                 }
00506             }
00507
00508             // Send GraftAck for this Graft message
00509             if (type == Graft)
00510                 sendPimGraftAck((PIMGraftAck *) (pkt));
00511             mrt->generateShowIPMroute();
00512 }
00513
00528 void pimDM::processStateRefreshPacket(PIMStateRefresh *pkt)
00529 {
00530     EV << "pimDM::processStateRefreshPacket" << endl;
00531
00532     // FIXME actions of upstream automat according to pruned/forwarding
state and Prune Indicator from msg
00533
00534     // first check if there is route for given group address and source
00535     MulticastIPRoute *route = mrt->getRouteFor(pkt->getGroupAddress(), pkt
->getSourceAddress());
00536     if (route == NULL)
00537     {
00538         delete pkt;
00539         return;
00540     }
00541     InterfaceVector outInt = route->getOutInt();
00542     bool pruneIndicator;

```

```

00543
00544 // check if State Refresh msg has came to RPF interface
00545 IPControlInfo *ctrl = (IPControlInfo*) pkt->getControlInfo();
00546 if (ctrl->getInterfaceId() != route->getInIntId())
00547 {
00548     delete pkt;
00549     return;
00550 }
00551
00552 // this router is pruned, but outgoing int of upstream router leading
to this router is forwarding
00553 if (route->isFlagSet(P) && !pkt->getP())
00554 {
00555     // send Prune msg to upstream
00556     if (route->getGrT() == NULL)
00557         sendPimJoinPrune(route->getInIntNextHop(), route->
getSource(), route->getGroup(), route->getInIntId());
00558     else
00559     {
00560         cancelEvent(route->getGrT());
00561         delete route->getGrT();
00562         route->setGrT(NULL);
00563     }
00564 }
00565
00566 // go through all outgoing interfaces, reser Prune Timer and send out
State Refresh msg
00567 for (unsigned int i = 0; i < outInt.size(); i++)
00568 {
00569     if (outInt[i].forwarding == Pruned)
00570     {
00571         // P = true
00572         pruneIndicator = true;
00573         // reset PT
00574         cancelEvent(outInt[i].pruneTimer);
00575         scheduleAt(simTime() + PT, outInt[i].pruneTimer);
00576     }
00577     else if (outInt[i].forwarding == Forward)
00578     {
00579         // P = false
00580         pruneIndicator = false;
00581     }
00582     sendPimStateRefresh(pkt->getOriginatorAddress(), pkt->
getSourceAddress(), pkt->getGroupAddress(), outInt[i].intId, pruneIndicator);
00583 }
00584 delete pkt;
00585 }
00586
00587
00600 void pimDM::processPruneTimer(PIMpt *timer)
00601 {
00602     EV << "pimDM::processPruneTimer" << endl;
00603
00604     IPAddress source = timer->getSource();
00605     IPAddress group = timer->getGroup();
00606     int intId = timer->getIntId();
00607
00608     // find correct (S,G) route which timer belongs to
00609     MulticastIPRoute *route = mrt->getRouteFor(group, source);
00610     if (route == NULL)
00611     {
00612         delete timer;
00613         return;
00614     }
00615
00616     // state of interface is changed to forwarding
00617     int i = route->getOutIdByIntId(intId);
00618     InterfaceVector outInt = route->getOutInt();
00619     if (i < (int) outInt.size())
00620     {
00621         EV << "Malezen out int" << endl;
00622         delete timer;
00623         outInt[i].pruneTimer = NULL;
00624         outInt[i].forwarding = Forward;
00625         route->setOutInt(outInt);
00626
00627         // if the router is pruned from multicast tree, join again
00628         if (route->isFlagSet(P) && (route->getGrT() == NULL))
00629         {
00630             if (!route->isFlagSet(A))
00631             {
00632                 EV << "Pruned cesta prejde do forwardu, posli
Graft" << endl;
00633                 sendPimGraft(route->getInIntNextHop(), source,
group, route->getInIntId());
00634                 PIMgrt* timer = createGraftRetryTimer(source,
group);

```

```

00635             route->setGrT(timer);
00636         }
00637         else
00638             route->removeFlag(P);
00639     }
00640     mrt->generateShowIPMroute();
00641 }
00642 }
00643
00656 void pimDM::processGraftRetryTimer(PIMgrt *timer)
00657 {
00658     EV << "pimDM::processGraftRetryTimer" << endl;
00659     MulticastIPRoute *route = mrt->getRouteFor(timer->getGroup(), timer->
getSource());
00660     sendPimGraft(route->getInIntNextHop(), timer->getSource(), timer->
getGroup(), route->getInIntId());
00661     timer = createGraftRetryTimer(timer->getSource(), timer->getGroup());
00662 }
00663
00673 void pimDM::processSourceActiveTimer(PIMsat * timer)
00674 {
00675     EV << "pimDM::processSourceActiveTimer: route will be deleted" << endl;
00676     MulticastIPRoute *route = mrt->getRouteFor(timer->getGroup(), timer->
getSource());
00677     delete timer;
00678     route->setSat(NULL);
00679     mrt->deleteRoute(route);
00680 }
00681 }
00682
00696 void pimDM::processStateRefreshTimer(PIMsrt * timer)
00697 {
00698     EV << "pimDM::processStateRefreshTimer" << endl;
00699     MulticastIPRoute *route = mrt->getRouteFor(timer->getGroup(), timer->
getSource());
00700     InterfaceVector outInt = route->getOutInt();
00701     bool pruneIndicator;
00702
00703     for (unsigned int i = 0; i < outInt.size(); i++)
00704     {
00705         if (outInt[i].forwarding == Pruned)
00706         {
00707             // P = true
00708             pruneIndicator = true;
00709             // reset PT
00710             cancelEvent(outInt[i].pruneTimer);
00711             scheduleAt(simTime() + PT, outInt[i].pruneTimer);
00712         }
00713         else if (outInt[i].forwarding == Forward)
00714         {
00715             pruneIndicator = false;
00716         }
00717         int intId = outInt[i].intId;
00718         sendPimStateRefresh(ift->getInterfaceById(intId)->ipv4Data()->
getIPAddress(), timer->getSource(), timer->getGroup(), intId, pruneIndicator);
00719     }
00720     delete timer;
00721     route->setSrt(createStateRefreshTimer(route->getSource(), route->
getGroup()));
00722 }
00723
00735 void pimDM::processPIMTimer(PIMTimer *timer)
00736 {
00737     EV << "pimDM::processPIMTimer: ";
00738
00739     switch(timer->getTimerKind())
00740     {
00741         case AssertTimer:
00742             EV << "AssertTimer" << endl;
00743             break;
00744         case PruneTimer:
00745             EV << "PruneTimer" << endl;
00746             processPruneTimer(check_and_cast<PIMpt *> (timer));
00747             break;
00748         case PrunePendingTimer:
00749             EV << "PrunePendingTimer" << endl;
00750             break;
00751         case GraftRetryTimer:
00752             EV << "GraftRetryTimer" << endl;
00753             processGraftRetryTimer(check_and_cast<PIMgrt *> (timer)
);
00754             break;
00755         case UpstreamOverrideTimer:
00756             EV << "UpstreamOverrideTimer" << endl;
00757             break;
00758         case PruneLimitTimer:
00759             EV << "PruneLimitTimer" << endl;

```

```

00760             break;
00761         case SourceActiveTimer:
00762             EV << "SourceActiveTimer" << endl;
00763             processSourceActiveTimer(check_and_cast<PIMsat *> (
timer));
00764             break;
00765         case StateRefreshTimer:
00766             EV << "StateRefreshTimer" << endl;
00767             processStateRefreshTimer(check_and_cast<PIMsrt *> (
timer));
00768             break;
00769         default:
00770             EV << "BAD TYPE, DROPPED" << endl;
00771             delete timer;
00772     }
00773 }
00774
00785 void pimDM::processPIMPkt (PIMPacket *pkt)
00786 {
00787     EV << "pimDM::processPIMPkt: ";
00788
00789     switch(pkt->getType())
00790     {
00791         case JoinPrune:
00792             EV << "JoinPrune" << endl;
00793             processJoinPruneGraftPacket(check_and_cast<PIMJoinPrune
*> (pkt), (PIMPacketType) pkt->getType());
00794             break;
00795         case Assert:
00796             EV << "Assert" << endl;
00797             // FIXME for future use
00798             break;
00799         case Graft:
00800             EV << "Graft" << endl;
00801             processJoinPruneGraftPacket(check_and_cast<PIMJoinPrune
*> (pkt), (PIMPacketType) pkt->getType());
00802             break;
00803         case GraftAck:
00804             EV << "GraftAck" << endl;
00805             processJoinPruneGraftPacket(check_and_cast<PIMJoinPrune
*> (pkt), (PIMPacketType) pkt->getType());
00806             break;
00807         case StateRefresh:
00808             EV << "StateRefresh" << endl;
00809             processStateRefreshPacket(
check_and_cast<PIMStateRefresh *> (pkt));
00810             break;
00811         default:
00812             EV << "BAD TYPE, DROPPED" << endl;
00813             delete pkt;
00814     }
00815 }
00816
00817
00831 void pimDM::handleMessage(cMessage *msg)
00832 {
00833     EV << "PIMDM::handleMessage" << endl;
00834
00835     // self message (timer)
00836     if (msg->isSelfMessage())
00837     {
00838         EV << "PIMDM::handleMessage:Timer" << endl;
00839         PIMTimer *timer = check_and_cast<PIMTimer *> (msg);
00840         processPIMTimer(timer);
00841     }
00842     // PIM packet from PIM neighbor
00843     else if (dynamic_cast<PIMPacket *>(msg))
00844     {
00845         EV << "PIMDM::handleMessage: PIM-DM packet" << endl;
00846         PIMPacket *pkt = check_and_cast<PIMPacket *>(msg);
00847         processPIMPkt(pkt);
00848     }
00849     // wrong message, mistake
00850     else
00851         EV << "PIMDM::handleMessage: Wrong message" << endl;
00852 }
00853
00863 void pimDM::initialize(int stage)
00864 {
00865     if (stage == 4)
00866     {
00867         EV << "pimDM::initialize" << endl;
00868
00869         // Pointer to routing tables, interface tables, notification
board
00870         rt = RoutingTableAccess().get();
00871         mrt = MulticastRoutingTableAccess().get();

```

