Technische Universität Braunschweig

Institut für Betriebssysteme und Rechnerverbund

Kommunikation und Multimedia

Prof. Dr.-Ing. Lars Wolf

Praktikum Computernetze im WS0910

- Task Description -

Betreuer: Felix Büsching, Tobias Pögel

Aufgabe: Network Simulator Programming-OSPF Routing Protocol

1. Objective

This task/assignment is about implementing the basic routing operations performed by OSPF routers. The scope of this task is limited to, the initialization of link state databases inside routers, the exchange of link state information among routers and finally the calculation of shortest paths and routes as per OSPF specifications. In order to keep the task simple, the simulator provided for this task simulates a very limited version of OSPF routing protocol (explained in the next section).

We have eliminated some of the functions from the simulator source such as, sharing link state information and, calculation of shortest paths and routes from the simulator. Students are required to re-write these functions so that the simulator eventually works according to the OSPF specifications. While writing these missing functions, a very limited access will be provided to the rest of the components of the simulator.

2. OSPF features supported/not-supported by the simulator

The simulator provided in this assignment supports very limited but core functionality of the OSPF routing protocol. Most of the optional and more technical components are eliminated from the scope of this assignment to keep it simple for the students. Following are the OSPF features which are not available in this simulator:

- ***Division of topology into Areas***. The topology is considered as one single area in this simulator. Consequently, all the features which are based on the concept of multiple areas are also not present. For example, individual ***Link State Database*** and ***Shortest Path Tree*** for each area are not available; every router has only one instance of these. Similarly, in OSPF

packets and ***Link State Advertisements***, wherever there is an option to mention Area ID, it is hard coded and user has been provided no control/access to it. ***Link State Advertisements*** of type ***Network***, ***Summary*** and ***AS-External*** are not implemented.

- ***Transit Networks***. In OSPF, a network attached to more than one routers is called ***transit network***. This simulator doesn't support existence of such nodes in the topology. As a result, the related concepts of ***Neighbor Router*** and ***Designated Router*** are also not provided.
- ***Hello Protocol***. This part of OSPF functionality is responsible for identifying Neighbor Routers and then electing Designated Router. As there is no support for ***Neighbor*** and ***Designated*** routers in this simulator, the ***Hello Protocol*** and ***Hello Packet*** are also missing.
- ***Bringing up Adjacencies***. During this operation, ***Neighbor Routers*** share ***Database Description Packets*** with each other to get initial information about their links and synchronize their ***Link State Databases***. Since the concept of ***Neighbor Routers*** is not present in our simulator, the process of ***Bringing up Adjacencies*** and the related items such as ***Database Description Packets***, ***Link State Request Packets*** etc. are also not provided. However, the ***initialize*** method is provided in Routers to identify their link status and initialize their ***Link State Databases*** before they start ***flooding***.
- ***Transmission Error, Acknowledgements, Checksums, Type of Service, Timers, Aging, Sequence Numbers***, all these concepts are eliminated to reduce the complications. ***Link State Acknowledgment Packets*** are not required any more. The verification of age and validity at different processes is also not required.

The supported OSPF features and operations which our simulator performs on behalf of the user are following:

- Reading the topology from file and initialize simulator data structures. This operation is performed in response to "***load filename***" command.
- Verification of ambiguities in the topology. This process is performed as a first step in response to "***run***" command so that there shouldn't be any complications/errors in the later operations.
- Initialization of Router data structures. This is the first step of OSPF functionality (executed after verification). Routers are provided references to their links. A call is made to the ***initialize*** method of every router so that necessary steps can be performed.
- The OSPF flooding. Repeated calls to the ***flood*** method of routers are performed. After every call, simulator looks for any available OSPF packets to exchange. The ***flood*** method is not called anymore if there are no packets to exchange.
- Data structures for ***OSPF packet header***, ***Link State Update Packet***, ***LSA header***, ***Router LSA***, ***Routing Table***, ***Shortest Path Tree***, and ***Link State Database*** are implemented in the simulator.
- This simulator also supports equal cost multiple paths feature of OSPF.

