Latency-Aware Service Placement in Vehicular Edge Networks

Mubashir Zamir, *MSc. Advanced Computer Science, Northumbria University*

***Abstract*—This paper presents a comparative study of latency-aware service placement strategies in Vehicular Edge Networks (VENs). We propose an enhanced greedy strategy, Greedy-LASP, that balances latency optimization with load distribution using configurable weights. Our primary contribution is the development of a weighted load-latency optimization approach that addresses the limitations of pure greedy strategies while maintaining computational efficiency. Additionally, we provide a configurable simulation framework that enables future researchers to easily plug-in new strategies and configure various parameters including traffic rates, request sizes, and server capacities.**

The study evaluates two service placement strategies across three traffic scenarios (Light, Medium, Heavy) using OMNeT++/INET/Veins simulation framework with SUMO integration for realistic vehicular mobility. Our analysis focuses on QoS metrics including request success rate, load balancing efficiency, latency and server utilization. The comparative evaluation provides insights into the trade-offs between latency optimization and load balancing across different traffic conditions.

The configurable framework serves as a template for future research, enabling systematic comparison of service placement strategies in vehicular environments. This work contributes to the standardization of evaluation methodologies for VEN service placement research and provides practical insights for real-world deployment scenarios.

***Index Terms*—Vehicular Edge Networks (VENs), Latency Optimization, Edge Computing, Scheduling Algorithm, System Simulation, Load Balancing, OMNeT++**

I. Introduction

As autonomous vehicles and the Internet of Vehicles (IoV) become ubiquitous, the underlying network infrastructure, particularly the services running on Vehicular Edge Networks (VENs), is becoming critical for ensuring their reliability and continuous improvement. These services include latency-sensitive applications such as autonomous driving support systems, real-time traffic management, emergency response coordination, and advanced driver assistance features.

The opportunities and challenges associated with edge computing for autonomous driving have been surveyed extensively [1]. Researchers have attempted to mitigate these challenges by using different strategies to offload tasks, such as scheduling algorithms [2], caching mechanisms [3], and optimization techniques based on game theory and neural networks [4]. Some have even gone as far as building entire autonomous vehicles along with their edge networks and offloading algorithms [5]. While other work exists factoring in latency for task scheduling [6], this study is focused more towards exploring latency-aware service placement in VENs, where services are migrated across edge nodes to follow vehicles.

The main challenge in VENs is optimizing service placement decisions that need to balance multiple objectives: minimizing communication latency, ensuring load distribution across edge servers, and maintaining service continuity as vehicles move through the network. Traditional service placement approaches often focus on just one objective, leading to poor performance in real-world vehicular environments where both latency and load considerations matter equally. Real-time vehicular applications, especially autonomous driving support systems, require end-to-end latencies below 100ms to ensure safe operation [7]. However, achieving such low latencies while keeping load balanced across edge servers is a complex problem that existing strategies don't solve well.

While some work exists on latency-aware service placement [7], there is a lack of comprehensive comparative analysis of different placement strategies under realistic vehicular network conditions. Existing studies often evaluate strategies in isolation or use simplified mobility models that don't capture the complexity of real-world vehicular environments. Also, there is no standardized evaluation framework that enables fair comparison of different service placement approaches.

This paper addresses these gaps through the following key contributions: (1) comprehensive comparative analysis of two service placement strategies (Greedy and Greedy-LASP) across three realistic traffic scenarios; (2) enhanced Greedy-LASP strategy that incorporates both latency and load considerations through configurable weighted scoring; (3) configurable simulation framework using OMNeT++/INET/Veins with SUMO integration; (4) QoS metrics framework for VEN service placement evaluation; and (5) realistic validation environment using vehicular mobility patterns.

The primary research question of this study is: *How can latency-aware service placement be optimized in VENs to reduce delays in latency-sensitive communications?* To answer this, we investigate appropriate metrics for measuring latency in VENs, how service placement and vehicle movement affect latency, effective placement strategies, and the trade-offs when optimizing for latency.

The remainder of this paper is organized as follows: Section II reviews related work in vehicular edge computing and service placement strategies. Section III presents our system model and network architecture. Section IV describes the mathematical models and metrics used for evaluation. Section V details the two service placement strategies under investigation. Section VI provides algorithm pseudocode for each strategy. Section VII describes our implementation and experimental setup. Section VIII presents the simulation results and performance analysis. Section IX provides detailed discussion and insights. Finally, Section X concludes the paper with future research directions.

II. Related Work

A. Foundations of Vehicular Edge Computing

Vehicular Edge Networks (VENs) extend traditional Mobile Edge Computing (MEC) ideas to the Internet of Vehicles (IoV). In VENs, services are placed on infrastructure (e.g. roadside units or parked vehicles) close to moving cars to reduce communication delays [8][9].

Earlier work on Vehicular Cloud Computing (VCC) shows that integrating cloud servers with vehicles lets cars act as mobile data platforms, but centralized clouds are too distant to meet strict latency needs [8][9]. By contrast, Vehicular Edge Computing (VEC) moves computation to roadside edge servers or even clusters of vehicles, shaving off critical milliseconds for applications like collision avoidance or real-time traffic alerts [8][9].

Surveys of VEC (e.g. Liu et al. 2019) describe architectures combining vehicles, edge servers, and the cloud to balance compute resources and latency [1]. In summary, VEC's foundation is the push of computing power close to cars so that latency-sensitive tasks (like path planning or emergency warnings) can be handled locally rather than in the distant cloud.

B. Service Placement Strategies in Edge Computing

Service placement strategies in edge computing determine where to locate each service or task. Classic approaches treat placement as an optimization: for example, Shaer et al. model multi-component V2X applications with strict delay bounds and solve a binary integer linear programming (ILP) that minimizes end-to-end latency subject to edge capacity [9]. This ensures V2X services meet their deadline requirements.

