Journal of Systems Architecture 150 (2024) 103119

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Evaluating the impact of microservice-centric computations in internet of vehicles

Muhammad Salah Ud Din [a], Muhammad Atif Ur Rehman [b], Muhammad Imran [a], Byung Seo Kim [a,*]

[a] Department of Software and Communications Engineering, Hongik University, Sejong City 30016, Republic of Korea
[b] Department of Computing & Mathematics at Manchester Metropolitan University, United Kingdom

ARTICLE INFO

Keywords:
Information-centric networking
Microservices
Vehicular networks
Edge
Cloud
Fog
IoT

ABSTRACT

This paper presents the design and implementation of a hierarchical and multilayered computing framework, specifically tailored to analyze the benefits of Named Data Networking (NDN) enabled microservice-centric in-network computations in autonomous vehicular networks. The proposed framework comprises three layers: an autonomous vehicular network layer, an Edge computing layer, and a centralized Cloud.Modifications were made to the vanilla NDN codebase to facilitate communication between the vehicular network layer, Edge computing and Cloud layer, utilizing the C++ Boost Asio library and IP tunneling mechanism. A microservices-based driver assistance application, consisting of various heterogeneous microservices, was emulated in .NET Core and deployed on different Edge computing terminals and the Cloud. Additionally, RESTful APIs have been developed to enable the vehicular network layer and the physical Edge layer to offload microservice-centric computation requests and retrieve the corresponding computational outcomes. The Entity framework is employed to ensure proper tracking and management of these requests. An HTML-based user interface has also been developed for a visual representation of the request pattern. Extensive testbed experiments reveal that the proposed system significantly optimizes bandwidth consumption, reduces latency, and increases the computed results delivery ratio when compared to conventional monolithic systems.

1. Introduction

The swift advancement in IoT computing, processing, sensing, and wireless communication capabilities, combined with their integration into vehicular onboard units (OBUs), has paved the way for the evolution of mission-critical systems, including autonomous vehicular applications, AR/VR [1], driver assistance [2], object detection [3], and assisted lane change [4]. Such applications demand extensive computing, processing, communication, and storage resources that exceed the capabilities of a single vehicle. Furthermore, these data and compute-intensive applications are bound by stringent latency and reliability requirements, essential for timely decision-making and ensuring a seamless user experience.

To efficiently address application demands, consumer vehicles typically delegate their computation requests to remote Cloud servers [5]. However, traditional centralized Cloud computing introduces significant delays due to extended transmission distances. Moreover, transferring large volumes of content to the cloud can exacerbate service latency and lead to congestion in the network.

To address these challenges, burgeoning Edge computing technology aiming to enable proximate computations was introduced. The consumer vehicles offload their intensive computation requests to nearby Edge computing nodes (e.g., Roadside Units (RSUs)) [6,7] rather than transferring them to the Cloud. This mechanism significantly reduces the chances of network congestion and service latency compared with the Cloud offloading. Along with the intrinsic benefits, these Edge computing nodes accompany great challenges. The RSU Edge servers have less computational capability compared with the Cloud. Therefore, during peak traffic hours, an RSU mounted in crowded places may not solely fulfill the consumer's intensive compute-oriented requests due to deficient resource. This situation compels the underlying RSU to offload the computations towards the Cloud at the expense of high latency and bandwidth consumption.

Conventional vehicular applications usually follow a monolithic application (discussed in detail in Section 2.1) development architecture where all the application components are tightly coupled in a single monolith. Due to tight coupling, a single component migration among the computing terminals (e.g., vehicles, Edge, or Cloud, etc.) requires

* Corresponding author.
E-mail address: jsnbs@hongik.ac.kr (B.S. Kim).

https://doi.org/10.1016/j.sysarc.2024.103119
Received 20 September 2023; Received in revised form 29 December 2023; Accepted 21 March 2024
Available online 26 March 2024

a complete monolith to be migrated, resulting in unnecessary delays, excessive bandwidth utilization, and network congestion.

To address this challenge, the microservice (MS) architecture emerged as a potential solution that splits the application into small, manageable, loosely coupled, and autonomous services. These services require less computational resources, promote low latency and fast migration with less bandwidth utilization, and require low maintenance costs. Additionally, loose coupling among the microservices ensures that a faulty microservice never lets down the entire application, minimizing computation losses and delayed results reception.

At present, vehicular communication is carried out via an address-centric mechanism, i.e., TCP/IP [8,9]. However, it is very difficult to achieve seamless communication in a dynamic vehicular environment through IP due to several reasons.

- Assigning unique IP addresses to vehicles requires infrastructural support; however, vehicular networks fall into the category of ad-hoc networks, which do not have any infrastructure support.
- It is not viable to seek a particular vehicular address to initiate communication in a continuously changing environment.
- Establishing and retaining the communication path between participating vehicles in a dynamic environment is crucial.

To handle the ingrained shortcomings of IP in dynamic vehicular environments, Information-Centric Networking (ICN) [10] (details in Section 2.2), and its fruition Named Data Networking (NDN) [10], has emerged as a potential solution. Unlike the conventional IP, the ICN/NDN architecture mainly centers on consumers' required content, regardless of the location of the actual content producer [11,12]. Moreover, ICN strips content from its origination and provides in-network caching, request aggregation, and multiparty communication [13].

Bridging the vehicular network with Edge and cloud computing terminals via NDN as underlying communication paradigm in order to carry out microservices-centric in-network computations to achieve onsite computations with miniaturized resource utilization in vehicular environment is quite challenging. To the best of our knowledge, a comprehensive solution that enables the analysis of the effectiveness of microservices-centric in-network computations, taking into consideration the collaboration of aforementioned entities is still missing.

Leveraging the intrinsic benefits of microservices centric computations and the potential of NDN as an underlying communication paradigm, this paper develops a microservices-centric in-network computing testbed[1] system aimed at analyzing the benefits of microservice-centric in-network computations in terms of efficient computations with minimal computational and network resource consumption. The framework encompasses: (1) an autonomous vehicular network simulated in the ndnSIM environment, (2) physical Edge computing nodes hosting a subset of the application's microservices, and (3) a Cloud server responsible for hosting the complete suite of the application's microservices. A microservices-centric naming schema is developed for name-based communication within the vehicular network, while RESTful APIs have been devised for external communications, such as from ndnSIM to Edge, inter-Edge, and Edge to Cloud server. Moreover, the Entity framework has been employed to store request logs, enabling traffic management entities to scrutinize consumer request patterns directed at specific microservices during particular time frames

The main contributions of our work can be summarized as follows.

1. A multi-layered framework has been designed that integrates a vehicular network layer, a Physical Edge computing layer, and a centralized Cloud layer to perform microservices-centric in-network computations and ensure timely result delivery to the consumer vehicles.
2. The proposed work modified the NDN codebase and incorporated the C++-based Boost ASIO library to enable communication between the vehicular network and Physical Edge devices. This allows consumer vehicles to offload Microservice-centric computation requests to Physical Edge servers and receive the corresponding results.
3. We adopted the microservices architecture and emulated a driver assistance application comprised of various heterogeneous microservices in *.NET core* and deployed on Edge servers and Cloud servers, furthering the objective of microservices-centric in-network computations.
4. The proposed work developed RESTful APIs in *.NET core* to enable vehicular network to Edge communication, inter-Edge communication, and Edge–Cloud communication.
5. The entity framework was employed to log consumer requests enabling traffic monitoring entities to evaluate consumer request patterns during peak and off-peak hours to facilitate informed decision-making.
6. Prototype implementation and extensive testbed experimentation revealed that microservices-centric in-network computing effectively optimizes bandwidth consumption, minimizes latency, and augments the computation satisfaction rate.

The rest of the paper is structured as follows: Section 2 provides background and related work. Section 3, presents the proposed framework, providing a detailed operational overview of its architecture and design. Section 4 is devoted to the performance evaluation and finally, Section 5 concludes the paper, summarizing findings and discussing their implications for future research and practice.

2. Background and related work

2.1. Microservices and monolithic architecture

The monolithic architecture, depicted in the left block of Fig. 1, is a traditional software development model where the entire application is developed as a single, unified, and self-contained unit. A monolithic system is a large computing system comprised of a bulky code base that couples all the business logic, presentation, and database together under a single monolith. In such systems, a single faulty component can compromise the entire application's operation due to the strong inter dependency among modules. Additionally, monolithic systems lack scalability; even minor changes in the codebase may necessitate updates to the entire application. Apart from the application perspective, migrating a particular component from one network to another involves moving the complete monolith, resulting in excessive network resource consumption. Given the continuously fluctuating channel conditions and delay-sensitive nature of vehicular applications, monolithic architectures may encounter performance issues including delays, high latency, and network congestion.

