Interdependent Microservice Offloading and Semantic Aware Results Transmission for 6G Vehicular Edge Networks

Muhammad Salah ud din, Muhammad Nadeem Ali, Ghulam Musa Raza, Muhammad Imran and Byung-Seo Kim, *Senior Member, IEEE*

Abstract—Microservices-centric in-network computations, coupled with Named Data Networking (NDN) and supported by vehicular edge computing (VEC), offer a promising platform to meet the requirements of vehicular applications. However, a significant obstacle hindering high efficiency is the interdependency of microservices (MS), which may delay the timely delivery of results due to Pending Interest Table (PIT) timer expiration, resulting in unsolicited packet drops. Moreover, voluminous data transmission in a resource-constrained environment may prevent the consumer from receiving results on time. Therefore, to avoid unsolicited computation losses and enable proximate computations, this article envisions interdependent microservices offloading and semantic aware results transmission (iMSoRT) for 6G vehicular edge networks. iMSoRT formulates an efficient strategy that allows traffic controller (T_c) to account for MS interdependencies and allocate the PIT timer based on computational and network resource requirements. Moreover, iMSoRT introduces a live forwarding information base (lFIB) that effectively filters out underutilized RSU-E servers and forwards computation requests to the optimal RSU-E, accelerating processing while minimizing communication costs. Furthermore, iMSoRT develops a Semantic Transformer (ST) that leverages YOLO and Convolutional Neural Networks (CNNs), placing it between the caching and forwarding modules of NDN. The ST extracts and forwards semantically meaningful information, thus reducing network resource utilization and enhancing information exchange efficiency. Simulation results revealed that iMSoRT achieved an impressive compute-hit ratio of over 90%, restrict cloud offloading to 12%, reduced computation delays over 50%, and optimized bandwidth utilization over threefold compared with benchmark schemes.

Index Terms—Named Data Networking, Vehicular Networks, Edge, Fog, Cloud, Internet of Things, Microservices.

1 Introduction

The evolution of sixth-generation (6G) communication technology, coupled with advancements in vehicular onboard resources [1]—including communication, computation, and storage—has greatly accelerated the development of autonomous vehicular applications such as augmented/virtual reality (AR/VR), driver assistance, object detection, and assisted lane changing [2]. These applications demand substantial computation and communication resources, often exceeding the capabilities of individual vehicles [3], [4]. However, due to high transmission delays, conventional cloud offloading is unlikely to satisfy their stringent computation deadlines and quality of service (QoS) requirements.

Vehicular Edge Computing (VEC) [5] has emerged as a promising paradigm to address the limitations of cloud computing. By deploying resource-rich edge servers alongside roadside units (RSU-E) in close proximity to end users, VEC reduces computation and communication delays, enabling applications to receive results within strict latency budgets [6]–[8]. Although RSU-E servers offer substantial computing power, their capabilities remain several orders of magnitude lower than those of cloud servers. During rush hours, a surge of compute-intensive, delay-sensitive application requests can overwhelm an RSU-E's processing capacity, making timely computation difficult to guarantee. In such cases, the overloaded RSU-E, leveraging its high transmission rate, forwards requests to the remote cloud to avoid computation losses—albeit at the cost of higher latency and degraded QoS.

Conventional automobile application development often follows the monolithic model [9], in which the entire application logic is bundled into a single artifact, or monolith. In this architecture, modules cannot be deployed or migrated independently due to their strong coupling with other components. Consequently, if a highly demanded module must be migrated between network entities (e.g., cloud–RSU or RSU–RSU), the entire monolith must be transferred, leading to unnecessary bandwidth consumption and increased delays. To address these limitations, the microservices (MS) architecture has emerged as a promising alternative. This approach decomposes applications into several atomic MS, each performing a specific task, requiring fewer computation and communication resources, and enabling efficient migration with minimal bandwidth usage.

At present, vehicular applications utilize the inefficient

- *M. Salah ud din is with Communication Systems Department, EURECOM, 06904 Sophia-Antipolis, France.*
- *Muhammad Nadeem Ali, Ghulam Musa Raza, and B. S. Kim are with the Department of Software and Communications Engineering, Hongik University, Sejong 30016, Republic of Korea.*
- *Muhammad Imran is with the School of Information Science and Technology, Harbin Institute of Technology, Shenzhen, Guangdong 518055, China, and also with Shenzhen Municipal Key Laboratory of AIoT Communications, Shenzhen, Guangdong 518055, China.*

Manuscript received Aug 21, 2024; revised xx xx, xxxx.
(Corresponding author: Byung-Seo Kim)

host-oriented TCP/IP as the underlying communication protocol [10]. By utilizing a unique IP address, the source vehicle locates the host vehicle to establish an end-to-end communication path. However, unique IP address assignments and retaining the end-to-end communication path are significantly complex due to a lack of infrastructure support, a dynamic environment, and unreliable and lossy links.

Information-Centric Networking (ICN) and its prominent realization, Named Data Networking (NDN) [11], [12], have emerged as promising approaches to address the inherent limitations of host-centric communication, particularly in dynamic vehicular environments. Leveraging name-based communication and in-network caching, NDN mitigates issues of intermittent connectivity and high mobility by decoupling services and content from specific physical locations.

The MS-centric in-network computations, coupled with NDN as the underlying communication protocol and supported by VEC, offer a promising paradigm for efficiently meeting the resource requirements of latency-sensitive and compute-intensive vehicular applications. However, a significant obstacle hindering high efficiency is the interdependency of MS. For instance, a specific MS may not initiate its execution until all predecessors of MS have finalized their execution.

Several NDN-based MS-centric in-network computation schemes have been proposed [9], [13]–[17] to meet the communication, computation, and latency requirements of advanced autonomous vehicular applications. However, these works primarily focus on one-shot MS offloading, overlook dependencies among MS, and assume that requests complete within NDN's default Pending Interest Table (PIT) timer value. In real-world vehicular applications, MS dependencies are common: the execution of one MS may require the output of a predecessor MS—possibly running on a different computing terminal—before proceeding. Consequently, executing MS with multiple dependencies may involve gathering outputs from several computing terminals, potentially causing the total execution time to exceed the default PIT timer value.

In such cases, the consumer may fail to receive results on time due to PIT timer expiration, leading to unnecessary computation losses. Moreover, existing NDN-based data forwarding schemes allow network nodes (e.g., producers or intermediate nodes) to forward requested content indiscriminately in response to incoming interest packets, without evaluating its effectiveness or relevance. This blind transmission of large data volumes increases network resource usage and congestion, potentially delaying delivery to the consumer vehicle—a risk that, in safety-critical scenarios, could have catastrophic consequences.

To address these challenges, this paper presents interdependent microservices offloading and semantic aware results transmission (iMSoRT) for 6G vehicular edge networks. iMSoRT considers the interdependencies of requested MS and formulates a dynamic PIT timer by taking into account the MS dependencies and required computational and network resources. Additionally, iMSoRT introduces a live forwarding information base (lFIB) that effectively filters out underutilized RSU-E servers and forwards computation requests to optimal RSU-E servers during peak hours. Furthermore, the proposed work developed a semantic transformer (ST) that leverages computer vision and convolutional neural networks (CNN) to enhance the efficiency of information exchange with miniaturized network resource utilization.

The main contributions of the proposed work are listed as follows:

1) iMSoRT developed an NDN-based vehicular edge computing framework comprising autonomous vehicles, traffic controller-administered Edge computing terminals, and cloud to enable proximate computation to MS-centric computation requests.
2) A dependency-aware dynamic MS-specific PIT timer is formulated, taking into account the dependencies of MS request as well as the required computational and network resources involved in execution to avoid unsolicited packet drops and optimize network resource consumption.
3) A live Forwarding Information Base (lFIB) is developed, comprising the updated load and resource conditions of underlying RSU-E terminals. lFIB enables the traffic controller to offload MS requests to the optimal RSU-E, thereby facilitating proximate computations and reducing backhaul traffic overhead.
4) iMSoRT introduced a novel semantic transformer (ST) and mounted it between the caching and forwarding modules in NDN. The ST avoids blind data forwarding by extracting and forwarding semantically meaningful information, thereby reducing bandwidth utilization and results retrieval latency.

The remainder of this paper is organized as follows. Section 2 provides a brief background on NDN and related work. The proposed scheme is described in Section 3. Section 4 reports the performance evaluation. The complexity analysis of the proposed scheme is presented in Section 5 and finally, Section 6 concludes the paper.

2 Background & Related Work

2.1 Vehicular Named Data Networking: An overview

In vehicular networks, the NDN-based content retrieval process begins when a consumer vehicle interested in fetching named content broadcasts an Interest packet within its coverage radius. Any receiving vehicle within the transmission range processes the Interest and Data packets using intrinsic NDN data structures: the Pending Interest Table (PIT), Content Store (CS), and Forwarding Information Base (FIB). Upon receiving an Interest packet, the node first checks the PIT for an entry with the same name. If such an entry exists, Interest aggregation is performed; otherwise, the CS is checked. If the requested content is found in the CS, the node returns a Data packet along the reverse path of the incoming interface. If not, a new PIT entry is created, and the Interest packet is forwarded toward the upstream node via FIB lookup. Data packets follow the chain of PIT entries in the reverse path to reach the consumer vehicle. If no PIT entry matches an incoming Data packet, the packet is considered unsolicited and discarded.

TABLE 1: Related work comparison

Ref	Title	MS/Task dependency	Reverse path maintenance	Semantic forwarding	In-network computations
[18]	NFN	×	×	×	✓
[19]	NFAAS	×	×	×	✓
[20]	FoggyEdge	×	✓	×	✓
[14]	CFEC	×	✓	×	✓
[13]	CLEDGE	×	✓	×	×
[21]	Resource Breadcrumbs	×	✓	×	×
[22]	MACPE	×	✓	×	×
[23]	VTDF	×	✓	×	×
[24]	DBPM	×	✓	×	×
[25]	MobiVNDN,	×	✓	×	×
[26]	SCD	×	✓	×	×
Proposed	iMSoRT	✓	✓	✓	✓

2.2 Related Work:

The enchanting features of NDN [11], such as name-based content acquisition, name-based Interest/Data forwarding, and in-network caching, enable NDN to meet the challenging requirements of autonomous vehicular networking scenarios. Several state-of-the-art vehicular NDN schemes have been proposed in the literature, aiming to improve network efficiency and QoS in continuously mobile environments. Table 1 presents a comparison of various state-of-the-art NDN-based schemes devoted to the literature.

In [18], NDN-based named-function networking (NFN) was proposed, which uses function names to identify remote compute resources and perform in-network computations. An extension of NFN, known as NFaaS, was proposed in [19]. NFaaS focuses on function placement within the network and their execution via virtual machines. In this framework, any in-network node can download functions using Uni-Kernels. While these schemes perform well under stable network conditions, they may suffer from performance degradation in highly dynamic vehicular environments due to their reliance on maintaining long-lasting network state during function execution.

