Technische Universität Braunschweig

Institut für Betriebssysteme und Rechnerverbund

Kommunikation und Multimedia

Prof. Dr. L. Wolf

Praktikum Kommunikationssysteme im SS06

Betreuer: Zefir Kurtisi, Habib-ur-Rehman, Alexej Beresnev

– Task Description –

Aufgabe 3: Simulator Programming-OSPF Routing Protocol

Termine

Konzept 06.07.06

Implementation 27.07.06

1. Background

Simulators are a crucial part of research, and particularly in networks area a major part of research is analyzed through simulators before actual implementation. Simulators are designed to give a virtual view of the actual real life scenario (not always exactly the same), so that the research outcomes can be evaluated and analyzed beforehand.

Another aspect of this story is that most of the time, a researcher is working on a relatively small component in the network domain but the performance of this component is dependent on many other components or can cause effects to many other components. Hence, for the true analysis of this component, the whole picture should be painted. For example, a change in the functionality of MAC layer can affect the behavior of layers above and below it. On the other hand, the performance of this change is also largely dependent on the behavior of other layers. Thus, during the analysis of this change in MAC layer, we require a complete implementation of all the layers rather than just the component which is being changed.

It is not a simple and easy job to design an application which simulates all the components of Network domain. One important aspect to keep in view while designing such simulators is to give the users adequate facilities to put in their components for analysis purpose. Research groups did joint efforts over the years to come up with some solutions which can help the fellow researchers in their work. NS (Network Simulator) is the most popular and widely used open source and free-license simulator designed with the collaborative effort of some US based agencies, research

organizations and universities. OPNET is another example. It contains a complete family of quality network R&D tools but unfortunately unlike NS it is a commercial product.

2. Objective

This task/assignment is added in this course to give the students a small experience of working with simulators. The major task in implementing some component into an existing full scale simulator (like NS or OPNET) for analysis purpose is, to understand the structure of simulator, identify the right location to place your own component and then make the integration of this component with the simulator functional.

In this assignment a student has to achieve the same objective, that is, implement a given protocol inside an existing simulator while being compliant with the restrictions of the simulator. We have designed a limited feature OSPF simulator for this purpose which simulates the OSPF operations performed by a router. For the assignment purpose, we eliminated some of the functions from this simulator and students are required to re-write those functions so that the simulator eventually works according to the OSPF specifications. And of course while writing that code; there will be very limited access to the rest of the components of the simulator.

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

The simulator provided in this assignment supports very limited but core functionality of OSPF. 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. Now 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 provided no control to access 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, so ***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***. This process, ***Database Description Packets*** and ***Link State Request Packets*** (required to demand updated link state information from neighbors) are not provided as there are no ***Neighbor Routers***. However, ***initialize*** method is provided in Routers to identify their link status and initialize their ***Link State Databases*** before starting ***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. Similarly, the verification of age and validity at different processes is needed not to consider.

The supported OSPF features and operations which simulator performs on behalf of 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 in the later operations.
- Initialization of Router data structures. This is the first step of OSPF functionality simulation (executed after verification). Routers are provided references to their links. A call is made to ***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 OSPF packets to send and forward them to relevant routers.
- 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.

4. Simulator Structure

The OSPF simulator is designed solely in C++ under Linux environment. 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. The class diagrams ahead show how these classes are interrelated in the simulator. All these classes are briefly described here while a detailed description of these classes is independently given in the document "OSPF-class-description":

main (main.h, main.cpp)

Basically these two files contain the ***main*** function and some other global functions and variables. None of these are 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 user. The relevant functions of ***Simulator*** are called to fulfill user commands. A complete list of available commands is provided in "OSPF-class-description".

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***. Details of routers, links and networks are loaded into ***routers, links, networks*** etc., from 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, therefore 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 references of every router's links to its instance.

