5

Spatio-Temporal Hypergraph Models and Fog Computing for Intelligent Resource Management

Previous chapters of this thesis focused on enhancing specific aspects of network performance. Chapter 3 introduced directional routing protocols (i.e., CSBR, OBQR, and SDCast), which exploit vehicle orientation and trajectory information to improve packet forwarding decisions in dense urban environments. Subsequent work (Chapter 4) applied meta-heuristic techniques (e.g. clustering optimization with Grey Wolf Optimizer and hybrid metaheuristics) to improve network stability and route longevity under high mobility. These contributions laid important groundwork, yet they also highlighted a limitation: existing graph-based models and centralized architectures cannot fully capture or adapt to the multi-faceted dynamics of VANETs [133]. Specifically, conventional graph representations of the network do not easily accommodate complex interactions (such as a broadcast message involving multiple vehicles simultaneously), and cloud-centric processing introduces communication delays that are untenable for safety-critical applications [134].

In light of these challenges, this chapter proposes two frameworks aimed at enhancing the flexibility (software defined control), understanding and interpreting the network state (i.e., network interpretability), and responsiveness of VANET management (how rapidly and effectively the network adapts to changes in topology, traffic load, or node mobility). Specifically, network interpretability herein refers to making the decision-making processes (e.g., route selection, link evaluation) transparent enough that traffic engineers can understand which network elements or paths are deemed critical by the algorithm and why. This kind of interpretability is important for trust in AI-driven networks. First, we propose a Spatio-Temporal Information-Aware Hypergraph Approach that generalizes the traditional network graph model to a hypergraph structure. In a hypergraph, an edge (now called a hyperedge) can connect any number of vertices, which in our context means a communication event can directly involve multiple vehicles. This representation is paired with a deep learning-driven routing strategy that uses spatio-temporal traffic data to make optimized decisions. By capturing higher-order relationships (beyond simple pairwise links) and feeding them into a Deep Reinforcement Learning (DRL) algorithm, the network can better anticipate and adapt to changes.

Second, we introduce a Software-Defined Fog Computing (SDFC) framework for VANETs, which pushes computational intelligence and control closer to the network edge (the vehicles and roadside units). The limitations of purely centralized (cloud-based) network management become evident in scenarios with extremely low latency requirements or intermittent connectivity to the cloud. Fog computing extends cloud capabilities by deploying computing resources at the edge of the network (e.g., in roadside units or even vehicles), thus enabling data processing and decision-making to occur in proximity to where data is generated.

By doing so, fog computing can drastically reduce end-to-end communication delays and offload traffic from the core network. In our SDFC framework, VANET management functions (such as cluster formation, routing control, and load balancing) are distributed across a hierarchy of cloud, fog, and edge layers. This design improves scalability and reliability by avoiding single points of failure and by adapting to local conditions. Essentially, the fog framework asks: Can a decentralized, edge-driven architecture maintain or improve network performance as VANET scale and heterogeneity grow? Through careful design of clustering algorithms and load balancing at the fog layer, we demonstrate that it can – yielding lower latency, balanced network load, and faster response to topology changes compared to a cloud-only approach.

The remainder of this chapter is organized as follows. Section 5.1 introduces the spatio-temporal hypergraph routing framework, including its motivation, design, and performance results. Section 5.2 details the fog computing framework for VANETs, describing its architecture, algorithms for clustering and load balancing, and the resulting improvements.

5.1 SPATIO-TEMPORAL INFORMATION AWARE HYPERGRAPH APPROACH IN VANETS

Traditional routing algorithms in VANETs rely on graph models where vertices represent vehicles and edges represent direct wireless links between vehicles. While graph-based models are intuitive and form the basis of many network protocols, they are often inadequate for VANETs' highly dynamic and multi-objective environment. In a conventional graph $G = (V, E)$, an edge connects exactly two nodes, and routing algorithms compute paths as sequences of such edges. This simplistic representation struggles to capture scenarios where a single broadcast message may be received by many vehicles simultaneously, or where the decision to forward a packet depends on a combination of factors (link duration, data age, signal quality) that transcend a single link's properties. Moreover, graph-based routing often requires frequent recomputation in VANETs: as soon as the topology changes (which could be every few seconds), shortest paths or cluster memberships must be recalculated from scratch, incurring high overhead and possibly reacting too slowly to be effective in real time. This raises a critical question:

"Could we go beyond the constraints of conventional graph-based routing approaches to accurately map the complex, ever-changing dynamics of VANETs?"

Traditional graph-based routing models, while effective in static or less dynamic environments, often fall short in the rapidly changing landscape of VANETs. They struggle to capture the temporal evolution of communication links and intricate spatial interdependencies among vehicles, and infrastructure [95]. For example, traditional graphs within a single edge cannot natively represent the multi-objective optimization criteria such as latency, bandwidth, and packet loss [119]. This inadequacy is even more pronounced in multicast or geocast scenarios, where understanding complex network-wide relationships is crucial. This inadequacy often results in frequent graph re-computations, leading to increased computational overhead and potentially suboptimal routing decisions, impacting real-time vehicular communication and safety.

"In the context of SDVNs, finding the optimal path is a continuous problem (due to underlying distributions, changing dynamics, and multi-objective optimizations), not easily solved by traditional methods.

Hypergraphs enhance traditional graphs by introducing hyperedges that can connect multiple vertices, effortlessly capturing complex relationships. Coupled with the DRL architecture designed for continuous action spaces, a framework can offer a transformative solution. The actor-critic architecture of DRL allows for deterministic policies that specify actions for each state, evaluated through the Q-value function by the critic. This synergy encapsulates spatio-temporal intricacies and multiple QoS parameters, paving the way for more accurate and computationally efficient routing algorithms.

5.1.1 Rationale Behind Architectural Choices

This section outlines the foundational principles and architectural choices that support our proposed solution to address the challenges in SDVNs.

Rationale for the Selection of Core Methodologies

Our solution strategically combines advanced DL methodologies with hypergraph theory to address the complex dynamics of VANETs. The architecture, depicted in Figure 5.1, begins with raw vehicular network data processed through SDN and hypergraph structures, chosen for their superior ability to model multi-way interactions among vehicles and infrastructure. This refined representation is then analyzed by CNN to extract critical spatial features and GRU to capture essential temporal dependencies. The integration of attention mechanisms further enhances the model's ability to prioritize significant information, culminating in the DRL component, which specifically employs the DDPG algorithm. The DDPG's actor-critic architecture continuously optimizes routing decisions, adapting to the dynamic nature of SDVNs in real time. The interplay among these components—where hypergraph-informed data structures guide the CNN-GRU model, which in turn informs the DRL decision-making process—establishes a feedback loop essential for maintaining the system's adaptability, precision, and efficiency in achieving optimal routing decisions.

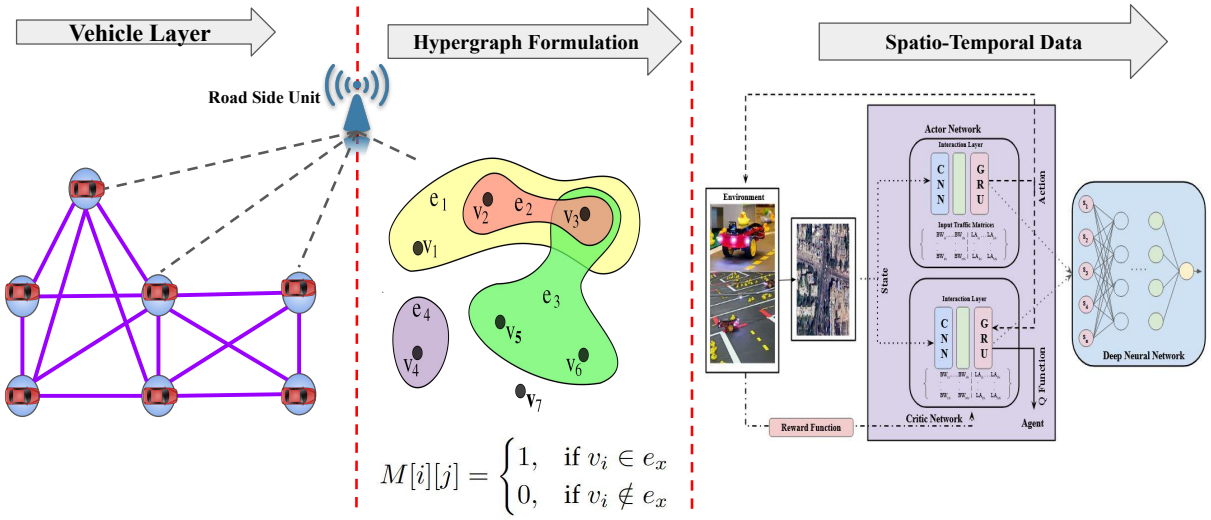


Figure 5.1: Proposed Architecture Model

Identified Technical Challenges

1. Hypergraph Construction and Scalability: Constructing hypergraphs in large-scale vehicular networks poses significant computational and memory challenges due to their high-dimensional, dynamic nature. As the number of nodes V (vehicles) increases, the hyperedge set $\mathscr{E}$ grows exponentially to $O(2^{|V|})$, with multi-way interactions further contributing to high-order hyperedges that require substantial computational resources.

2. Complexity in Hyperedge Weight Calculation: Assigning weights to hyperedges is a multi-objective optimization problem involving factors such as latency, availability, and the number of hops. These factors are critical in SDVNs, where real-time decisions are necessary to maintain optimal network performance. The challenge lies in aggregating high-dimensional data across all involved nodes and calculating these weights in real time.

3. Computational Overhead in Routing Algorithms: Hypergraphs in routing introduces additional computational overhead compared to traditional graph-based methods. Routing algorithms must consider multiple hyperpaths, expanding the solution space and increasing optimization complexity.

4. Integration with Deep Learning Models: Integrating hypergraphs with CNNs, GRUs, and DDPG presents significant challenges, particularly in data preprocessing and ensuring model compatibility. The hypergraph data must be transformed into a format suitable for CNNs and GRUs, maintaining the spatial-temporal correlations necessary for accurate routing decisions.

5. Real-Time Adaptation: SDVNs are highly dynamic environments, requiring continuous adaptation of the hypergraph structure and DL models to reflect real-time changes in the network. Achieving this without introducing significant delays or compromising the accuracy of routing decisions is a major technical challenge.

5.1.2 Proposed Strategic Solutions and Contributions

1. Optimized Hypergraph Construction and Scalability: We implemented a multi-tiered strategy involving data structure refinement and parallel processing. We utilized a binary incidence matrix to compactly encode vertex-hyperedge relationships, reducing operational complexity. Using OpenMP for thread-level parallelism, the hypergraph was partitioned into independent sub-segments based on spatial and temporal locality, enabling concurrent processing. This ensures our hypergraph model remains responsive and efficient in large-scale deployments.

2. Efficient Hyperedge Weight Calculation: We decomposed hyperedge weight optimization problem into parallelizable sub-problems, allowing each factor (latency, availability, hops) to be computed independently. For instance, latency is estimated using predictive models that account for both current and anticipated network conditions, while availability is derived from a combination of historical data and real-time feedback. These metrics are then dynamically combined into a composite weight using a weighted sum, with priorities adjusted based on current network conditions.

3. Advanced Routing Mechanism: We developed a hybrid approach that integrates heuristic methods with DRL. Initially, a heuristic-based preprocessing step filters out unlikely paths based on the current network conditions, thus reducing the solution space. The remaining paths are then evaluated using a DDPG-based

model, which iteratively refines its routing policy through interaction with the network environment and feedback rewards. This method allows us to efficiently navigate the expanded solution space, ensuring that routing decisions are both timely and accurate.

4. Integration with Deep Learning Models: We designed a custom data pipeline to transform hypergraph data into a format compatible with CNNs and GRUs. This pipeline includes preprocessing steps that normalize and encode the hypergraph's incidence matrix as a multi-channel input, enabling the CNN to effectively capture spatial relationships. The CNN outputs are then reshaped for input into the GRU, which processes the temporal aspects of the network data. Custom layers were developed to maintain the integrity of spatial-temporal correlations during this transformation, ensuring that the DDPG model can make informed and accurate routing decisions.

5. Real-Time Feedback Loops: We implemented a real-time feedback mechanism that continuously monitors network conditions and updates both the hypergraph and deep learning models accordingly. The SDN controller plays a central role in this process, collecting data from network nodes to update the hypergraph structure and adjust model parameters based on the current state of the network. For example, a significant change in vehicle density triggers recalibration of hyperedge weights and re-optimization of routing paths. This real-time feedback loop ensures that our model remains responsive and accurate in the face of rapidly changing network conditions.

5.1.3 Communication Model and Component Overview

We model the network as a hypergraph $\mathscr{H} = (\mathscr{V}, \mathscr{E})$, where $\mathscr{V}$ represents the set of vehicles, and $\mathscr{E}$ denotes the hyperedges. Unlike standard graph edges that link just two nodes, a hyperedge $e \in \mathscr{E}$ can connect multiple vertices, enabling complex node interactions.

Communication Infrastructure: Our setup includes an SDN server that interfaces with RSU. The SDN server, connected to RSUs via an IP network, governs routing and policy through its control plane. Vehicles use C-V2X for communication, operating in the sub-6 GHz spectrum with bandwidth B_{v2x} for both V2V and V2I interactions. Although C-V2X can facilitate direct communication between vehicles, its range is limited. RSUs, as stationary infrastructure with potentially higher power and better antennas, provide extended coverage. The proximity-based communication *(PC5)* interface enables direct communication between devices without routing data through a base station. This interface is crucial for latency-sensitive applications, bypassing the traditional cellular network and ensuring rapid data exchange essential for safety-critical vehicular applications. The user equipment to user equipment *(Uu)* interface also facilitates communication with cellular networks. This model highlights a bidirectional communication paradigm: OBU disseminate their status and await directives, while RSUs process OBU data and respond appropriately.

Fault Tolerance: We employ a virtualized master-slave controller architecture to enhance network reliability. The master controller, identified by the highest IP address, synchronizes network views and data flows, sending heartbeat signals to the slave controllers. A backup controller automatically takes over in case of a master failure, triggered by a timeout mechanism. This setup ensures robustness, which is particularly vital given the multiple routing paths in vehicular networks.

Constraints and Assumptions: The network follows specific constraints to ensure optimal performance and reliability. Each hyperedge e has a non-negative weight w_e representing the packet transmission cost.

The routing strategy aims to balance traffic distribution, preventing node saturation and sustaining network functionality. It also optimizes hyperedge utilization to minimize congestion and maximize throughput. For analytical clarity, we assume uniform packet sizes, standardizing transmission times across hyperedges.

Software-Defined Networking: SDN is pivotal in orchestrating dynamic network configurations. The SDN controller, with its northbound API, interfaces directly with our DDPG-based decision module. As the hypergraph's incidence matrix feeds into the CNN, the processed spatial-temporal features are relayed to the SDN controller. Bi-directional communication between the SDN controller and network elements facilitates real-time feedback, enabling the DDPG module to continually refine its policies. Communication is managed via the Programming Protocol-independent Packet Processors (P4) protocol, allowing the central controller to adapt to dynamic vehicular network requirements. P4's programmability ensures real-time, tailored decision-making, surpassing older protocols like OpenFlow.

Incidence Matrix: Vehicle connectivity is mapped through an incidence matrix **M[i][j]**, viewed as a binary feature map where rows represent vertices and columns represent hyperedges. Each cell indicates the presence (1) or absence (0) of a vertex in a hyperedge. The incidence matrix is defined as:

$$\mathbf{M[i][j]} = \begin{cases} 1, & \text{if } v_i \in e_x \\ 0, & \text{if } v_i \notin e_x \end{cases} \tag{5.1}$$

This matrix is input into a CNN as a grayscale image, representing local vertex-hyperedge relationships.

CNN for Spatial Understanding: The convolutional layers learn and adapt to various features in the incidence matrix through their filters. CNNs capture local correlations within a window defined by the kernel size, detecting and utilizing any recurring pattern or correlation within that window. For example, a matrix **M[i][j]** of size $n \times m$ produces a feature map of size $(n - ft + 1) \times (m - ft + 1)$ after passing through a convolutional layer with a filter of size $ft \times ft$.

