PROJECT DOCUMENT

LINK STATE ROUTING PROTOCOL USING SERVLETS & JSPs

PROJECT

DESIGNED BY:

NIKHIL BIRUR
A20357121
SAMI AHMAD KHAN
A20352677

COURSE: CS 542 COMPUTER NETWORKS 1 - FUNDAMENTALS

INTRODUCTION

There are two main classes of routing protocols:

1. Link State Routing Protocol, and
2. Distance Vector Routing Protocol

Link-State Routing protocols are routing protocols whose algorithms calculate the best paths to networks differently than Distance Vector routing protocols. Whereas **Distance Vector protocols** know routes by measures of distance and vector(direction) as reported by neighboring routers, Link-State routing protocols calculate their network routes by building a complete topology of the entire network area and then calculating the best path from this topology or map of all the interconnected networks.
Examples of link-state routing protocols include *open shortest path first (OSPF)* and *intermediate system to intermediate system (IS-IS).*
The link-state protocol is performed by every *switching node* in the network (i.e., nodes that are prepared to forward packets; in the Internet, these are called routers). The basic concept of link-state routing is that every node constructs a *map* of the connectivity to the network, in the form of a graph, showing which nodes are connected to which other nodes. Each node then independently calculates the next best logical *path* from it to every possible destination in the network. The collection of best paths will then form the node's routing table.
This contrasts with distance-vector routing protocols, which work by having each node share its **routing table** with its neighbors. In a link-state protocol the only information passed between nodes is **connectivity related**.
Link-state algorithms are sometimes characterized informally as each router 'telling the world about its neighbors".

Link-State Characteristics

- SPF algorithm - Link-State routing protocols are designed around Dijkstra's Shortest Path First Algorithm (SPF) in which the shortest path from point A to point B is build around a metric of cost.
- Cost metric - SPF algorithm finds the shortest path based on a metric network link costs. Each router measures the cost of its own directly

connected networks or "links." Cost is a measure of the quality of a link based mostly on bandwidth.

- Hello packets - Link-State routing protocols establish adjacencies with neighboring routers using hello packets.
- Link State Packets (LSP) - Initial flooding of link-states to all routers in the network.
- Topology or SPF Tree - Link-State routing protocols build and maintain a complete map or topology of the network area.

Link-State Advantages

- *Faster Convergence* - Unlike Distance Vector routing protocols which run algorithm calculations before sending updates, Link-State routing protocols send link-state updates to all routers in the network before running route calculations.
- *Triggered Updates* - Unlike Distance Vector routing protocols (except EIGRP) which send periodic updates at regular intervals, Link-State routing protocols send LSPs during router startup (flooding) and when a link changes states like going up or down. If there are no changes in the network the protocol only sends hello packets to maintain adjacencies.
- *Scalability* - Link-State routing protocols support the ability to configure multiple routing "areas" which allows an administrator to segment a routing protocol processes to defined areas which supports the expansion and troubleshooting of much larger networks.

Link-State Disadvantages

- *Greater Processing Requirements* - Link-State routing protocols typically demand greater processing power and memory resources from the router.
- *Greater Administrator Knowledge* - Link-State routing protocols can demand advanced administrator knowledge to configure and troubleshoot the network area.

What is Dijkstra's Shortest Path First Algorithm (SPF)?

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956.

This algorithm has been used in the GPS navigation systems. It:-

- Assumes non-negative edge weights.
- Even if we're only interested in the path from *s* to a single destination, *d*, we need to find the shortest path from *s* to *all* vertices in *G* (otherwise, we might have missed a shorter path).
- If the shortest path from *s* to *d* passes through an intermediate node *u*, i.e., $P = \{s, \ldots, u, \ldots, d\}$, then $P' = \{s, \ldots, u\}$ must
- be the shortest path from *s* to *u*.

Algorithm complexity: *N* nodes.
- each iteration: extract minHeap $O(\log|N|)$.
- Total $O(|N|\log|N|)$.

Each neighbor of each node could also potentially go thru the minHeap once: $\boldsymbol{O(|E|\log|N|)}$
Total: $O(|N|\log(|N|)+|E|\log(|N|)) = O(|E|\log|N|)$ • ($|E| \geq |N| - 1$ for a connected graph)

Oscillations possible:
e.g., link cost = amount of carried traffic, asymmetric link cost.