3. Simulator Structure

The OSPF simulator is designed solely in C++ under Linux environment; a windows as well as OS/X version is also available. It contains classes for different components to simulate an OSPF

instance inside a single router as well as to integrate these router instances so that they can exchange OSPF related information with each other.

These classes are described in different header files (.h) provided in the source code package. The class diagrams in this document show how these classes are interrelated in the simulator. All these classes are briefly described in this document; a detailed description of these classes is independently given in the document "PCn-OSPF-class-description.pdf":

Fig 1: Basic Simulator Structure

main (main.{h | cpp})

These two files contain the ***main*** function and some other global functions and variables. None of these is really important in this assignment. In the ***main*** function, an instance of ***Simulator*** is

created and a command prompt is displayed to accept input from the user. The relevant functions of ***Simulator*** are called to fulfill user commands. A complete list of available commands is provided in "PCn-OSPF-class-description.pdf".

Simulator (Simulator.{h | cpp})
This class represents the simulator instance. A simulator instance contains routers, links and networks in the topology. Data members of a simulator instance are populated in response to user commands. For example, ***"load filename"*** assigns topology file name to the ***topologyFile*** member. Details of routers, links and networks are loaded into the members such as ***routers, links, networks*** etc., from the topology file in response to ***"run"*** command.

Router (Router.{h | cpp})
This class represents an OSPF router. It contains all necessary components of an OSPF router like Link State Database, Shortest Path Tree, Routing table and queues for incoming and outgoing OSPF packets. As this implementation of OSPF doesn't contain the concept of areas, each router has only one Link State Database. Similarly, there is only one instance of Shortest Path Tree in each router. A ***router*** instance also contains references to its links. During the execution of ***run*** command, before starting core OSPF operations, ***simulator*** instance passes the references of every router's links to its instance.

Network (Network.{h | cpp})
This class represents a network. Only OSPF "***stub networks***" are implemented in this simulator.

Link (Link.{h | cpp})
This class represents a link. Only two types of OSPF links i.e., "***Point to Point***" and "***Stub Network***" are implemented.

LSDatabase (LSDatabase.{h | cpp})
This class represents a Link State Database. Since only Router LSAs are implemented in this simulator, every instance of ***LSDatabase*** contains only Router LSAs.

SP_Tree (SP_Tree.h)
This class represents an OSPF shortest path tree. In OSPF, a router has an independent shortest path tree for every area, but this simulator doesn't support multiple areas; therefore, there will be only one instance of ***SP_Tree*** in every router. Furthermore, this class is provided blank to the students in this praktikum. This implies that the students are free to design this class according to their choice and ease.

Routing_Table (Routing_Table.{h | cpp})
This class describes the Routing Table of each router. Entries are defined as a structure (***rtEntry***) and every instance of this class contains multiple instances of this structure. As OSPF supports multiple paths, therefore every entry contains more than one path (***struct path***).

OSPF_Packet_Queue (OSPF_Packet_Queue.{h | cpp})
Every router exchanges routing information with its neighbors through OSPF packets. For this purpose, two queues are implemented in each router. The ***inQ*** contains packets sent by the other routers to this router, while the ***outQ*** contains packets to be sent by this router. During the execution of OSPF functionality, simulator takes packets from the ***outQs*** of every router and put them in the ***inQs*** of relevant routers.

OSPF_Header (OSPF_Header.{h | cpp})
All types of OSPF packets carry a similar header. This class describes this header and related functionality.