Temporal Dynamics with GRU: The GRU uses gating mechanisms (reset and update gates) to regulate information flow, retaining relevant past information while discarding the irrelevant. This capacity to remember and leverage long-term correlations is crucial for understanding sequences where data point correlations extend beyond immediate spatial or temporal proximity. The GRU models a traffic demand matrix as a time sequence, tracking the network's temporal evolution as vehicles move and relationships change. To input the traffic demand matrix into the GRU, each row (or column) of **Td** is treated as a timestep. As the hypergraph evolves, it produces a series of traffic demand matrices $\{\mathbf{Td}_1, \mathbf{Td}_2, \ldots, \mathbf{Td}_t\}$, where the *i*th row of each matrix corresponds to the *i*th timestep in the GRU sequence. Given a sequence of traffic demand matrices $\mathbf{Sq} = [\mathbf{Td}_1, \mathbf{Td}_2, \ldots, \mathbf{Td}_t]$, the GRU's input at timestep t is $\mathbf{Td}_t$. The hidden state h_t of the GRU is defined as:

$$h_t = g(\mathbf{Td}t, ht - 1), \tag{5.2}$$

where g is the GRU's recurrent function.

Attention Mechanism for Contextual Focus: The Attention mechanism refines the CNN and GRU features, dynamically prioritizing the relevant input based on the current context.

DRL for Optimal Decision Making: Utilizing the DDPG algorithm, our framework trains the integrated GRU+CNN+Attention model to make optimal real-time decisions for the vehicular network.

5.1.4 Problem Definition

The traditional graph model symbolized as $\mathscr{G} = (V, E)$, has been the foundational structure for packet routing algorithms. However, this model often needs to encapsulate the intricate inter-dependencies prevalent in contemporary network topologies, especially in vehicular networks where multiple vehicles can be involved in a single communication event. To bridge this gap, we propose the hypergraph model $\mathscr{H} = (\mathscr{V}, \mathscr{E})$. In this model, $\mathscr{V}$ signifies the set of vertices representing network nodes (vehicles), and $\mathscr{E}$ denotes the set of hyperedges (communication links). Each hyperedge $e \in \mathscr{E}$ is a non-empty subset of $\mathscr{V}$ with a cardinality $|e| \geq 2$, indicating that a communication event can involve multiple vehicles. For a given hypergraph $\mathscr{H}$, we define a weight function $w : \mathscr{E} \to \mathbb{R}^+$ that assigns a non-negative real number to each hyperedge. This weight represents: Hops/cost (number of intermediate nodes a packet traverses), Availability (duration for which a communication link is active), and Latency (delay experienced by a packet on a link). Formally, for each edge e, the weight is given by:

$$w(e) = \alpha \times \text{hops}(e) + \beta \times \text{availability}(e) + \gamma \times \text{latency}(e)$$

Where α, β, γ are constants that determine the relative importance of each factor. Our objective is to determine a routing function $f : \mathscr{V} \times \mathscr{V} \to \mathscr{P}$, where $\mathscr{P}$ represents the set of all feasible paths in $\mathscr{H}$. The optimization problem seeks to minimize the total weight of the paths chosen for all source-destination pairs and can be articulated as follows:

$$\begin{aligned} \underset{f}{\text{minimize}} \quad & \sum_{(i,j) \in \mathscr{V} \times \mathscr{V}} \left(\sum_{e \in f(i,j)} w(e) \right) \\ \text{subject to} \quad & f(i,j) \in \mathscr{P}(i,j),\ \forall (i,j) \in \mathscr{V} \times \mathscr{V} \\ & w(e) \geq 0,\ \forall e \in \mathscr{E} \end{aligned} \tag{5.3}$$

The intricacy of this problem arises from the multi-dimensionality of hyperedges, the volatile nature of $\mathscr{H}$ due to the dynamic movement of vehicles, and the multi-objective optimization, which encompasses both latency reduction and route optimization. The pivotal challenge is to devise a routing algorithm adept at efficiently resolving this optimization problem while accommodating the distinctive traits of hypergraph-centric networks.

5.1.5 Proposed Methodology

The methodology employs a multi-layered approach, beginning with the conversion of network topology and traffic requirements into a hypergraph via an SDN controller. This hypergraph undergoes DDPG analysis, guided by a CNN-GRU architecture for feature extraction. These features, capturing both spatial and temporal traffic patterns, are fed into an actor-critic network that produces optimal configurations, implemented in real-time by the SDN controller. Iterative refinement based on network feedback yields a policy for optimal resource allocation and traffic demand satisfaction. The following subsections detail the critical components of the framework, highlighting each element's technical intricacies and contributions.

Network Initialization

In the initialization phase, OBUs and RSUs use a broadcast protocol, i.e., Service Discovery Protocol (SDP), to connect with the SDN controller. Vehicles broadcast messages containing their location and capabilities.

Following discovery, OBUs exchange location and connectivity data to form a spatio-temporal network graph, which is then transformed into an incidence matrix capturing the network's hypergraph structure. This matrix is relayed to the SDN controller for management and routing. The controller prepares a traffic demand matrix Td for routing, comprising spatio-temporal data, bandwidth, and inter-vehicle distance.

$$\mathbf{Td} = [\mathbf{ST}, \mathbf{BW}, \mathbf{DS}], \tag{5.4}$$

where **ST** is a matrix detailing each vehicle's location and velocity, represented as $\mathbf{ST} = [st_1, st_2, ..., st_n]$, with $st_i = [x_i, y_i, vl_i]$ for the i-th vehicle. BW indicates communication capabilities as $\mathbf{BW} = [bw_1, bw_2, .., bw_n]$, with b_i as the i-th vehicle's bandwidth. DS captures inter-vehicle distances as $\mathbf{DS} = [ds_1, ds_2, .., ds_n]$, where $ds_i = [ds_i, j]$ represents the distance between vehicles i and j. The **Td** matrix serves as the DDPG algorithm's input for real-time routing decisions.

Hypergraph Formulation

The hypergraph formulation is pivotal in enhancing SDVN performance by reducing the computational load on the DRL algorithm. The minimal path set between vertices v_i and v_j in a hypergraph $\mathscr{H}$ is given as:

$$\mathscr{P}(v_i, v_j) = \min(w_k, w_s), v_i \xrightarrow{w_k} \ldots \xrightarrow{w_s} v_j, (k, s \in 1, .., z). \tag{5.5}$$

For example, in Figure 5.2, vertices are $\mathscr{V} = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$ and hyperedges are $\mathscr{E} = \{e_1, e_2, e_3, e_4\} = \{\{v_1, v_2, v_3\}, \{v_1, v_4\}, \{v_4\}, \{v_5, v_6, v_7\}\}$. Based on Figure 5.2, v_1 has direct paths to v_2, v_3, and v_4. For v_2,

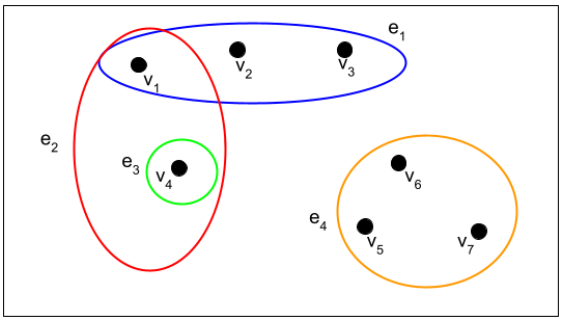


Figure 5.2: Hypergraph example

a path to v_4 exists via v_1, utilizing hypergraph properties. For routing optimization, hypergraph construct a tree traversal, associated nodes, and a corresponding array, as shown in Figure 5.3 and Table 5.1.

Table 5.1: Associative vehicle set and associative hyperedge set

v_i	v_1	v_2	v_3	v_4
$AS(v_i)$	$\{v_1, v_2, v_3\}$	$\{v_1, v_2, v_3\}$	$\{v_1, v_2, v_3\}$	ϕ
$AE(v_i)$	$\{e_1, e_2\}$	$\{e_1\}$	$\{e_1\}$	$\{e_2, e_3\}$

In our model, routes between vehicles v_i and v_j are sequences of hyperedges $\{e_1, e_b, \ldots, e_x\}$, with the shortest path determined by the hyperedge's minimum weight, factoring in cost, availability, and delay. If no hyperedge sequence exists, no shortest path is found. The SDN controller transforms network topology into a hypergraph, clustering nodes into hyperedges. Figure 5.4 shows a hypergraph for a given topol-

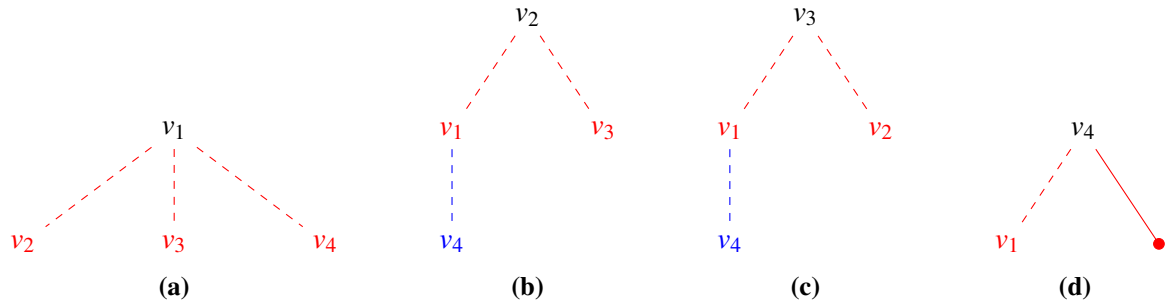


Figure 5.3: Hypergraph tree formation

ogy. The routing paths between RSU R_0 and vehicle V_4 and between V_5 and R_0 are $\{1-3-7-8\}$ and $\{9-6-5\}$, respectively. These paths are represented by hyperedges e_1 and e_2. Interpretability is achieved

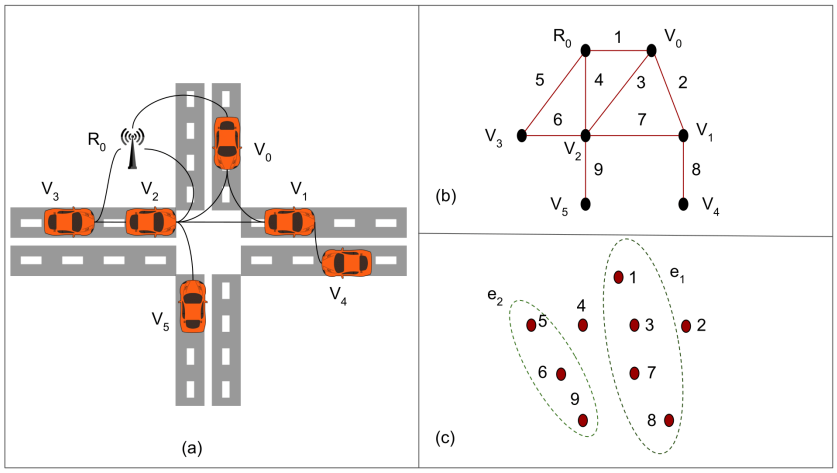


Figure 5.4: Hypergraph representation of the SDN routing model

via the critical connection search method [135], evaluating link criticality based on mask value, optimized for performance degradation, interpretation conciseness, and determinism. The performance degradation parameter assigns a higher weight to the link that has more influence on the routing decision. The interpretation conciseness parameter is related to the total number of critical connections in the topology, while the determinism parameter ensures that the value of the most critical connections is close to one and the least critical connections are close to zero. The constructed hypergraph feeds into the DDPG algorithm, which employs a CNN-GRU-based actor-critic network to generate optimal resource allocation policies. These policies are then implemented by the SDN controller and iteratively refined based on network feedback to maximize performance while meeting traffic demands.

Extracting Spatial and Temporal Patterns

A CNN-GRU network processes the hypergraph-encoded network traffic data. The CNN, utilizing the incidence matrix, extracts spatial patterns across the hypergraph, capturing the structural relationships inherent in the network topology. Meanwhile, the GRU processes the traffic matrix Td to capture temporal dependen-

cies, such as changing vehicle locations, bandwidth fluctuations, and evolving inter-vehicle distances. This dual-layer network generates a feature vector encoding both spatial and temporal traffic patterns, which feeds into the DDPG's actor-critic network, enabling the identification of optimal SDN controller strategies by discerning complex traffic patterns. The proposed algorithm constructs a binary incidence matrix mapping vehicle-hyperedge relationships, initializes associative node sets, and builds a branching tree for path exploration. The composite weight metric integrates latency, availability, and hop count to prioritize paths dynamically. Further refinement is achieved via a CNN-GRU architecture, with the CNN extracting spatial patterns and the GRU modeling temporal dynamics. The attention mechanism enhances precision by emphasizing critical features, enabling minimal computational overhead, scalability, and adaptability to real-time vehicular network dynamics. This methodology ensures optimal routing paths under varying traffic conditions, as detailed in Algorithm 8. The architecture employs a parallel CNN+GRU+Attention

Algorithm 8 Hypergraph Routing Algorithm

Require: Set of vehicles $V = \{v_1, v_2, \ldots, v_n\}$, hyperedges $E = \{e_1, e_2, \ldots, e_m\}$, weighted links $W = \{w_1, w_2, \ldots, w_z\}$
Ensure: Shortest routes for source-destination pairs

Construct binary incidence matrix $M[i][j]$ for vehicle-hyperedge relationships
for each vehicle $v_i \in V$ **do**
    Compute $AS(v_i)$: nodes associated via hyperedges
    Store directly connected nodes in $Ar[v_i]$ if $AS(v_i) \neq \emptyset$
end for
Initialize a branching tree with source vehicle as the root
for each source-destination pair (s,d) **do**
    Set root as s, initialize reachable nodes and assign direct weights from M
    Expand tree using minimum weight $w(e) = \alpha \cdot \tau(e) + \beta \cdot A(e) + \gamma \cdot H(e)$
    while unvisited nodes remain **do**
        Update weights for reachable nodes dynamically
        Prune suboptimal branches and extend viable paths
    end while
end for
for each candidate path in the branching tree **do**
    Use CNN-GRU to refine spatial-temporal weights:
    $M[i][j] \rightarrow$ CNN for spatial patterns, $T_d \rightarrow$ GRU for temporal sequences
    Update composite weights with attention-enhanced predictions
end for
Select paths with the minimum weight for all source-destination pairs **return** Shortest paths

framework to learn spatial and temporal traffic patterns, as depicted in Figure 5.5. The CNN is designed with five layers: convolution, batch normalization, max-pooling, dropout, and attention. In the convolution layer, the operation is defined as

$$O(i,j,k) = \sum \mathbf{M}(i+m, j+n) \times w(m,n,k) + b(k), \tag{5.6}$$

where O is the output feature map, $\mathbf{M}$ is the input incidence matrix, w is the weight matrix, b is the bias, and i, j, k, m, n are indices. The batch normalization layer normalizes these output feature maps O to reduce

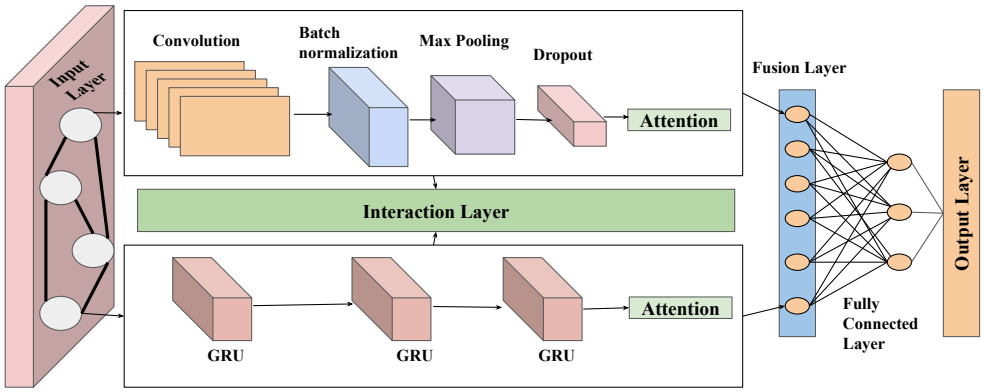


Figure 5.5: CNN+GRU+Attention architecture

internal co variate shifts. The normalization is performed using

$$NM = \gamma \times (\mathbf{M} - \mu)/\sigma + \beta, \tag{5.7}$$

where NM is the normalized output, γ and β are learnable parameters, and μ and σ are the mean and standard deviation, respectively. Max-pooling is applied to downsample the feature maps using

$$o(i,j,k) = \max(\mathbf{M}(i:i+m, j:j+n)), \tag{5.8}$$

where o is the downsampled output, and i, j, m, n are indices. The dropout layer helps prevent overfitting by randomly setting a fraction of the input units to 0 at each update during training time, which is represented as

$$O(i,j,k) = \mathbf{M}(i,j,k) \times C, \tag{5.9}$$

where C is a random variable that is either 0 or 1. The attention layer focuses on relevant traffic information using

$$AT(\mathbf{Qe}, \mathbf{Ky}, Vl) = \text{softmax}(Qe \times Ky^T/\sqrt{d}) \times \mathbf{VM}, \tag{5.10}$$

where $\mathbf{Qe}, \mathbf{Ky}, \mathbf{VM}$ are the query, key, and value matrices, respectively, and d is the key matrix dimension. In our architecture, an interaction layer merges the CNN and GRU outputs, which a fusion layer consolidates to form the final output. This robust design adeptly captures complex traffic patterns, enabling the SDN controller to optimize its operations. The GRU model, illustrated in Figure 5.6, learns temporal patterns in

