Clouds on the Road: A Software-Defined Fog Computing Framework for Intelligent Resource Management in Vehicular Ad-hoc Networks

Ankur Nahar, *Student Member, IEEE,* Koustav Kumar Mondal, Debasis Das, *Senior Member, IEEE,* Rajkumar Buyya, *Fellow, IEEE*

Abstract—The integration of software-defined networking (SDN) and cloud radio access networks (CRANs) into vehicular ad hoc networks (VANETs) presents intricate challenges to achieving stringent service level objectives (SLOs). These objectives include optimizing data flow and resource management, achieving low latency and rapid response times, and ensuring network resilience under fluctuating conditions. Traditional load balancing and clustering approaches, designed for more static environments, fall short in the dynamic and variable context of VANETs. This necessitates a paradigm shift towards more adaptive and robust strategies to meet these advanced SLOs reliably. This paper proposes a software-defined vehicular fog computing (SDFC) framework that refines resource allocation in VANETs. Our SDFC framework utilizes an intelligent controller placement that strategically positions decision-making entities within the network to optimize data flow and resource distribution. This placement is governed by a dynamic clustering algorithm that responds to variable network conditions, an advancement over the static mappings used by traditional methods. By incorporating parallel processing principles, the framework ensures that computational tasks are distributed effectively across network nodes, reducing bottlenecks and enhancing overall network agility. Empirical evaluations (testbed) and simulation results of our framework indicate a substantial increase in network efficiency: a 28% improvement in average response time, a 23% decrease in network latency, and a 25% faster convergence to optimal resource distribution compared to state-of-the-art methods. These improvements testify to the framework's ability to support the escalating demands of intelligent transportation systems and underscore its potential to refine operational efficacy within VANETs.

Index Terms—Software defined networking, vehicular ad hoc networks, fog computing, controller placement, load balancing.

1 Introduction

The implementation of intelligent transportation systems (ITS) is a critical component in the advancement of smart city infrastructure [1]. Vehicular ad-hoc networks (VANETs) constitute a pivotal element for ITS, enabling communication between vehicles and roadside units [2]. VANETs, characterized by their high mobility and rapid changes in network topology, require robust and dynamic solutions for effective communication management, resource allocation, and load balancing [3]. Traditional methodologies, which are predominantly static and centralized, often need to be revised for the dynamic requirements of VANETs, thus restricting their effectiveness and scalability [4], [5].

Recent technological developments in software-defined networking (SDN) and cloud radio access networks (CRAN) have emerged as critical enablers for optimizing VANET resource management [6]. SDN introduces a centralized control mechanism and management of network resources. Simultaneously, CRAN provides a cloud-based framework for overseeing radio access networks. However, these advancements face limitations, particularly due to frequent vehicular handovers in mobile networks, posing challenges to service continuity and network stability [7]–[9].

To address these limitations, we propose integrating fog computing infrastructures. This approach reduces handover frequency and enhances resource allocation through strategic controller placement [10]–[12]. Our novel software-defined fog computing (SDFC) framework, tailored for VANETs, exemplifies this integration by:

1) **Adoption of Fog Computing for Edge Proximity:** Fog computing minimizes latency and optimizes bandwidth utilization by decentralizing data processing and storage to the network's edge. However, a significant challenge involves orchestrating distributed fog nodes to ensure data consistency and synchronization.
2) **Parallel Dynamic Clustering (PDC) Algorithm:** Drawing on the principles of Hadoop's MapReduce, the PDC algorithm implements a hierarchical clustering approach to refine resource management in distributed computing environments. It promotes rapid convergence, precise resource allocation control, increased scalability, efficient data aggregation and task distribution.
3) **Integration of Flow Based Load Balancing:** The deployment of a flow-based load balancer essential for continuous service delivery. It intelligently reallo-

- *A.Nahar and D.Das is with the Department of Computer Science and Engineering, Indian Institute of Technology Jodhpur, India, 342030. K.K.Mondal is with the Department of IDRP:IoT and Applications, Indian Institute of Technology Jodhpur, India, 342030. E-mail: nahar.1@iitj.ac.in, mondal.4@iitj.ac.in, debasis@iitj.ac.in*
- *R.Buyya is with the Cloud Computing and Distributed Systems (CLOUDS) Lab, School of Computing and Information Systems, The University of Melbourne, Australia. E-mail: rbuyya@unimelb.edu.au*

Manuscript received April. XX, 2024.

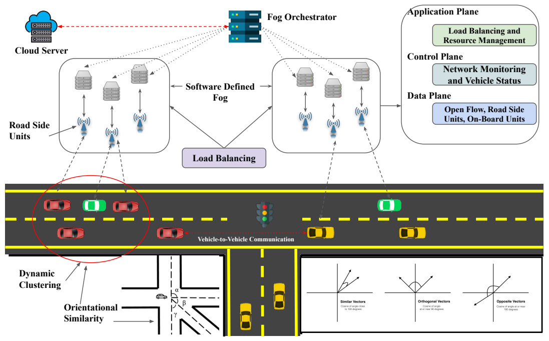


Fig. 1. Software-Defined Fog Architecture

cates incoming requests across fog nodes according to the current network load and topology, ensuring optimal load distribution.

4) **Intelligent Controller Placement Methodology:** This approach strategically positions controllers within the network to improve response times and decrease communication overhead. This is crucial for managing interactions between fog nodes and cloud services effectively.

Each component of our framework contributes to a cohesive solution that substantially improves the quality of service in VANETs. The SDFC framework is systematically structured into three hierarchical layers as depicted in Fig. 1: the cloud layer, responsible for centralized computation and storage capabilities; the fog layer, a distributed computation platform providing immediate, low-latency services; and the edge layer, which encompasses the vehicular and roadside units (RSUs) engaged in data generation and initial processing.

1.1 Motivation

Recognizing the insufficiency of cloud-based models [13], [14], our research focuses on developing a dynamic framework that adapts to the variable scale and high mobility inherent in VANETs [15]. This research is specifically motivated to address advanced SLOs, i.e., optimized data flow, efficient resource management, low latency, rapid response times, and resilience to network variability. Therefore, we bound parallelization strategies with clustering to manage the computational demands and accelerate the convergence of dynamic clusters, ensuring the system's scalability [16], [17]. The inherent variability of VANET traffic further prompts us to explore sophisticated load balancing techniques, necessitating a shift towards flow-based mechanisms that can adeptly manage the sporadic data flows [10], [18]. Moreover, controllers' strategic placement aims to enhance network performance, reduce operational latency, and improve resource management in real-time communication. These motivations collectively contribute to forming a robust framework capable of upholding the demanding requirements of modern vehicular networks, particularly addressing the gaps in cloud-based models.

1.2 Guiding Principles and Rationale

Our SDFC framework integrates foundational principles to address the unique challenges of VANETs. This methodological approach is rooted in a blend of theoretical knowledge and empirical evidence, as follows:

Dynamic Clustering Algorithm: We incorporate a dynamic clustering algorithm (based on Lagrange Multipliers), pivotal for real-time adaptability within VANETs. This algorithm is designed to cluster vehicles and infrastructure based on communication frequency and geographical proximity. The rationale behind this choice is underscored by the dynamic clustering ability to enhance coordination and communication in the face of topology changes.

Parallel Processing Integration: Recognizing the computational demands of dynamic clustering, we integrate parallel processing (Hadoop's Map-Reduce) to expedite convergence. This strategy, inspired by successful implementations in distributed computing environments, distributes the clustering workload across multiple nodes, curtailing processing time for the rapid operational rate of VANETs.