PROJECT SCHEDULING

SNo.	TASK NAME	TASK LEAD	DURATION (approx.)	START DATE	FINISH DATE	% COMPLETE
1.	Project Initiation					
a.	Group Meeting	NA	-	11/07/15	11/07/15	0%
b.	Planning Project Framework	Nikhil	3 hrs	11/07/15	11/07/15	1%
c.	Creating Project UMLs	Sami	3 hrs	11/08/15	11/09/15	2%
d.	Dividing & allocating work	Nikhil	1 hr	11/09/15	11/09/15	2%
e.	Initiation Completed	-	Total: 7 hours	11/07/15	11/09/15	5 %
2.	Requirement Phase					
a.	Discussing application requirements	NA	2 hrs	11/10/15	11/10/15	1%
b.	Define Scope	Sami	1 hr	11/10/15	11/10/15	2%
c.	Discussing document and presentation requirements	Sami	3 hrs	11/11/15	11/11/15	2%
c.	Identify Hardware	NA	0.5 hr	11/11/15	11/11/15	0%
d.	Define GUI requirements	Nikhil	1 hr	11/12/15	11/12/15	1%
c.	Application Testing requirements	Nikhil	2 hrs	11/12/15	11/12/15	2%
d.	Requirements Phase completed	NA	Total: 9.5 hrs	11/10/15	11/12/15	8%

3.	**Development Phase - 1**					
a.	Prepare flowchart	Nikhil	1 hr	11/13/15	11/13/15	2 %
b.	Pseudo code implementation	Sami	1 hr	11/13/15	11/13/15	4%
c.	Examining Dijkstra's algorithm	Nikhil	1 hr	11/14/15	11/14/15	4%
d.	Examining network topology diagrams	Nikhil	0.5 hr	11/14/15	11/14/15	3%
e.	Finalizing the flow of the code	Sami	1.5 hr	11/14/15	11/14/15	5%
f.	Phase-1 completed	-	Total: 5 hours	11/13/15	11/14/15	18%
4.	**Development Phase – 2 (Servlets)**					
a.	Start coding with Java	Nikhil	0.5 hr	11/14/15	11/14/15	0%
b.	Create Network Topology	Nikhil	4 hr	11/14/15	11/14/15	3%
c.	Built connection table	Nikhil	4 hr	11/15/15	11/15/15	5%
d.	Shortest path – Djikstra implementation in Java	Nikhil	8 hr	11/16/15	11/17/15	8%
e.	Topology modification implementation	Sami	3.5 hr	11/17/15	11/18/15	8%
f.	Phase-2 completed	-	Total: 20 hours	11/14/15	11/18/15	24%

5.	Development Phase-3 (JSP coding for GUI)					
a.	Start coding with JSP	Nikhil	1 hr	11/16/15	11/16/15	2%
b.	GUI creation	Nikhil	12 hr	11/16/15	11/18/15	8%
c.	Finalizing GUI	Nikhil	4 hr	11/19/15	11/19/15	5%
d.	Phase-3 completed	-	Total: 17 hours	11/16/15	11/19/15	15%
6.	Design Phase					
a.	Data Collection	Sami	1.5 hrs	11/15/15	11/15/15	2%
b.	Data Analysis	Sami	2 hrs	11/15/15	11/15/15	2%
c.	Rough Design sketching	Sami & Nikhil	4.5 hrs	11/16/15	11/16/15	3%
d.	Designed Scheduling	Sami	6 hrs	11/17/15	11/17/15	5%
e.	Designed Resource allocation	Sami	3 hrs	11/17/15	11/18/15	5%
f.	Designed High & Low design	Sami & Nikhil	4.5 hrs	11/18/15	11/19/15	5%
g.	Designing power point slides	Sami & Nikhil	6 hrs	11/20/15	11/21/15	8%
h.	Design Phase completed		Total: 27.5 hours	11/15/15	11/21/15	30%

Total Time to complete the project: 86 hours
Percentage completion: 100%

RESOURCE ALLOCATION

Resources allocation for feature development for: -

A). NIKHIL BIRUR:

Task Allocated	Task Complexity	Time Taken	Responsibility
Project Framework planning	Low	3 hours	• Sketched and laid framework for the project. • Designed flowchart.
Allocation of tasks	Low	1 hour	• Took the responsibility for dividing and allocating tasks.
Writing codes for GUI Implementation using JSP.	Medium	17 hours	• Designed the GUI part of the project.
Writing codes for building connection table & shortest path (Djikstra)	High	12 hours	• Developed code for creating connection table. • Also implemented Djikstra's algorithm through Java.
Removing and displaying node	Medium	3.5 hours	• Implemented removing of a node from the existing topology.

