Detailed Description of Classes in OSPF Simulator

Simulator

This is the main class of the simulator which controls and runs the whole simulator application. In the main function, a single instance of this class is created as soon as user runs simulator. Most of the attributes of this class are loaded from the topology described by the user.

```
private:
  int num_Routers;
  int num_Links;
  int num_Networks;
  Router *routers;
  Link *links;
  Network *networks;
public:
  char topologyFile[256];
```

num_Routers

Total number of routers in the topology. In an active topology this number can't be less than 1.

num_Links

Total number of links in the topology. In an active topology this number can't be less than 1.

num_Networks

Total number of networks in the topology.

routers

List of routers in the topology. The length of this list should be ***num_Routers***.

links

List of links in the topology. The length of this list should be ***num_Links***.

networks

List of networks in the topology. The length of this list should be ***num_Networks.***

topologyFile[256]

A string which contains the name of the currently loaded topology file. If this string is empty then it is considered that there is no active topology.

```
private:
  int set_Routers (int index, int number, char* id);
  int set_Links (int index, int id, char typ, int id1, int id2,
      unsigned short cost);
  int set_Networks (int index, int num, char* id, char* mask);
  int getRouterIndex(int routerNumber);
  int getNetworkIndex(int networkNumber);
  int getLinkIndex(int linkNumber);
  int existRouter(int num);
  int existRouter(char* ospfid);
  int existLink(int id);
  int existNetwork(int num);
  int existNetwork(char* ospfid);
public:
  void dispRouter(char* index);
  void dispTopology();
  void dispLSDB(char* routerIndex);
  void dispRTBL(char* routerIndex);
  int read_Topology ();
  int unload_Topology();
  int verify();
  int initialize();
  int simulate();
  int out();
private:
  int addRouterLinks();
  int addLinkEndIPs();
  int countRouterLinks();
```

set_Routers, set_Links, set_Networks

Modifies/sets an entry in ***routers, links***, or ***networks*** list. The parameter "index" specifies the position of this item in the corresponding list

getRouterIndex, getLinkIndex, getAreaIndex

Return the position of a router, link or network in ***routers, links*** or ***networks*** list if its number is provided.

existRouter, existLink, existNetwork

The "string (char*)" input versions of these functions check whether there exists a router or network with the specified OSPF id (IP address), while the "int" input versions check whether there exists a router, link, or network with the specified number. ***existLink*** has no string version.

dispRouter

This function displays the detailed information including Link State Database and Routing table of the router specified. Input is the OSPF id (IP address) of the router.

dispTopology

This function displays a summary of the contents of the currently loaded topology file.

dispLSDB

This function displays the Link State Database of the router with given OSPF id (IP address).

dispRTBL

This function displays the Routing table of the router whose OSPF id is given as input.

read_Topology

Read in the topology from the topology file.

unload_Topology

Removes the current active topology from simulator. All fields containing the previous topology information are flushed.

verify

Verifies topology for correctness; looks for logical consistency in different parameter values.

initialize

Performs initialization phase of OSPF.

simulate

Performs flooding, shortest path calculation and routing table calculation phase of OSPF

addRouterLinks()

Pass references of each router's links to its ***Router*** instance for performing OSPF operations.

addLinkIPs

Identifies the OSPF id of the network or router attached to each link and place this information into that ***Link*** instance.

countRouterLinks()

Identifies and counts the links attached to each router.

Link

This class describes an OSPF link. All the links kept in Simulator and Router class are instances of this class. Every point to point link between two routers is specified as two independent entries in the topology file and hence two independent instances are created each belonging to one of the end routers.

```
private:
  int id;
  int end1;
  char type;
  int end2;
  unsigned short cost;
  IP end1IP;
  IP end2IP;
  IP end2Mask;
```

id

A unique number that identifies the link instance inside simulator.

end1

Router number to which this link belongs.

type

Type of the link. Links can be of two types (in this simulator): Point to Point i.e. between two routers or to a Stub Network. ***P2P*** and ***SN*** are the constants used to represent these types in this simulator.

end2

Network number or router number of the network/router on the other end of the link.