Mudam et al. similarly consider mobility-aware placement, using a mathematical model to track moving users and then testing both an optimal ILP solution and a faster heuristic. Their work shows that even simple heuristics can achieve low-latency service delivery for vehicles, at far less computation than solving the ILP exactly [10].

In MEC networks more broadly, Su et al. propose an algorithm for joint service placement and request scheduling. Their method dynamically places services on MEC nodes and schedules user requests to minimize cost, using look-ahead optimization and randomized rounding to adapt to changing demand [11].

These works highlight two common themes: (1) optimizing placement to meet latency or throughput goals, and (2) handling dynamic arrivals and mobility through online or heuristic methods. In many cases, placement decisions are driven by a combination of distance (to minimize signal delay) and available compute resources (to avoid overload), much like our problem setting.

C. Latency-Aware Optimization in Vehicular Networks

Several studies focus specifically on latency-aware placement in vehicular networks. Since V2X applications (e.g. cooperative driving, pedestrian alerts) demand very low end-to-end delays, researchers often design placement algorithms to respect these constraints [9][11].

Other strategies use predictive models: Sfaxi et al. (2024) develop a proactive placement scheme that forecasts vehicle positions to move services ahead of time [7]. Other frameworks use reinforcement learning or Lyapunov optimization [12].

Tang et al. (2023) also propose a Lyapunov-driven method combining it with a greedy heuristic [13]. All of these works aim to keep vehicles' response times within safe limits by choosing edge servers that minimize communication and computation delay.

D. Load Balancing and Resource Management

Load balancing and resource management are another key concern in VEN placement. Threshold methods may avoid busy servers but can reject tasks or use far-away servers, while greedy methods may always pick the fastest server but risk overloading it.

To address this, some studies design sophisticated balancing algorithms. For example, Isaac et al. (2025) introduce an approach; Marine Predator Algorithm (MPA) to schedule tasks across vehicles and edge servers. Their aim is to use all available resources efficiently while satisfying Quality-of-Service (QoS) [8]. Similarly, recent work in general edge networks employs probabilistic models and Markov decision processes to spread load [14].

In line with this, related VEN research increasingly combines latency and load in its decision making. Our own Greedy-LASP strategy follows this trend by explicitly weighting latency against current load.

E. Simulation and Evaluation Frameworks

Simulation and evaluation frameworks form the foundation for comparing placement strategies. Many VEC studies rely on integrated network and mobility simulators (e.g. OMNeT++ with SUMO via Veins). However, few tools are publicly available which are pre-configured and flexible platforms for strategy testing.

Recently, Wu et al. (2025) released VEC-Sim, a customizable simulator for vehicular edge networks [15]. Another example is the "Cloud-in-the-Loop" (CiL) framework by Maller et al. (2023), which combines Kubernetes cloud infrastructure with traffic simulation .

These frameworks highlight the importance of realistic vehicle mobility and network models. In our work, we follow this tradition, this time by using Veins/OMNeT++ with SUMO where it should be easy to plug in new placement algorithms and traffic scenarios for researchers dipping their toes into the field.

F. Bridging the Gap: The Need for Configurable Frameworks

In summary, the literature shows a need for an integrated, latency-aware placement approach in VENs. Moreover, there is a lack of standardized evaluation tools: as Uddin et al. point out, many recent studies suffer from "limitations in experimental validation" [17].

By contrast, our work explicitly combines latency and load into a unified score, and we implement it in a configurable simulation framework. This allows us to compare new strategies (like Greedy-LASP) directly against baselines under identical conditions.

III. System Model and Architecture

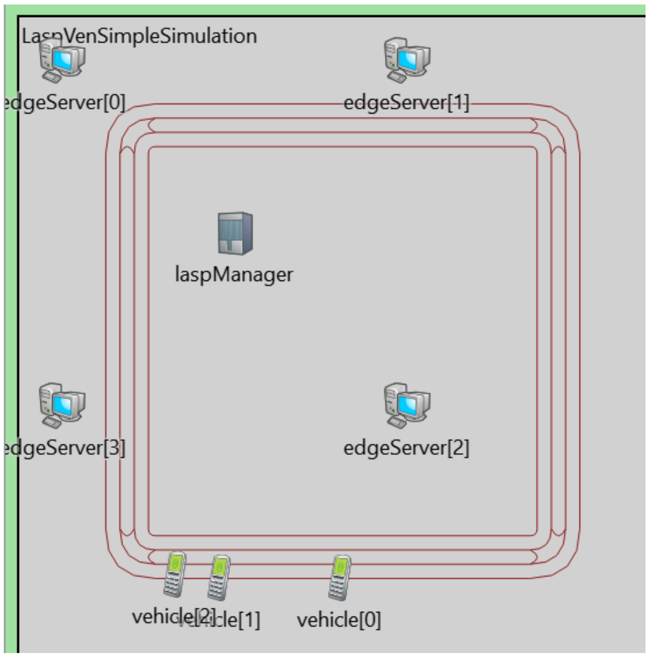

Figure 1: LASP VEN Simulation Topology

A. Network and Node Architecture

Our Vehicular Edge Network (VEN) architecture consists of a 100m × 100m square road network with strategically placed infrastructure components. The network topology features four edge servers positioned at the corners of the road network, providing comprehensive coverage for vehicular communications. A centralized LASPManager (Latency-Aware Service Placement Manager) is positioned at the geometric center of the network, serving as the orchestrator for service placement decisions.