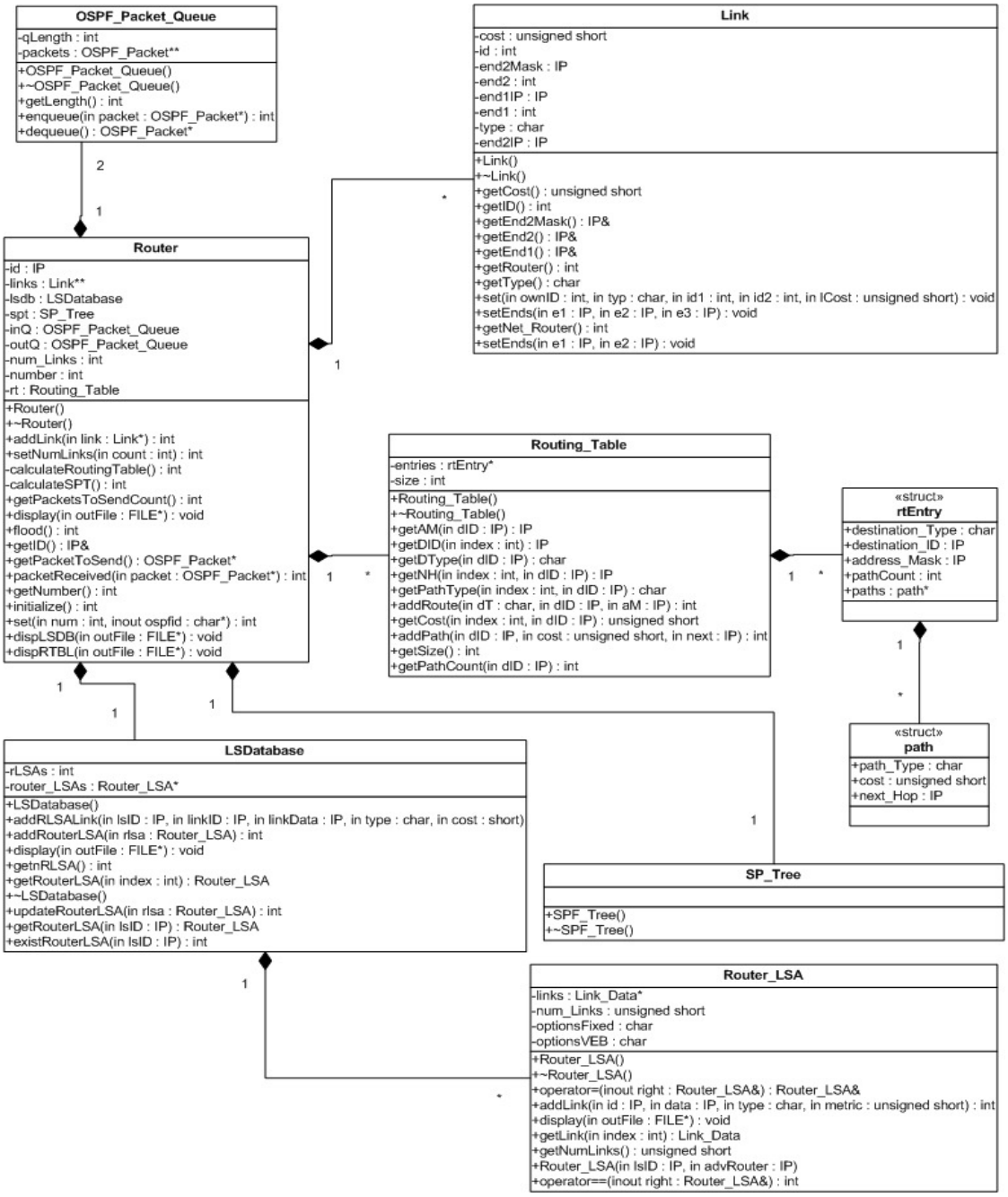

Fig 2: Router and related classes

OSPF_Packet_Base (OSPF_Packet_Base.{h | cpp})

OSPF has five types of packets for routing operations. Although this simulator implements only one of them, but still parent-child class structure is provided for future extension. This class contains the features common to all OSPF packet types such as header.

OSPF_Packet (OSPF_Packet_Queue.{h | cpp})

This structure describes an individual element in the ***OSPF_Packet_Queue***.

Link_State_Update_Packet (Link_State_Update_Packet.{h | cpp})

This class describes the only OSPF packet type implemented in this simulator. Link State Update packets are used by OSPF routers to advertise their link states to other routers. This class provides all the relevant OSPF functionality.