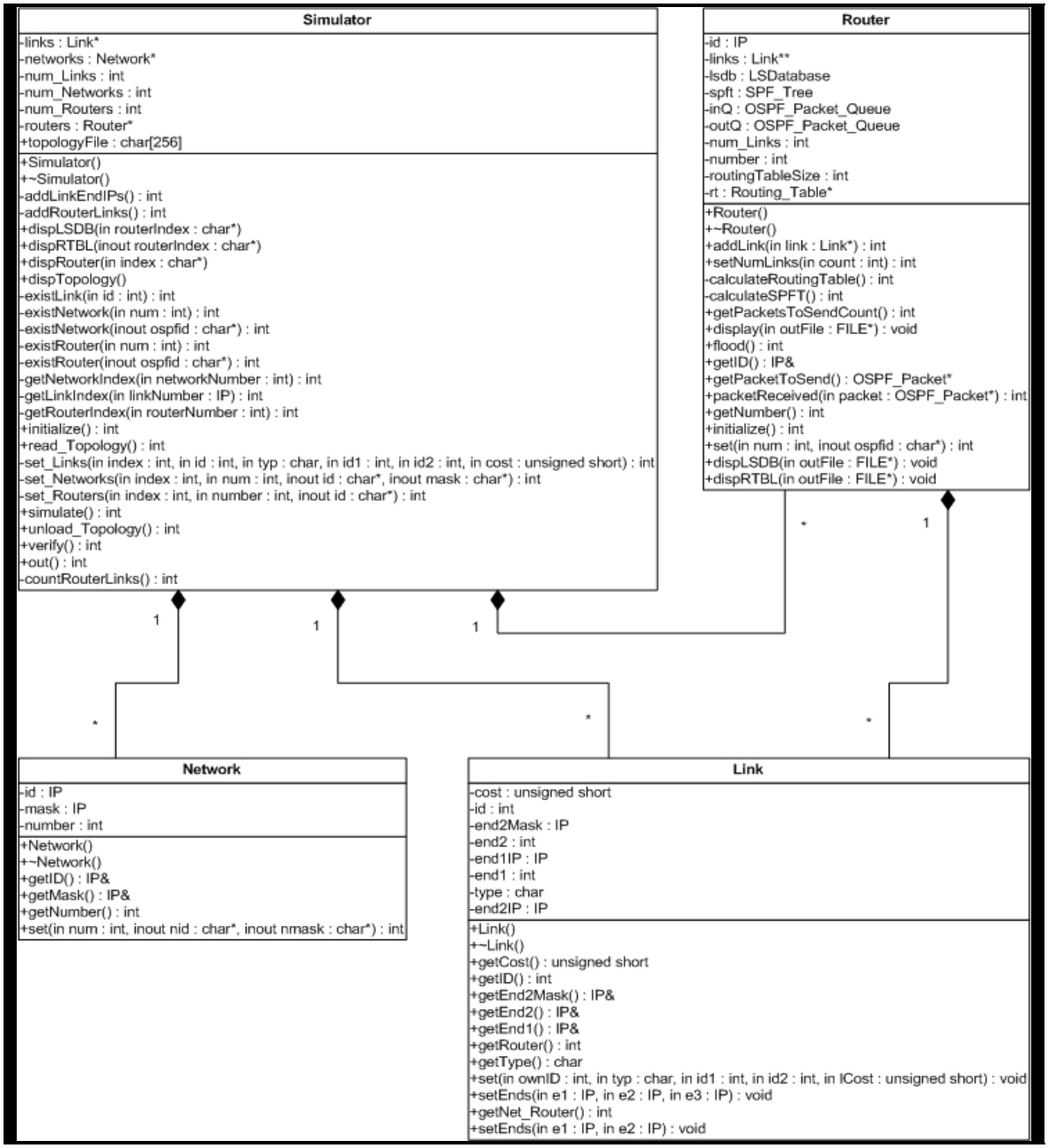

Fig 1: Basic Simulator Structure

Network (Network.{h/cpp})

This class represents a network. Only OSPF "***stub networks***" are implemented in this simulator.