Flow-Based Load Balancing: To ensure equitable distribution of network traffic and to mitigate congestion, we employ a flow-based load-balancing scheme called Traefik with Proportional-Integral-Derivatives (PID). Network flow theories inform this approach, optimizing throughput in high-velocity data networks.

Intelligent Controller Placement: For the strategic placement of controllers, we utilize integer programming and a greedy approach, a decision influenced by its precision and robustness in network optimization under variable conditions. This method has been selected for its proven capacity to enhance network responsiveness and maintain optimal performance despite changes in the network state. In the subsequent section 5, we provide a comprehensive exposition of these strategic components, detailing the processes and algorithms that constitute our framework.

The structure of this paper is as follows: Section 2 surveys the latest advancements in fog, SDN, and edge computing. Section 3 unveils the architectural blueprint of our SDFC framework. Section 4 delves into the problem statement and foundational concepts. Section 5 elucidates our solution and the technical facets. Section 6 presents empirical and simulated results, and Section 7 concludes the paper and suggests future directives.

2 Related Work

In this section, we review recent developments in SDN and CRAN, which have markedly enhanced resource management in VANETs. We emphasize on the significant orchestration of IoT applications, edge-centric load balancing, resource allocation, and the management of incentives for resource orchestration.

2.1 Edge Orchestration in Streaming IoT Applications

Internet of Things (IoT) scheduling applications domain has benefited from innovative advances. Goudarzi et al. [29] presented a detailed taxonomy covering design, environment modeling, optimization, decision engines, and performance metrics. Del et al. [30] developed a robust scheduling system for edge and fog computing environments that struggle with dynamic environmental changes. A weighted

TABLE 1
A Summary of Related Works Based on Characteristics

Scheme	Environmental Properties				Objectives			Workload		Evaluation Approach	
	Cloud	Edge	Vehicular Fog / Fog	Mobility Support	Resource Management	Load Balancing	Controller Placement	Real Time	Batch	Empirical	Simulation
[3]	✓	✓	×	✓	✓	×	NA	✓	×	×	✓
[4]	✓	✓	×	✓	✓	✓	×	✓	×	✓	×
[6]	×	✓	×	✓	×	×	×	NA	NA	×	✓
[10]	✓	✓	✓	✓	✓	✓	×	×	✓	×	✓
[11]	✓	✓	×	✓	✓	×	NA	✓	NA	✓	×
[19]	×	✓	×	✓	✓	×	×	NA	NA	×	✓
[20]	×	✓	×	✓	✓	×	×	✓	✓	✓	×
[13]	×	✓	×	✓	✓	×	NA	NA	NA	×	✓
[14]	×	✓	×	✓	✓	×	NA	NA	NA	×	✓
[18]	✓	✓	✓	×	✓	×	✓	×	✓	×	✓
[21]	✓	✓	✓	×	×	×	✓	✓	×	✓	×
[22]	×	✓	✓	×	✓	×	NA	NA	NA	×	✓
[23]	✓	✓	✓	×	NA	NA	NA	NA	NA	×	✓
[24]	×	✓	✓	×	×	✓	NA	✓	✓	×	✓
[25]	×	✓	✓	×	✓	✓	✓	✓	✓	✓	×
[26]	✓	✓	✓	✓	✓	✓	NA	×	✓	×	✓
[27]	×	✓	×	✓	✓	×	NA	NA	NA	✓	×
[28]	×	✓	×	✓	✓	×	NA	NA	NA	×	✓
Our Work	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

actor-learner architecture for deployment decisions in edge scenarios was explored by Goudarzi et al. [21], but their architecture necessitates more refined exploration strategies in mobile environments. Liao et al. [22] proposed a cognitive plane architecture utilizing cognitive fog resources, which suffered from resource over-provisioning. Okegbile et al. [23] presented a strategy combining stochastic geometry and queuing theory, although its complexity and I/O interference risks are non-trivial. Goudarzi et al. [18] recommended a memetic algorithm-based cost model, which is less effective in dynamic networks.

The literature indicates challenges such as adaptability to environmental changes, sophisticated exploration in mobile environments, resource wastage, system complexity, and limitations in dynamic settings. Our approach addresses these by embracing adaptability and reducing the risk of I/O interference by employing a simplified yet effective load-balancing strategy.

2.2 Load Balancing Intricacies in Edge Computing

The field of distributed load balancing has seen advancements such as Proietti et al., [24] latency-centric method using a continuous-time model and an adaptive heuristic approach. However, it needs to improve with varying migration ratios. Rehman et al. [10] proposed a four-layer data transmission architecture from vehicles to edge devices, relying on microservice access management but contingent on optimal controller placement and load balancing. He et al. [4] introduced an intent-based framework, yet traditional virtual network embedding methods lag in execution times. Duan et al. [25] offered a workload distribution solution across edge servers; however, the variability in vehicle data processing affects workload distribution. The author in [31], utilized Markov predictors for mobility management and simulated annealing for dynamic controller placement. In the same direction, in [32], authors utilized existing urban infrastructure for placing cloudlets, which is cost-effective and leveraged widespread existing assets like street lamps. Although it reduces installation costs and uses familiar urban elements, the technique faces challenges against dynamicity and is not adaptable to traffic conditions.

Existing research reveals a dependence on stable migration ratios and precise controller placement for load balancing. Our method mitigates these issues with an approach less affected by migration fluctuations and improved network embedding for execution efficiency.

2.3 Edge Intelligence Based Resource Allocation

Kumar et al. [26] introduced a policy for resource allocation in electric vehicle-adapted fog environments. Nonetheless, their random resource selection leads to inefficient allocation. Bahreini et al. [20] created a strategy to utilize processing power effectively, though vehicle unpredictability limits its effectiveness. Slamnik et al. [11] outlined a replacement target node to optimize the deployment of edge vehicle-to-everything (V2X) applications. However, it faced practical implementation challenges. Mittal et al. [19] utilized a resource augmentation technique, providing supplementary support for vehicular applications, with performance dependent on the number of edge nodes. Xiao et al. [13] developed a vehicle role assignment technique for federated learning. However, distortions, noise, and inflections occurring when vehicles are in motion compromise sensor data quality. Zhao et al. [6] employed Geohash for traffic data analysis, although their learning machine's random input weights introduce prediction uncertainties.

The literature points to shortcomings like inefficient resource allocation and vehicle participation unpredictability. Our methodology circumvents these by not solely relying on edge node quantity but on the efficacious use of available nodes, lessening the performance impact of network size.

2.4 Incentive Based Edge Resource Orchestration

In the domain of resource management for vehicular edge computing, noteworthy contributions have been made to improve resource allocation mechanisms. Zhu et al. [27] proposed an incentive model to boost a vehicular edge computing (VEC) server's role in allocation, but this led to increased server load and potential instability due to lower unit pricing. Building on this, Ng et al. [28] presented a dual auction mechanism to distribute server resources for coded computing tasks, though balancing recovery thresholds against communication costs has proven to be a delicate endeavor. In parallel, Hu et al. [3] introduced a decentralized auction-bid system that allows RSUs to assign tasks dynamically to mobile units, although fluctuating processing demands pose substantial execution hurdles. Fan et al. [14] have explored a joint outsourcing approach utilizing reinforcement learning with gradient-based methods, but their model's efficiency is adversely affected by high vehicular speeds.