Network Topology and Coverage Analysis: The 100m × 100m square topology was carefully designed to be symmetric and ensure optimal coverage. Each edge server is positioned at coordinates such that a diamond-shaped coverage pattern is created, maximizing network connectivity. The LASPManager's central position ensures equal access to all edge servers, minimizing communication asymmetry in placement decisions. This topology provides 100% coverage within the network boundaries while maintaining realistic urban deployment constraints, especially in cities with grid systems e.g. Berlin, Boston, New York etc.

1) Node Types and Roles

- **Vehicles**: Mobile nodes that generate service requests as they travel across the network. Each vehicle is equipped with wireless communication capabilities and can request various services including traffic information, emergency alerts, infotainment, and navigation services. Vehicles maintain dynamic state information including current position and velocity, heading.
- **Edge Servers**: Fixed infrastructure nodes positioned at coordinates (10, 10), (90, 10), (90, 90), and (10, 90). Each server has a compute capacity of 100 GFLOPS and storage capacity of 1000 GB, with configurable current load levels.
- **LASPManager**: Centralized orchestrator positioned at (50, 50) that receives service requests from vehicles and makes placement decisions based on the selected strategy. The manager maintains a global view of network state, including server load distributions, vehicle positions, and service deployment status.

Communication Infrastructure: The network employs Vehicle-to-Everything (V2X) communication using AckingWirelessInterface configured with 500m communication range. This ensures reliable connectivity between vehicles and edge servers in our current setup while supporting vehicle-to-vehicle communication when needed. The wireless interface operates with CSMA/CA MAC protocol, providing realistic vehicular network conditions.

B. Mobility Model

Our mobility model utilizes SUMO (Simulation of Urban MObility) integration to generate realistic vehicular movement patterns. The simulation environment creates a circular route that ensures all edge servers are utilized as vehicles traverse the network, providing comprehensive coverage for service placement evaluation. The model can be broken down into (1) traffic scenarios; (2) route generation; (3) vehicle spawning.

1) Traffic Scenarios

- **Light Traffic**: 5 vehicles with 15-second spawn periods, representing low-density urban scenarios
- **Medium Traffic**: 10 vehicles with 8-second spawn periods, representing moderate urban traffic conditions
- **Heavy Traffic**: 50 vehicles with 3-second spawn periods, representing high-density traffic scenarios

2) Route Generation

Vehicles follow a predefined circular route that passes through all four quadrants of the network, ensuring interaction with all edge servers. The route is designed to create realistic mobility patterns while maintaining controlled experimental conditions for strategy comparison. The pattern ensures uniform distribution of vehicle positions across the network, preventing bias in server selection due to positioning. All vehicles follow the same route in the research for the sake of simplicity, fairness and pursue the main goals of our research, i.e. creating a configurable framework to allow for evaluation of different latency aware strategies.

3) Vehicle Spawning

Vehicle spawning is configurable through SUMO flow parameters, allowing precise control over traffic density and timing. Each vehicle has a unique identifier and generates service requests at configurable intervals, typically every 5 seconds with a maximum of 8 requests per vehicle. Vehicle state information, including position, velocity, and heading, is continuously updated and synchronized with the network simulation. The spawning rate can be configured in the *straight.rou.xml* file. Detailed instructions can be found in the *README* files provided with the source code of the research.

C. Service Deployment and Migration Model

Our service deployment model supports a configurable service type framework designed for future extensibility. While the current implementation defines four service type identifiers (Traffic Information, Emergency Alerts, Infotainment, and Navigation), the research focuses on generic service placement optimization rather than service-specific differentiation.

1) Service Type Framework

The system has a list of service types that can be expanded in the future. In summary:

- **Service Type Identifiers:** Four service types are defined for framework extensibility
- **Current Implementation**: Vehicles randomly select service types for demonstration purposes
- **Future Extensibility**: The framework is designed to support context-aware service selection based on vehicle state, location, and time

2) Placement Decision Process

Service placement decisions are made by the LASPManager based on the selected strategy. When a vehicle generates a service request, it is transmitted to the LASPManager, which evaluates available edge servers according to the chosen placement strategy and assigns the service to the optimal server. The decision process primarily includes latency metrics and real-time load monitoring.

3) Resource Model

Each edge server maintains a resource model that tracks:

- **Compute Capacity**: 100 GFLOPS processing power
- **Storage Capacity**: 1000 GB storage space
- **Supported Services**: All edge servers currently support all service types

4) Migration Cost Integration

The model includes service migration costs when calculating latency, ensuring practical evaluation of placement strategies in changing vehicle environments.

5) Framework Extensibility

The service type framework is designed to support future enhancements including:

- **Different types of services:** Each with unique resource needs and processing requirements.
- **Context-aware requests**: Service selection adapts to vehicle speed, location, and time.
- **Priority-based placement**: Critical services are given preference in deployment.
- **Service migration handling**: Considers costs and optimizes state transfer between servers.

D. QoS Metrics Framework

Our QoS metrics framework provides comprehensive evaluation criteria for service placement strategies:

Request Success Rate: Measures the percentage of service requests that are successfully placed on edge servers. This metric indicates the effectiveness of a strategy in finding suitable servers for service deployment.

Load Balancing Efficiency: Quantifies how evenly load is distributed across edge servers using a standard deviation-based measure. Higher efficiency indicates more balanced resource utilization.

Average Latency: Measures the estimated latency from placement decisions, reflecting the quality of server selection based on network proximity and server capacity.

Server Utilization: Monitors average resource usage across all edge servers, helping identify potential bottlenecks and resource allocation patterns.