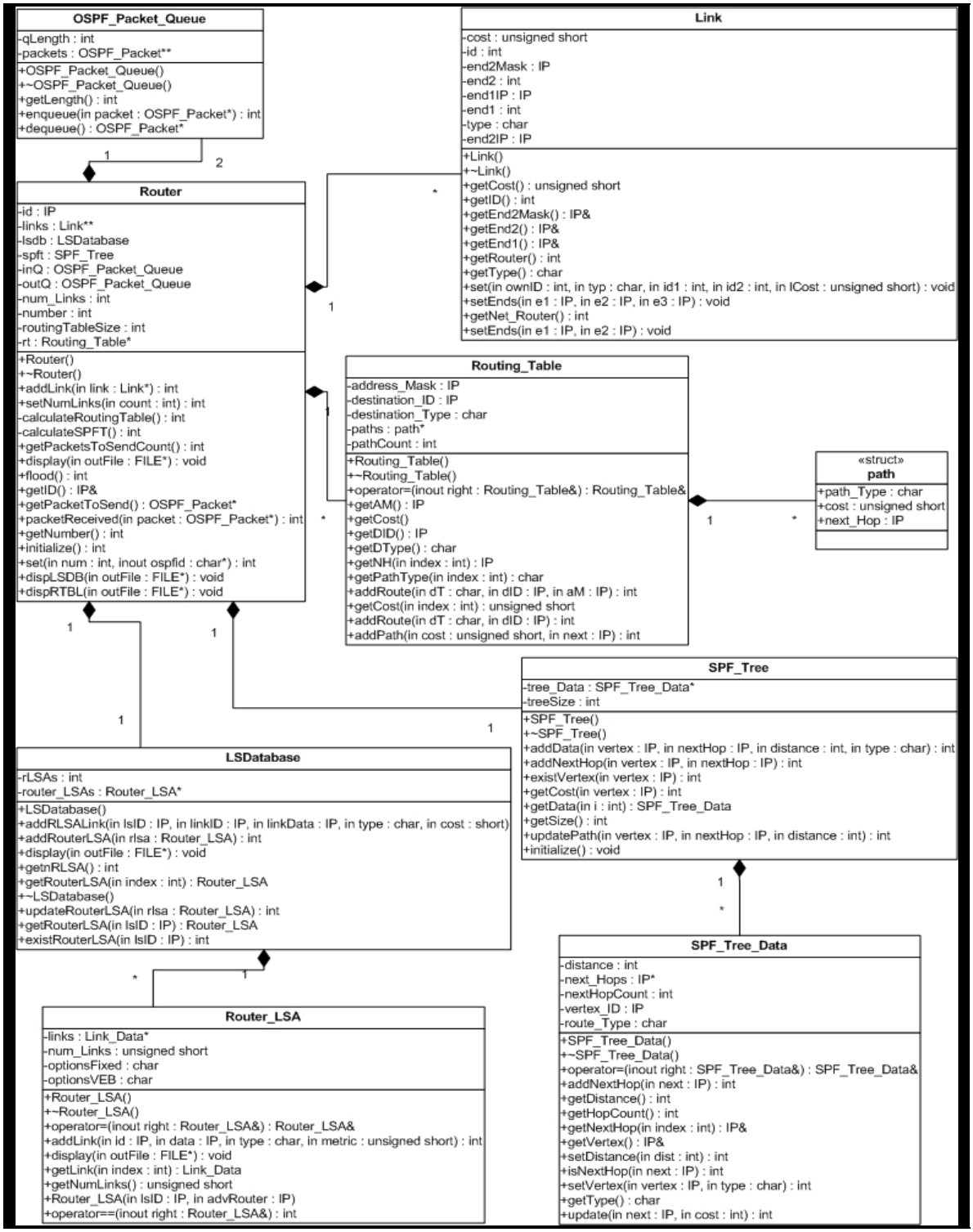

Fig 2: Router and related classes

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. As only Router LSAs are implemented in this simulator, therefore, every instance of ***LSDatabase*** contains Router LSAs only.