B). SAMI AHMAD KHAN:

Task Allocated	Task Complexity	Time Taken	Responsibility
Creating UMLs for the program	Medium	3 hours	• Designed UML diagrams that shows the flow of the program as well as the classes, attributes and methods that will be required for the program
Discussing and defining the scope of the program and documentation	Low	4 hours	• Designed the scope of the program. • Designed the requirements for the program as well as its presentation.
Implemented topology modification in Java	Medium	3.5 hours	• Designed adding of a node. • Designed deleting a particular node.
Project Document design	Medium-High	20 hours	• Collected and analyzed data. • Designed project scheduling, resource allocation and High/Low design. • Created presentation slides.
User Manual	Low	1 hour	• Created manual for the user as to how to install and run the application in their system.

HIGH DESIGN

This High design section explains the design and functionality of various components of our project at the system level.

The project is built using Java Servlets and JSP (Java Servlet pages).

Following are the various *components* that we used for building our system: -

- Java Servlets:
  A **servlet** is a small Java program that runs within a Web server. **Servlets** receive and respond to requests from Web clients, usually across HTTP.
  A **Java servlet** is a Java program that extends the capabilities of a server. Although servlets can respond to any types of requests, they most commonly implement applications hosted on Web servers.

  The core coding of the system's logical implementation in our program is done via coding using Java Servlets, where we have created classes for:
    - Uploading Matrix.
    - Creating Connection Tables.
    - Finding shortest path using Djikstra's Algorithm.
    - Viewing Network topology.
    - And Deleting a node from the topology.

  These were the major components that were created using Java Servlets.

- <u>Java Server Pages:</u>

  JavaServer Pages (JSP) is a technology that helps software developers create dynamically generated web pages based on HTML, XML, or other document types. JSP is similar to PHP and ASP, but it uses the Java programming language.

  The Graphical User Interface that we built was designed using **Java Server Pages (JSP)**. As soon as user runs the program, he/she will be greeted with an interface displaying the menu of operations that he/she can select to be performed.

LOW DESIGN

The Low level design explains the functionality at the code level (functionality of our classes, their methods etc.).

Following are the various components that describes the functionality of the code and how it is being implemented: -

The servlet coding has the following classes and their working: -

- ***OptionSelectServlet*** **Class –** This class inherits from ***HttpServlet*** class. The ***HttpServlet*** class Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site.
  The ***OptionSelectServlet*** Class does the following functions: -
  - ➢ It takes a option entered by the user.
  - ➢ And directs the user to the corresponding JSP page in the application.
- ***Connection Table*** Class – This class displays the interfaces between the source and destination routers.
  Methods Used in this class:
  - ➢ ***connection_Table()*** – Return Type – ***Object*** : This method does the following:
    - It reads the matrix from the file.
    - See what all routers are there in the network.
    - And how the routers are connected to each other.
    - And then displays the connection table.
- ***MatrixUploadServlet*** Class – This class reads the matrix from a file that user will give and uploads it to the interface. It is also inherited from the same ***HttpServlet*** class.

Methods Used in this class:

- ➢ ***init()*** – Return type – **void:** This method read value from the config file. The config file defines the folder where the matrix that a user enters will be stored.
- ➢ **doGet()** - Return type – **void:** This is a default HttpServlet class's method. This is used when small amount of data and insensitive data like a query has to be sent as a request.
- ➢ **doPost()** - Return type – **void:** This is a default HttpServlet class's method.
- ➢ **checkMatrixFormat()**- Return type – **void:** Checks whether the input is in the expected format.

- ***<u>DeleteNode</u>*** Class – Deletes the node from the network topology.
  Method Used in this class:
  - ➢ **delete_node*()*** – Return type – **void**: Copies all the original matrix elements to this matrix ,except for the deleted node elements.

- <u>DeleteNodeServlet</u> Class – This class acts as an interface between the **.java** class file and the corresponding **.jsp** file.

- <u>Dijkstras</u> Class – Displays the shortest route between the source and destination routers.
  Methods Used in this class:
  - ➢ **dijkstras()** - Return Type – ***Object*** : This method calculate and finds out the shortest path between the source and destination routers and displays on the interface.

- <u>DijkstrasServlet</u> Class – This class acts as an interface between the **.java** class file and the corresponding **.jsp** file.

- ViewMatrix Class – It displays the matrix on the web page, which the user can view.

  Methods Used in this class:

  - **matrix()** - Return Type – ***void*** : It gets the total number of rows and column and return the whole matrix to be displayed.

And corresponding to all the above Java Servlet classes, we have designed JSP pages that will implement them in a graphically designed webpage.