To address these challenges, the microservices (MS) architecture, illustrated in the right block of Fig. 1, has emerged as a viable alternative due to its advantages, such as reduced communication and computation overhead, optimal resource utilization, low bandwidth needs, and decreased maintenance costs. Unlike monolithic architecture, the microservices architecture decomposes the application into several independent microservices (as shown in Fig. 1), each capable of executing autonomously. These microservices perform distinct tasks, thereby diminishing the complexity of redeployment and maintenance while facilitating straightforward modifications.

Each microservice is designed to perform a specific function, requires fewer computational resources, and communicates with others (when necessary) via well-defined APIs, such as RESTful, SOAP (Simple Object Access Protocol), or RPC (Remote Procedure Calls) and enables effective migration with minimum bandwidth consumption and minimizes the chances of congestion and delays in the network.

[1] The terms "framework" and "testbed" are used interchangeably throughout the paper to denote our proposed work.

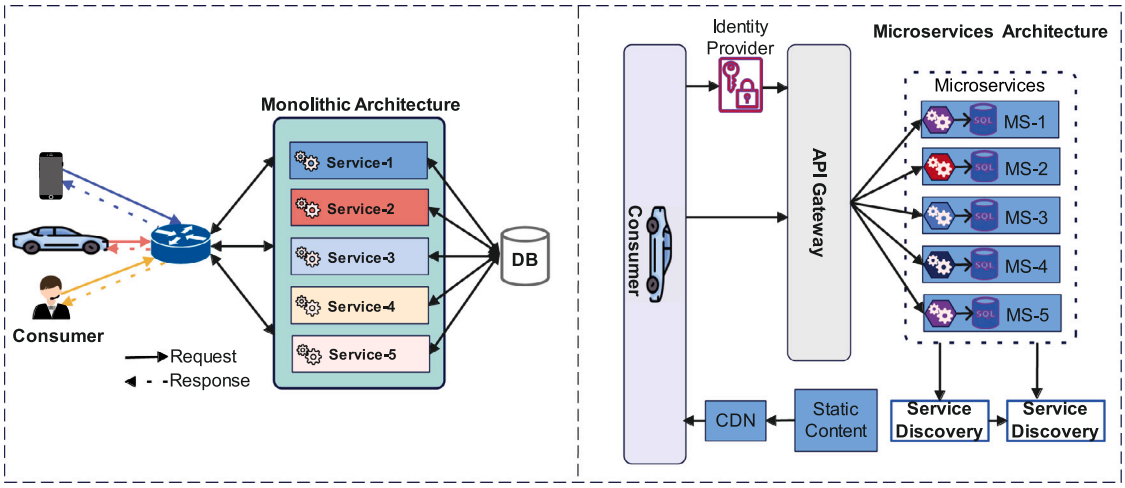

Fig. 1. Monolithic and microservices system architecture.

Moreover, deploying microservices at the vehicular Edge and Cloud can significantly enhance handling application-type requests, ensuring minimal delays and improving application QoS/QoE. These features collectively enable latency-sensitive applications to receive their required computations promptly, offering a marked improvement over the monolithic architecture.

2.2. ICN for the internet of vehicles

The ICN and its realization NDN effectively addresses the shortcoming of IP, especially in a vehicular environment. With the name-based and content-centric communication philosophy, in-network caching, requests aggregation, and content-level security, NDN effectively handles intermittent connectivity and high mobility issues of vehicular networks by decoupling the services or content from any specific location.

The detailed communication procedure is depicted in Fig. 2 where the consumer (e.g., C1, C2) vehicles seek the required data or computations by forwarding an Interest packet (I_pkt) in its vicinity. Each receiving vehicle (e.g., R1) maintains a pending Interest table (PIT) to keep track of forwarded Interest packets. Upon I_pkt reception, the receiving vehicle inquiries about the existence of the same-named I_pkt entry in its PIT. Interest aggregation is performed (shown by R1 in the figure) if found. If a PIT entry is not found, the content store (CS) check is performed for the requested data. If the requested content is found in CS, the data packet (D_pkt) is generated and returned back to the consumer in the reverse path. In case of Data unavailability in CS, the request is forwarded to the potential upstream faces via forwarding information base (FIB) lookup. The D_pkt follows the chain of PIT entries maintained by the vehicular nodes in the reverse path to reach back to the consumer. If no PIT entry exists against an incoming D_pkt packet, the D_pkt is unsolicited and is discarded.

2.3. Related work

Several state-of-the-art techniques have been devoted to the literature aiming to provide timely results for compute-intensive and latency-sensitive vehicular applications. Table 1 presents a comparison of various state-of-the-art NDN-based schemes developed for vehicular networks based on several metrics such as Name based communication, MS Naming, Layered architecture and Edge/Cloud consideration whereas, a comprehensive review of the current state-of-the-art Vehicular Named Data Networking schemes is provided below.

Zhang et al. [14] developed a cloud-based mobile edge computing and computation offloading mechanism to improve the computations

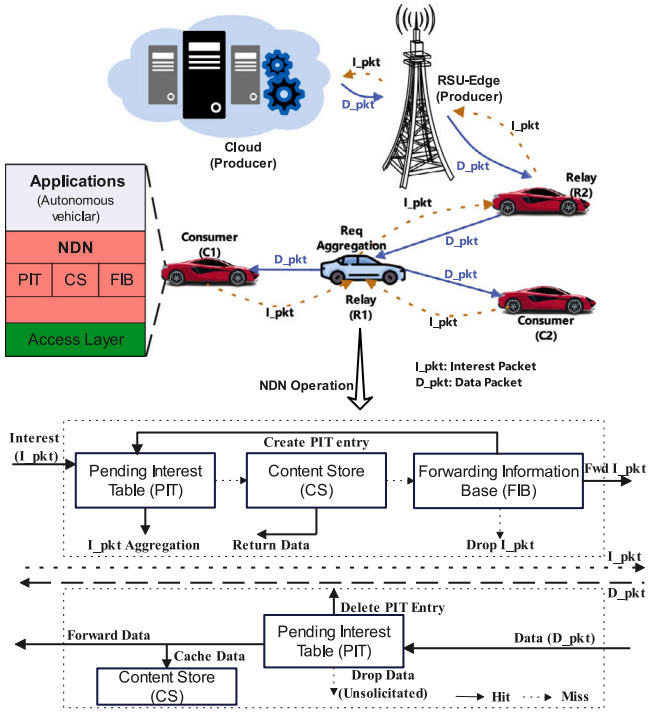

Fig. 2. Vehicular named data networking architecture.

transfer strategy considering the task execution time and vehicles' mobility. A three-layer vehicular fog computing (VFC) framework was developed by Ning et al. [15] to reduce latency in distributed vehicular scenarios. To do so, the authors leveraged both parked and mobile vehicles and formulated a VFC-enabled computation offloading strategy as an optimization problem. Sun et al. [16] proposed an efficient task scheduling scheme by considering the unstable and heterogeneous computing resources to improve computation efficiency in mobile vehicular networks. Mekki et al. [17] integrated fog computing with MEC and developed a robust four-layered architecture to provide computational resources in close vicinity to users. Hou et al. [18] adopted software-defined networking principles and combined mobile edge with fixed edge computing principles to meet the latency requirements of delay-intolerant and computationally intensive vehicular applications.

Table 1
Related work comparison.

Ref	Name based comm.	MS naming	Layered arch	Edge /cloud
[14]	×	×	×	×
[19]	×	×	✓	×
[15]	×	×	×	✓
[20]	✓	×	×	×
[17]	×	×	×	✓
[18]	×	×	×	✓
[21]	✓	×	×	×
[22]	✓	×	×	×
[23]	×	×	✓	✓
Proposed	✓	✓	✓	✓

A mobility-aware service migration scheme was proposed by Chen et al. [19] to address the problem of service migration in MEC due to user mobility and dynamic network conditions. It introduces the mobility-aware service migration (MSM) scheme, which minimizes system delays by combining user's historical Wi-Fi data, association pattern analysis, and deep reinforcement learning. Samanta et al. [24] proposed an innovative auction-based mechanism designed to effectively manage resource distribution among microservices, ensuring resource optimization. Samanta and Tang [23] proposed a dynamic microservice scheduling scheme for mobile edge computing (MEC) to improve the performance of IoT applications in dynamic network conditions. Their work mainly focused on maximizing energy efficiency and Quality-of-Service while reducing the total network delay and cost. Another state-of-the-art scheme, named RAISE, a novel resource-agnostic microservice offloading scheme for mobile IIoT devices, was proposed by Samanta et al. [25]. RAISE efficiently estimates and manages the fluctuating resource requirements of IIoT devices, enhances network throughput and Quality-of-Service. Through experimental results, RAISE showed superior performance over benchmark schemes in terms of cost and reliability.