SPF_Tree (SPF_Tree.{h/cpp})
This class represents an OSPF shortest path tree. In OSPF, a router has an independent shortest path tree for every area, but as this simulator doesn't support areas therefore there will be only one instance of ***SPF_Tree*** in every router.

SPF_Tree_Data (SPF_Tree_Data.{h/cpp})
This class describes an individual element of a shortest path tree.

Routing_Table (Routing_Table.{h/cpp})
Although this class is named as Routing_Table, but basically it describes an individual entry of OSPF routing table. As OSPF supports multiple paths, therefore every entry contains more than one 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*** basically contains packets sent by other routers to this router, while the ***outQ*** contains packet to be sent by this router. During the execution of OSPF functionality, simulator takes packets from the ***outQ*** 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.

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 like header.

OSPF_Packet (OSPF_Packet_Queue.{h/cpp})
This structure basically 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.

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 like header.

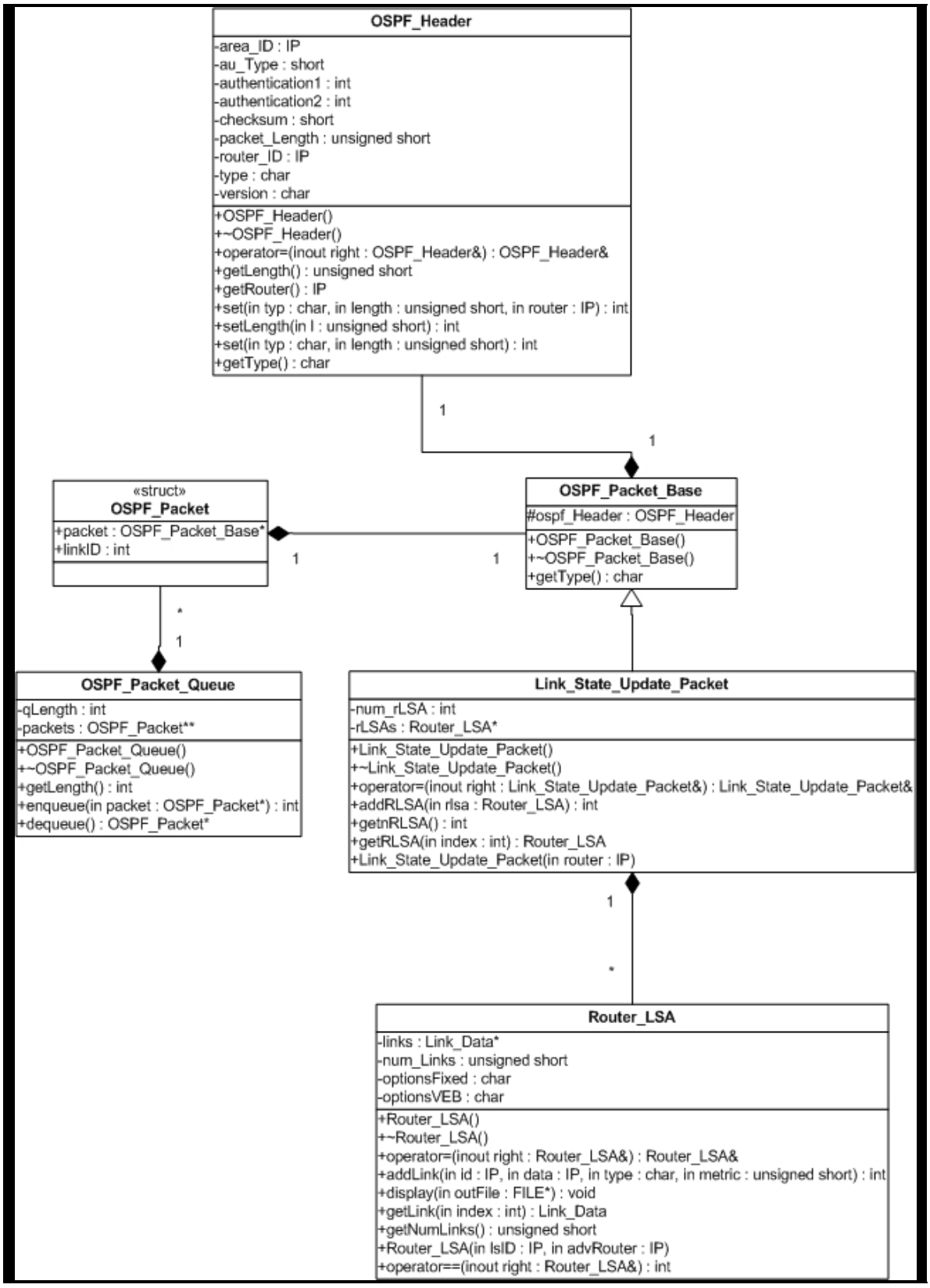

Fig 3: OSPF Packet hierarchy in the simulator

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 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.