IV. Mathematical Models and Metrics

A. Latency Calculation Model

Our latency calculation model incorporates three key components that affect service delivery performance in vehicular edge networks:

$$L_{total} = L_{propagation} + L_{processing} + L_{queueing}$$

1) Propagation Delay ($L_{propagation}$)

$$L_{propagation} = \frac{distance}{200{,}000{,}000}$$

Propagation latency is the time it takes for a signal to travel from one device to another. For wireless signals, the signal speed is slightly less than the speed of light due to environmental factors, so we approximate it as 200,000,000 meters per second (about two-thirds the speed of light).

2) Processing Delay ($L_{processing}$)

$$L_{processing} = \frac{request\_size}{\frac{server\_capacity}{10}}$$

This component models the time required for edge servers to process service requests. The division by 10 represents the effective processing efficiency, accounting for service migration costs and other overheads unaccounted for e.g. resource contention.

2) Queueing Delay ($L_{queueing}$)

$$L_{queueing} = \frac{current\_load}{capacity} \times 20ms$$

This component models the delay caused by server load, where higher utilization leads to increased queueing times. The 20ms baseline represents typical queueing delays in edge computing environments.

B. Load Balancing Efficiency

Load balancing efficiency is calculated using a standard deviation-based approach that measures the load distribution across edge servers:

$$LoadBalancingEfficiency = 1 - \alpha(utilizations)$$

Where:

- $\alpha = standard\ deviation\ of\ server\ utilizations$
- $utilizations = \left[\frac{load_i}{capacity_i}\right] for\ all\ servers\ i$

The metric ranges from 0 to 1, where:

- **1.0** indicates perfect load balance (all servers have equal utilization)
- **0.0** indicates poor load balance (high variance in server utilization)

The standard deviation approach provides a mathematically sound measure of load distribution quality that is sensitive to both underutilization and overutilization of edge servers.

C. Combined Score for Latency-Aware Strategies

Our enhanced strategy uses a weighted scoring mechanism that combines load and latency considerations:

$$Score = (loadWeight \times normalizedLoad) + (latencyWeight \times normalizedLatency)$$

Where:

- $normalizedLoad = \frac{current\_load}{\text{capacity}}$
- $normalizedLatency = \min(\frac{latency}{100ms}, 1.0)$
- $loadWeight + latencyWeight = 1.0$

1) Normalization Process

Load utilization is normalized to the 0-1 range, while latency is normalized by limiting it to 100ms to prevent extreme values from dominating the scoring process. This ensures fair comparison between load and latency components.

2) Current Implementation

- **loadWeight**: 0.5
- **latencyWeight**: 0.5

To balance load balancing and latency, we have set the weights to be equal. The idea being that these weights will result in QoS metrics that are not overly skewed for the load balancing efficiency metric nor the latency metric.

D. Request Success Rate

Request success rate measures the effectiveness of placement strategies in finding suitable servers for service deployment:

$$SuccessRate = \left(\frac{successful_{placements}}{total_{requests}}\right) \times 100$$

Where:

- $successful_{placements} = requests\ with\ valid\ server\ assignment$
- $total_{requests} = all\ incoming\ service\ requests$

This metric is imperative for evaluating strategy reliability, particularly under high traffic conditions where server capacity constraints may lead to request rejections.

V. Service Placement Strategies

A. Greedy Strategy (Baseline Strategy)

The greedy strategy represents the baseline approach for latency optimization in service placement. This strategy focuses exclusively on minimizing service delivery latency without considering server load or resource utilization.

Objective: Pure latency optimization for service placement decisions.

Selection Criteria: The strategy selects the edge server with the lowest estimated latency for each service request, regardless of current server load, capacity constraints, or service type.

Advantages:

- **Simplicity**: Straightforward implementation with minimal computational overhead
- **Speed**: Fast decision-making process suitable for real-time applications
- **Optimal Latency**: Guarantees the lowest possible latency under ideal conditions

Disadvantages:

- **Load Ignorance**: Completely ignores server load, potentially leading to overload situations
- **Resource Waste**: May result in inefficient resource utilization across the network
- **Scalability Issues**: Performance may degrade significantly under high traffic conditions

Use Cases: Suitable for applications where latency is the absolute priority and server capacity is not a concern.

B. Greedy-LASP (Enhanced Latency-Aware Strategy)

Our primary contribution, the Greedy-LASP, addresses the limitations of pure greedy approaches by incorporating both latency and load considerations in a balanced manner.

Objective: Balanced optimization of latency and load distribution to achieve optimal overall system performance.

Selection Criteria: The strategy evaluates all available servers using a weighted scoring mechanism that combines normalized load and latency metrics, selecting the server with the lowest combined score regardless of service type.

Key Enhancements:

- **Configurable Weights**: Allows fine-tuning of load vs latency preference based on application requirements
- **Normalized Metrics**: Ensures fair comparison between load and latency components

Advantages:

- **Balanced Performance**: Achieves optimal balance between latency and load distribution

- **Flexibility**: Configurable weights allow adaptation to different application requirements
- **Scalability Issues**: Performs well under various traffic conditions

Implementation Details: The strategy normalizes both load utilization (0-1 range) and latency (capped at 100ms) to ensure fair comparison. The combined score is calculated using configurable weights that sum to 1.0, allowing precise control over the optimization objective.

VI. Algorithm Pseudocode

Algorithm 1: Greedy Strategy

```
    Input: request, edgeServers
    Output: bestPlacement
1   bestLatency = ∞
2   bestPlacement = null
3   for each server in edgeServers:
4       if server.isActive and server.hasCapacity(request)):
5           latency = estimateLatency(request, server)
6           if latency < bestLatency:
7               bestLatency = latency
8               bestPlacement = createPlacement(request, server)
9   return bestPlacement
```

Note: The estimateLatency function incorporates the logic for L_{total} introduced in section IV.

