SPRING ‘16

CS 2520: A SIMPLE ROUTING PROTOCOL

PROJECT REPORT

TEAMMATE 1: DEBARUN DAS (PITT ID: DED59)
TEAMMATE2: NANNAN WEN (PITT ID: NAW66)

UNIVERSITY OF PITTSBURGH |

1.Introduction

This document serves as the ProjectReport of the term project ―Simple Routing Protocol inCS 2520. The routing protocol required to be implemented is a Simple Link State Routing Protocol (SLSRP). This documents gives a brief overview of the different stages of implementation and the design choices and decision making strategies for efficient interplay between different components of the system. A Link State Routing Protocol consists of an autonomous system of Routers which exchange routing information via a common routing protocol. Every router has a local copy of the entire topology and each router independently calculates its own best path. Further, the overall performance of the topology is tested by sending files from one end system to another that are connected to each other by a series of routers.

Various stages of SLSRP include configurations of each router using the Configuration file. Also, the user is allowed to issue commands and interact with the system. After a router is configured, it establishes adjacency relationship with neighboring routers. Then, the cost of the link with each of its neighbors is calculated. Periodic messages are sent to check the status of its neighbors at regular intervals. This followed by the generation and flooding of Link State Advertisements (LSA), building the Link State Database (LSD) and ultimately useLSD to generate the routing table using Dijkstra's Shortest Path Algorithm.

We used C++ to implement this system.

2.System Overview

This section identifies the components of the proposed system and defines them by their functionality without getting into too much details on how to implement them, it just briefly talks about the structure of the system. It also talks about the layering of the system.

2.1 SYSTEM COMPONENTS

The whole system consists of the following major components.

Name Server (NS):

This application component runs on a set of end systems of the network and this node will be providing services to the routers to register themselves to the system and in services to the client to query the details of other routers in the topology.

Server (S):

This is the piece of application that runs on an end system of the application which are going to be known as Server. They will be responsible for providing services to the Client's request to send and/or receive files via routers.

Client (C):

This component will send requests to the server to send files via the router. This part of application will be exposed to end users to issue several requests using known commands to transfer files, change configuration files etc.

Routers (R):

This component will register themselves to the *NS*, and also send request to the server in the topology initiating phase, they will also send messages to each other when trying to form a topology, and then a routing table will be generated for each router. When the client requests the server to transfer the file, it will need to use the information in the routing table to form a path, so that it can transfer the file successfully.

2.2 SYSTEM LAYERING

The components of the system takeup separate role with different functionalities. They participate in the system layering and integrate as a whole system.

User Interface (UI):

This is the top most layer which provides a platform for the users to interact with the underlying system. We have implemented the *command-line interface (CLI)*for user to issue commands for

the configuration of the routers and file transfer.

Generating Routing Table (GRT):

This step is done by using Dijkstra's Algorithm, in this phase we used a two dimensional weighted adjacency matrix to represent the directed graph of the entire topology. This is the core of this project, it will provide the route when the client transfers a file to the user.