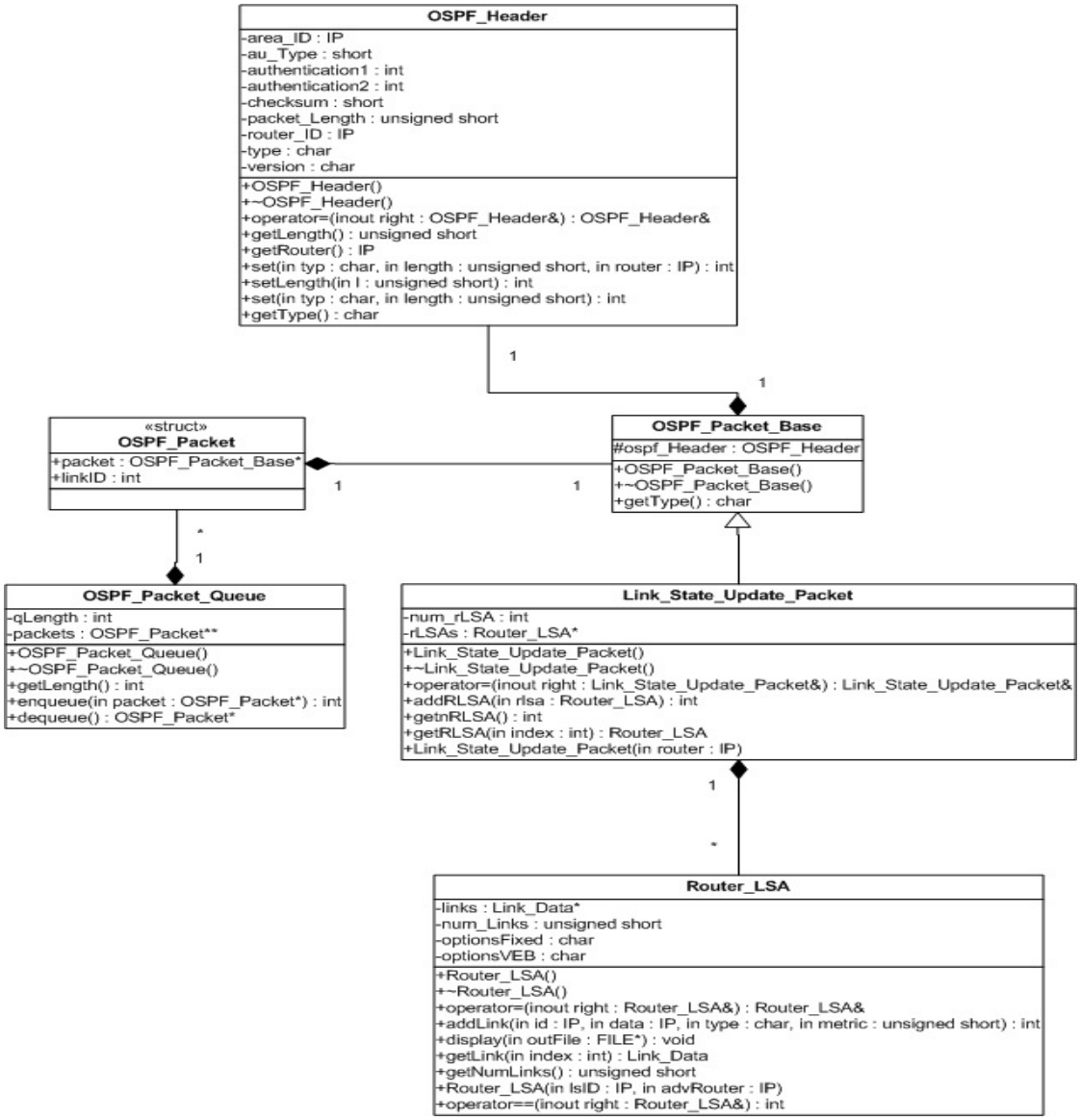

Fig 3: OSPF Packet hierarchy in the simulator

LSA_Header (LSA_Header.{h | cpp})

OSPF has four different types of Link State Advertisements (LSA). All these LSA types have a similar header. This class describes that common header available in all those LSAs and the required functionality.