cost

cost of the link

end1IP

OSPF id of the router mentioned as ***end1***.

end2IP

OSPF id of the router/network mentioned as ***end2***.

end2Mask

If it is a SN link, then end2 is a network having some network mask. This mask is stored in this member.

```
public:
  void set(int ownId, char typ, int id1, int id2, unsigned short
      lCost);
  void setEnds(IP e1, IP e2);
  void setEnds(IP e1, IP e2, IP e3);
  int getID();
  int getRouter();
  char getType();
  int getNet_Router();
  unsigned short getCost();
  IP& getEnd1();
  IP& getEnd2();
  IP& getEnd2Mask();
```

set

Called first time when this object is created to set main attributes of this object.

setEnds

Sets the OSPF id of both the ends of this link. The three parameter version is used when it is an SN and the third parameter specifies the network mask.

getID, getRoute, getType, getNet_Router, getCost, getEnd1, getEnd2, getEnd2Mask

Get functions for different attributes

Network

This class describes a network in the topology.

```
private:
  int number;
  IP id;
  IP mask;
```

number

A unique identifier assigned to this network in simulator.

id

The IP address of the network.

mask

The subnet mask of the network.

```
public:
  int set(int num, char* nid, char* nmask);
  int getNumber();
  IP& getID();
  IP& getMask();
```

set

Called first time when this object is created to set main attributes of this object

getNumber, getID() getMask()

Get functions for different attributes.

Router

This class describes an OSPF router.

```
private:
  int number;
  IP id;
  int num_Links;
  Link **links;
  OSPF_Packet_Queue inQ;
  OSPF_Packet_Queue outQ;
  int routingTableSize;
  Routing_Table rt;
  LSDatabase lsdb;
  SP_Tree spt;
```

number

A unique identifier assigned to this router in OSPFSIM

id

The IP address of this router

num_Links

Number of links of this router

links

List of links attached to this router

inQ

Contains OSPF packets received from other routers.

outQ

Contains OSPF packets generated to transmit to other routers.

routingTableSize

Number of entries in routing table.

rt

The routing table of this router.

lsdb

Link State Database of this router. The OSPF routers have an independent LSDB for each area to which they are connected, but in this simulator we have no concept of areas; therefore, each router has only one LSDB.

spt

Just like LSDBs, OSPF routers have independent shortest path trees for each area to which they are connected. Since there are no areas in this simulator, so again each router has only one SPT.

```
public:
  int getNumber();
  IP& getID();
  int set(int num, char* ospfid);
  int setNumLinks(int count);
  int getPacketsToSendCount();
  OSPF_Packet* getPacketToSend();
  int packetReceived(OSPF_Packet* packet);
  int initialize();
  int flood();
private:
  int calculateSPT();
  int calculateRoutingTable();
public:
  int addLink(Link* link);
  void display(FILE* outFile);
  void dispLSDB(FILE* outFile);
  void dispRTBL(FILE* outFile);
```

getNumber, getID

Get functions for different attributes

set

Called in the beginning when this object is created to set main attributes for this object

setNumLinks

Set the value of ***num_Links***.

getPacketsToSendCount

Returns the length of ***outQ***

getPacketToSend

De-queues a packet from ***outQ***

packetReceived

En-queues a packet in ***inQ.***

initialize

Performs initialization of this router during OSPF initialization phase *flood*

Performs the flood phase of OSPF for this router.

calculateSPFT

Calculates shortest path tree for this router.

calculateRoutingTable

Calculates routing table for this router.

addLink

Adds a link to this router's list of links.

display

Displays attributes of this router including its Link State Database and Routing Table.

displayLSDB

Displays Link State Database of this router.

displayRTBL

Displays Routing Table of this router.