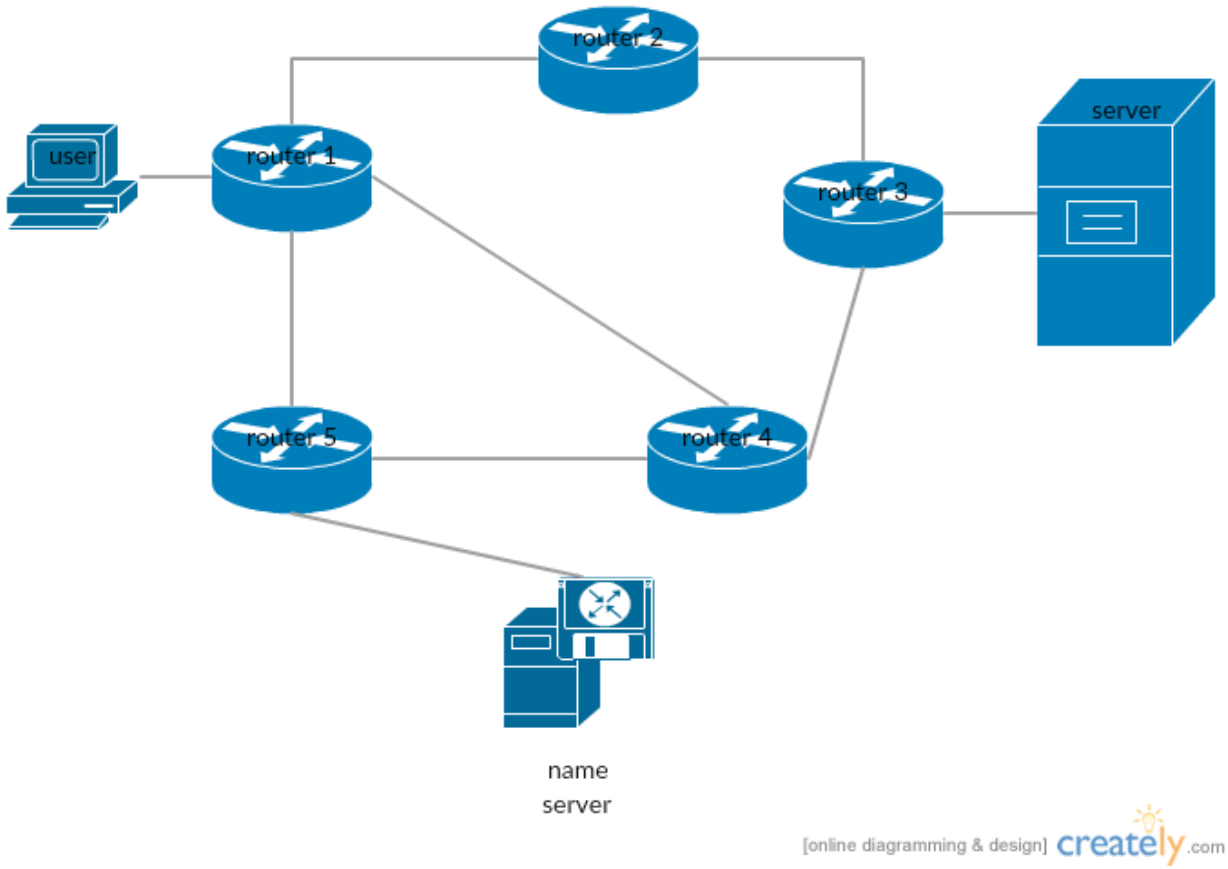

Figure 2.1: example of the overall topology

3 System Architecture and Layering

This section explains functionality and architecture of the components of the system and layered architecture of the LSA in details.

3.1 DETAILS OF SYSTEM COMPONENTS

3.1.1 NAME SERVER

As mentioned in Section 2, this is the top most component of the application that is initialized in the beginning or we can say that the system bootstraps by initializing the NS first.
It is used by the router to register and resolve. So before any router can send messages to other router to form connection, the router will register itself with NS. The Name Server maintains a hash table in its primary memory. The hash is keyed by the combination of the IP of the router and the port number. The value of the hash entry is a data structure as shown in following containing IP address, Port number of the router that is alive.

3.1.2 CONFIGURATION OF THE ROUTER

A configurable file helps to remove the need for hard-coding in the implementation by placing them somewhere accessible by the implementation globally. We implement the configurable item in two fashions: one is *static* and also *dynamic*.
By static we mean by placing them in an external configurable file, so that the router can access the file in initialization phase. By dynamic it means we also use user inputs for changing the router configuration.

3.1.2.1 ROUTER INITIALIZATION

The router gets its own IP address (referred to as Rip) and an available Port number dynamically. One thing to be noted is that the IP address is not the loopback IP address of the machine. For design of the router topology in the real world, each interface of the Router is identified with a unique IP address. However, since this project demands implementation of a simpler version of the Link State Routing protocol, we associate only a single IP address with a single router and indicate the different interfaces with different available port numbers in the machine. After this, router determines it Router Id and registers itself in the Name Server.

3.1.2.2 CONFIGURATION FILE

The router is configured using the external configuration file and the commands provided by the user at the User Interface. The list of commands available to the user will be described later. The following are the list of configurable items to be provided in the *external configurable file*:

- *HelloInterval:* This is the interval of time after which Alive messages will be sent to all the neighboring routers.

- *UpdateInterval:* Interval sfter which LSAs are flooded and generated. Also, the interval after which routing table is updated.

- Pe: Probability of flipping a byte in a packet to introduce error.

- Pc: Probability of dropping packets.

- MaxAge: Maximum Age of an entry in the Link State Database.

- NameServer: Path to the NameConfig file that contains the IP and Port of the Name Server.

- Debug: If set to 1, it will print all the messages in details. Otherwise, it will print only selectively.

3.2 DETAILS OF PROTOCOLCOMPONENTS

3.2.1 NEIGHBOR ACQUISITION PROTOCOL

After the configuration of a router with initial parameters in the external configuration file, the first step is the discovery of neighboring routers. The fact that routers are neighbors is not sufficient to guarantee an exchange of Link State Advertisements (LSA), they must form adjacencies to exchange LSAs. A collection of packets (together referred to as the *Hello packets* sometimes) are sent out of each functioning router interface to discover and maintain relationship with neighbor routers. Before describing the types of messages that are transmitted for neighbor acquisition, we describe the design of a data structure that will store the list of neighbors and some of their related fields, for each router in the Autonomous system.

3.2.1.1 NEIGHBOR_TABLE

We implemented this table using an array of a structure.
A Neighbor_Table is maintained by each router in the Autonomous System. This table will maintain the required details of the neighbors of the router. It is initialized with the list of all the Routers in the Name Server to, which are not present in list of Routers in RBlackListSend. It has the following fields:

- *NeighborRId*: This field stores the router ID of the neighbor. Since the Router ID isidentical to the IP address of the router, this field essentially stores the IP address of the neighboring router.

- *NeighborPort:* This field represents the port number of the Neighbor router whichreceives the neighbor messages. It is generally the port number of the router with

type *Receive_Message*in the nested Hash Table maintained by the Name Server (see Section 2and Section 9).

- *SendNFlag:* This field will store either 0 or 1. The value 1 indicates that *Be_Neighbors_Request*packet has been sent to this router. Otherwise, its value is 0.

- *ACKNFlag:* This field will store either 0 or 1. A value of 1 indicates that the router hasreceived the *Be_Neighbors_Confirm* message as acknowledgement from the router it has sent *Be_Neighbors_Request* to.

- *SendCFlag:* This field holds either 0 or 1. It is set to 1 when the router sends a *Cease_Neighbors_Request* to the neighboring router.

- *ACKCFlag:* This field holds either 0 or 1. It is set to 1 when the router receives a *Cease_Neighbors_Confirm* message from the neighboring router to which it previouslysent *Cease_Neighbors_Request* packet.

- *Cost:* This field contains the cost of the link to the neighbor in the network topology. Avalue of 0 indicates that the Link Cost is yet to calculated

3.2.1.2 REQUESTS NEIGHBER CAN SEND

After the configuration phase of each router, the router can send the following types of neighbor acquisition messages:

- **Be_Neighbors_Request:** This message packet is sent to all the routers whose*SendNFlag*and *SendCFlag* are 0 in the *Neighbor_Table*. After successfully sending the *Be_Neighbors_Request* packet, the *SendNFlag* and the *WaitACKNFlag*corresponding tothe recipient router in the*Neighbor_Table* is set to 1.

