CS 2520
Project 1
Final Report
Joe Baker, Evangelos Karageorgos

Simple Link State Router

Usage

The router can be started on the command line. For easier deployment with Docker containers, see the README.md in the project repository. 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:

```
./routed <router_ip> <file_dir> [link_list=<file_path>]
[corrupt_msgs=<bool>] [auto_start=<bool>] [log_file=<file_path>]
```

`router_ip`	Required. The IP address the router will bind to. Acts as this router's ID.
`file_dir`	Required. Relative or absolute path to a directory that the router will read and write files from for scp functionality.
`link_list`	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.
`corrupt_msgs`	Optional. When set to 1, the router will simulate both packet and network corruption.
`auto_start`	Optional. When set to 1, the router will start automatically.
`log_file`	Optional. Absolute path to a file to write logs to (default: stderr).

CS 2520
Project 1
Final Report
Joe Baker, Evangelos Karageorgos

Examples:

- `./routed 10.0.0.1 /home/user/route_dir link_list=/home/user/route_links.txt corrupt_msgs=1`
    - Starts the router on port 5678, 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.
- `./routed 192.168.0.2 /home/user/route_dir`
    - Starts the router on port 5678. Does not simulate packet corruption and does not initialize any links.

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.

Links in our routing protocol are directed edges in a graph. To establish duplex connections between two routers, two separate links must be configured between each router.

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.
`start`	Start communicating with other routers. Configuration cannot be changed manually after this point.
`show-graph`	Prints this node's view of the current network graph.

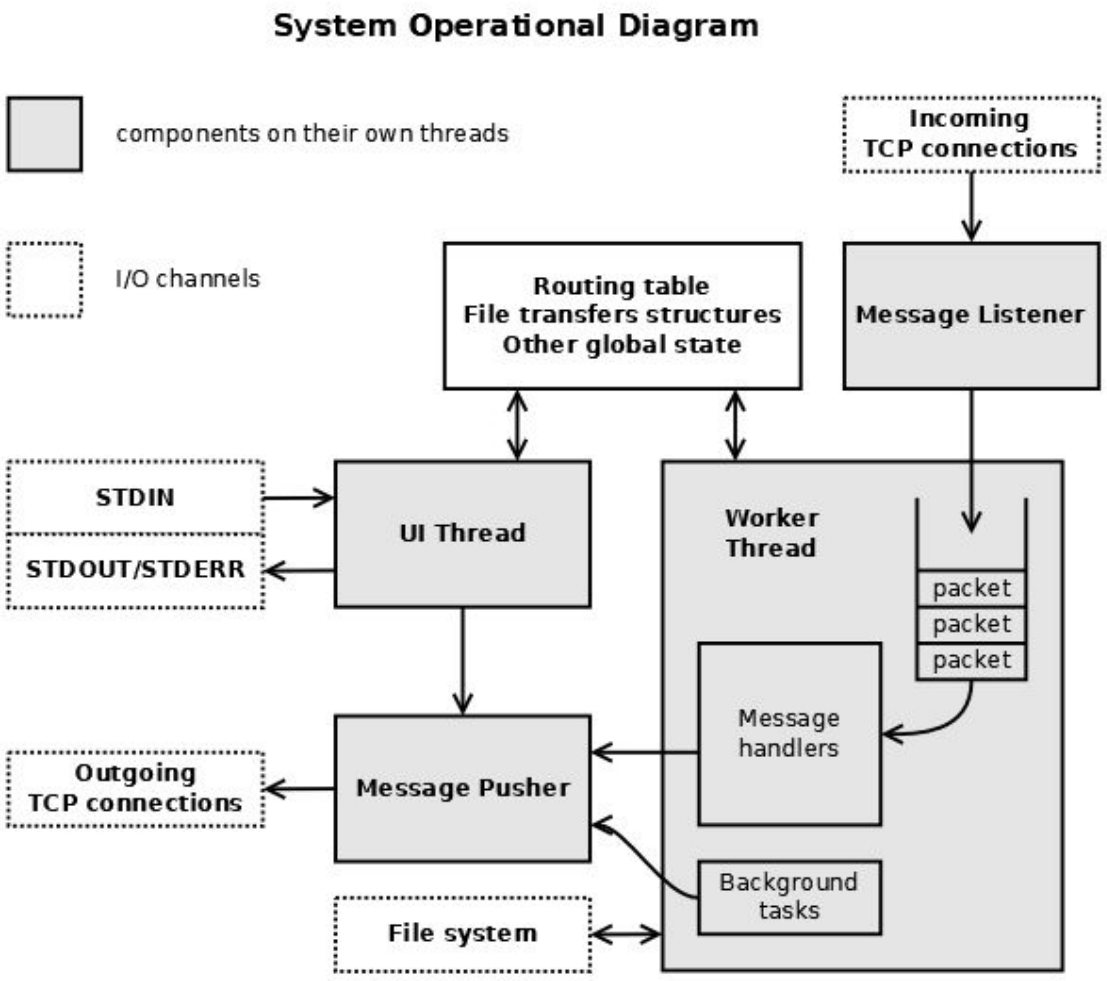
`scp <file_path> <destination_ip>`	Transfers the file at `file_path`, from via the router process, through the sLSRP network, to the filesystem on the router at `destination_ip`.
`set-version <version>`	Change the sLSRP version. `version` must be an integer greater than 0.
`set-link <link_id> <destination_router_ip> <cost> <hello> <update>`	Updates the local router's link state database. If the `destination_router_ip` does not exist, this creates a new entry. To delete a link, specify the correct `destination_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.