LSA_Header

All LSA's begin with a common 20 byte header. This header contains enough information to uniquely identify the LSA (LS type, Link State ID, and Advertising Router). This class is a representation of LSA header as per appendix A.4.1 of the RFC 2328.

```
private:
  unsigned short ls_Age;
  char options;
  char ls_Type;
  IP link_State_ID;
  IP advertising_Router;
  int ls_SequenceNumber;
  short ls_CheckSum;
  unsigned short ls_Length;
```

ls_Age

The time in seconds since the LSA was originated. This field is always set to zero as aging is not implemented in this simulator.

options

The optional capabilities supported by the described portion of the routing domain. Always set to zero.

ls_Type

The type of the LSA. Each LSA type has a separate advertisement format.

link_State_ID

This field identifies the portion of the internet environment that is being described by the LSA. The contents of this field depend on the LSA's LS type. In Router-LSAs the Link State ID is set to the IP interface address of the router attached to this link.

advertising_Router

The Router ID of the router that has originated this LSA.

ls_SequenceNumber

Detects old or duplicate LSAs. Successive instances of an LSA are given successive LS sequence numbers. This field is always set to zero due to the non-availability of this feature in our simulator.

ls_CheckSum

The Fletcher checksum of the complete contents of the LSA, including the LSA header but excluding the LS age field. Also set to zero as this simulator doesn't support any checksums.

ls_Length

The length in bytes of the LSA. This includes the 20 byte LSA header.

```
public:
  int set(char type, IP lsID, IP advRouter);
  int set(char type);
  int setLength(unsigned short length);
  unsigned short getLength();
  IP& getLSID();
  void display(FILE* outFile);
```

set

Called in the beginning when this object is created to set main attributes for this object

setLength, getLength, getLSID

Get/set functions for different attributes.

display

Displays attributes of this header.

LSA_Base

This is the base class for different types of LSAs. However, only Router_LSAs are implemented in this simulator.

```
protected:
  LSA_Header lsa_Header;
```

lsa_Header

The header of this LSA

```
public:
  IP& getLSID();
  void displayHeader(FILE* outFile);
```

getLSID

Get function for ***link_State_ID*** of the header of this LSA.

displayHeader

Displays attributes of the header of this LSA.

Router_LSA : public LSA_Base

Router-LSAs are the Type 1 LSAs. Each router in an area originates a router-LSA. The LSA describes the state and cost of the router's links (i.e., interfaces) to the area. All of the router's links to the area must be described in a single router-LSA. This class is an implementation of appendix A.4.2 of RFC 2328.

```
private:
  char optionsVEB;
  char optionsFixed;
  unsigned short num_Links;
  Link_Data *links;
```

optionsVEB

Specifies V, E and B bits for representing the type of the router. Always set to '\0' in this simulator.

optionsFixed

Always '\0' in OSPF.

num_Links

The number of router links described in this LSA

links

A list of links used to describe each router link

```
public:
  Link_Data getLink(int index);
  unsigned short getNumLinks();
  int addLink(IP id, IP data, char type, unsigned short metric);
  void display(FILE* outFile);
```

getLink

Returns the details of a link from the list of links, with the specified index.

getNumLinks

Get function for ***num_Links***.

addLink

Adds a link to this LSA

display

Displays the contents of this LSA.

Link_Data

This class describes each router link stated in a Router LSA. Optional TOS fields are not added in this description.

```
private:
  IP link_ID;
  IP link_Data;
  char type;
  char tosNumber;
  unsigned short metric;
```

link_ID

Identification of the object on the other end of this link. The value depends upon the link's Type. For connections to stub networks, it specifies the network's IP address mask, while in case of point to point router links, it is the IP address of the other end router.

link_Data

This field also depends upon the Type field. For connections to stub networks, it specifies the network's IP address mask.

type

Type of this router link. Only point to point or stub network links are supported in this simulator.

tosNumber

The number of different TOS metrics given for this link, not counting the required link metric. This field is set to 0 in this simulator.

metric

The cost of using this router link.

```
public:
  int set(IP id, IP data, char typ, unsigned short cost);
  IP getID();
  IP getData();
  char getType();
  char getTOS();
  unsigned short getMetric();
  void display(FILE* outFile);
```

set

Called in the beginning when this object is created to set main attributes for this object

getID, getData, getType, getTOS, getMetric

Get/set functions for different attributes

display

Displays the attributes of this link.

