DIVINE: Data Offloading In Vehicular Networks with QoS Provisioning

Yasir Saleem, Nathalie Mitton, Valeria Loscri

DIVINE: Data Offloading In Vehicular Networks with QoS Provisioning

Yasir Saleem, Nathalie Mitton, Valeria Loscri

Inria Lille – Nord Europe, France

Abstract

In vehicular networks, vehicles may carry various types of data that need to be offloaded to the RoadSide Units (RSUs) through Vehicle-to-Infrastructure (V2I) communications when vehicles come into their coverage. RSUs are not widely deployed everywhere, which causes intermittent connectivity between vehicles and RSUs. In this paper, we propose DIVINE, a Data offloading In VehIcular NEtworks scheme with QoS provisioning, which enables a vehicle to offload its data to RSU directly through V2I communications or using other neighboring vehicles through Vehicle-to-Vehicle (V2V) communications. DIVINE considers the connectivity time of an offloading vehicle with the RSU, with other vehicles heading either on the same or opposite direction, offloading capacity, expected time to reach RSU and contact duration with neighboring vehicles. Additionally, the Quality of Service (QoS) is an important consideration for data offloading in vehicular networks due to the coexistence of urgent data to offload (e.g., accident or emergency data). Therefore, for QoS provisioning, DIVINE uses three QoS functions: traffic classification, overload control and admission control. DIVINE is presented with algorithms and procedures, as well as with illustrative examples. The performance evaluation in network simulator OMNeT++ with Veins and SUMO frameworks shows that DIVINE outperforms other schemes in terms of average offloading delay, maximum offloading delay and running time for a varying number of vehicles, maximum speed values, number of RSUs and RSUs' capacity. It also best behaves in terms of amount of offloaded important data.

Keywords: Data offloading, Road-side Unit (RSU), Quality of Service (QoS) provisioning, Vehicular network, Vehicle-to-Vehicle (V2V) communication, Vehicle-to-Infrastructure (V2I) communication.

1. Introduction

In vehicular networks, offloading is a core component. There are two types of offloading in vehicular networks: data offloading [1] and task offloading [2]. Data offloading is further classified into two types. Firstly, vehicles carry data that has to be offloaded to the RoadSide Units (RSUs) and secondly, vehicles request

some services (such as online videos) from RSUs that have to be offloaded from RSUs to requested vehicles. In task offloading, vehicles have some tasks to calculate (such as processing of captured images), however vehicles are equipped with limited computation power, therefore such tasks need to be offloaded to RSUs where an edge server computes the task and the results are sent back to the vehicles. Both, the data and task offloading are important in vehicular networks because either the vehicles carry data that needs to be reached to the decision support centres for taking appropriate actions timely or the vehicles need to compute some tasks (e.g., route finding or image processing) which they cannot perform themselves due to their limited resources. Therefore, for both purposes, there is a need of some mechanisms that can offload both the data and the tasks of vehicles efficiently and timely using the help of road infrastructures (i.e., RSUs). We focus on the first type of offloading, i.e., the data carried by vehicles to be offloaded to RSUs. Therefore, throughout this article, the data offloading refers to this type.

A vehicle can carry various types of data, such as accident data of nearby incident, emergency health data of patients, road traffic conditions, periodic health monitoring data, entertainment (e.g., streaming of online gaming and videos) and backup data. The format of data could be video, image or text. Such data needs to be offloaded to RSUs through Vehicle-to-Infrastructure (V2I) communications (i.e., V2I data offloading) when vehicles come into their coverage so that RSUs, which are equipped with edge servers, can analyze and process the data for taking required actions or forward the data to the cloud. The RSUs have been developed for more than ten years, however they are still not widely deployed everywhere [3]. Therefore, there is an intermittent connectivity of RSUs that causes a vehicle to be not in the coverage of RSUs all the time.

Some data carried by vehicles can be very urgent and needs to be offloaded to RSUs as soon as possible, such as accident data of nearby incident and emergency health data. However, with the intermittent connectivity of RSUs, there might be substantial delay in offloading such data to RSUs. Vehicle-to-Vehicle (V2V) communications (i.e., V2V data offloading) can be used to reduce such an offloading delay. For instance, if there is another vehicle heading on the same direction at a higher speed or another vehicle on the opposite direction that is going to meet an RSU sooner, it can be used to relay the vehicle's data up to an RSU. Moreover, since the vehicles are moving, their contact duration needs to be considered while selecting V2V data offloading.

The V2I and V2V data offloading should be decided based on the connectivity modeling, such as the connectivity time of a vehicle with an RSU, connectivity time of a vehicle with other vehicles heading on the same or opposite direction, offloading capacity (i.e., the maximum data that could be offloaded) and expected time to reach RSU.

There are some Quality of Service (QoS) considerations that are imperative to be provisioned. For instance, since a vehicle carries various types of data, urgent data must be offloaded first. Additionally, how to avoid RSUs from getting overloaded for smooth data offloading, how to offload urgent data if an

RSU is overloaded and finally, which data to offload using V2V data offloading should be considered.

In this paper, we propose DIVINE, a Data offloading In VehIcular NEtworks scheme with QoS provisioning. The main objective is to offload vehicles' data to RSUs as soon as possible. DIVINE models the connectivity for deciding V2I or V2V data offloading. It decides V2V data offloading based on the type of data to offload, contact duration of vehicles and the expected time to reach the next RSU. It ensures the QoS by using the three functions of traffic classification, overload control and admission control. To the best of our knowledge, there does not exist any data offloading scheme in the literature that jointly considers and investigates these features.

DIVINE provides three main contributions to offer an efficient data offloading system for vehicular networks which are summarized as follows:

C.1 DIVINE models and characterizes the connectivity of a vehicle with an RSU, with vehicles heading on the same and opposite direction, the offloading capacity and the expected time to reach an RSU by a vehicle.

C.2 DIVINE provisions the QoS by using the functions of traffic classification, overload control and admission control. The traffic classification prioritizes the data for offloading into three priorities: high, medium and low. The overload control defines the threshold values of the maximum allowed loads at the RSUs for medium and low priority data. The admission control allows RSUs and vehicles to stop servicing an existing vehicle in order to provide service to another vehicle having higher priority data and avoiding the offloading of low priority data through V2V data offloading.

C.3 DIVINE integrates data offloading procedures and algorithms with QoS provisioning, specifically the selection of node (RSU/vehicle) for data offloading request, the processing of data offloading request at an RSU and a vehicle, and the data offloading decision. Additionally, we present illustrative examples for better understanding of V2I and V2V data offloading procedures.

The remainder of this paper is organized as follows. Section 2 presents the related work of existing data offloading schemes, as well as QoS provisioning schemes in vehicular networks. It also describes how our proposed scheme is positioned with respect to the literature and how it is novel. Section 3 presents the system model that is comprised of three layers: vehicles, edge and cloud. It also provides the assumptions that are considered in this paper. Section 4 presents the connectivity modeling of vehicles, specifically, the connectivity of vehicles with RSUs and with other vehicles (heading on the same and opposite directions). It also models the offloading capacity and the expected time to reach an RSU. Section 5 presents our proposed solution DIVINE. It first provides an overview of DIVINE followed by the QoS provisioning using the functions of traffic classification, overload control and admission control. Then

it presents the algorithms and procedures of data offloading followed by illustrative examples. Section 6 analyses the message overhead and time complexity of DIVINE. Section 7 presents the performance evaluation, results and discussion. It first presents the simulation setup and parameters, performance metrics and comparison schemes. Then it presents the detailed performance evaluation with respect to the varying number of vehicles, vehicles' maximum speed, number of RSUs and the RSUs' capacity. Finally, section 8 presents the conclusion and future work.