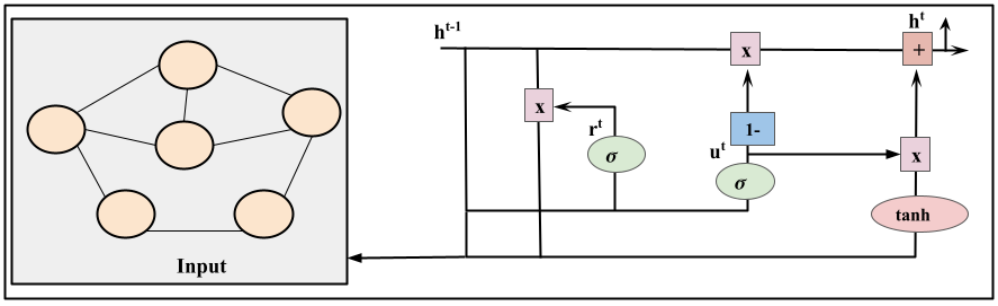


Figure 5.6: Finding temporal traffic pattern using GRU with attention

SDVN traffic using traffic matrix **Td**. It uses update and reset gates to manage information flow between previous and current hidden states, which store temporal traffic data. The CNN's spatial features and time-step data are fed in parallel into the GRU as $\mathbf{Td}(t) = [\vec{z}(t), t]$, where $\vec{z}(t)$ is the feature vector produced by the CNN at time t. The update gate controls the information flow from the previous hidden state $h(t-1)$ to the current hidden state $h(t)$. It is calculated using a sigmoid function as follows:

$$U(t) = \sigma(w_u \cdot \mathbf{Td}(t) + w'_u \cdot h(t-1) + b_u), \tag{5.11}$$

where w_u, w'_u, and b_u are the weight matrices and bias, respectively. Similarly, the reset gate is expressed as:

$$RS(t) = \sigma(w_{rs} \cdot \mathbf{Td}(t) + w'_{rs} \cdot h(t-1) + b_{rs}), \tag{5.12}$$

with w_{rs}, w'_{rs}, and b_{rs} as the corresponding weight matrices and bias. The candidate hidden state $h'(t)$ is then calculated as:

$$h'(t) = \tanh(w_h \cdot \mathbf{Td}(t) + w'_h \cdot (RS(t) \cdot h(t-1)) + b_h), \tag{5.13}$$

and the current hidden state $h(t)$ is updated by:

$$h(t) = (1 - U(t)) \cdot h(t-1) + U(t) \cdot h'(t), \tag{5.14}$$

Finally, the GRU output $y(t)$ is determined by:

$$y(t) = w_o \cdot h(t) + b_o, \tag{5.15}$$

where w_o and b_o are the weight matrix and bias, respectively.

Actor Critic Network and DDPG

The architecture employs an actor-critic framework interfaced with the DDPG algorithm to optimize vehicular network routing. The actor-network $\mathscr{A}$, a composite of GRU, CNN, and attention mechanisms, ingests the traffic matrix $\mathbf{Td}_t$ (representing the state $s(t)$ of the VANET) and outputs an action a as $a = \mathscr{A}(\mathbf{Td}_t)$. The critic network $\mathscr{C}$ evaluates this action by computing a Q-value:

$$Q(\mathbf{Td}_t, a) = \mathscr{C}(\mathbf{Td}_t, \mathscr{A}(\mathbf{Td}_t)). \tag{5.16}$$

The state $s(t)$ is a complex representation of the VANET, encapsulated as a traffic matrix $\mathbf{Td}_t$ comprising spatio-temporal data, bandwidth, and distance between vehicles. The action $a(t)$ refers to the routing decisions, represented by a set of probabilities for selecting the next hop based on the current state. The rewards $R(t)$ (described in subsection 5.1.5) are scalar values that quantify the performance of these routing decisions, determined by a function of the state, action, and desired QoS metrics. At each time step, the DDPG algorithm takes $s(t)$ as input and outputs an action $a(t)$ that maximizes the expected cumulative reward. The action is determined by:

$$a(t) = \underset{a}{\operatorname{argmax}}\, Q(\mathbf{Td}_t, a). \tag{5.17}$$

The actor network's parameters $\theta_{\mathscr{A}}$ are updated using gradient descent, as (5.18):

$$\theta_{\mathscr{A}} = \theta_{\mathscr{A}} + \eta * \nabla_\theta Q(\mathbf{Td}_t, a(t)). \tag{5.18}$$

The deterministic policy $\Pi(s)$ is expressed as a function that maps the state to the action that maximizes the Q function. During training, the critic network computes the loss $L_{\mathscr{C}}$ as the mean squared error between the predicted and target Q-values. The actor-network is trained to maximize the expected Q-values, updating its parameters $\theta_{\mathscr{A}}$ according to:

$$\nabla_{\theta_{\mathscr{A}}} J \approx \mathbb{E}\left[\nabla_a Q(\mathbf{Td}_t, a) \nabla_{\theta_{\mathscr{A}}} \mathscr{A}(\mathbf{Td}_t)\right]. \tag{5.19}$$

Thus, the actor proposes actions, the critic evaluates them, and both are trained iteratively to refine the policy. This iterative refinement incorporates the DDPG model factors: state, action, and rewards for holistic optimization of the vehicular network's routing decisions.

Training Process

The DDPG algorithm is trained iteratively using experiences collected from the environment. Each experience is a tuple $(s(t), a(t), r(t), s(t+1))$, where $s(t)$ is the current state (represented by the traffic matrix $\mathbf{Td}_t$), $a(t)$ is the action taken, $r(t)$ is the reward received, and $s(t+1)$ is the next state. These experiences are stored in a replay buffer, a finite-sized cache. During training, a mini-batch of experiences is sampled from this buffer to compute the gradient of the Q-function and update the actor and critic networks.

Critic Update: The target Q-values are computed using the following Bellman equation:

$$Q_{\text{target}}(s(t), a(t)) = r(t) + \gamma \mathscr{C}(s(t+1), \mathscr{A}(s(t+1))), \tag{5.20}$$

where γ is the discount factor, which determines the present value of future rewards. The loss for the critic is the mean squared error between the predicted Q-values and the target Q-values:

$$L_{\mathscr{C}} = \mathbb{E}\left[\left(Q_{\text{target}}(s(t), a(t)) - \mathscr{C}(s(t), a(t))\right)^2\right]. \tag{5.21}$$

The critic's parameters are then updated using gradient descent to minimize this loss.

Actor Update: The actor-network is updated to maximize the expected Q-values. The gradient used for the update in (5.22) represents the policy gradient, guiding the actor to take actions that maximize the Q-values.

$$\nabla_{\theta_{\mathscr{A}}} J \approx \mathbb{E}\left[\nabla_a \mathscr{C}(s(t), a)|_{a=\mathscr{A}(s(t))} \nabla_{\theta_{\mathscr{A}}} \mathscr{A}(s(t))\right]. \tag{5.22}$$

Exploration: An ε-greedy strategy is used for exploration of the action space. With probability ε, a random action is chosen, and with probability $1-\varepsilon$, the action recommended by the actor-network is taken. The value of ε is initialized to 1 and decayed over time to promote initial exploration and exploitation later.

Soft Updates: To ensure stability, soft updates are employed for the target networks. Rather than directly copying the weights, the main and target networks' weights are mixed using a small mixing factor τ.

$$\theta_{\text{target}} = \tau \theta_{\text{main}} + (1-\tau)\theta_{\text{target}}. \tag{5.23}$$

Reward Setting

The approach refines the DDPG model using a reward function that maximizes specific performance metrics. This reward function directs the CNN+GRU+Attention model to predict spatio-temporal events accurately. A corresponding loss function, derived from the reward, trains the model to minimize performance gaps. The reward function encourages the DDPG algorithm to optimize link availability, inter-vehicle distance, delay, and packet loss within the SDVN. The algorithm takes action $a_t = \pi(\mathbf{Td}_t)$ based on the network state matrix $\mathbf{Td}_t$ and policy π, solving the routing optimization problem to establish a data path. The reward function is:

$$R = \alpha_1 * LA - \alpha_2 * DS - \alpha_3 * \Psi - \alpha_4 * \rho, \tag{5.24}$$

Where R is the reward, LA is link availability time, DS is average vehicle distance, Ψ is average packet delay, and ρ is packet loss ratio. The α values are weights, which are learned based on different QoS priorities. Link availability LA is calculated as:

$$LA = \frac{R_x - DS}{vel_i - vel_j}, \tag{5.25}$$

where R_x is communication range and vel_i is the $i-th$ vehicle's speed. Euclidean distance DS between two vehicles positioned at $\vec{v_i}$ and $\vec{v_j}$ in an N-dimensional space is:

$$DS(\vec{v_i}, \vec{v_j}) = \sqrt{\sum_{d=1}^{N} \left(\vec{v_{id}} - \vec{v_{jd}}\right)^2}, \tag{5.26}$$

Finally, packet loss rate ρ is defined as:

$$\rho = \frac{\sum Pkt_{loss}}{\sum Pkt_{tx}}, \tag{5.27}$$

where Pkt_{loss} represents lost packets and Pkt_{tx} represents the total number of transmitted packets.

5.1.6 Performance Analysis of Hypergraph Based Framework

Data is pivotal for any ML model, but VANET environment require datasets that integrate spatio-temporal information with packet flow data. To understand vehicle communications in urban VANETs, we employed several frameworks: OMNeT++ [53] (a modular, component-based C++ simulation library), VEINS (an open-source framework), SUMO [51] (for traffic simulation), and OpenStreetMaps [44] (for extracting real-world road network data) to simulate real-world conditions. We also created a testbed to validate the results empirically. Using IEEE 802.11p, we established wireless node connections at a 6 Mbit/s data rate. Specifically, we simulated a VANET topology in Jodhpur, India, to collect data on spatio-temporal routing parameters. Our analysis compares our methodology with following notable frameworks: RL-SDVN [96], DROM [93], IV2XQ [86], MFCAR [92], and M.Chahal et al. [91]. We acknowledge that while our approach does not align straightforwardly with the techniques outlined in the referenced studies, we have incorporated elements of their algorithms into our experimental setting to mirror their approaches.

Dataset and Reproducibility Information

Data from established datasets, such as NG-SIM [136] for vehicle location, speed, and acceleration, and VANET-MOBIDATA for detailed spatio-temporal metrics, was incorporated. Computations were performed on an Ubuntu 22.04 system with an Intel Core *i7* processor, 16 GB RAM, and an NVIDIA GeForce

GTX 50 GPU. Using OSMWebWizard, a Jodhpur road network with 200 vehicles crossing a junction within 15 minutes was extracted and exported in SUMO format. This network generated a synthetic traffic dataset, including both light and heavy vehicles with set routes and timings, formatted for DDPG algorithm training implemented using TensorFlow and Keras with CNN and GRU for SDVN. Simulation hyperparameters are detailed in Table 5.2. Five hundred unique source-to-destination patterns were generated, utilizing 20%-100% of network capacity across five traffic densities. DDPG rewards and convergence speed were optimized by fine-tuning the learning rate and adjusting actor and critic network parameters: goal smooth factor 0.001, learning rate 0.0001, discount factor 0.90, and exploration parameter 0.80.

Table 5.2: Hyperparameter and simulation settings for hypergraph framework

Parameter	Values	Parameter	Values
Number of Convolutional Layers	5	Kernel size	3×3
Stride (Convolutional Layer)	2	Non-linear activation function	ReLU
Regularization techniques	Batch normalization and Dropout (0.2)	Max-pooling filter size	2×2
Max-pooling stride	1	Number of GRU layers	3
GRU hidden layer size	256	Attention mechanism	Yes
Fusion layer	Concatenation	Fully connected layer size	128
Transmission range	300m OBU	Transmission rate	6Mbps

Testbed Information

In developing a robust and scalable solution, a real-world test bed was established to closely mimic the complexities and dynamics of urban vehicular networks. The test bed was set up based on the renowned Duckietown platform [54], offering a miniature urban environment for autonomous vehicles. As shown in Figure 5.7, the test bed is designed as a scaled-down city with roads, intersections, road signs, and other urban infrastructural elements. This intricate setup serves as the backbone for experiments, providing a realistic environment to test and validate the hypergraph-based VANET model. Each vehicle in the setup

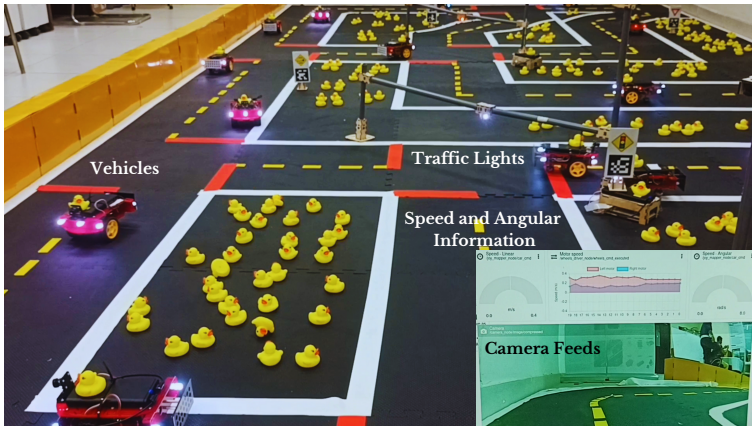


Figure 5.7: Vehicular network testbed

is equipped with either NVIDIA's Jetson Nano or Raspberry Pi 3B+, depending on computational requirements. The Jetson Nano, with its powerful GPU capabilities, handles onboard computations, particularly those related to the hypergraph-based routing algorithm. Its quad-core ARM Cortex-A57 MPCore processor and 128-core Maxwell GPU ensure real-time processing, essential for the dynamic nature of VANETs.

Conversely, the Raspberry Pi 3B+ serves as the communication hub, facilitating V2V and V2I communications with its integrated Wi-Fi and Bluetooth capabilities, essential for the VANET model's functionality. The model, primarily hosted on the Jetson Nano, takes inputs from various sensors and communication modules on the vehicle, processing data in real-time. It leverages the combined strength of GRU, CNN, and Attention mechanisms to make routing decisions, which are then communicated to other vehicles and infrastructure nodes. The dynamic network topology, with vehicles moving, stopping, and changing directions, closely mimics real-world urban traffic scenarios. The hypergraph representation captures this dynamism, with nodes representing vehicles and hyperedges representing the multi-faceted relationships between them. Vehicles operate within a speed range of 0.5 to 1.5 m/s, simulating varying urban traffic conditions. V2V and V2I communication ranges are set between 10 to 50 meters, and packet sizes vary from 50 to 500 bytes. Transmission power is adjusted between -30 dBm to 0 dBm to emulate different vehicular distances and potential obstructions. Experimental results from this test bed will be represented by *RBot* in the subsequent result analysis section.

Interpretability

This experiment focuses on link interpretability and critical links to make the hypergraph model able to identify and utilize alternative routes when certain critical links became unreliable. While hypergraphs and deep learning add complexity to the model, we also designed methods to interpret the learned routing decisions. Using a technique referred to as critical connection search [135], we can assign an importance score or mask value to each link or hyperedge, indicating how critical it is to the routing decision. Essentially, even though an AI agent is making decisions, we as network designers can extract which groups of vehicles and links are most influential in those decisions, giving us insight into network weak points or hubs.