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.

CS 2520
Project 1
Final Report
Joe Baker, Evangelos Karageorgos



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
Final 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 listener thread. When the worker thread consumes a new task, it de-multiplexes the message into an appropriate message handler. The Message Listener thread places messages into the Worker Thread's queue, so the Worker Thread is protected with a lock. The worker thread also has tasks that are run on a fixed interval of 1 second regardless of the size of the work queue. These tasks check timeouts for pending message ACKs, advance the state machines for link states and file transfer states, and send Hello and Cost Update messages on their set intervals. This is the main component that is responsible for reacting to incoming packets. It has a number of message handlers that correspond to different packet types.

UI Thread

The UI thread handles the event loop that drives the CLI. It then parses input and executes user commands. It is implemented as a separate thread so the user can monitor the state of the links, test to see the current path through the network, and initiate file transfers.

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 were implemented to handle each sLSRP message type and file transfer messages. Message handlers for received sLSRP Hello and Cost Update messages are placed on the same work queue as all others instead of handled immediately in the listener thread. This reduces the response time, andhelps 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 whole shortest path for a user request, or the next hop in the shortest path to the destination for routing file transfer messages. 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
Final Report
Joe Baker, Evangelos Karageorgos

Message Pusher

This component is responsible for sending packets across a link. It can send a packet both synchronously and asynchronously. It makes sure the packets are sent successfully, by retransmitting them in case of integrity errors.

File Transport protocol

The file transfer packets and sLSRP packets travel between routers over the same low-level TCP connection, but they 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.

The file transfer service consists of messages to initiate a transfer operation and send the data and their acknowledgement messages.

Specifically, we have 4 types of messages:

1. `FileInit`:
    - `SessionID` - a unique number identifying the operation
    - `source` - the sender router id
    - `destination` - the receiver router id
    - `filename` - the name of the file
    - `number_of_chunks` - the number of file chunks
2. `FileInit_Response`
    - `SessionID` - the identifying session id
    - `source` - the sender router id
    - `destination` - the receiver router id
3. `FileChunk`
    - `SessionID` - the identifying session id
    - `source` - the sender router id
    - `destination` - the receiver router id
    - `sequence_no` - the sequence number identifying the chunk
    - `chunk_size` - the size of the chunk in bytes
    - `chunk_data` - the chunk data stream