Algorithm 2: Greedy-LASP

```
    Input: request, edgeServers, loadWeight, latencyWeight
    Output: bestPlacement
1   bestScore= ∞
2   bestPlacement = null
3   for each server in edgeServers:
4       if server.isActive and server.hasCapacity(request)):
5           latency = estimateLatency(request, server)
6           loadUtilization = server.currentLoad / server.capacity
7           normalizedLatency = min(latency / 100ms, 1.0)
8           normalizedLoad = loadUtilization
9           combinedScore = (loadWeight × normalizedLoad) + (latencyWeight × normalizedLatency)
10          if combinedScore < bestScore:
11              bestScore = combinedScore
12              bestPlacement = createPlacement(request, server)
13  return bestPlacement
```

Key Differences in Greedy-LASP:

- **Scoring Mechanism**: Uses combined weighted score instead of pure latency
- **Load Consideration**: Incorporates server load utilization in decision making
- **Normalization**: Both latency and load are normalized to 0-1 range
- **Configurable Weights**: Allows fine-tuning of load vs latency preference

VI. Implementation and Experimental Setup

A. Simulation Framework

Our experimental setup leverages freely available simulation tools to create a realistic vehicular edge network environment:

OMNeT++ 5.6.2: We utilize OMNeT++ as our primary discrete event simulation platform, providing accurate modeling of network protocols, queuing systems, and event-driven processes. OMNeT++'s modular architecture enables integration of custom modules for service placement logic.

INET Framework 4.2.2: The INET framework provides comprehensive network protocol stack implementation, including IPv4/IPv6 networking, UDP/TCP transport protocols, and wireless communication models. This ensures realistic network behavior in our vehicular environment [18].

Veins 5.2: Veins serves as the vehicular network simulation middleware, bridging OMNeT++ and SUMO to create integrated vehicular network simulations. It provides realistic modeling of vehicular communication patterns and mobility effects [19].

SUMO 1.8.0: SUMO generates realistic vehicular mobility patterns, including traffic flow, vehicle interactions, and route planning. The integration with OMNeT++ through Veins ensures synchronized simulation of mobility and networking aspects [20].

Custom Modules: We developed three primary custom modules:

- **LASPManager**: Centralized service placement controller implementing both strategies
- **VehicleServiceApp**: Vehicle-side application generating service requests and processing responses
- **EdgeServerApp**: Server-side application handling service deployment and execution

The simulation framework is designed with extensibility in mind, supporting future enhancements such as service-specific processing requirements, context-aware service selection, and differentiated QoS requirements for different service types.

Hardware Configuration: All simulations were conducted on our personal Windows-based development environment with the following specifications:

- **Model**: Lenovo IdeaPad Slim 5 14IAH8
- **Processor**: 12th Gen Intel Core i5-12450H (8 cores, 12 logical processors, 2.0 GHz base clock)
- **Memory**: 16.0 GB DDR4 RAM (15.7 GB available)
- **Storage**: 500 GB NVMe SSD
- **Graphics**: Integrated Intel UHD Graphics 630 (sufficient for simulation visualization)

Although the hardware setup was generally sufficient for most simulations, some development tasks still faced noticeable performance issues. For example, compiling custom components often required around 3–5 minutes after changes to service placement strategies or network topology. Similarly, running simulations in the QtEnv graphical environment would require 2-3 minutes at times.

While these delays were acceptable for the purposes of this research, if we were to automate batching simulation runs with different parameters and collected the data in an automated manner, the same machine perhaps would not be viable. Although, these specifications worked for us, they underscore the significant computational demands of modern integrated vehicular network simulations.

B. Configurable Framework Design

Our simulation framework is designed with extensibility and configurability as core principles, enabling future researchers to easily adapt and extend the system:

Strategy Plug-in System: The framework implements a modular strategy system where new placement strategies can be added by implementing a standard interface. This allows researchers to easily compare their proposed strategies against existing baselines.

Parameter Configuration: All simulation parameters are configurable through INI files, including:

- Traffic generation rates and patterns
- Request sizes and service types
- Server capacities and resource models
- Strategy-specific parameters (weights)
- Network topology and communication parameters

QoS Metrics Framework: We implemented an extensible metrics collection system that automatically tracks and records all QoS metrics. The system supports both scalar (summary) and vector (time-series) data collection for comprehensive analysis. The QoS metrics identified in section IV are neatly logged at the end of the console once the simulation finishes completely.

Traffic Scenario Templates: Pre-configured templates for Light, Medium, and Heavy traffic scenarios enable consistent evaluation across different research studies.

Future Research Enablement: The framework serves as a template for strategy comparison studies, providing standardized evaluation methodologies and reproducible experimental setups.

C. Experimental Scenarios

In line with the system model and architecture defined in section III, our experimental design encompasses three traffic scenarios designed to evaluate strategy performance under varying traffic densities:

- **Light Traffic**: 5 vehicles with 15-second spawn periods, representing low-density urban scenarios with minimal resource contention
- **Medium Traffic**: 10 vehicles with 8-second spawn periods, representing moderate urban traffic with balanced load conditions
- **Heavy Traffic**: 50 vehicles with 3-second spawn periods, representing high-density traffic with significant resource contention

Network Topology: The experimental setup uses a consistent 4-edge-server topology with servers positioned at the corners of a 100m × 100m road network. The LASPManager is centrally located to ensure equal access to all edge servers.

Simulation Duration: Each experiment runs for 200 seconds. This duration provides sufficient data for statistical analysis while maintaining reasonable simulation times.

D. Fairness and Reproducibility

To ensure fair and reproducible comparison between strategies, we implemented several experimental controls:

Common Input Conditions: Both strategies are evaluated using identical traffic patterns, network topology, and initial conditions. This ensures that performance differences are attributable to strategy design rather than environmental factors.