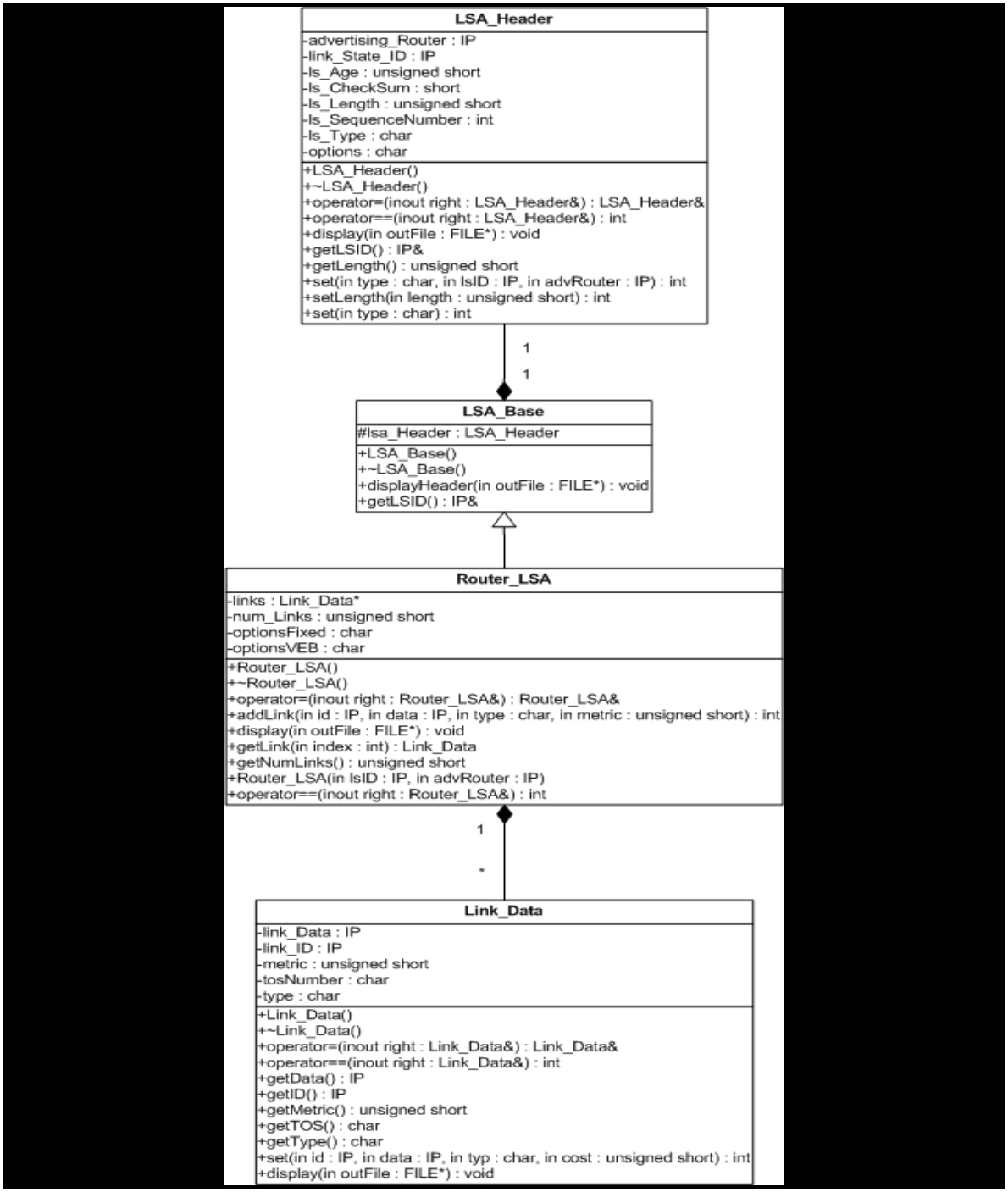

Fig 4: Link State Advertisement hierarchy in the simulator

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

5. 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 simulator is also given below to enhance the clarity.