- **Cease_Neighbors_Request:** This message is sent to a router to which the sending routerdoes not want to be neighbors with, anymore. However, this message can only be sent to routers whose *ACKNFlag*is set to 1 in the *Neighbor_Table*. The *SendCFlag* of the *Neighbor_Table* is set to 1.

- **Be_Neighbors_Confirm:** This message can be sent as an acknowledgement to themessage *Be_Neighbors_Request*. There are two criteria which are checked before a router sends a *Be_Neighbors_Request*. The version number of the protocol that is run by the two routers should be identical and the router which sends the *Be_Neighbors_Request* should not be in the *RBlackListReceive* of the router which sends *Be_Neighbors_Confirm*. After a *Be_Neighbors_Confirm* message is received by a router as acknowledgement to

  Be_Neighbors_Request, the new neighbor is added to its *Neighbor_Table* with theACKNFlag set to 1.

- **Be_Neighbors_Refuse:** This message can be sent as an acknowledgement to the routerwhich sends the message *Be_Neighbors_Request*. Ideally, there are two instances in which this is done. First, if the router that requests to be neighbor is in the *RBlackListReceive* of the receiving router and second, when the protocol version of thetwo routers do not match. The requesting router, on receiving the *Be_Neighbors_Refuse* message resets the *SendNFlag* and *WaitACKNFlag* of the *Neighbor_Table* to zero.

- **Cease_Neighbors_Confirm:** This message is sent as an acknowledgement of*Cease_Neighbors_Request* that is received from a router. The router that sends the*Cease_Neighbors_Request*, on receiving the *Cease_Neighbors_Confirm* message, resetsthe value of *SendNFlag* and ACKNFlag of *Neighbor_Table* to zero. Also, the values of
  SendCFlag and *ACKCFlag* are set to 1.

NeighborRId	NeighborPort	SendNFlag	WaitACKNFlag	ACKNFlag	SendCFlag	ACKCFlag	Cost
RId_2	2000	1	0	1	0	0	10
RId_3	3800	1	1	0	0	0	0
RId_4	2500	0	0	0	0	0	0

Figure 3.1: Sample Neighbor_Table of A Router with Router ID RId_1

3.3 DETAILS OF DIFFERENT MESSAGES

3.3.1 ALIVE MESSAGES

Alive messages are periodic messages that are sent to all the routers with which neighboringrelationship has been established. This is essentially the procedure for monitoring the network status that contributes to robustness.
Alive messages will be sent out after a fixed amount of time called the Hello Interval. We send multiple *Alive* messages in a *session* before concluding about the state of a neighboring router. Each such *session* will initiate every time a timer for the Update Interval starts and Alive Messages will be sent to the neighbor routers after every *HInterval* for the entire duration of the *UInterval*, until it receives an acknowledgement (message of type *Alive_ACK*) to one of the *Alive* Messages that it sent out to a particular neighboring router. If no acknowledgement message is received from a neighbor router in a particular session, then the neighbor router is assumed to be *dead* and new LSA packets are generated to be flooded out at the end of the *UInterval*. Additionally, the *Neighbor_Table* is modified to reflect that the adjacency with the unreachable adjacent router isbroken. Generally, entry for the unreachable adjacent router is erased from the *Neighbor_Table*. Also, the Hash Table that maintains the record of all the routers in the topology

is also updated.
Sending multiple *Alive* messages before coming to a conclusion about the state of a neighboring router ensures that the network is not highly sensitive and more stable.

3.3.2 LINK COST MESSAGES

Link Cost Messages are sent after the Neighbor Acquisition phase to determine the cost of a link to a neighboring router. The following strategy is used for measuring the cost of a link to a neighboring router. A series of messages of the type *LinkCost* are sent sequentially to every neighboring router from the *LinkCost* thread (see Section 9). Half of the total time for sending a *LinkCost* message and receiving an acknowledgement message *LinkCost_ACK*(or the RoundTrip Time) is ideally the cost of the link to an adjacent router. We send multiple Link Cost Messages (around 20) and find the mean of the Round Trip Times (RTTs)to determine the cost of a link.

A simple strategy to implement this is by using two timestamp variables named *start* and *end* placed before sending the message and after receiving an Acknowledgement respectively.These two variables will store time before sending and after receiving Acknowledgement using *gettimeofday()* function in C. Then we used smooth average algorithm we learned in the class to calculate the round trip time as following:

$SRTT(k+1) = \alpha SRTT(k)+(1-\alpha)RTT.$

$RTT = \sum(end_i – start_i))$ for i = 1 to k (k= no. of *LinkCost* Messages sent with acknowledgement)

So, Link Cost = RTT/2.

For simplification, we apply a transformation function this Link cost obtained to transform it to a positive integer value. We classify the actual delays that we get to several range of costs (from 5 to 75).

The final Link Cost that we get for a particular neighbor is stored in the *Cost* field of the *Neighbor_Table.*

3.4 DETAILS OF LINK STATE ADVERTISEMENT

Each router in the Autonomous System originates one or more Link State Advertisement (*LSA*) packets. A LSA is to describe a router’s local part of the routing topology domain. After a router has established its adjacencies and determined the cost of each of its link, a router can build its LSAs which contains the link-state information of its links.

3.4.1 BUILDING LSA

A link state advertisement packet consists of Router ID, LSA Type and LS Sequence Number etc. Which contains the following:

- **Router ID:** This is a 32 bites field. This field contains the Router ID (*RId*) of the router thatoriginates the LSA.