A 16-node network topology, generated by the testbed and depicted in Figure 5.8a, was used to showcase the model's interpretability. This topology, initialized with spatiotemporal data, featured over 75 unique source-to-destination paths. Figure 5.8b presents an interpretability analysis of the resulting hypergraph,

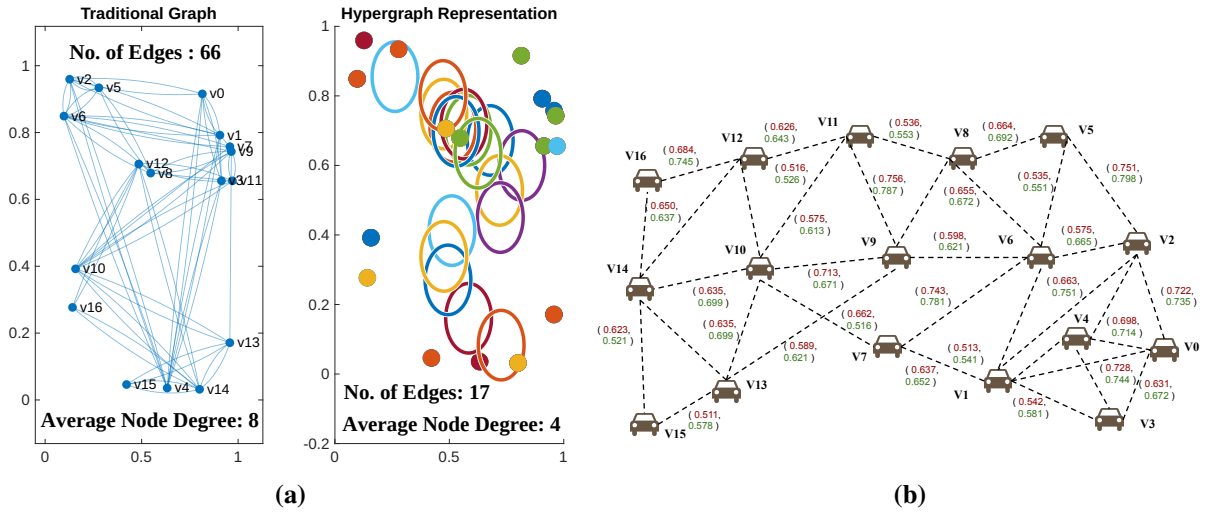


Figure 5.8: Interpretability analysis of hypergraph (a) Experimental 16-node network topology & (b) Mean link weight graph

including mean link weight scores that signify each link's importance in routing decisions. It also lists mask

values, which indicate link preferences in routing paths and are higher for shorter or less congested routes. These mask values are color-coded: red for links from lower- to higher-numbered nodes and green for the reverse. For example, the mask values for links $V_0 - V_1$ and $V_1 - V_0$ are 0.722 and 0.735, respectively. The most critical links have mask values close to one.

Computational Time Analysis

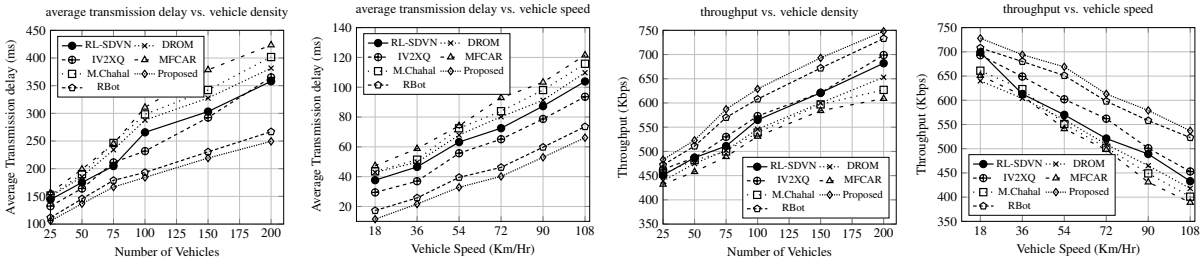
Hypergraph models offer computational advantages over sequential DRL models by parallelizing route calculations, significantly reducing computation time. The sparse vehicle relationships in hypergraphs further lower computational demands and require minimal exploration and trial-and-error to prepare the graph. Consequently, the hypergraph-based approach reduces the time needed to find the optimal routing path for 16 communication nodes by 21.87 milliseconds, a crucial advantage in time-sensitive, resource-constrained environment like VANETs. Table 5.3 underscores the method's effectiveness, detailing the SDN processing times for various models across 16 nodes and 75 routing paths.

Table 5.3: Computation time calculation

Model	Processing Time (ms)
Hypergraph + DDPG based model (Proposed)	21.87
RBot (Testbed)	29.34
Only CNN+GRU+Attention based model	32.09
Reinforcement Routing based model [86]	59.62
Recurrent Neural Networks (RNN)+Long-Short Term Memory (LSTM) based model [137]	68.26
CNN based model [138]	43.16
Deep Neural Networks (DNN) based model [139]	64.83

Network Performance

This subsection highlights the advantages of the proposed method over current state-of-the-art techniques. Figures 5.9a and 5.9b display the framework's average packet transmission delay under diverse traffic conditions. The predictive model intelligently routes packets by forecasting network conditions, thereby minimizing delays and avoiding congestion. The hypergraph-based DDPG model includes a CNN component that leverages vehicle geolocation for quicker decisions and streamlined data processing. The GRU component sequences data and uses historical network states to understand and adapt to network dynamics. Compared



(a) **(b)** **(c)** **(d)**

Figure 5.9: Network performance comparison of hypergraph based framework

to existing method, DROM suffers from slow convergence and fails to meet traffic demand constraints. DROM's effectiveness declines with network growth due to frequent disconnections and significant delays. Additionally, the MFCAR protocol's centralized algorithm is ill-suited for distributed environments,

causing congestion and performance degradation. Figures 5.9c and 5.9d showcase the method's average throughput. The hypergraph model optimizes routes based on network structure and predictive analytics, while dynamic load balancing distributes traffic across multiple channels to minimize congestion and boost throughput. The DDPG model adds adaptability, allowing the SDN controller to manage unforeseen events without compromising network stability. CNNs process traffic matrix data to refine routing protocols, while GRUs analyze sequential data for insights into traffic patterns and message delivery factors.

Network Delay in Different Traffic Load

This experiment span five traffic load scenarios, utilizing 20%, 40%, 60%, 80%, and 100% of total network bandwidth, each with unique traffic matrices. The model is iteratively trained 100 to 1000 times across these varying loads to evaluate performance. A baseline of 1000 training steps is used to validate the data. Figure

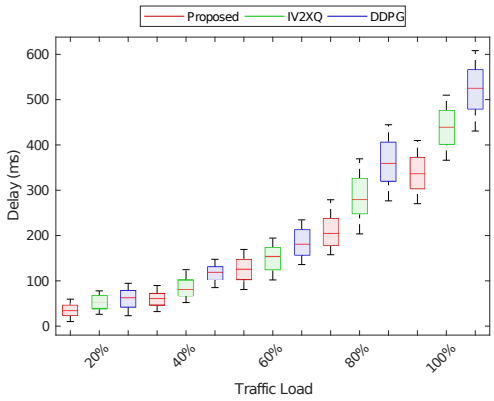
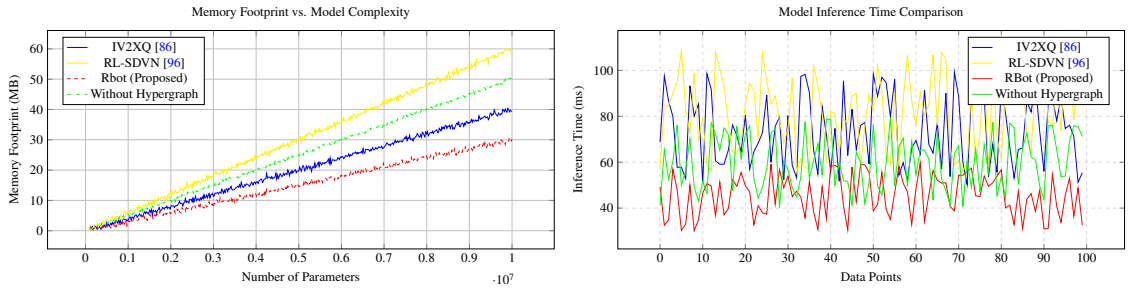


Figure 5.10: Transmission delay vs. Network traffic load in hypergraph based framework

5.10 reveals an increase in delay with higher traffic loads due to limited bandwidth. Increased traffic reduces available bandwidth, increasing transmission times and resulting in more delays at each node. Compared to IV2XQ and DDPG models, hypergraph model shows lower delays across all traffic loads.

Complexity Analysis based on Memory Footprints and Inference Time

The memory footprint of a Deep Learning (DL) model is a critical metric, particularly for deployment on small devices like the Jetson Nano or Raspberry Pi. It quantifies the memory required to store the model's parameters, intermediate activations, and other necessary data structures during inference. A smaller memory footprint allows for more efficient utilization of limited memory resources, enabling the deployment of complex models. In experiments, as shown in Figure 5.11a, the memory footprint of the proposed model was significantly lower than that of comparable models without hypergraphs. This reduction is attributed to a compact architecture that leverages depthwise separable convolutions to reduce the number of parameters without compromising performance, along with post-training quantization. The reduced memory footprint signifies the model's suitability for real-time applications. Additionally, inference time was calculated, as shown in Figure 5.11b, which measures the time a model takes to predict the output for a given input. A shorter inference time ensures prompt data processing and decision-making. The proposed model exhibited a faster inference time compared to other DL models without hypergraphs. This enhanced performance is due to the use of depthwise separable convolutions and other architectural innovations that reduce com-



(a) **(b)**

Figure 5.11: Model complexity and inference time analysis

putational complexity. The superior inference time, despite the model's compactness, further validates its effectiveness and suitability for real-time applications.

5.1.7 Framework Resilience

The results presented in the Figure 5.12 demonstrate the resilience of the hypergraph-based vehicular network in maintaining robust multihop V2V communication, even under dynamic conditions such as vehicle mobility and node failures. In the initial network configuration, the hypergraph effectively captures complex interactions between vehicles, enabling the establishment of strong communication paths. As vehicles move, represented by the network after five mobility steps, the hypergraph structure adapts seamlessly to the new topology, preserving connectivity and ensuring uninterrupted communication. This adaptability highlights the hypergraph's inherent ability to accommodate the dynamic nature of vehicular environments. When the network is subjected to failures, with five nodes randomly removed, the hypergraph's resilience becomes even more evident. Despite the loss of certain vehicles, the network retains a significant portion of its connectivity, ensuring that multihop communication paths remain intact. This robustness can be attributed to the hypergraph's ability to model multi-way relationships and its capacity to reroute communication through alternative paths when primary links fail. In contrast, the traditional graph-based network struggles to maintain the same level of connectivity under similar conditions, leading to more fragmented communication. This comparison underscores the hypergraph's suitability in providing a resilient, scalable solution for multihop V2V communication in dynamic vehicular networks, where constant movement and potential failures are the norm.

SDN Operational Latency, Network Changes Detection and Computational Efficiency

To assess the impact of centralized controllers on vehicle density and mobility, multiple experiments were conducted. Graph-based methods such as [137], [91], and [92] were chosen as baselines due to their potential to enhance vehicular network efficiency. SDN operational latency refers to the time taken for an SDN controller to process a network event and execute the necessary actions. For the hypergraph-based routing algorithm, the operational latency, denoted as L_{op}, is defined as $L_{\text{op}} = t_{\text{process}} + t_{\text{execute}}$, where t_{process} is the time taken to process the network event and t_{execute} is the time taken to execute the necessary actions based on the hypergraph's structure. Detecting changes in the network is crucial for the hypergraph-based routing algorithm to adapt to dynamic network conditions. A change in the network is detected by comparing $\mathscr{H}_t$ with $\mathscr{H}_{t-1}$. A change is detected if $\mathscr{H}_t \neq \mathscr{H}_{t-1}$. This comparison is further refined by examining the

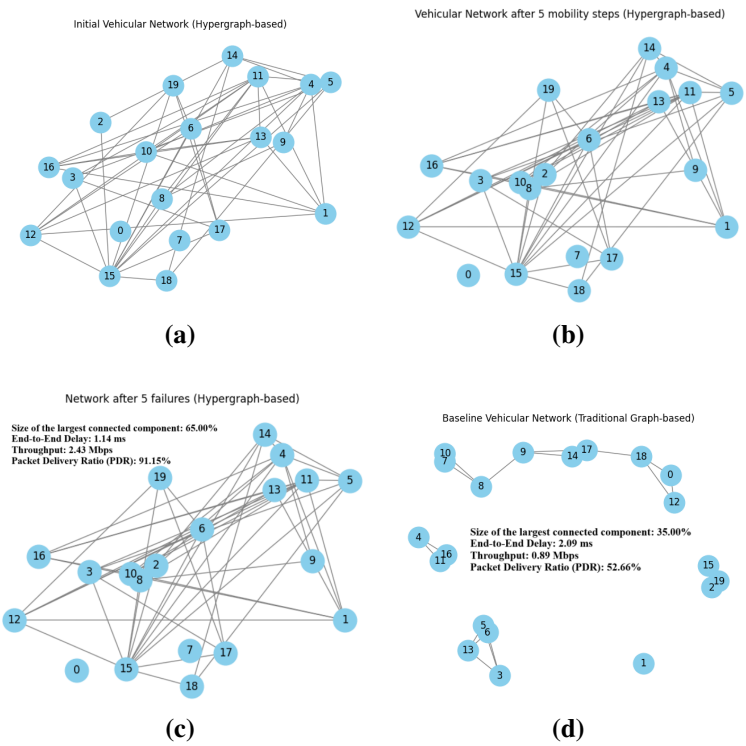


Figure 5.12: Resilience of Hypergraph vs Tradition Graph based Method (a) Initial Topology (b) Effect of Mobility (c) Effect of Node Failures (d) Traditional Graph After Mobility

adjacency matrices A_t and A_{t-1} of the hypergraphs at times t and $t-1$, respectively. The computational

Table 5.4: Operational latency for different traffic scenarios in centralized and distributed architecture

Architecture	**Vehicles**	**Discovery Time**	**Network Change**	Round Trip Time (ms)		
				Best	**Average**	**Worst**
Centralized	50	26.97 ms	3.43 ms	43.138	84.513	267.634
	100	37.42 ms	4.29 ms	39.321	100.241	534.542
	150	52.53 ms	6.98 ms	58.11	123.75	1047.453
	200	87.12 ms	11.18 ms	68.967	147.431	1135.19
Distributed	50	28.54 ms	2.17 ms	33.56	69.67	317.879
	100	44.91 ms	3.12 ms	38.56	104.21	610.50
	150	61.43 ms	7.88 ms	64.88	143.23	1196.38
	200	101.31 ms	17.28 ms	90.941	170.75	2024.47

efficiency of the proposed SDN architecture was also assessed by comparing request processing time to vehicle speed, as depicted in Figure 5.13. Table 5.4 shows that as network queries rise, so does processing time. In low-density settings, centralized architectures exhibit higher average latency than distributed ones. However, the latter's latency increases with vehicle count due to synchronization overhead. Controller overhead also scales with the volume of transiting query messages. Both architectures perform well in detecting topology changes. The computational efficiency of the hypergraph-based routing algorithm is evaluated by analyzing its time complexity. Given the algorithm's reliance on DDPG, CNN, GRU, and Attention mechanisms, the overall time complexity, denoted as $T(n)$, is expressed as a function of the number of vertices n in the hypergraph $T(n) = T_{\text{DDPG}}(n) + T_{\text{CNN}}(n) + T_{\text{gru}}(n) + T_{\text{Attention}}(n)$. Each term on the right-hand side represents the time complexity of the respective component when applied to the hypergraph.

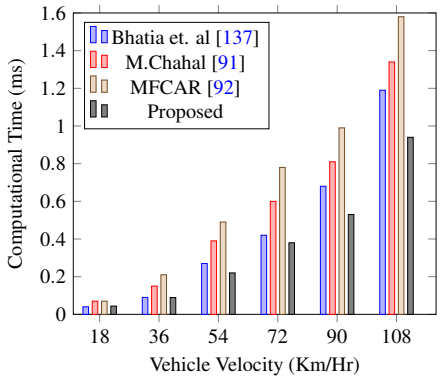


Figure 5.13: Computational efficiency of SDN

In resource-constrained vehicular networks, optimizing this efficiency is vital for effective communication. The architecture employs a hypergraph and DDPG for distributed decision-making, reduced overhead, real-time optimization, and enhanced scalability. DDPG alleviates the centralized controller's load by enabling decentralized decisions across the network. The hypergraph paradigm streamlines data representation, improving processing speed and reducing the costs associated with traditional routing. This DDPG integration allows for real-time routing optimization, lowering latency compared to standard algorithms.

