Kadie Clancy and Brian Falkenstein

Project Design Report

User Interface

We propose to implement a command line interface to be utilized at the client level. A command line interface will allow us to focus on the functionality of the sLSRP and the file transfer system. If additional time remains, we will implement a GUI for increased usability. We plan to implement the following commands:

Configuration Commands		
Command Functionality	Command Syntax	Command Arguments
Update configuration info by reading a new policy management file	new_policyfile	filepath
Display current configuration parameters	print_config	

Routing Table Commands		
Command Functionality	Command Syntax	Command Arguments
View transmitted routing table	print_transmittedRT	
View received routing table	print_recievedRT	

Shortest Path Commands		
Command Functionality	Command Syntax	Command Arguments
Display the shortest path to each router	print_paths	

File Transfer Commands		
Command Functionality	Command Syntax	Command Arguments

Send a file	send_file	Filepath, IP address
View a file	view_file	

General Commands		
Command Functionality	Command Syntax	Command Arguments
Display help information (that will lead to sub-menus to provide help for each command in the form of a description of the command and the syntax)	help	
Exit the user interface system	exit	

Modular Decomposition of Each Component of the System

Overview

We propose to design our sLSRP using a three layer model consisting of the application layer, the packet layer, and the socket layer. Figure 1 illustrates the intended functionality of each layer. We chose to modularize into three layers for a few reasons. Namely, this modularization allows us to reduce design complexity by grouping similar tasks into the same layer. For example, on the application layer we are only concerned with messages and files for the end user and thus this layer does not need to have any knowledge of packets or the paths that that the packets traveled. The socket layer, on the other hand, is concerned only with packets and only with their destination. They need to know nothing more about the packet, including the type of packet or the data it contains, than its path. This means that middle packet layer does much of the intermediate work needed to take commands from the user and transmit data along the network. By grouping routing, packet segmentation, and network topography into the same layer we can simplify implementation. This also allows us to test and debug components of the design separately.

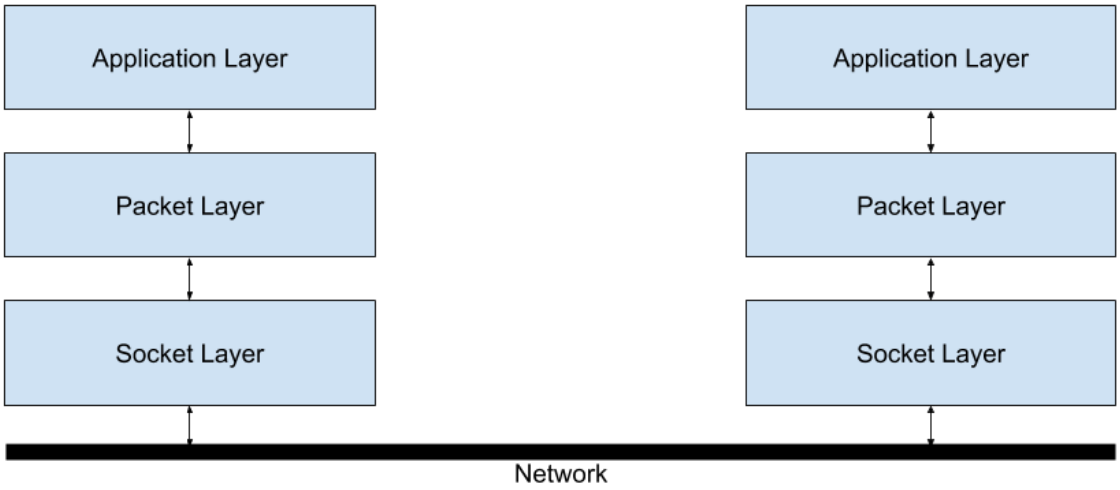

Figure 1
https://docs.google.com/drawings/d/16HjrWcD0zX5LnAjmY27n-zvSfZQRlKmVFTTWP5FCqrc/edit?usp=sharing

Application Layer

The application layer is the the user interface at both the end clients and intermediate routers. The application layer will pass messages (such as the initial configuration parameters and routing table info) and files (as part of the file transfer system) to the packet layer. The application layer will also receive files and messages from the packet layer.