Wang et al. [20] proposed Cost-efficient data retrieval based on the integration of Vehicular Cloud and NDN (CDVRC) to optimize the data retrieval process with miniaturized network cost. The CDVRC provides reliable data delivery via VC while the NDN was utilized as the underlying communication protocol for efficient content retrieval. CDVRC requires an additional network infrastructure to support both VC and NDN which may not be feasible for the limited resources networks.

Ahmed et al. [22] proposed Codie: controlled data and interest evaluation in vehicular named data networks to alleviate the frequent Interest and Data disseminations in the vehicular networks by using hop count information. Simulation results showed that CODIE significantly improved the network performance by minimizing the overall packet transmissions in the network.

3. Proposed microservice centric in-network computations framework

In this section, to elucidate the features of the proposed work, we first provide the motivation behind our proposed scheme. Subsequent to that, a detailed explanation of the proposed work is provided.

3.1. Motivation

As discussed earlier, the internet of Vehicles (IoV) involves various autonomous vehicles with heterogeneous onboard resources, significantly contributing to Internet traffic. These vehicles are equipped with communication and computation resources, supporting intelligent traffic management, interactive vehicular applications, and efficient vehicle control within the IoV paradigm. Typically, these applications adopt a monolithic development architecture, necessitating substantial computational resources for timely responses to achieve effective operation. However, due to the limited computational capacity and the constantly changing vehicular environment, a single vehicle may never perform its operations promptly, leading to performance degradation. While remote cloud offloading can alleviate computational burdens, it often introduces unacceptable delays for latency-sensitive applications.

To address the limitations of cloud computing, Edge computing enabled Roadside Units (RSUs), has been introduced to bring cloud resources closer to consumer vehicles. This approach enables vehicles to offload tasks within their vicinity. However, during peak hours, an influx of compute-intensive requests from various smart vehicular applications can overwhelm the resources of a single RSU, causing bottlenecks. Furthermore, the conventional address based TCP/IP model, to forward Edge offloaded requests to the cloud servers often exacerbates network delays and congestion in dynamic environments due to frequent connectivity issues.

The integration of MS at the application layer, edge cloud computing at the compute layer, and NDN as the network layer brings forth various new challenges. Among them, to the best of our knowledge, a comprehensive simulation/experimentation framework or testbed is missing to carryout an effective evaluation of the proposals involving the aforementioned technologies. The primary motivation of our proposed work is to develop a testbed framework comprised of autonomous vehicles, edge computing nodes, and cloud infrastructure, coupled with NDN as the underlying communication model. This is to analyze the benefits of NDN enabled MS-centric in-network computations, in terms of communication and computation resources utilization, computation request satisfaction time, and bandwidth utilization, compared with a monolithic system. To this end, the proposed work modifies the NS3 and NDNsim codebases and develops mechanisms for managing nodes' resources which include providing resources to the node, determining available and unavailable resources, assigning available resources to incoming requests, and freeing resources upon completion of requested computations. In addition, MS-centric interest and data packets have been developed to offload the requests to compute nodes e.g., vRSU, Edge or Cloud. These nodes are specifically designed to host microservices based vehicular application developed in .NET Core, thereby facilitating the requested microservice computations. It should be noted that the framework can be utilized by researchers working in the field of in-network computation as a baseline framework to test their proposals.

3.2. Proposed framework

The proposed multilayered framework architecture, as depicted in Fig. 3, is composed of three primary components:

1. **Vehicular network:** The vehicular network consists of a modern autonomous vehicular network and a virtual RSU (vRSU) within the ndnSIM environment. Autonomous vehicles send compute-intensive microservices-centric requests to the vRSU. The vRSU either executes these requests onsite or offloads them to higher layers depending on resource availability and finally returns the results to the requesting vehicle.
2. **Physical Edge servers:** Equipped with high computational resources, physical Edge servers host microservices and perform specific computations in response to requests from vRSUs.
3. **The Cloud server:** The Cloud server hosts the complete suite of the application's microservices. It either processes microservice operations based on the input received or forwards the microservice code to the requesting Edge server for execution.

A detailed component-level description of each layer is provided as follows.

3.3. Vehicular network

In the proposed framework, an autonomous vehicular network is developed within the ndnSIM simulator, comprising both mobile

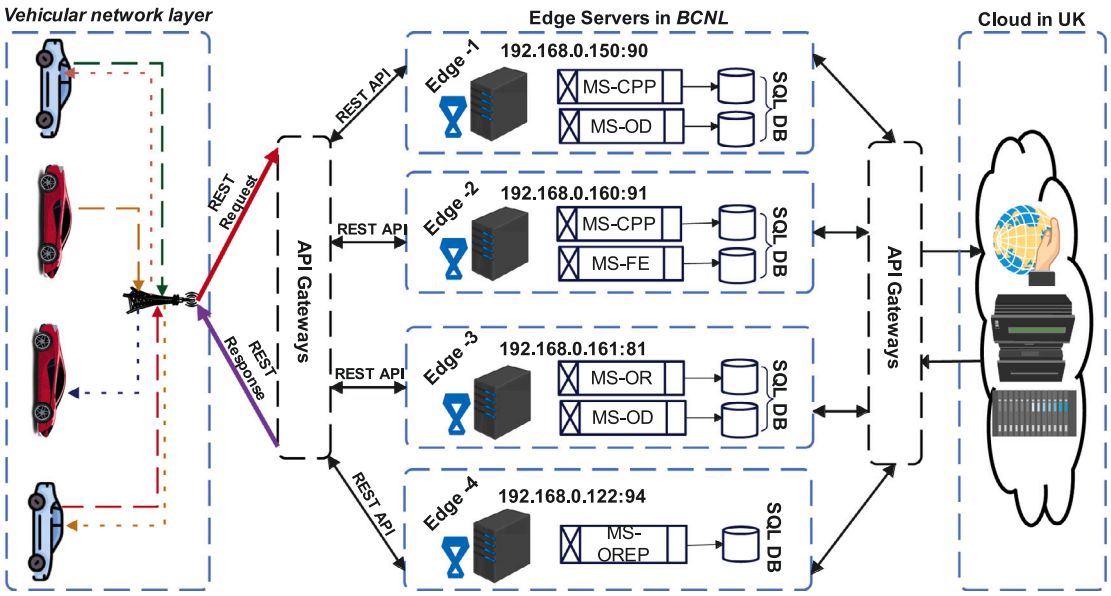

Fig. 3. MS-centric in-network computation framework architecture.

vehicular nodes and static vRSUs, as illustrated in Fig. 3. The framework modifies the vanilla NDN codebase to develop a microservices-centric naming schema which enables vehicular nodes moving at different velocities to forward Named-based microservices-centric computation requests to the corresponding vRSU. In addition, a mechanism for managing node resources has been developed, which includes allocating resources to nodes, identifying available and unavailable resources, assigning available resources to incoming requests, and releasing resources after completing computations.

Upon receiving a request the vRSU either fulfills it locally, depending on the availability of resources, microservice code, or cached results. If adequate resources are not available, the vRSU offloads the requests to the Physical Edge server for computation. This enhancement involves modifying the *Producer.hpp* and *Producer.cpp* files in the Vanilla NDN codebase and implementing the C++ based BOOST ASIO library, which hands over the request to the physical Edge machine using RESTful APIs.

3.3.1. C++ BOOST ASIO

The Boost Asio is an open-source cross-platform network programming library that provides an asynchronous input/output model using C++ programming language to achieve networking operations. The library is used for handling network programming tasks, such as developing servers (i.e. physical Edge servers) and clients (i.e.,vRSUs) processing data asynchronously. Due to the its efficiency and ability to simplify complex network programming tasks, the proposed work utilize this library to enable vRSUs to offload the computation requests to the physical Edge servers in order to achieve the requested computations.

In order to enable the computation offloading from the vRSU to the Physical Edge server we implemented the TCP client–server model where the vRSU acts as a client and the Physical Edge as a server.

1. **vRSU Configurations:** In the proposed work, the *Producer.hpp* and *Producer.cpp* files of the NDN protocol were modified to implement the functionalities of *boost asio*. We created the *io_service* object from the Asio library (i.e., *boost::asio::io_service*) to create the socket. After successfully creating the socket, we establish a connection to the Physical Edge server using its IP addresses (i.e., *192.168.0.150, 192.168.0.160, 192.168.0.161, 192.168.0.122*) and port numbers (*90, 91, 81, 94*), as illustrated in Fig. 3. The vRSU offloads Microservice-centric computation requests to the physical Edge server via RESTful APIs using IP tunneling (detailed in Section 3.7). The Edge server responds with the computed results to the vRSU, which uses the *boost::asio::read* function to fetch the computed results. The vRSU then transform these results into NDN Data packets and offload to the consumer vehicle via an ad-hoc interface.
2. **Edge Server Configurations:** Each Edge server is configured with an IP address to handle client requests for microservices-centric computations. It listens for connections and responds accordingly. To facilitate the reception of client requests and the delivery of computed results, the proposed framework employs a *tcp socket* for communication. Two functions from *boost::asio*, namely *boost::asio::read_until* and *boost::asio::write*, are implemented for read and write operations, respectively. Additionally, *boost::asio::buffer* is utilized to create a data buffer for the communication process.