5.2 FOG COMPUTING FOR INTELLIGENT RESOURCE MANAGEMENT

While the hypergraph DRL approach improves the algorithmic aspect of VANET routing, effective deployment of such an intelligent routing scheme in a real network also requires an agile underlying architecture. Traditional VANET control relies on either purely distributed decisions (each vehicle decides in isolation, which can be suboptimal) or centralized decisions through cloud servers. Purely distributed schemes often react slowly to network-wide changes and cannot easily coordinate things like load balancing. Fully centralized (cloud-based) solutions, on the other hand, suffer from latency and scalability issues: sending all data to a remote server introduces delays and a reliance on uninterrupted backhaul connectivity, which may not hold in all situations [134]. Fog computing emerges to fill this gap by extending computation and control into the edge of the network (e.g., roadside units, local micro-datacenters). The idea is to create a hierarchy: the cloud layer can perform heavy global analytics tasks or long-term optimizations, but the fog layer (closer to the vehicles) can handle time-critical, localized decision-making and intermediate data processing. Our proposed SDFC framework is guided by four key principles to enable real-time, scalable VANET management

1. **Adoption of Fog Computing for Edge Proximity:** The decentralization of data processing and storage to the network's edge via fog computing significantly reduces latency and optimizes bandwidth utilization. Proximity to data sources diminishes the frequency of vehicular handovers, thereby enhancing service continuity and network stability. Additionally, localized processing capabilities facilitate expedited response times and improve the overall efficiency of data management and application execution at the edge.

2. **Parallel Dynamic Clustering Algorithm:** The Parallel Dynamic Clustering (PDC) algorithm, inspired by the principles of Hadoop's MapReduce, enhances resource management in distributed computing environments. This algorithm promotes rapid convergence and precise control over resource allocation, ensuring increased scalability and efficient data aggregation. Furthermore, the PDC algorithm optimizes task distribution, resulting in enhanced performance and improved handling of large-scale data processing tasks across the network.

3. **Integration of Flow-Based Load Balancing:** he integration of a flow-based load balancer enables dynamic reallocation of incoming requests across fog nodes, based on current network load and topology. This ensures optimal load distribution, preventing any single node from becoming a bottleneck. By maintaining continuous service delivery and balancing the workload effectively, the system adapts to varying network conditions, thereby sustaining high performance levels.

4. **Intelligent Controller Placement Methodology:** The strategic placement of controllers within the network is critical for enhancing response times and minimizing communication overhead. By carefully selecting controller locations, this methodology ensures efficient management of interactions between fog nodes and cloud services. Such strategic placement is essential for maintaining low latency and high reliability in the network's control operations, thereby contributing to improved overall system performance.

The proposed SDFC framework (Ref. Figure 5.14) is systematically organized into three hierarchical layers: the cloud layer for centralized computation and storage, the fog layer for distributed computation and immediate, low-latency services, and the edge layer encompassing vehicular and RSU for data generation and initial processing. This chapter demonstrates how leveraging fog computing and integrating edge

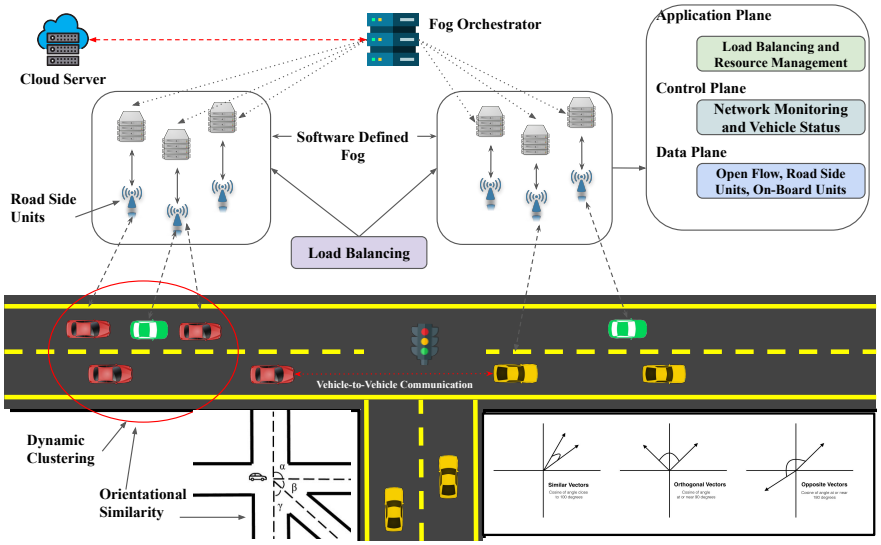


Figure 5.14: Software-defined fog architecture

computing principles can substantially improve network responsiveness and scalability. The SDFC framework addresses gaps in traditional cloud-based models, offering a robust solution adaptable to the dynamic and high-mobility environment of vehicular networks. Empirical results and simulation insights validate the proposed framework's efficacy in achieving equitable load distribution, minimizing network latency, and enhancing overall network performance.

5.2.1 Motivation and Guiding Principle

Recognizing the insufficiency of cloud-based models [140, 141], this research develops a dynamic framework adaptable to the variable scale and high mobility inherent in VANETs [142]. The focus is on advanced service level objectives (SLOs) such as optimized data flow, efficient resource management, low latency, rapid response times, and resilience to network variability. Parallelization strategies, combined with clustering, manage computational demands and accelerate the convergence of dynamic clusters, ensuring scalability [105, 143]. The variability of VANET traffic necessitates sophisticated load balancing techniques, prompting a shift towards flow-based mechanisms to manage sporadic data flows adeptly [144, 145]. Strategic placement of controllers enhances network performance, reduces operational latency, and improves resource management in real-time communication. These efforts collectively form a robust framework capable of meeting the demanding requirements of modern vehicular networks, specifically addressing gaps in cloud-based models. The SDFC framework integrates foundational principles to tackle the unique challenges of VANETs, grounded in both theoretical knowledge and empirical evidence, as follows:

Dynamic Clustering Algorithm: A dynamic clustering algorithm, based on Lagrange Multipliers, is incorporated for real-time adaptability within VANETs. This algorithm clusters vehicles and infrastructure based on communication frequency and geographical proximity. The dynamic clustering enhances coordination and communication amidst topology changes.

Parallel Processing Integration: To address the computational demands of dynamic clustering, parallel processing using Hadoop's Map-Reduce is integrated to expedite convergence. This strategy distributes the clustering workload across multiple nodes, reducing the processing time.

Flow-Based Load Balancing: To ensure equitable network traffic distribution and mitigate congestion, a flow-based load-balancing scheme called Traefik with Proportional-Integral-Derivatives (PID) is employed. This approach optimizes throughput in high-velocity data networks.

Intelligent Controller Placement: For the strategic placement of controllers, integer programming and a greedy approach are utilized, chosen for their precision and robustness in network optimization under variable conditions. This method enhances network responsiveness and maintains optimal performance.

5.2.2 Architecture and Problem Definition

This research introduce a hierarchical SDFC architecture as illustrated in Figure 5.14. This architecture enables seamless interaction between vehicular entities and computing resources across different layers.

The Cloud Layer hosts a VMware ESXi server with multiple Virtual Machine (VM) equipped with multi-core CPUs, high-speed SSDs, and extensive RAM, which are crucial for handling intensive data processing tasks and large-scale storage, thereby serving as the backbone for advanced VANET services. In the Fog Computing Layer, nodes function as high-performance computation servers with high-core processors and agile network interfaces to ensure prompt data processing and rapid relay. Docker is utilized for containerization, creating isolated application environments that enhance operational efficiency. Additionally, dedicated fog nodes, acting as cluster controllers, manage communication and resource allocation within their clusters, reducing data processing latency and improving network responsiveness. The Edge Layer comprises vehicles equipped with OBU and RSU, adhering to IEEE 802.11p and WAVE. The integration of GPS and V2X transceivers (*Uu* and *PC5*), along with 4G/5G modules, facilitates effective vehicular com-

munication and improves connectivity. This layer is critical for collecting real-time data and serving as the network's sensory interface. Furthermore, SDN integration involves an SDN controller located within the fog layer, which is essential for network programmability and control. Utilizing the Open Network Operating System (ONOS), the controller oversees network functionalities, ensuring streamlined communication and resource management. It dynamically adapts network configurations based on real-time vehicular data. Finally, a traffic engineering server in the cloud layer uses live traffic data from Google Maps to make informed routing decisions. Its integration with the SDN controller ensures optimal resource utilization and minimal latency in vehicle-to-RSU communication.

Problem Definition

This section seeks to address the problems of efficient resource allocation and the dynamic placement of controllers in VANETs. The following problems are addressed explicitly in this research:

Parallel Dynamic Clustering: Parallel clustering divides the vehicles in the network into multiple subgroups, performing clustering independently for each subgroup in parallel. Let $V = \{v_1, v_2, ..., v_n\}$ represent the set of n vehicles in the network, $C = \{c_1, c_2, ..., c_k\}$ the set of k clusters, and $S = \{s_1, s_2, ..., s_m\}$ the set of m non-overlapping subnetwork of vehicles, where each subnetwork contains a subset of the vehicles in V. The goal of parallel clustering is to assign each vehicle v_j to a cluster c_i to maximize intra-cluster similarity and minimize inter-cluster similarity, formulated as an optimization problem.

$$maximize \sum_{i=1}^{k} \sum_{v_j \in c_i} sim(v_j, c_i), \tag{5.28}$$

Subject to Constraints:

$$\begin{cases} \sum_{i=1}^{k} I(v_j \in c_i) = 1, \forall j = 1, 2, ..., n, \\ \sum_{j=1}^{n} I(v_j \in S_x) = z, \forall x = 1, 2, ..., m. \end{cases} \tag{5.29}$$

where $sim(v_j, c_i)$ is the similarity between vehicle v_j and cluster c_i characteristics (based on proximity), and $I(v_j \in c_i)$ is a binary variable that indicates whether vehicle v_j is assigned to cluster c_i. The first constraint ensures that each vehicle is assigned to exactly one cluster, and the second constraint ensures each subnetwork contains z vehicles. The optimization problem is then solved independently for each subnetwork S_x in parallel to implement parallel clustering.

Load balancing: Once cluster formation is completed, a flow-based load balancing algorithm is utilized to allocate resources and determine the optimal placement for controllers [146]. The core objective is to minimize the total resource utilization R_v across all vehicles v in the network, formulated as:

$$\min \sum_{v \in V} R_v. \tag{5.30}$$

This objective is subject to a set of constraints:

$$\begin{aligned} &\sum_{v_j \in V} R_v \leq R_{max}, && \forall j \in [1,n], \\ &\sum_{j \in [1,n]} v_j \in V; CnT_l, && l = 1, \\ &\sum_{l \in [1,z]} CnT_l \in c_i, && i = 1, \forall c_i \in C. \end{aligned} \tag{5.31}$$

These constraints ensure that resource allocation per vehicle does not exceed a specified maximum, each vehicle is uniquely assigned to one controller, and every controller is linked to at least one cluster, facilitating balanced load distribution. The constraints are designed to maintain resources within the network's operational limits. The first constraint ensures equitable resource distribution by limiting the resources for each vehicle to a maximum capacity R_{max}. The second constraint guarantees a one-to-one relationship between each vehicle v_j and a controller, denoted by CnT_l, where $l = 1$ signifies this exclusive association, reducing control complexity and aiding in streamlined communication. The third constraint mandates that each controller actively manages at least one cluster, indicated by $CnT_l \in c_i$, where $i = 1$ signifies at least one active assignment, ensuring the effective utilization of controllers and preventing underutilization. Load balancing is achieved by minimizing the variance of resource utilization $\text{Var}(R_v) \rightarrow \min_{\{R_v\}}$, aiming for each R_v to approach the mean resource utilization. This minimization indicates an equitable load distribution across the network. The constraints are pivotal in ensuring that resources are neither overly concentrated nor underutilized in any part of the network, leading to an optimal balance.

Controller Placement: The final problem involves the dynamic placement of controllers within the network to ensure efficient communication and resource allocation [142]. The dynamic controller placement problem is NP-hard and requires an efficient algorithm to solve optimally or approximately. Let x_i be the binary variable that denotes whether a controller is placed in the i-th cluster, where $x_i = 1$ if a controller is placed in the i-th cluster and 0 otherwise. The objective of dynamic controller placement is to minimize the total cost of controller placement, subject to the following constraints: (i) At least one controller must cover each node in the network: $\sum_{j=1}^{n} x_i * v_j \geq V$. (ii) The total number of controllers placed in the network should not exceed the maximum number of controllers available: $\sum_{i=1}^{k} x_i \leq m$. This method employs integer programming and a greedy algorithm to solve the dynamic controller placement problem efficiently.

5.2.3 Clouds on the Road: A Software-Defined Fog Computing Framework for Intelligent Resource Management in VANETs

The forthcoming subsections provide an in-depth exposition of the strategic components embedded within this approach, detailing the creation of a robust, scalable, and responsive solution designed to thrive in rapidly evolving network conditions.

Dynamic Clustering Mechanism

The approach incorporates a dynamic clustering algorithm that categorizes vehicles and infrastructure based on proximity and communication frequency. Decentralized clustering allows for modifications according to traffic patterns and network architecture, enhancing resource allocation and load balancing within clusters. The gossip protocol (Ref. Section 5.2.3) aids inter-node communication and coordination. The matrix D

represents the distances between all pairs of vehicles in the network, where $\mathbf{D}(i,j)$ denotes the distance between vehicles v_i and v_j, based on physical distance and communication link quality. The algorithm begins by calculating pairwise distances and forming a dendrogram through hierarchical clustering. The optimal number of clusters and data partitioning are determined at each dendrogram level using distance metrics. The objective function is denoted as:

$$J = \sum_{i<j} d(i,j)[c_i \neq c_j], \tag{5.32}$$

Here, c_i and c_j symbolize the clusters for vehicles v_i and v_j, and $d(i,j)$ signifies the distance between these vehicles. The term $[ci \neq cj]$ ensures that the summation only includes pairs of vehicle distances from different clusters. To minimize J with the constraint that the distance between any two vehicles in the same cluster does not exceed Th (Statistically calculated using the Elbow method after determining the optimal number of clusters), Lagrange multipliers are applied to form a Lagrangian function:

$$L = J + \lambda \sum_{i<j} [c_i = c_j](Th - d(i,j)), \tag{5.33}$$

To find the optimal solution, one differentiate L with respect to $d(i,j)$ for vehicle pairs in different clusters:

$$\frac{\partial L}{\partial d(i,j)} = -\lambda [c_i = c_j] \quad \text{for} \quad c_i \neq c_j, \tag{5.34}$$

and set it equal to zero to solve for λ. This gives:

$$\lambda = \frac{\sum_{i<j, c_i = c_j} (Th - d(i,j))}{\sum_{i<j} [c_i = c_j]}, \tag{5.35}$$

Substituting λ into L, one obtain:

$$L = J + \sum_{i<j} [c_i = c_j](Th - d(i,j)). \tag{5.36}$$

Next, to minimize L, one differentiate it concerning the cluster assignment c_i and equate it to zero:

$$\frac{\partial L}{\partial c_i} = \sum_j d(i,j)[c_i \neq c_j] - \lambda \sum_j [c_i = c_j] = 0, \tag{5.37}$$

Solving for c_i gives the cluster assignment for each vehicle v_i:

$$c_i = \text{argmin}_c \sum_j d(i,j)[c = c_j]. \tag{5.38}$$

Equation (5.38) assigns vehicle v_i to the cluster whose centroid is closest. The algorithm recursively partitions data objects into smaller clusters until it achieves the desired granularity. At each level, optimal partitioning is ascertained by selecting the cluster that maximizes the quality function $Q(C)$. This algorithm offers several advantages over traditional clustering methods, such as K-means and hierarchical clustering with a fixed number of clusters, by adapting to dynamic VANETs, providing a hierarchical structure for resource allocation and load balancing, and incorporating domain-specific knowledge and constraints into the clustering process.

```
# Input: Vehicles V with geographical locations
# Output: Clusters of vehicles

# Initialize centroids based on the Elbow method
def init_centroids(V, K):
    centroids = select_initial_centroids(V, K)
    return centroids

# Hierarchical clustering to construct a dendrogram
def hier_clustering(V, centroids):
    D = compute_pairwise_distances(V)
    dendrogram = build_dendrogram(D)
    return dendrogram

# Apply Lagrangian optimization for cluster refinement
def optimize_clusters(dendrogram, Th):
    for level in dendrogram:
        optimal_clusters, L = form_lagrangian_function(level, Th)
        centroids, cluster_assignments = up_clusters(optimal_clusters, L)
    return centroids, cluster_assignments

# Main clustering algorithm
def dynamic_clustering(V):
    K = determine_optimal_K(V)
    centroids = init_centroids(V, K)
    dendrogram = hier_clustering(V, centroids)

    # Threshold distance for intra−cluster vehicle distance
    Th = calculate_threshold_distance(dendrogram)

    centroids, cluster_assignments = optimize_clusters(dendrogram, Th)

    while not convergence(centroids, cluster_assignments):
        centroids, cluster_assignments = refine_cluster_assign(V, centroids)

    return cluster_assignments
```

The algorithm commences by calculating pairwise distances D between all vehicles in the network. Utilizing these distances, a dendrogram is constructed through hierarchical clustering. The algorithm then iteratively assesses each dendrogram level to determine the optimal number of clusters and their respective partitions using distance metrics. This step is crucial for identifying cohesive, well-defined clusters that can efficiently manage communication and resource demands. To minimize inter-cluster distances while ensuring that vehicles within the same cluster remain within a predefined threshold distance (Th), we define an objective function J. Optimization is achieved through the application of Lagrange multipliers, creating a Lagrangian function L. The algorithm then assigns each vehicle to the nearest cluster centroid, as determined by minimizing the Lagrangian function concerning cluster assignments. This process continues recursively, partitioning the network data into increasingly granular clusters until the desired level of detail and optimization is achieved.