LSA_Base (LSA_Base.{h | cpp})

This simulator includes implementation of only one of the LSA types i.e., router LSAs, but still parent-child class structure is provided for future extension. This class contains the features common to all OSPF LSA types such as header.

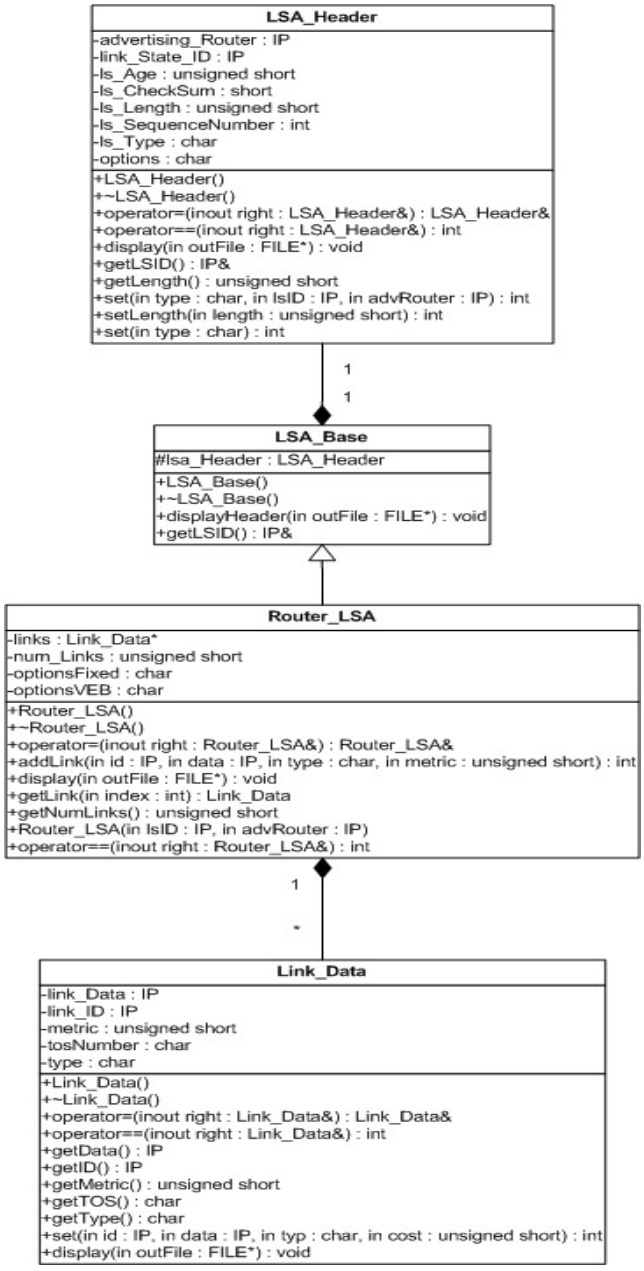

Fig 4: Link State Advertisement hierarchy in the simulator

Router_LSA (Router_LSA.{h/cpp})

OSPF routers share their link states in the form of router LSAs. Every router keeps its own router LSAs and the router LSAs received from the other routers in its Link State Database. These LSAs

are packed into Link State Update Packet during the advertisement process. This class describes an OSPF Link State Advertisement of type "***Router LSA***" and related functionality.

Link_Data (Link_Data.{h | cpp})
Every router LSA contains information about all the links of the originating router of this LSA. This information is provided inside an instance of ***Router_LSA*** as multiple instances of ***Link_Data*** class.

IP (IP.{h/cpp})
This is a utility class which basically represents an IP address and provides necessary operations to work with an IP address.

4. How it works

This section will describe the sequence of operations inside the simulator to provide an understanding about its working. A sequence diagram about the main operations of the simulator is given below to enhance the clarity.