- **LS Sequence Number:** This is ideally a 32-bit unsigned number that represents thesequence number of a LSA packet. We choose unsigned integer over a signed integer for simplicity in testing the implementation. The sequence number of the first LSA packet that is flooded by the router that generates it, is equal to the value of the parameter *InitialSequenceNo* in the external configuration file. The sequence number is incrementedeach time a new instance of the LSA is generated.

- **LS Age:** This field is a 16-bit unsigned integer which represents the age of an LSApacket in seconds. It must be incremented by the parameter *InfTransDelay* in the external configuration file on every hop during the *flooding* procedure. Ideally, the value of *InfTransDelay* is 1.The LSAs are also aged as they are held in a router's database.

- **Link Id:** This field uniquely identifies the Link to an adjacent router. For convenience inidentifying the link, we choose it uniquely as the combination of the port number of the interface of the originating router that originates the link and the *Rid* of the adjacent router that is connected by that link. This combination is chosen over integers because it helps in uniquely identifying the link and can be directly used in building the Link State Database.

- **Link Cost:** This field contains the cost of the link.

- **LS Checksum:** This field is the checksum of all the contents of the LSA, except the LSAge field, so that the LS Age field can be incremented without updating the checksum. So, the difference between the length of the LSA and the length of the LS Age field is the amount of data to checksum. It can never have the value of zero (which indicates a checksum failure). We implement this by using CRC16.

3.4.2 FLOODING LSA

Flooding is part of the protocol that distributes LSA packets and synchronizes the Link StateDatabase between all the routers in the Autonomous System. The LSA that is generated by a router is sent to all its adjacent neighbor router. In turn, each received LSA is copied and forwarded to every neighbor except to the one that sent the LSA. *Flooding* is a reliable process, that is, we use the protocol Automatic Repeat Request (ARQ). Thus, when a router receives LSAs from a neighboring router, it has to send an acknowledgement message (called *LSAACK*) to the router which sent the LSA.

3.4.2.1 LSA VALIDATION

For each LSA that is received by a router from any of its adjacent routers, the following steps are taken to determine if the LSA will be accepted or not:

a) The LSA's checksum is validated. If it is invalid, hen it is discarded. No Acknowledgement is sent to the sending router so that it resends the LSA packet with the

correct checksum.

b) The LSA's type is validated. If it is invalid, then the LSA is discarded. In our implementation, a valid LSA type is 1. No acknowledgement is sent back is such a case so that retransmission occurs.

If the above two checks are satisfied, then if the router's Link State Database does not contain any LSA of the same originating router, then the received LSA is stored in the Link State Database. However, if a copy of the same originating router's LSA is contained in the local Link State Database of the router, then the following checks are done to determine which is more recent of the two LSAs:

i. The LSA's sequence number is compared with sequence number of the copy of the LSA in the router's local link state database. If the sequence number of the received LSA is less than that of the LSA already contained in the local Link State Database, then it is discarded and an acknowledgement is sent to the sender so that the LSA is not retransmitted again.

ii. If the sequence numbers are equal, then their checksums are compared. The LSA with the highest unsigned checksum is more recent and stored in the Link State Database. Again, an acknowledgement is sent to the sender.

iii. If none of the above conditions are met, the two LSAs are considered identical and the received LSA is discarded. An acknowledgement is sent to the sending router so that the LSA is not retransmitted.

3.4.2.2 Data Structures to be Used for Efficient Flooding

The LSA packets that are accepted are first stored in a vector of LSA packets called *Incoming_List*. There is one other data structures that are maintained for this purpose. They are described as follows.

LINK STATE DATABASE

Every router is maintaining the LSAs as a series of records. The LSD is designed as a Hash Table named LSD with the combination of the RId of the router that originates it and a Link ID as the key of the Hash Table. The values of the Hash Table LSD are all the fields of a LSA packet: LS Sequence Number, LS Age, LS Checksum, Cost (which is contained in the Link Data field of a LSA).

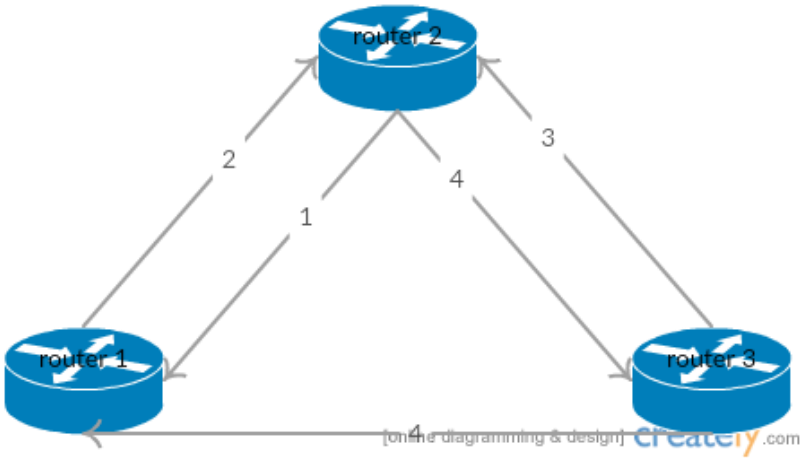

Figure 3.4.2.a: An Autonomous System of Three Routers with the Value on Top of the Edges Representing the Cost of the Link

Router_ Id	Link_ ID	Cost	Sequence	Age	Type	Checksum	No. Of Links
RId_1	RId_2	2	1	3	LS_1	C1	2
RId_1	RId_3	4	1	3	LS_1	C1	2
RId_2	RId_1	1	5	2	LS_1	C2	1
RId_3	RId_2	4	2	1	LS_1	C3	2
RId_3	RId_1	3	2	1	LS_1	C3	2

Figure 3.4.2.b: Link State Database for the Topology in Figure 2a.