LSDatabase

This class describes the Link State Database. Originally in OSPF, every router has an independent Link State Database for each Area. However, in this simulator every router has only one Link State Database. Furthermore, this class contains only router LSAs as other types of LSAs are not supported.

```
private:
  int rLSAs;
  Router_LSA *router_LSAs;
```

rLSAs

Number of router LSAs present in this Link State Database

router_LSAs

Lists of router LSAs present in this database

```
public:
  int getnRLSA();
  int existRouterLSA(IP lsID);
  Router_LSA getRouterLSA(int index);
  Router_LSA getRouterLSA(IP lsID);
  int addRLSALink(IP lsID, IP linkID, IP linkData, char type, short
      cost);
  int addRouterLSA(Router_LSA rlsa);
  int updateRouterLSA(Router_LSA rlsa);
  void display(FILE* outFile);
```

getnRLSA

Returns number of router LSAs stored

existRLSA

Verifies whether a router LSA with specified link state ID present in the database.

getRouterLSA

Returns router LSA specified by index or link state ID.

addRLSALink

Adds link to already existing router LSA with specified link state ID.

addRouterLSA

Adds a new router LSA.

updateRouterLSA

Updates the contents of an existing router LSA with same link state ID; the new contents are passed in the form of a Router_LSA.

display

Displays all the stored LSAs.

OSPF_Header

This class is an implementation of the appendix A.3.1 of RFC 2328. Every OSPF packet has with a standard 24 byte header. This header contains all the information necessary to determine whether the packet should be accepted for further processing or not.

```
private:
  char version;
  char type;
  unsigned short packet_Length;
  IP router_ID;
  IP area_ID;
  short checksum;
  short au_Type;
  int authentication1;
  int authentication2;
```

version

The OSPF version number (always set to 2).

type

The OSPF packet type.

packet_Length

The length of the OSPF protocol packet in bytes. This length includes the standard OSPF header.

router_ID

The Router ID of the packet's source.

area_ID

A 32 bit number identifying the area to which this packet belongs. All OSPF packets are associated with a single area. This value is always set to 0.0.0.0 in this simulator.

checksum

The standard IP checksum of the entire contents of the packet, starting with the OSPF packet header but excluding the 64-bit authentication field. This field is always set to 0.

au_Type

Identifies the authentication procedure to be used for the packet. Always 0 in this implementation.

authentication1, authentication2

A 64-bit field for use by the authentication scheme. Both are not used in this simulator and always have value 0.

```
public:
  IP getRouter();
  int set(char typ, unsigned short length, IP router);
  int set(char typ, unsigned short length);
  unsigned short getLength();
  int setLength(unsigned short l);
  char getType();
```

set

Called first time when this object is created to set main attributes for this object *getRouter, getLength, setLength, getArea*

Get/set functions for different attributes.

OSPF_Packet_Base

This is the base class for different types of OSPF packet classes. However, only Link State Update Packets are implemented in this simulator.

```
protected:
  OSPF_Header ospf_Header;
```

ospf_Header

The header of this OSPF packet

```
public:
  char getType();
```

set

Called first time when this object is created to set main attributes for this object

getType

Returns type of this packet.

struct OSPF_Packet

This structure is used as a queue element for exchanging OSPF packets among routers.

```
  OSPF_Packet_Base* packet;
  int linkID;
```

packet

The OSPF packet to send.

linkID

The link on which this packet should be sent.

OSPF_Packet_Queue

This class defines incoming and outgoing OSPF packet queues of routers. These queues basically contain list of *OSPF_Packet* elements.

```
private:
  int qLength;
  OSPF_Packet **packets;
```

qLength

The length of this queue i.e. the number of packets currently present in the queue.

packets

List of OSPF packets.

```
public:
  int getLength();
  int enqueue(OSPF_Packet* packet);
  OSPF_Packet* dequeue();
```

getLength

Returns length of this queue

enqueue

Adds a packet to this queue.

dequeue

Removes a packet from this queue.