2. Related Work

This section presents the related work and discusses how DIVINE is different and novel from the state-of-the-art related works. As discussed earlier, offloading in vehicular networks has mainly been categorized into data offloading and task offloading. Much research works focus on task offloading in vehicular networks, however there are few studies that focus on data offloading in vehicular networks. Since our focus is on data offloading in this article, therefore we cover the state-of-the-art related to data offloading in vehicular networks.

Huang *et al.* [3] proposed V2V2I offloading that constructs multihop V2V paths connected towards RSUs in which a vehicle normally uses cellular network for data offloading. However, when a vehicle comes into the coverage of another vehicle (i.e., offloading agent) that has a connected k-hop V2V path towards RSU, the vehicle uses this V2V path to get connected to RSU for data offloading. The authors studied the selection of offloading agent, the construction of k-hop V2V path using Mobile Edge Computing (MEC) and path repairing if the k-hop V2V2I offloading path is disconnected. Although we are also considering V2I and V2V data offloading, however we are not considering multihop V2V path because each vehicle contains data that needs to be offloaded to RSU. Since, each vehicle has limited connectivity time with RSU, therefore if it has more data to offload that needs higher connectivity time, then even if there is a multihop path and it is part of this multihop path, it cannot provide service to other vehicles in relaying their data.

Ancona *et al.* [4] dealt with Floating Car Data (FCD), i.e., the information collected by vehicles that needs to be transmitted to control centers for analysis. FCD is used for various applications, such as monitoring of vehicles remotely, traffic management and urban sensing. The amount of FCD is generally huge that is collected by a large number of vehicles and uploaded to the control centers through cellular communications by each vehicle. This causes the cellular network to be overloaded with FCD. The authors tried to solve this problem of relieving the overloaded cellular networks by utilizing V2V communications for FCD offloading. Hence, vehicles offload their FCD to a subset of uploader vehicles using Dedicated Short-Range Communication (DSRC) technology and subsequently, such uploader vehicles aggregate all FCD and upload the aggregated FCD to the control centers through cellular communications. For aggregating FCD, the authors explored the aggregation models and proposed a simple distributed heuristic algorithm that achieves near-optimal performance

under any FCD aggregation model. This work is mainly focused on partially relieving the cellular networks from FCD by aggregating the FCD at uploader vehicles, however our focus is mainly on data offloading from vehicles to RSUs either directly through V2I communications or indirectly through V2V communications without aggregating the data, as all the data need to be reached to RSUs. Additionally, there is no QoS provisioning in this work.

Xiangming *et al.* [5] investigated the integration of cellular and opportunistic vehicular networks by considering the contact duration of vehicles for mobile data offloading. Considering the contact duration, the authors proposed a mathematical framework for contact-aware optimal resource allocation offloading scheme that was formulated as utility maximization problem under limited storage constraints of vehicles. The authors evaluated the performance in MATLAB by using Shanghai and Beijing vehicular mobility traces and compared with other schemes. This work is about data offloading from cellular network to vehicles in which the vehicles further disseminate the data to other vehicles in an opportunistic manner, mainly focusing on the contact duration among vehicles. However, on the other hand, we focus on data offloading from vehicles to RSUs in which the vehicles have data that needs to be offloaded to RSUs.

Lee *et al.* [6] proposed a data offloading technique for the reduction of cellular traffic load for in-vehicle data services requests. The authors assumed that the majority of in-vehicle service requests are about some popular content that cause redundancy in cellular traffic. Hence, the relay nodes (also called offloading positions) in vehicular networks keep the popular content and deliver them to moving vehicles that request the content. In this manner, vehicles do not need to use cellular network that results in the reduction of cellular traffic load. The authors utilized vehicles trajectories and formulated the selection of offloading position as spatio-temporal set-covering problem. Subsequently, using the vehicles trajectories, the authors proposed a time-prediction based set-covering algorithm to select the minimum number of offloading positions. This work is also about data offloading from cellular network to vehicles using intermediate relay nodes (offloading positions) for reducing cellular traffic load, however, we are focused on data offloading from vehicles to RSUs.

Sun *et al.* [7] studied the cooperative downloading mechanism of online video content in vehicular networks to offload the cellular network as an optimization problem. The RSUs fetch the appropriate data from the Internet by acting as traffic managers and offload them to vehicles. The authors designed a storage time aggregated graph based on vehicles' mobility prediction and throughput estimation for planning transmission scheme. Subsequently, they proposed an iterative greedy algorithm for a sub-optimal solution with a polynomial time complexity. This work is also about video data offloading from cellular network to vehicles using RSUs for reducing cellular traffic load, however, we are focused on data offloading from vehicles to RSUs.

Feng and Feng [8] worked with non-uniformity of data traffic that causes cellular network in hot zones, mostly at street intersections, to be overloaded because of the pedestrians surfing the mobile Internet during waiting at red light, while inefficient use of wireless resources of cellular networks in light-

loaded (i.e., non-intersection) zones. The authors proposed a vehicle-assisted offloading (VAO) scheme to solve this problem in which the BSs of light-loaded cellular networks offload delay-insensitive data to vehicles, and subsequently after entering the street intersection and stopping at the red light, the vehicles send the data to pedestrians. In this manner, the pedestrians do not use cellular network to surf the Internet that reduce their load. This work also investigated reducing the cellular traffic load by providing the content to pedestrian from vehicles. However, we do not consider pedestrian in our work and do not consider cellular traffic overload. Rather, we are mainly focused on data offloading from vehicles to RSUs.

Song *et al.* [9] researched on backward data delivery for V2V data transmissions under the problematic scenario of traffic hole in which no vehicles are available to relay the data of source vehicle towards the destination vehicle. The authors took assistance of RSUs for data transmissions and proposed RSU-Assisted Backward Delivery (RABD) that employed two methods: backward data relaying among vehicles and the data relaying by RSUs. For reducing the resource consumption and fulfilling the various demands of data delivery, the authors investigated the single-copy and multiple copies schemes, respectively and evaluated their trade-offs and performance through simulations in NS-2 and SUMO using Taxi-ROMA dataset. However, firstly, this is a routing problem of data delivery and secondly, it is mainly focused on backward data delivery under the traffic hole problem, while we are mainly focused on V2I and V2V data offloading.

Guntuka *et al.* [10] proposed a Smart Ranking based Data Offloading (SRDO) scheme for the selection of RSU and to switch from cellular network to RSU whenever possible. SRDO is based on Software Defined Networking (SDN) controller for the centralized selection of RSUs using Q-learning. SRDO relies on a centralized entity, while DIVINE is a distributed scheme without any centralized entity. Additionally, SRDO does not consider V2V data offloading. The QoS considerations in this work include packet loss, delay and throughput, while DIVINE uses the QoS functions of traffic classification, overload control and admission control.

Wu and Zheng [11, 12] investigated an important QoS metric of delay in vehicular networks in a highway scenario which is an important metric for data offloading in vehicular networks. The authors analysed the uplink local delay [11] and download local delay [12] using stochastic geometry in edge-based vehicular networks. They first derived an analytical model to study the average uplink local delay of sending a packet from a vehicle to RSU node and the average downlink local delay to receive a packet by a vehicle from RSU. The vehicles and RSUs are spatially distributed following the independent one-dimensional homogeneous Poisson point process model and they use carrier sense multiple access (CSMA) for channel access.