```

00872         ift = AnsaInterfaceTableAccess().get();
00873         nb = NotificationBoardAccess().get();
00874         pimIft = PimInterfaceTableAccess().get();
00875         pimNbt = PimNeighborTableAccess().get();
00876
00877         // is PIM enabled?
00878         if (pimIft->getNumInterface() == 0)
00879         {
00880             EV << "PIM is NOT enabled on device " << endl;
00881             return;
00882         }
00883
00884         // subscribe for notifications
00885         nb->subscribe(this, NF_IPv4_NEW_MULTICAST_DENSE);
00886         nb->subscribe(this, NF_IPv4_NEW_IGMP_ADDED);
00887         nb->subscribe(this, NF_IPv4_NEW_IGMP_REMOVED);
00888         nb->subscribe(this, NF_IPv4_DATA_ON_PRUNED_INT);
00889         nb->subscribe(this, NF_IPv4_DATA_ON_NONRPF);
00890         nb->subscribe(this, NF_IPv4_DATA_ON_RPF);
00891         //nb->subscribe(this, NF_IPv4_RPF_CHANGE);
00892         nb->subscribe(this, NF_IPv4_ROUTE_ADDED);
00893     }
00894 }
00895
00906 void pimDM::receiveChangeNotification(int category, const cPolymorphic *details)
00907 {
00908     // ignore notifications during initialize
00909     if (simulation.getContextType() == CTX_INITIALIZE)
00910         return;
00911
00912     // PIM needs addition info for each notification
00913     if (details == NULL)
00914         return;
00915
00916     Enter_Method_Silent();
00917     printNotificationBanner(category, details);
00918     IPControlInfo *ctrl;
00919     MulticastIPRoute *route;
00920     addRemoveAddr *members;
00921
00922     // according to category of event...
00923     switch (category)
00924     {
00925         // new multicast data appears in router
00926         case NF_IPv4_NEW_MULTICAST_DENSE:
00927             EV << "pimDM::receiveChangeNotification - NEW
MULTICAST DENSE-" << endl;
00928             route = (MulticastIPRoute *) (details);
00929             newMulticast(route);
00930             break;
00931
00932         // configuration of interface changed, it means some change
00933         from IGMP, address were added.
00934         case NF_IPv4_NEW_IGMP_ADDED:
00935             EV << "pimDM::receiveChangeNotification - IGMP change -
address were added." << endl;
00936             members = (addRemoveAddr *) (details);
00937             newMulticastAddr(members);
00938             break;
00939
00940         // configuration of interface changed, it means some change
00941         from IGMP, address were removed.
00942         case NF_IPv4_NEW_IGMP_REMOVED:
00943             EV << "pimDM::receiveChangeNotification - IGMP change -
address were removed." << endl;
00944             members = (addRemoveAddr *) (details);
00945             oldMulticastAddr(members);
00946             break;
00947
00948         case NF_IPv4_DATA_ON_PRUNED_INT:
00949             EV << "pimDM::receiveChangeNotification - Data appears
on pruned interface." << endl;
00950             ctrl = (IPControlInfo *) (details);
00951             dataOnPruned(ctrl->getDestAddr(), ctrl->getSrcAddr());
00952             break;
00953
00954         // data come to non-RPF interface
00955         case NF_IPv4_DATA_ON_NONRPF:
00956             EV << "pimDM::receiveChangeNotification - Data appears
on non-RPF interface." << endl;
00957             ctrl = (IPControlInfo *) (details);
00958             dataOnNonRpf(ctrl->getDestAddr(), ctrl->getSrcAddr(),
ctrl->getInterfaceId());
00959             break;
00960
00961         // data come to RPF interface

```

```

00960         case NF_IPv4_DATA_ON_RPF:
00961             EV << "pimDM::receiveChangeNotification - Data appears
on RPF interface." << endl;
00962             route = (MulticastIPRoute *) (details);
00963             dataOnRpf(route);
00964             break;
00965
00966             // RPF interface has changed
00967         case NF_IPv4_ROUTE_ADDED:
00968             EV << "pimDM::receiveChangeNotification - RPF interface
has changed." << endl;
00969             IPRoute *entry = (IPRoute *) (details);
00970             vector<MulticastIPRoute*> routes = mrt->
getRoutesForSource(entry->getHost());
00971             for (unsigned int i = 0; i < routes.size(); i++)
00972                 rpfIntChange(routes[i]);
00973             break;
00974         }
00975     }
00976
00977 void pimDM::rpfIntChange(MulticastIPRoute *route)
00991 {
00992     IPAddress source = route->getSource();
00993     IPAddress group = route->getGroup();
00994     InterfaceEntry *newRpf = rt->getInterfaceForDestAddr(source);
00995     int rpfId = newRpf->getInterfaceId();
00996
00997     // is there any change?
00998     if (rpfId == route->getInIntId())
00999         return;
01000     EV << "New RPF int for group " << group << " source " << source << " is
" << rpfId << endl;
01001
01002     // set new RPF
01003     inInterface oldRpf = route->getInInt();
01004     route->setInInt(newRpf, rpfId, pimNbt->getNeighborsByIntID(rpfId)[0].
getAddr());
01005
01006     // route was not pruned, join to the multicast tree again
01007     if (!route->isFlagSet(P))
01008     {
01009         sendPimGraft(route->getInIntNextHop(), source, group, rpfId);
01010         PIMgrt* timer = createGraftRetryTimer(source, group);
01011         route->setGrt(timer);
01012     }
01013
01014     // find rpf int in outgoing imterfaces and delete it
01015     InterfaceVector outInt = route->getOutInt();
01016     for(unsigned int i = 0; i < outInt.size(); i++)
01017     {
01018         if (outInt[i].intId == rpfId)
01019         {
01020             if (outInt[i].pruneTimer != NULL)
01021             {
01022                 cancelEvent(outInt[i].pruneTimer);
01023                 delete outInt[i].pruneTimer;
01024                 outInt[i].pruneTimer = NULL;
01025             }
01026             outInt.erase(outInt.begin() + i);
01027             break;
01028         }
01029     }
01030
01031     // old RPF should be now outgoing interface if it is not down
01032     if (!oldRpf.intPtr->isDown())
01033     {
01034         outInterface newOutInt;
01035         newOutInt.intId = oldRpf.intId;
01036         newOutInt.intPtr = oldRpf.intPtr;
01037         newOutInt.pruneTimer = NULL;
01038         newOutInt.forwarding = Forward;
01039         newOutInt.mode = Densemode;
01040         outInt.push_back(newOutInt);
01041     }
01042
01043     route->setOutInt(outInt);
01044     mrt->generateShowIPMroute();
01045 }
01046
01047 void pimDM::dataOnRpf(MulticastIPRoute *route)
01058 {
01059     cancelEvent(route->getSat());
01060     scheduleAt(simTime() + SAT, route->getSat());
01061 }
01062
01063

```

```

01077 void pimDM::dataOnNonRpf(IPAddress group, IPAddress source, int intId)
01078 {
01079     EV << "pimDM::dataOnNonRpf, intID: " << intId << endl;
01080
01081     // load route from mroute
01082     MulticastIPRoute *route = mrt->getRouteFor(group, source);
01083     if (route == NULL)
01084         return;
01085
01086     // in case of p2p link, send prune
01087     // FIXME There should be better indicator of P2P link
01088     if (pimNbt->getNumNeighborsOnInt(intId) == 1)
01089     {
01090         // send Prune msg to the neighbor who sent these multicast data
01091         IPAddress nextHop = (pimNbt->getNeighborsByIntID(intId))[0].
getAddr();
01092         sendPimJoinPrune(nextHop, source, group, intId);
01093
01094         // find incoming interface
01095         int i = route->getOutIdByIntID(intId);
01096         InterfaceVector outInt = route->getOutInt();
01097
01098         // the incoming interface has to change its state to Pruned
01099         if (outInt[i].forwarding == Forward)
01100         {
01101             outInt[i].forwarding = Pruned;
01102             PIMpt* timer = createPruneTimer(route->getSource(),
route->getGroup(), intId, PT);
01103             outInt[i].pruneTimer = timer;
01104             route->setOutInt(outInt);
01105
01106             // if there is no outgoing interface, Prune msg has to
be sent on upstream
01107             if (route->isOilistNull())
01108             {
01109                 EV << "pimDM::dataOnNonRpf: oilist is NULL,
send prune msg to upstream." << endl;
01110                 route->addFlag(P);
01111                 if (!route->isFlagSet(A))
01112                     sendPimJoinPrune(route->getInIntNextHop
(), route->getSource(), route->getGroup(), route->getInIntId());
01113             }
01114             mrt->generateShowIPMroute();
01115         }
01116     }
01117
01118     //FIXME in case of LAN
01119 }
01120
01131 void pimDM::dataOnPruned(IPAddress group, IPAddress source)
01132 {
01133     EV << "pimDM::dataOnPruned" << endl;
01134     MulticastIPRoute *route = mrt->getRouteFor(group, source);
01135     // if GRT is running now, do not send Prune msg
01136     if (route->isFlagSet(P) && (route->getGrt() != NULL))
01137     {
01138         cancelEvent(route->getGrt());
01139         delete route->getGrt();
01140         route->setGrt(NULL);
01141     }
01142     // otherwise send Prune msg to upstream router
01143     else if (!route->isFlagSet(A))
01144         sendPimJoinPrune(route->getInIntNextHop(), source, group, route
->getInIntId());
01145 }
01146
01158 void pimDM::oldMulticastAddr(addRemoveAddr *members)
01159 {
01160     EV << "pimDM::oldMulticastAddr" << endl;
01161     vector<IPAddress> oldAddr = members->getAddr();
01162     PimInterface * pimInt = members->getInt();
01163     bool connected = false;
01164
01165     // go through all old multicast addresses assigned to interface
01166     for (unsigned int i = 0; i < oldAddr.size(); i++)
01167     {
01168         EV << "Removed multicast address: " << oldAddr[i] << endl;
01169         vector<MulticastIPRoute*> routes = mrt->getRouteFor(oldAddr[i])
;
01170
01171         // there is no route for group in the table
01172         if (routes.size() == 0)
01173             continue;
01174
01175         // go through all multicast routes
01176         for (unsigned int j = 0; j < routes.size(); j++)
01177         {

```