3.4. Edge servers in BCNL

The proposed work developed an Edge environment comprised of four Physical Edge computers equipped with different computational resources placed in the BCN lab at Hongik University South Korea. Considering the principles of microservices architecture, we used *.NET Core* as the development framework to emulate a smart vehicular application. The application is comprised of various autonomous and independently deployable microservices. Each microservice is deployed on different Edge servers utilizing Internet Information Services (*IIS*). An SQL server database, consisting of log tables, has been created to store user request patterns and relevant information. This database enables traffic monitoring departments to analyze user request patterns at different times of the day, providing valuable insights for traffic management and planning.

A detailed description of microservices in *.NET core*, SQL database, the entity framework, and *IIS* in terms of our proposed system is provided as follows.

1. **Microservices in *.NET core*:** In our proposed framework we emulated a vehicular assistance application in *.NET core* composed of various MS.
   A detailed explanation and operation of each microservices are provided as follows:
   - **Content preprocessing (MS-CPP or MS1):** MS-CPP or MS1 performs raw data manipulation by applying preprocessing operations on input data.

- **Feature extraction (MS-FE or MS2):** MS-FE or MS2 exploits the most discerning features in the captured Data.
- **Object detection (MS-OD or MS3):** MS-OD or MS3 is responsible for identifying the presence of potential objects from the captured visual content.
- **Object representation (MS-ORep or MS4):** MS-ORep or MS4 is responsible for the visual representation of the detected object from the captured scene.
- **Object recognition (MS-OR or MS5):** MS-OR or MS5 identifies the most relevant objects in the captured scene.

2. **SQL Server and Entity Framework:** The proposed framework utilized SQL Server and Entity Framework and develops a database named *MsLogsDB* containing a Logs table to store the incoming and outgoing requests information such as *request Date, MS name, Node Name, Request time, Response time, and Result*. Such information enables the traffic monitoring organization to analyze the consumer's computation request behavior for efficient load-balancing, route planning, and compute-aware service migration in a dynamic and error-prone vehicular environment. The proposed work employs the Entity Framework for interacting with data stored in the database, enabling the execution of CRUD (Create, Read, Update, and Delete) operations. The entity framework provides an extensible and lightweight object-relational mapping (ORM) that enables the developers to add, delete, and/or modify specific components from the ORM as per requirements.
3. **Internet Information Services (*IIS*):** *IIS* is a web server bundled with the Microsoft Windows operating system. It processes and responds to consumer requests efficiently. By integrating *IIS* with the *.NET Core* Hosting bundle, the developed microservices are deployed on Edge servers at various IP addresses such as *192.168.0.150, 192.168.0.160, 192.168.0.161*, and *192.168.0.122*, each operating on port numbers *90, 91, 81*, and *94*, respectively. When a vRSU node receives a consumer's request, it transform using IP tunneling and offload it to the Physical Edge. The IIS at corresponding physical Edge server accepts the request, forwards it to the application for requested computations, receive computations, and subsequently returns the computed results to the vRSU.

3.5. Cloud

The entire suite of application microservices is deployed on a Microsoft Cloud server. The underlying Edge servers offload their computing requests or download the required microservice code from the Cloud, depending on their requirements.

3.6. RESTful APIs: Bridging vehicular networks, physical edge, and the cloud

The Representational State Transfer Application Programming Interface (RESTful API) adheres to a set of rules, protocols, and routines which enable a consumer application to interact with a specific microservice to perform desired operations and receive the computed results. This interaction occurs over the HTTP protocol, which supports various data formats, such as JavaScript Object Notation (JSON), HTML, or plain text. It is to be noted that the proposed framework utilizes the JSON format for sending requests and receiving corresponding results. The rationale is the JSON's language-agnostic nature and its ease of interpretation by both humans and machines.

A detailed description of RESTful APIs is presented as follows: The Microservice-centric computation request from vRSU in the ndnSIM environment is composed of four major components.

- **Endpoint:** The Endpoint locates the MS that a vehicular application wants to access, structured as *"root-endpoint/?"*.
- **Method:** This component is used for sending MS-centric requests and retrieving corresponding computations. The five standard methods include *GET, POST, PUT, PATCH*, and *DELETE*. The proposed framework primarily utilizes the *GET* and *POST* methods for its operations.
- **Headers (–H or –header):** The headers perform authentication and provide information about the content of the body.
- **Data (or body):** It refers to the request sent by the smart vehicular application to the Edge server or from the Edge server to the Cloud. This is done using the –d or –data option with *POST* or *GET* requests.

Following the principles of REST APIs, our Edge server responds with computed results. The main components of the response are as follows:

1. **Status Code:** The status code consists of a three-digit number that indicates request success or failure. For instance, *2XX* codes (e.g., *200*) indicate success, while *400* and *500* signify errors, and *300* series codes indicate URL redirection.
2. **Message Body:** The Edge/Cloud server responds with the computed results of the corresponding microservice request in JSON format, which is then handed over to the requesting consumer. For example, if the consumer requests MS-CPP computations,the Edge server returns the results in JSON format, e.g., *"requestedMs": "MS-CPP", "ComputedResults": "XXXX", "Size": "123KB"*.
3. **Headers:** The response includes headers and other metadata, providing details such as server type, encoding, timestamp, and content type, among others.

3.7. IP tunneling

Since the NDN is not yet widely adopted therefore, to facilitate NDN-based communication between the vRSU and physical edge servers, we employed the concept of IP tunneling where the two ends of the tunnels are NDN nodes that follows NDN principals. This method allows vRSU to communicate with physical edge servers by encapsulating NDN packets within IP packets, enabling their transmission over IP networks using REST APIs. The vRSU first transforms the NDN packets into IP packets using the aforementioned mechanism and then forwards them to the physical edge servers. The physical edge servers are equipped with NDN protocols and follow all NDN principals (i.e., interest aggregation, in-network caching). In addition, the edge server performs computations and return the results using NDN's inherent reverse path mechanism. Upon receiving the data packets, the vRSU converts them back into NDN data packets and hands them over to the consumer. It is to be noted that same mechanism is applied to enable communication between physical Edge servers and Cloud.

3.8. Microservice-centric computation offloading process

The computation offloading mechanism starts when a consumer autonomous vehicle offloads a computation request to a vRSU within its communication range. Upon receiving the request, the vRSU checks the availability of computational resources using our node management mechanism. If adequate resources are available, the vRSU executes the computation and hands over the results to the consumer. In cases where adequate resources are unavailable, the vRSU forwards the computation request to the Physical Edge server.

To facilitate the process, we modified the `ndnSIM/apps/ndn-producer.hpp` and `ndnSIM/apps/ndn-producer.cpp` files and implemented a function `GetComputationsFromEdge (MS, input)`. This function accepts parameters such as the requested microservice name and input data (e.g., captured image or video) and performs offloading to the physical Edge server. As discussed earlier, NDN is not yet widely adopted, to enable NDN-based communication