The above Figure 3.4.2.b shows a tentative LSD that is formed for the routing topology in Figure 3.4.2.a.In the LSD of Figure 3.4.2.b, the columns Link_ID and Router_ID represent the key of the Hash table that represents the LSD. For simplicity, the Link_Id in Figure 3.4.2.b is represented as the Router Id of the neighbor.

3.4.2.3 FORWARDING LSA

The LSA packets that are stored in the Linked List Incoming_Listare used to update the LSA_Status table and the LSD before flooding to the neighbors other than the sending

router. The LSAs in the Incoming_List are flooded at the end of every UInterval. When a self-generated LSA is to beflooded, it is added to the Incoming_List. As Acknowledgements are received from each of the Neighbor Routers, the LSA_Status table is updated and the respective entry in the Incoming_List is removed. After every Update Interval (UInterval), the LSAs in the Incoming_List are flooded to the neighbors. This takes care of the fact that LSA packets that were not acknowledged are re-transmitted. We wait for the Update Interval to flood messages to avoid a Flood Storm.

3.4.2.4 MANAGING LSA ACK

The LSA Acknowledgement packets (referred to as LSA_ACK) contain the fields Router_Id (RId of the originating router), Type (which is set to LSA_ACK), LS Checksum. When a message of type LSA_ACK is received, the originating Router ID is checked. If it matches with any of the originating router IDs of a LSA in the Incoming_List, then it is removed from the list and updates are made in the LSA_Status Table. If the required LSA is not in the Incoming_List, then we simply discard the LSA_ACK packet.

3.4 DETAILS OF ROUTING TABLE

In this part, we use Dijkstra's Algorithm to generat routing table.

The Routing Table is implemented as a Hash Table with the key as the*Destination_RouterID* and the value as the fields *Next_Hop* and *Cost*. The *Next_Hop* fieldindicates the Router ID to which the next hop is to be made to reach the *Destination_RouterID* using the shortest path. A basic structure of the routing table is shown in Figure4b of the router RId_1 in the routing topology shown in Figure 3.4.1a. In the routing table of Figure 3.4.1b, the key is the column Destination_RId.

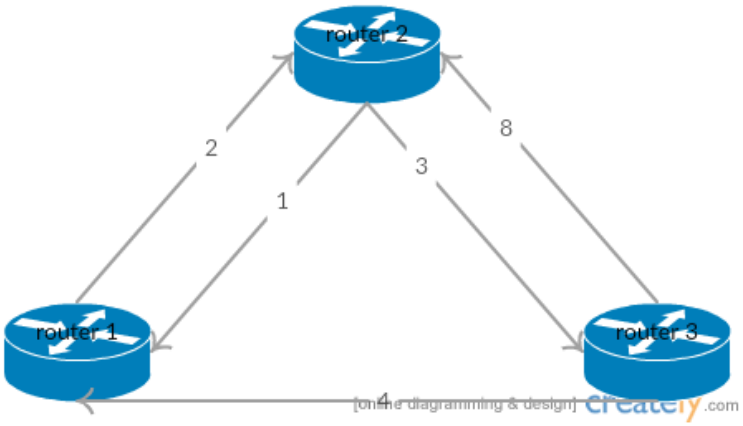

Figure 3.4.1a: An Autonomous System of three routers with respective link costs between them

Destination_RId	Next_Hop	Cost
RId_3	RId_2	6
RId_2	RId_2	2

Figure 3.4.1b: Routing table of Router RId_1 for the topology in Figure 3.4.1a

3.5 DETAILS OF USER INTERFACE

There are two different types of user interface in this system.

3.5.1 USER INTERFACE FOR ROUTERS

The user can use the following commands in this interface:

- *BL*: This command allows the user to enter the list of Blacklisted routers.
- *Age*: This command allows the user to change the MaxAge parameter.
- *Pe:* Allows user to change the *Pe* parameter initialized in by the configuration file.
- *Pc:* Allows user to change the *Pc* parameter initialized in by the configuration file.
- *R:* Allows the user to display the routing table.
- *N:* Allows the user to display the Neighbor Table.
- *Ld:* Allow the user to display the link state database.
- *L:* Allows the user to display the Incoming List of LSAs
- *Cs:*Allows the user to send a "Cease_Neighbors_Request" to an existing neighbor.

3.5.2 USER INTERFACE FOR END SYSTEM

*Server:*The user can choose the edge router to connect to when the server boots up.

Client: The user initially gets to choose the edge router to connect to.

Later, the user gets to select the number of files to be sent and then send the files one by one. To the server at the other end.

3.6 DETAILS OF THREADS IN THE SYSTEM

3.6.1CONFIGURATION THREAD

This is the main thread that configures the router initially. After router configuration, new child threads will be generated for each of the different stages of the protocol. Each thread will use a new socket connection with a new port number. All the data structures – *Neighbor_Table,Incoming_List, LSA_Status table, LSD, Shortest Path Tree* and the *Routing Table* aremaintained in this thread. Thefollowing Threads are generated:

i. Neighbor Acquisition Thread(s).
ii. Alive Thread
iii. Link Cost Thread
iv. LSA Generation and Flooding Thread
v. Routing Table Generation Thread
vi. Receive Thread
vii. Increase Age Thread

3.6.2 NEIGHBOR ACQUISITION THREADS

There is one Neighbor Acquisition thread for each adjacent link to a neighbor. It establishes neighborhood relationships with another router. Forthis, it *resolves* the available neighbor router IP and port in the Name Server and sends *Be_Neighbors_Request* to a router that is not in its*BlackList* and updates the *Neighbor_Table*. Finally, it spawns the Link Cost Thread.