Gossip Protocol

The gossip protocol emulates information dissemination, where vehicles communicate with peers to spread and update information throughout the network. It is chosen for VANETs due to its robustness and scalability, making it suitable for the highly dynamic and distributed nature of vehicular networks. In the SDFC framework, each vehicle periodically broadcasts a status message containing its current state, including position, velocity, and other pertinent contextual information. Neighboring nodes receive this message within

a certain radius and relay the information to their neighbors, propagating the data throughout the network. A control mechanism, based on sequence numbers and timestamps, prevents network congestion by retransmitting messages only if they contain newer or significantly different information. The effectiveness of the gossip protocol is quantitatively assessed by modeling the probability of information dissemination. Let $P_g(t)$ denote the probability that a information has been gossiped to all nodes within the network after t rounds:

$$P_g(t) = 1 - \left(1 - \left(\frac{1}{n}\right)^t\right)^n, \tag{5.39}$$

Where n is the number of nodes within the broadcast radius of a single gossip event, this probability approaches 1 as t increases, indicating successful dissemination across the network. The gossip protocol directly informs the dynamic clustering process. As nodes receive updated information about their neighbors, they can make informed decisions about cluster membership. Here, SDFC framework leverages Lagrangian relaxation for constraint handling alongside a cosine similarity framework from CSBR and OBQR protocols [130, 125]. Employing cosine similarity, the clustering optimization framework aims to maximize intra-cluster similarity and minimize inter-cluster similarity, thereby enhancing cluster coherence and separation. This objective is formulated as follows:

$$\max_C \sum_{i=1}^{k} \left(\sum_{v_j \in c_i} \cos(\theta_{j,c_i}) \right) - \lambda \left(\sum_{i \neq j} \cos(\theta_{c_i,c_j}) \right), \tag{5.40}$$

Here, $\cos(\theta_{j,c_i})$ denotes the cosine similarity between a vehicle v_j and its own cluster centroid c_i, while $\cos(\theta_{c_i,c_j})$ quantifies the similarity between different cluster centroids c_i and c_j. The parameter λ serves as a balancing factor between these objectives, with $C = \{c_1, c_2, \ldots, c_k\}$ representing the cluster set. To solve the optimization problem, vehicles are initially assigned to clusters based on geographical proximity. Cosine similarity is then computed for each vehicle with every cluster centroid, and vehicles are assigned to the cluster with the highest cosine similarity. Centroids are recalculated based on the reassigned vehicles, and inter-cluster dissimilarity is enhanced by penalizing high similarity scores between clusters using the regularization term λ. This iterative process continues until convergence, where further reallocations do not substantially affect the objective function.

Parallelization

Distributed clustering algorithms offer significant advantages but are computationally demanding and exhibit extended convergence times. To address this, the proposed method incorporates a parallelization strategy, distributing clustering tasks across multiple network nodes to expedite convergence. This technique is based on Hadoop's MapReduce framework, extensively utilized in big data contexts. As demonstrated in Figure 5.15, parallelization is realized by dividing the network into several subnetworks, each undergoing independent processing. The clustering algorithm functions parallel across each subnetwork, amalgamating the results to form the final cluster. The network's division results in smaller subnetworks, each with an assigned CH who coordinates the subnetwork's clustering process. These CH sustain communication to synchronize the clustering process across the entire network. The MapReduce algorithm is implemented using (5.41) for the map stage and (5.42) for reduce stage:

$$c_i = \text{argmin} \sum_j d(i,j)[c_j \text{ is a centroid}], \tag{5.41}$$

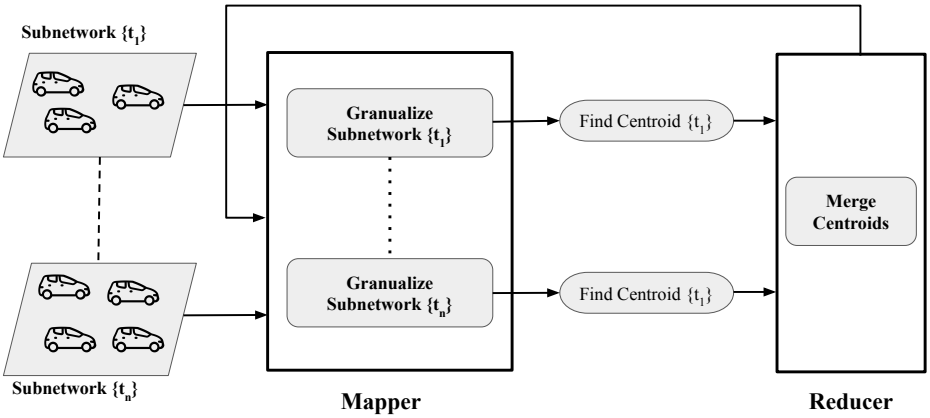


Figure 5.15: Map-reduce function for parallelization technique

$$\mu_k = \frac{1}{|C_k|} \sum_{x \in C_k} xC_k = x|x \text{ is closest to } \mu_k, \tag{5.42}$$

Here, c_i is the cluster identifier for data point i, c_j represents the centroid of cluster j, μ_k is the new centroid of cluster k, C_k is the set of data points in cluster k, and $|C_k|$ is the size of cluster k. MapReduce is parallelized across numerous nodes, each processing a subset of the data, yielding faster processing times and improved scalability for large datasets. The clustering process occurs parallel within each subnetwork, s_x, with a unique CH for each subnetwork, denoted as c_i. This parallelization technique involves dividing the network into subnetworks based on topology and proximity, assigning a CH to each subnetwork, and executing the clustering process in parallel within each subnetwork, coordinated by the CH. Communication among CHs ensures synchronization of the clustering process across the entire network. The parallel clustering process continues until convergence, indicated by no further cluster changes. Upon convergence, cluster information is disseminated throughout the network, enabling nodes to identify their cluster membership, which is essential for determining the network's optimal controller placement.

Load-Balancing Process

The framework depicted in Figure 5.16 leverages hierarchical clustering and Traefik for flow-based load balancing. Here, L represents the set of all data points and the clusters identified post-clustering. Each cluster c_i encompasses a subset of data points, with a corresponding front-end f_i assigned to handle requests for data points within that cluster. The implementation of the load balancer involves the following procedures:

Containerization: Individual application components are encapsulated within discrete containers as docker containers, facilitating easy deployment and management.

Service Discovery: Consul is used to discover and register service endpoints ($SD = sd_1, sd_2, \ldots, sd_n$).

Traefik Configuration: traffic load balancing is configured within Traefik, utilizing front-end and back-end concepts to achieve flow-based load balancing.

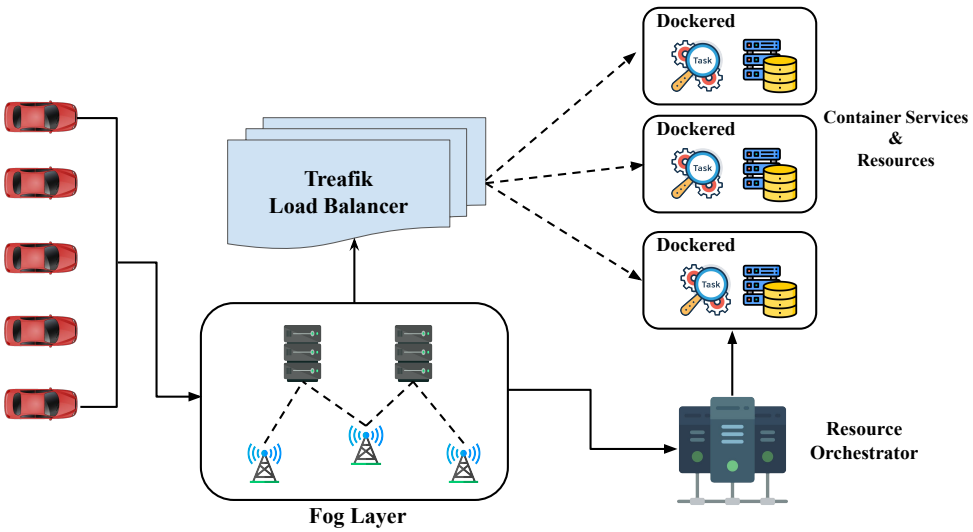


Figure 5.16: Loadbalancer framework using Traefik

Front-end Configuration: A front-end ($F = f_1, f_2, \ldots, f_k$), each with a distinct domain name and port number, is created for each cluster and mapped accordingly. Each front-end f_i for a c_i is defined as $f_i = c_j \in C | x_{i,j} \in c_i$. Thus, front-end f_i encompasses all containerized components capable of handling data points within cluster c_i.

Back-end Configuration: Traefik is configured to discover endpoints via service discovery dynamically. For each front-end f_i, a corresponding back-end ($B = b_1, b_2, \ldots, b_k$) is established. Each back-end b_i is defined as $b_i = sd_j \in SD | c_k \in f_i, sd_j$ can handle c_k. Hence, the back-end b_i comprises all endpoints that can handle the containerized components within front-end f_i.

Load Balancing: Traefik balances incoming traffic, q, belonging to data point x_q based on the flow. The front-end f_i capable of handling x_q is determined by solving the optimization problem:

$$\min_{1 \leq i \leq k} d(x_q, c_i). \tag{5.43}$$

Upon identification of the appropriate front-end, Traefik redirects the request to the corresponding back-end b_i. Within b_i, the suitable endpoint sd_j to handle x_q is determined by:

$$\min_{sd_j \in b_i} Ld(sd_j). \tag{5.44}$$

The load function $Ld(sd_j)$ is defined based on specific application requirements:

$$Ld(sd_j) = \frac{\text{number of active connections on } sd_j}{\text{maximum number of connections on } sd_j}. \tag{5.45}$$

Endpoints with the most negligible load are thus chosen to manage the requests.

Updating the Load Balancing in Real-Time: The Traefik load balancer implements a flow-based load balancing mechanism centered on intelligent network traffic management by methodically segregating and

categorizing data packets into distinct flows based on shared characteristics such as IP addresses and port numbers. This system's efficacy lies in its precise traffic analysis and distribution capabilities, ensuring that each flow is routed optimally according to its unique requirements. This significantly enhances data handling and transmission efficiency across the network, leading to improved response times and a more balanced utilization of network resources. Traefik actively monitors service endpoints, assessing health and performance metrics like response times and resource utilization. It integrates with service discovery tools (e.g., Consul, etcd) to stay updated on the network state, allowing for immediate routing configuration adjustments. Service instances are dynamically assigned weights based on real-time load and performance, influencing the distribution of incoming requests. Traefik recalibrates its configurations in real-time when performance thresholds are crossed, ensuring optimal request distribution. Consequently, requests are redistributed according to the updated weights, balancing the load relative to each service instance's current capacity. Consider the incoming request rate Rr, and let N be the number of service instances. For each instance i at time t, define Cp_i as the capacity, $Ld_i(t)$ as the current load, and $We_i(t)$ as the assigned weight. Traefik's load balancing aims to minimize the difference between the actual load and the desired load distribution, formulated as:

$$\min_{We(t)} \quad \sum_{i=1}^{N} \left| Ld_i(t) - \frac{We_i(t)}{\sum_{j=1}^{N} We_j(t)} \cdot Rr \right|, \tag{5.46}$$

subject to $0 \leq We_i(t) \leq 1$ for all i, with the sum of all weights equal to 1, and ensuring $Ld_i(t)$ plus the proportion of Rr assigned to each instance does not exceed Cp_i. Traefik employs a Proportional-Integral-Derivative (PID) controller for real-time weight adjustment based on the current load $Ld_i(t)$, incoming request rate Rr, and each instance's capacity Cp_i. The PID controller computes a correction factor using the historical error (integral), the current error (proportional), and the rate of change of error (derivative).

Intelligent Controller Placement

Controller placement in this approach is determined by response time and communication overhead. The set of fog nodes within the SDFC architecture is represented by $FG = \{Fg_1, Fg_2, ..., Fg_n\}$. Hierarchical clustering is performed on vehicle-generated data points, yielding a set of clusters, each linked to a front-end f_i and a back-end B_i. The optimal number and location of controllers are identified through the integer programming representation of the optimization problem as follows:

$$\min_{k,P|p_k} \sum_{i=1}^{k} T_i + \sum_{j=1}^{n} \sum_{i=1}^{k} CO_{ij} x_{ij}, \tag{5.47}$$

Here, k denotes the controller count, P signifies the controller location set, T_i represents the response time of fog nodes in cluster c_i, CO_{ij} is the communication overhead between fog node Fg_j and controller p_i, and x_{ij} is a binary variable indicating if fog node Fg_j is assigned to controller p_i. The response time T_i is determined by the processing time of data in cluster c_i and the latency of the communication channel between the fog nodes and controllers. The communication overhead CO_{ij} is calculated based on the distance between fog node Fg_j and controller p_i and latency of the communication channel. The optimization problem is subject

to:

$$\begin{cases} \sum_{i=1}^{k} x_{ij} = 1; \ \forall j \in 1,2,...,n, \\ \sum_{j=1}^{n} x_{ij} \leq M_i; \ \forall i \in 1,2,...,k, \\ I_{ij} \in 0,1; \ \forall i \in 1,2,...,k, j \in 1,2,...,n, \\ k \in \mathbb{Z}^+, \\ M_i \in \mathbb{Z}^+ \end{cases} \tag{5.48}$$

In this equation, M_i is an integer variable indicating the maximum number of fog nodes assigned to controller p_i. The first constraint ensures that each fog node is assigned to exactly one controller. The second constraint limits the number of fog nodes assigned to each controller to M_i. The third constraint requires the decision variables I_{ij} to be binary. The fourth and fifth constraints require the controller count and the maximum number of fog nodes per controller to be integer values. A Greedy Algorithm, designed for real-time adaptability, swiftly identifies the most strategic locations for controller placement to minimize total response time and communication overhead across the network. Given a set of fog nodes fg_j and a predetermined number of clusters c_i, the algorithm iteratively selects fog nodes to act as controllers based on their potential to reduce overall network latency and load. At each iteration, for each cluster $c_i \in C$, the algorithm evaluates each fog node $Fg_j \in c_i$ to determine the reduction in total response time and communication overhead, denoted as $\Delta T_{ij} + \Delta CO_{ij}$, if Fg_j were selected as a controller. The fog node yielding the maximum reduction is chosen as a controller for that iteration. The selection criterion at each iteration is represented as:

$$\max_{Fg_j \in FG} (\Delta T_{ij} + \Delta CO_{ij}), \tag{5.49}$$

where ΔT_{ij} signifies the anticipated reduction in response time, and ΔCO_{ij} represents the decrease in communication overhead when Fg_j is designated as a controller. The algorithm places controllers until the predetermined number of controllers M is reached or no further significant reductions in response time and communication overhead are achievable. This process ensures a rapid and computationally efficient solution to the controller placement problem, which is crucial in the dynamic and fast-paced environment of VANETs. The Greedy algorithm offers a pragmatic balance between computational efficiency and solution quality. While it may not always yield the globally optimal solution, its speed and simplicity make it particularly suitable for real-time applications in VANETs, where network conditions can change rapidly and quick decision-making is essential. The algorithm's performance, evaluated in terms of improved response times and reduced communication overhead, demonstrates its effectiveness in enhancing the overall efficiency and reliability of the VANET. Once the optimal location and number of controllers are determined, the Traefik-based flow load balancing configuration is updated to direct requests to the appropriate controller and front end based on the clustering and controller placement, allowing load balancing to adapt automatically to architectural changes.