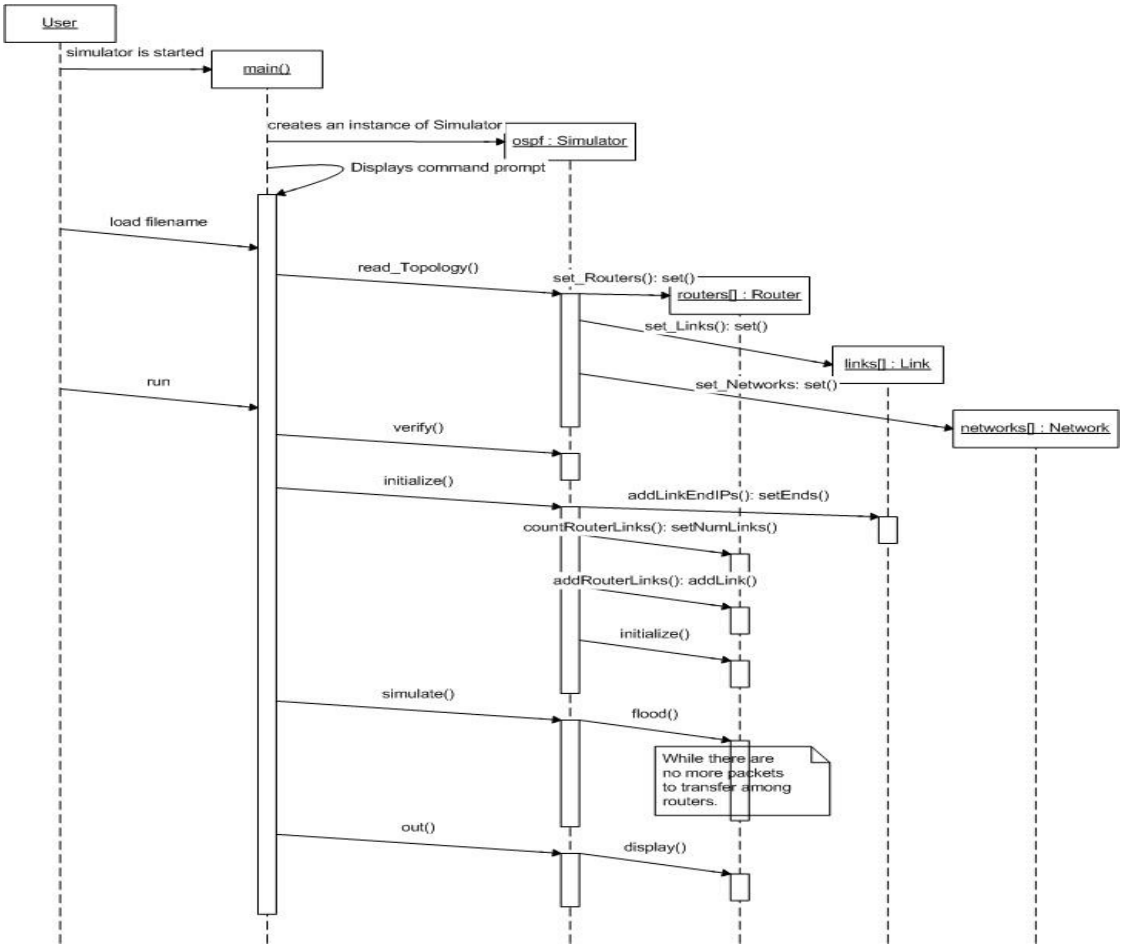

Fig 5: Sequence diagram showing sequence of operations inside "load" and "run" command

The simulator executable is named ***ospf.o*** and when invoked it displays a command prompt as following:

It also creates an instance of ***Simulator*** to perform further operations. A complete list of available commands and their output is given in a later section. The two commands "***load***" and "***run***" are important as these provide the core functionality of OSPF. In this section, the sequence of operations performed by simulator in response to these two commands will be described.

load filename

This command directs the simulator to load the network topology given in the file "***filenam***e". In response, the name of topology file is assigned to the ***topologyFile*** member of the ***Simulator*** instance and ***read_Topology*** method is invoked. This method reads the file line by line and identifies different parameters from the file. The class members such as ***routers***, ***links*** and ***networks*** are initiated, then descriptions of individual routers, links or networks are assigned to the relevant instances after parsing (using ***set_Routers*** or ***set_Links*** or ***set_Networks***) and values are assigned to ***routers***, ***links*** and ***networks*** elements by calling ***set*** methods of these classes.

run

This command invokes the sequence of operations to simulate the OPSF behavior on the topology loaded previously.

