Network Node Control System

Design Overview

The Network Node Control System is a Python-based application designed to manage and control network nodes. It facilitates communication, link-state advertisement, and dynamic updates to the network topology. The system is implemented using a combination of threads and sockets, allowing it to maintain connections to neighboring nodes, share link-state information, and manage the node's operation.

Components

Node Configuration

The system starts by loading node details from a configuration file, including the node's name, UUID, and information about its peers (neighbors).

UUIDs are unique identifiers used to distinguish nodes and establish connections.

This information is stored in a Python dictionary for easy access and modification.

Communication

The system uses UDP (User Datagram Protocol) to establish communication between nodes.

Threads are employed to manage both client and server roles, enabling the sending and receiving of messages.

KeepAlive

A "ping" and "pong" mechanism ensures the liveness of neighboring nodes.

Nodes send "ping" messages to their neighbors and expect "pong" responses.

This mechanism helps in detecting inactive or disconnected neighbors.

Link-State Advertisement

Nodes periodically send link-state advertisements to their neighbors. These advertisements contain information about the node's identity, sequence number, and its immediate neighbors.

The sequence number helps nodes determine the freshness of link-state information.

The system dynamically updates a network map based on the received link-state advertisements.

Routing

The system employs Dijkstra's algorithm to calculate the shortest path and metrics to other nodes.

The network map is continually updated as link-state advertisements are received, reflecting changes in the network topology.

User Interaction

The system provides a command-line interface to allow users to perform actions such as viewing neighbors, modifying node details, and querying network metrics.

Operation

Initialization

The system initializes by loading node details from a configuration file.

Threads are started for server, client, and message processing roles.

Server threads bind to specific ports to listen for incoming messages.

Message Processing

A separate thread continuously receives and processes messages.

Incoming "ping" messages are responded to with "pong," ensuring liveness.

Link-state advertisements are processed, and the network map is updated accordingly.

Link-State Advertisement

A separate thread is responsible for periodically sending link-state advertisements to neighbors.

This ensures that neighbors are kept informed about the node's status and changes in its neighborhood.

User Interaction

Users can interact with the system via the command-line interface to view neighbors and modify the node's information.

The system provides network metrics, including the shortest path to other nodes.

Conclusion

The Network Node Control System provides a flexible and robust solution for managing network nodes and dynamically adapting to changes in the network topology. By using a combination of threads and sockets, it effectively handles communication, link-state advertisements, and routing metrics, ensuring reliable and efficient node management.

This system is well-suited for various network management tasks and can be easily extended to include additional features and capabilities as needed.

Functions and their brief mechanism:

dijkstra(network_map, start_node)

Function to calculate the shortest path and metrics in the network using Dijkstra's algorithm.

Initializes distance metrics and updates them based on link-state information.

Removes self-node and distances of 0, and arranges metrics in ascending order.

send_link_state_advertisement(node_details)

Continuously sends link-state advertisements to neighbors.

Generates sequence numbers, prepares advertisements, and sends them.

Ensures that neighbors receive updated link-state information.

send_advertisement(advertisement, server_ip, server_port)

Sends a link-state advertisement to a specific neighbor.

Converts the advertisement to JSON and transmits it via a UDP socket.

handle_received_advertisement(node_details, advertisement_json)

Handles incoming link-state advertisements.

Updates the network map and sequence numbers.

Forwards advertisements to own neighbors to keep them informed.

neighbor(node_details, response, neighbor_details, server_port)

Function used in the client thread to identify and manage neighbors.

Extracts information about neighboring nodes based on responses received.

connect_to_server(node_details, server_ip, server_port)

Initiates a client thread to connect to a specific neighbor.

Used during the system's initialization to establish connections to neighbors.

connect_to_servers(node_details, servers)

Connects to multiple neighbors in parallel by creating and managing client threads.

Facilitates efficient neighbor connection during system startup.

add_neighbor(node_details, neighbor_data, link_state_details, client_threads)

Function for adding a new neighbor to the node's configuration.

Parses neighbor data and establishes a connection to the new neighbor.

Modifies the node's details and client threads accordingly.

uuid(node_details)

Extracts and stores the UUID of the current node from its configuration.

Stores the UUID in a global dictionary for easy access.

user_input_thread(node_details, link_state_details)

A thread handling user input via a command-line interface.

Allows users to view neighbors, modify node details, and query network metrics.

Executes user commands in real-time.

run_client(node_details, server_ip, server_port)

Manages the client-side of node-to-node communication.

Implements a "ping" and "pong" mechanism to maintain connectivity with neighbors.

Detects inactive or disconnected neighbors and updates the neighbor details.

run_server(node_details)

Operates as a server to receive incoming messages from neighbors.

Responds to "ping" messages with "pong" and processes link-state advertisements.

Handles socket-level communication and message routing within the node.

load_node_details(filename)

Loads node details from a configuration file, including the node's name, UUID, and peer information.

Returns a dictionary containing node-specific data.

load_link_state(node_details)

Extracts link-state information from node details, including neighbor UUIDs and distance metrics.

Provides a structured representation of the link-state information.

These functions collectively facilitate communication, link-state advertisement, neighbor management, and user interaction within the Network Node Control System.