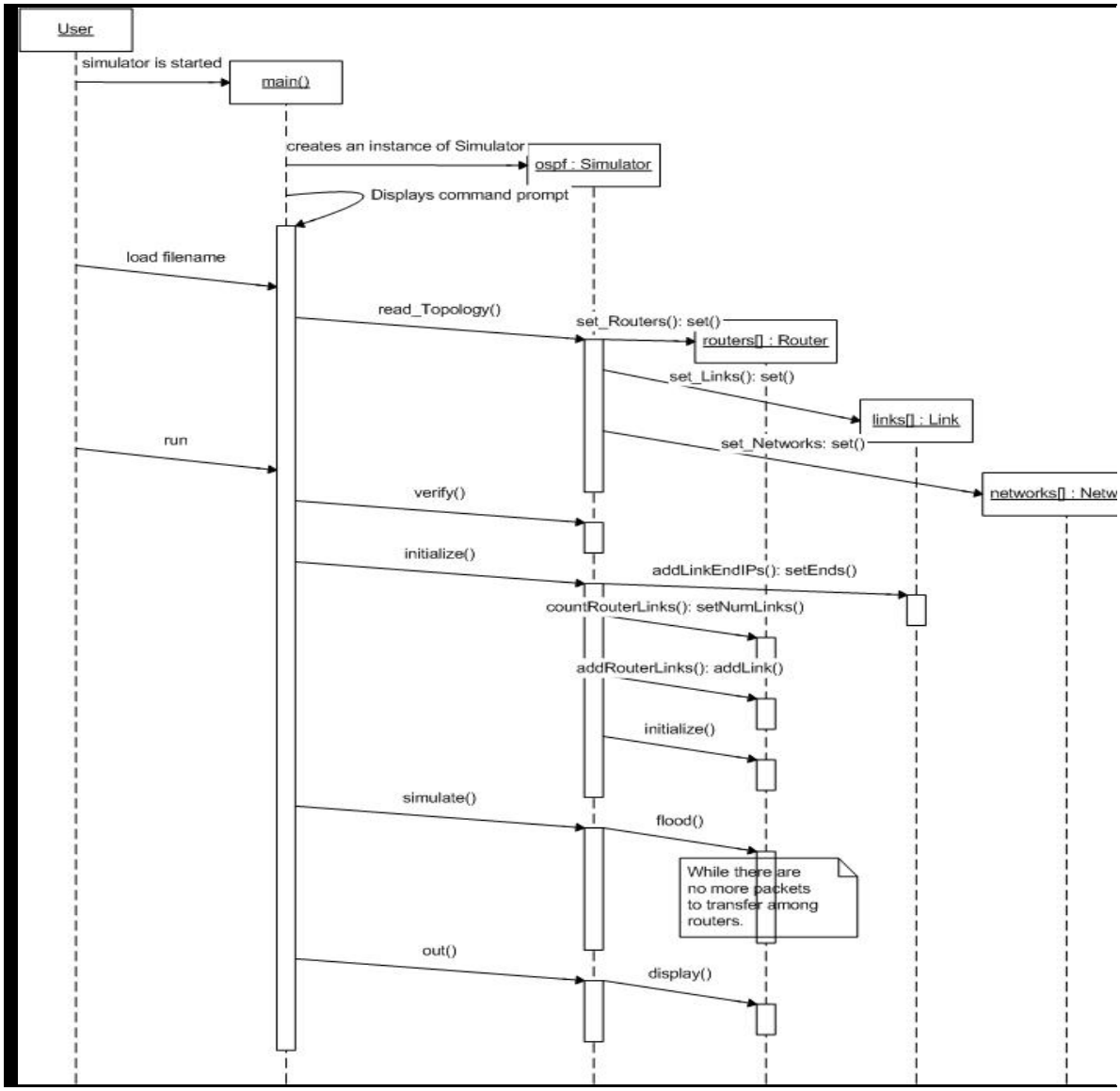

Fig 5: Sequence diagram showing sequence of operations inside "load" and "run" command

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

```
OSPF Simulator Command Line>>
```

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 ***Simulator*** instance and ***read_Topology*** method is invoked. This method reads the file line by line and identifies different parameters from the file. The members, ***routers***, ***links*** and ***networks*** are initiated and 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 ***verify*** method of ***Simulator*** instance is called to verify any ambiguities in the topology like invalid or duplicate addresses etc.

In 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 but just sets the number of the links in each router by calling ***setNumLinks***. These link references are later passed by ***addRouterLinks*** to each ***Router*** instance by calling their ***addLink*** method. Finally, the ***initialize*** of each ***Router*** instance is called.

Third step actually performs the OSPF flooding. 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 relevant ***Router*** instances. This process is performed repeatedly, until all the ***inQs*** are found empty.

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 actually writes Link State Database and routing table to the output file.

6. Your task