```

01178         MulticastIPRoute *route = routes[j];
01179         InterfaceVector outInt = route->getOutInt();
01180         unsigned int k;
01181
01182         // is interface in list of outgoing interfaces?
01183         for (k = 0; k < outInt.size(); k++)
01184         {
01185             if (outInt[k].intId == pimInt->getInterfaceID()
01186 )
01187             {
01188                 EV << "Interface is present, removing
it from the list of outgoing interfaces." << endl;
01189                 outInt.erase(outInt.begin() + k);
01190             }
01191             else if(outInt[k].forwarding == Forward)
01192             {
01193                 if ((pimNbt->getNeighborsByIntID (outInt
[k].intId)).size() == 0)
01194                     connected = true;
01195             }
01196             route->setOutInt (outInt);
01197
01198             // if there is no directly connected member of group
01199             if (!connected)
01200                 route->removeFlag(C);
01201
01202             // there is no receiver of multicast, prune the router
from the multicast tree
01203             if (route->isOilistNull())
01204             {
01205                 EV << "There is no receiver for the group ->
prune from the tree" << endl;
01206                 // if GRT is running now, do not send Prune msg
01207                 if (route->isFlagSet(P) && (route->getGrt() !=
NULL))
01208                 {
01209                     cancelEvent(route->getGrt());
01210                     delete route->getGrt();
01211                     route->setGrt(NULL);
01212                     sendPimJoinPrune(route->getInIntNextHop
(), route->getSource(), route->getGroup(), route->getInIntId());
01213                 }
01214
01215                 // if the source is not directly connected,
sent Prune msg
01216                 if (!route->isFlagSet(A) && !route->isFlagSet(P
))
01217                     sendPimJoinPrune(route->getInIntNextHop
(), route->getSource(), route->getGroup(), route->getInIntId());
01218
01219                 route->addFlag(P);
01220             }
01221         }
01222     }
01223     mrt->generateShowIPMroute();
01224 }
01225
01239 void pimDM::newMulticastAddr(addRemoveAddr *members)
01240 {
01241     EV << "pimDM::newMulticastAddr" << endl;
01242     vector<IPAddress> newAddr = members->getAddr();
01243     PimInterface * pimInt = members->getInt();
01244     bool forward = false;
01245
01246     // go through all new multicast addresses assigned to interface
01247     for (unsigned int i = 0; i < newAddr.size(); i++)
01248     {
01249         EV << "New multicast address: " << newAddr[i] << endl;
01250         vector<MulticastIPRoute*> routes = mrt->getRouteFor(newAddr[i])
;
01251
01252         // there is no route for group in the table in this moment
01253         if (routes.size() == 0)
01254             continue;
01255
01256         // go through all multicast routes
01257         for (unsigned int j = 0; j < routes.size(); j++)
01258         {
01259             MulticastIPRoute *route = routes[j];
01260             InterfaceVector outInt = route->getOutInt();
01261             unsigned int k;
01262
01263             // check on RPF interface
01264             if (route->getInIntId() == pimInt->getInterfaceID())
01265                 continue;
01266

```



```

01267         // is interface in list of outgoing interfaces?
01268         for (k = 0; k < outInt.size(); k++)
01269         {
01270             if (outInt[k].intId == pimInt->getInterfaceID()
01271 )
01272             {
01273                 EV << "Interface is already on list of
01274 outgoing interfaces" << endl;
01275                 if (outInt[k].forwarding == Pruned)
01276                     outInt[k].forwarding = Forward;
01277                 forward = true;
01278                 break;
01279             }
01280         }
01281         // interface is not in list of outgoing interfaces
01282         if (k == outInt.size())
01283         {
01284             EV << "Interface is not on list of outgoing
01285 interfaces yet, it will be added" << endl;
01286             outInterface newInt;
01287             newInt.intPtr = pimInt->getInterfacePtr();
01288             newInt.intId = pimInt->getInterfaceID();
01289             newInt.mode = Densemode;
01290             newInt.forwarding = Forward;
01291             newInt.pruneTimer = NULL;
01292             outInt.push_back(newInt);
01293             forward = true;
01294             route->setOutInt(outInt);
01295             route->addFlag(C);
01296             // route was pruned, has to be added to multicast tree
01297             if (route->isFlagSet(P) && forward)
01298             {
01299                 EV << "Route was pruned -> router has to join
01300 to multicast tree" << endl;
01301                 // if source is not directly connected, send
01302                 Graft to upstream
01303                 if (!route->isFlagSet(A))
01304                 {
01305                     sendPimGraft(route->getInIntNextHop(),
01306 route->getSource(), route->getGroup(), route->getInIntId());
01307                     PIMgrt *timer = createGraftRetryTimer(
01308 route->getSource(), route->getGroup());
01309                     route->setGrt(timer);
01310                 }
01311                 else
01312                     route->removeFlag(P);
01313             }
01314             mrt->generateShowIPMroute();
01315         }
01316     }
01317 void pimDM::newMulticast(MulticastIPRoute *newRoute)
01318 {
01319     EV << "pimDM::newMulticast" << endl;
01320     // only outgoing interfaces are missing
01321     PimInterface *rpfInt = pimIft->getInterfaceByIntID(newRoute->getInIntId
01322 ());
01323     bool pruned = true;
01324     // insert all PIM interfaces except rpf int
01325     for (int i = 0; i < pimIft->getNumInterface(); i++)
01326     {
01327         PimInterface *pimIntTemp = pimIft->getInterface(i);
01328         int intId = pimIntTemp->getInterfaceID();
01329         //check if PIM interface is not RPF interface
01330         if (pimIntTemp == rpfInt)
01331             continue;
01332         // create new outgoing interface
01333         outInterface newOutInt;
01334         newOutInt.intId = pimIntTemp->getInterfaceID();
01335         newOutInt.intPtr = pimIntTemp->getInterfacePtr();
01336         newOutInt.pruneTimer = NULL;
01337         switch (pimIntTemp->getMode())
01338         {
01339             case Dense:
01340                 newOutInt.mode = Densemode;
01341                 break;
01342             case Sparse:

```

```

01358             newOutInt.mode = Sparsemode;
01359             break;
01360         }
01361         // if there are neighbors on interface, we will forward
01362         if((pimNbt->getNeighborsByIntID(intId)).size() > 0)
01363         {
01364             newOutInt.forwarding = Forward;
01365             pruned = false;
01366             newRoute->addOutInt(newOutInt);
01367         }
01368         // if there is member of group, we will forward
01369         else if (pimIntTemp->isLocalIntMulticastAddress(newRoute->
01370 getGroup()))
01371         {
01372             newOutInt.forwarding = Forward;
01373             pruned = false;
01374             newRoute->addFlag(C);
01375             newRoute->addOutInt(newOutInt);
01376         }
01377         // in any other case interface is not involved
01378     }
01379     // directly connected to source, set State Refresh Timer
01380     if (newRoute->isFlagSet(A))
01381     {
01382         //FIXME record TTL (I do not know why???)
01383         PIMsrt* timerSrt = createStateRefreshTimer(newRoute->getSource(),
01384 newRoute->getGroup());
01385         newRoute->setSrt(timerSrt);
01386     }
01387     // set Source Active Timer (liveness of route)
01388     PIMsat* timerSat = createSourceActiveTimer(newRoute->getSource(),
01389 newRoute->getGroup());
01390     newRoute->setSat(timerSat);
01391     // if there is no outgoing interface, prune from multicast tree
01392     if (pruned)
01393     {
01394         EV << "pimDM::newMulticast: There is no outgoing interface for
01395 multicast, send Prune msg to upstream" << endl;
01396         newRoute->addFlag(P);
01397         if (!newRoute->isFlagSet(A))
01398             sendPimJoinPrune(newRoute->getInIntNextHop(), newRoute
01399 ->getSource(), newRoute->getGroup(), newRoute->getInIntId());
01400         // FIXME set timer which I do not use
01401     }
01402     // add new route record to multicast routing table
01403     mrt->addRoute(newRoute);
01404     EV << "PimSplitter::newMulticast: New route was added to the multicast
01405 routing table." << endl;
01406 }
01407 }
01408

```

7.17 F:/ANSA/src/ansa/pim/modes/pimDM.h File Reference

File implements PIM dense mode.