[13] studied the problem of RSU deployment in a vehicular network and studied the information delivery delay. It considers a highway scenario in which two neighboring RSUs cannot communicate with each other and the vehicles are distributed on the road containing the information of road condition that

is randomly generated between the two neighboring RSUs. Considering this scenario, it developed a mathematical model to model the relationship between the deployment distance of two neighboring RSUs and the delay to deliver the road condition data by considering the speed and density of vehicles and the probability of an incident. This model is applicable for RSUs deployment in estimating the maximum allowed distance between two neighboring RSUs.

Wang and Zheng [14] considered a one-way highway scenario having one entry and exit with multiple uniformly distributed RSUs. The authors analysed the connectivity probability by deriving an analytical model in which the highway road is distributed into multiple segments for which the connectivity probability is derived using probability distribution theory. The analytical model considers the arrival rate of vehicles, speed, the number of RSUs and the probability of vehicle's driving-through at the entry and exit. Finally, the connectivity probability of each sub-segment is used to derive the connectivity probability of the highway road.

Li *et al.* [15] proposed an adaptive QoS-based routing protocol for vehicular networks (AQRV) that considered three QoS metrics: delay, packet delivery ratio and probability of connectivity. It selected the routes that satisfy these QoS metrics by mathematically formulating as a constrained optimization problem using ant colony optimization. Sodhro *et al.* [16] investigated QoS issues in vehicular networks and provides three contributions. Firstly, it proposed a QoS-aware green, sustainable, reliable and available (QGSRA) algorithm for multimedia transmission in edge-based vehicular networks. Secondly, it implemented a QoS optimization for QGSRA algorithm. Thirdly, it proposed the QoS metrics of energy efficiency, packet loss ratio and coverage for QGSRA algorithm.

In [17], we proposed V2I-Q, a V2I data offloading scheme for vehicular network with QoS provisioning. V2I-Q offloads the data using V2I data offloading. It also uses QoS functions of traffic classification, overload control and admission control for QoS provisioning. Our current paper is an extension of [17] and we differ with it in several ways. We have enabled vehicles to take advantage of data offloading through other vehicles, i.e., we have added V2V data offloading and presented the algorithms, procedures and illustrative examples for V2V data offloading. We have also modeled the connectivity of vehicles with other vehicles heading on the same or opposite direction, as well as the expected time to reach the next RSUs. We also extended the QoS functions of overload control and admission control and applied them on vehicles. We have compared our proposed scheme DIVINE with V2I-Q as well.

Compared to the above works, our proposed scheme DIVINE is different from them in many ways. For instance, we focus on data offloading from vehicles to RSUs using V2I and V2V data offloading. We provide the QoS using three functions of traffic classification, overload control and admission control. We use DSRC technology based on IEEE 802.11p which is dedicated for vehicular networks. Finally, we take various considerations into data offloading procedure, such as type of data to offload, contact duration of vehicles and the expected time to reach the next RSU.

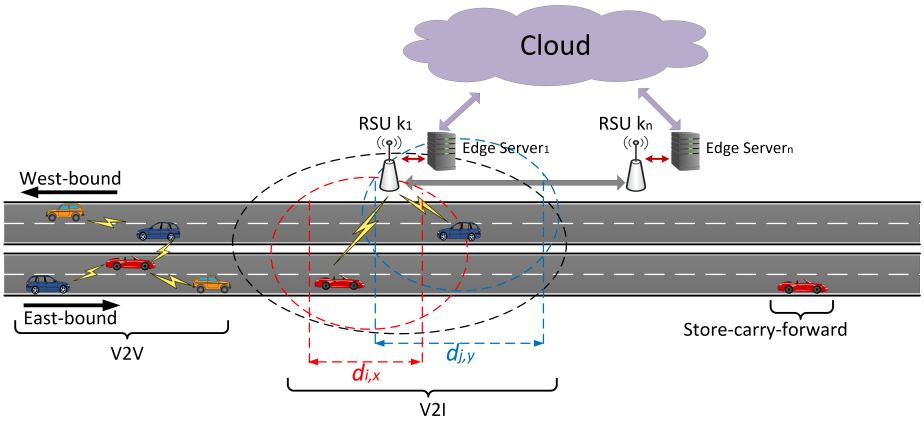


Figure 1: System model.

3. System Model

We consider a vehicular network having $\mathcal{K} = \{1, 2, \ldots, K\}$ RSUs and $\mathcal{N} = \{1, 2, \ldots, N\}$ vehicles such that the total number of RSUs is less than the total number of vehicles in the network, i.e., $\mathcal{K} < \mathcal{N}$. The network has intermittent connectivity so a Vehicle $i \in \mathcal{N}$ can have connectivity with an RSU $k \in \mathcal{K}$, however, sometimes, a Vehicle i enters an area with very low or no connectivity with RSUs but can have connectivity with other neighboring vehicles. However, it could also happen that a Vehicle i has connectivity neither with an RSU nor with other vehicles. The system model is presented in Figure 1 and is comprised of three layers: vehicles, edge computing and cloud computing layers.

The first layer is comprised of vehicles on east-bound and west-bound highway roads having multiple lanes on each direction. A vehicle offloads its data to RSU either directly or through other vehicles. The vehicles move at a variable speed. The on-board unit (OBU) is mounted on each vehicle to provide wireless communication capability. The vehicles communicate with RSUs and other vehicles using DSRC technology that is based on IEEE 802.11p standard [18]. The vehicles support V2I communications, V2V communications and store-carry-forward mechanisms. V2I communication is used to communicate with RSUs. V2V communication is used to communicate with other vehicles. Store-carry-forward mechanism [19] is used when a vehicle has neither connectivity with an RSU nor with other vehicles. We assume a vehicle can be connected to a maximum one RSU at a time. Each vehicle knows its speed and the distance of its last connected RSU. All this information (vehicle's speed, distance of its last connected RSU, supported data rate etc.) is broadcast through periodic control messages and hence, each vehicle approximates the expected time to reach the next RSU based on information received from neighboring vehicles. The connectivity of vehicles with RSUs varies from lane to lane as illustrated in the system model (e.g., the red Vehicle i moving on lane x that is farther from the RSU k_1 has shorter connectivity than the blue Vehicle j moving on

lane y that is nearer to the RSU k_1). We assume all vehicles have the same coverage area, while RSUs have higher coverage area, and therefore, if vehicles have connectivity with RSUs (i.e., vehicles have RSUs in their coverage area), then implicitly, RSUs also have connectivity with vehicles.

The second layer is comprised of RSUs and edge servers. Each RSU is connected through broadband connection with an edge server having storage and processing capabilities. Each RSU is also connected with previous and next RSUs and communicate through broadband connection as well. The RSUs collect data from vehicles using DSRC communication technology and send this data to their connected edge server. Subsequently, the edge server forwards this data to the cloud. We assume that the coverage areas of different RSUs do not overlap, and as a consequence, a vehicle does not have more than one RSU in its coverage at a time and it needs to establish connection with each RSU separately. This layer supports V2I and Infrastructure-to-Infrastructure (I2I) communications methods.

The third layer is the cloud layer. After RSUs send their collected data from vehicles to edge servers, each edge server forwards data to the cloud for a centralized storage and management. We do not deal with cloud layer and therefore, this third layer is out of the scope of this paper.