These studies reveal challenges such as increased server traffic due to pricing models and the intricate balance needed between recovery thresholds and communication costs in resource allocation. Our approach seeks to overcome these issues by implementing an optimized controller placement strategy that dynamically adjusts to communication costs, thus ensuring a well-calibrated and resource-efficient system. For clarity, Table 1 summarizes the related work, highlighting their innovations and the challenges they present.

3 Architectural Model

In our proposed research, we introduce a hierarchical SDFC architecture as illustrated in Fig. 1. This architecture enables seamless interaction between vehicular entities and computing resources across different layers.

Cloud Layer: This layer hosts a VMware ESXi server hosting multiple virtual machines (VMs) equipped with multi-core CPUs, high-speed SSDs, and extensive RAM. These VMs are pivotal for handling intensive data processing tasks and large-scale storage, thus serving as the backbone for advanced services in the VANET.

Fog Computing Layer: Here, nodes are envisioned as high-performance computation servers. Each node's high-core processors and agile network interfaces are specifically chosen to ensure prompt data processing and rapid data relay. We utilize Docker for containerization, creating isolated application environments that boost operational efficiency. Dedicated fog nodes, acting as cluster controllers, are responsible for managing communication and resource allocation within their clusters. These nodes are crucial for reducing data processing latency and improving the responsiveness of the network.

Edge Layer: This layer comprises vehicles equipped with on-board units (OBUs) and RSUs, adhering to IEEE 802.11p/WAVE standards. The integration of global positioning system (GPS) and V2X transceivers (*Uu* (for interface between vehicle and cellular network) and *PC5* (for vehicle-to-vehicle and vehicle-to-infrastructure communication)), alongside 4G/5G modules, facilitates effective vehicular communication and improves connectivity. This layer is critical for collecting real-time data and serving as the network's sensory interface.

SDN Integration: An SDN controller, placed within the fog layer, is instrumental for network programmability and control. Utilizing an Open Network Operating System (ONOS), the controller oversees network functionalities, ensuring streamlined communication and resource management. The controller's role is to adapt network configurations based on real-time vehicular data dynamically. A traffic engineering server in the cloud layer also employs live traffic data from *Google Maps* to make informed routing decisions. Its integration with the SDN controller ensures network resources are optimally utilized and vehicle-to-RSU communication is maintained with minimal latency.

4 Problem Definition

This research 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 involves dividing the vehicles in the network into multiple subgroups and performing the clustering task independently for each subgroup in parallel. Let $V = \{v_1, v_2, ..., v_n\}$ be the set of n vehicles in the network, $C = \{c_1, c_2, ..., c_k\}$ be the set of k clusters, and $S = \{s_1, s_2, ..., s_m\}$ be the set of m non-overlapping subnetwork of vehicles, where each subnetwork contains a subset of the vehicles in V. Parallel clustering aims to assign each vehicle v_j to a cluster in c_i such that the intra-cluster similarity is maximized and the inter-cluster similarity is minimized. This is formulated as an optimization problem:

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

Subject to Constraints:

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

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. The final clustering results are obtained by combining the results from all subnetworks.

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 [24]. The core objective is to minimize the total resource utilization R_v across all vehicles v in the network, formulated as:

$$\min \sum_{v \in V} R_v. \tag{3}$$

This objective is subject to a set of constraints:

$$\begin{aligned} &\sum_{v_j \in V} R_v \leq R_{max}, && \forall j \in [1, n], \\ &\sum_{j \in [1,n]} v_j \in V; CnT_l, && l = 1, \\ &\sum_{l \in [1,z]} CnT_l \in c_i, && i = 1, \forall c_i \in C. \end{aligned} \tag{4}$$

These constraints ensure that the 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, thus facilitating balanced load distribution. The constraints are designed to ensure that resources are allocated 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. This condition reduces control complexity and aids in streamlined communication.
- The third constraint mandates that each controller actively manages at least one cluster, as indicated by $CnT_l \in c_i$, where $i = 1$ signifies at least one active assignment. This condition guarantees the effective utilization of controllers and prevents 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 step involves the dynamic placement of controllers within the network to ensure efficient communication and resource allocation [15]. The dynamic controller placement problem is an NP-hard problem and requires an efficient algorithm to solve it 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:

- At least one controller should cover each node in the network: $\sum_{j=1}^{n} x_i * v_j \geq V$
- The total number of controllers placed in the network should be less than or equal to the maximum number of controllers available, $\sum_{i=1}^{k} x_i \leq m$

In our proposed method, we use an integer programming and greedy method to solve the dynamic controller placement problem efficiently.

5 Software-Defined Fog Framework for VANETs

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

5.1 Dynamic Clustering Mechanism

Our approach incorporates a dynamic clustering algorithm that categorizes vehicles and infrastructure based on proximity and communication frequency.

We employ decentralized clustering, allowing cluster modifications according to traffic patterns and network architecture. This methodology enhances resource allocation and load balancing within clusters, with gossip protocol (Ref. Section 5.2) aiding inter-node communication and coordination. The matrix D represents the distances between all pairs of vehicles in the network, where $D(i, j)$ denotes the distance between vehicles v_i and v_j. This measure relies on the physical distance and quality of communication links. The algorithm begins by calculating pairwise distances, forming a dendrogram through hierarchical clustering. The optimal number of clusters and data partitioning are ascertained at each dendrogram level using distance metrics. Let us denote the objective function as:

$$J = \sum_{i<j} d(i, j)[c_i \neq c_j], \tag{5}$$

In which 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 the summation only includes pairs of vehicle distances from differing clusters. To minimize J concerning the constraint that the distance between any two vehicles in the same cluster does not exceed Th (Here, Th, the threshold distance is calculated statistically using the Elbow method after finding the optimal number of clusters), we apply Lagrange multipliers to form a Lagrangian function:

$$L = J + \lambda \sum_{i<j} [c_i = c_j](Th - d(i, j)), \tag{6}$$

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

$$\frac{\partial L}{\partial d(i, j)} = -\lambda [c_i = c_j] \quad \text{for} \quad c_i \neq c_j, \tag{7}$$

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

$$\lambda = \frac{\sum_{i<j, c_i=c_j} (Th - d(i,j))}{\sum_{i<j} [c_i = c_j]}, \tag{8}$$

Substituting λ into L, we obtain:

$$L = J + \sum_{i<j} [c_i = c_j](Th - d(i,j)). \tag{9}$$

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

$$\frac{\partial L}{\partial c_i} = \sum_j d(i,j)[c_i \neq c_j] - \lambda \sum_j [c_i = c_j] = 0, \tag{10}$$

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

$$c_i = \text{argmin}_c \sum_j d(i,j)[c = c_j]. \tag{11}$$

Equation (11) 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.