```

#include <omnetpp.h>
#include "PIMPacket_m.h"
#include "PIMTimer_m.h"
#include "AnsaInterfaceTableAccess.h"
#include "MulticastRoutingTableAccess.h"
#include "RoutingTableAccess.h"
#include "NotificationBoard.h"
#include "NotifierConsts.h"
#include "PimNeighborTable.h"
#include "PimInterfaceTable.h"
#include "IPControlInfo.h"
#include "IPv4InterfaceData.h"

```

Classes

- class [pimDM](#)

Class implements PIM-DM (dense mode).

Defines

- `#define PT 180.0`
- `#define GRT 3.0`
- `#define SAT 210.0`
- `#define SRT 60.0`

7.17.1 Detailed Description

File implements PIM dense mode.

Date

29.10.2011

Author

: Veronika Rybova

Implementation according to RFC3973.

Definition in file [pimDM.h](#).

7.17.2 Define Documentation

7.17.2.1 `#define PT 180.0`

Prune Timer = 180s (3min).

Definition at line 25 of file [pimDM.h](#).

7.17.2.2 `#define GRT 3.0`

Graft Retry Timer = 3s.

Definition at line 26 of file [pimDM.h](#).

7.17.2.3 `#define SAT 210.0`

Source Active Timer = 210s, Cisco has 180s, after that, route is flushed

Definition at line 27 of file [pimDM.h](#).

7.17.2.4 `#define SRT 60.0`

State Refresh Timer = 60s.

Definition at line 28 of file [pimDM.h](#).

7.18 pimDM.h

```

00001
00009 #ifndef HLIDAC_PIMDM
00010 #define HLIDAC_PIMDM
00011
00012 #include <omnetpp.h>
00013 #include "PIMPacket_m.h"
00014 #include "PIMTimer_m.h"
00015 #include "AnsaInterfaceTableAccess.h"
00016 #include "MulticastRoutingTableAccess.h"
00017 #include "RoutingTableAccess.h"
00018 #include "NotificationBoard.h"
00019 #include "NotifierConsts.h"
00020 #include "PimNeighborTable.h"
00021 #include "PimInterfaceTable.h"
00022 #include "IPControlInfo.h"
00023 #include "IPv4InterfaceData.h"
00024
00025 #define PT 180.0
00026 #define GRT 3.0
00027 #define SAT 210.0
00028 #define SRT 60.0
00033 class pimDM : public cSimpleModule, protected INotifiable
00034 {
00035     private:
00036         IRoutingTable                *rt;
00037         MulticastRoutingTable        *mrt;
00038         IInterfaceTable              *ift;
00039         NotificationBoard             *nb;
00040         PimInterfaceTable            *pimIft;
00041         PimNeighborTable             *pimNbt;
00043         // process events
00044         void receiveChangeNotification(int category, const cPolymorphic *
details);
00045         void newMulticast(MulticastIPRoute *newRoute);
00046         void newMulticastAddr(addRemoveAddr *members);
00047         void oldMulticastAddr(addRemoveAddr *members);
00048         void dataOnPruned(IPAddress destAddr, IPAddress srcAddr);
00049         void dataOnNonRpf(IPAddress group, IPAddress source, int intId);
00050         void dataOnRpf(MulticastIPRoute *route);
00051         void rpfIntChange(MulticastIPRoute *route);
00052
00053         // process timers
00054         void processPIMTimer(PIMTimer *timer);
00055         void processPruneTimer(PIMpt * timer);
00056         void processGraftRetryTimer(PIMgrt *timer);
00057         void processSourceActiveTimer(PIMsat * timer);
00058         void processStateRefreshTimer(PIMSrt * timer);
00059
00060         // create timers
00061         PIMpt* createPruneTimer(IPAddress source, IPAddress group, int
intId, int holdTime);
00062         PIMgrt* createGraftRetryTimer(IPAddress source, IPAddress group);
00063         PIMsat* createSourceActiveTimer(IPAddress source, IPAddress group);
00064         PIMSrt* createStateRefreshTimer(IPAddress source, IPAddress group);
00065
00066         // process PIM packets
00067         void processPIMPkt(PIMPacket *pkt);
00068         void processJoinPruneGraftPacket(PIMJoinPrune *pkt, PIMPacketType
type);
00069         void processPrunePacket(MulticastIPRoute *route, int intId, int
holdTime);
00070         void processGraftPacket(IPAddress source, IPAddress group,
IPAddress sender, int intId);
00071         void processGraftAckPacket(MulticastIPRoute *route);
00072         void processStateRefreshPacket(PIMStateRefresh *pkt);
00073
00074         //create PIM packets
00075         void sendPimJoinPrune(IPAddress nextHop, IPAddress src, IPAddress
grp, int intId);
00076         void sendPimGraft(IPAddress nextHop, IPAddress src, IPAddress grp,
int intId);
00077         void sendPimGraftAck(PIMGraftAck *msg);
00078         void sendPimStateRefresh(IPAddress originator, IPAddress src,
IPAddress grp, int intId, bool P);
00079
00080
00081
00082     protected:
00083         virtual int numInitStages() const {return 5;}
00084         virtual void handleMessage(cMessage *msg);
00085         virtual void initialize(int stage);
00086 };
00087
00088 #endif

```

7.19 F:/ANSA/src/ansa/pim/modes/pimSM.cc File Reference

File implements PIM sparse mode.

```
#include "pimSM.h"
```

Functions

- **Define_Module** ([pimSM](#))

7.19.1 Detailed Description

File implements PIM sparse mode.

Date

29.10.2011

Author

: Veronika Rybova

Implementation will be done in the future according to RFC4601.

Definition in file [pimSM.cc](#).

7.20 pimSM.cc

```
00001
00009 #include "pimSM.h"
00010
00011
00012 Define_Module(pimSM);
00013
00014 void pimSM::handleMessage(cMessage *msg)
00015 {
00016     EV << "PIMSM::handleMessage" << endl;
00017
00018     // self message (timer)
00019     if (msg->isSelfMessage())
00020     {
00021         EV << "PIMSM::handleMessage:Timer" << endl;
00022         PIMTimer *timer = check_and_cast<PIMTimer*>(msg);
00023     }
00024     else if (dynamic_cast<PIMPacket*>(msg))
00025     {
00026         EV << "PIMSM::handleMessage: PIM-SM packet" << endl;
00027         PIMPacket *pkt = check_and_cast<PIMPacket*>(msg);
00028         EV << "Verze: " << pkt->getVersion() << ", typ: " << pkt->getType()
<< endl;
00029     }
00030     else
00031         EV << "PIMSM::handleMessage: Wrong message" << endl;
00032 }
00033
00034 void pimSM::initialize(int stage)
00035 {
00036     ;
00037 }
00038
```

7.21 F:/ANSA/src/ansa/pim/modes/pimSM.h File Reference

File implements PIM sparse mode.

```
#include <omnetpp.h>
#include "PIMPacket_m.h"
#include "PIMTimer_m.h"
```

Classes

- class [pimSM](#)

Class implements PIM-SM (sparse mode).

7.21.1 Detailed Description

File implements PIM sparse mode.

Date

29.10.2011

Author

: Veronika Rybova

Implementation will be done in the future according to RFC4601.

Definition in file [pimSM.h](#).

7.22 pimSM.h

```
00001
00009 #ifndef HLIDAC_PIMDM
00010 #define HLIDAC_PIMDM
00011
00012 #include <omnetpp.h>
00013 #include "PIMPacket_m.h"
00014 #include "PIMTimer_m.h"
00015
00019 class pimSM : public cSimpleModule
00020 {
00021     protected:
00022         virtual int numInitStages() const {return 5;}
00023         virtual void handleMessage(cMessage *msg);
00024         virtual void initialize(int stage);
00025 };
00026
00027 #endif
```

7.23 F:/ANSA/src/ansa/pim/PimSplitter.cc File Reference

File contains implementation of PIMSplitter.

```
#include "PimSplitter.h"
```

Functions

- **Define_Module** ([PimSplitter](#))

7.23.1 Detailed Description

File contains implementation of PIMSplitter.

Date

3.12.2011

Author

Veronika Rybova

Splitter is common for all PIM modes. It is used to resent all PIM messages to correct PIM mode module. It also does work which is same for all modes, e.g. it send Hello messages and it manages table of PIM interfaces.

Definition in file [PimSplitter.cc](#).

7.24 PimSplitter.cc

```

00001
00011 #include "PimSplitter.h"
00012
00013 using namespace std;
00014
00015 Define_Module(PimSplitter);
00016
00017
00027 PIMHello* PimSplitter::createHelloPkt(int iftID)
00028 {
00029     PIMHello *msg = new PIMHello();
00030     msg->setName("PIMHello");
00031
00032     IPControlInfo *ctrl = new IPControlInfo();
00033     IPAddress gal("224.0.0.13");
00034     ctrl->setDestAddr(gal);
00035     //ctrl->setProtocol(IP_PROT_PIM);
00036     ctrl->setProtocol(103);
00037     ctrl->setTimeToLive(1);
00038     ctrl->setInterfaceId(iftID);
00039     msg->setControlInfo(ctrl);
00040
00041     return msg;
00042 }
00043
00053 void PimSplitter::sendHelloPkt()
00054 {
00055     EV << "PIM::sendHelloPkt" << endl;
00056     int intID;
00057     PIMHello* msg;
00058
00059     // send to all PIM interfaces
00060     for (int i = 0; i < pimIft->getNumInterface(); i++)
00061     {
00062         intID = pimIft->getInterface(i)->getInterfaceID();
00063         msg = createHelloPkt(intID);
00064         send(msg, "transportOut");
00065     }
00066
00067     // start Hello timer
00068     PIMTimer *timer = new PIMTimer("Hello");
00069     timer->setTimerKind(HelloTimer);
00070     scheduleAt(simTime() + HT, timer);
00071 }
00072
00087 void PimSplitter::processHelloPkt(PIMPacket *msg)
00088 {
00089     EV << "PIM::processHelloPkt" << endl;
00090
00091     IPControlInfo *ctrl = dynamic_cast<IPControlInfo *>(msg->getControlInfo
00092     ());
00093     PimNeighbor newEntry;
00094     PIMnlt *nlt;
00095
00096     // get information about neighbor from Hello packet
00097     newEntry.setAddr(ctrl->getSrcAddr());
00098     newEntry.setInterfaceID(ctrl->getInterfaceId());
00099     newEntry.setInterfacePtr(ift->getInterfaceById(ctrl->getInterfaceId()))

```