Throughout the paper, the vehicle that is offloading its data to RSU or other vehicle is referred as the offloading vehicle, while the vehicle that is receiving data from offloading vehicle is referred as the offloaded vehicle. Table 1 sums up all the notations used in this paper.

4. Connectivity Modeling and Characterization

A vehicle carries data that needs to be offloaded to RSU through either V2I or V2V data offloading. The connectivity time plays an important role in data offloading, more specifically, in calculating the offloading capacity, i.e., how much data a vehicle can offload to RSU or other vehicles. The connectivity time of a vehicle varies for RSU, neighboring vehicle heading on the same direction and neighboring vehicle heading on the opposite direction.

In this section, we first model and characterize the connectivity time of a vehicle in the three above mentioned cases. Subsequently, we model the offloading capacity and the expected time to reach the RSU (that is a metric for selecting a neighboring vehicle in V2V data offloading).

4.1. Connectivity Time

4.1.1. Connectivity Time with RSU

In this case, a vehicle has direct connectivity with an RSU. After having an RSU in the coverage, a vehicle sends a data offloading request to RSU containing the amount of each type of data it wants to offload, its speed and data rate. As a first step, the RSU calculates the connectivity time of a vehicle with it before analyzing the data offloading request as follows:

Table 1: Notations used in this article.

Notation	Description
$\mathcal{K}$	Set of RSUs
$\mathcal{N}$	Set of vehicles
$\mathcal{P}$	Set of data priorities
$d_{k,x}$	Coverage area of lane x by RSU k
d_i	Coverage area of Vehicle i
r_i	Data rate of Vehicle i
V_i	Speed of Vehicle i
H_i	Heading direction of Vehicle i
$T^c_{i,k}$	Connectivity time of Vehicle i with RSU k
$T^c_{i,l}$	Connectivity time of Vehicle i with Vehicle l
$\mathcal{S}^{max}_{i,k}$	Maximum allowed data size to offload by RSU k to Vehicle i
$\mathcal{S}^{max}_{i,l}$	Maximum allowed data size to offload by Vehicle l to Vehicle i
$\tau_{i,k}$	Registration time of a Vehicle i with RSU k
$\tau_{i,l}$	Registration time of a Vehicle i with Vehicle l
S^{high}_i	Size of high priority data carried by Vehicle i
S^{med}_i	Size of medium priority data carried by Vehicle i
S^{low}_i	Size of low priority data carried by Vehicle i
$\gamma^{max,med}_k$	Threshold value of maximum allowed medium priority data at RSU k
$\gamma^{max,low}_k$	Threshold value of maximum allowed low priority data at RSU k
$\omega^{min,c}_{v2v}$	Threshold value of the least allowed contact duration between vehicles for V2V data offloading
Ω^{arr}_{RSU}	Threshold value of the allowed difference of reaching RSU for two vehicles in V2V data offloading
ϑ_k	Current load at RSU k
ϑ^{max}_k	Maximum tolerable load at RSU k
$\Phi^{grant}_{k,i}$	Granted set from RSU k to Vehicle i containing sizes of different priority data to offload
$S^{max,t}_i$	Maximum allowed requested data size
$T^{arr}_{i,RSU}$	Expected time to reach RSU by Vehicle i
$T^{wait}_{i,v2i}$	Waiting time duration for Vehicle i after sending data offloading request in V2I data offloading
$T^{wait}_{j,v2v}$	Waiting time duration for Vehicle j after sending data offloading request in V2V data offloading
$T^{ofld}_{i,j}$	Time required by Vehicle i to offload its data to Vehicle j
l^{rsu}_i	The distance of last connected RSU with Vehicle i
nb_i	Neighboring nodes (RSU or vehicles) of Vehicle i
nb^{sent}_i	Neighboring nodes (RSU or vehicles) of Vehicle i to whom it has requested data offloading
$T^{lifetime}_{offl,req}$	Offloading request lifetime

$$T^c_{i,k} = \frac{d_{i,x}}{V_i} - \tau_{i,k} - T^{wait}_{i,v2i} \tag{1}$$

where $i \in \mathcal{N}$ is the vehicle that requested to offload its data to RSU $k \in \mathcal{K}$, $d_{i,x}$ is the distance traveled by Vehicle i on lane x under the coverage of RSU k (as shown of Figure 1), V_i is the speed of Vehicle i, $\tau_{i,k}$ is the registration time of Vehicle i with RSU k and $T^{wait}_{i,v2i}$ is the waiting time for Vehicle i after sending data offloading request to RSU k in V2I data offloading.

4.1.2. Connectivity Time with Vehicles Heading on the Same Direction

When two vehicles are moving on the same direction, they will remain connected for a longer period of time depending upon the difference of theirs speeds.

The connectivity time for this case is calculated as:

$$T_{i,j}^{c} = \frac{min(d_i, d_j)}{|V_i - V_j|} - \tau_{i,j} - T_{i,v2v}^{wait} \tag{2}$$

where $i, j \in \mathcal{N}$ are two vehicles heading on the same direction, d_i and d_j are the coverage areas of Vehicles i and j, V_i and V_j are their speeds, $|V_i - V_j|$ is the modulus of difference of speeds of both vehicles, $\tau_{i,j}$ is the registration time of Vehicle i with its neighboring Vehicle j and $T_{i,v2v}^{wait}$ is the waiting time of Vehicle i after sending a data offloading request to neighboring Vehicle j in V2V data offloading. Note that although we considered all the vehicles to have same coverage area, for the sake of generalization, we used $min(d_i, d_j)$ so that even if the coverage areas of Vehicles i and j are different, our equation can still hold.

4.1.3. Connectivity Time with Vehicles Heading on the Opposite Direction

When two vehicles are moving on the opposite direction, they will stay connected for a shorter duration and the connectivity will be limited. If there is a slight difference in their speeds, the connectivity time will be almost half of the connectivity time of a vehicle with RSU. The connectivity time for this case is calculated as:

$$T_{i,l}^{c} = \frac{min(d_i, d_l)}{V_i + V_l} - \tau_{i,l} - T_{i,v2v}^{wait} \tag{3}$$

where $i \in \mathcal{N}$ is a vehicle that wants to offload its data to neighboring Vehicle $l \in \mathcal{N}$ heading on the opposite direction, d_i and d_l are the coverage areas of Vehicle i and neighboring Vehicle l, V_i and V_l are their speeds, $\tau_{i,l}$ is the registration time of a Vehicle i with its neighboring Vehicle l and $T_{i,v2v}^{wait}$ is the waiting time for Vehicle i after sending a data offloading request to neighboring Vehicle l in V2V data offloading. We used $min(d_i, d_l)$ for the same reason discussed above.