In [9], FoggyEdge—an information-centric computation offloading and management framework for edge-based vehicular fog computing—was proposed to enable proximate service computations with minimized delay. The framework integrates vehicular fog with edge computing and leverages microservices to perform in-network computations. Performance evaluations demonstrated that FoggyEdge reduces computational delays compared to benchmark schemes. Another NDN-based approach, CFEC [14], utilizes the idle computational resources of on-road vehicles, effectively transforming them into a large-scale distributed computing platform for proximate, microservice-centric processing. CFEC introduces a Zonal Traffic Controller to manage load conditions across RSUs, preventing resource overutilization through intelligent forwarding decisions. Experimental results showed that CFEC outperforms benchmark schemes in terms of reduced latency, lower bandwidth usage, and decreased backhaul traffic overhead. However, both FoggyEdge and CFEC consider only independent microservices and do not account for microservice dependencies or the semantics of data during computation and forwarding.

In [13], the authors proposed CLEDGE: a hybrid cloud-edge computing framework over information-centric networking. CLEDGE developed an in-network computing framework by combining edge and cloud computing resources to provide proximate computations and address the diverse latency requirements of advanced and diverse-natured computation tasks of modern IoT devices. The authors claimed that CLEDGE achieved over 90% timely task completion with modest overheads.

In [21], the authors proposed resource breadcrumbs: discovering edge computing resources over NDN, which focuses on addressing edge resource overutilization caused by blindly transmitting requests to specific edge nodes without considering their resource availability—often resulting in request failures. The proposed solution introduces a robust resource discovery mechanism in which each compute node periodically shares its available resource status within a limited scope using scoped flooding, enabling neighboring nodes to make informed forwarding decisions.

In [27], the authors proposed a mechanism for service discovery and invocation in data-centric Edge networks to tackle challenges related to service discovery, invocation, and user mobility management in edge computing. The proposed framework effectively leverages available edge resources and enhances network performance in terms of computation satisfaction rate and reduced execution delays.

In [22], the authors introduced mobility-aware content provisioning in Edge-based Content-Centric IoV (MACPE), aimed at ensuring seamless content delivery in mobile vehicular environments. MACPE implements a zone-based management system that allows content routers to maintain up-to-date information on mobile content producers, thereby improving content delivery efficiency.

In [28], the authors developed an efficient scheme to mitigate packet losses resulting from frequent path disruptions caused by high vehicular mobility. The approach incorporates parameters such as received signal strength (RSS), GPS coordinates, and vehicle speed to dynamically estimate vehicle locations in real-time and facilitate adaptive data packet forwarding.

Another similar scheme named Vehicle tracking-based data packet forwarding (VTDF) for vehicle tracking in the VNDN environment [23] was proposed, which mainly addresses link breakages between RSUs and vehicles. VTDF utilized a Tabu Node Search (TNS) to track vehicle direction in order to efficiently deliver data packets.

Another TNS-based scheme called a Data packet forwarding strategy (DPFM) [29] primarily focuses on tracking the producer vehicle to fetch and deliver content. However,

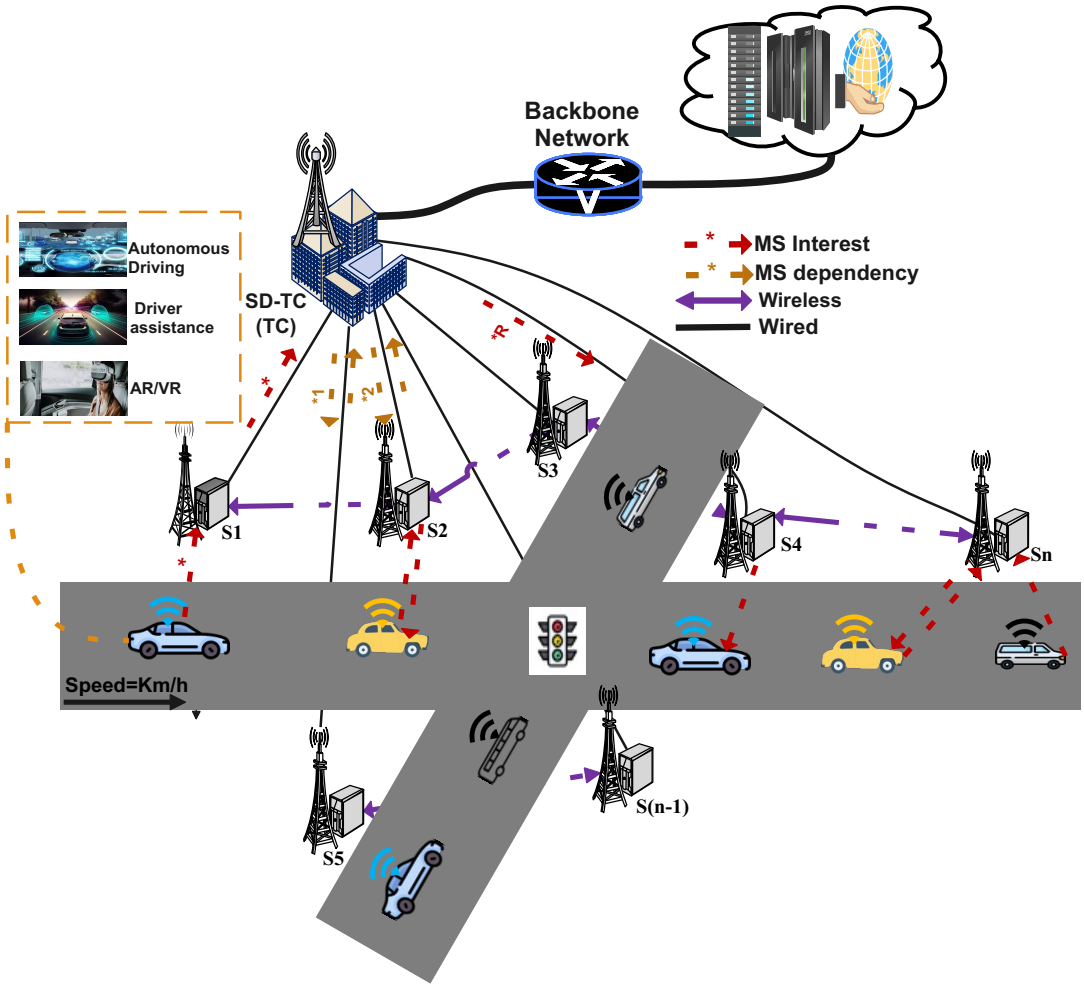


Fig. 1: iMSoRT system architecture.

DPFM assumes that the location of the vehicle is known, which may not be feasible in highly mobile vehicular networks.

Another notable work addressing reverse path partitioning is presented in [24]. The authors propose a cluster chain-based content delivery method (DBPM) for vehicular named data networking scenarios. In DBPM, the cluster chain is designed to create a backbone that enhances the stability of reverse paths. This allows cluster members to aggregate requests and share content along these reverse paths. A scheme named SCD [26] was developed to address the issue of reverse path partitioning in Vehicular Named Data Networks (VNDN). SCD introduces a forwarding strategy based on the social attributes of vehicular nodes, such as centrality, similarity, and intimacy. These metrics are used to select the optimal next-hop forwarder. However, the approach faces challenges in maintaining accurate social attribute values in dynamic vehicular environments and does not account for the coalition time—the duration vehicles remain within communication range. As a result, the scheme often experiences frequent packet retransmissions and redundant broadcasts, which can lead to network congestion and resource overutilization, ultimately limiting its effectiveness.

The authors in [25] developed a mobility management framework called MobiVNDN, aiming to enable seamless content retrieval in a vehicular environment. MobiVNDN employs a Distributed Hash Table (DHT) to store and retrieve location-based content. However, maintaining accurate location information in a highly dense and mobile vehicular environment is quite challenging.

In [24], the authors proposed a Data packet backhaul prediction method (DBPM) for vehicular-named data networking. DBPM utilized GPS and a convex programming location algorithm (CPLA) at RSUs to obtain the positioning information of vehicles in clusters. However, maintaining stable clusters under highly dynamic conditions is significantly complex.

The aforementioned state-of-the-art schemes neither consider microservice (task) dependencies nor data semantics before forwarding. However, modern applications typically involve interdependent microservices, where the execution of one microservice relies on the output of its predecessors. Neglecting these dependencies may result in incomplete or delayed computation responses, potentially exceeding the default static PIT timer. Additionally, ignoring data semantics and indiscriminately forwarding large volumes of data may lead to inefficient resource utilization,

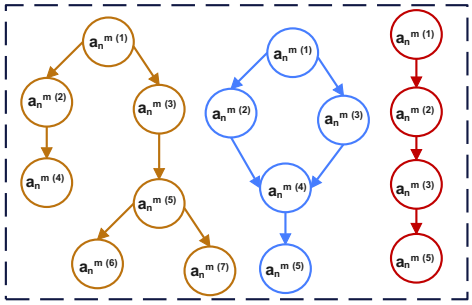


Fig. 2: Application model.

increased latency, and degraded Quality of Service — challenges that are especially critical in resource-constrained and highly dynamic vehicular environments.

3 iMSoRT: Proposed Work

Before delving into the operation of iMSoRT, we first shed light on iMSoRT system architecture, as well as the system model and assumptions. Subsequent to that, a detailed proposed methodology is presented.

3.1 iMSoRT System Architecture

The iMSoRT system architecture (depicted in Fig. 1) provides a resourceful platform for compute-intensive and latency-strict vehicular applications. The proposed architecture is comprised of four linked layers by the power and importance from the bottom to the top such as:

- **Autonomous vehicles:** This layer is composed of autonomous vehicles (i.e., the request initiators) that forward MS-centric computation requests to upstream RSU-E servers.
- **The RSU Edge Servers:** This layer is composed of lightweight Edge computing nodes co-located with the fixed roadside unit (RSU). iMSoRT calls these computing terminals RSU Edge Servers (RSU-E).
- **Software-defined traffic controller (SD-TC or T_c):** SD-TC or T_c manages traffic on roads by coordinating RSU-Es, CCTV cameras, and other infrastructure units, ensuring road users have a comfortable and safe journey. Upon receiving MS requests, the T_c analyzes the dependencies, computes the PIT timer of request, and offloads the predecessor MS (if any) to the optimal RSU-E in its vicinity.
- **Centralized cloud (CC):** CC corresponds to several diverse-natured server clusters equipped with powerful computation, communication, and storage facilities. T_c routes the computation request towards the CC when fails to perform the computation task locally or via underutilized RSU-E.