Packet Layer

The packet layer is responsible for several functions. This layer is responsible for maintaining the Routing Table (calculated using Dijkstra's Shortest Path First), the Cost Table, the Link State Database, and the Packet Buffer.

We propose to implement the Cost Table and the Link State Database, as hash tables. The hash tables are indexed by the IP addresses of the routers. The Cost Table will simply store the IP address and associated cost for each router in the network. The Link State Database will contain information about topology for each router in the network, specifically as link statuses and costs. The information stored in the Link State Database at the time of evoking Dijkstra's algorithm will be what is used to compute the Routing Table.

The Routing Table will also be maintained as a hash table, indexed by the destination router. The table will simply store the IP of the next hop to get to the destination router in the shortest time (as computed by Dijkstra's). Although the shortest path to every router is computed with Dijkstra's, we are only concerned with the destination of the next hop. If reliable flooding is performed correctly, each router should have the same Link State Database, and thus the same optimal paths, so each router need only concern itself with forwarding each packet to the next hop.

The Packet Buffer will be stored in a queue implemented with a linked list. This ensures new packets arriving are added to the end of the queue, and processed after other packets that arrived first. The socket layer should handle and pass packets along to the packet layer in the order that they arrive. The process of reassembling packets into messages and files is left to the packet layer.

The packet layer is responsible for receiving messages or files from the application layer. The packet layer will receive the initial configuration message from the application layer and update parameters accordingly. For files to be sent, this layer will segment the file into packets and attach necessary header information based on the Routing Table. This layer is also responsible for generating (and responding when appropriate) neighbor acquisition messages, alive messages, and link cost messages.

This layer is subsequently responsible for assembling received packets back into the original message or file (and thus maintains the packet buffer). For files, this layer passes the reassembled file back to the application layer.

Socket Layer

The socket layer is responsible only for routing packets along the network to the intended destination. This layer receives packets from the packet layer to send along the network. This layer also listens to the network, receives packets intended for it and then passes them to the packet layer.

Interfaces Between Layers

Application and Packet Layer

The application and packet layers can communicate with one another with five types of messages, the structure of these messages is illustrated in Figure 2. The first is Config_Params which allows the application layer to send the initial configuration parameters (protocol_version, update_interval, alive_interval, neighbor_acquisition, link_cost) provided by the policy management file at the user interface. When the user requests to print the current parameters from the user interface, the application layer can send the packet layer a Config_Params_Request message with the configuration parameters. The packet layer can reply with Config_Params message sending the config data back to the application layer. The application layer can send a Paths_Request message when prompted by the user interface to print all shortest paths. The packet layer can reply with a Paths message with the shortest path information.

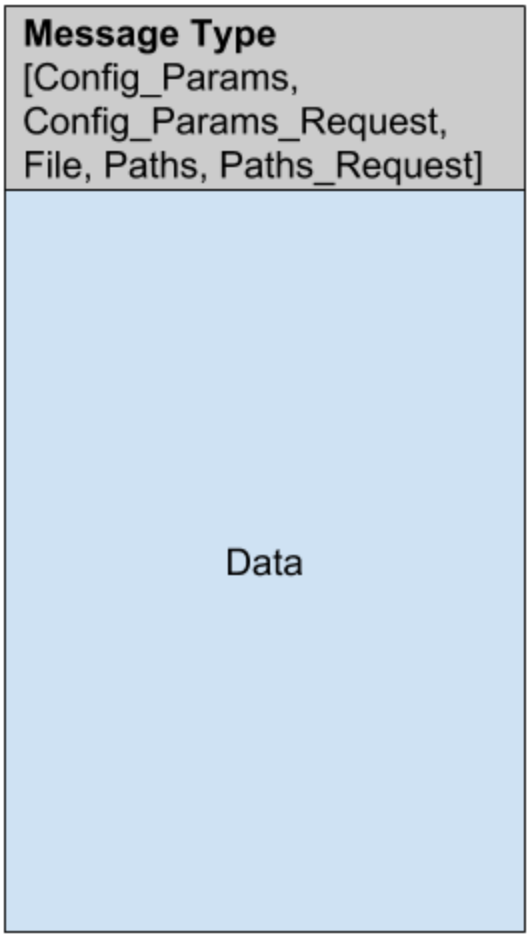

Figure 2
https://docs.google.com/drawings/d/1B-wsLTP1KAPGw2GwFbFP1eCRiiFHoRmT3OszQLu7vpc/edit?usp=sharing