3.6.3 ALIVE THREAD

The Alive Threadwaitstill there is a valid neighbor entry in the *Neighbor_Table*. Once it finds avalid neighbor in the *Neighbor_Table,* it sends *Alive* messages to its neighbors.

3.6.4LINK COST THREAD

It is spawned by the Neighbor_Acquisition thread. It determines the Link Cost to the adjacent neighbor and continues executing till all the valid neighbors in the Neighbor_Table have a valid *Cost* entry.

3.6.5 LSA GENERATION AND FLOODING THREAD

It remains idle initially until there is a valid neighbor entry in the *Neighbor_Table.* However, it will also generate new LSA on other events like when an adjacent router fails or restarts etc. The events in which new LSAs will generated. After a LSA is generated, it is stored in the vector*Incoming_List*. It also updates the LSD with the LSA that was generated. The LSAs are flooded after every Update Interval.

3.6.6ROUTING TABLE GENERATION THREAD

It remains idle until there is an entry in the LSD. It generates the routing table after every Update Interval.

3.6.7 RECEIVE THREAD

This thread is responsible for receiving different types of messages. It registers itself in the Name Server using its port number and functionality *Receive_Message*. The different messages that it can receive include the *Be_Neighbors_Request*, *Cease_Neighbors_Request*, *Alive* message, *LinkCost* message and a *LSA* message.

3.6.8 INCREASE AGE THREAD

Increases the age of each LSD entry every second.

Figure 3.6.1 below shows the relationship between a router and the Name Server and the relation between the user interface, the configuration file with the threads of the router

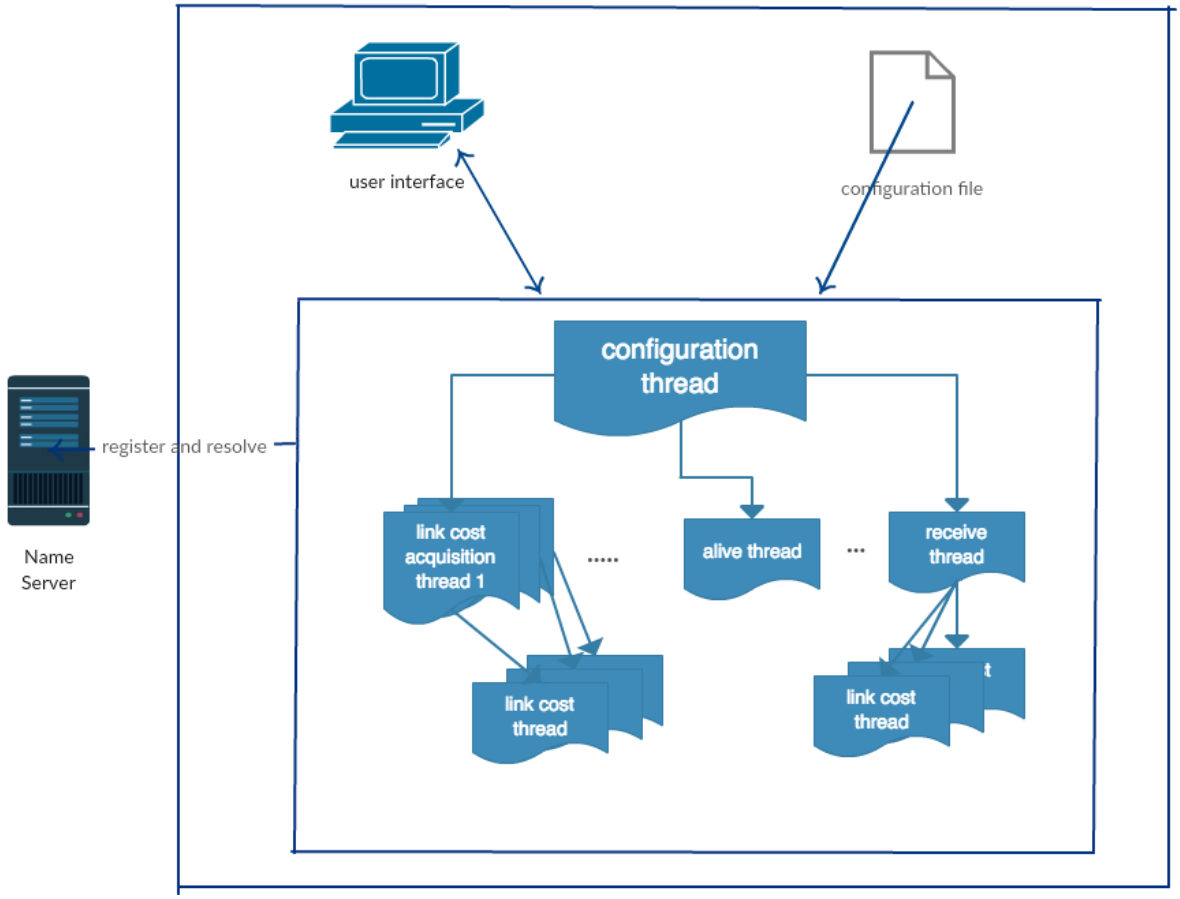

Figure 3.6.1: A Configuration of a Single Router and Registration of the Threads in the NameServer

3.7 DETAILS OF ERROR DETECTION IN THE SYSTEM

3.7.1 PACKET ERROR DETECTION

As mentioned in the project description, errors are introducedin the packet by altering at least one bit with probability P_e (that is generated randomly). So, for detecting this error, we use *CRC16*.

