CS 2520
Project 1
Design Report
Joe Baker, Evangelos Karageorgos

Simple Link State Router

Usage

The router can be started on the command line. It will run the main logic in background threads and a UI thread to print and change router state. Each router should be run on its own system (this could be a VM or container). The router accepts command line arguments for initial configuration:

```
./route <router_ip> <file_dir> [link_list=<file_path>] [port=<int>] [corrupt_msgs]
```

`router_ip`	Required. The IP address the router will bind to. Acts as this router's ID.
`file_dir`	Required. Absolute path to a directory that the router will read and write files from for scp functionality.
`port=<int>`	Optional. Specify a port for routed to use for router-to-router communication over TCP and UDP. Default is 50001.
`corrupt_msgs`	Optional. When provided, the router will simulate both packet and network corruption.
`link_list=<file_path>`	Optional. `file_path` is an absolute path to a text file. Each row in the text file is a link connected to the router with all of its parameters. The router will add all of these links at startup for quick setup.

Examples:

- `./route 10.0.0.1 /home/user/route_dir link_list=/home/user/route_links.txt port=123 corrupt_msgs`
    - Starts the router on port 123, initialized with the links defined in `route_links.txt`. Packet corruption will be simulated. Files in `/home/user/route_dir` are used with the `scp` command.
- `./route 192.168.0.2 /home/user/route_dir`
    - Starts the router on port 50001. Does not simulate packet corruption and does not initialize any links.

CS 2520
Project 1
Design Report
Joe Baker, Evangelos Karageorgos

User Interface

Overview

The user interface is a separate thread which allows users to modify the router configuration, view router state, and transmit a file over the network of routers. After the router starts up the UI thread manages standard I/O and presents a simple CLI with commands to do each of the above actions and a help menu.

UI Commands

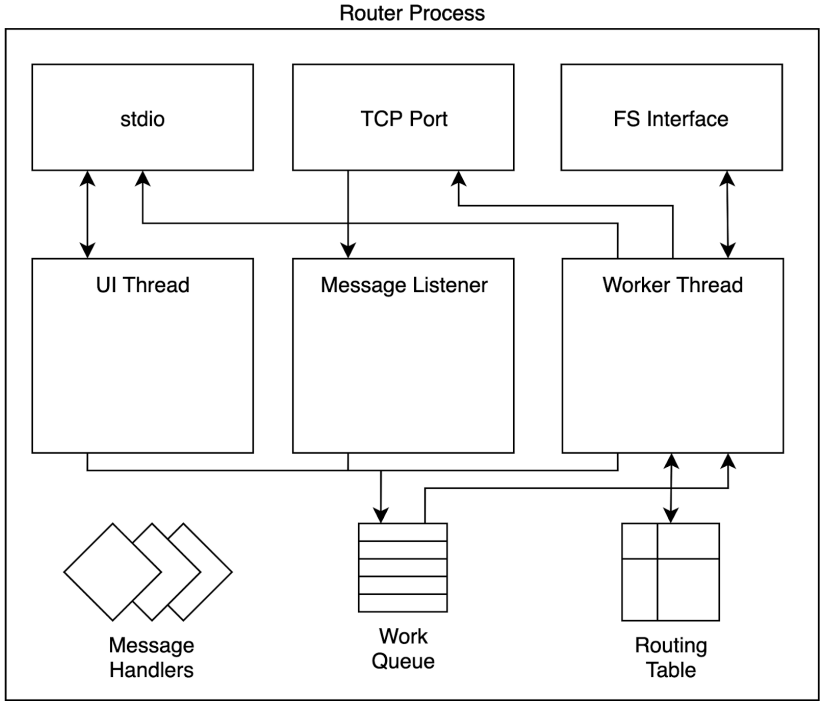
Command	Description
`help [command]`	If `command` is NULL, displays a summary of all commands. If `command` matches a known command, displays a detailed description of that command.
`scp <file_path> <destination_ip>`	Transfers the file at `file_path`, from via the router process, through the sLSRP network, to the filesystem on the router at `destination_ip`.
`set-version <version>`	Change the sLSRP version. `version` must be an integer greater than 0.
`set-link <router_ip> <cost> <hello> <update>`	Updates the local router's link state database. If the `router_ip` does not exist, this creates a new entry. To delete a link, specify the correct `router_ip` and set `cost`, `hello`, and `update` to 0. Otherwise, all values must be greater than 0.
`show-path <destination_ip>`	Computes the shortest path to `destination_ip` using the current routing table, and displays the total cost and each hop in the path.
`show-msgs`	Prints the messages sent and received by this router. This can be interrupted to return to the CLI with `ctrl-c`.