3.2 System Model and Assumptions

iMSoRT considers a hybrid vehicular network composed of L mobile vehicular nodes denoted as V= {v_1, v_2, v_3... v_l.. v_L}, N vehicular applications denoted by A = { a_l, a_2, a_3... a_n.. a_N }, M microservices represented by M={m_l, m_2, m_3 ... m_m.. m_M}. Each application comprises a collection of dependent and independent microservices, such as $a_N \in A$ = ($a_N^{(m1)}$, $a_N^{(m2)}$, $a_N{}^{(m1,m2)m3)}$,.). We consider a set of R $RSU-E$ servers denoted by S = {s_1, s_2, s_3... s_l.. s_R}. These servers handle computation tasks offloaded by the $v \in V$ which are beyond the computational capability of v.

TABLE 2: Symbols and Definitions

Symbol	Definition
RSU-E	RSU Edge server
SD-TC or T_c	Software-defined traffic controller
CC	Central cloud
V	Set of vehicular nodes
A	Set of vehicular applications
M	Set of microservices
R	Set of RSU-E
v_L	L_{th} vehicle
s_R	R_{th} RSU-E
m_M	M_{th} microservice
$t^{Tx}_{(\{v_l\},\{s_R\})}$	Request uploading time from v_L and S_R
$\partial^{input}_{\{m_M\}}$	Input size of m_M
K	Data rate
$B_{(\{v_l\},\{s_R\})}$	Link bandwidth between v_L and S_R
ρ_{v_l}	Transmission power
$g^{s_R}_{v_l}$	Channel gain
$\xi^{m_M}_{s_R} a_n$	Execution time
$\mathbb{C}_{(a_n,m_M)}$	Required computational resources
f_{s_R}	Computational capability
$t^{Rx}_{(s_R,v_l)}$	Transmission time
T_{PIT}	Computed PIT timer
dT	Default PIT timer
PEL	PIT Entry lifetime
$T^{ready}_{m_M}$	Ready time of m_M
$T^{start}_{m_M}$	Start time of m_M
$T^{finish}_{m_M}$	Finish time of m_M
$T^{comp}_{m_M}$	Total compute time of m_M
$\Gamma^{s_i}_k$	Available k_{th} resources of i_{th} s
$\Re^{m_M}_{req_k}$	Required k resources
ρ_{s_i}	Health status of i_{th} s
$\hat{\lambda}$	Images dataset
λ	Input Image
H, W, C	Height, Width and Colour channels
TP, FP, FN	True positive, False positive, and False negative

We consider an T_c that monitors underlying RSU-E ($s \in S$) in terms of available resources, load conditions, and available bandwidth. The T_c analyzes the dependencies in the received MS-centric computation requests, computes a dynamic PIT timer, and determines the most suitable $s_R \in S$ to offload the received request. If the computed PIT timer exceeds the default value, the T_c informs $v \in V$ to adjust the PIT time accordingly via the $s \in S$ server currently covering the vehicle.

iMSoRT assumes that each vehicle is equipped with a location module (e.g., GPS for location estimation using polar or cartesian coordinates), speed sensors, and direction sensors, in addition to computation, storage, and communication units. Furthermore, the RSU-E communicates with neighboring units via wireless backhauls, while a wired connection is employed for communication between the RSU-E and T_c, as well as between the T_c and CC.

Table 2 presents the notations and their definitions used in the proposed work.

3.3 Application Model:

We consider a MS-centric *intelligent autonomous vehicular application* composed of both dependent and independent

microservices, as illustrated in Fig. 2. The former refer to those that cannot execute independently and require input from one or more predecessor microservices while the later can execute without any such dependencies.

To enhance clarity, the MS-centric application—denoted as $a_n \in A$—can be represented as a Directed Acyclic Graph (DAG), defined as $G = (\gamma, \varepsilon)$, as shown in Eq. 1:

$$\varepsilon = \begin{pmatrix} 0 & e_{1,2} & e_{1,3} & \cdots & e_{1,m} \\ 0 & 0 & e_{2,3} & \cdots & e_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{(M-1),M} \end{pmatrix} \tag{1}$$

Where $e_{i,j} \in \{0, 1\}$ and $e_{i,j}$ = 1 indicates that microservice j is dependent on microservice i, meaning j requires the output of i to begin execution. Conversely, $e_{i,j} = 0$ implies that microservices i and j are independent.

3.4 iMSoRT: Operation

This section provides a comprehensive description the proposed work.

3.4.1 Microservice adaptive PIT timer computation:

Consider a vehicle $v_L \in V$ (shown in Fig.3) offloads an MS-centric computation request (i.e., $m_M \in M$) to s_R. To compute the execution time of m_M, it is necessary to identify the processes involved in its execution. Specifically, the completion time of m_M is mainly comprised of:

1) **Request uploading time:** Refers to the time taken to route the request to s_R.
2) **Request execution time:** Refers to the time required to execute the request by the given s_R.
3) **Results delivery time:** Defines the time to deliver the computed results back to v_L.

Consider the scenario where a vehicle v_L offloads the m_M to the covering s_R (shown by yellow dotted arrowhead in Fig.3).

The request forwarding time from v_L to s_R is denoted as $t^{Tx}_{(vL,sR)}$ and is defined as follows:

$$t^{Tx}_{(v_L,s_R)} = \frac{\partial^{input}_{m_M}}{\kappa} \tag{2}$$

Where $\partial^{\text{input}}_{mM}$ represents the input data size of microservice mM, and κ denotes the transmission rate between v_L and s_R, such that:

$$\kappa = B_{(v_L,s_R)} \log(1 + \frac{\rho_{v_l} g^{s_R}_{v_L}}{N_0}) \tag{3}$$

$B_{(v_L,s_R)}$ represents the link bandwidth between v_L and s_R, ρ_{v_L} denotes the transmission power of v_L, $g^{s_R}_{v_L}$ is the channel gain, and N_0 denotes the noise.

Upon receiving the m_M request, the corresponding s_R determine whether to execute it locally or offload it to T_c. We define a decision variable i.e., ${}_{s_R}X^{a_n}_{m_M}$ such that:

$${}_{s_R}X^{a_n}_{m_M} = \begin{cases} 1, & \text{if required resources and } m_M \text{ is available} \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

If adequate computational resources and the MS code (including any dependencies) are available (i.e.,${}_{s_R}X^{a_n}_{m_M}$ =1), the s_R proceeds to compute the execution time of the request using the following equation:

$${}_{s_R}\xi^{a_n}_{m_M} = \frac{\mathbb{C}_{(a_n,m_M)}}{f_{s_R}} \tag{5}$$

Here, ${}_{s_R}\xi^{a_n}_{m_M}$ denotes the time taken by the s_R to compute the m_M of application a_n. $\mathbb{C}_{(a_n,m_M)}$ represents the amount of computational resources required to execute m_M, while f_{s_R} denotes the computational capability of s_R respectively.

If ${}_{s_R}\xi^{a_n}_{m_M}$ exceeds the default PIT timer (i.e., dT), the s_R informs the v_L to adjust the PIT timer presented as follows.

$$y = \begin{cases} \xi^{a_n}_{m_M} > dT & \text{adjust the PIT timer} \\ \text{otherwise} & \text{compute and forward} \end{cases} \tag{6}$$

After computing the ${}_{s_R}\xi^{a_n}_{m_M}$, the result transmission time (i.e., $t^{Rx}_{(s_R,v_L)}$) from s_R to v_L is presented as follows:

$$t^{Rx}_{(s_R,v_L)} = \frac{\partial^{output}_{m_M}}{\kappa} \tag{7}$$

Where $\partial^{output}_{m_M}$ shows the output results size and κ represents the data transmission rate.

Considering the aforementioned computations, the PIT time set by the consumer is the sum of the dT, ${}_{s_R}\xi^{a_n}_{m_M}$ and $t^{Rx}_{(s_R,v_L)}$ presented as follows:

$$T_{PIT} = dT + {}_{s_R}\xi^{a_n}_{m_M} + t^{Rx}_{(s_R,v_L)} \tag{8}$$

The s_R forwards an acknowledgment message containing the updated timer information to v_L, which then updates the PIT timer entry corresponding to the respective interest.

Apart from independent microservices, an MS-centric computation request may also depend on its predecessor microservices, requiring their outputs before it can begin execution. Consider a scenario where a vehicle v_L, offloads a MS request (e.g., m_M indicated by a yellow bold arrowhead in Fig. 3) to s_R. s_R verifies whether a dependency is required to execute m_M. If execution of any interdependent MS is required, s_R checks for the availability of the interdependent MS code and the required resources for execution. If available, s_R executes and hands over the results. If not available, s_R offloads the request to the T_c.

Upon receiving the m_M request, the T_c first identifies the predecessors of m_M and exploits the potential RSU-E (i.e., $s \in S$), to perform computations. Since the request has dependencies and involve multiple Edge servers to complete the execution, the overall execution time may exceed the default PIT timer due to multiple transmissions, computations, and results receptions between T_c and underlying s_R. To address this, the T_c computes a dynamic PIT timer by taking into account the predecessor MS, alongwith the required processing and communication resources.

The process for computing the PIT time while considering MS dependencies is outlined as follows:

A dependent MS-centric request (e.g., m_M) can begin execution only after the completion of its predecessor MS (e.g., m_{Mx}), as the successor requires the predecessor's output as input. Consequently, the start time of m_M is determined by the finish time of m_{Mx}. It is important to note that the execution of m_M at s_R may be delayed if its predecessor m_{Mx} is offloaded to a different server. In such

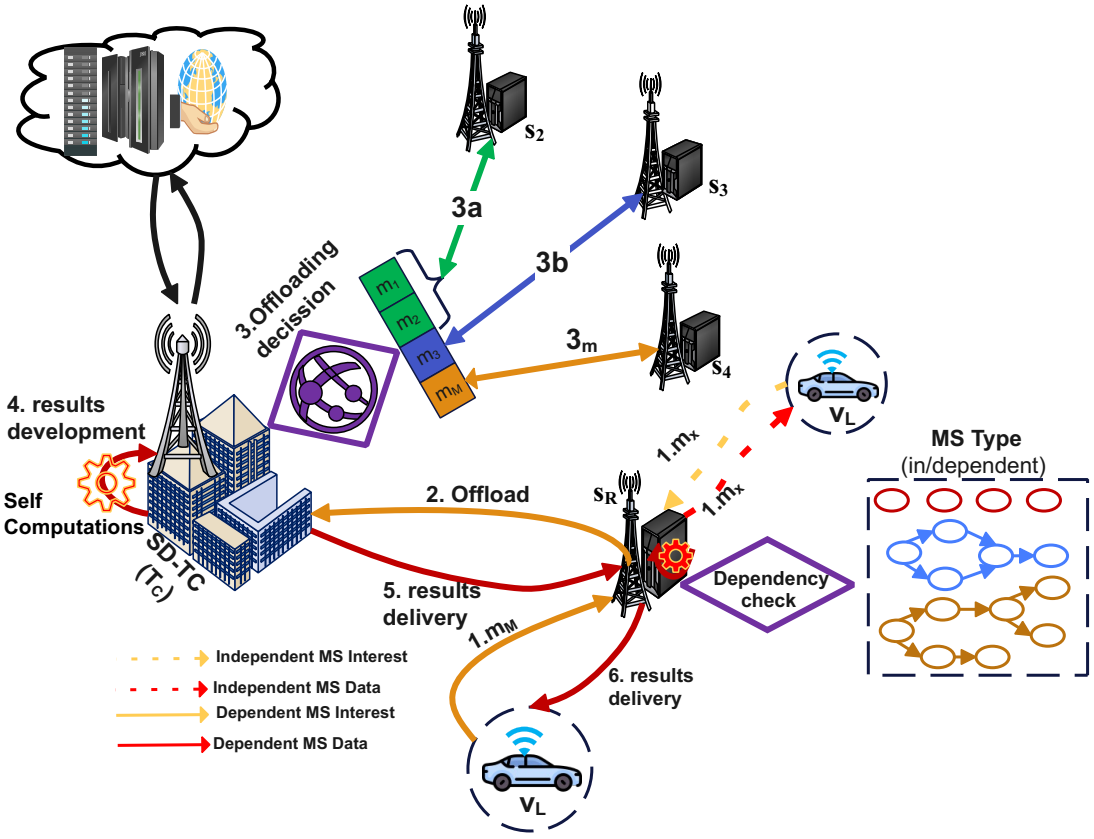


Fig. 3: Dynamic PIT timer computation mechanism

cases, s_R must wait for the output from the predecessor before initiating the execution of m_M.