The simulator provided for this assignment purpose is fully functional (as per exceptions mentioned earlier). However, we eliminated the code from ***initialize***, ***flood***, ***calcualteSPFT*** and ***calcualteRoutingTable*** methods of ***Router*** class. 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 follow 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, SPF_Tree.h, SPF_Tree_Data.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 implementation of functions in these files will be provided in compiled ***(.o)*** form. However, ***Router.o*** will not have the implementation of ***initialize***, ***flood***, ***calcualteSPFT*** 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 classes except ***Router***, ***SPF_Tree*** and ***SPF_Tree_Data***.
- 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. They have to simply run the command ***./compile-ospf*** from OS/X command prompt. This will generate an executable with name ***ospf.exe***.
- As a first step, they have to submit a brief draft describing their approach to solve this task. It may also contain a description of functions or attributes they are going to add in ***Router.h, SPF_Tree.h, SPF_Tree_Data.h,*** and ***ospf.cpp.***
- In the ***Konzept*** phase, they have to submit a report describing the tentative changes they are planning to make in ***Router.h, SPF_Tree.h, SPF_Tree_Data.h,*** and ***ospf.cpp.*** It should also contain a flow of the OSPF functionality they have to implement inside the simulator.
- Finally, they have to submit their version of ***Router.h, SPF_Tree.h, SPF_Tree_Data.h,*** and ***ospf.cpp.***

7. Reading Material helpful in this assignment

- The main reading source for this assignment is OSPF version 2 RFC 2328. A copy of this RFC is provided. 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 required tasks.
- 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.