Packet and Socket Layer

The packet and socket layers communicate to one another via packets of a predetermined maximum size. The setup of a general packet is illustrated in Figure 3. The socket layer is only concerned with the Router Name, as this is how it determines where the packages need sent.

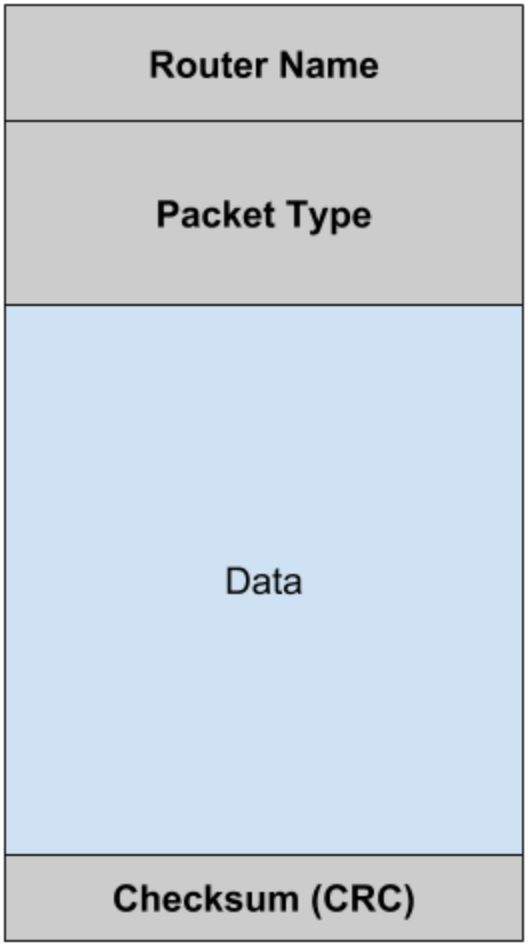

Figure 3
https://docs.google.com/drawings/d/17GsbXw4h0dEZawjLqKIa9oDbEM2wyCk53IrHIfa6hO0/edit?usp=sharing

The packet layer needs to send many types of packets in order to complete the various tasks it is responsible for. These message types include the following:

Neighbor Acquisition Messages	
Packet Type	Description
Be_Neighbor_Request	Sent by a router to invite a neighboring router to become neighbors. The data in this packet type will include Hello_Interval,

	Update_Interval, Protocol_Number, Alive_Max, and Router_ID
Be_Neighbors_Refuse	Sent to reply to a Be_Neighbor_Request
Be_Neighbors_Confirm	Sent to reply to a Be_Neighbor_Request
Hello_Message	Sent after Be_Neighbor_Confirm is received and includes parameters for communcation
Cease_Neighbors_Request	Sent by a current neighbor router to stop being neighbors
Cease_Neighbors_Confirm	Sent to reply to a Cease_Neighbors_Request

Alive Messages	
Packet Type	Description
Alive	Sent periodically by one router to a neighbor router to test whether the neighbor is still alive (according to a predefined time interval)
Alive_Ack	Reply to an Alive message to acknowledge that the router is still alive

Link Cost Messages	
Packet Type	Description
Link_Cost_Request	Sent periodically (every x seconds) to each neighboring router to determine link cost
Link_Cost_Ack	Reply to a Link_Cost_Request; the average time of all replies within the Update_Interval are averaged for the link cost between routers