```

;
00099     newEntry.setVersion(msg->getVersion());
00100
00101
00102     // new neighbor (it is not in PIM neighbor table)
00103     // insert new neighbor to table
00104     // set Neighbor Livness Timer
00105     if (!pimNbt->isInTable(newEntry))
00106     {
00107         nlt = new PIMnlt("NeighborLivenessTimer");
00108         nlt->setTimerKind(NeighborLivenessTimer);
00109         nlt->setNtId(pimNbt->getIdCounter());
00110         scheduleAt(simTime() + 3.5*HT, nlt);
00111
00112         newEntry.setNlt(nlt);
00113         pimNbt->addNeighbor(newEntry);
00114         EV << "PimSplitter::New Entry was added: addr = " << newEntry.
getAddr() << ", iftID = " << newEntry.getInterfaceID() << ", ver = " <<
newEntry.getVersion() << endl;
00115     }
00116     // neighbor is already in PIM neighbor table
00117     // refresh Neighbor Livness Timer
00118     else
00119     {
00120         nlt = pimNbt->findNeighbor(ctrl->getInterfaceId(), ctrl->
getSrcAddr())->getNlt();
00121         cancelEvent(nlt);
00122         scheduleAt(simTime() + 3.5*HT, nlt);
00123     }
00124
00125     delete msg;
00126 }
00127
00138 void PimSplitter::processNLTimer(PIMTimer *timer)
00139 {
00140     EV << "PIM::processNLTimer"<< endl;
00141     PIMnlt *nlt = check_and_cast<PIMnlt *>(timer);
00142     int id = nlt->getNtId();
00143     IPAddress neighbor;
00144
00145     // if neighbor exists store its IP address
00146     if (pimNbt->getNeighborsByID(id) != NULL)
00147         neighbor = pimNbt->getNeighborsByID(id)->getAddr();
00148
00149     // Record in PIM Neighbor Table was found, can be deleted.
00150     if (pimNbt->deleteNeighbor(id))
00151         EV << "PIM::processNLTimer: Neighbor " << neighbor << "was
removed from PIM neighbor table." << endl;
00152
00153     delete nlt;
00154 }
00155
00168 void PimSplitter::processPIMPkt(PIMPacket *pkt)
00169 {
00170     EV << "PIM::processPIMPkt" << endl;
00171
00172     IPControlInfo *ctrl = dynamic_cast<IPControlInfo *>(pkt->getControlInfo
());
00173     int intID = ctrl->getInterfaceId();
00174     int mode = 0;
00175
00176     // find information about interface where packet came from
00177     PimInterface *pimInt = pimIf->getInterfaceByIntID(intID);
00178     if (pimInt != NULL)
00179         mode = pimInt->getMode();
00180
00181     // according to interface PIM mode send packet to appropriate PIM
module
00182     switch(mode)
00183     {
00184         case Dense:
00185             send(pkt, "pimDMOut");
00186             break;
00187         case Sparse:
00188             send(pkt, "pimSMOut");
00189             break;
00190         default:
00191             EV << "PIM::processPIMPkt: PIM is not enabled on
interface number: "<< intID << endl;
00192             delete pkt;
00193     }
00194 }
00195
00210 void PimSplitter::handleMessage(cMessage *msg)
00211 {
00212     EV << "PimSplitter::handleMessage" << endl;
00213

```



```

00214         // self message (timer)
00215         if (msg->isSelfMessage())
00216         {
00217             PIMTimer *timer = check_and_cast<PIMTimer*>(msg);
00218             if (timer->getTimerKind() == HelloTimer)
00219             {
00220                 EV << "PIM::HelloTimer" << endl;
00221                 sendHelloPkt();
00222                 delete timer;
00223             }
00224             else if (timer->getTimerKind() == NeighborLivenessTimer)
00225             {
00226                 EV << "PIM::NeighborLivenessTimer" << endl;
00227                 processNLTimer(timer);
00228             }
00229         }
00230
00231         // PIM packet from network layer
00232         else if (dynamic_cast<PIMPacket*>(msg) && (strcmp(msg->getArrivalGate()->
getName(), "transportIn") == 0))
00233         {
00234             PIMPacket *pkt = check_and_cast<PIMPacket*>(msg);
00235
00236             if (pkt->getType() == Hello)
00237                 processHelloPkt(pkt);
00238             else
00239                 processPIMPkt(pkt);
00240         }
00241
00242         // PIM packet from PIM mode, send to network layer
00243         else if (dynamic_cast<PIMPacket*>(msg))
00244             send(msg, "transportOut");
00245         else
00246             EV << "PIM:ERROR - bad type of message" << endl;
00247     }
00248 }
00249
00261 void PimSplitter::initialize(int stage)
00262 {
00263     // in stage 2 interfaces are registered
00264     // in stage 3 table pimInterfaces is built
00265     if (stage == 4)
00266     {
00267         EV << "PimSplitter::initialize" << endl;
00268         hostname = par("hostname");
00269
00270         // Pointer to routing tables, interface tables, notification
00271         board
00272         {
00273             rt = RoutingTableAccess().get();
00274             mrt = MulticastRoutingTableAccess().get();
00275             ift = AnsaInterfaceTableAccess().get();
00276             nb = NotificationBoardAccess().get();
00277             pimIft = PimInterfaceTableAccess().get();
00278             pimNbt = PimNeighborTableAccess().get();
00279
00280             // subscription of notifications (future use)
00281             nb->subscribe(this, NF_IPv4_NEW_MULTICAST);
00282             nb->subscribe(this, NF_INTERFACE_IPv4CONFIG_CHANGED);
00283
00284             // is PIM enabled?
00285             if (pimIft->getNumInterface() == 0)
00286                 return;
00287             else
00288                 EV << "PIM is enabled on device " << hostname << endl;
00289
00290             // send Hello packets to PIM neighbors (224.0.0.13)
00291             PIMTimer *timer = new PIMTimer("Hello");
00292             timer->setTimerKind(HelloTimer);
00293             scheduleAt(simTime() + uniform(0,5), timer);
00294         }
00295     }
00296
00306 void PimSplitter::receiveChangeNotification(int category, const cPolymorphic *
details)
00307 {
00308     // ignore notifications during initialize
00309     if (simulation.getContextType() == CTX_INITIALIZE)
00310         return;
00311
00312     // PIM needs details
00313     if (details == NULL)
00314         return;
00315
00316     Enter_Method_Silent();
00317     printNotificationBanner(category, details);
00318 }

```

```

00319         // according to category of event...
00320         switch (category)
00321         {
00322             // new multicast data appears in router
00323             case NF_IPv4_NEW_MULTICAST:
00324                 EV << "PimSplitter::receiveChangeNotification - NEW
MULTICAST" << endl;
00325                 IPControlInfo *ctrl;
00326                 ctrl = (IPControlInfo *) (details);
00327                 newMulticast(ctrl->getDestAddr(), ctrl->getSrcAddr());
00328                 break;
00329
00330             // configuration of interface changed, it means some change
00331             from IGMP
00332             case NF_INTERFACE_IPv4CONFIG_CHANGED:
00333                 EV << "PimSplitter::receiveChangeNotification - IGMP
change" << endl;
00334                 InterfaceEntry * interface = (InterfaceEntry *) (details
);
00335                 igmpChange(interface);
00336                 break;
00337         }
00338     };
00339 void PimSplitter::igmpChange(InterfaceEntry *interface)
00340 {
00341     EV << "PimSplitter::igmpChange" << endl;
00342     int intId = interface->getInterfaceId();
00343     PimInterface * pimInt = pimIf->getInterfaceByIntID(intId);
00344
00345     // save old and new set of multicast IP address assigned to interface
00346     vector<IPAddress> multicastAddrsOld = pimInt->getIntMulticastAddresses(
);
00347     vector<IPAddress> multicastAddrsNew = pimInt->deleteLocalIPs(interface
->ipv4Data()->getMulticastGroups());
00348
00349     // vectors of new and removed multicast addresses
00350     vector<IPAddress> add;
00351     vector<IPAddress> remove;
00352
00353     // which address was removed from interface
00354     for (unsigned int i = 0; i < multicastAddrsOld.size(); i++)
00355     {
00356         unsigned int j;
00357         for (j = 0; j < multicastAddrsNew.size(); j++)
00358         {
00359             if (multicastAddrsOld[i] == multicastAddrsNew[j])
00360                 break;
00361         }
00362         if (j == multicastAddrsNew.size())
00363         {
00364             EV << "Multicast address " << multicastAddrsOld[i] << "
was removed from the interface " << intId << endl;
00365             remove.push_back(multicastAddrsOld[i]);
00366         }
00367     }
00368
00369     // which address was added to interface
00370     for (unsigned int i = 0; i < multicastAddrsNew.size(); i++)
00371     {
00372         unsigned int j;
00373         for (j = 0; j < multicastAddrsOld.size(); j++)
00374         {
00375             if (multicastAddrsNew[i] == multicastAddrsOld[j])
00376                 break;
00377         }
00378         if (j == multicastAddrsOld.size())
00379         {
00380             EV << "Multicast address " << multicastAddrsNew[i] << "
was added to the interface " << intId << endl;
00381             add.push_back(multicastAddrsNew[i]);
00382         }
00383     }
00384
00385     // notification about removed multicast address to PIM modules
00386     addRemoveAddr *addr = new addRemoveAddr();
00387     if (remove.size() > 0)
00388     {
00389         // remove new address
00390         for(unsigned int i = 0; i < remove.size(); i++)
00391             pimInt->removeIntMulticastAddress(remove[i]);
00392
00393         // send notification
00394         addr->setAddr(remove);
00395         addr->setInt(pimInt);
00396         nb->fireChangeNotification(NF_IPv4_NEW_IGMP_REMOVED, addr);
00397     }
00398 }

```