CS 2520
Project 1
Final Report
Joe Baker, Evangelos Karageorgos

4. `FileChunk_Response`
    - `SessionID` - the identifying session id
    - `source` - the sender router id
    - `destination` - the receiver router id
    - `sequence_no` - the sequence number identifying the chunk

The file transfer functionality involves two routers, the sender and receiver, and four packet types, `FileInit`, `FileChunk`, `FileInit_Response` and `FileChunk_Response`. First, the transfer is initiated with the successful exchange of `FileInit` and `FileInit_Response` messages between the sender and receiver, and then the fragments of the file are sent in individual chunks using exchanges of `FileChunk` and `FileChunk_Response` messages. Both sender and receiver maintain state for the transfer while it's ongoing, and the process goes like this:

- In order to send a file, the sender router (source) initializes an `OutgoingFileTransfer` structure for that file and initiates the transfer by sending a FileInit message towards the destination.
- The receiving router (destination), upon receiving the `FileInit` message, initializes an `IncomingFileTransfer` structure for the file and answers by sending a `FileInit_Response` message back towards the sender.
- When the sender gets the `FileInit_Response` message, it starts sending the file chunks independently, with a `FileChunk` message for every chunk.
- When the receiver gets a `FileChunk` message, it identifies the specific file transfer operation and chunk sequence number, it reads the chunk data and acknowledges the chunk by sending a `FileChunk_Response` message back to the sender.
- Both the sender and the receiver know the number of chunks, because the initial `FileInit` message had this information.
- They are also both aware of the chunks that have already sent/received, since they keep track of them.
- When the sender gets an acknowledgement for all the chunks, it considers the file transfer operation successful and destroys the `OutgoingFileTransfer` structure for that file.
- In the same manner, when the receiver gets all the chunks, it creates the file on the disk and considers the transfer successful by destroying the `IncomingFileTransfer` structure for the file.

CS 2520
Project 1
Final Report
Joe Baker, Evangelos Karageorgos

The protocol accounts for lost or duplicate packets every step of the way.

- Every file transfer operation is uniquely identified by a session ID and the source router ID. This means that duplicate messages will never cause a problem on the sender or the receiver.
- If the `FileInit` packet gets lost, it will be sent again. Specifically, if the sender doesn't get the `FileInit_Response` message after a timeout, it will send the `FileInit` message again.
- If the receiver gets a `FileInit` message for a specific file transfer operation multiple times, it will always reply with a `FileInit_Response`.
- Similarly, if a `FileChunk` message or a `FileChunk_Response` message gets lost, it will be accounted for by retransmission.
- Also, if a `FileChunk` message is received twice for the same chunk, the receiver will only take the first into account, while responding to both of them.

All messages relevant to a file transfer operation will be routed to their destination, utilizing the routing algorithm.

- Every file-related message (`FileInit`, `FileInit_Response`, `FileChunk`, `FileChunk_Response`) has a source and destination field.
- The messages `FileInit` and `FileChunk` are supposed to be routed towards the destination, while the messages `FileInit_Response` and `FileChunk_Response` are supposed to be routed towards the source.
- Any router that is neither the source or the destination, upon receiving one of these packet types, it will retransmit the packet to the appropriate neighbour link, utilizing the routing table.

If a rogue `FileInit_Response` or `FileChunk_Response` message gets to the sender after the `OutgoingFileTransfer` structure gets destroyed, it will be silently ignored.
Similarly, If a rogue FileInit of `FileChunk` message gets to the receiver after the `IncomingFileTransfer` structure gets destroyed, it will automatically respond with `FileInit_Response` or `FileChunk_Response` messages accordingly.