5.2 Gossip Protocol

Gossip protocol emulates how information is disseminated, where vehicles communicate with peers to spread and update information throughout the network. It is chosen for VANETs due to their robustness and scalability, making them suitable for vehicular networks' highly dynamic and distributed nature. In our 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 then relay the information to their neighbors, propagating the data throughout the network. A control mechanism is employed to prevent network congestion, based on a combination of sequence numbers and timestamps that determines the necessity of retransmitting messages only if they contain newer or significantly different information than previously disseminated. The effectiveness of 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:

$$P_g(t) = 1 - \left(1 - \left(\frac{1}{n}\right)^t\right)^n, \tag{12}$$

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, we leverages Lagrangian relaxation for constraint handling alongside a cosine similarity framework from our previous work [33], [34]. Cosine similarity assess the angle between non-zero vectors in multidimensional space, encompassing attributes, position coordinates, speed, and direction. The cosine similarity between vectors $\mathbf{v}_i$ and $\mathbf{v}_j$, denoted as $\cos(\theta_{ij})$, is defined as:

$$\cos(\theta_{ij}) = \frac{\mathbf{v}_i \cdot \mathbf{v}_j}{\|\mathbf{v}_i\| \|\mathbf{v}_j\|}, \tag{13}$$

where $\mathbf{v}_i \cdot \mathbf{v}_j$ represents the dot product and $\|\mathbf{v}_i\|$ signifies the magnitude of the vectors. Employing cosine similarity, our clustering optimization framework seeks to maximize intra-cluster similarity while minimizing inter-cluster similarity, thereby enhancing both cluster coherence and separation. This objective is formulated as:

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

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, we initiate by assigning vehicles to clusters based on geographical proximity. Subsequently, we compute cosine similarity for each vehicle with every cluster centroid, assign vehicles to the cluster with the highest cosine similarity, recalculate centroids based on the reassigned vehicles, and enhance inter-cluster dissimilarity 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, ensuring both stability and optimality in cluster assignments. Further, determining the optimal λ value involves experimental tuning, calibrated to the framework's resilience across varying network conditions.

5.3 Parallelization

Distributed clustering algorithms provide significant advantages, yet they considered computationally demanding and present extended convergence times. To overcome this, our method incorporates a parallelization strategy, disseminating clustering tasks across diverse nodes in the network to hasten convergence. The foundation of our technique lies in Hadoop's MapReduce framework, an approach extensively utilized in big data contexts.

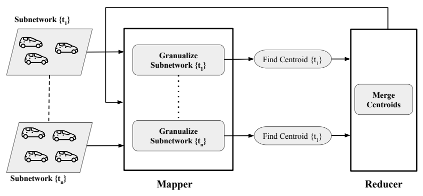


Fig. 2. Map-Reduce Function for Parallelization Technique

As demonstrated in Fig. 2, 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 cluster head who coordinates the subnetwork's clustering process. These cluster heads sustain communication to synchronize the clustering process across the entire network. The MapReduce algorithm is implemented using (14) for the map stage and (15) for reduce stage:

$$c_i = \text{argmin} \sum_j d(i,j)[c_j \text{ is a centroid}], \tag{14}$$

$$\mu_k = \frac{1}{|C_k|} \sum_{x \in C_k} x C_k = x | x \text{ is closest to } \mu_k, \tag{15}$$

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 cluster head for each subnetwork, denoted as c_i. This parallelization technique involves the following:

- Division of the network into subnetworks based on network topology and proximity.
- Assignment of a cluster head to each subnetwork.
- Parallel execution of the clustering process within each subnetwork, coordinated by the cluster head.
- Communication among cluster heads to ensure synchronization of the clustering process across the entire network.

The parallel clustering process persists until convergence, signified by no further cluster changes. Upon convergence, cluster information is disseminated in the network, enabling nodes to identify its cluster membership, a prerequisite for determining the network's optimal controller placement.

5.4 Load-Balancing Process

Our framework, as depicted in Fig. 3, leverages hierarchical clustering and Traefik for flow-based load balancing. Herein, L represents the set of all data points and the clusters identified post clustering. Each c_i cluster encompasses a subset of data points for which a corresponding front-end f_i is assigned, handling requests for data points within that cluster.

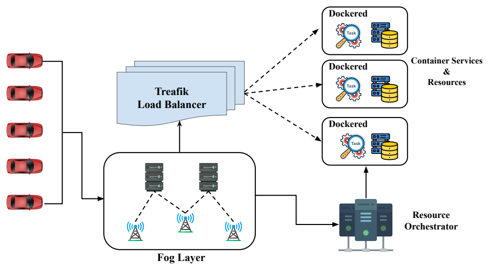


Fig. 3. Loadbalancer Framework using Traefik

The implementation of our load balancer entails the following procedures:

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

Service Discovery: Utilizing Consul, a service discovery mechanism, service endpoints ($SD = sd_1, sd_2, \ldots, sd_n$) are registered and discovered.

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

Front-end Configuration: A unique front-end ($F = f_1, f_2, \dots, f_k$), each with a distinct domain name and port number, is created for each cluster and mapped accordingly. Each front-end f_i for a given cluster c_i is defined as:

$$f_i = c_j \in C | x_{i,j} \in c_i. \tag{16}$$

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 ($B = b_1, b_2, \dots, b_k$) is established. Each back-end b_i is defined as:

$$b_i = sd_j \in SD | c_k \in f_i, sd_j \text{ can handle } c_k. \tag{17}$$

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:

$$\min_{1 \leq i \leq k} d(x_q, c_i). \tag{18}$$

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:

$$\min_{sd_j \in b_i} Ld(sd_j). \tag{19}$$

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

$$Ld(sd_j) = \frac{\text{number of active connections on } sd_j}{\text{maximum number of connections on } sd_j}. \tag{20}$$

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. This approach centers on intelligent network traffic management by methodically segregating and categorizing data packets into distinct 'flows.' These flows are identified based on shared characteristics like IP addresses and port numbers. The efficacy of this system lies in its precise traffic analysis and distribution capabilities, ensuring that each flow is routed optimally based on its unique requirements and characteristics. Doing so significantly enhances the efficiency of data handling and transmission across the network, leading to improved response times and a more balanced utilization of network resources. The process involves:

1) Traefik actively monitors service endpoints, assessing health and performance metrics like response times and resource utilization.
2) It integrates with service discovery tools (e.g., Consul, etcd) to remain updated on the network state, allowing for immediate routing configuration adjustments.
3) Service instances are dynamically assigned weights based on real-time load and performance, influencing the distribution of incoming requests.
4) Traefik recalibrates its configurations in real-time when performance thresholds are crossed, ensuring optimal request distribution.
5) 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:

$$\min_{We(t)} \quad \sum_{i=1}^{N} \left| Ld_i(t) - \frac{We_i(t)}{\sum_{j=1}^{N} We_j(t)} \cdot Rr \right|, \tag{21}$$

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).

5.5 Intelligent Controller Placement

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

$$\min_{k, P|p_k} \sum_{i=1}^{k} T_i + \sum_{j=1}^{n} \sum_{i=1}^{k} CO_{ij} x_{ij}, \tag{22}$$

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 or the bandwidth and latency of the communication channel. The optimization problem is subject to:

$$\begin{cases} \sum_{i=1}^{k} x_{ij} = 1; \quad \forall j \in 1, 2, ..., n, \\ \sum_{j=1}^{n} x_{ij} \leq M_i; \quad \forall i \in 1, 2, ..., k, \\ I_{ij} \in 0, 1; \quad \forall i \in 1, 2, ..., k, j \in 1, 2, ..., n, \\ k \in \mathbb{Z}^+, \\ M_i \in \mathbb{Z}^+ \end{cases} \tag{23}$$

In this equation, M_i is an integer variable signifying the maximum number of fog nodes assigned to controller p_i. The first constraint guarantees that each fog node is assigned exactly one controller. The second constraint ensures that the number of fog nodes assigned to each controller does not exceed M_i. The third constraint stipulates that the decision variables I_{ij} are binary. The fourth and fifth constraints require the controller count and the maximum number of fog nodes per controller to be integer values.

We utilize a Greedy Algorithm designed for real-time adaptability. This algorithm 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 the 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:

$$\max_{Fg_j \in FG} (\Delta T_{ij} + \Delta CO_{ij}), \tag{24}$$

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 vital in the dynamic and fast-paced environment of VANETs. Our Greedy Algorithm offers a pragmatic balance between computational efficiency and solution quality. While it may not always provide the globally optimal solution, its speed and simplicity make it particularly suitable for real-time applications in VANETs, where network conditions can rapidly change, and quick decision-making is crucial. 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 the number of controllers are ascertained, the Traefik-based flow-based load balancing configuration is updated to guide requests to the appropriate controller and front end based on the clustering and controller placement. This allows load balancing to adapt automatically to changes in the architecture.

