SDN Controller Design Document

Created: May 09, 2025

Version 1.0

Table of Contents

1. Introduction
2. Architecture Overview
3. Core Components
4. Routing Algorithms
5. Flow Management
6. Policy Implementation
7. Visualization Components
8. Command Line Interface
9. Performance Considerations

1. Introduction

This document provides a comprehensive overview of the design and implementation of our Software-Defined Networking (SDN) controller. The controller is designed to centralize network control logic, enabling dynamic management of network resources and traffic flows.

The SDN paradigm separates the network control plane from the data plane, allowing for more flexible and programmable network management. Our controller implements this separation by maintaining a global view of the network topology and making centralized routing decisions that are then pushed to network devices as flow table entries.

Key features of our SDN controller include:
- Network topology discovery and maintenance
- Shortest path computation and traffic engineering
- Flow table management for network devices
- Link failure detection and path reconfiguration
- Traffic prioritization and quality of service enforcement
- Load balancing across multiple paths
- Visualization of network state and traffic flows

2. Architecture Overview

The SDN controller follows a modular architecture to ensure extensibility and maintainability. The architecture consists of the following high-level layers:

Has a unique hash that is used to uniquiley indentify this document as relating to my work and and computed based on my student ID.

1. Core Controller Layer: Maintains the network state and handles core functions
2. Network Abstraction Layer: Provides abstractions for interacting with network devices
3. Routing and Policy Layer: Implements algorithms for path computation and policy enforcement
4. Visualization Layer: Renders network state and statistics for monitoring

5. CLI Interface Layer: Exposes controller functionality to operators

The controller uses a graph-based representation of the network internally, with nodes representing switches and edges representing links between them. This representation facilitates efficient path computation and network state analysis.

3. Core Components

3.1 Topology Management

The Topology class manages the network graph representation. It provides methods to:

- Add and remove nodes (switches) from the network
- Add and remove links between nodes with associated attributes (bandwidth, latency)
- Query the network state (existing nodes, links, and their properties)
- Compute paths between nodes using various algorithms

Below is a snippet of code from earlier in the development process, when I originally intended to use NX graph but it didn't work properly because of a version error so I had to handle some of my own graph code.

```
import networkx as nx
import matplotlib.pyplot as plt

class Topology:
    def __init__(self):
        """Initialize an empty network topology."""
        self.graph = nx.Graph()

    def add_node(self, node_id):
        """Add a node to the topology."""
        if not self.graph.has_node(node_id):
            self.graph.add_node(node_id)
            return True
        return False

    def add_edge(self, source, destination, bandwidth):
        """Add an edge between two nodes with given bandwidth."""
        if not self.graph.has_node(source):
            self.add_node(source)
        if not self.graph.has_node(destination):
```

```
            self.add_node(destination)

        self.graph.add_edge(source, destination, weight=(1/bandwidth), bandwidth=bandwidth)
        # Add reverse direction for bidirectional links
        self.graph.add_edge(destination, source, weight=(1/bandwidth), bandwidth=bandwidth)

    def remove_edge(self, source, destination):
        """Remove an edge from the topology."""
        if self.has_edge(source, destination):
            self.graph.remove_edge(source, destination)
            self.graph.remove_edge(destination, source)
            return True
        return False
```

Arguably, this was the biggest development hurdle because it slowed me down so much.

The topology is represented using a custom graph implementation with adjacency lists and dictionaries, allowing for efficient storage and lookup of network elements.

3.2 Flow Table Management

The FlowTable class handles the management of flow entries that will be installed on network devices. Key functionality includes:

- Creating and validating flow entries based on match and action fields
- Storing flow entries associated with each switch
- Retrieval of flows based on various criteria (switch ID, match fields, etc.) - Prioritization of flows based on policy requirements

Each flow entry contains match fields (source, destination, etc.), actions (forward to port, drop, etc.), and metadata like priority and statistics.

4. Routing Algorithms

4.1 Shortest Path Algorithm

The controller uses Dijkstra's algorithm to compute the shortest path between any two nodes in the network. Our implementation uses a priority queue (min-heap) to efficiently select the node with the minimum distance in each iteration.