Upon receiving the output results of m_{Mx}, m_M becomes ready for execution, as represented below:

$$T_{m_M}^{ready} = \max_{T_{m_{Mx}} \in pre(T_{m_M})} T_{m_{Mx}}^{finish} \tag{9}$$

Where $T_{m_M}^{ready}$ represents a ready time of m_M and $T_{m_{Mx}}^{finish}$ represents the finish time of all predecessors i.e., m_{Mx}.

When m_M is ready to execute at s_R, its execution can begin immediately if no other MS is currently running and the required resources are available. However, if another microservice is already in execution at s_R, m_M must wait until the necessary resources become available.

Thus, the start time of execution (i.e., $T_{m_M}^{start}$) of m_M is presented as:

$$T_{m_M}^{start} = \max\{{}_{s_R}T_{m_M}^{ready}, {}_{s_R}T_{m_M}^{available}\} \tag{10}$$

Where ${}_{s_R}T_{m_M}^{available}$ is the time when the resources are available at s_R to execute m_M. The execution time of m_M at s_R is provided as follows:

$${}_{s_R}\xi_{m_M} = \frac{\mathbb{C}_{m_M}}{f_{s_R}} \tag{11}$$

Where, ${}_{s_R}\xi_{m_M}$ denotes the time taken by the s_R to compute the m_M, f_{s_R} and $\mathbb{C}_{m_M}$ represent the amount of required computational resources and computational capability of s_R respectively. The finish time of m_M is provided as follows.

$$T_{m_M}^{finish} = T_{m_M}^{start} + {}_{s_R}\xi_{m_M} \tag{12}$$

The total time taken by the s_R to compute the m_M (e..g., $T_{m_M}^{comp}$) is provided as follows

$$T_{m_M}^{comp} = T_{m_M}^{finish} - T_{m_M}^{start} \tag{13}$$

Considering the formulations, the PIT time computed by the T_c is presented as follows.

$$T_{PIT} = dT + t_{(s_R,T_c)}^{Tx} + \sum_{i=1}^{n}\left[t_{(T_c,s_i)}^{Tx} + {}_{s_i}\xi_{pre(m_M)}^{a_n} + t_{(s_i,T_c)}^{Rx}\right] + T_{m_M}^{comp} + t_{(T_c,s_R)}^{Rx} + t_{(s_R,v_L)}^{Rx} \tag{14}$$

The T_c forwards the computed PIT timer (i.e., T_{PIT}) value to the v_L via the s_R, which is currently covering the v_L.

After sharing the PIT timer, T_c delegates the predecessor microservice to the most suitable RSU-E servers in its vicinity using the proposed live Forwarding Information Base (lFIB).

3.5 Live Forwarding Information Base (lFIB) dependency aware request offloading:

An RSU-Edge server (i.e., $s \in S$) is considered optimal if it possesses high computational resources, is located at a shorter distance from T_c, has ample storage capacity, and offers sufficient bandwidth to perform computations efficiently. It is important to note that resource availability at RSU-Es can fluctuate over time depending on their location. For example, an RSU-E situated at a major junction typically experiences a higher workload compared to those in less congested or rural areas.

To manage this variability, each RSU-E proactively shares its current resource availability with T_c whenever its load exceeds a predefined application-specific threshold. Based on the collected load information from underlying RSU-Es, T_c constructs the live Forwarding Information Base (lFIB), in which each interface is ranked according to its real-time load condition—referred to as its health status.

The formulation of RSU-E health status and the mechanism for developing the lFIB are outlined as follows:

Problem Formulation: As already discussed, an RSU-E is considered an optimal offloading candidate if it has enough resources to accomplish the incoming computations requests i.e.,

$$_{av}\Gamma_k^{s_i} \; > \; _{req}\Re_k^{m_M} \; \forall \, _{req}\Re_k \; \in \; \mathbb{R}, \, m \; \in \; M \tag{15}$$

Where $_{av}\Gamma_k^{s_i}$ represent the available k_{th} resources of i_{th} s and $_{req}\Re_k^{m_M}$ denote the required k resources to execute m_{th} microservice. In order to evaluate the overall health status of each s, the T_c computes the individual score (i.e., Ψ_k) of each available resource and normalizes the individual score to a specific range i.e., [u v] using the following equations.

$$\Psi_k \; = \; \frac{_{av}\Gamma_k^{s_i}}{_{req}\Re_k^{m_M}} \tag{16}$$

$$\rho_{s_i} = \sum_{k \, \in \, \mathbb{R}} \varphi^k [u + \frac{(\Psi_k) - \min{(\Psi_k)} \; \times \; (v-u)}{\max{(\Psi_k)} - \min{(\Psi_k)}}], \quad s \in \; S \tag{17}$$

Where ρ_{s_i} corresponds to the health status of i_{th} RSU-E (i.e., $s \in S$) and φ^k is the decision maker's assigned application-specific weight factor of k_{th} resource and $\mathbb{R}$ is the set of resources. An RSU-E that bears the highest health value is considered a potential offloading candidate.

It is important to note that vehicles frequently change their locations; therefore, due to high mobility, it is likely that v_L may not remain within the coverage area of the same RSU-E server to which it initially offloaded the request. Therefore, T_c routes the response through the RSU-E currently serving v_L.

3.6 Semantic Transformer:

Modern autonomous vehicular applications rely on data from smart sensing equipment—such as inductive loops, ultrasonic sensors, radar, and CCTV cameras—to make informed decisions that enhance passenger comfort and safety. These devices generate vast amounts of data, including high-definition images and continuous video streams. For example, a single CCTV camera can produce 25–30 frames per second during video transmission, and even low frame rate sequences (e.g., 1.25 Hz) can generate over 100 MB of data per second [30]. As a result, substantial computational, storage, and bandwidth resources are required to process, manage, and transmit such data in real-time.

In resource-constrained vehicular environments, existing NDN-based forwarding schemes [9], [11], [13], [14], [24], [26], [29] allow nodes—such as producers or intermediate forwarders—to blindly forward requested content in response to incoming Interests, without evaluating the relevance or effectiveness of the transmitted data. This approach leads to increased network resource consumption and congestion, which can delay the delivery of critical data to the consumer vehicle and, in extreme cases, result in catastrophic consequences.

To overcome these challenges, we modified the NDN protocol suite and introduced a Semantic Transformer (ST), positioned between the caching and forwarding modules. Powered by Convolutional Neural Networks (CNNs) and the YOLO (You Only Look Once) computer vision algorithm, the ST enhances the efficiency of information exchange by extracting semantically meaningful content from requested data. For example, when a consumer requests visual content (e.g., images or videos) of a specific road segment captured by an RSU-E's CCTV cameras, the RSU-E first processes the raw data through the ST. The ST identifies and extracts relevant semantic information—such as objects, events, or anomalies—and forwards only this distilled information to the consumer via the selected FIB interface. This mechanism significantly reduces unnecessary data transmission, conserves network bandwidth, and delivers precise situational awareness to the consumer with low latency.

A complete process of ST development and semantic extraction depicted in Fig.4 is provided as follows:

3.6.1 CNN-enabled image segmentation:

Consider an $s_R \in S$ receiving a request from a consumer autonomous vehicle requiring the traffic condition of a specific road segment at a certain time. The corresponding s_R fulfills the request either from its local cache or by capturing the real-time traffic information using the CCTV camera mounted at the location, depending upon the nature (i.e., real-time or historical data) and availability of requested content.

Consider the input image $\lambda \in \hat{\lambda}$ e.g., $\lambda \in \hat{\lambda} \subset \beta^{H \times W \times C}$ captured by the CCTV camera with height H, width W, and color channels C and data set $\hat{\lambda}$ and integer grey values β. Each input image is composed of grey values e.g., $\lambda_i \in \beta^C$ at every pixel position $|I| = H.W$ where I denotes the set of pixel positions with the cardinality $|I| = H.W$. A specific patch of the image is denoted by $\lambda_{I_i} \in \beta^{h \times w \times C}$ where h, w, and c denote the height, width, color channels, pixel position $I_i \subseteq I$ and $|I_i| = h.w$.

A CNN (depicted in the upper half of Fig.4) is comprised of various layers (i.e., L). Each respective layer (i.e., $l \in L$) has feature map activation i.e., $f_l(\lambda) \in \mathbb{R}^{H_l \times W_l \times C_l}$, input image i.e., $\lambda \in \hat{\lambda}$ in the first layer, the feature map height H_l, width W_l and the number of color channels C_l. When the input image is fed into the CNN for image classification, it outputs a probability score i.e., $P(s|\lambda) \in \mathcal{I}$ for each class i.e., $s \in S$ where $\mathcal{I} = [0, 1]$, S denotes the set of available classes while $N = |S|$ is the total number of classes. The predicted class of input image λ is presented as follows.

$$s^*(\lambda) = \arg\max_{s \in S} \; P(s|\lambda) \tag{18}$$

From now on, we will consider a CNN capable of performing semantic segmentation ($\wp$). Given the input image λ and the class $s \in S$, the CNN provides a probability i.e., $P(s|i, \lambda)$, against each pixel position i.e., $i \in I$. For all pixel positions (i.e., $i \in I$) and classes (e.g., $s \in S$), the output class score is as follows.