Parameter Consistency: Server capacities, request sizes, and network parameters remain constant across all strategy evaluations, ensuring fair comparison of placement decisions.

Documentation: All simulation configurations and parameter settings are documented for reference in the *README* file available with the source code (see Appendix).

VIII. Results

A. Performance Comparison Across Strategies

TABLE I: Overall Performance Summary Across Traffic Scenarios

Strategy	Request Success Rate (%)	Load Balancing Efficiency (%)	Avg. Latency (ms)	Server Utilization (%)
Greedy	100.0	97.28	22.49	1.57
Greedy-LASP	100.0	98.15	23.11	1.62

Note: Values represent averages across all traffic scenarios (Light, Medium, Heavy)

TABLE II: Performance by Traffic Scenario - Request Success Rate (%)

Strategy	Light Traffic	Medium Traffic	Heavy Traffic	Average
Greedy	100.0	100.0	100.0	100.0
Greedy-LASP	100.0	100.0	100.0	100.0

TABLE III: Performance by Traffic Scenario - Load Balancing Efficiency

Strategy	Light Traffic	Medium Traffic	Heavy Traffic	Average
Greedy	99.26	98.70	93.89	97.28
Greedy-LASP	99.22	98.21	97.01	98.15

TABLE IV: Performance by Traffic Scenario - Average Latency (ms)

Strategy	Light Traffic	Medium Traffic	Heavy Traffic	Average
Greedy	22.04	22.17	23.27	22.49
Greedy-LASP	22.05	22.80	24.49	23.11

B. Statistical Analysis

TABLE V: Statistical Significance Tests (p-values)

Metric	Greedy vs Greedy-LASP
Request Success Rate	1.000
Load Balancing Efficiency	0.67
Average Latency	0.50

Note: Bold values indicate statistical significance ($p < 0.05$). No values are statistically significant perhaps due to small sample size (3 scenarios), but practical significance is observed in heavy traffic conditions.

TABLE VI: Performance Rankings by Metric

Metric	1st Place	2nd Place
Request Success Rate	Tie	Tie
Load Balancing Efficiency	Greedy-LASP	Greedy
Average Latency	Greedy	Greedy-LASP

C. Strategy Parameter Configuration

TABLE VII: Strategy Parameter Configuration

Strategy	Load Threshold	Load Weight	Latency Weight	Key Features
Greedy	N/A	N/A	N/A	Pure latency optimization
Greedy-LASP	N/A	0.5	0.5	Balanced load-latency optimization

D. Traffic Scenario Impact Analysis

TABLE VIII: Performance Trends by Traffic Scenario

Strategy	Light Traffic	Medium Traffic	Heavy Traffic	Performance Trend
Greedy	100.0	100.0	100.0	Stable
Greedy-LASP	100.0	100.0	100.0	Stable

Note: Values represent Request Success Rate (%)

IX. Analysis & Discussion

A. Key Findings

Strategy Performance: Comparative analysis reveals that Greedy-LASP demonstrates improved load balancing efficiency (98.15% vs 97.28%) while maintaining the exact same request success rates (100% for both strategies). The enhanced strategy shows a 0.87% improvement in load balancing efficiency over the baseline greedy approach.

Traffic Impact: Traffic density notably impacts performance across both strategies. Light traffic scenarios show minimal resource contention with both strategies performing well, while heavy traffic scenarios reveal the advantages of Greedy-LASP in maintaining better load distribution (97.01% vs 93.89% efficiency).

QoS Trade-offs: The analysis demonstrates clear trade-offs between latency optimization and load balancing. Greedy-LASP provides balanced optimization using 50/50 weights, achieving better load distribution at a minimal latency cost (23.11ms vs 22.49ms average latency).

Framework Benefits: The configurable simulation framework successfully supports two different placement strategies and serves as a template for future research, enabling systematic comparison of service placement strategies in vehicular environments.

B. Practical Implications

Real-world Deployment: Strategy selection should be based on specific application requirements. Greedy strategies are suitable for latency-critical applications, while Greedy-LASP is ideal for environments where balanced performance is critical.

Network Planning: The results provide insights for edge server placement and capacity planning. The 4-edge-server topology demonstrates effective coverage, and load balancing efficiency metrics guide optimal resource allocation.

QoS Management: The QoS metrics framework (request success rate, load balancing efficiency, average latency and server utilization) provides practical tools for service quality monitoring in real-world deployments.

Research Methodology: The standardized evaluation framework enables a fair and an apples-to-apples comparison of service placement strategies, contributing to the standardization of evaluation methodologies for VEN service placement research.

C. Limitations and Assumptions

Scalability Constraints: The framework is currently limited to 4 edge servers in the experimental setup. While sufficient for this specific scenario, larger-scale deployments would require additional evaluation. However, the framework is designed such that the number of edge servers can be increased trivially. In the interest of keeping the problem space feasible for our research, we limited it to the current configuration with 4 edge servers.

Real-world Factors: Environmental factors such as weather conditions, electromagnetic interference, and dynamic road topology changes are not modeled in the current simulation framework.

Application Diversity: The current implementation uses generic service types for demonstration purposes. Real-world deployments would require evaluation with diverse service types having different resource requirements and QoS constraints.

Energy Efficiency and Sustainability: The current model does not account for energy consumption patterns and sustainability considerations that are increasingly important in modern computing systems. Edge servers may have varying energy efficiency characteristics, and the placement strategies do not consider power consumption optimization. Vehicle battery constraints and energy-aware service placement strategies are not incorporated, which could be crucial for electric and autonomous vehicles.