Prior to transmitting a packet to R2, a router R1 computes and appends the *CRC16* to the packet, forming a codeword. When the receiver R2 receives the packet, R2 performs a *CRC* on the whole codeword and compares the resulting check value with an expectedresidue constant, if the check values do not match, then the packet contains a data error, then R2 should request the retransmission of the packet.

3.7.2 DETECT ROUTER DOWN AND UP

This has the potential to create Link StateDatabase inconsistencies. The method is solved using LS Age.

3.7.3 ADD AND REMOVE A ROUTER

When a new router is added, then theLSDs of all the other routers will be updated as soon as this new router's LSAs are flooded. On removing a router, the failed router is detected by using *Alive* messages and the LSDs are updated as soon as new LSAs reflecting this topology change are generated and flooded.

3.8 CONNECTIONS AND PACKET FORMAT

We used TCP in this project because reliability is a major issue in Link State Routing Protocol. Moreover, most of the connections like the connections for Neighbor Acquisition, measuring Link Cost, Flooding LSAs etc are short lived, so resources are kept dedicated to ensure reliability. The packet format of different messages is shown below.

Source	Destination	RouterId	PacketType	Data	Checksum

Figure 3.8.1: A general packet format for Packet Types Be_Neighbors Request, Cease_Neighbors Request, Alive Message and Link Cost Message

In Figure 3.8.1, the Data field will generally contain the port number of the Router Interface and the current IP address of the sending Router.

Source	Destination	RouterId	Type	Checksum

Figure 3.8.2: A general packet format of the Acknowledgement Messages

In Figure 3.8.2, the Type field will hold *Be_Neighbors_Confirm, Be_Neighbors_Refuse,Cease_Neighbors_Confirm, LSA_ACK, LinkCost_ACK, Alive_ACK*. There is no Data field andthe Router ID field is the Router Id of the sending router.

Source
Destination
RouterId
Sequence
Age
No. of links
LinkId1
LinkData1
LinkId2
…
…
…
LSA Checksum

Figure 3.8.3: LSA packet format.

4. Statistical Analysis

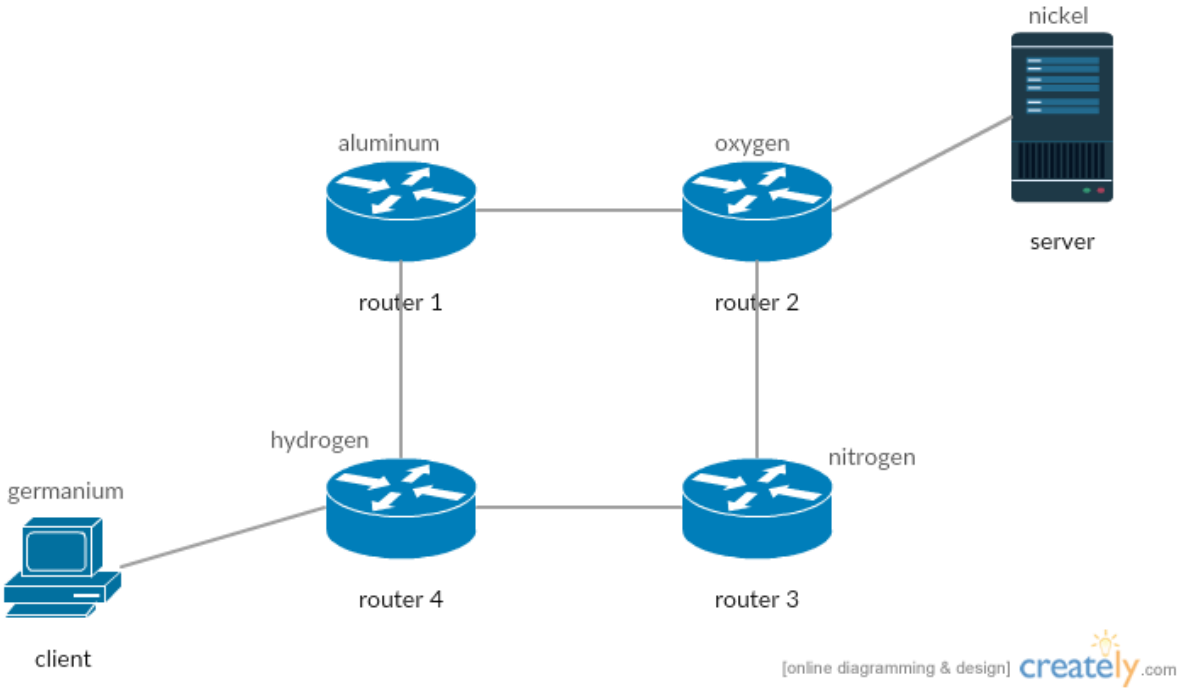

Fig 4.1: Topology Used To Get The Statistical Analysis

The topology in Figure 4 is used to generate statistical analysis. We transfer files from the client (germanium.cs.pitt.edu) to the server (nickel.cs.pitt.edu) and see variations in Mean File Transfer Time with file size and packet size.