```

00407
00408 // notification about new multicast address to PIM modules
00409 if (add.size() > 0)
00410 {
00411     // add new address
00412     for(unsigned int i = 0; i < add.size(); i++)
00413         pimInt->addIntMulticastAddress(add[i]);
00414
00415     // send notification
00416     addr->setAddr(add);
00417     addr->setInt(pimInt);
00418     nb->fireChangeNotification(NF_IPv4_NEW_IGMP_ADDED, addr);
00419 }
00420 }
00421
00433 void PimSplitter::newMulticast(IPAddress destAddr, IPAddress srcAddr)
00434 {
00435     EV << "PimSplitter::newMulticast - group: " << destAddr << ", source: "
00436     << srcAddr << endl;
00437
00438     // find RPF interface for new multicast stream
00439     InterfaceEntry *inInt = rt->getInterfaceForDestAddr(srcAddr);
00440     if (inInt == NULL)
00441     {
00442         EV << "ERROR: PimSplitter::newMulticast(): cannot find RPF
00443         interface, routing information is missing.";
00444         return;
00445     }
00446     int rpfId = inInt->getInterfaceId();
00447     PimInterface *pimInt = pimIft->getInterfaceByIntID(rpfId);
00448
00449     // if it is interface configured with PIM, create new route
00450     if (pimInt != NULL)
00451     {
00452         // create new multicast route
00453         MulticastIPRoute *newRoute = new MulticastIPRoute();
00454         newRoute->setGroup(destAddr);
00455         newRoute->setSource(srcAddr);
00456
00457         // Directly connected routes to source does not have next hop
00458         // RPF neighbor is source of packet
00459         IPAddress rpf;
00460         const IPRoute *routeToSrc = rt->findBestMatchingRoute(srcAddr);
00461         if (routeToSrc->getSource() == IPRoute::IFACENETMASK)
00462         {
00463             newRoute->addFlag(A);
00464             rpf = srcAddr;
00465         }
00466         // Not directly connected, next hop address is saved in routing
00467         table
00468         else
00469             rpf = rt->getGatewayForDestAddr(srcAddr);
00470
00471         newRoute->setInInt(inInt, inInt->getInterfaceId(), rpf);
00472
00473         // notification for PIM module about new multicast route
00474         if (pimInt->getMode() == Dense)
00475             nb->fireChangeNotification(NF_IPv4_NEW_MULTICAST_DENSE,
00476             newRoute);
00477     }
00478 }
00479
00500 bool PimSplitter::LoadConfigFromXML(const char *filename)
00501 {
00502     // file loading
00503     cXMLElement* asConfig = ev.getXMLDocument(filename);
00504     if (asConfig == NULL)
00505         return false;
00506
00507     // first element <Router id="192.168.10.7">
00508     std::string routerXPath("Router[@id='");
00509     IPAddress routerId = rt->getRouterId();
00510     routerXPath += routerId.str();
00511     routerXPath += "']";
00512
00513     cXMLElement* routerNode = asConfig->getElementByPath(routerXPath.c_str(
00514 ));
00515     if (routerNode == NULL)
00516     {
00517         error("No configuration for Router ID: %s", routerId.str().
00518         c_str());
00519         return false;
00520     }
00521
00522     // Routing element
00523     cXMLElement* routingNode = routerNode->getElementByPath("Routing");
00524     if (routingNode == NULL)

```

```

00523         return false;
00524
00525         // Multicast element
00526         cXMLElement* multicastNode = routingNode->getElementByPath("Multicast");
00527     };
00528     if (multicastNode == NULL)
00529         return false;
00530
00531     // Multicast has to be enabled
00532     const char* enableAtt = multicastNode->getAttribute("enable");
00533     if (strcmp(enableAtt, "1"))
00534         return false;
00535
00536     // Where is PIM protocol enabled?
00537     // Interfaces element
00538     cXMLElement* iftNode = routerNode->getElementByPath("Interfaces");
00539     if (iftNode == NULL)
00540         return false;
00541
00542     // list of interfaces, where PIM is enabled
00543     cXMLElementList childrenNodes = iftNode->getChildrenByTagName("Interface");
00544     //EV << "PimSplitter::Interface" << endl;
00545     if (childrenNodes.size() > 0)
00546     {
00547         //EV << "PimSplitter::Interface size: " << childrenNodes.size() <<
00548         endl;
00549         for (cXMLElementList::iterator node = childrenNodes.begin(); node !=
childrenNodes.end(); node++)
00550         {
00551             cXMLElement* pimNode = (*node)->getElementByPath("Pim");
00552             if (pimNode == NULL)
00553                 continue;
00554             //EV << "PimSplitter::PIM interface" << endl;
00555             // get ID of PIM interface
00556             InterfaceEntry *interface = ift->getInterfaceByName((*node)->
getAttribute("name"));
00557
00558             // create new PIM interface
00559             PimInterface newentry;
00560             newentry.setInterfaceID(interface->getInterfaceId());
00561             newentry.setInterfacePtr(interface);
00562
00563             // register pim multicast address 224.0.0.13 on pim interface
00564             vector<IPAddress> intMulticastAddresses = interface->ipv4Data
()->getMulticastGroups();
00565             intMulticastAddresses.push_back("224.0.0.13");
00566             interface->ipv4Data()->setMulticastGroups(
intMulticastAddresses);
00567
00568             // get PIM mode for interface
00569             cXMLElement* pimMode = pimNode->getElementByPath("Mode");
00570             if (pimMode == NULL)
00571                 return false;
00572
00573             const char *mode = pimMode->getNodeValue();
00574             //EV << "PimSplitter::PIM interface mode = " << mode << endl;
00575             if (!strcmp(mode, "dense-mode"))
00576                 newentry.setMode(Dense);
00577             else if (!strcmp(mode, "sparse-mode"))
00578                 newentry.setMode(Sparse);
00579             else
00580                 return false;
00581             pimIft->addInterface(newentry);
00582         }
00583     }
00584     else
00585         return false;
00586     return true;
00587 }

```

7.25 F:/ANSA/src/ansa/pim/PimSplitter.h File Reference

File contains implementation of PIMSplitter.

```
#include <omnetpp.h>
#include "PIMPacket_m.h"
#include "PIMTimer_m.h"
#include "IPControlInfo.h"
#include "IPv4InterfaceData.h"
#include "AnsaInterfaceTableAccess.h"
#include "MulticastRoutingTableAccess.h"
#include "RoutingTableAccess.h"
#include "AnsaInterfaceTable.h"
#include "AnsaRoutingTable.h"
#include "NotificationBoard.h"
#include "NotifierConsts.h"
#include "InterfaceStateManager.h"
#include "IPvXAddress.h"
#include "PimNeighborTable.h"
#include "PimInterfaceTable.h"
```

Classes

- class [PimSplitter](#)

Class implements PIM Splitter, which splits PIM messages to correct PIM module.

Defines

- #define [HT](#) 30.0

7.25.1 Detailed Description

File contains implementation of PIMSplitter.

Date

3.12.2011

Author

Veronika Rybova

Splitter is common for all PIM modes. It is used to resent all PIM messages to correct PIM mode module. It also does work which is same for all modes, e.g. it send Hello messages and it manages table of PIM interfaces.

Definition in file [PimSplitter.h](#).

7.25.2 Define Documentation

7.25.2.1 #define HT 30.0

Hello Timer = 30s.

Definition at line 32 of file [PimSplitter.h](#).

7.26 PimSplitter.h

```

00001
00011 #ifndef PIMSPLITTER_H_
00012 #define PIMSPLITTER_H_
00013
00014 #include <omnetpp.h>
00015 #include "PIMPacket_m.h"
00016 #include "PIMTimer_m.h"
00017 #include "IPControlInfo.h"
00018 #include "IPv4InterfaceData.h"
00019 #include "AnsaInterfaceTableAccess.h"
00020 #include "MulticastRoutingTableAccess.h"
00021 #include "RoutingTableAccess.h"
00022 #include "AnsaInterfaceTable.h"
00023 #include "AnsaRoutingTable.h"
00024 #include "NotificationBoard.h"
00025 #include "NotifierConsts.h"
00026 #include "InterfaceStateManager.h"
00027 #include "IPvXAddress.h"
00028 #include "PimNeighborTable.h"
00029 #include "PimInterfaceTable.h"
00030
00031
00032 #define HT 30.0
00041 class PimSplitter : public cSimpleModule, protected INotifiable
00042 {
00043     private:
00044         IRoutingTable                *rt;
00045         MulticastRoutingTable        *mrt;
00046         IInterfaceTable              *ift;
00047         NotificationBoard             *nb;
00048         PimInterfaceTable            *pimIft;
00049         PimNeighborTable             *pimNbt;
00050         const char *                 hostname;
00052         void processPIMPkt(PIMPacket *pkt);
00053         void processNLTimer(PIMTimer *timer);
00054
00055         // methods for Hello packets
00056         PIMHello* createHelloPkt(int iftID);
00057         void sendHelloPkt();
00058         void processHelloPkt(PIMPacket *pkt);
00059
00060         // process notification
00061         void receiveChangeNotification(int category, const cPolymorphic *
details);
00062         virtual void newMulticast(IPAddress destAddr, IPAddress srcAddr);
00063         void igmpChange(InterfaceEntry *interface);
00064
00065         // not in use
00066         bool LoadConfigFromXML(const char *filename);
00067
00068     protected:
00069         virtual int numInitStages() const {return 5;}
00070         virtual void handleMessage(cMessage *msg);
00071         virtual void initialize(int stage);
00072
00073     public:
00074         PimSplitter(){};
00075 };
00076
00077
00078 #endif /* PIMSPLITTER_H_ */
00079
00080

```

7.27 F:/ANSA/src/ansa/pim/tables/PimInterfaceTable.cc File Reference

File implements table of PIM interfaces.