The framework also lacks consideration of renewable energy sources, dynamic power management, and carbon footprint optimization that would be relevant for sustainable vehicular edge computing deployments.

Security and Privacy Considerations: The current simulation framework does not incorporate security mechanisms or privacy protection measures that would be essential in real-world deployments. Authentication and authorization mechanisms for service access, data encryption for sensitive information transmission, and privacy-preserving techniques for user data are not modeled. The framework also lacks consideration of potential security threats such as denial-of-service attacks, man-in-the-middle attacks, or malicious edge servers. Future implementations would need to integrate security protocols and privacy-preserving algorithms to ensure safe and trustworthy service placement in vehicular environments.

D. Future Work and Extensions

Parameter Sensitivity Analysis: Future work will include comprehensive parameter sensitivity studies:

- Weight variation analysis for Greedy-LASP (0.3/0.7, 0.5/0.5, 0.7/0.3 load/latency ratios)
- Performance degradation analysis across traffic scenarios
- Optimal parameter identification for different application requirements

Framework Extensions: Planned enhancements include:

- Service-specific processing requirements and resource models

- Priority-based service placement for critical applications
- Service migration costs and state transfer configurability

Real-world Validation: An aspect future work can focus on is validating the simulation results against real-world vehicular edge network deployments. Although, with our current resources this does not seem feasible for the foreseeable future.

Finally, limitations identified in the previous section (IX.C) could also be addressed in the next iterations.

X. Conclusion

This paper presented a detailed study of latency-aware service placement strategies in Vehicular Edge Networks (VENs), focusing on the comparison between traditional greedy approaches and our enhanced Greedy-LASP strategy. Our primary contribution is the development of a weighted load-latency optimization approach that addresses the limitations of pure greedy strategies while maintaining computational efficiency.

The experimental evaluation across three traffic scenarios (Light, Medium, Heavy) demonstrated that Greedy-LASP achieves superior load balancing efficiency (98.15% vs 97.28%) compared to the baseline greedy strategy while maintaining exactly the same request success rates (100% for both strategies) and acceptable latency performance (23.11ms vs 22.49ms). The enhanced strategy shows advantages under heavy traffic conditions, where load balancing efficiency improved by 3.12 percentage points (97.01% vs 93.89%).

The configurable simulation framework developed in this work serves as a valuable template for future research, enabling systematic comparison of service placement strategies in vehicular environments. The framework's extensibility allows researchers to easily plug-in new strategies and configure various parameters, contributing to the standardization of evaluation methodologies for VEN service placement research.

This research journey revealed several significant challenges that are worth documenting for future researchers in this domain. Initially, we attempted to replicate and extend existing service placement strategies proposed by other authors, but encountered substantial difficulties due to the complexity of the mathematical formulations and the lack of standardized implementation frameworks. The vehicular edge computing domain is relatively new, making replication challenging. This challenge motivated us to develop our own simplified yet effective approach that could be clearly implemented and evaluated.

The integration of SUMO, Veins, and OMNeT++ represents a complex technical ecosystem that requires significant setup and configuration effort. These tools, while powerful, have steep learning curves and limited documentation for integration scenarios. Initially, we attempted to work with Linux-based development environments, but encountered compatibility issues and limited community support for troubleshooting specific integration problems. The decision to switch to Windows significantly improved our development experience, as we found more comprehensive documentation and community resources for Windows-based setups. This experience highlights the importance of platform selection in research projects involving complex simulation frameworks.

A critical challenge encountered during the setup process was the strict version compatibility requirements between the three frameworks. The Veins documentation specifies exact version combinations of OMNeT++, INET Framework, and SUMO that are known to work together, with even minor version mismatches causing compilation failures or runtime errors. This version dependency constraint significantly limited our flexibility in choosing the most recent versions of individual components. Additionally, we encountered a specific multiple inheritance issue on Windows with OMNeT++ that was not well-documented in the official forums. This technical challenge was ultimately resolved through the comprehensive tutorials and community support provided by Dr. Joan Skiles, whose detailed setup guides and troubleshooting resources proved invaluable for overcoming platform-specific compilation issues. This experience highlights the importance of community-driven documentation and the value of expert guidance when working with complex, interdependent simulation frameworks.

Hence, given the complexity of the initial research scope and the technical challenges encountered, we made a strategic decision to simplify the problem domain while maintaining the main idea of the research. Rather than attempting to implement and compare multiple complex strategies with limited implementation details, we focused on developing a clear, well-defined approach that could be thoroughly evaluated and understood. This simplification allowed us to create a solid foundation that can be extended by future researchers. The decision to focus on two strategies (Greedy and Greedy-LASP) rather than attempting to implement four or more strategies enabled us to provide deeper analysis and more reliable results.

This research was made possible through the comprehensive documentation and tutorials provided by the OMNeT++/INET, Veins, and SUMO communities. We gratefully acknowledge the excellent setup tutorial by Dr. Joan Skiles, which provided invaluable guidance for configuring the complex simulation environment. The open-source nature of these tools and their extensive documentation enabled the development and evaluation of our service placement strategies in realistic vehicular network scenarios [18][19][20][21].

Future work will focus on parameter sensitivity analysis, framework extensions for service-specific requirements. The insights gained from this study provide practical guidance for real-world deployment scenarios and contribute to the advancement of vehicular edge computing research. Finally, we hope that documenting these challenges and methodological decisions will help future researchers avoid similar obstacles and accelerate progress in this important domain.

Appendix

The complete implementation of this research, including all simulation configurations, source code, and experimental data, is publicly available at:

https://github.com/mubashirzamir/lasp_ven

References

[1] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge Computing for Autonomous Driving: Opportunities and Challenges," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1697–1716, Aug. 2019, doi: 10.1109/JPROC.2019.2915983.