1. **Mean File Transfer Time vs File Size:**

File Size(in KB)	Mean Transfer Time(in sec)	Std Deviation
0.026 KB	0.02948958	0.006845376
2 KB	0.04507346	0.006069376
10 KB	0.0848582	0.008927065
60 KB	0.1887064	0.021550979
150 KB	0.495709	0.051501606

Figure 4.2: MTT vs File Size Table

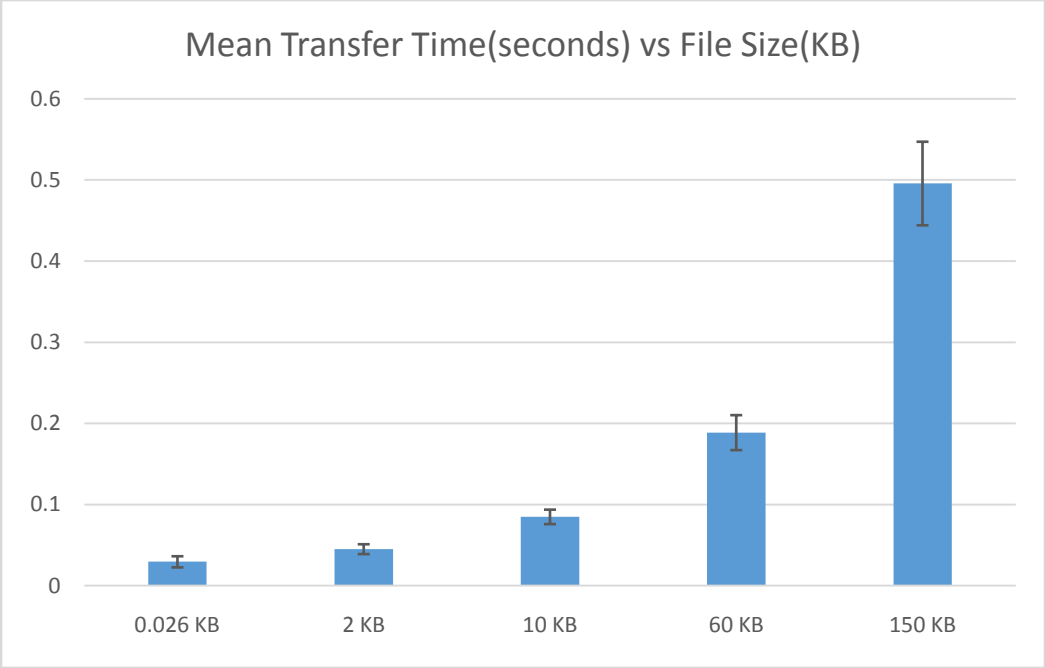

Figure 4.3: MTT vs File Size Chart with Error Bars

The above plot shows that the transfer time of the file increases significantly with the increase in file size. Also, the deviations increase as the File Size increase. This is normal as the MTTs are higher for large files.

2. **Mean File Transfer Time vs Packet Size**

Packet Size(Bytes)	Mean Transfer Time(sec)	Std Deviation
25B	0.257861	0.04529567
64B	0.236539	0.04259347
128B	0.206101	0.03628776
512B	0.1951927	0.024129847
1024B	0.1857816	0.020197679

Figure 4.4: MTT vs Packet Size Table

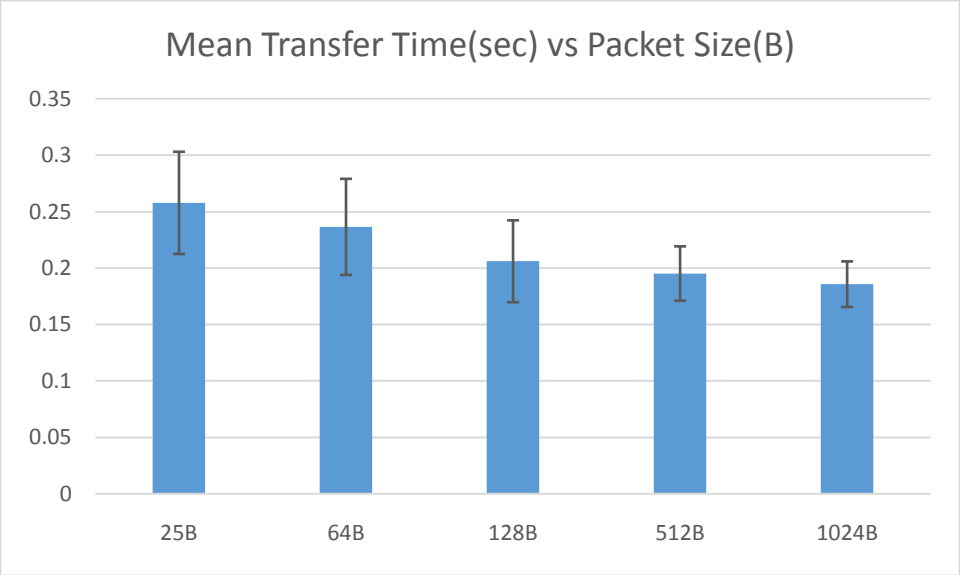

Figure 4.5: MTT vs File Size Chart with Error Bars

The above plot shows that the file transfer time increases with decrease in the packet size that is transferred at a time. Again, this shows expected behavior.

5. Conclusion

Our main goal is to implement a simple routing protocol and analyze the performance of the implemented protocol. We were successful in testing the performance of CRC, file transfer and Dijkstra's Algorithm along with other functionalities. In future we would like to make more changes to the project, so that it can be more convenient for the user and can be adapt to any topology.

6. Bibliography:

1. https://supportforums.cisco.com/discussion/11060926/ospf-lsa-aged-maxage

2. https://www.ietf.org/rfc/rfc2328.txt

3. http://people.cs.pitt.edu/~znati/wans.htm

3. http://www.ciscopress.com/articles/article.asp?p=24090&seqNum=4

4. https://en.wikipedia.org/wiki/Checksum

5. https://en.wikipedia.org/wiki/Convergence_(routing)

6. https://en.wikipedia.org/wiki/Cyclic_redundancy_check

7. Data Communications And Networking (4th Edition) — Behrouz A. Forouzan

8. http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/

9. http://people.cs.pitt.edu/~znati/wans.html