The algorithm assigns weights to links based on the inverse of their bandwidth, meaning that higher bandwidth links are preferred over lower bandwidth ones. This approach ensures that paths with higher capacity are selected when available.

Key aspects of our implementation:

- Time complexity: O((E + V) log V) where E is the number of edges and V is the number ofvertices
- Space complexity: O(V) for storing distances and previous nodes
- Support for arbitrary link weights based on various metrics (bandwidth, latency, etc.)

4.2 Multi-Path Discovery

For load balancing and backup path purposes, the controller can find multiple paths between source and destination using a depth-first search (DFS) based approach. The algorithm finds all simple paths between two nodes and sorts them based on a cost function.

This allows the controller to:

- Distribute traffic across multiple paths to avoid congestion
- Have backup paths ready in case of primary path failure
- Select paths based on specific traffic requirements (latency, bandwidth, etc.)

The implementation limits the number of paths returned to avoid excessive computation for densely connected networks.

5. Flow Management

Flow management is a core responsibility of the SDN controller. When a new flow needs to be established between two endpoints, the controller:

1. Computes the best path based on current network conditions and policies
2. Generates appropriate flow table entries for each switch along the path
3. Installs these entries on the switches via the southbound interface
4. Monitors the flow for performance and compliance with policies

The controller maintains an inventory of active flows in the network, tracking their paths, bandwidth consumption, and other statistics. This information is used for:

- Link utilization calculations
- Flow rerouting when network conditions change
- Enforcement of quality of service policies
- Visualization and monitoring of network state

6. Policy Implementation

The controller implements several routing and traffic management policies:

1. Load Balancing: When multiple paths of similar quality exist between source and destination,traffic is distributed across these paths to maximize network utilization and minimize congestion.

2. Traffic Prioritization: Flows can be assigned different priority levels based on their source,destination, or traffic type. Higher priority flows are given precedence in routing decisions and resource allocation.

3. Backup Path Provisioning: For critical flows, the controller maintains backup paths that can bequickly activated in case of failure along the primary path.

4. Link Failure Handling: When a link failure is detected, the controller:
- Identifies affected flows
- Computes alternative paths for these flows
- Updates flow tables to reroute traffic
- Updates the topology representation to reflect the new network state

7. Visualization Components

The visualization component provides a graphical representation of the network state, helping operators monitor and understand network behavior. Key features include:

- Network topology visualization showing switches and links
- Link coloring based on utilization levels (green for low, yellow for medium, red for high)
- Link thickness representing bandwidth capacity
- Flow path highlighting to show active traffic flows
- Statistics display for link and switch performance metrics

The visualization uses matplotlib for rendering and can export network views as image files for reporting or documentation purposes. For environments where graphical display is not available, a text-based representation is provided as a fallback.

8. Command Line Interface

The Command Line Interface (CLI) provides operators with a text-based interface to interact with the controller. The CLI supports the following operations:

- Network Topology Management:
- Adding and removing switches
- Creating and deleting links between switches
- Viewing the current network topology

- Flow Management:
- Adding new flows with specified source, destination, bandwidth, and priority
- Listing active flows in the network
- Removing flows

- Network Monitoring:
- Viewing link statistics (bandwidth, utilization)
- Computing paths between specified endpoints
- Displaying the network topology

- Testing and Simulation:
- Simulating link failures to test network resilience
- Injecting test traffic flows

The CLI provides help documentation for each command and validates input parameters to prevent errors.

9. Performance Considerations

The controller is designed with performance in mind, particularly for key operations that may occur frequently:

- Path Computation: Uses efficient algorithms to minimize computation time
- Topology Updates: Optimized data structures for quick addition and removal of networkelements
- Flow Table Management: Efficient lookup and modification of flow entries

For large networks, additional optimizations are implemented:

- Caching of computed paths for frequently used routes
- Batching of flow table updates to reduce overhead
- Incremental topology updates to avoid full recomputation

The controller is also designed to be horizontally scalable, allowing for distribution of load across multiple instances for very large network deployments.