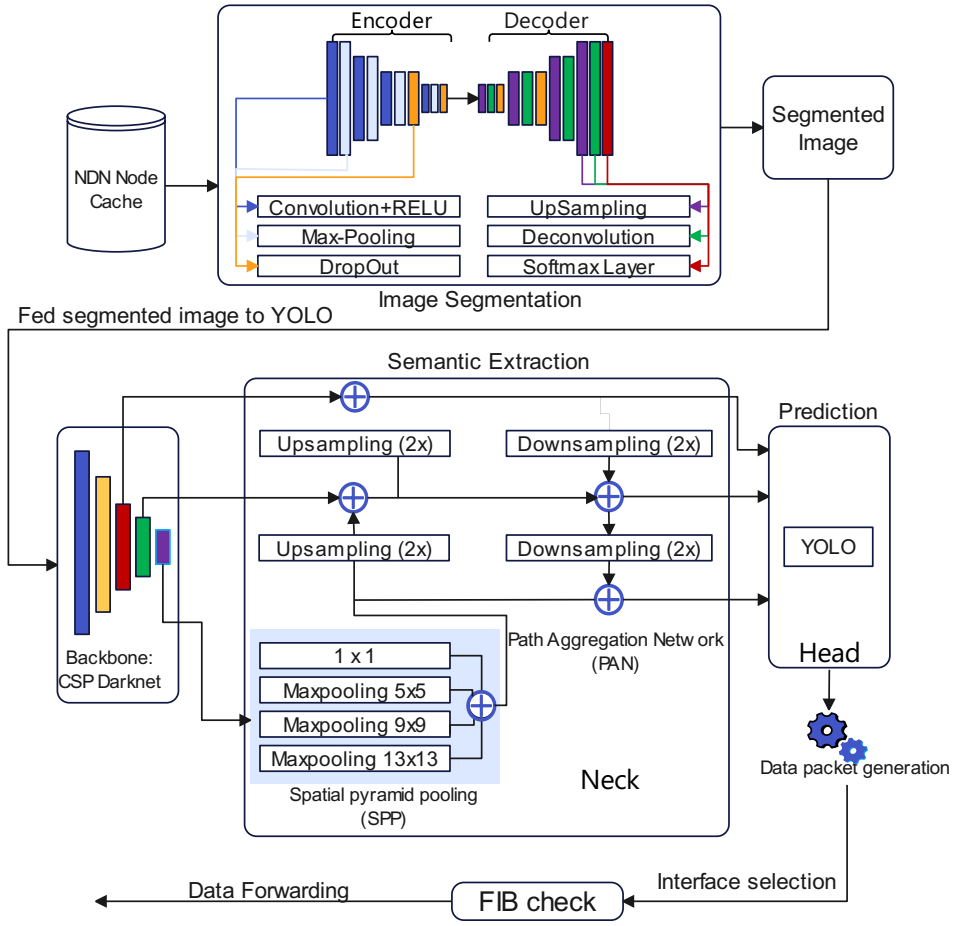


Fig. 4: Semantic transformer

$$p(\lambda) = \wp(\lambda) \in I^{H \times W \times N} \tag{19}$$

$$\wp : \beta^{H \times W \times C} \rightarrow I^{H \times W \times N} \tag{20}$$

The semantic segmentation mask denoted by $m(\lambda) \in S^{H \times W}$ composed of predicted class (i.e., $m_i(\lambda) = s_i^*(\lambda)$) at every pixel position of the input image λ is computed as follows.

$$m(\lambda) = \arg\max_{s \in S} P(\lambda) \tag{21}$$

Finally, the mean intersection over union (i.e., ∂) is utilized to measure the performance of CNN.

$$\partial = \frac{1}{N} \sum_{s \in S} \frac{TP(s)}{TP(s) + FP(s) + FN(s)} \tag{22}$$

Where TP, FP, and FN denote true positive, false positive, and false negative respectively.

3.6.2 YOLO-based object detection, localization and labeling:

The segmented image obtained from the CNN is fed into $YOLO$ depicted in the lower half of Fig.4 to identify and locate objects within the image. $YOLO$ enables real-time object detection with high accuracy, making it an optimal choice for use in autonomous vehicular decision-making. The proposed work adopts $YOLO - v4$ that employs $CSPDarkNet - 53$ as its backbone to extract features from the provided input image. The backbone comprises five residual blocks. The output of these residual blocks is passed to the $NECK$, where the spatial pyramid pooling (SPP) module resides. The SPP module, using kernels of size$1x1$, 5×5, 9×9, and 13×13 (stride=1 for the max-pooling operation), concatenates the max-pooling results of the low-resolution feature map to extract the most representative features. The output of the SPP module is then fused with high-resolution feature maps using a Path Aggregation Network (PAN). The PAN employs $up - sampling$ and $down - sampling$ operations to establish bottom-up and top-down paths for combining low-level and high-level features. The PAN module outputs a set of aggregated feature maps for predictions. The $YOLO - v4$ network has three detection heads, each of which is a $YOLO - v3$ network that computes the final predictions. The YOLO-v4 network outputs feature maps of sizes 19×19, 38×38, and 76×76 to predict bounding boxes, classification scores, and abjectness scores. By combining the speed of $YOLO$ with the precision of image segmentation, autonomous vehicular applications achieve more balanced and efficient use of network and communication resources.

The ST ensures that only critical information is processed and transmitted efficiently, thereby enhancing the overall safety, responsiveness, and performance of NDN-enabled

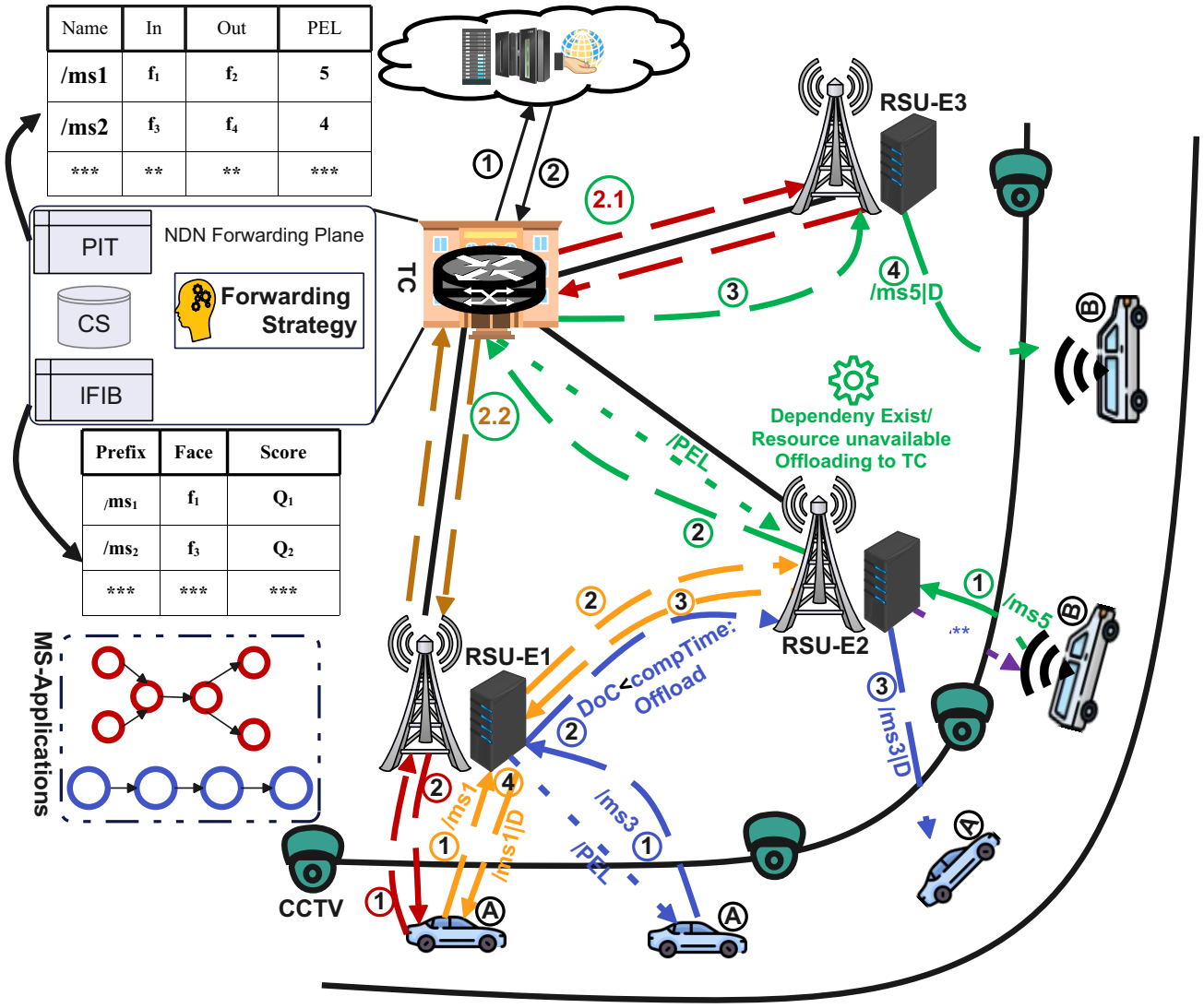


Fig. 5: Computations offloading and results delivery

autonomous vehicles.

3.7 iMSoRT: Computation Philosophy and Offloading Mechanism

3.7.1 MS Centric Naming Schema:

A MS-centric naming design plays a crucial role in computation offloading, migration, and result delivery. iMSoRT adopts an MS-centric naming schema that follows the hierarchical and semantically meaningful principles of NDN. We consider an *intelligent driver-assistance application* composed of various dependent and independent MS.

To name the MS, we incorporate vehicle trajectory information—such as speed, direction, current location, and destination coordinates—in addition to the MS name (e.g., ***/drivAssist/vehNum/curLoc/destLoc/speed/direction*|*MS?P1,P2***). These additional components in the namespace, including current and destination coordinates, vehicle speed, and direction, enable the corresponding RSU-E or T_c to avoid reverse-path partitioning and to ensure guided result delivery in a continuously varying environment. A vertical pipe symbol (i.e., "|") is used to separate the vehicular dynamics from the MS name. The MS name also includes its input parameters (e.g., "P1, P2"), which are prefixed by a question mark (i.e., "?") and separated by commas (i.e., ",").

3.7.2 MS centric computation Offloading and results delivery:

A computation offloading process mainly triggers when the required resources to perform a specific MS-centric computation exceed the available resource capacity of the autonomous vehicle. In such scenarios, the vehicle offloads the computation tasks to the RSU-E currently covering the region. We consider different MS-centric computation offloading and results handover scenarios starting from obvious to complex cases as depicted in Fig.5.

The detailed MS offloading and computations delivery procedure is provided as follows.