CS 2520
Project 1
Final Report
Joe Baker, Evangelos Karageorgos

Message transmission

All the messages that the protocol utilizes share a common form and transfer mechanism. Specifically, the form of the packet of any message is this:

`routerID`	The id of the router that sent the packet. Since the ID is the device IP, it is sent as a string (32 bit integer length and character array)
`packet_type`	The identifier of the type of message (32 bit integer)
`data`	The payload of the message (variable length byte array)
`checksum`	A CRC32 checksum of the rest of the packet (32 bit integer)

When the Message Pusher component tries to send the packet to an outgoing link, it calculates its checksum and attempts to send it across the link using a TCP socket. If we have set the command-line argument `corrupt_msgs=1`, we will occasionally simulate packet corruption or packet loss, by randomly flipping a bit on the packet or by not sending the packet at all. The protocol makes sure that if there is packet corruption (the checksum doesn't agree with the calculated CRC for the packet), a retransmission of the packet will be triggered. The transmission cycle is performed in the sender's `Message Pusher` component and the receiver's `Worker Thread` component

LSRP Messages

All messages sent in LSRP are tracked with timers that are evaluated in the fixed interval part of our worker thread. That is, every 1 second, the state of all pending messages are checked for timeouts, and retransmissions or state adjustments are made as needed.

File Transport Messages

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.

Link State Messages

The link state messages are the core messages that keep routers connected. LSA messages are created when any link on the router changes (cost update, activated/deactivated links from alive checks). The LSA message contains the fields listed in the assignment, including an entry for every link on the router. When a router receives a valid LSA message, it will update its own routing table and forward the message to all its peers except the originating link.

All messages in this part of the protocol use eventual consistency to make forward progress in the global state of the router network. This allows our protocol to operate in the face of dropped, delayed, or corrupted packets. Forward progress for these messages is tracked by using a monotonically increasing sequence number in each message when it is created at the advertising router. When routers receive a LSA message, they check if the message originated from themselves, or if the message is out of date by keeping a map of the last sequence number received from each other router ID. Any message that is too old, or would cause a loop, is dropped. However, each router *always* sends an ACK back to the sending router to confirm receipt if the message could be parsed.

BeNeighbor Messages

BeNeighbor messages are sent when a link is first created, or in an attempt to bring a dead link back into commission. BeNeighbor messages contain the fields listed in the assignment, including the router LSRP protocol version. When a router receives a BeNeighbor message, it checks the version field. If the router versions match BeNeighborAccept is sent in response. Otherwise, BeNeighborReject is sent. When the requesting router receives a BeNeighborReject or no response after a timeout, the link is set to DEAD state, removed from the routing table, and a new LSA is sent. DEAD links will retry a BeNeighbor request during a new Hello Interval. BeNeighborAccept messages also cause the router to update their routing table and send an LSA. In this case the link moves to ALIVE and the Hello Interval timer is again set, but in this case to check that the link is still active.

Because links are directed edges in our implementation, we chose to not implement CeaseNeighbor messages. This approach does not cause the overall behavior of the implementation to deviate from the assignment's requirements as CeaseNeighbor messages in this implementation would have no impact because the destination router would have no state to update.

CS 2520
Project 1
Final Report
Joe Baker, Evangelos Karageorgos

Alive Messages

When the Hello Interval timer expires, either due to an ALIVE or DEAD link timing out, an Alive message is sent over that link. If a response is received before a timeout, the link moves to the ALIVE state. If the previous state was DEAD, the routing table is updated and a new LSA is broadcasted.

Cost Update Messages

Cost Update messages test the network latency for a link to determine a current approximation of cost to use that link. When a link becomes ALIVE, a Cost Update timer is set. When that timer expires, the link is moved to the COST_UPDATE state. During the COST_UPDATE state, the link is still usable for routing.

A new CostPing message is put on the work queue,and when a response is received the links ping counter is updated and a new CostPing message is put on the work queue. Finally when the Cost Update Interval is exceeded, the CostPing messages are stopped. If the number of pings was 0, the link moves to DEAD state, removed from the routing table, and an LSA is broadcasted.

If the ping count was 1 or greater, the new approximate cost is calculated. The best cost is 1 and the worst cost is anything larger. The new cost for each link is updated as the max number of pings for any links on this router divided by the number of pings for this link. This formula is applied for all local links. This formula gives the best cost of 1 to the fastest link. After calculating new costs, the routing table is updated and an LSA is broadcasted.