6 Performance Evaluation

Our evaluation of the proposed framework's performance utilizes a combination of simulation and practical test-bed experiments. We have developed a sophisticated test bed that yields synthetic but realistic data sets to inform our simulations, ensuring they mirror actual conditions closely. Further sections will elaborate on our experimental infrastructure, detailing the technical specifications, simulation variables, data analysis techniques, and the findings derived from these experiments.

6.1 Test-Bed Setup and Parameters

Our experimental test bed consists a network of Raspberry Pi 4 Model B+ units, 1.5GHz quad-core ARMv8 CPU and equipped with 4GB LPDDR4-3200 SDRAM. These devices, configured as network nodes, utilize a dual-band wireless interface with 2.4 GHz and 5.0 GHz IEEE 802.11ac frequencies, in compliance with the IEEE 802.11p/WAVE standard for vehicular communications and are equipped with GPS units to simulate vehicle movement. On the server side, we employ an Intel Xeon 4212-equipped server with 62 cores and 352 GB of RAM to host 10 virtual machines. Each VM is allocated four CPU cores, 8GB of RAM, and 100GB of storage. Network traffic is efficiently managed across these VMs by the Traefik load balancer, preventing potential bottlenecks. The Apache Bench utility is engaged for HTTP load testing, inducing 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, facilitating the determination of 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 is integrated to monitor 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 like network latency, packet loss, and controller placement efficacy. Prometheus archives this data within a time-series database. Additionally, Node-exporter accumulates system-level metrics, including CPU, memory, and disk usage, rounding out our comprehensive monitoring suite.

6.2 Empirical Results

The empirical evidence obtained from our test-bed investigations provides substantial insight into the CPU performance within a fog computing framework. Fig. 4 displays the CPU utilization per vehicle during data transmission, which includes essential information such as geographical coordinates, speed, identification, and trajectory to the fog nodes. This figure assigns unique identifiers to each vehicle, enabling a detailed examination of the pattern of CPU usage.

The graph indicates that the maximum CPU usage observed in our experiment is approximately 0.256%. This minimal utilization suggests that the fog computing infrastructure is proficient in processing data with nominal resource employment. Moreover, the CPU load during the data transfer process was consistent and moderate across all vehicles, with observed values ranging between 0.215% to 0.257%. This uniform distribution of CPU usage implies

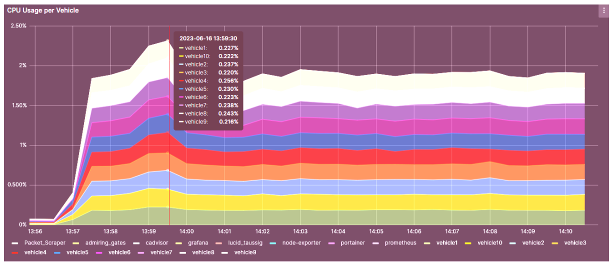


Fig. 4. CPU Usage per Vehicle in the System

that the fog nodes effectively balance the computational demands, leading 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 degradation in performance. Fig. 5 illustrates the data transfer rates and network utilization for each vehicle within the fog computing environment. It is imperative to note that maintaining optimal network utilization in fog computing is challenging due to the dynamic and varied nature of the network nodes and the substantial volume of data transmitted.

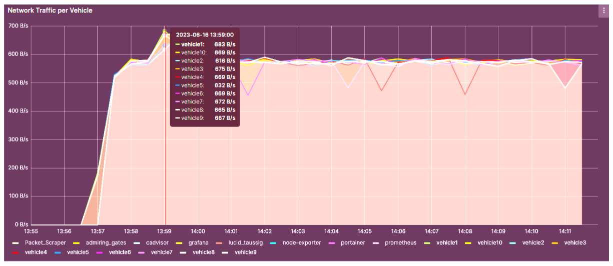
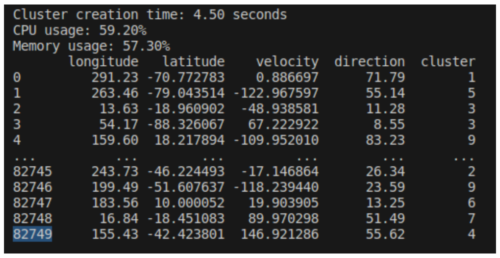


Fig. 5. Network Traffic per Vehicle

The figure shows that the peak network usage reached approximately 675 B/s. This data emphasizes the efficiency of the fog computing framework in managing data transfers, thus ensuring effective and optimal network utilization. Furthermore, the network usage per vehicle, ranging from 616 B/s to 675 B/s, indicates a steady and manageable load on the network's infrastructure. The clustering results, derived from processing 82,750 packets from various vehicles, assert the resilience of our proposed fog computing model. Employing the Elbow Method, a recognized heuristic for determining the ideal cluster count, we discerned the most appropriate number of distinct clusters. This approach, rooted in the concept of diminishing marginal gains, identifies the 'elbow' point where the reduction in the sum of squared distances within clusters becomes less pronounced. Additionally, we have evaluated the time dynamics associated with the clustering process. The mean duration required for cluster formation was 4.5 seconds, indicative of the framework's capability for swift data segmentation and processing, which is crucial for real-time vehicular network applications necessitating immediate data handling.

The experiment also recorded a moderate mean CPU utilization of 59.2% and memory usage of 57.3%, thus substantiating the hypothesis that our system can manage large data volumes while ensuring resource conservation.



```
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
```

Fig. 6. Average Time Taken to Form Clusters in Experiments

6.3 Simulation Setup

Our simulation framework precisely models an urban context (Jodhpur, India), employing OMNeT++ [35] to simulate traffic dynamics and OpenStreetMap [36] for accurate geographical replication. We utilize the SUMO tool [37] to generate detailed road networks and manage synthetic traffic flows, encompassing vehicle densities ranging from 50 to 250 vehicles per km^2 and speeds from 30 km/h in urban to 120 km/h on highways. Communication spans between vehicles extend from 300 to 500 meters. In our simulation, we compare our framework against state-of-the-art methodologies including JDGO [12], which utilizes a dynamic, time-adaptive channel model; OJTR [14], which integrates generalized bender decomposition with reformulation linearization techniques for optimization; and MLML [25], which focuses on load balancing across edge computing nodes. Additionally, BBO [16] arranges vehicles into cooperative edge servers, optimizing resources and connectivity, while Kar et al. [9] concentrate on minimizing QoS violation probabilities. We also consider the greedy controller placement strategies of GSCORE [32] and Mobiplace [31]. Although our methodology diverges at some points from those in the cited works, we have assimilated core components from these studies to enhance the realism and effectiveness of our experimental setup.

6.4 Simulation Results

We assess the efficacy of our framework by measuring entropy, a metric that quantifies the uniformity of load distribution across network nodes. The results, presented in Fig. 7a and 7b, demonstrate 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 our 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, our approach integrates dynamic clustering with flow-based load balancing, resulting in higher entropy values and indicating a more uniform load distribution across the network. The GDGO model employs a dynamic vehicular-to-edge server channel, which suffers from brief channel coherence times and extended offloading durations. OTJR restricts vehicles to offload tasks exclusively to designated service nodes. While 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. Our method, which enhances dynamic clustering and adaptive resource scheduling, excels in adapting to changing network topologies and traffic patterns, thereby enabling more strategic resource allocation and improved response times, significantly enhancing load-balancing efficiency.