1) **Consumer Vehicle to Edge Offloading:** Let us start with a simple case (as illustrated in steps 1-2 in red in Fig.5)

TABLE 3: Simulation parameters

Parameter	Value
Simulator	NS-3 (ndnSIM)
Communication stack	NDN
Mobility generator	SUMO
Traffic Scenario	Urban
Number of RSU-Es	5
Wireless interface	IEEE 802.11p
Network size (i.e., number of vehicles)	30
RSU tx Range	300m
Vehicle tx Range	250m
Average vehicle speed	40Kmph, 65Kmph
Total number of MS	7 (dependent = 4, independent=3)
MS request rate	5-30 requests/sec
MS request distribution	Random
Simulation time	200s

TABLE 4: Training parameters

Parameter	Value
Batch size	4
Conv	$3 \times 3 \times 3$
Max-pooling	$2 \times 2 \times 2$
Kernel size	2
Learning rate	0.01
Epochs	30
Loss function	Mean square error (MSE)
Optimizer	Adam
Activation function	ReLU
Training dataset	2780 (80%)
Validation dataset	695 (20%)

where the consumer vehicle "A" requires the optimal route between two locations. It offloads a MS-centric request ms1 (e.g.,***/drivAssist/../..*** | ***RoutePlan?loc1,loc2***), comprising the initial location (i.e., loc1) and final destination coordinates (i.e, loc2).

Upon receiving the request, the corresponding RSU-E first computes the DoC [15] with "A" and verifies whether the requested MS has any dependencies. In this case, the DoC exceeds the computation time, and the requested MS does not require the output from any predecessor MS. Furthermore, the computation time is within the default PIT timer value. The RSU-E then calculates the optimal route based on the received location coordinates and returns the results via the reverse path, following vanilla NDN principles.

2) **Inter Edge Offloading:** Consider a scenario (shown in Fig. 5, steps 1-3 in blue color) where the fast-moving vehicle "A" offloads a computation request to RSU-E1 (e.g.,***/../..*** | ***liveRoad?loc1,loc2/<timestamp>***), requiring live traffic conditions of a particular road segment within the coverage area of RSU-E2. Upon receiving the request, RSU-E1 computes the DOC as well as the required computation time based on the nature of the received request.

   As the computation time exceeds both the DoC and the default PIT entry lifetime (PEL), RSU-E1 sends a response message containing an updated PIT entry lifetime, allowing the consumer to adjust the PIT timer of the Interest and thereby avoid unsolicited data losses. Simultaneously, RSU-E1 appends the *adhoc response* to the incoming Interest packet and forwards it to RSU-E2. RSU-E2 then obtains live traffic conditions via a CCTV camera mounted within its coverage area, performs the semantic transformations using ST, and delivers the results to the consumer over the ad-hoc interface. "B".

3) **Edge to T_c Offloading:** Consider a scenario (steps 1-4 in green in Fig. 5) where a vehicle "B" offloads a dependent MS-centric computation request to RSU-E2. The RSU-E2 first authenticates the request, computes DoC, and verifies the availability of the required resources (e.g., computational resources, requested MS, and its predecessors) to perform the execution. Due to resource unavailability, the RSU-E2 appends the DoC to the Interest name and forwards it to the T_c. The T_c verifies the request and computes the required computation time, considering the dependencies of the requested MS.

   If the computation time exceeds the default PIT entry lifetime, the T_c first verifies the DoC value of the consumer with RSU-E2. Since the DoC is not expired, the T_c forwards the updated PEL to the consumer vehicle via the reverse path (i.e., /PEL denoted by the green dotted arrow in Fig. 5). If the DoC has expired, the T_c forwards the updated PEL to the consumer vehicle via the RSU currently covering consumer vehicle. After sharing the updated PEL, the T_c exploits the optimal RSU Edge-E servers (via lFIB) capable of executing the predecessor MS (as illustrated in steps 2.1 and 2.2 in red and brown in Fig. 5) to offload the computation request and obtain the subsequent computation results.

   Once the results of the predecessors are obtained, the T_c performs the requested MS computations, compiles the results, and hands them over to the consumer via the RSU-E3 currently managing the consumer.

4) **T_c to Cloud Offloading:** This process is initiated when the load on an RSU-E exceeds a specified threshold or when the required MS code to fulfill a user request is unavailable. In such cases, T_c either offloads the consumer request to the CC or retrieves the MS code to execute the requested computations. This mechanism prevents computation losses and enhances the QoS of consumer applications (illustrated in Fig. 5, steps 1–2 in black).

4 Performance Evaluation

This section deals with detailed performance evaluation.

4.1 Simulation Setup

To analyze and evaluate the performance of iMSoRT, extensive simulations were conducted in $ndnSIM$ (an NS-3-based network simulator). The $OpenStreetMap$ [31] and $SUMO$ [32] mobility generator is integrated with $ndnSIM$ to envision a realistic urban traffic environment.

The proposed work is compared with state-of-the-art schemes, specifically the Cluster Routing-Based Data Packet Backhaul Prediction Method in Vehicular Named Data Networking ($DBPM$) [24] and the Social Attributes-Based Content Delivery for Sparse Vehicular Content-Centric Network (SCD) [26], to benchmark performance. The simulation environment comprises five RSU-E servers with heterogeneous computing capabilities, a T_c responsible for managing the RSU-E servers within its vicinity, and a cloud station.

The number of vehicular nodes in the simulations ranges from 5 to 30, with speeds varying between 40 km/h and 65 km/h. The setup also includes seven different types of dependent and independent MS, each with distinct computational requirements. To simulate computational behavior, iMSoRT employs the ndnCSIM [33] codebase, which provides the core functionality for MS execution and node resource management.

For ST, we employed the CNN and $YOLO$-v4 algorithms. The models were trained using the Cityscapes dataset [34], which is well-suited for analyzing intelligent vehicular networking scenarios. To further evaluate the performance of ST, we collected a custom dataset from road conditions around Jochiwon Station in Sejong City, Korea. This dataset contains diverse images featuring vehicles of varying sizes (small, medium, and large) and surrounding environments, including buildings, trees, pedestrians, and traffic signals. The trained model was then integrated into $ndnSIM$ to perform semantic extractions.

The complete simulation and training parameters are summarized in Table 3 and Table 4, respectively.

To evaluate the performance of iMSoRT against the benchmark schemes the following metrics are considered:

1) **Compute hit ratio (CHR):** The compute hit ratio (CHR) is defined as the ratio of the number of MS-centric interest packets that successfully received the computation results (i.e., Data packets) within the PIT timer. Mathematically, the CHR can be represented as:

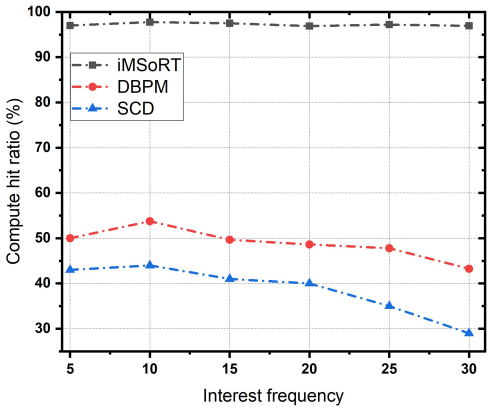
$$CHR = \left[\frac{\sum_{1}^{n} I_{hit}}{\sum_{1}^{n} I_{hit} + \sum_{1}^{n} I_{miss}} \right] \times 100 \qquad (23)$$

Where I_{hit} denotes MS-centric Interest packets that successfully received computation results and I_{miss} denotes MS-centric Interest packets that failed to receive any computation results.

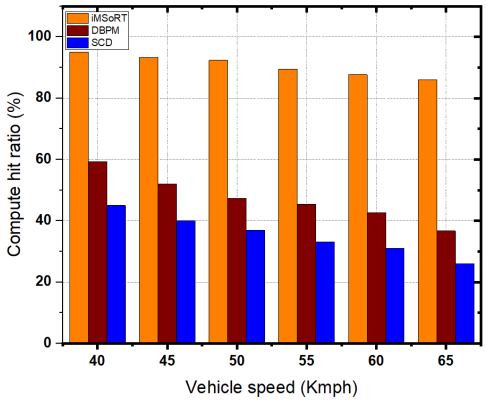
2) **Computation satisfaction delays (CSD):** The computation satisfaction delay (CSD) is the total time taken by an MS-centric request to reach the corresponding compute node, the request processing time, and the results delivery time to the consumer.
3) **Average hop count:** The average hop count corresponds to the number of hops the packet traverses from a content source (i.e., compute node) to the consumer node.
4) **Edge resource utilization:** Edge resource utilization is defined as the fraction of resources utilized to the total available resources of the Edge computing node during different workload conditions.
5) **Task offloading to cloud:** It measures the proportion of MS-centric computation requests offloaded to the cloud.
6) **Bandwidth consumption:** The bandwidth consumption is the total bandwidth consumed in Data packet transmissions following both semantic and conventional data forwarding strategies.

4.2 Evaluation Results

1) **Compute hit ratio (CHR):** We evaluate the efficacy of the proposed work by measuring the CHR as a function of Interest frequency and vehicle speed, as shown in Fig. 6(a) and Fig. 6(b), respectively. The results clearly demonstrate the effectiveness of iMSoRT in maintaining high CHR in both scenarios. As Interest frequency and vehicular speed increase, iMSoRT achieves CHR values exceeding 95% and 85%, respectively, outperforming the baseline techniques. This improvement is attributed to the proposed dynamic PIT timer mechanism, which adjusts the timer according to the resource requirements of both dependent and independent MS-centric requests. This prevents premature PIT timer expiration and ensures that vehicles receive computation results in a timely manner.