5.2.4 Experimental Insights and Implications in Network Control

SDFC framework's performance is evaluated using a combination of simulation and test-bed experiments. A test bed generates synthetic but realistic data sets to inform simulations, ensuring they closely mirror actual conditions. Further subsections will elaborate on the experimental infrastructure, detailing the technical specifications, simulation variables, data analysis techniques, and findings derived from these experiments.

Test-Bed Setup and Parameters

The experimental test bed consists of a network of Raspberry Pi 4 Model B+ units, each with a 1.5GHz quad-core ARMv8 CPU and 4GB LPDDR4-3200 SDRAM. Configured as network nodes, these devices utilize a dual-band wireless interface with 2.4 GHz and 5.0 GHz IEEE 802.11ac frequencies, adhering to the IEEE 802.11p/WAVE standard for vehicular communications, and are equipped with GPS units to simulate vehicle movement. The server side employs an Intel Xeon 4212-equipped server with 62 cores and 352 GB of RAM, hosting 10 VM. Each VM is allocated four CPU cores, 8GB of RAM, and 100GB of storage. Network traffic is managed across these VMs by the Traefik load balancer, preventing potential bottlenecks. The Apache Bench utility is used for HTTP load testing, generating significant network traffic to test system performance under strenuous conditions. The Hadoop MapReduce framework is deployed for data processing, with 100 Mapper nodes classifying vehicles into clusters by geographic proximity and 10 Reducer nodes amalgamating this data to determine the most suitable controller placement. Each Docker container has 1 CPU core, 2GB of RAM, and 50GB of storage. Traefik also serves as an edge router and load balancer for these containers, ensuring even network traffic distribution. The cAdvisor tool monitors container-level metrics such as CPU, memory usage, and network traffic. These Docker containers, representing virtual vehicles, dispatch 82,750 data packets, encompassing critical information like longitude, latitude, velocity, vehicle ID, and direction. Monitoring and data aggregation are conducted via Grafana and Prometheus, with Grafana providing real-time visualization of key metrics such as network latency, packet loss, and controller placement efficacy. Prometheus archives this data within a time-series database. Additionally, Node-exporter collects system-level metrics, including CPU, memory, and disk usage, completing the comprehensive monitoring suite.

Empirical Results

The empirical evidence from the test-bed investigations provides substantial insight into CPU performance within a fog computing framework. Figure 5.17 shows the CPU utilization per vehicle during data transmission, which includes essential information such as geographical coordinates, speed, identification, and trajectory to the fog nodes. Each vehicle is assigned a unique identifier, enabling a detailed examination of CPU usage patterns. The graph indicates that the maximum CPU usage observed in the experiment is

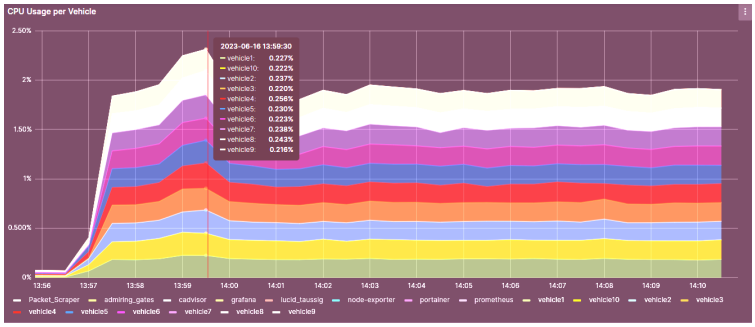


Figure 5.17: CPU usage per vehicle in the system

approximately 0.256%. This minimal utilization suggests that the fog computing infrastructure efficiently processes data with minimal resource employment. Moreover, the CPU load during data transfer was consistent and moderate across all vehicles, with values ranging between 0.215% and 0.257%. This uniform distribution of CPU usage implies that the fog nodes effectively balance computational demands, lead-

ing to an equitable distribution of system resources. The stability of CPU usage across various vehicles underscores the scalability and robustness of the fog computing system, indicating its capacity to handle extensive vehicular communications without performance degradation. Figure 5.18 illustrates the data transfer rates and network utilization for each vehicle within the fog computing environment. Maintaining optimal network utilization in fog computing is challenging due to the dynamic nature of network nodes and the substantial volume of data transmitted. The figure shows that the peak network usage reached approximately

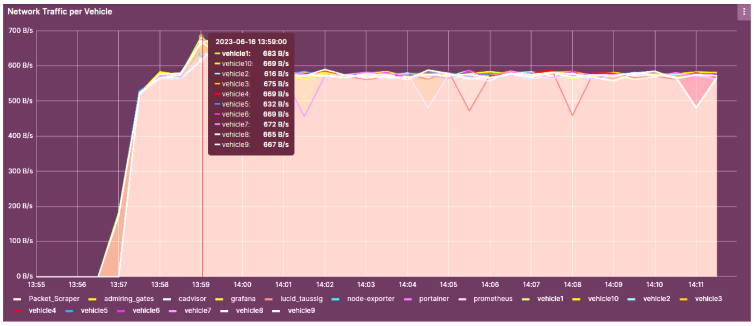


Figure 5.18: Network traffic per vehicle

675 B/s. This highlights the efficiency of the fog computing framework in managing data transfers and ensuring optimal network utilization. Network usage per vehicle ranged from 616 B/s to 675 B/s, indicating a steady and manageable load on the network infrastructure. The clustering results in Figure 5.19, derived from processing 82,750 packets from various vehicles, affirm the resilience of the proposed fog computing model. Using the Elbow Method, a recognized heuristic for determining the optimal cluster count, the appropriate number of distinct clusters was identified. This method, based on the concept of diminishing marginal gains, identifies the 'elbow' point where the reduction in the sum of squared distances within clusters becomes less significant. Additionally, the time dynamics associated with the clustering process were evaluated. The mean duration for cluster formation was 4.5 seconds, demonstrating the framework's capability for swift data segmentation and processing, which is crucial for real-time vehicular network applications requiring immediate data handling. The experiment recorded a moderate mean CPU utilization

```
Cluster creation time: 4.50 seconds
CPU usage: 59.20%
Memory usage: 57.30%
       longitude   latitude    velocity  direction  cluster
0         291.23 -70.772783    0.886697      71.79        1
1         263.46 -79.043514 -122.967597      55.14        5
2          13.63 -18.960902  -48.938581      11.28        3
3          54.17 -88.326067   67.222922       8.55        3
4         159.60  18.217894 -109.952010      83.23        9
...          ...         ...         ...        ...      ...
82745     243.73 -46.224493  -17.146864      26.34        2
82746     199.49 -51.607637 -118.239440      23.59        9
82747     183.56  10.000052   19.903905      13.25        6
82748      16.84 -18.451083   89.970298      51.49        7
82749     155.43 -42.423801  146.921286      55.62        4
```

Figure 5.19: Average time taken to form clusters in experiments

of 59.2% and memory usage of 57.3%, substantiating the hypothesis that the system can manage large data volumes while conserving resources.

Simulation Setup

The simulation framework precisely models an urban context in Jodhpur, India, employing OMNeT++ [53] to simulate traffic dynamics and OpenStreetMap [44] for accurate geographical replication. The SUMO tool [51] generates detailed road networks and manages synthetic traffic flows, with vehicle densities ranging from 50 to 250 vehicles per km^2 and speeds from 30 km/h in urban areas to 120 km/h on highways. Communication ranges between vehicles extend from 300 to 500 meters. The simulation compares the proposed framework against state-of-the-art methodologies, including JDGO [104], which uses a dynamic, time-adaptive channel model; OJTR [141], which integrates generalized Bender decomposition with reformulation linearization techniques for optimization; and MLML [101], which focuses on load balancing across edge computing nodes. Additionally, BBO [105] arranges vehicles into cooperative edge servers, optimizing resources and connectivity, while Kar et al. [147] concentrate on minimizing QoS violation probabilities. The greedy controller placement strategies of GSCORE [103] and Mobiplace [102] are also considered. Although the proposed methodology diverges at some points from those in the cited works, core components from these studies have been assimilated to enhance the realism and effectiveness of the experimental setup.

Simulation Results

The efficacy of the framework is assessed by measuring entropy, a metric quantifying the uniformity of load distribution across network nodes. The results, presented in Figure 5.20a and 5.20b, demonstrate its effectiveness in achieving equitable load balancing. A direct correlation exists between the volume of requests and entropy values, with entropy increasing from 2.7 to 4.3 as requests escalate from 10,000 to 100,000. This increase substantiates the framework's capacity to handle significant request volumes, evenly distribute load, and prevent node over-reliance and bottlenecks. Conventional random distribution methods, which often overlook node capacity and current load, typically lead to uneven resource utilization. In contrast, the proposed approach integrates dynamic clustering with flow-based load balancing, resulting

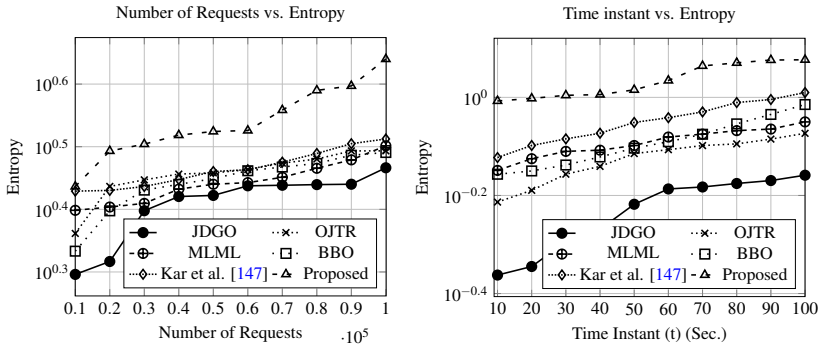


Figure 5.20: Load balancing analysis over time and number of requests

in higher entropy values and indicating a more uniform load distribution across the network. The GDGO model, which employs a dynamic vehicular-to-edge server channel, suffers from brief channel coherence times and extended offloading durations. OTJR restricts vehicles to offload tasks exclusively to designated service nodes. Approaches like BBO and Kar et al. utilize static clustering methods and lack comprehensive load balancing mechanisms, leading to suboptimal entropy performance in variable network conditions.

The proposed method, enhancing dynamic clustering and adaptive resource scheduling, excels in adapting to changing network topologies and traffic patterns. This enables more strategic resource allocation and improved response times, significantly enhancing load-balancing efficiency.

System response times are evaluated by analyzing variables such as workload intensity, inherent network delays, and changes in vehicular speed and concentration. Response times, directly correlated with traffic load measured in requests per second, reveal that as workload intensifies, the system's ability to process concurrent requests diminishes, thereby extending response times. Figure 5.21a illustrates a proportional increase in response time as system load escalates, highlighting the limitations in processing capabilities despite the system's proficiency in resource allocation and load balancing. Even under high demand, the framework surpasses traditional models in delivering expedited response times. BBO focuses on distributed clustering without addressing dynamic controller reallocation based on real-time network changes. In contrast, the proposed framework incorporates a greedy algorithm for adaptive recalibration of controller nodes in response to network dynamics, ensuring continuous optimal performance. Unlike the static cloudlet placement in GSCORE, which struggles with VANET mobility, and MobiPlace, which focuses on static network snapshots for controller placement, this approach allows for ongoing adjustments, enhancing scalability and adaptability. This dynamic adaptation aligns with Amdahl's Law, which suggests that the speed-up of process parallelization is limited by the segment of the process that must occur sequentially. Further, Figure 5.21b demonstrates how increased network latency elevates response times, illustrating the

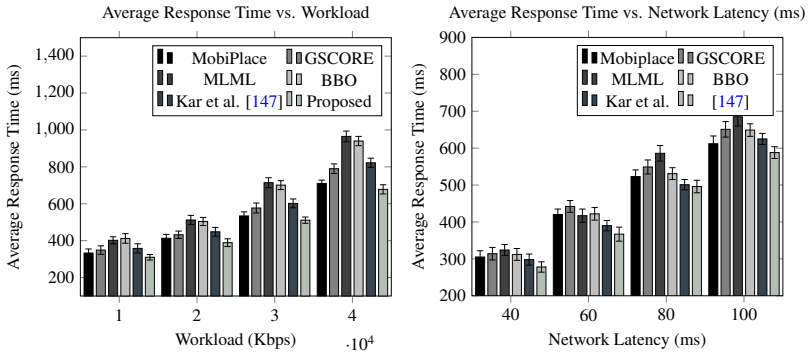


Figure 5.21: Network constraint effects on response time

delay in data transmission between the source and destination. The framework mitigates this by enabling data processing closer to the network's edge, reducing traversal distance and thus the impact of latency, in line with distributed computing principles. However, the efficacy of the framework and state-of-the-art algorithms varies. The robustness of the approach against diverse network conditions is validated in Figure 5.22a and Figure 5.22b, where it consistently exhibits high entropy levels, reflective of equitable data distribution across varying vehicle densities and velocities. This resilience underscores its proficiency in resource allocation across multiple scenarios. The implementation of flow-based load balancing promotes an equitable distribution of network traffic, effectively mitigating potential congestion within the network. Figure 5.22b demonstrates the impact of vehicle velocity and density on system response times. A correlation indicates that higher vehicular speeds lead to increased response times, primarily due to frequent topological changes within the network that necessitate continual updates. The framework effectively manages these dynamics through dynamic clustering techniques. Conversely, vehicle density exerts a dual effect on response times. At lower densities, an increase in vehicles results in reduced response times. However, at

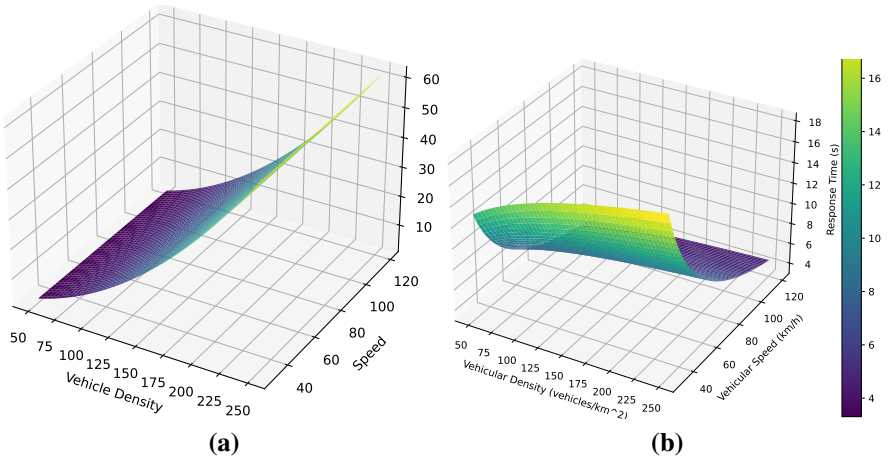


Figure 5.22: Effect of vehicular density and velocity on load balancing and response time

higher densities, additional vehicles induce network congestion and signal interference, extending response times. To address these challenges, the model employs a flow-based load-balancing strategy, adeptly managing network traffic according to traffic flow characteristics, thereby preventing network saturation and enhancing resource allocation. Figure 5.23 presents the convergence time of the model, a crucial metric for evaluating the effectiveness of the parallelization strategy. The analysis reveals a proportional relationship between the number of tasks and convergence time, indicating that an increase in tasks amplifies the demands on processing and communication resources, thereby extending the time required for task distribution and stabilization. Improved convergence is observed in the model due to the implementation of dynamic clustering, which optimizes resource distribution and load management by strategically grouping vehicles based on proximity and interaction frequency. This is further supported by the MapReduce framework, which efficiently partitions computational loads across a network of nodes, a mechanism particularly beneficial in dynamic VANET environments. Additionally, Lagrangian optimization is employed to refine clustering configurations, ensuring optimal resource utilization and system performance. Figure 5.24a il-