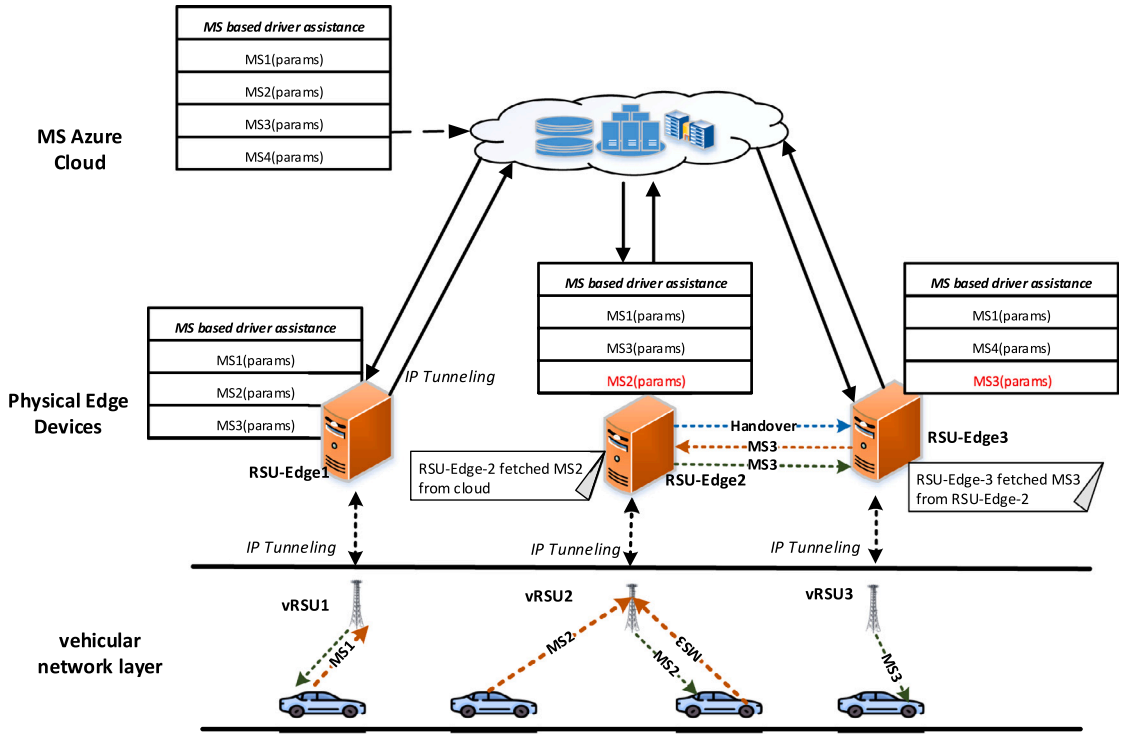

Fig. 4. MS-centric computations offloading and results delivery. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

between the vRSU and physical edge servers, as well as between physical edge servers and the Cloud, we employed IP tunneling where the two ends of the tunnels (e.g., vRSU, Edge servers and Cloud) are NDN nodes that adhere to NDN principles. This mechanism allows vRSUs to communicate with physical edge servers and the physical Edge servers to communicate with the Cloud by encapsulating NDN packets within IP packets.

A glimpse of the computation offloading process from the vehicular network layer, and the response generated by the Physical Edge machines, as well as the Cloud server, is provided as follows.

1. **Vehicular Network Layer to Edge Offloading:** As shown in Fig. 4, the consumer vehicle from the vehicular network layer forwards the *MS1* Interest packet, e.g., `MSTB:/Driver Assistance/MS1?P1, P2`, towards *vRSU1* (indicated by the orange arrow). Here, "Driver Assistance" denotes the autonomous vehicular application, *"MS1"* is the name of a specific microservice, and "*P1, P2*" denotes input parameters. Upon receiving the incoming Interest, the corresponding *vRSU1* inquires about the availability of *MS1* and the required resources to perform the requested computations. Due to the unavailability of adequate resources, the corresponding *vRSU1* utilizes IP tunneling to encapsulate NDN packets within IP packets and offloads computations to the Physical Edge via *RESTful APIs*. The *IIS* at *RSU-Edge1* accepts the request from the *vRSU*, performs computations, and hands over the results via the reverse path. *vRSU1*, upon receiving the results, generates a named-based response and forwards it directly to the consumer vehicle (shown by the green arrow in Fig. 4).
2. **Edge to Cloud Offloading:** As the consumer vehicle leaves the coverage area of vRSU1 and enters the range of *vRSU2*. It forwards the *MS2* request. On receiving the request, due to resource unavailability, the *vRSU2* offloads the request to the *RSU-Edge2* via RESTful APIs. The *RSU-Edge2* inquires about the availability of code and adequate resources. Due to code unavailability, the *RSU-Edge2* first download the missing microservice code from the Cloud performs computations, and return the results to the consumer in the reverse path (shown by the green arrow in Fig. 4).
3. **Edge to Edge Offloading:** Before leaving the *vRSU2*, the consumer forwards the *MS3* Interest packet shown in Fig. 4. The *vRSU2* verifies that the consumer vehicle will move out from its coverage radius shortly, and it hands over the request to the *vRSU3* informing about the availability of requesting *MS3* code. Since the *vRSU3* lacks the availability of resources to compute *MS3*, it requests the *RSU-Edge3* for computations. Upon receiving the request, the *RSU-Edge3* fetches the *MS3* code from *RSU-Edge2* performs computations, and handover the results to the consumer via *vRSU3* (shown by the green arrow in Fig. 4).

3.9. Advantages of NDN as underlying communication model

The proposed NDN-enabled microservice-centric in-network computing framework is highly beneficial in various computational and communication aspects. Some of the major aspects are presented as follows:

1. **Computation & Communication Latency reduction:** The NDN inherent cache facility enables the node to store the microservice code and execute microservice-centric requests at the network layer without offloading it to the Edge, resulting in minimal communication and computation latency.
2. **Optimized Bandwidth and Cache Utilization:** Microservices are usually smaller in size and consume less bandwidth to download from the cloud, and utilize less node's cache. This significantly reduces the network burden in terms of bandwidth utilization compared with downloading the entire application code, which may also require more storage and might not be available in resource-deficient environments.
3. **Optimized Traffic Overhead:** The request aggregation mechanism in NDN allows network devices to aggregate identical microservice requests, thereby avoiding redundant offloading. This significantly reduces traffic overhead in the network.

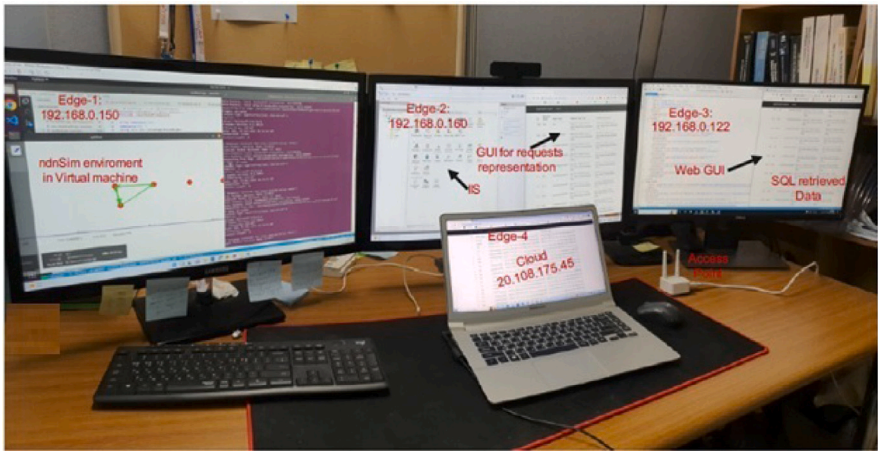

Fig. 5. MS centric In-network Computing testbed.

4. **Flexible microservice migration:** Using NDN, the inter-node microservice migration is highly beneficial, where a node having cached microservice code may share it with its neighboring nodes. This is particularly useful in dynamic vehicular environments where the demand for different microservices may fluctuate wildly.

4. Performance evaluation

4.1. Environmental setup

For the performance evaluation, we consider a microservices-centric driver assistance application (discussed previously), where the entire application is composed of various microservices such as (1) MS-CPP or MS1, (2) MS-FE or MS2, (3) MS-OD or MS3, (4) MS-ORep or MS4, (5) MS-OR or MS5. These microservices differ from each other in terms of size, latency, and computation requirements. The testbed enviorment shown in Fig. 5 is comprised of autonomous vehicular network in ndnSIM simulation environment, four Physical Edge computing servers with heterogeneous resource capabilities*i.e., (1) 16 GB RAM, core-i7, 9700-CPU and @ 3 GHz core, (2) 8 GB RAM, core-i5, 9400-CPU and @ 2.9 GHz core, (3) 8 GB RAM, core-i5, 33U-CPU and @ 1.9 GHz core and (4) 8 GB RAM, core-i5, 33U-CPU and @ 1.8 GHz core* placed in BCN labs Hongik University South Korea, and a *Microsoft Azure* trial hosting for Cloud in UK South. It is notable that for this evaluation, we utilized the limited-time trial hosting of the *Microsoft Azure*, exploiting the freely available resources to assess our framework's performance. As mentioned earlier, the operation of our framework initiates when the mobile consumer vehicle from the ndnSIM environment randomly generates the microservice-centric computation requests to the corresponding Edge server. To communicate between the Physical Edge machine and the simulation-based consumer, we modified the producer code of ndnSIM, implemented the *boost/asio* library, and utilized the *asio::ip::tcp* to make the requests. We emulated a microservices-based driver assistance application in the *.NET core* comprised of diverse microservices and deployed it on various Edge terminals and the Cloud. We utilized Internet *IIS* for the deployment that receives the consumer's request and responds with the requested data accordingly. We utilized the *SQL server 2017* developer version to log the received requests. To observe monolithic application behavior, we simulated a holistic monolithic architecture-based driver assistance application, which was hosted on multiple Edge servers and the Cloud.