(a)



(b)

Fig. 6: Compute hit Ratio

Furthermore, iMSoRT effectively addresses the reverse-path partitioning problem by estimating the consumer vehicle's current location from its trajectory information—including speed, current location, and destination coordinates—and offloading computation results via the RSU-E server currently covering the vehicle. This significantly improves CHR. In contrast, neither $DBPM$ nor SCD account for MS dependencies or adapt the PIT timer to the computation and communication requirements of MS requests. As a result, compute nodes may be unable to complete processing within the default 2-second PIT lifetime, causing unnecessary computation losses. Additionally, $DBPM$ employs RSU clustering to predict vehicle locations, while SCD leverages social

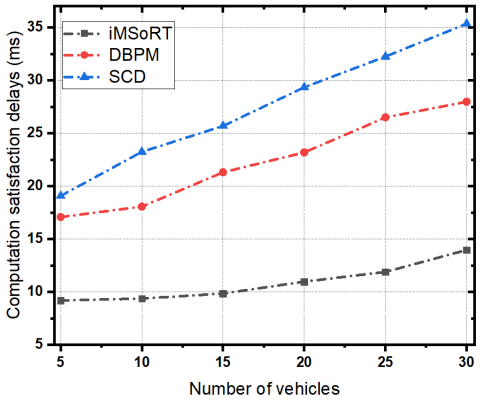


Fig. 7: Computation satisfaction delays

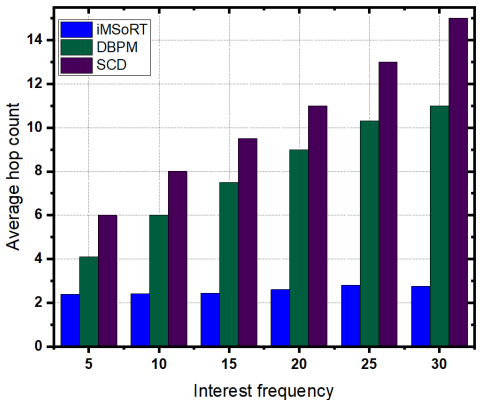
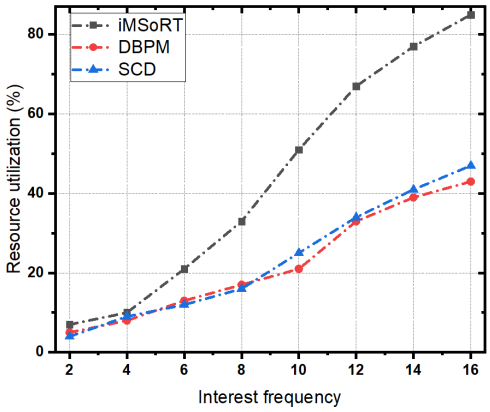
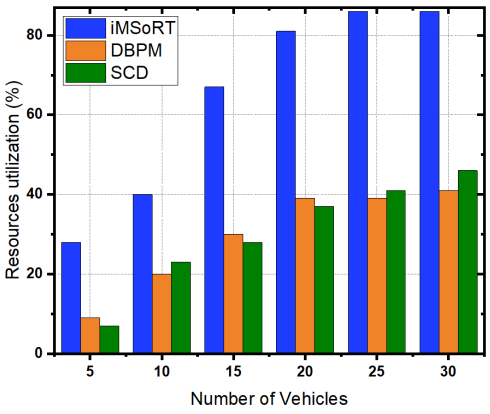


Fig. 8: Average hop count



(a)



(b)

Fig. 9: Resource utilization

metrics (e.g., centrality, similarity, intimacy) to avoid path breakage during result delivery. In highly mobile traffic conditions, maintaining stable connectivity solely based on these factors—without considering vehicle speed and trajectory—limits timely result delivery and reduces CHR.

2) **Computation satisfaction delays (CSD):**
We evaluate the computation satisfaction delay (CSD) by varying the number of vehicular nodes, as shown in Fig. 7. The CSD is directly correlated with traffic load; as the workload increases, the RSU-E's ability to process concurrent MS requests decreases, resulting in longer delays. Fig. 7 shows that CSD rises across all schemes as the number of vehicles grows. However, the proposed method consistently outperforms the baseline schemes, achieving lower CSD even under high traffic conditions. This improvement is attributed to iMSoRT's proximate computation policy, which enables T_c to maintain updated resource information for the underlying RSU-E servers in its lFIB table. Consequently, under heavy load, an overloaded RSU-E can offload computation requests to T_c, which then forwards them to idle or underutilized RSUs, thereby reducing traversal distance and minimizing latency.
In comparison, although $DBPM$ outperforms SCD, both exhibit higher CSD than iMSoRT. This is due to the limited availability of computational resources in SCD and $DBPM$, which constrains the compute node's onsite processing capacity. As a result, tasks must be offloaded to the cloud, leading to extended CSD.

3) **Average hop count:** The comparison of average hop count (AHC) as a function of Interest frequency is shown in Fig. 8. For this analysis, the Interest packet frequency was varied from 5 to 30 interests per second. The results indicate that iMSoRT consistently outperforms both benchmark schemes in terms of AHC. This improvement stems from iMSoRT's ability to enable T_c to account for vehicular dynamics—such as speed, direction, and trajectory—when returning computed results to the consumer vehicle. Consequently, T_c forwards the results directly to the consumer through the RSU-E currently covering the vehicle, minimizing hop count.
In contrast, $DBPM$ requires the cluster-member RSU to forward the Interest packet to a distant cluster-head RSU, which either performs the computation locally or offloads it to the producer RSU-E, thereby increasing the number of hops. Similarly, in SCD, the Interest packet is forwarded to the node with the highest social weight, continuing this process until it reaches a potential compute node—also resulting in a higher hop count.

4) **Resource utilization:** Fig. 9(a) and Fig. 9(b) show the impact of compute resource utilization as a function of Interest frequency and the number of vehicles, respectively. The results indicate that the proposed scheme

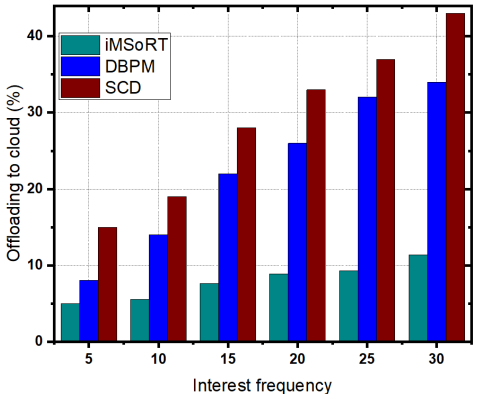


Fig. 10: Tasks offloading to cloud

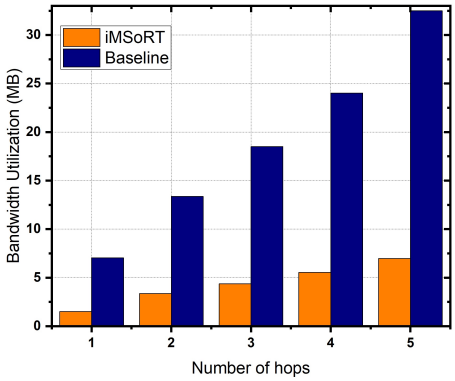
utilizes more resources compared to $DBPM$ and SCD. This is because iMSoRT avoids cloud offloading and maximizes the use of available in-network resources to provide proximate computation for consumer requests. To achieve this, the proposed scheme leverages the interdependencies of MS-centric computation tasks. When necessary—due to resource constraints or unavailability of MS code—iMSoRT exploits the resources of optimal RSU-E servers in its vicinity to offload and execute computation requests. The involvement of multiple RSU-E servers in request execution leads to higher resource utilization, as observed in Fig. 9(a) and Fig. 9(b).

In contrast, both $DBPM$ and SCD ignore MS interdependencies and execute only independent requests, resulting in lower resource utilization.

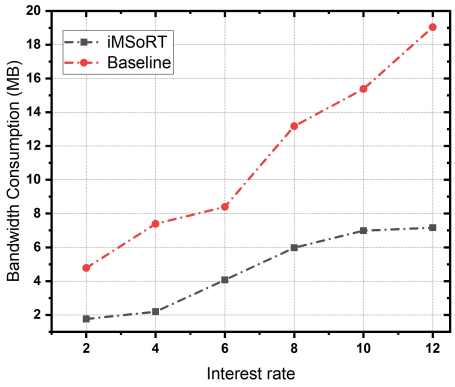
5) **Task Offloading to cloud:** The analysis of the number of MS-centric requests offloaded to the cloud as a function of Interest frequency is shown in Fig. 10. As illustrated, in all schemes, the number of requests forwarded to the cloud increases with Interest frequency. However, iMSoRT significantly reduces cloud offloading compared to $DBPM$ and SCD. This improvement is attributed to the proposed confined computing strategy, which maximizes the use of in-network computing resources to provide proximate computations, thereby avoiding unnecessary cloud routing.

   In contrast, both $DBPM$ and SCD employ blind transmission of requests to specific compute nodes—namely, the cluster-head node in $DBPM$ or the high-social-weight node in SCD—without considering current resource conditions. This approach leads to resource bottlenecks, forcing the compute node to offload requests to the cloud and resulting in high backhaul traffic.

6) **Bandwidth Consumption:** The bandwidth utilization of the proposed scheme compared with the benchmark schemes is presented in Fig. 11(a) and Fig. 11(b). To evaluate performance, we varied the hop count and Interest frequency with step sizes of 1 and 2, respectively. The results clearly demonstrate that iMSoRT outperforms the benchmark schemes, reducing bandwidth utilization by approximately threefold in both cases. This improvement is attributed to iMSoRT's ST, powered by CNN and $YOLO$ algorithms and positioned between the caching and forwarding modules of the NDN stack. The ST



(a)



(b)

Fig. 11: Bandwidth utilization

performs data distillation on large volumes of data before forwarding, extracting and transmitting only semantically meaningful content to the consumer. This process significantly reduces data size by removing redundant and irrelevant details, thereby decreasing bandwidth consumption across the network.

In contrast, both the baseline and conventional NDN forwarding schemes blindly forward all requested data to the consumer without considering data semantics, resulting in higher bandwidth utilization.

5 iMSoRT Computational and Space Complexity Analysis

The computational and space complexities of iMSoRT and the benchmark schemes are analyzed in terms of both time and space, as outlined below.

The time complexity of iMSoRT primarily depends on the number of microservices involved in resolving interdependencies. The worst-case scenario can be expressed as $O(M^2)$, where M denotes the number of microservices. Additionally, each microservice requires a certain execution time, contributing a linear time complexity of $O(M)$. Another factor is the management of valid PIT timers, which depends on the length of the dependency chain. This introduces an additional complexity of $O(D)$, where D represents the number of dependent microservices in a

sequence. Moreover, since microservice-based requests may traverse multiple hops, the hop-related latency adds a component of $O(H)$. Combining these factors, the overall time complexity of the iMSoRT operation can be approximated as $O(M^2 + D + H)$.

In comparison, benchmark schemes such as DBPM, which use Cluster Position Location Awareness (CPLA) with bisection sensing, incur a time complexity of $O(N^2 + \log R)$, where N is the number of Road Side Units (RSUs) and R is the communication radius. DBPM also incorporates a cluster-based routing algorithm, which introduces intra-cluster and inter-cluster complexities of $O(N^2 + V \times M)$ and $O(N^2 + M + H)$, respectively, where V is the number of vehicles. Thus, the overall time complexity of DBPM can be stated as $O(N^2 + V \times M)$. Another benchmark, the SCD scheme, employs social behavior-based metrics for content delivery. Although the time complexity of iMSoRT may be higher than that of DBPM and SCD, this is justified by its ability to handle more complex functionalities, particularly in managing dependent microservices.