```
#include "PimInterfaceTable.h"
```

Functions

- **Define_Module** ([PimInterfaceTable](#))
- `std::ostream & operator<<` (`std::ostream &os`, `const PimInterface &e`)

- `std::ostream & operator<< (std::ostream &os, const PimInterfaceTable &e)`

7.27.1 Detailed Description

File implements table of PIM interfaces.

Date

19.3.2012

Author

: Veronika Rybova

PIM interface table contains information about all interfaces which are configured by PIM protocol. Information are obtained from configuration file.

Definition in file [PimInterfaceTable.cc](#).

7.27.2 Function Documentation

7.27.2.1 `std::ostream& operator<< (std::ostream & os, const PimInterface & e)`

Printout of structure [PimInterface](#).

Definition at line 15 of file [PimInterfaceTable.cc](#).

```
{
    int i;
    std::vector<IPAddress> intMulticastAddresses = e.
    getIntMulticastAddresses();

    os << "ID = " << e.getInterfaceID() << "; mode = ";
    if (e.getMode() == Dense)
        os << "Dense";
    else if (e.getMode() == Sparse)
        os << "Sparse";
    os << "; Multicast addresses: ";

    int vel = intMulticastAddresses.size();
    if (vel > 0)
    {
        for(i = 0; i < (vel - 1); i++)
            os << intMulticastAddresses[i] << ", ";
        os << intMulticastAddresses[i];
    }
    else
        os << "Null";
    return os;
};
```

7.27.2.2 `std::ostream& operator<< (std::ostream & os, const PimInterfaceTable & e)`

Printout of structure PimInterfaces Table.

Definition at line 41 of file [PimInterfaceTable.cc](#).

```
{
    for (int i = 0; i < e.size(); i++)
        os << " ";
    //os << "ID = " << e.getInterface(i)->getInterfaceID() << ";
    mode = " << e.getInterface(i)->getMode();
    return os;
};
```

7.28 PimInterfaceTable.cc

```

00001
00010 #include "PimInterfaceTable.h"
00011
00012 Define_Module(PimInterfaceTable);
00013
00015 std::ostream& operator<<(std::ostream& os, const PimInterface& e)
00016 {
00017     int i;
00018     std::vector<IPAddress> intMulticastAddresses = e.
getIntMulticastAddresses();
00019
00020     os << "ID = " << e.getInterfaceID() << "; mode = ";
00021     if (e.getMode() == Dense)
00022         os << "Dense";
00023     else if (e.getMode() == Sparse)
00024         os << "Sparse";
00025     os << "; Multicast addresses: ";
00026
00027     int vel = intMulticastAddresses.size();
00028     if (vel > 0)
00029     {
00030         for(i = 0; i < (vel - 1); i++)
00031             os << intMulticastAddresses[i] << ", ";
00032         os << intMulticastAddresses[i];
00033     }
00034     else
00035         os << "Null";
00036     return os;
00037 };
00038
00039
00041 std::ostream& operator<<(std::ostream& os, const PimInterfaceTable& e)
00042 {
00043     for (int i = 0; i < e.size(); i++)
00044         os << " ";
00045     //os << "ID = " << e.getInterface(i)->getInterfaceID() << ";
mode = " << e.getInterface(i)->getMode();
00046     return os;
00047 };
00048
00050 std::string PimInterface::info() const
00051 {
00052     std::stringstream out;
00053     out << "ID = " << intID << "; mode = " << mode;
00054     return out.str();
00055 }
00056
00064 void PimInterface::removeIntMulticastAddress(IPAddress addr)
00065 {
00066     for(unsigned int i = 0; i < intMulticastAddresses.size(); i++)
00067     {
00068         if (intMulticastAddresses[i] == addr)
00069         {
00070             intMulticastAddresses.erase(intMulticastAddresses.begin
() + i);
00071             return;
00072         }
00073     }
00074 }
00075
00086 std::vector<IPAddress> PimInterface::deleteLocalIPs(std::vector<IPAddress>
multicastAddr)
00087 {
00088     EV << "PimInterface::deleteLocalIPs" << endl;
00089
00090     for(int j = 0; j < (multicastAddr.size()); j++)
00091         EV << multicastAddr[j] << ", ";
00092     EV << endl;
00093
00094     std::vector<IPAddress> newMulticastAddresses;
00095     for(unsigned int i = 0; i < multicastAddr.size(); i++)
00096     {
00097         EV << multicastAddr[i] << endl;
00098         if (!multicastAddr[i].isLinkLocalMulticast())
00099         {
00100             EV << "isLinkLocalMulticast" << endl;
00101             newMulticastAddresses.push_back(multicastAddr[i]);
00102         }
00103     }
00104     EV << "Velikost vysledku: " << newMulticastAddresses.size() << endl;
00105     return newMulticastAddresses;
00106 }
00107
00116 bool PimInterface::isLocalIntMulticastAddress (IPAddress addr)

```



```

00117 {
00118     for(unsigned int i = 0; i < intMulticastAddresses.size(); i++)
00119     {
00120         if (intMulticastAddresses[i] == addr)
00121             return true;
00122     }
00123     return false;
00124 }
00125
00126
00127
00133 void PimInterfaceTable::handleMessage(cMessage *msg)
00134 {
00135     opp_error("This module doesn't process messages");
00136 }
00137
00138 void PimInterfaceTable::initialize(int stage)
00139 {
00140     WATCH_VECTOR(pimIft);
00141 }
00142
00149 void PimInterfaceTable::printPimInterfaces()
00150 {
00151     for(std::vector<PimInterface>::iterator i = pimIft.begin(); i < pimIft.
end(); i++)
00152     {
00153         EV << (*i).info() << endl;
00154     }
00155 }
00156 }
00157
00168 PimInterface *PimInterfaceTable::getInterfaceByIntID(int intID)
00169 {
00170     for(int i = 0; i < getNumInterface(); i++)
00171     {
00172         if(intID == getInterface(i)->getInterfaceID())
00173         {
00174             return getInterface(i);
00175             break;
00176         }
00177     }
00178     return NULL;
00179 }

```

7.29 F:/ANSA/src/ansa/pim/tables/PimInterfaceTable.h File Reference

File implements table of PIM interfaces.

```

#include <omnetpp.h>
#include "AnsaInterfaceTable.h"
#include "IPAddress.h"

```

Classes

- class [PimInterface](#)
Class represents one entry of [PimInterfaceTable](#).
- class [PimInterfaceTable](#)
Class represents Pim Interface Table.
- class [PimInterfaceTableAccess](#)
Class gives access to the [PimInterfaceTable](#).
- class [addRemoveAddr](#)
Class is needed by notification about new multicast addresses on interface.

Enumerations

- enum [PIMmode](#) { **Dense** = 1, **Sparse** = 2 }

7.29.1 Detailed Description

File implements table of PIM interfaces.

Date

19.3.2012

Author

: Veronika Rybova

PIM interface table contains information about all interfaces which are configured by PIM protocol. Information are obtained from configuration file.

Definition in file [PimInterfaceTable.h](#).

7.29.2 Enumeration Type Documentation

7.29.2.1 enum PIMmode

PIM modes configured on the interface.

Definition at line 20 of file [PimInterfaceTable.h](#).

```
{
    Dense = 1,
    Sparse = 2
};
```

7.30 PimInterfaceTable.h

```
00001
00010 #ifndef PIMINTERFACES_H_
00011 #define PIMINTERFACES_H_
00012
00013 #include <omnetpp.h>
00014 #include "AnsaInterfaceTable.h"
00015 #include "IPAddress.h"
00016
00020 enum PIMmode
00021 {
00022     Dense = 1,
00023     Sparse = 2
00024 };
00025
00031 class INET_API PimInterface: public cPolymorphic
00032 {
00033     protected:
00034         int intID;
00035         InterfaceEntry * intPtr;
00036         PIMmode mode;
00037         std::vector<IPAddress> intMulticastAddresses;
00039     public:
00040         PimInterface(){intPtr = NULL;};
00041         virtual ~PimInterface() {};
00042         virtual std::string info() const;
00043
00044         // set methods
00045         void setInterfaceID(int iftID) {this->intID = iftID;};
00046         void setInterfacePtr(InterfaceEntry *intPtr) {this->intPtr =
00047             intPtr;};
00048         void setMode(PIMmode mode) {this->mode = mode;};
00049         //get methods
00050         int getInterfaceID() const {return intID;};
00051         InterfaceEntry *getInterfacePtr() const {return intPtr;};
00052         PIMmode getMode() const {return mode;};
00053         std::vector<IPAddress> getIntMulticastAddresses() const {return
00054             intMulticastAddresses;};
00055         // methods for work with vector "intMulticastAddresses"
00056         void setIntMulticastAddresses(std::vector<IPAddress>
```