4.2. Results and discussion

We analyzed the performance in terms of (1) Bandwidth utilization, (2) Average computation satisfaction delays, (3) Total packets processed, and (4) Computation result delivery ratio. Additionally, we performed a reliability analysis of our proposed framework in terms of backhaul traffic overhead and computation time and compared it with the Edge-only and cloud-only solutions [26]. In addition, we have also analyzed the number of requests aggregated at each layer of the framework.

The detailed results analysis is provided as follows.

4.2.1. Bandwidth utilization

We evaluated the performance of the microservices-centric framework and compared it with the monolithic system in terms of bandwidth utilization as shown in Figs. 6(a), 6(b), and 6(c). We analyzed the bandwidth consumption by varying the number of requests (i.e., between 2 to 14) as illustrated in Fig. 6(a). Additionally, we varied the number of hops between 1 to 4 and analyzed the bandwidth consumption at low traffic conditions (i.e., 1 req/s and 2 reqs/s) and high traffic conditions (i.e., 8 reqs/s and 12 reqs/s) as shown in Fig. 6(b) and Fig. 6(c), respectively. It is clear from the results that, in all cases, the proposed microservices-centric framework outperformed the monolithic schemes in terms of bandwidth consumption. The rationale is that microservices, by virtue of their small size, necessitate significantly lower bandwidth during the migration phase. Moreover, owing to their lightweight nature, microservices demand fewer computational resources during their execution phase. This subsequently minimizes the risk of computational overload, attenuates the likelihood of packet drops, and thereby curtails bandwidth consumption stemming from recurrent re-transmissions.

In contrast, the benchmark schemes consumed orders of magnitude higher bandwidth in all scenarios compared with the proposed work. The reason is that in the monolithic model, migrating a single component of the system from one edge to another necessitates the migration of the entire system, resulting in high bandwidth consumption. In addition, the dynamic vehicular environment with scarce bandwidth resources and heavy monolithic migration induces network congestion, which increases the packet drop rate and boosts the packet re-transmission frequency, collectively escalating the overall network bandwidth consumption (see Fig. 7).

4.2.2. Total packets processed in the network

Total packets processed in the network refers to the number of packets that have been transmitted, received, and processed by a network. The total packets processed in the network against the number of requests as well as the number of hops in both low and high traffic conditions can be visualized in Figs. 7(a), 7(b) and 7(c). It is evident from the results that less number of packets are processed in the network in the case of MS architecture. This efficiency stems from

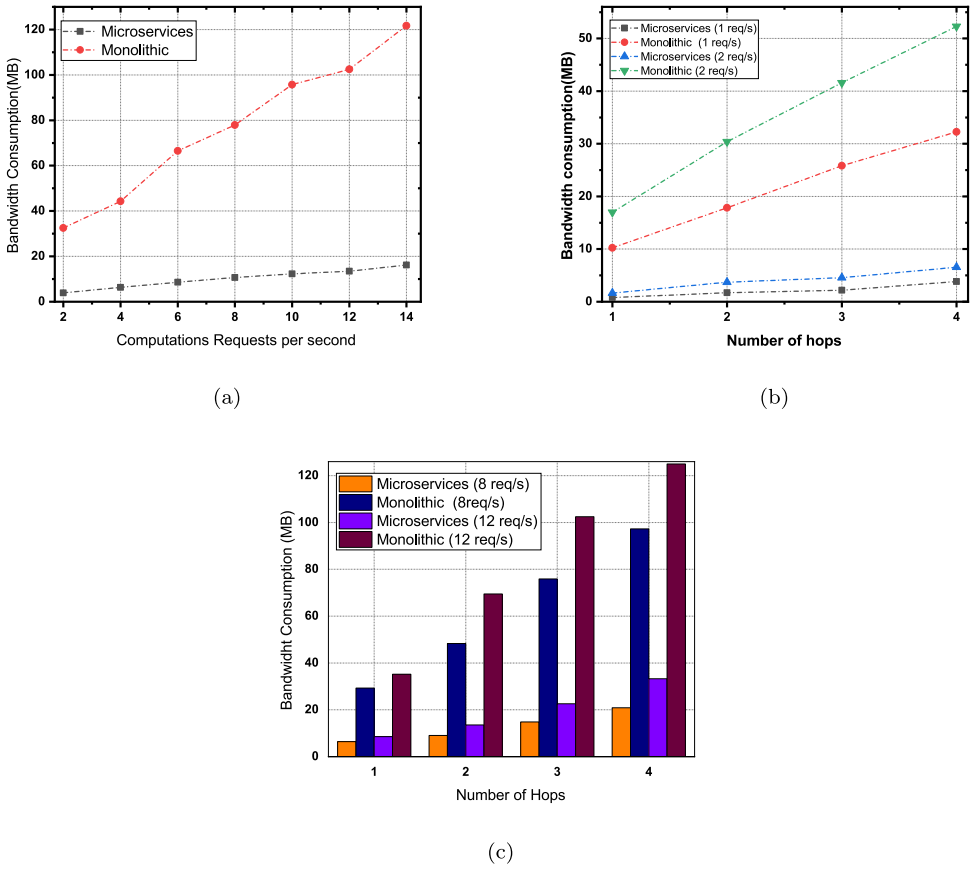

Fig. 6. Bandwidth Consumption (a) as a function of Req/s (b) as a function of the number of Hops (Low traffic) (c) as a function of hop count (High traffic).

its modular approach, which allows each microservice to be scaled and managed independently based on specific demand and resource needs, leading to more efficient network utilization. Such efficiency helps avoid network congestion and reduces the necessity for packet retransmissions. In contrast, a large number of packets are generated in monolithic cases as shown in the results. The reason is that all the components of an application are tightly coupled and run in the same process, meaning they share the same memory space. When a request is made to the application, the entire monolith must be loaded into memory, including all the components that may not be needed for that particular request. This can result in higher utilization of system resources and network bandwidth, leading to network congestion when multiple requests are made simultaneously.

4.2.3. Average computation delays

Average computation delays refer to the average time it takes to complete a computing task, from the point when the task is initiated to the point when the results are delivered. Fig. 8 illustrates the performance of both the proposed microservices-centric and monolithic systems in terms of computation satisfaction delays. For the analysis, we varied the computation request frequency, ranging from 10 req/s to 30 req/s. The results indicate that the proposed microservices-centric scheme experiences shorter computation satisfaction delays compared to the monolithic system. The rationale is that microservices, owing to their compact size, utilize fewer computational resources, facilitate faster computations, and diminish the risk of network congestion, subsequently lowering computation satisfaction delays. Conversely, the monolithic system, due to its larger size and significant bandwidth demands, increases the likelihood of congestion, leading to notable computational satisfaction delays.

4.2.4. Computation results delivery ratio

The computation results delivery ratio is defined as the number of successfully received compute results in response to generated microservices-centric computation requests. For the computational results delivery ratio, we vary the computation requests per second between 6 requests/s and 16 requests/s as shown in Fig. 9. The results clearly show that the proposed work surpasses the benchmark system in terms of successfully delivered packets. The reason is that microservices, owing to their compact sizes, necessitate fewer network resources during migration which minimizes the potential for congestion within the dynamically evolving vehicular network, thereby amplifying the proportion of successfully delivered computation results to the end consumer.

In monolithic architecture, the entire bulky monolith migration between compute nodes requires high network resources and induces network congestion, increasing the packet drop rate and reducing the packet delivery ratio.

4.2.5. Reliability analysis

The reliability of the proposed framework is analyzed in terms of computation satisfaction time and backhaul traffic overhead, and it is compared with Edge-only and cloud-only solutions. Additionally, the number of requests aggregated at the vRSUs, Physical Edge, and Cloud tiers has also been analyzed through extensive experimentation. A detailed description of the results is provided as follows.