CS 2520
Project 1
Design Report
Joe Baker, Evangelos Karageorgos

Components

Overview

The majority of sLSRP takes place in our background threads. They interface with a UI thread and the network to share state information, maintain the links to other routers, handle messages from other routers, perform the core LSRP algorithm, and send files to other routers leveraging the routing table and links. The router implementation is decomposed into threads for receiving and processing messages. Additionally, the logic to process each message is abstracted into message handlers for extensibility. The routing table is kept separate and simple with a limited public interface and no locking required for easy modification. This modular design makes the router more extensible, and easier to build as a team.



Message Listener

The router listens for sLSRP messages using the TCP protocols on a configurable IP address and port. The message listener runs in it's own thread. The message listener is responsible for receiving messages, validating checksums (or introducing errors for debugging), and quickly forwarding them to the router's work thread queue. The message listener uses TCP to transport sLSRP messages for routing functionality and file messages for scp functionality.

CS 2520
Project 1
Design Report
Joe Baker, Evangelos Karageorgos

Worker Thread

In addition to the message listener thread, the router also has a worker thread. The worker thread consumes messages placed on the queue by the UI and listener threads. When the worker thread consumes a new task, it de-multiplexes the message into an appropriate message handler. The worker queue can be accessed by the worker thread and the UI and message listener threads so it must be protected by a locking mechanism. The worker thread also monitors when sLSRP Hello messages need to be sent and insert message handlers to the work queue to broadcast these messages.

UI Thread

The UI thread handles the event loop that drives the CLI. It then parses input and places work items on the work queue.

Message Handlers

Message handlers are functions that process the payload of a message that the router receives, and respond by sending messages of their own. Specific message handlers will need to be implemented to handle each sLSRP message type, file transfer messages, and UI requests. Message handlers for received sLSRP Update messages are placed on the queue instead of handled immediately in the listener thread. This reduces the response time, but helps the router to accurately express its current load to the requestor. Thus our router considers its current load as part of its link cost.

Routing Table

The routing table stores a map of all known routers, links, and costs. It also implements the core link state routing algorithm (Dijsktra's shortest path). The routing table exposes an interface to message handlers so they can update the table and query it for the shortest path. The routing table is only expected to be accessed by a message handler running on the single worker thread, so it does not need to be protected by locking mechanisms.

CS 2520
Project 1
Design Report
Joe Baker, Evangelos Karageorgos

File Transport Messages

The file transfer service will consist of messages to initiate a transfer operation, send the data over, and finalize the operation. Specifically, we will have 3 types of messages:

1. FileTransferInit:
    - transferID
    - Filename
2. FileTransferChunk
    - transferID
    - sequenceNo
    - isLastChunk
    - Chunk data
3. FileTransferAck
    - transferID
    - sequenceNo

One router can push a file towards another router under the command of the UI. First, the source router selects a random transferID and sends an initial FileTransferInit packet. After that, the source will keep sending FileTransferChunk packets with appropriate sequence numbers (they start at 1), the file fragment data, and the isLastChunk flag set to true only on the last chunk of the file. The destination router must acknowledge every correctly received packet with FileTransferAck (sequenceNo=0 for the FileTransferInit).

The file transfer packets and sLSRP packets travel between routers over the same low-level TCP connection, but are distinct messages. Because the file is split into packets and each packet can take a different route due to routing table changes or debug-level packet corruption, we need multiple file transfer message types.