```

        intMulticastAddresses) {this->intMulticastAddresses = intMulticastAddresses;}
00057     void addIntMulticastAddress(IPAddress addr) {this->
        intMulticastAddresses.push_back(addr);}
00058     void removeIntMulticastAddress(IPAddress addr);
00059     bool isLocalIntMulticastAddress (IPAddress addr);
00060     std::vector<IPAddress> deleteLocalIPs(std::vector<IPAddress>
        multicastAddr);
00061 };
00062
00063
00068 class INET_API PimInterfaceTable: public cSimpleModule
00069 {
00070     protected:
00071         std::vector<PimInterface>          pimIft;
00072     public:
00073         PimInterfaceTable(){};
00074         virtual ~PimInterfaceTable(){};
00075         virtual PimInterface *getInterface(int k){return &this->pimIft[
        k];}
00076         virtual void addInterface(const PimInterface entry){this->
        pimIft.push_back(entry);}
00077         //virtual bool deleteInterface(const PimInterface *entry){};
00078         virtual int getNumInterface() {return this->pimIft.size();}
00079         virtual void printPimInterfaces();
00080         virtual PimInterface *getInterfaceByIntID(int intID);
00081     protected:
00082         virtual void initialize(int stage);
00083         virtual void handleMessage(cMessage *);
00084 };
00085
00092 class INET_API PimInterfaceTableAccess : public ModuleAccess<PimInterfaceTable>
00093 {
00094     private:
00095         PimInterfaceTable *p;
00096     public:
00097         PimInterfaceTableAccess() : ModuleAccess<PimInterfaceTable>("
        PimInterfaceTable") {p=NULL;}
00098         virtual PimInterfaceTable *getMyIfExists()
00099         {
00100             if (!p)
00101             {
00102                 cModule *m = findModuleWherever("PimInterfaceTable",
        simulation.getContextModule());
00103                 p = dynamic_cast<PimInterfaceTable*>(m);
00104             }
00105             return p;
00106         }
00107 };
00108
00117 class addRemoveAddr : public cPolymorphic
00118 {
00119     protected:
00120         std::vector<IPAddress> addr;
00121         PimInterface *pimInt;
00122     public:
00123         addRemoveAddr(){};
00124         virtual ~addRemoveAddr(){};
00125         virtual std::string info() const
00126         {
00127             std::stringstream out;
00128             for (unsigned int i = 0; i < addr.size(); i++)
00129                 out << addr[i] << endl;
00130             return out.str();
00131         }
00132         void setAddr(std::vector<IPAddress> addr) {this->addr = addr;}
00133
00134         void setInt(PimInterface *pimInt) {this->pimInt = pimInt;}
00135         std::vector<IPAddress> getAddr () {return this->addr;}
00136         int getAddrSize () {return this->addr.size();}
00137         PimInterface *getInt () {return this->pimInt;}
00138 };
00139
00140
00141 #endif /* PIMINTERFACES_H_ */

```

7.31 F:/ANSA/src/ansa/pim/tables/PimNeighborTable.cc File Reference

File implements table of PIM neighbors.

```
#include "PimNeighborTable.h"
```

Functions

- **Define_Module** ([PimNeighborTable](#))
- `std::ostream & operator<<` (`std::ostream &os`, `const PimNeighbor &e`)

7.31.1 Detailed Description

File implements table of PIM neighbors.

Date

19.3.2012

Author

: Veronika Rybova

Table of neighbors contain information about all PIM neighbor routers which has also configured PIM protocol. Information about neighbors are obtained from Hello messages.

Definition in file [PimNeighborTable.cc](#).

7.31.2 Function Documentation

7.31.2.1 `std::ostream& operator<<` (`std::ostream &os`, `const PimNeighbor &e`)

Printout of structure Neighbor table ([PimNeighbor](#)).

Definition at line 19 of file [PimNeighborTable.cc](#).

```
{
    os << e.getId() << ": ID = " << e.getInterfaceID() << "; Addr = " << e.
        getAddr() << "; Ver = " << e.getVersion();
    return os;
};
```

7.32 PimNeighborTable.cc

```
00001
00011 #include "PimNeighborTable.h"
00012
00013 Define_Module(PimNeighborTable);
00014
00015 using namespace std;
00016
00017
00019 std::ostream& operator<<(std::ostream& os, const PimNeighbor& e)
00020 {
00021     os << e.getId() << ": ID = " << e.getInterfaceID() << "; Addr = " << e.
        getAddr() << "; Ver = " << e.getVersion();
00022     return os;
00023 };
00024
00026 std::string PimNeighbor::info() const
00027 {
00028     std::stringstream out;
00029     out << id << ": ID = " << intID << "; Addr = " << addr << "; Ver = " <<
        ver;
00030     return out.str();
00031 }
00032
```

```

00038 void PimNeighborTable::handleMessage(cMessage *msg)
00039 {
00040     opp_error("This module doesn't process messages");
00041 }
00042
00043 void PimNeighborTable::initialize(int stage)
00044 {
00045     WATCH_VECTOR(nt);
00046     id = 0;
00047 }
00048
00054 void PimNeighborTable::printPimNeighborTable()
00055 {
00056     for(std::vector<PimNeighbor>::iterator i = nt.begin(); i < nt.end(); i++)
00057     {
00058         EV << (*i).info() << endl;
00059     }
00060 }
00061
00070 std::vector<PimNeighbor> PimNeighborTable::getNeighborsByIntID(int intId)
00071 {
00072     vector<PimNeighbor> nbr;
00073
00074     for(int i = 0; i < getNumNeighbors(); i++)
00075     {
00076         if(intId == getNeighbor(i)->getInterfaceID())
00077         {
00078             nbr.push_back(*getNeighbor(i));
00079         }
00080     }
00081     return nbr;
00082 }
00083
00092 PimNeighbor *PimNeighborTable::getNeighborsByID(int id)
00093 {
00094     for(int i = 0; i < getNumNeighbors(); i++)
00095     {
00096         if(id == getNeighbor(i)->getId())
00097         {
00098             return getNeighbor(i);
00099             break;
00100         }
00101     }
00102     return NULL;
00103 }
00104
00113 bool PimNeighborTable::deleteNeighbor(int id)
00114 {
00115     for(int i = 0; i < getNumNeighbors(); i++)
00116     {
00117         if(id == getNeighbor(i)->getId())
00118         {
00119             nt.erase(nt.begin() + i);
00120             return true;
00121         }
00122     }
00123     return false;
00124 }
00125
00134 bool PimNeighborTable::isInTable(PimNeighbor entry)
00135 {
00136     for(int i = 0; i < getNumNeighbors(); i++)
00137     {
00138         if((entry.getAddr() == getNeighbor(i)->getAddr()) && (entry.
getInterfaceID() == getNeighbor(i)->getInterfaceID()))
00139             return true;
00140     }
00141     return false;
00142 }
00143
00153 PimNeighbor *PimNeighborTable::findNeighbor(int intId, IPAddress addr)
00154 {
00155     for(int i = 0; i < getNumNeighbors(); i++)
00156     {
00157         if((addr == getNeighbor(i)->getAddr()) && (intId == getNeighbor
(i)->getInterfaceID()))
00158             return getNeighbor(i);
00159     }
00160     return NULL;
00161 }
00162
00171 int PimNeighborTable::getNumNeighborsOnInt(int intId)
00172 {
00173     std::vector<PimNeighbor> neighbors = getNeighborsByIntID(intId);
00174     return neighbors.size();
00175 }

```

7.33 F:/ANSA/src/ansa/pim/tables/PimNeighborTable.h File Reference

File implements table of PIM neighbors.

```
#include <omnetpp.h>
#include "AnsaInterfaceTable.h"
#include "PIMTimer_m.h"
```

Classes

- class [PimNeighbor](#)
Class represents one entry of [PimNeighborTable](#).
- class [PimNeighborTable](#)
Class represents Pim Neighbor Table.
- class [PimNeighborTableAccess](#)
Class gives access to the [PimNeighborTable](#).

7.33.1 Detailed Description

File implements table of PIM neighbors.

Date

19.3.2012

Author

: Veronika Rybova

Table of neighbors contain information about all PIM neighbor routers which has also configured PIM protocol. Information about neighbors are obtained from Hello messages.

Definition in file [PimNeighborTable.h](#).

7.34 PimNeighborTable.h

```
00001
00011 #ifndef PIMNEIGHBOR_H_
00012 #define PIMNEIGHBOR_H_
00013
00014 #include <omnetpp.h>
00015 #include "AnsaInterfaceTable.h"
00016 #include "PIMTimer_m.h"
00017
00018
00019
00026 class INET_API PimNeighbor: public cPolymorphic
00027 {
00028     protected:
00029         int id;
00030         int intID;
00031         InterfaceEntry *intPtr;
00032         IPAddress addr;
00033         int ver;
00034         PIMnlt *nlt;
00036     public:
00037         PimNeighbor(){};
00038         virtual ~PimNeighbor(){};
00039         virtual std::string info() const;
00040
00041         // set methods
00042         void setId(int id) {this->id = id;}
00043         void setInterfaceID(int intID) {this->intID = intID;}
```

```

00044         void setInterfacePtr(InterfaceEntry *intPtr) {this->intPtr =
intPtr;}
00045         void setAddr(IPAddress addr) {this->addr = addr;}
00046         void setVersion(int ver) {this->ver = ver;}
00047         void setNlt(PIMnlt *nlt) {this->nlt = nlt;}
00050         // get methods
00051         int getId() const {return id;}
00052         int getInterfaceID() const {return intID;}
00053         InterfaceEntry *getInterfacePtr() const {return intPtr;}
00054         IPAddress getAddr() const {return addr;}
00055         int getVersion() const {return ver;}
00056         PIMnlt *getNlt() const {return nlt;}
00057     };
00058
00063     class INET_API PimNeighborTable: public cSimpleModule
00064     {
00065     protected:
00066         int id;
00067         std::vector<PimNeighbor> nt;
00069     public:
00070         PimNeighborTable(){};
00071         virtual ~PimNeighborTable(){};
00072
00073         virtual PimNeighbor *getNeighbor(int k){return &this->nt[k];}
00074         virtual void addNeighbor(PimNeighbor entry){entry.setId(id);
this->nt.push_back(entry); id++;}
00075         virtual bool deleteNeighbor(int id);
00076         virtual int getNumNeighbors() {return this->nt.size();}
00077         virtual void printPimNeighborTable();
00078         virtual std::vector<PimNeighbor> getNeighborsByIntID(int intID)
;
00079         virtual PimNeighbor *getNeighborsByID(int id);
00080         virtual int getIdCounter(){return this->id;}
00081         virtual bool isInTable(PimNeighbor entry);
00082         virtual PimNeighbor *findNeighbor(int intId, IPAddress addr);
00083         virtual int getNumNeighborsOnInt(int intId);
00084
00085     protected:
00086         virtual void initialize(int stage);
00087         virtual void handleMessage(cMessage *);
00088     };
00089
00093     class INET_API PimNeighborTableAccess : public ModuleAccess<PimNeighborTable>
00094     {
00095     public:
00096         PimNeighborTableAccess() : ModuleAccess<PimNeighborTable>("
PimNeighborTable") {}
00097     };
00098
00099     #endif /* PIMNEIGHBOR_H_ */

```