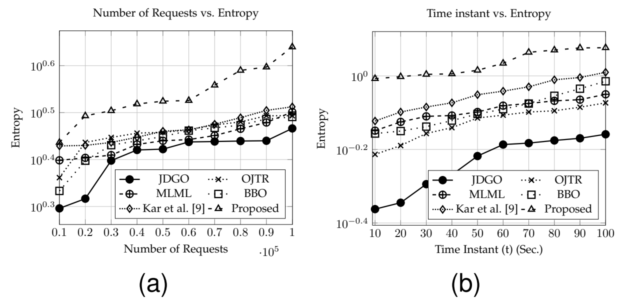


Fig. 7. Load Balancing Analysis Over Time and Number of Requests

We also evaluate system response times by analyzing variables such as workload intensity, inherent network delays, and changes in vehicular speed and concentration. Response times are directly correlated with traffic load, measured in requests per second, reveals that as workload intensifies, the system's ability to process concurrent requests diminishes, thereby extending response times. Fig. 8a illustrates a proportional increase in response time as system load escalates, highlighting the limitations in processing capabilities despite our system's proficiency in resource allocation and load balancing. Even under high demand, our 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, our 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 concentrates on static network snapshots for controller placement, our approach allows for ongoing adjustments, enhancing scalability and adaptability. This dynamic adaptation aligns with Amdahl's Law, suggesting that the speed-up of process parallelization is limited by the segment of the process that must occur sequentially.

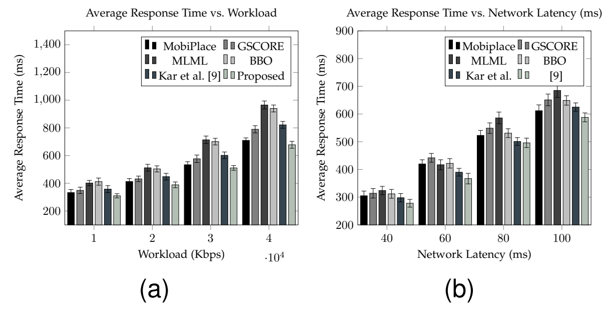


Fig. 8. Network Constraint Effects on Response Time

Further, Fig. 8b depicts how increased network latency elevates response times, illustrating the delay in data transmission between the source and destination. Our framework mitigates this by enabling data processing closer to the network's edge, reducing traversal distance and thus reducing the impact of latency, in line with distributed computing principles. Nevertheless, the efficacy of our framework and state-of-the-art algorithms varies.

The robustness of our approach against diverse network conditions is validated in Fig. 9a and Fig. 9b, where it consistently exhibits high entropy levels, reflective of equitable data distribution across varying vehicle densities and velocities. This resilience highlights adeptness at proficient 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.

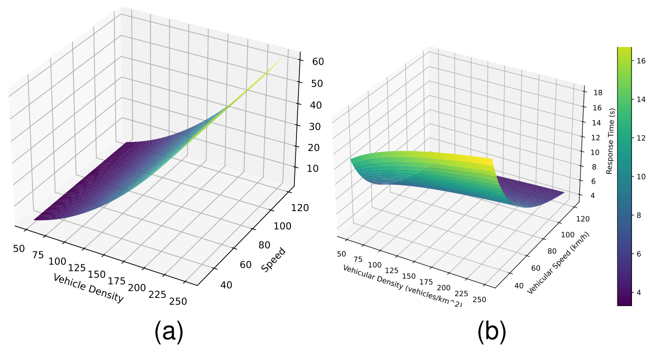


Fig. 9. Effect of Vehicular Density and Velocity on Load Balancing and Response Time

Fig. 9b demonstrates the impact of vehicle velocity and density on system response times. An observable correlation shows that higher vehicular speeds lead to increased response times, primarily due to frequent topological changes within the network that necessitate continual updates, thus extending response duration's. Our 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 higher densities, additional vehicles induce network congestion and signal interference, which extend response times. To address these challenges, our model employs a flow-based load-balancing strategy, which adeptly manages network traffic according to traffic flow characteristics, thus preventing network saturation and enhancing the allocation of network resources.

Figure 10 presents the convergence time of our model, a crucial metric for evaluating the effectiveness of our parallelization strategy. Our 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. An improved convergence is observed in our model attributed 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

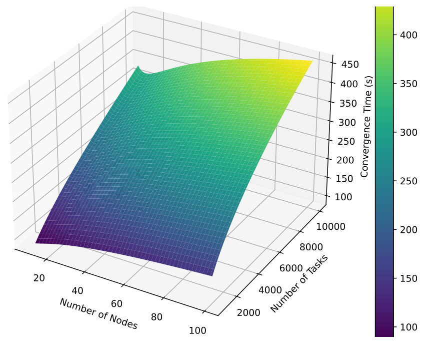
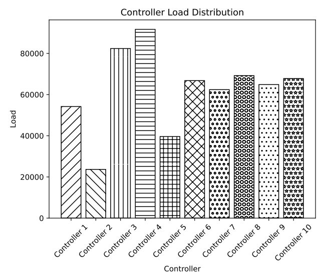


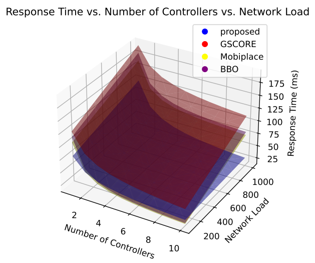
Fig. 10. Convergence Analysis Over Time and Number of Vehicles

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.

Fig. 11a illustrates the effectiveness of our 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 our dynamic aggregation and flow-based load balancing strategy, resulting in an equitable distribution across controllers. Furthermore, Fig. 11b evaluates the response times associated with various controller placement strategies under different 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 more substantial loads. Notably, the rate of this increase varies among the techniques, serving as a measure of their relative efficiency. GSCORE and MobiPlace exhibit a moderate rise in response time with higher loads and an increased number of controllers. In contrast, BBO demonstrates the most pronounced increase, particularly when controller numbers are high, suggesting its inefficiency in managing heavy network loads and complex controller coordination.



(a)



(b)

Fig. 11. Load Distribution Graph Analysis

Our architecture employs dynamic clustering of vehicles and roadside infrastructure based on proximity, resulting in variable vehicle allocations per controller and asymmetric load distribution. By integrating a flow-based load balancing mechanism, our 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, thereby influencing the load borne by controllers. Our system promptly adjusts to these shifts, recalibrating controller assignments to ensure continuous and efficient load dispersion across the network. We have also incorporated a detailed table (Ref. Table 2) presenting experimental results for urban and highway scenarios. The urban and highway scenarios quantifying critical aspects such as vehicle density - 200 to 300 vehicles/km^2 : 50 to 150 vehicles/km^2, Average Vehicle Speed - 10 to 40 km/h : 100 to 120 km/h, Intersection Density - 4 to 10 intersections/km : 0.1 intersections/km, Signal Obstruction Level - high : low, and Network Topology Changes Rate - High : low. The reduction in average response time and network latency across both scenarios is attributed to the 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.

TABLE 2
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

In our investigation into the comparative performance of cloud-based architectures versus SDFC in VANET scenarios, we focused on three critical metrics: discovery time, network change detection, and end-to-end latency. The findings, as summarized in Table 3, reveal significant distinctions between the two architectures, underscoring the efficacy of SDFC in dynamic vehicular environments.