First of all, the ***verify*** method of the ***Simulator*** instance is called to verify any ambiguities in the topology such as invalid or duplicate addresses etc.

In the second step, the ***initialize*** method is called. This method copies the IP addresses of link ends to each ***Link*** instance (required in later operations) by calling ***addLinkEndIPs***, which eventually calls ***setEnds*** of each ***Link***. Then, ***countRouterLinks*** is called which identifies the links of each router; however, this function just sets the number of the links in each router by calling ***setNumLinks***. The link references are later passed by ***addRouterLinks*** to each ***Router*** instance by calling its ***addLink*** method. Finally, the ***initialize*** of each ***Router*** instance is called.

In third phase, the OSPF flooding is performed multiple times. In every iteration, first the ***flood*** method of every ***Router*** instance is called. After that, ***inQs*** of all the ***Router*** instances are checked. If there are any packets, then these packets are transferred in the ***outQs*** of the relevant ***Router*** instances. This process is not repeated anymore if all the ***inQs*** are found empty at the end of the iteration, indicating that the routers do not have new information to share and the topology is stable.

Finally, the ***out*** method is called. This method writes the status of all ***Router*** instances to a file "***Routing_Table.txt***". The ***out*** method calls ***display*** method of each ***Router*** instance which writes Link State Database and routing table to the output file.

5. Your task

The simulator provided for this assignment is fully functional (as per exceptions mentioned earlier). However, we eliminated the code from ***initialize***, ***flood***, ***calcualteSPT*** and ***calcualteRoutingTable*** methods of the ***Router*** class. Furthermore, the ***SP_Tree*** class is provided without implementation. Students have to implement these methods following the OSPF protocol specifications (only for those features which are originally supported by the simulator). However, during this implementation, they also have to take care for the following rules:

- They will be provided with following header files only:

main.h, Simulator.h, Router.h, Link.h, Network.h, IP.h, LSDatabase.h, SP_Tree.h, Routing_Table.h, Link_Data.h, OSPF_Header.h, OSPF_Packet_Queue.h, OSPF_Packet_Base.h, Link_State_Update_Packet.h, LSA_Base.h, LSA_Header.h, Router_LSA.h

The respective ***.cpp*** files containing the implementation of functions defined in these files will be provided in compiled ***(.o)*** form. However, ***Router.o*** will not have the implementation of ***initialize***, ***flood***, ***calcualteSPT*** and ***calcualteRoutingTable***. The blank definitions of these functions will be provided in an independent file ***ospf.cpp***.

- Students have to write their code only in this ***ospf.cpp*** file. They cannot add any functions or data members to any of the existing classes except the ***Router*** and the ***SP_Tree***. They can create additional classes if required.
- Few test topologies with their output on original simulator are provided so that students can verify the correctness of their code. These files contain the output generated in response to ***run*** command.
- For compiling the simulator code with their additions, a script file ***compile-ospf*** is also provided. Students have to simply run the command ***./compile-ospf*** from OS/X command prompt. This will generate an executable with name ***ospf***.
- Students have to submit their version of ***Router.h, SP_Tree.h*** and ***ospf.cpp.***

6. Reading Material helpful in this assignment

- The main reading source for this assignment is the OSPF version 2 RFC 2328. A copy of this RFC is provided on the course website. Required sections are already marked for convenience. Few sections are beyond the scope of this assignment but are highlighted so that the student's can have enough background knowledge about the task requirements.
- Many major books on networking provide brief introduction on OSPF's working. That can also be helpful to have a basic understanding about it. One good option can be "Routing in Internet" by Christian Huitema.