- **Computation satisfaction time:** The average computation satisfaction time of the proposed framework, compared with the Edge-only and cloud-only schemes, is visualized in Fig. 10(a). The results demonstrate a significant reduction (over 50%) in computation satisfaction time, with the proposed scheme outperforming the benchmarks. The rationale is that NDN with its

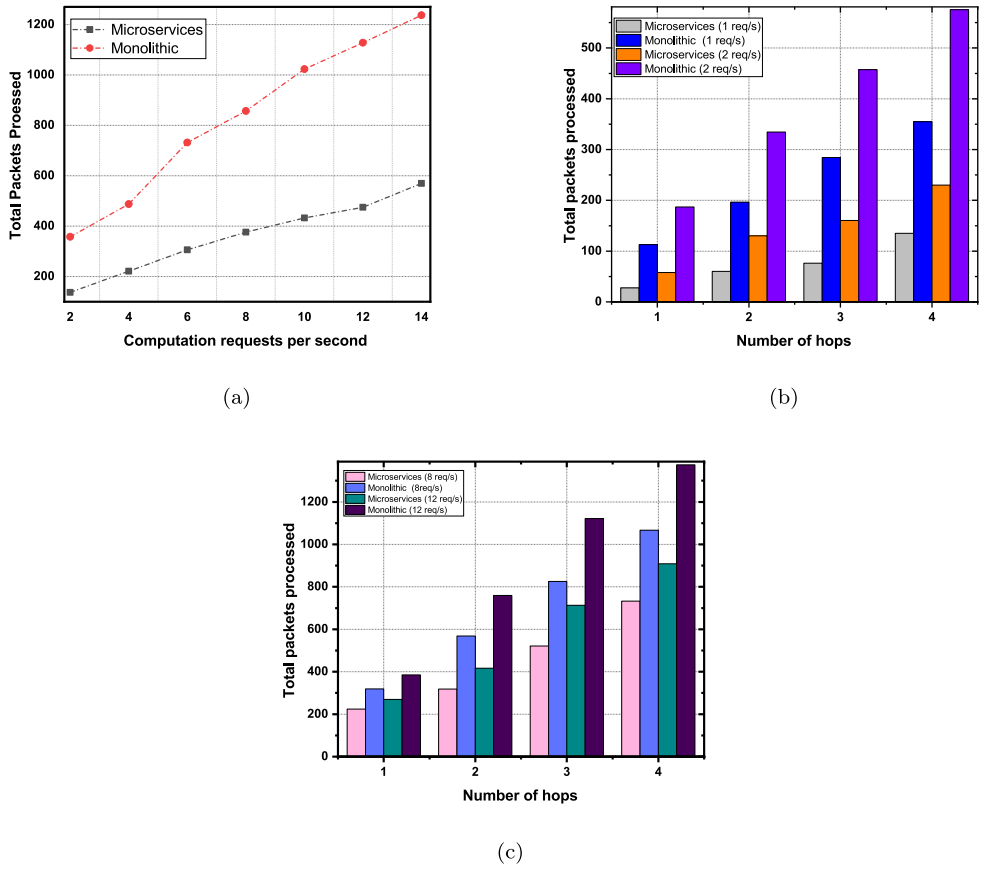

Fig. 7. Total Packets processed (a) as a function of Req/s (b) as a function of the number of Hops (Low traffic) (c) as a function of hop count (High traffic).

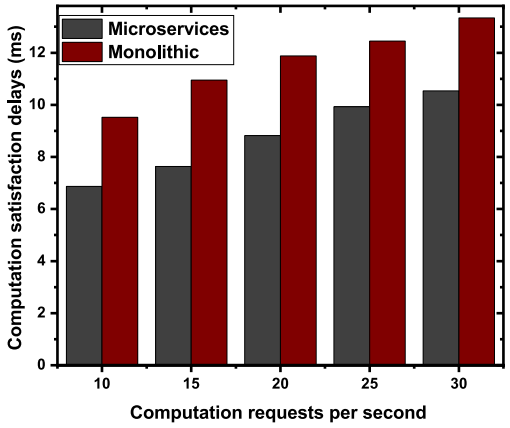

Fig. 8. Average computation satisfaction delays as a function of computation requests per second.

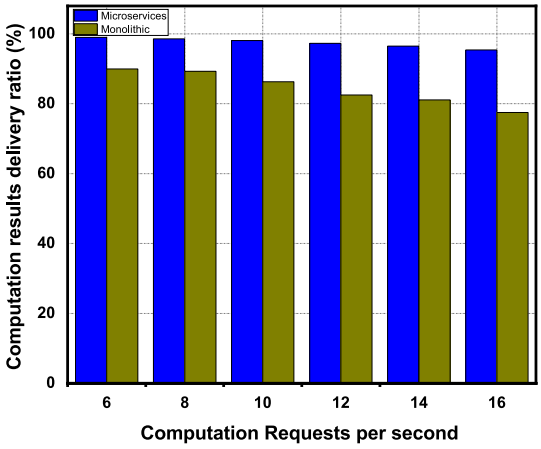

Fig. 9. Computation results delivery ratio as a function of computation requests per second.

in-network caching mechanism, enable the vRSU to cache microservice code and execute the requested computations directly, rather than offloading the request to a specific compute node. This approach not only reduces the computation satisfaction time but also decreases the number of requests offloaded to physical Edge computing nodes. While Edge-only schemes perform better than cloud-only solutions, they tend to offload requests to the cloud under heavy load due to resource limitations, resulting in increased computation times.

- **Backhaul traffic overhead:** Fig. 10(b) illustrates the backhaul traffic overhead generated by the proposed framework compared to the Edge-only and cloud-only schemes. The proposed framework significantly reduces backhaul traffic overhead compared with the benchmark schemes. The reason is that the NDN caching feature, coupled with microservices-centric computations, empowers the vRSU to handle the requested computations and avoid the frequent offloading. In contrast, Edge-only and cloud-only schemes experience significantly higher backhaul traffic. During periods of intense computation requests and bursty traffic conditions, Edge computing nodes may lack the necessary resources and offload requests to cloud terminals, leading to increased backhaul traffic.

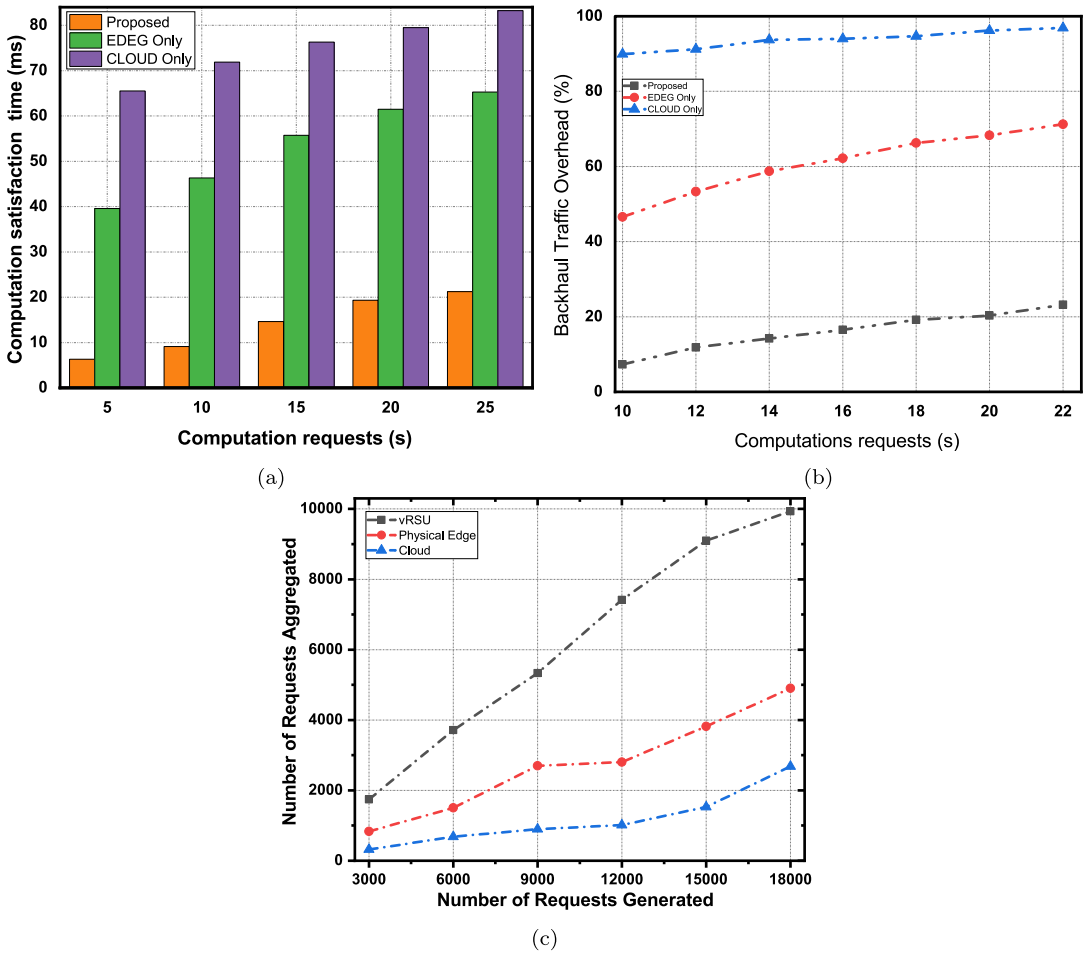

Fig. 10. Reliability analysis (a) Computation satisfaction time as function of computation requests (b) Backhaul traffic as a function of the number computation requests (c) Requests Aggregation as Total packets generated.

- **Requests aggregation analysis:** The impact of request aggregation in different layers of the proposed framework is analyzed against varying number of requests, as shown in Fig. 10(c). The graph indicates that, on average, over 60% of packets are aggregated at the vRSU level due to the request aggregation feature of NDN. Only a limited number of requests are offloaded to the physical Edge nodes or cloud servers. The experimental results clearly suggest that NDN is a highly promising framework, effectively reducing network resource utilization and backhaul traffic across Edge and Cloud Tiers.