4.2. Offloading Capacity

Offloading capacity is defined as the maximum allowed data size that a vehicle can offload to RSU or other vehicles within the connectivity time. The offloading capacity is directly proportional to the connectivity time. If the connectivity time is higher, the offloading capacity will be higher, while if the connectivity time is lower, the offloading capacity will also be lower. It is a metric for limiting the size of data offloading and is calculated as:

$$\mathcal{S}_{i,j\in\{\mathcal{K},\mathcal{N}\}}^{max} = T_{i,j\in\{\mathcal{K},\mathcal{N}\}}^{c} \times min(r_i, r_{j\in\{\mathcal{K},\mathcal{N}\}}) \tag{4}$$

where $i \in \mathcal{N}$ is a vehicle wanting to offload its data either to RSU $j \in \mathcal{K}$ or other Vehicle $j \in \mathcal{N}$, r_i is the data rate of Vehicle i and $r_{j\in\{\mathcal{K},\mathcal{N}\}}$ is the data rate of RSU $j \in \mathcal{K}$ or Vehicle $j \in \mathcal{N}$. Note that it might be possible that vehicles are using different devices that support different data rate. For example, it can

happen that Vehicle i supports higher data rate, while RSU or other vehicle to which it is going to offload its data supports lower data rate that can cause data loss. Therefore, we consider the minimum data rate $min(r_i, r_{j \in \{\mathcal{K},\mathcal{N}\}})$ of both offloading Vehicle i and offloaded RSU $j \in \mathcal{K}$ or Vehicle $j \in \mathcal{N}$ for smooth data transfer.

4.3. Expected Time To Reach an RSU

As vehicles exchange the distance they have traveled from the last RSU they encountered and we assume a highway scenario which is a straight road, this information can be used by vehicles heading on the opposite direction to estimate the expected time to reach the next RSU (i.e., arrival time to next RSU). Indeed, in such a scenario, the last encountered RSU for a vehicle will be the next RSU to meet for a vehicle heading on the opposite direction. We use it in the selection of a neighboring vehicle in V2V data offloading. It is calculated as:

$$T^{arr}_{i,RSU} = \frac{l^{rsu}_j}{V_i} \tag{5}$$

where $i \in \mathcal{N}$ is a vehicle approximating the expected time to reach the next RSU, l^{rsu}_j is the distance of last connected RSU with a neighboring Vehicle $j \in \mathcal{N}$ heading on the opposite direction and V_i is the speed of Vehicle i.

5. DIVINE: Proposed Solution

This section presents our proposed solution, DIVINE. It first presents an overview of DIVINE followed by QoS provisioning. Then it presents algorithms and procedures of node (RSU/vehicle) selection for data offloading request, processing of data offloading request at RSUs and vehicles, and data offloading decision. Finally, it presents illustrative examples for better understanding of DIVINE.

5.1. Overview

A vehicle carries multiple types of data having $\mathcal{P} = \{high, medium, low\}$ priorities that needs to be offloaded with respect to the priority to RSU either directly or through other vehicles with QoS provisioning, i.e., the offloading will be performed with respect to the priority of the data (see Section 5.2.1), with overload control at RSU (see Section 5.2.2) and with admission control at both RSUs and vehicles (see Section 5.2.3).

When vehicles have connectivity with RSUs, they use V2I data offloading, while when they do not have connectivity with RSUs but have connectivity with other vehicles, they may offload their data to RSUs through other vehicles using V2V data offloading in a distributed manner within some conditions. In V2V data offloading, a vehicle may select a neighboring vehicle heading either on the same or opposite direction. When vehicles neither have connectivity with RSUs nor with other vehicles, they store their data, keep it and when they

find RSUs or other vehicles in their coverage, they offload their data to them. When a vehicle wants to offload its data through V2I or V2V data offloading, it first needs to obtain the permission grant from RSU or vehicle by sending a data offloading request, respectively, for each type of its data in order to ensure that higher priority data gets offloaded before lower priority data (see Section 5.3). Subsequently, RSUs or vehicles processes the data offloading request and decides whether and how much data to grant and sends a data offloading reply back to the vehicle (see Section 5.4). Finally, on permission granted, the vehicle takes offloading decision and starts data offloading (see Section 5.5).

5.2. Quality of Service (QoS) Provisioning

The QoS can be provisioned through various functions, such as traffic classification, admission control, traffic conditioning, scheduling and overload control [20]. DIVINE uses three functions: traffic classification, overload control and admission control for QoS provisioning.

5.2.1. Traffic classification

The data carried by the vehicles is classified into three priorities: high (urgent data, such as accident and emergency health data), medium (standard data, such as traffic conditions, periodic health monitoring data, and data streaming, i.e., online gaming, multimedia applications and video conferencing), and low (delay-tolerant data, such as advertisement, updates and backup data). The QoS is provisioned based on the priority and the size of data. For instance, a vehicle first offloads the high priority data, then the medium priority data and finally the low priority data. If a vehicle has multiple types of data with the same priority, it uses a greedy approach and first offloads the data having the largest size.

5.2.2. Overload control

We use overload control in order not to overload the RSUs and keep RSUs' resources available for future vehicles having high priority data. For this purpose, we define an offload criterion by considering threshold values of the maximum allowed load at RSU for medium and low priority data (e.g., $\gamma_k^{max,med}$ and $\gamma_k^{max,low}$ are the threshold values of maximum allowed medium and low priority data, respectively) for RSU k. For instance, an RSU k will allow a Vehicle i having medium priority data to offload its data if the current load ϑ_k at RSU k is below the threshold value of maximum allowed medium priority data $\gamma_k^{max,med}$ (i.e., $\vartheta_k < \gamma_k^{max,med}$). Similar is the case for low priority data. Note that there is no threshold value of maximum allowed high priority data because an RSU k will always try to service vehicles having high priority data even if it has to remove existing vehicles offloading medium or low priority data (see next Section 5.2.3).

5.2.3. Admission control

DIVINE also provisions QoS using admission control function in two ways. Firstly, an RSU has the authority to stop servicing a vehicle that is currently offloading its data in order to provide service to another vehicle. As an example, in V2I data offloading, a Vehicle i having high priority data wants to offload its data to RSU k. However, the current load ϑ_k at RSU k is already equal to the maximum tolerable load ϑ_k^{max} (i.e., $\vartheta_k = \vartheta_k^{max}$). In this case, in order to allow the Vehicle i to offload its high priority data, the RSU k will stop servicing one of the Vehicle j that is currently offloading its low priority data (or medium priority data if there is no vehicle with low priority data) to reduce its current load ϑ_k and will grant permission to Vehicle i to offload its high priority data. Similarly, in V2V data offloading, if an offloaded vehicle j is receiving medium priority data from a neighboring vehicle l and it receives a new data offloading request for high priority data from another neighboring vehicle i, then the offloaded vehicle j has the authority to stop servicing currently offloading vehicle l having medium priority data in order to allow the new vehicle i to offload its high priority data.

Secondly, in V2V data offloading, a vehicle can only offload its high and medium priority data to other vehicles. It will not offload its low priority data to other vehicles, instead it will keep it and will offload it to RSU through V2I data offloading when it gets connectivity to RSU.

5.3. Node Selection for Data Offloading Request

Algorithm 1 presents the procedure of selection of node (i.e., RSU or neighboring vehicle) to request data offloading. There are two cases depending on the connectivity of Vehicle i with RSU or other neighboring vehicles. A Vehicle i is aware about its neighboring nodes nb_i, as well as the neighboring nodes nb_i^{sent} to whom it has requested data offloading within offloading request lifetime $T_{offl,req}^{lifetime}$ interval. The list nb_i^{sent} is used to avoid requesting data offloading again to neighboring nodes that are requested within $T_{offl,req}^{lifetime}$ interval. A vehicle first sends a data offloading request to RSU or neighboring vehicles in order to obtain permission to offload its data.

