Project 2: Content Distribution - Design Document

Introduction

This design document details the implementation of a content distribution server that uses a link-state routing protocol to discover network topology and compute shortest paths. The server implements keepalive-based neighbor detection, LSA flooding for topology dissemination, and Dijkstra's algorithm for path computation. The implementation is structured around five concurrent threads that handle message reception, periodic keepalive transmission, LSA flooding, neighbor timeout detection, and command processing.

Keepalive Message Implementation

The keepalive mechanism enables nodes to detect neighbor reachability through periodic heartbeat messages sent via UDP. The JSON heartbeat messages include the following information: “type”, “uuid”, “name” and “sequence_number”.

The timing parameters are sending one keepalive every 3 seconds, and a timeout threshold of 9 seconds. This is equivalent to 3 missed keepalives.

This is performed by a keepalive loop thread that sends keepalive messages to all peers regardless of active status. When a peer receives a keepalive, the last_seen timestamp in the neighbours dictionary is updated to reflect that. A separate timeout_loop thread runs to remove any neighbor from the dictionary whose last seen entry exceeds the timeout interval (9 seconds).

This dictionary implementation includes automatic bidirectional updates even if only 1 side configures the relationship. This means if a node receives keepalives from an unconfigured peer with the addneighbor command, it will automatically add that peer to its conf with a default metric of 1. E.g. A adds B unilaterally, sends keepalives to B, and once B receives a keepalive from unknown peer it adds A automatically.

Link-State Advertisement Implementation

The LSA protocol implements flooding-based topology discovery, allowing each node to construct a complete network graph. The JSON messages include the following information: “type”, “uuid”, “name”, “sequence_number” and “neighbors” dictionary.

LSA loop thread generates a new LSA every 5 seconds including only active neighbors from the neighbors dictionary. Each LSA has an incrementing seq number and is included in the link state database (LSDB) before being flooded to all active neighbors. The flooding protocol is as follows. When a LSA is received, the node compares its seq number against the stored value.

- Higher sequence number → Accept LSA and forward to all neighbors except sender
- Lower or equal sequence number → Discard LSA

Sender identification is done using UDP source address / port matching against peer configs to prevent forwarding loops. This is done so each LSA spreads through the network exactly every time it's updated.

The LSDB maps each node's UUID to its latest LSA (seq num and neighbors dictionary). This is kept so the reconstruction of the network graph can be accomplished by the map and rank.

Shortest Path Computation and Map Generation

Dijkstra's Algorithm: The rank command uses a standard Dijkstra implementation operating on a graph constructed from the LSDB. Each node's adjacency list comes directly from its LSA's neighbors dictionary. The algorithm computes shortest distances from the current node to all reachable nodes.

Consistent Reachable Map: The get_map() function first runs Dijkstra to identify all reachable nodes (distance < ∞). Only reachable nodes are included in the output map, and edges are filtered to point exclusively to other reachable nodes. This prevents stale LSDB entries from creating inconsistent maps when nodes become unreachable but their cached LSAs remain in the database. This is done by building a reachable_nodes set by filtering Dijkstra results for non infinite distance nodes. The map output iterates only over nodes in reachable_nodes, and for each node, only includes edges to neighbors that are also in reachable_nodes.

Libraries Used

All libraries are part of Python 3's standard library (socket, threading, JSON, argparse, time, sys).

Design Decisions

Concurrency: The implementation uses five concurrent threads to handle different responsibilities. Main thread (processes stdin commands and output), receive_loop (receives and sends UDP messages (keepalive/LSA)), keepalive_loop (sends keepalives every 3 seconds), lsa_loop (sends LSAs every 5 seconds) and timeout_loop (checks neighbor timeouts every second). All shared state (peers, neighbors, lsdb, neighbor_names) is protected by a single lock acquired by all command handlers and message processors. This ensures atomicity of state updates and prevents race conditions between threads accessing the same data structures.

Graceful Termination: The kill command sets a flag stopping all background threads and closes the socket cleanly without sending termination messages to neighbors. Neighbor nodes detect the failure through keepalive timeouts.

LSDB Persistence: The LSDB retains entries even after nodes timeout as neighbors. Old LSAs remain until overwritten by higher sequence numbers from that node. The reachability filtering in get_map() makes sure only reachable nodes appear in output while maintaining relevant information for potential reconnections. This means if a node comes back online, its previous LSA information is still available until it sends an updated LSA.