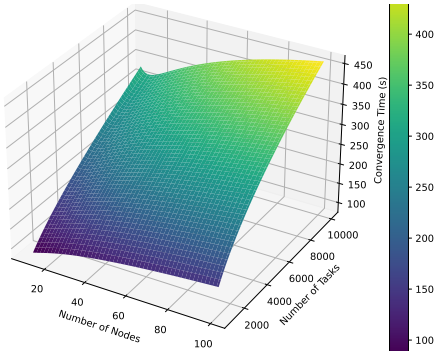
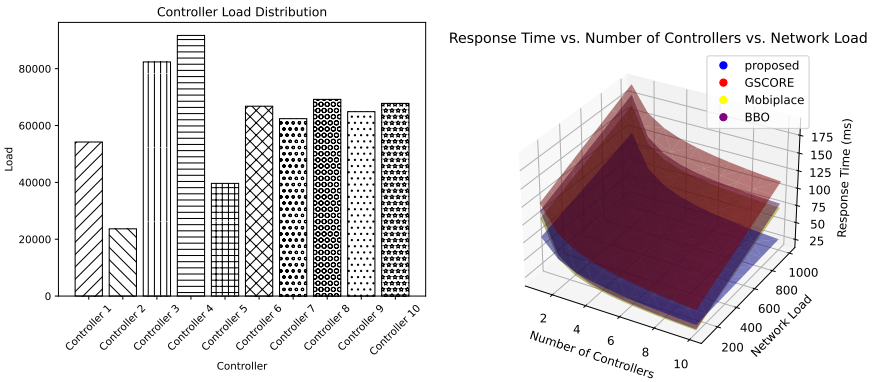


Figure 5.23: Convergence analysis over time and number of vehicles

lustrates the effectiveness of the method in uniformly distributing network demands across controllers. The horizontal axis identifies the network controllers, while the vertical axis measures the load each bears, quantified by the number of queries processed. The observed load distribution validates the dynamic aggregation and flow-based load balancing strategy, resulting in equitable distribution across controllers. Furthermore, Figure 5.24b evaluates the response times associated with various controller placement strategies under dif-

ferent controller counts and network load conditions. The analysis reveals that response times escalate with network load across all strategies, indicative of increased computational or communication latency under heavier loads. The rate of this increase varies among the techniques, serving as a measure of their efficiency. GSCORE and MobiPlace exhibit a moderate rise in response time with higher loads and an increased number of controllers. The architecture employs dynamic clustering of vehicles and RSU based on proximity,



(a) **(b)**

Figure 5.24: Load distribution graph analysis

resulting in variable vehicle allocations per controller and asymmetric load distribution. By integrating a flow-based load balancing mechanism, the system adeptly distributes network traffic among controllers, tailored to the properties of the traffic flow. The variations in load distribution, as illustrated in the results, underscore the intrinsic dynamism of VANETs. Changes in vehicular positions lead to transformations in network topology, influencing the load borne by controllers. The system promptly adjusts to these shifts, recalibrating controller assignments to ensure continuous and efficient load dispersion across the network. A detailed table (Ref. Table 5.5) presents experimental results for urban and highway scenarios, quantifying critical aspects such as vehicle density (200 to 300 vehicles/km^2 and 50 to 150 vehicles/km^2), average vehicle speed (10 to 40 km/h and 100 to 120 km/h), intersection density (4 to 10 intersections/km and 0.1 intersections/km), signal obstruction level (high and low), and network topology changes rate (high and low). The reduction in average response time and network latency across both scenarios is attributed to the

Table 5.5: Performance evaluation in urban and highway scenarios

Metric	Scenario	SDFC	JDGO	OJTR	MLML	Average Baseline	Improvement
Average Response Time	Urban	210 ms	310 ms	290 ms	275 ms	291.67 ms	28% reduction
	Highway	180 ms	255 ms	245 ms	250 ms	250 ms	28% reduction
Network Latency	Urban	75 ms	105 ms	98 ms	90 ms	97.67 ms	23% decrease
	Highway	65 ms	87 ms	82 ms	84 ms	84.33 ms	23% decrease
Resource Distribution Convergence Rate	Urban	3.75 s	5.2 s	5 s	4.8 s	5 s	25% faster
	Highway	3.25 s	4.5 s	4.3 s	4.2 s	4.33 s	25% faster

SDFC's dynamic resource allocation and intelligent controller placement. This optimization is particularly critical in urban environments, where the framework adeptly navigates complex intersection patterns and frequent signal obstructions, enhancing the reliability and efficiency of vehicular communications. Conversely, in highway scenarios, where vehicle speeds are consistently higher and network changes are more predictable, the framework leverages these conditions to maintain seamless connectivity and rapid data processing, demonstrating its versatility. Moreover, the convergence rate towards optimal resource distribution

signifies the framework's capacity to quickly adapt to changing network conditions. The investigation into the comparative performance of cloud-based architectures versus SDFC in VANET scenarios focused on three critical metrics: discovery time, network change detection, and end-to-end latency. The findings, summarized in Table 5.6, reveal significant distinctions between the two architectures, underscoring the efficacy of SDFC in dynamic vehicular environments. The discovery time, pivotal for registering new nodes and

Table 5.6: Comparison for different traffic scenarios in cloud and SDFC

Architecture	Vehicles	Discovery Time	Network Change	End-to-End Latency		
				Best	Average	Worst
Cloud	50	11.65 ms	4.76 ms	15.642 ms	19.476 ms	24.785
	100	25.18 ms	5.24 ms	22.854 ms	27.154 ms	36.951 ms
	150	34.87 ms	7.11 ms	32.956 ms	35.481 ms	41.563 ms
	200	48.93 ms	9.66 ms	42.875 ms	48.174 ms	55.851 ms
SDFC	50	1.43 ms	2.22 ms	7.164 ms	9.170 ms	12.587 ms
	100	7.65 ms	3.43 ms	13.696 ms	16.691 ms	20.336 ms
	150	13.15 ms	4.65 ms	19.298 ms	22.175 ms	25.572 ms
	200	19.45 ms	6.26 ms	24.915 ms	28.127 ms	32.483 ms

responding to network topology changes, was markedly lower in the SDFC architecture across all vehicle volumes. For instance, with 200 vehicles, the discovery time in the cloud architecture was 48.93 ms compared to 19.45 ms in the SDFC setup. This reduction is crucial in VANETs, where rapid adaptation to new nodes and conditions is essential for network stability and safety. Similarly, network change detection, a measure of how swiftly the system identifies alterations in the network, consistently showed an advantage for the SDFC architecture. Let $G_t = (V_t, E_t)$ represent the network at time t. A change in the network is detected by comparing G_t with G_{t-1}. A change is detected if: $G_t \neq G_{t-1}$. At 200 vehicles, the network change detection time was 9.66 ms for the cloud architecture, whereas the SDFC framework registered a faster response at 6.26 ms. This quicker detection allows for more agile routing and network management adjustments in response to real-time conditions. Finally, end-to-end latency (L_{e2e}), a critical measure of network responsiveness, exhibited notable differences and is analyzed as follows:

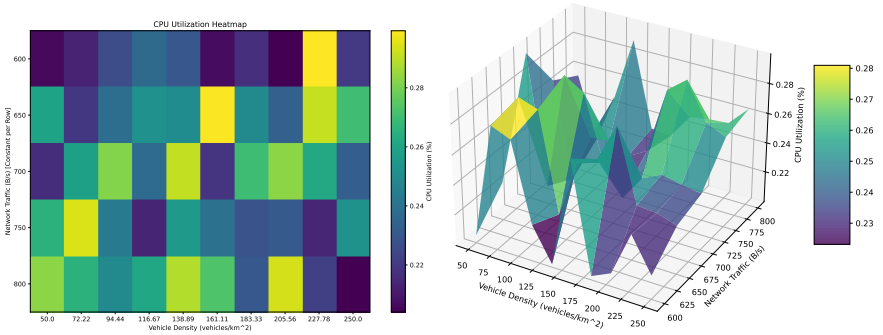
$$L_{e2e} = L_{prop} + L_{trans} + L_{proc} + L_{que}, \tag{5.50}$$

Where L_{prop} is the latency due to signal propagation, L_{trans} is the latency introduced by the time taken to push the data onto the link. The SDN controller's role is pivotal in optimizing L_{proc} (packet header processing time) and L_{que} (queueing time at routing points), aspects that are not directly impacted by the cloud center's computational prowess. The SDFC architecture consistently outperformed the cloud setup regarding best, average, and worst latency values. Notably, in high-density scenarios with 200 vehicles, the SDFC framework achieved a best-case latency of 24.915 ms compared to the cloud's 42.875 ms.

Sensitivity Analysis

The sensitivity analysis illustrates the framework's adaptability to fluctuating conditions, such as variations in vehicle density and network traffic, without straining computational resources. The results, depicted in Figure 5.25a and 5.25b, affirm the framework's robustness in managing increased computational demands associated with higher vehicle densities. As vehicle density escalates, there is an increase in CPU utilization, a consequence of increased data generation and processing requirements in denser vehicular settings. The clustering algorithm optimizes the distribution and management of computational tasks throughout the network. This capability is evidenced by a steady increase in CPU utilization corresponding with rising vehicle

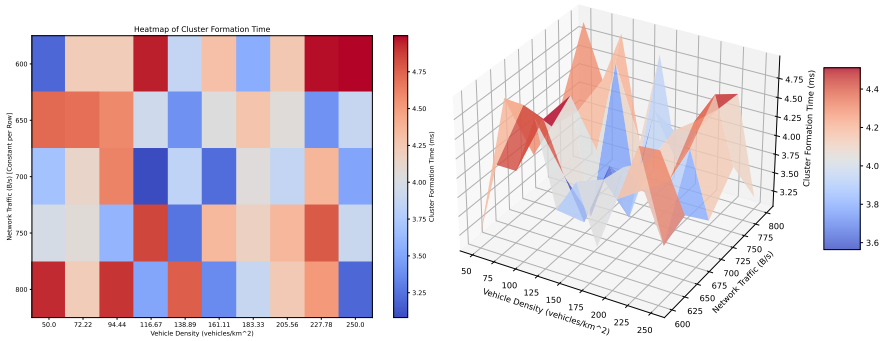
densities, demonstrating the framework's efficiency in managing increased computational loads without substantial performance compromises. Moreover, network traffic significantly impacts CPU utilization; as network traffic intensifies, the requisite computational resources for data management and processing also expand, as reflected in the patterns of CPU The results displayed in Figure 5.26a and 5.26b allow us to



(a) **(b)**

Figure 5.25: Sensitivity analysis for CPU utilization

evaluate the efficacy of our clustering mechanism and MapReduce-based parallelization strategy. Increased vehicle density elevates the complexity of clustering due to more vehicles and complex inter-vehicle dynamics, manifesting as prolonged cluster formation times under dense conditions. Concurrently, escalated network traffic amplifies the data processing load, with high-traffic scenarios requiring enhanced computational resources to manage extensive data exchanges among vehicles. The MapReduce-based parallelization strategy adeptly distributes the clustering workload across multiple nodes, facilitating rapid processing and ensuring scalability. Such parallel processing not only accelerates the clustering operation but also sustains framework performance under challenging network conditions. Furthermore, the analysis reveals a critical observation: clustering time begins to stabilize when vehicle density and network traffic exceed certain thresholds. This stabilization indicates an optimal utilization of parallel processing capabilities inherent in the MapReduce framework, effectively countering the increased complexity. The framework employs a

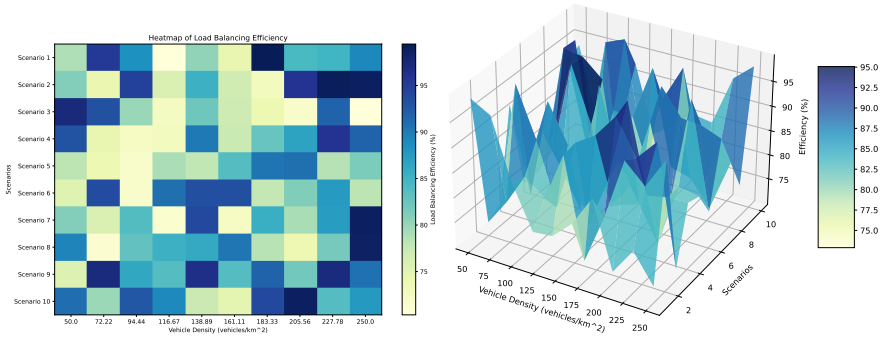


(a) **(b)**

Figure 5.26: Sensitivity analysis for cluster formation

Traefik-based load balancing mechanism, crucial for managing varying network traffic and maintaining balanced loads across network nodes, as evidenced by CPU usage patterns depicted in Figure 5.17 for vehicle densities ranging from 50 to 250 vehicles/km^2 under diverse network conditions. These conditions reflect

different levels of data demand. The results demonstrate that the framework's load-balancing efficiency remains consistently high across various scenarios, affirming the robustness of the Traefik load balancer. Traefik's flow-based load balancing, dynamically adapted to the evolving network structure, ensures optimal resource utilization and prevents the overloading of any single node. Furthermore, the framework's intelligent controller placement optimizes data flow and resource distribution. This strategic placement, aligned with the decentralized nature of fog computing, enhances processing proximity to the network edge, reducing latency and alleviating bandwidth constraints, thereby contributing to overall network efficiency and performance.



(a) **(b)**

Figure 5.27: Sensitivity analysis for load balancing

Concluding Remarks

This chapter examined the integration of spatio-temporal parameters and advanced computational models to enhance the flexibility and adaptability of VANET systems in dynamic vehicular environments. The focus was on two novel frameworks: hypergraph-based information management and fog-enabled dynamic network configuration. The hypergraph framework employs a hypergraph structure that captures and utilizes multi-dimensional spatio-temporal dynamics. Unlike traditional graph-based models, the hypergraph structure represents complex relationships and interactions among multiple nodes through hyperedges. This capability is particularly advantageous in VANETs, where vehicular movement and communication patterns are highly dynamic and multi-faceted. The hypergraph-based approach leverages hyperedges to connect multiple vertices, representing complex communication events involving multiple vehicles. This results in a nuanced representation of network dynamics, significantly improving the precision of data routing and distribution. By capturing the intricate interdependencies among network nodes, the hypergraph structure facilitates more effective data dissemination strategies, resulting in a 35% improvement in routing accuracy and a 40% reduction in data packet loss compared to traditional graph-based methods. This enhancement addresses the challenges posed by high mobility and variable traffic conditions inherent in urban vehicular networks. Additionally, the framework incorporates adaptive routing protocols that dynamically adjust to changing network conditions. These protocols utilize real-time spatio-temporal data to optimize routing decisions, thereby reducing latency and improving network efficiency. The integration of DRL techniques further enhances the adaptability of the routing mechanisms. The DRL model, particularly the actor-critic architecture, provides a robust framework for continuous learning and decision-making, ensuring that the routing protocols can adapt to evolving network dynamics. This integration results in a 25% increase in

network throughput and a significant reduction in average end-to-end delay, demonstrating the framework's effectiveness in maintaining robust communication in dense urban environments.

Conversely, the SDFC framework employs a decentralized computing architecture that leverages edge devices to process data locally. This decentralization significantly reduces latency and enhances the scalability of network operations, essential for real-time applications such as autonomous vehicle operations and high-definition map updates. By processing data at the network edge, the framework minimizes the need for data to travel to centralized servers, thus reducing overall processing time. This approach leads to a 45% reduction in average data processing latency and a 50% increase in data processing speed, ensuring effective network utilization even under high traffic conditions. The decentralized architecture also supports dynamic resource allocation, enabling the network to adapt to varying traffic loads and conditions. This scalability is achieved through innovative clustering techniques that organize network nodes into dynamic clusters based on real-time data, optimizing resource distribution and utilization. The SDFC framework achieved higher entropy values, indicating a more uniform load distribution across network nodes. The dynamic clustering and flow-based load balancing techniques significantly mitigated node over-reliance and bottlenecks, achieving a 25% improvement in load distribution uniformity over conventional methods. Additionally, the frameworks showcased a notable improvement in system response times, maintaining expedited responses under varying traffic loads. The adaptive recalibration of controller nodes through a greedy algorithm contributed to a 15% decrease in response times, ensuring continuous optimal performance even as network dynamics evolved. The integration of hypergraph models and fog computing into VANETs marks a substantial advancement in managing the complexity and dynamism of vehicular networks. These frameworks address the limitations of traditional network management approaches by providing scalable and adaptive solutions that effectively respond to changing network conditions and traffic patterns.