5.3.1. Case 1: Direct Connectivity with RSU

In the first case, as presented in Part I of the algorithm, when a Vehicle i has direct connectivity with RSU k (i.e., $k \in nb_i$), it does not check its neighboring vehicles. Rather, it requests V2I data offloading by sending a data offloading request to RSU k by informing the RSU k about its parameters (e.g., speed V_i and data rate r_i) and the sizes of each priority data it wants to offload (i.e., S_i^{high}, S_i^{med} and S_i^{low}). Subsequently, it waits for a time duration $T_{i,v2i}^{wait}$ to receive a data offloading reply from RSU k.

5.3.2. Case 2: No Connectivity with RSU But Direct Connectivity with Neighboring Vehicles

In the second case, a Vehicle i has no connectivity with RSU but has direct connectivity with neighboring vehicles (nb_i) heading either on the same or oppo-

Algorithm 1 Node selection for data offloading request.

Vehicle i is aware about nb_i, nb_i^{sent};
/ Part I: V2I data offloading request to RSU k. */*
if $k \in nb_i$ **then**
    Send V_i, r_i, S_i^{high}, S_i^{med} and S_i^{low} to RSU k;
    Wait for $T_{i,v2i}^{wait}$ duration;
/ Part II: Selection of neighboring vehicle for V2V data offloading request. */*
else
    Calculate $T_{nb_i,RSU}^{arr}$ using Eq. (5);
    $nb_i^{sort} \leftarrow$ Sort nb_i based on $min\ (T_{nb_i,RSU}^{arr})$;
    $reqSent \leftarrow false$;
    while (!$reqSent$ AND $nb_i^{sort} \neq \emptyset$) **do**
        $j \leftarrow u \in nb_i \mid T_{u,RSU}^{arr} = \min_{u \in nb_i} T_{u,RSU}^{arr}$;
        $nb_i.remove(j)$;
        if $j \notin nb_i^{sent}$ **then**
            if $(H_i = H_j)$ **then**
                Calculate $T_{i,j}^{c}$ using Eq. (2);
            else if $(H_i \neq H_j)$ **then**
                Calculate $T_{i,j}^{c}$ using Eq. (3);
            end if
            if $(T_{j,RSU}^{arr} \geq T_{i,RSU}^{arr})$ OR $(|T_{i,RSU}^{arr} - T_{j,RSU}^{arr}| < \Omega_{RSU}^{arr})$ **then**
            / Quit because data offloading is not preferable. */*
                return;
            else if $(T_{i,j}^{c} < \omega_{v2v}^{min,c})$ **then**
                continue;
            else
                Send data offloading request to Vehicle j;
                $reqSent \leftarrow true$;
                $nb_i^{sent} \leftarrow nb_i^{sent} \cup j$;
                Wait for $T_{i,v2v}^{wait}$ duration;
            end if
        end if
    end while
end if

site direction and the Vehicle i checks whether to request V2V data offloading. Just to recall, in V2V data offloading, the vehicles only offload their high and medium priority data, while low priority data is offloaded only through V2I data offloading. As presented in Part II of the algorithm, Vehicle i first calculates the expected time to reach the next RSUs $T_{nb_i,RSU}^{arr}$ for its neighboring vehicles nb_i using Eq. (5). It then checks its neighboring vehicles from nb_i one by one until either it finds a suitable neighboring vehicle to request data offloading (i.e., $reqSent = true$) or it checks all the neighboring vehicles and no more neighboring vehicles are available ($nb_i^{sort} = \emptyset$).

The Vehicle i selects a Vehicle j from neighboring vehicles nb_i having the least expected time to reach the next RSU (i.e., $j \leftarrow u \in nb_i \mid T_{u,RSU}^{arr} =$

$\min_{u \in nb_i} T^{arr}_{u,RSU}$). Depending on the heading direction of neighboring Vehicle j, the Vehicle i calculates its connectivity time $T^{c}_{i,j}$ with neighboring Vehicle j using Eq. (2) or (3), if the heading direction of both vehicles is same (i.e., $H_i = H_j$) or opposite (i.e., $H_i \neq H_j$), respectively. Subsequently, Vehicle i checks whether data offloading is preferable. There are three scenarios in which V2V data offloading is not preferred. Firstly, if the expected time to reach the next RSU $T^{arr}_{j,RSU}$ for neighboring Vehicle j is greater than or equal to the expected time to reach the next RSU $T^{arr}_{i,RSU}$ for Vehicle i itself, it means that the Vehicle i can reach the next RSU earlier than or at the same time as neighboring Vehicle j. Secondly, the difference of the expected time to reach RSUs for Vehicles i and j is lower than the threshold value Ω^{arr}_{RSU} of the allowed difference of reaching RSU for two vehicles in V2V data offloading which means that there is a slight difference for Vehicle i and j to reach RSUs. Since the list nb^{sort}_i is sorted based on the minimum expected time for neighboring vehicles to reach the next RSU, therefore it is obvious that all the subsequent vehicles in the list nb^{sort}_i will also have greater expected time to reach the next RSU than Vehicle i, as well as the difference of expected time to reach RSUs for Vehicles i and subsequent vehicles in the list nb^{sort}_i will be lower than the threshold value Ω^{arr}_{RSU}. Hence, Vehicle i quits and does not request V2V data offloading because it can offload its data directly to RSU through V2I data offloading sooner. Thirdly, if the connectivity time of Vehicles i and j is lower than the threshold value $\omega^{min,c}_{v2v}$ of the least allowed contact duration between vehicles for V2V data offloading, it means that the connectivity time is too short and it is better not to offload the data. In this scenario, the Vehicle i checks the next neighboring vehicle in the list nb^{sort}_i.

Otherwise, it is preferred to exploit V2V data offloading and hence, Vehicle i sends the data offloading request to Vehicle j. It then sets the flag $reqSent$ to $true$ and adds the Vehicle j into requested neighbor list nb^{sent}_i. Finally, Vehicle i waits for a time duration $T^{wait}_{i,v2v}$ to receive a data offloading reply from Vehicle j.

5.4. Processing of Data Offloading Request

Once data offloading is requested by Vehicle i to either RSU k or neighboring Vehicle j, it is being processed differently by RSU k and neighboring Vehicle j. The RSU k analyzes the sizes of each priority data at Vehicle i and calculates which and how much data it can allow Vehicle i to offload. On the other hand, Vehicle j analyzes its current situation and informs the Vehicle i about the maximum data that Vehicle i can offload to it. The Vehicle i then calculates the actual data sizes that it can offload to Vehicle j. We discuss each case below.

5.4.1. Processing of Data Offloading Request at RSU

Algorithm 2.1 presents the procedure of processing of data offloading request at RSU k received from Vehicle i. RSU k first calculates its connectivity time $T^{c}_{i,k}$ with Vehicle i and the offloading capacity (i.e., the maximum allowed data size) $\mathcal{S}^{max}_{i,k}$ that Vehicle i can offload to it using Eqs. (1) and (4), respectively.