Link_State_Update_Packet : public OSPF_Packet_Base

Described in appendix A.3.5 of RFC 2328. These packets are used to flood the LSAs. Each Link State Update packet carries a collection of LSAs one hop further from their origin. Several LSAs may be included in a single packet.

```
private:
  int num_rLSA;
  Router_LSA *rLSAs;
```

num_rLSA

Number of router LSAs present

rLSAs

Lists of router LSAs present.

```
public:
  int getnRLSA();
  Router_LSA getRLSA(int index);
  int addRLSA(Router_LSA rlsa);
```

getnRLSA

Get function for ***num_rLSA***.

getRLSA

Returns router LSA with specified index from the list of stored LSAs.

addRLSA

Adds router LSA to this Link State Update Packet.

IP

This class provides definition of a 32 bit IP address and means to work on it.

```
private:
  unsigned char octets[4];
```

octets

The four octets of an IPv4 address

```
public:
  void set(char *id);
  void setOctet(int index, unsigned char oc);
  unsigned char getOctet(int index);
```

set

Takes a string in "x.x.x.x" format and converts it to IPv4 address.

setOctet, getOctet

Get/set functions for ***octets***.

Routing_Table

This class represents an OSPF Routing Table (section 11 of RFC 2328). It contains multiple instances of ***rtEntry*** structure each representing an entry. As OSPF supports multiple equal cost paths, therefore, each routing table entry contains list of all the best paths to a particular destination.

```
private:
  int size;
  rtEntry* entries;
```

size

Number of routing entries.

entries

List of all routing entries.

```
public:
  int addRoute(char dT, IP dID, IP aM);
  int addRoute(char dT, IP dID);
  int addPath(IP dID, unsigned short cost, IP next);
  char getDType(IP dID);
  IP getDID(int index);
  IP getAM(IP dID);
  int getPathCount(IP dID);
  unsigned short getCost(IP dID, int index);
  IP getNH(IP dID, int index);
  char getPathType(IP dID, int index);
  int getSize();
  int initialize();
```

addRoute

Calls first time when a new entry is created. The three-parameters version is for networks (with subnet mask) while two-parameters version is for routers.

addPath

Adds a path to already added destination.

getDType, getDID, getAM, getPathCount, getCost, getNH, getPathType,getSize()

Get/set functions for different attributes.

initialize

Initializes the routing table. Size field is set to zero and entries is set to null.

struct rtEntry

This structure represents an individual entry in the routing table.

```
private:
  char destination_Type;
  IP destination_ID;
  IP address_Mask;
  int pathCount;
  path* paths;
```

destination_Type

Destination type is either network “N” or router “R”.

destination_ID

The destination’s identifier or name. This depends on the Destination Type. For networks, the identifier is their associated IP address. For routers, the identifier is the OSPF Router ID.

address_Mask

Only defined for networks. The network’s IP address together with its address mask defines a range of IP addresses.

path_Count

Number of different equal cost paths to this destination.

paths

The list of different paths.

struct path

This structure describes the details of each individual path to the destination.

```
private:
  char path_Type;//always set to "INTRA_AREA"
  unsigned short cost;
  IP next_Hop;
```

path_Type

Type of this path. Only “INTRA_AREA” is supported in this simulator.

cost

The link state cost of the path to the destination.

next_Hop

The IP address of the next hop router to the destination.

SP_Tree

This class describes the shortest path tree instance. Originally, OSPF routers have independent shortest path tree for every Area, but in this simulator, every router has only one such tree. This class is not implemented so that the users can have an implementation according to their ease.

Commands supported at simulator command line

This section describes the syntax of different commands that can be used at simulator's command line to perform different operations.

```
load  topology_file
```

This command loads the topology described in the file name specified.

```
unload
```

This command unloads the active topology.

```
exit
```

Exits from OSPFSIM and returns to OS

```
run
```

Starts OSPF simulation on active topology. The output (router Link State Databases and Routing Tables) is dumped to a file having same name as topology file but with additional extension ***. out***

```
display     topology
```

Displays a summary of the topology file contents.

```
display     router {number}
```

Displays a specified (number as in topology file) router's Link State Database and routing table.

```
display     lsdb {number}
```

Displays a specified (number as in topology file) router's Link State Database only.

```
display     rtable {number}
```

Displays a specified (number as in topology file) router's routing table only.