The space complexity of the iMSoRT scheme is expressed as $O(M^2 + R)$, where M is the number of microservices and R is the number of active requests. The proposed lFIB mechanism performs tasks such as tracking available resources and monitoring node health status, resulting in an additional space complexity of $O(K \times (S + M))$, where S is the number of RSU servers and K denotes the types of resources being monitored. These values represent realistic upper bounds for practical deployment. For comparison, the DBPM scheme's space complexity includes several components: the CPLA algorithm $O(N)$, Kalman filter prediction $O(1)$, cluster routing $O(V \times M + C)$, and inter-cluster communication $O(C + V)$, where N is the number of RSUs, V the number of vehicles, M the number of cluster members, and C the number of clusters. The SCD scheme exhibits a higher space complexity, given by $O(N^2 + N \times (A_I + C_n) + N_c)$, where N is the number of vehicles, A_I the average number of Interests, C_n the number of content items, and N_c the number of communities. Due to its quadratic growth with the number of vehicles and communities, SCD demonstrates greater spatial overhead compared to iMSoRT and DBPM.

Overall, the iMSoRT scheme presents a computationally efficient and scalable approach for managing complex microservice dependencies. By integrating semantic-aware data transmission and dynamic resource allocation, it offers significant functional advantages in the context of vehicular edge networks.

6 Conclusion

This paper presented MAKS, an efficient knowledge sharing framework designed to provide timely knowledge to consumer vehicles in highly dynamic vehicular environments. To address the challenges of mobility, we introduced MaFIB-assisted knowledge forwarding for guided and reliable data delivery in frequently changing topologies. Furthermore, novel upstream and downstream recovery mechanisms were devised to sustain communication during unforeseeable path disruptions, thereby minimizing redundant transmissions, optimizing network resource utilization, and improving overall QoS. Simulation results demonstrate that MAKS achieves a KDR exceeding 85%, RSD by over 55%, and decreases bandwidth utilization by nearly 50% compared to benchmark schemes.

For future work, we plan to develop a testbed using robotic vehicles equipped with Raspberry Pi devices to evaluate the real-world performance of the proposed framework. Additionally, we are implementing the NDN protocol in MS vanets simulator by incorporating 5G NR as underlying Data link layer and evaluating the proposed work on large scale more realistic scenarios.

Acknowledgments

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No.2022R1A2C1003549).

References

[1] J. Posner, L. Tseng, M. Aloqaily, and Y. Jararweh, "Federated learning in vehicular networks: Opportunities and solutions," *IEEE Network*, vol. 35, no. 2, pp. 152–159, 2021.

[2] C. Liu, K. Liu, H. Ren, X. Xu, R. Xie, and J. Cao, "Rtds: real-time distributed strategy for multi-period task offloading in vehicular edge computing environment," *Neural Computing and Applications*, pp. 1–15, 2021.

[3] Z. Ning, K. Zhang, X. Wang, L. Guo, X. Hu, J. Huang, B. Hu, and R. Y. K. Kwok, "Intelligent edge computing in internet of vehicles: A joint computation offloading and caching solution ," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 4, pp. 2212–2225, 2021.

[4] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Deep learning empowered task offloading for mobile edge computing in urban informatics ," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7635–7647, 2019.

[5] W. Fan, Y. Zhang, G. Zhou, and Y. Liu, "Deep reinforcement learning-based task offloading for vehicular edge computing with flexible rsu-rsu cooperation," *IEEE Transactions on Intelligent Transportation Systems*, 2024.

[6] X. Hou, J. Wang, Z. Fang, Y. Ren, K.-C. Chen, and L. Hanzo, "Edge intelligence for mission-critical 6g services in space-air-ground integrated networks," *IEEE Network*, vol. 36, no. 2, pp. 181–189, 2022.

[7] K. Fu, W. Zhang, Q. Chen, D. Zeng, and M. Guo, "Adaptive resource efficient microservice deployment in cloud-edge continuum," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 8, pp. 1825–1840, 2022.

[8] X. Hou, Z. Ren, J. Wang, W. Cheng, Y. Ren, K.-C. Chen, and H. Zhang, "Reliable computation offloading for edge-computing-enabled software-defined iov," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7097–7111, 2020.

[9] M. A. U. Rehman, M. Salah ud din, S. Mastorakis, and B.-S. Kim, "Foggyedge: An information-centric computation offloading and management framework for edge-based vehicular fog computing," *IEEE Intelligent Transportation Systems Magazine*, vol. 15, no. 5, pp. 78–90, 2023.

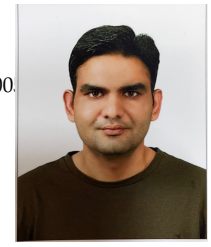
[10] M. Amadeo, C. Campolo, and A. Molinaro, "Crown: Content-centric networking in vehicular ad hoc networks," *IEEE Communications Letters*, vol. 16, no. 9, pp. 1380–1383, 2012.

[11] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[12] M. A. U. Rehman, R. Ullah, B.-S. Kim, B. Nour, and S. Mastorakis, "Ccic-wsn: An architecture for single-channel cluster-based information-centric wireless sensor networks," *IEEE Internet of Things Journal*, vol. 8, no. 9, pp. 7661–7675, 2021.

[13] M. W. Al Azad, S. Shannigrahi, N. Stergiou, F. R. Ortega, and S. Mastorakis, "Cledge: A hybrid cloud-edge computing framework over information centric networking," in *2021 IEEE 46th Conference on Local Computer Networks (LCN)*. IEEE, 2021, pp. 589–596.

[14] M. Salahuddin, M. A. u. Rehman, B. S. Kim *et al.*, "Cfec: An ultra-low latency microservices-based in-network computing framework for information-centric iovs," *Authorea Preprints*, 2023.
[15] M. Salah Ud Din, M. Atif Ur Rehman, M. Imran, and B. S. Kim, "Evaluating the impact of microservice-centric computations in internet of vehicles," *Journal of Systems Architecture*, vol. 150, p. 103119, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762124000
[16] M. Imran, M. N. Ali, M. S. U. Din, M. A. U. Rehman, and B.-S. Kim, "An efficient communication and computation resources sharing in information-centric 6g networks," *IEEE Internet of Things Journal*, pp. 1–1, 2024.
[17] M. Imran, M. S. U. Din, M. A. U. Rehman, and B.-S. Kim, "Mia-ndn: Microservice-centric interest aggregation in named data networking," *Sensors*, vol. 23, no. 3, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/3/1411
[18] C. Scherb, B. Faludi, and C. Tschudin, "Execution state management in named function networking," in *2017 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 2017, pp. 1–6.
[19] M. Król and I. Psaras, "Nfaas: Named function as a service," in *Proceedings of the 4th ACM Conference on Information-Centric Networking*, 2017, pp. 134–144.
[20] M. A. U. Rehman, M. Salah ud din, S. Mastorakis, and B.-S. Kim, "Foggyedge: An information centric computation offloading and management framework for edge-based vehicular fog computing ," *arXiv preprint arXiv:2304.10204*, 2023.
[21] D. Kondo, T. Ansquer, Y. Tanigawa, and H. Tode, "Resource breadcrumbs: Discovering edge computing resources over named data networking," *IEEE Transactions on Network and Service Management*, 2024.
[22] S. Rizwan, M. Ahmed, G. Husnain, M. Khanam, and S. Lim, "Macpe: Mobility aware content provisioning in edge based content-centric internet of vehicles," *IEEE Access*, 2024.
[23] S. Zhou, M. Cui, R. Hou, and L. Zhao, "Data packet forwarding strategy based on vehicle tracking in named data networking," in *2019 2nd International Conference on Hot Information-Centric Networking (HotICN)*, 2019, pp. 66–71.
[24] R. Hou, S. Zhou, Y. Zheng, M. Dong, K. Ota, D. Zeng, J. Luo, and M. Ma, "Cluster routing-based data packet backhaul prediction method in vehicular named data networking," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 3, pp. 2639–2650, 2021.
[25] J. M. Duarte, T. Braun, and L. A. Villas, "Mobivndn: A distributed framework to support mobility in vehicular named-data networking," *Ad Hoc Networks*, vol. 82, pp. 77–90, 2019.
[26] X. Wang and X. Chen, "Social attributes-based content delivery for sparse vehicular content-centric network," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 12, pp. 14 406–14 414, 2023.
[27] S. Mastorakis and A. Mtibaa, "Towards service discovery and invocation in data-centric edge networks," in *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE, 2019, pp. 1–6.
[28] S. A. Khan and H. Lim, "Real-time vehicle tracking-based data forwarding using rls in vehicular named data networking," *IEEE Transactions on Intelligent Transportation Systems*, 2024.
[29] S. Zhou, M. Cui, R. Hou, and L. Zhao, "Data packet forwarding strategy based on vehicle tracking in named data networking," in *2019 2nd International Conference on Hot Information-Centric Networking (HotICN)*. IEEE, 2019, pp. 66–71.
[30] S. Wan, S. Ding, and C. Chen, "Edge computing enabled video segmentation for real-time traffic monitoring in internet of vehicles," *Pattern Recognition*, vol. 121, p. 108146, 2022.
[31] M. Haklay and P. Weber, "Openstreetmap: User-generated street maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.
[32] 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.
[33] "Ndncsim: A microservices based compute simulator for ndn," 2021. [Online]. Available: https://github.com/11th-ndn-hackathon/ndncompute-simulator
[34] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

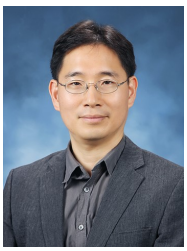


Muhammad Salah ud din is a Postdoctoral Research fellow at the Communication Systems Department, EURECOM - Campus SophiaTech France. He earned a Ph.D. degree in Electronics and Computer Engineering from BCN Lab Hongik University, South Korea in August 2023. His major interests are in the field of Autonomous vehicular communication, Knowledge centric networking, Edge/Fog/Cloud Computing, Named data networking, IoT, and 5G/6G communication.

Muhammad Nadeem ALi is currently pursuing the Ph.D. degree in Computer Engineering with the Department of Electronics and Computer Engineering in Graduate School, Hongik University, South Korea. His major interests are in the field of Machine learning for Wireless networks, Deep learning, Internet of things (IOT), Named Data Networking.

Ghulam Musa Raza is currently pursuing the Ph.D. degree in Computer Engineering with the Department of Electronics and Computer Engineering in Graduate School, Hongik University, South Korea. His major interests are in the field of Natural Language Processing, Internet of things (IOT), Information-Centric Networking and Named Data Networking.

Muhammad Imran is a Postdoctoral Fellow at the Shenzhen Municipal Key Laboratory of AIoT Communications, Harbin Institute of Technology, Shenzhen, China. He earned his Ph.D. in Software and Communication Engineering from Hongik University, South Korea, in February 2025. His research focuses on wireless communication, IoT, edge/cloud computing, ICN/NDN, future networks, AI-driven networking, and reinforcement learning. He received the Best Researcher Award in 2022–2023 and 2024–2025.



Byung-Seo Kim (M'02-SM'17) received the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Florida, in 2001 and 2004, respectively. From 2005 to 2007, he was with Motorola Inc., Schaumburg, IL, USA, as a Senior Software Engineer in networks and enterprises. He is a Professor at the Department of Software and Communications Engineering, at Hongik University, South Korea.