In Part I of the algorithm, the RSU k performs admission control for enabling Vehicle i to offload high priority data if the current load at RSU k is maximum. If the Vehicle i has high priority data (i.e., $S_i^{high} > 0$), the RSU k checks the amount of current high priority data $S_{i,k}^{curr}$ that Vehicle i can offload. If the size of high priority data S_i^{high} of Vehicle i is less than the offloading capacity $\mathcal{S}_{i,k}^{max}$, it sets the current amount of data $S_{i,k}^{curr}$ to be S_i^{high} (i.e., $S_{i,k}^{curr} \leftarrow S_i^{high}$), otherwise, it sets $S_{i,k}^{curr} \leftarrow \mathcal{S}_{i,k}^{max}$ because the Vehicle i cannot offload more data than the offloading capacity $\mathcal{S}_{i,k}^{max}$. Subsequently, RSU k performs admission control (see Section 5.2.3) if the sum of its current load ϑ_k and the amount of current data $S_{i,k}^{curr}$ that Vehicle i needs to offload is above the maximum tolerable load ϑ_k^{max} (i.e., $\vartheta_k + S_{i,k}^{curr} \geq \vartheta_k^{max}$). RSU k selects an existing Vehicle j that is offloading its data and stops providing service to it (partially or completely) in order to reduce its current load so that the Vehicle i can be able to offload its high priority data. Vehicle j will be the one offloading its low priority data and if there is no vehicle offloading low priority data, then the Vehicle j will be the one offloading its medium priority data. RSU k informs Vehicle j about the service termination so that Vehicle j can be aware about it and can offload its remaining data to RSU k in case of partial service termination or can offload its data to another RSU or vehicle in case of complete service termination. Subsequently, RSU k updates its current load ϑ_k by subtracting the current load $S_{j,k}^{curr}$ of removed Vehicle j (i.e., $\vartheta_k \leftarrow \vartheta_k - S_{j,k}^{curr}$). RSU k repeats this process until its current load enables it to allow Vehicle i to offload its high priority data. Note that if there is no Vehicle j offloading the low or medium priority data, it means that all the existing vehicles are offloading their high priority data, and hence the RSU k will inform Vehicle i that it cannot offload its data.

In Part II, RSU k analyzes the sizes of high, medium and low priority data that Vehicle i requested to offload and decides which priority data and how much data it can grant Vehicle i to offload. If the size of high priority data S_i^{high} requested by Vehicle i is higher than the offloading capacity $\mathcal{S}_{i,k}^{max}$ of i, RSU k sets the granted set $\Phi_{k,i}^{grant}$ (containing the sizes of different priority data that Vehicle i can offload) equal to the offloading capacity (i.e., $\Phi_{k,i}^{grant} \leftarrow S_i^{high} \mid S_i^{high} = \mathcal{S}_{i,k}^{max}$). Otherwise, it grants to offload all the high priority data and sets it into the granted set $\Phi_{k,i}^{grant}$ (i.e., $\Phi_{k,i}^{grant} \leftarrow S_i^{high}$). Subsequently, if the size of granted set $\Phi_{k,i}^{grant}$ is lower than the offloading capacity $\mathcal{S}_{i,k}^{max}$ (i.e., $\Phi_{k,i}^{grant} < \mathcal{S}_{i,k}^{max}$) and the sum of current load ϑ_k at RSU k and granted set $\Phi_{k,i}^{grant}$ is below the threshold value $\gamma_k^{max,med}$ of the maximum allowed medium priority data (i.e., $\vartheta_k + \Phi_{k,i}^{grant} < \gamma_k^{max,med}$), it means that RSU k can also take medium priority data from Vehicle i. It then takes as much data it can favoring medium priority data over low priority data.

Finally, RSU k fetches the granted sizes of high, medium and low priority data from granted set $\Phi_{k,i}^{grant}$, calculates the current data size $S_{i,k}^{curr}$ that Vehicle i will offload to RSU k by taking their sum (i.e., $S_{i,k}^{curr} \leftarrow S_i^{high} + S_i^{med} + S_i^{low}$) and updates it current load ϑ_k by adding the current Vehicle i's granted data

Algorithm 2.1 Processing of data offloading request at RSU k from Vehicle i.

RSU k calculates $T^c_{i,k}$ and $\mathcal{S}^{max}_{i,k}$ using Eqs. (1) and (4) resp.;
$\Phi^{grant}_{k,i} \leftarrow \emptyset$;
/ Part I: Admission control.*/*
if $S^{high}_i > 0$ **then**
    $S^{curr}_{i,k} \leftarrow (S^{high}_i \leq \mathcal{S}^{max}_{i,k})\ ?\ S^{high}_i : \mathcal{S}^{max}_{i,k}$;
    while $(\vartheta_k + S^{curr}_{i,k} \geq \vartheta^{max}_k)$ **do**
        Found $\leftarrow$ RSU k selects existing Vehicle $j \mid S^{low}_i > 0$;
        if not Found **then**
            Found $\leftarrow$ RSU k selects existing Vehicle $j \mid S^{med}_i > 0$;
        end if
        if not Found **then**
            Declines data offloading request from Vehicle i; Return;
        end if
        Informs Vehicle j about service termination and stops servicing it;
        $\vartheta_k \leftarrow \vartheta_k - S^{curr}_{j,k}$;
    end while
end if
/ Part II: Check which priority data and how much data can be allowed to offload.*/*
if $S^{high}_i > \mathcal{S}^{max}_{i,k}$ **then**
    $\Phi^{grant}_{k,i} \leftarrow S^{high}_i \mid S^{high}_i = \mathcal{S}^{max}_{i,k}$;
else
    $\Phi^{grant}_{k,i} \leftarrow S^{high}_i$;
end if
if $(\Phi^{grant}_{k,i} < \mathcal{S}^{max}_{i,k})$ and $(\vartheta_k + \Phi^{grant}_{k,i} < \gamma^{max,med}_k)$ **then**
    if $\Phi^{grant}_{k,i} + S^{med}_i > \mathcal{S}^{max}_{i,k}$ **then**
        $\Phi^{grant}_{k,i} \leftarrow \Phi^{grant}_{k,i} \cup S^{med}_i \mid S^{med}_i = \mathcal{S}^{max}_{i,k} - \Phi^{grant}_{k,i}$;
    else
        $\Phi^{grant}_{k,i} \leftarrow \Phi^{grant}_{k,i} \cup S^{med}_i$;
    end if
end if
if $(\Phi^{grant}_{k,i} < \mathcal{S}^{max}_{i,k})$ and $(\vartheta_k + \Phi^{grant}_{k,i} < \gamma^{max,low}_k)$ **then**
    if $\Phi^{grant}_{k,i} + S^{low}_i > \mathcal{S}^{max}_{i,k}$ **then**
        $\Phi^{grant}_{k,i} = \Phi^{grant}_{k,i} \cup S^{low}_i \mid S^{low}_i = \mathcal{S}^{max}_{i,k} - \Phi^{grant}_{k,i}$;
    else
        $\Phi^{grant}_{k,i} = \Phi^{grant}_{k,i} \cup S^{low}_i$;
    end if
end if
Fetch S^{high}_i, S^{med}_i and S^{low}_i from $\Phi^{grant}_{k,i}$;
$S^{curr}_{i,k} \leftarrow S^{high}_i + S^{med}_i + S^{low}_i$;
$\vartheta_k \leftarrow \vartheta_k + S^{curr}_{i,k}$;
Send $\Phi^{grant}_{k,i}$ to Vehicle i;

size to its current load (i.e., $\vartheta_k \leftarrow \vartheta_k + S^{curr}_{i,k}$). It then sends the granted set $\Phi^{grant}_{k,i}$ as data offloading reply to Vehicle i.