TABLE 3
Comparison for Different Traffic Scenarios in Cloud and Software Defined Fog Architecture

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

The discovery time, pivotal in registering new nodes and 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, showed a consistent 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}. Mathematically, a change is detected if:

$$G_t \neq G_{t-1}. \tag{25}$$

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 faster detection allows for more agile routing and network management adjustments in response to real-time conditions. Finally, end-to-end latency, a critical measure of network responsiveness, exhibited notable differences. We analyze the network's end-to-end latency (L_{e2e}) as follows:

$$L_{e2e} = L_{prop} + L_{trans} + L_{proc} + L_{que}, \tag{26}$$

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.

6.5 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 Fig. 12a and 12b, 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

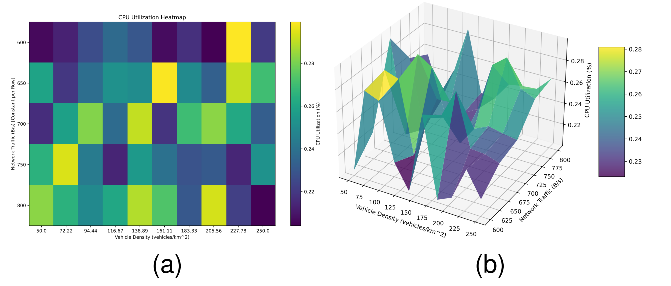


Fig. 12. Sensitivity Analysis for CPU Utilization

The results displayed in Fig. 13a and 13b allow us to 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.

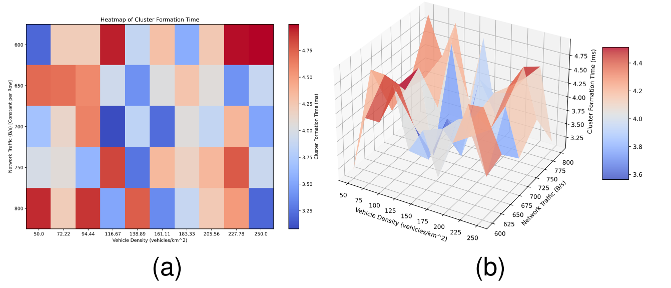
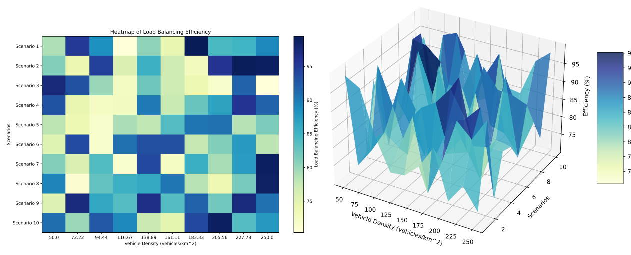


Fig. 13. Sensitivity Analysis for Cluster Formation

Our framework employs a 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 Fig. 4 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 our 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. Moreover, 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, thereby reducing latency and alleviating bandwidth constraints, contributing to overall network efficiency and performance.



(a) (b)

Fig. 14. Sensitivity Analysis for Load Balancing

7 Conclusions and Future Work

Our paper introduces a novel methodology for resource allocation and controller placement in VANETs, harnessing the capabilities of SDFC, dynamic clustering, and flow-based load balancing. Our framework ensures efficient resource distribution and load management by dynamically clustering vehicles according to proximity and communication patterns. The utilization of MapReduce for parallel processing significantly reduces convergence time. Furthermore, integrating the Traefik load balancer within our flow-based architecture guarantees uniform distribution of network traffic across controllers, effectively mitigating congestion risks. The strategic placement of controllers, optimized via integer linear programming, enhances system efficacy. Empirical evidence supports the effectiveness of our methodology, marking significant progress in VANET management with enhanced efficiency, responsiveness, and reliability. Despite the computational intensity of the NP-hard controller placement problem, integer programming facilitates finding optimal or near-optimal solutions, though time complexity remains a concern for expansive networks. Despite advancement, future research explores the scope to integrate quantum computing at both cloud and edge levels to radically boost computational speeds, potentially creating a paradigm shift towards near-instantaneous transport network services. This exploration aims to guide in a new era of network management tailored for the increasing demands of modern vehicular networks.

References

[1] S. A. Celtek and A. Durdu, "A novel adaptive traffic signal control based on cloud/fog/edge computing," *International Journal of Intelligent Transportation Systems Research*, vol. 20, no. 3, pp. 639–650, 2022.
[2] F. Busacca, C. Grasso, S. Palazzo, and G. Schembra, "A smart roadside unit in a microeolic box to provide edge computing for vehicular applications," *IEEE Transactions on Green Communications and Networking*, 2022.
[3] B. Hu, Y. Shi, and Z. Cao, "Adaptive energy-minimized scheduling of real-time applications in vehicular edge computing," *IEEE Transactions on Industrial Informatics*, 2022.
[4] T. He, A. N. Toosi, N. Akbari, M. T. Islam, and M. A. Cheema, "An intent-based framework for vehicular edge computing," in *2023 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2023, pp. 121–130.
[5] R. Meneguette, R. De Grande, J. Ueyama, G. P. R. Filho, and E. Madeira, "Vehicular edge computing: architecture, resource management, security, and challenges," *ACM Computing Surveys (CSUR)*, vol. 55, no. 1, pp. 1–46, 2021.
[6] L. Zhao, W. Zhao, A. Hawbani, A. Y. Al-Dubai, G. Min, A. Y. Zomaya, and C. Gong, "Novel online sequential learning-based adaptive routing for edge software-defined vehicular networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 5, pp. 2991–3004, 2020.
[7] S. Yu, Y. Guo, N. Li, D. Xue, and C. Liu, "Dynamic resource allocation on vehicular edge computing and communication," in *2022 the 8th International Conference on Communication and Information Processing*, 2022, pp. 205–211.
[8] H. Peng, H. Wu, and X. S. Shen, "Edge intelligence for multi-dimensional resource management in aerial-assisted vehicular networks," *IEEE Wireless Communications*, vol. 28, no. 5, pp. 59–65, 2021.
[9] B. Kar, K.-M. Shieh, Y.-C. Lai, Y.-D. Lin, and H.-W. Ferng, "Qos violation probability minimization in federating vehicular-fogs with cloud and edge systems," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 12, pp. 13 270–13 280, 2021.
[10] M. A. U. Rehman, S. Mastorakis, B.-S. Kim *et al.*, "Foggyedge: An information-centric computation offloading and management framework for edge-based vehicular fog computing," *IEEE Intelligent Transportation Systems Magazine*, 2023.
[11] N. Slamnik-Kriještorac, M. C. Botero, L. Cominardi, S. Latré, and J. M. Marquez-Barja, "An ml-driven framework for edge orchestration in a vehicular nfv mano environment," in *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*. IEEE, 2023, pp. 728–733.
[12] J. Gao, Z. Kuang, J. Gao, and L. Zhao, "Joint offloading scheduling and resource allocation in vehicular edge computing: A two layer solution," *IEEE Transactions on Vehicular Technology*, 2022.
[13] H. Xiao, J. Zhao, Q. Pei, J. Feng, L. Liu, and W. Shi, "Vehicle selection and resource optimization for federated learning in vehicular edge computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 8, pp. 11 073–11 087, 2021.
[14] W. Fan, Y. Su, J. Liu, S. Li, W. Huang, F. Wu, and Y. Liu, "Joint task offloading and resource allocation for vehicular edge computing based on v2i and v2v modes," *IEEE Transactions on Intelligent Transportation Systems*, 2023.
[15] L. Chang, X. Deng, J. Pan, and Y. Zhang, "Edge server placement for vehicular ad hoc networks in metropolitans," *IEEE Internet of Things Journal*, vol. 9, no. 2, pp. 1575–1590, 2021.
[16] J. Wang, K. Zhu, B. Chen, and Z. Han, "Distributed clustering-based cooperative vehicular edge computing for real-time offloading requests," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 1, pp. 653–669, 2021.
[17] R. Mahmud, S. Pallewatta, M. Goudarzi, and R. Buyya, "ifogsim2: An extended ifogsim simulator for mobility, clustering, and microservice management in edge and fog computing environments," *Journal of Systems and Software*, vol. 190, p. 111351, 2022.
[18] M. Goudarzi, H. Wu, M. Palaniswami, and R. Buyya, "An application placement technique for concurrent iot applications in edge and fog computing environments," *IEEE Transactions on Mobile Computing*, vol. 20, no. 4, pp. 1298–1311, 2020.
[19] S. Mittal, D. Garg, R. S. Bali, and G. S. Aujla, "Edge computing based resource supplementation for software defined vehicular networks," in *2022 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2022, pp. 1693–1698.
[20] T. Bahreini, M. Brocanelli, and D. Grosu, "Vecman: A framework for energy-aware resource management in vehicular edge computing systems," *IEEE Transactions on Mobile Computing*, vol. 22, no. 2, pp. 1231–1245, 2023.
[21] M. Goudarzi, M. S. Palaniswami, and R. Buyya, "A distributed deep reinforcement learning technique for application placement in edge and fog computing environments," *IEEE Transactions on Mobile Computing*, vol. 22, no. 5, pp. 2491–2505, 2023.
[22] S. Liao, J. Wu, S. Mumtaz, J. Li, R. Morello, and M. Guizani, "Cognitive balance for fog computing resource in the internet of things: An edge learning approach," *IEEE Transactions on Mobile Computing*, vol. 21, no. 5, pp. 1596–1608, 2020.
[23] S. D. Okegbile, B. T. Maharaj, and A. S. Alfa, "A multi-user tasks offloading scheme for integrated edge-fog-cloud computing environments," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 7, pp. 7487–7502, 2022.
[24] G. Proietti Mattia, M. Magnani, and R. Beraldi, "A latency-levelling load balancing algorithm for fog and edge computing," in *Proceedings of the 25th International ACM Conference on Modeling Analysis and Simulation of Wireless and Mobile Systems*, 2022, pp. 5–14.