Link State Advertisement Messages	
Packet Type	Description

Link_State_Advertisement	Sent periodically by a router to its neighbors and then reliably flooded to all other nodes in the network in the same way; the format follows the required one provided in the project description document and includes the following information Router_ID, LSA_Type, LS_Age, LS_Sequence_Number, Length, Number_of_Links, Link_ID, Link_Data
Link_State_Ack	Reply to sender of Link_State_Advertisement (and then floods neighbors with the original Link_State_Advertisement)

File Transfer Messages	
Packet Type	Description
File	Sent when an end user wishes to send a file to another end user. This message type will include Num_Packets so that the receiver will know when it has received the full file
File_Ack	Reply to the original end user to acknowledge that the packet was received

Error Messages	
Packet Type	Description
Error	Sent when a receiver detects a corrupted packet (according to the CRC) to request that a certain packet is resent

Error Checking

To check for packet transmission errors, the packet layer of the sending router will compute the CRC of the packet and append it to the end of the packet. The packet layer of the receiving router will compute the CRC of the packet and check this value against the CRC provided. If the values match, then the receiver will send an acknowledgement. If not, the receiver will generate and send an error message requesting that the corrupt packet be resent. This ensures that the socket layer does no error checking, and simply receives or sends packets and forwards them to the packet layer.

Flow Control

When a single file or message consists of multiple packets, we propose to implement Stop-and-Wait flow control to ensure that all packets are received and in order. We have chosen this form of flow control as it is simple, efficient, and ensures that all packets are delivered to the receiving router for reassembly.

Bootstrapping and Initial Setup

The end user will start the user interface at the end system/router using a predetermined port number. The application layer will then read the policy management file containing the configuration parameters. The application layer will send the packet layer a Config_Params message with this information. The packet layer will receive the Config_Params message and initialize internal parameters based on this message. The packet layer will also initialize the Link State Database, Routing Table, Cost Table and Packet Buffer. Based on the neighbors in the policy management file, the router will send out neighbor acquisition messages to adjacent routers in the form of Be_Neighbors_Request messages. The other routers will receive these request messages and either send Be_Neighbor_Accept messages to initialize communication parameters via a Hello_Message or send a Be_neighbor_deny message (based on their policy management file) and do nothing further.

The router's packet layer will update its Link State Database and Cost Table (with constant link costs initially) then send Link_State_Adverstisement messages to its new neighbors which will be forwarded to the entire network via reliable flooding. The packet layer will also begin sending out Link_Cost_Requests to all neighbors every x seconds. When the first Update_Interval is reached, the router will update the Link Cost Database and sends out Link_State_Advertisement messages to neighbor routers (and so on with reliable flooding). This completes the initial setup process. From then on, the router will periodically receives Link_State_Advertisements from neighbors. The router will verify that it is not corrupted and then install the new Link_State_Advertisement into its Link State Database, update its Routing Table, send an Link_State_Ack to the sender and also resend the Link_State_Advertisement to all other neighbors.

Periodic Network Activity

Periodically, every x number of seconds, each router sends a Link_Cost_Request message to each of its neighboring routers. Each neighbor will reply with a Link_Cost_Ack. Every Update_Interval, these times calculated every x seconds will be averaged and updated in the Cost Table.

Periodically, every Alive_Max number of seconds agreed upon during neighbor acquisition, each router sends a Alive message to each of its neighboring routers. Each neighbor will reply with a Alive_Ack. If no Alive_Ack is received, the sending router will declare the link dead, requiring an update to the Link State Database.

Any time that a router needs to update its local link state database, it will need to send a Link_State_Advertisement to all of its neighboring routers, and so on via reliable flooding.

Occasional Network Activity

End users will occasionally send files to other end users and thus other end users will occasionally receive files. This means that File and File_Ack type packets will be sent through the network at random times. Since we will implement the capability to change the policy management file, the network topology may be altered at anytime. If new routers come online at various time points, Neighbor Acquisition type messages will be sent through the network at random times.

Programming Language

We propose to use python as the programming language for this project, if granted permission. We will utilize the python socket API for low-level network interfacing.