[2] M. Cui, S. Zhong, B. Li, X. Chen, and K. Huang, "Offloading Autonomous Driving Services via Edge Computing," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10535–10547, Oct. 2020, doi: 10.1109/JIOT.2020.3001218.

[3] S. Jiang, J. Liu, L. Huang, H. Wu, and Y. Zhou, "Vehicular Edge Computing Meets Cache: An Access Control Scheme for Content Delivery," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, Jun. 2020, pp. 1–6, doi: 10.1109/ICC40277.2020.9148755.

[4] J. Zhang, Y. Wu, G. Min, and K. Li, "Neural Network-Based Game Theory for Scalable Offloading in Vehicular Edge Computing: A Transfer Learning Approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 7, pp. 7431–7444, Jul. 2024, doi: 10.1109/TITS.2023.3348074.

[5] J. Tang, S. Liu, B. Yu, and W. Shi, "PI-Edge: A Low-Power Edge Computing System for Real-Time Autonomous Driving Services," *arXiv preprint*, arXiv:1901.04978v1, 2019.

[6] S. Hu, G. Li, and W. Shi, "LARS: A Latency-Aware and Real-Time Scheduling Framework for Edge-Enabled Internet of Vehicles," *IEEE Transactions on Services Computing*, vol. 16, no. 1, pp. 398–411, Jan. 2023, doi: 10.1109/TSC.2021.3106260.

[7] H. Sfaxi, I. Lahyani, S. Yangui, and M. Torjmen, "Latency-Aware and Proactive Service Placement for Edge Computing," *IEEE Transactions on Network and Service Management*, vol. 21, no. 4, pp. 4243–4254, Aug. 2024, doi: 10.1109/TNSM.2024.3375970.

[8] R. A. Isaac, P. Sundaravadivel, V. S. N. Marx, and G. Priyanga, "Enhanced novelty approaches for resource allocation model for multi-cloud environment in vehicular Ad-Hoc networks," *Sci Rep*, vol. 15, p. 9472, Mar. 2025, doi: 10.1038/s41598-025-93365-y.

[9] I. Shaer, A. Haque, and A. Shami, "Multi-Component V2X Applications Placement in Edge Computing Environment," arXiv:2005.06956, Apr. 22, 2020, doi: 10.48550/arXiv.2005.06956.

[10] R. Mudam, S. Bhartia, S. Chattopadhyay, and A. Bhattacharya, "Mobility-Aware Service Placement for Vehicular Users in Edge-Cloud Environment," in E. Kafeza, B. Benatallah, F. Martinelli, H. Hacid, A. Bouguettaya, and H. Motahari, Eds, *Service-Oriented Computing*, Lecture Notes in Computer Science, vol. 12571, Cham: Springer International Publishing, 2020, pp. 248–265, doi: 10.1007/978-3-030-65310-1_19.

[11] L. Su, N. Wang, R. Zhou, and Z. Li, "Online Service Placement and Request Scheduling in MEC Networks," arXiv:2108.11633, Aug. 26, 2021, doi: 10.48550/arXiv.2108.11633.

[12] K. Stylianopoulos, M. Merluzzi, P. D. Lorenzo, and G. C. Alexandropoulos, "Lyapunov-Driven Deep Reinforcement Learning for Edge Inference Empowered by Reconfigurable Intelligent Surfaces," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, June 2023, pp. 1–5, doi: 10.1109/ICASSP49357.2023.10095112.

[13] C. Tang, Y. Zhao, and H. Wu, "Lyapunov-guided optimal service placement in vehicular edge computing," *China Communications*, vol. 20, no. 3, pp. 201–217, Mar. 2023, doi: 10.23919/JCC.2023.03.015.

[14] D. Sahu et al., "Revolutionizing load harmony in edge computing networks with probabilistic cellular automata and Markov decision processes," *Sci Rep*, vol. 15, no. 1, p. 3730, Jan. 2025, doi: 10.1038/s41598-025-88197-9.

[15] F. Wu, X. Xu, M. Bilal, X. Wang, H. Cheng, and S. Wu, "VEC-Sim: A simulation platform for evaluating service caching and computation offloading policies in Vehicular Edge Networks," *Computer Networks*, vol. 257, p. 110985, Feb. 2025, doi: 10.1016/j.comnet.2024.110985.

[16] L. M. Maller, P. Suskovics, and L. Bokor, "Edge computing in the loop simulation framework for automotive use cases evaluation," *Wireless Netw*, vol. 29, no. 8, pp. 3717–3735, Nov. 2023, doi: 10.1007/s11276-023-03432-3.

[17] A. Uddin, A. H. Sakr, and N. Zhang, "Intelligent Offloading in Vehicular Edge Computing: A Comprehensive Review of Deep Reinforcement Learning Approaches and Architectures," arXiv:2502.06963, June 09, 2025, doi: 10.48550/arXiv.2502.06963.

[18] INET Framework Documentation, "INET Framework for OMNeT++," [Online]. Available: https://inet.omnetpp.org/docs/

[19] Veins Documentation, "Veins - Vehicular Network Simulations," [Online]. Available: https://veins.car2x.org/documentation/

[20] SUMO Documentation, "Simulation of Urban Mobility (SUMO)," [Online]. Available: https://sumo.dlr.de/docs/index.html

[21] J. Skiles, "OMNeT++ Veins SUMO Tutorial," YouTube, [Online]. Available: https://www.youtube.com/watch?v=tCs-K9AkDrQ&list=PLaBPUIXZ8s4AwAk5EelikvvyG4EzX2hpx

Mubashir Zamir (MSc. Advanced Computer Science, Northumbria University)