5. Conclusion and future work

In this paper, by leveraging the intrinsic benefits of microservices architecture while utilizing the NDN as an underlying communication model, we developed a multilayered computing framework to analyze the benefits of microservice-centric in network computations in autonomous vehicular networks. The experimental results demonstrate that, compared with monolithic systems, the microservices-oriented in-network computing approach substantially conserves network bandwidth, reduces latency, and augments the delivery ratio of computed results under diverse traffic scenarios.

In future work, we plan to extend our framework by integrating the NDN stack on Raspberry Pi devices and deploying them in robotic vehicles aiming to analyze the effectiveness of real-time name-based microservice-oriented request/response mechanisms and in-network computations. Furthermore, we aim to implement Reinforcement Learning techniques in our proposed framework to foster intelligent traffic-aware microservice placement, thereby refining both application quality of service (QoS) and user quality of experience (QoE). Additionally, we plan to develop robust strategies to manage potential Edge server failures to ensure seamless computations in a dynamic network environment.

CRediT authorship contribution statement

Muhammad Salah Ud Din: Conceptualization, Investigation, Methodology, Validation, Writing – original draft. **Muhammad Atif Ur Rehman:** Methodology, Resources, Writing – review & editing. **Muhammad Imran:** Methodology, Software, Validation, Writing – review & editing. **Byung Seo Kim:** Conceptualization, Funding acquisition, Methodology, Supervision, Validation, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This research was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2022R1A2C1003549) , and in part by Strategic Networking & Development Program funded by the Ministry of Science and ICT through the National Research Foundation of Korea (RS-2023-00277267).

References

[1] C. Liu, K. Liu, H. Ren, X. Xu, R. Xie, J. Cao, RtDS: real-time distributed strategy for multi-period task offloading in vehicular edge computing environment, Neural Comput. Appl. (2021) 1–15.

[2] G. Cai, B. Fan, Y. Dong, T. Li, Y. Wu, Y. Zhang, Task-efficiency oriented V2X communications: Digital twin meets mobile edge computing, IEEE Wirel. Commun. (2023) 1–8, http://dx.doi.org/10.1109/MWC.012.2200465.

[3] P. Podder, A. Neishaboori, S.D. Gupta, A. Afanasyev, SPA: A scalable pedestrian-awareness application using NDN over CV2x, in: 2023 International Conference on Computing, Networking and Communications, ICNC, 2023, pp. 249–253, http://dx.doi.org/10.1109/ICNC57223.2023.10074070.

[4] M.S. Mohammed, A.M. Abduljabar, B.M. Faisal, S.H. Abdulhussain, W. Khan, P. Liatsis, A. Hussain, Low-cost autonomous car level 2: Design and implementation for conventional vehicles, Results Eng. 17 (2023) 100969.

[5] R. Ullah, M.A.U. Rehman, M.A. Naeem, B.-S. Kim, S. Mastorakis, ICN with edge for 5G: Exploiting in-network caching in ICN-based edge computing for 5G networks, Future Gener. Comput. Syst. 111 (2020) 159–174, http://dx.doi.org/10.1016/j.future.2020.04.033, URL https://www.sciencedirect.com/science/article/pii/S0167739X19328900.

[6] X. Hou, Z. Ren, J. Wang, W. Cheng, Y. Ren, K.-C. Chen, H. Zhang, Reliable computation offloading for edge-computing-enabled software-defined IoV, IEEE Internet Things J. 7 (8) (2020) 7097–7111, http://dx.doi.org/10.1109/JIOT.2020.2982292.

[7] K. Fu, W. Zhang, Q. Chen, D. Zeng, M. Guo, Adaptive resource efficient microservice deployment in cloud-edge continuum, IEEE Trans. Parallel Distrib. Syst. 33 (8) (2022) 1825–1840, http://dx.doi.org/10.1109/TPDS.2021.3128037.

[8] M. Amadeo, C. Campolo, A. Molinaro, CRoWN: Content-centric networking in vehicular ad hoc networks, IEEE Commun. Lett. 16 (9) (2012) 1380–1383.

[9] M.S.U. Din, M.A.U. Rehman, B.-S. Kim, CIDF-WSN: A collaborative interest and data forwarding strategy for named data wireless sensor networks, Sensors 21 (15) (2021) 5174.

[10] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang, Named data networking, ACM SIGCOMM Comput. Commun. Rev. 44 (3) (2014) 66–73.

[11] I. Ali, H. Lim, NameCent: Name centrality-based data broadcast mitigation in vehicular named data networks, IEEE Access 9 (2021) 162438–162447, http://dx.doi.org/10.1109/ACCESS.2021.3133016.

[12] B. Nour, S. Mastorakis, R. Ullah, N. Stergiou, Information-centric networking in wireless environments: Security risks and challenges, IEEE Wirel. Commun. 28 (2) (2021) 121–127, http://dx.doi.org/10.1109/MWC.001.2000245.

[13] M.A.U. Rehman, R. Ullah, B.-S. Kim, B. Nour, S. Mastorakis, CCIC-WSN: An architecture for single-channel cluster-based information-centric wireless sensor networks, IEEE Internet Things J. 8 (9) (2021) 7661–7675, http://dx.doi.org/10.1109/JIOT.2020.3041096.

[14] K. Zhang, Y. Mao, S. Leng, Y. He, Y. Zhang, Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading, IEEE Veh. Technol. Mag. 12 (2) (2017) 36–44.

[15] Z. Ning, J. Huang, X. Wang, Vehicular fog computing: Enabling real-time traffic management for smart cities, IEEE Wirel. Commun. 26 (1) (2019) 87–93.

[16] F. Sun, F. Hou, N. Cheng, M. Wang, H. Zhou, L. Gui, X. Shen, Cooperative task scheduling for computation offloading in vehicular cloud, IEEE Trans. Veh. Technol. 67 (11) (2018) 11049–11061.

[17] T. Mekki, I. Jabri, A. Rachedi, M.B. Jemaa, Towards multi-access edge based vehicular fog computing architecture, in: 2018 IEEE Global Communications Conference, GLOBECOM, IEEE, 2018, pp. 1–6.

[18] X. Hou, Z. Ren, J. Wang, W. Cheng, Y. Ren, K.-C. Chen, H. Zhang, Reliable computation offloading for edge-computing-enabled software-defined IoV, IEEE Internet Things J. 7 (8) (2020) 7097–7111.

[19] W. Chen, M. Liu, F. Wu, H. Wu, Y. Miao, F. Lyu, X. Shen, MSM: Mobility-aware service migration for seamless provision: A data-driven approach, IEEE Internet Things J. 10 (17) (2023) 15690–15704, http://dx.doi.org/10.1109/JIOT.2023.3265434.

[20] X. Wang, X. Wang, D. Wang, Cost-efficient data retrieval based on integration of VC and NDN, IEEE Trans. Veh. Technol. 70 (1) (2021) 967–976, http://dx.doi.org/10.1109/TVT.2021.3049795.

[21] R.W. Coutinho, A. Boukerche, X. Yu, A novel location-based content distribution protocol for vehicular named-data networks, in: 2018 IEEE Symposium on Computers and Communications, ISCC, IEEE, 2018, pp. 01007–01012.

[22] S.H. Ahmed, S.H. Bouk, M.A. Yaqub, D. Kim, H. Song, J. Lloret, CODIE: Controlled data and interest evaluation in vehicular named data networks, IEEE Trans. Veh. Technol. 65 (6) (2016) 3954–3963, http://dx.doi.org/10.1109/TVT.2016.2558650.

[23] A. Samanta, J. Tang, Dyme: Dynamic microservice scheduling in edge computing enabled IoT, IEEE Internet Things J. 7 (7) (2020) 6164–6174, http://dx.doi.org/10.1109/JIOT.2020.2981958.

[24] A. Samanta, L. Jiao, M. Mühlhäuser, L. Wang, Incentivizing microservices for online resource sharing in edge clouds, in: 2019 IEEE 39th International Conference on Distributed Computing Systems, ICDCS, 2019, pp. 420–430, http://dx.doi.org/10.1109/ICDCS.2019.00049.

[25] A. Samanta, T.G. Nguyen, T. Ha, S. Mumtaz, Distributed resource distribution and offloading for resource-agnostic microservices in industrial IoT, IEEE Trans. Veh. Technol. 72 (1) (2023) 1184–1195, http://dx.doi.org/10.1109/TVT.2022.3206137.

[26] M.A.U. Rehman, S. Mastorakis, B.-S. Kim, et al., FoggyEdge: An information-centric computation offloading and management framework for edge-based vehicular fog computing, IEEE Intell. Transp. Syst. Mag. (2023).