[25] W. Duan, X. Gu, M. Wen, Y. Ji, J. Ge, and G. Zhang, "Resource management for intelligent vehicular edge computing networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 9797–9808, 2021.

[26] N. Kumar, T. Dhand, A. Jindal, G. S. Aujla, H. Cao, and L. Yang, "An edge-fog computing framework for the cloud of things in vehicle to grid environment," in *2020 IEEE 21st International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2020, pp. 354–359.

[27] X. Zhu, Y. Luo, A. Liu, N. N. Xiong, M. Dong, and S. Zhang, "A deep reinforcement learning-based resource management game in vehicular edge computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 3, pp. 2422–2433, 2021.

[28] J. S. Ng, W. Y. B. Lim, Z. Xiong, D. Niyato, C. Leung, and C. Miao, "A double auction mechanism for resource allocation in coded vehicular edge computing," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 2, pp. 1832–1845, 2021.

[29] M. Goudarzi, M. Palaniswami, and R. Buyya, "Scheduling iot applications in edge and fog computing environments: a taxonomy and future directions," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–41, 2022.

[30] E. Del-Pozo-Puñal, F. García-Carballeira, and D. Camarmas-Alonso, "A scalable simulator for cloud, fog and edge computing platforms with mobility support," *Future Generation Computer Systems*, vol. 144, pp. 117–130, 2023.

[31] I. Maity, R. Dhiman, and S. Misra, "Mobiplace: Mobility-aware controller placement in software-defined vehicular networks," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 1, pp. 957–966, 2021.

[32] J. Gedeon, M. Stein, J. Krisztinkovics, P. Felka, K. Keller, C. Meurisch, L. Wang, and M. Mühlhäuser, "From cell towers to smart street lamps: Placing cloudlets on existing urban infrastructures," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 187–202.

[33] A. Nahar, H. Sikarwar, and D. Das, "Csbr: A cosine similarity based selective broadcast routing protocol for vehicular ad-hoc networks," in *2020 IFIP networking conference (networking)*. IEEE, 2020, pp. 404–412.

[34] A. Nahar, D. Das, and S. K. Das, "Obqr: orientation-based source qos routing in vanets," in *Proceedings of the 23rd international ACM conference on modeling, analysis and simulation of wireless and mobile systems*, 2020, pp. 199–206.

[35] A. Varga, "Omnet++," in *Modeling and tools for network simulation*. Springer, 2010, pp. 35–59.

[36] M. Haklay and P. Weber, "Openstreetmap: User-generated street maps," *IEEE Pervasive computing*, vol. 7, no. 4, pp. 12–18, 2008.

[37] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of sumo-simulation of urban mobility," *International journal on advances in systems and measurements*, vol. 5, no. 3&4, 2012.

Ankur Nahar is currently pursuing a Ph.D. in Computer Science and Engineering at the Indian Institute of Technology Jodhpur, focusing on Routing Optimization and Fast Message Dissemination in Vehicular Ad-hoc Networks. His scholarly achievements include recognition as a BRICS Young Scientist in 2023. He is interested in vehicular ad-hoc networks, vehicular edge computing, fifth-generation cellular networks, and routing optimization.

Koustav Kumar Mondal is pursuing his Ph.D. under the Inter-Disciplinary Research Division - IoT and Application at Indian Institute of Technology Jodhpur. He has completed his M.Tech in the Internet of Things from the Maulana Abul Kalam Azad University of Technology, and his research interests include AI-empowered mobile Edge Computing, IoT, IoV, Machine Learning, Deep Learning, and Blockchain.

Dr. Debasis Das is an Associate Professor in the Department of Computer Science and Engineering and the School of Artificial Intelligence and Data Science (AIDE) at the Indian Institute of Technology (IIT) Jodhpur, Rajasthan, India. His research interests include vehicular ad hoc networks (VANETs), blockchain, the Internet of Things (IoT), machine learning (ML), and artificial intelligence (AI). He has been awarded the Early Career Research(ECR) Award and BRICS Young Scientist Award from the Science & Engineering Research Board (SERB), Department of Science & Technology, Govt. of India. He is a senior member of IEEE and VTS and a professional member of ACM.

Prof. Rajkumar Buyya is a Redmond Barry Distinguished Professor and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He has authored over 850 publications and seven textbooks, including "Mastering Cloud Computing," published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese, and international markets, respectively. Dr. Buyya is recognized as a "Web of Science Highly Cited Researcher" for six consecutive years since 2016, a Fellow of IEEE, Foreign Fellow of Academia Europaea, Scopus Researcher of the Year 2017 with Excellence in Innovative Research Award by Elsevier and "Lifetime Achievement Awards" from two Indian universities for his outstanding contributions to Cloud computing and distributed systems. He has been recognized as the "Best of the World" twice for research fields (in Computing Systems in 2019 and Software Systems in 2021) as well as "Lifetime Achiever" and "Superstar of Research" in "Engineering and Computer Science" discipline twice (2019 and 2021) by the Australian Research Review.