5.4.2. Processing of Data Offloading Request at Vehicle

Algorithm 2.2 presents the processing of a data offloading request from Vehicle i received by Vehicle j. This latter has to decide whether to accept or reject it. Part I of the algorithm presents the first scenario in which Vehicle j is already receiving offloaded data from another neighboring Vehicle l. In this scenario, Vehicle j analyzes data priority of both vehicles and similarly to offloading to RSU process, may decide to stop servicing l if its data is of lower priority than the ones of j and serves j instead (see Section 5.2.3). If so, Vehicle j sets the granted set $\Phi_{j,i}^{grant}$ by calculating the maximum allowed data sizes of high and medium priority data that it can allow Vehicle i to offload (i.e., using CalculateMaxSizes( ) function, discussed in the next paragraph). It then sends the request acceptance confirmation and granted set $\Phi_{j,i}^{grant}$ as data offloading reply to Vehicle i, calculates the time duration $T_{i,j}^{ofld}$ required by Vehicle i to offload its data $S_{i,j}^{*}$ and informs the Vehicle l to try again later after $T_{i,j}^{ofld}$ duration.

Otherwise, if both vehicles l and i have same priority level data, Vehicle j calculates the time duration $T_{l,j}^{ofld}$ required by Vehicle l to offload its data $S_{l,j}^{*}$ to Vehicle j and informs the Vehicle i to try again later after $T_{l,j}^{ofld}$ duration.

The calculation of the maximum allowed data sizes (i.e., CalculateMaxSizes( ) function) of high and medium priority data (i.e., $S_{i,j}^{max,high}$ and $S_{i,j}^{max,med}$) that Vehicle j can allow Vehicle i to offload works as follows. Vehicle j first estimates the offloading capacity S_{j}^{max} that it can offload to an RSU (in the future) and calculates the maximum allowed size of high priority data $S_{i,j}^{max,high}$ that it can allow Vehicle i to offload, taking in priority higher priority data.

Part II and III of the algorithm present the second and third scenarios in which Vehicle j is offloading its data to RSU k and another Vehicle m, respectively. In both scenarios, Vehicle j calculates the time duration $T_{j,k}^{ofld}$ and $T_{j,m}^{ofld}$ that it requires to offload its data $S_{j,k}^{*}$ and $S_{j,m}^{*}$ to RSU k and Vehicle m, respectively. Accordingly, it informs Vehicle i to try again later after $T_{j,k}^{ofld}$ and $T_{j,m}^{ofld}$ duration for second and third scenarios, respectively.

Finally, as presented in Part IV, if Vehicle j is neither offloading its data to RSU or other neighboring vehicle nor receiving the offloaded data from another neighboring vehicle, the Vehicle j sets the granted set $\Phi_{j,i}^{grant}$ by calculating the maximum allowed data sizes of high and medium priority data that it can allow Vehicle i to offload (i.e., using CalculateMaxSizes( ) function, discussed in Part I). It then sends the request acceptance confirmation and granted set $\Phi_{j,i}^{grant}$ as data offloading reply to Vehicle i.

5.5. Data Offloading Decision

When a Vehicle i receives data offloading reply from RSU k or neighboring Vehicle j, it decides and starts the data offloading which procedure is presented in Algorithm 3.

Algorithm 2.2 Processing of offloading request at Vehicle j.

Received data offloading request from Vehicle i;
/ Part I: Vehicle j is receiving offloaded data from another neighboring Vehicle l. */*
if (Vehicle j is receiving offloaded data from Vehicle l) **then**
    if ($S_l^{high} = 0$ and $S_i^{high} > 0$) **then**
        Inform Vehicle l about service termination;
        Stop servicing Vehicle l;
        $\Phi_{j,i}^{grant} \leftarrow$ CalculateMaxSizes( );
        Send $\Phi_{j,i}^{grant}$ to Vehicle i;
        Calculate $T_{i,j}^{ofld}$ from $S_{i,j}^{*}$;
        Inform Vehicle l to try again later after $T_{i,j}^{ofld}$ duration;
    else if ($S_l^{high} > 0$ and $S_i^{high} > 0$) OR ($S_l^{high} = \emptyset$ and $S_i^{high} = \emptyset$) **then**
        Calculate $T_{l,j}^{ofld}$ from $S_{l,j}^{*}$;
        Inform Vehicle i to try again later after $T_{l,j}^{ofld}$ duration;
    end if
/ Part II: Vehicle j is offloading its data to RSU k.*/*
else if (Vehicle j is offloading its data to RSU k) **then**
    Calculate $T_{j,k}^{ofld}$ from $S_{j,k}^{*}$;
    Inform Vehicle i to try again later after $T_{j,k}^{ofld}$ duration;
/ Part III: Vehicle j is offloading its data to another neighboring vehicle.*/*
else if (Vehicle j is offloading its data to Vehicle m) **then**
    Calculate $T_{j,m}^{ofld}$ from $S_{j,m}^{*}$;
    Inform Vehicle i to try again later after $T_{j,m}^{ofld}$ duration;
/ Part IV: Vehicle j is neither offloading its data to RSU or other vehicle nor receiving the offloaded data from other vehicle. */*
else
    $\Phi_{j,i}^{grant} \leftarrow$ CalculateMaxSizes( );
    Send $\Phi_{j,i}^{grant}$ to Vehicle i;
end if

function CalculateMaxSizes( )
    $S_j^{max} \leftarrow$ Estimated offloading capacity that Vehicle j can offload to RSU using Eq. (4);
    $S_{i,j}^{max,high} = 0$; $S_{i,j}^{max,med} = 0$;
    if ($S_j^{max} > S_j^{high}$) **then**
        $S_{i,j}^{max,high} \leftarrow S_j^{max} - S_j^{high}$;
        if ($S_{i,j}^{max,high} > S_i^{high}$) **then**
            $S_{i,j}^{max,high} \leftarrow S_i^{high}$;
        end if
        $allocated \leftarrow S_j^{high} + S_{i,j}^{max,high} + S_j^{med}$;
        if ($S_j^{max} > allocated$) **then**
            $S_{i,j}^{max,med} \leftarrow S_j^{max} - allocated$;
            if ($S_{i,j}^{max,med} > S_i^{med}$) **then**
                $S_{i,j}^{max,med} \leftarrow S_i^{med}$;
            end if
        end if
    end if
    $\Phi_{j,i}^{grant} \leftarrow S_{i,j}^{max,high} \cup S_{i,j}^{max,med}$;
    return $\Phi_{j,i}^{grant}$;
end function

Algorithm 3 Data offloading decision.

Received data offloading reply $\Phi_{j,k}^{grant}$ from RSU k or $\Phi_{j,i}^{grant}$ from Vehicle j
*/*Part I: Received data offloading reply from RSU k*/*
if received $\Phi_{j,k}^{grant}$ from RSU k **then**
    Fetch $S_i^{grant,high}$, $S_i^{grant,med}$, $S_i^{grant,low}$ from $\Phi_{j,k}^{grant}$;
    Offload S_i^{high}, S_i^{med}, S_i^{low} to RSU $k \mid S_i^{high} = S_i^{grant,high}$, $S_i^{med} = S_i^{grant,med}$, $S_i^{low} = S_i^{grant,low}$;
    Update $S_i^{high} \leftarrow S_i^{high} - S_i^{grant,high}$;
    Update $S_i^{med} \leftarrow S_i^{med} - S_i^{grant,med}$;
    Update $S_i^{low} \leftarrow S_i^{low} - S_i^{grant,low}$;
*/*Part II: Received data offloading reply from Vehicle j*/*
else if received $\Phi_{j,i}^{grant}$ from Vehicle j **then**
    Fetch $S_{i,j}^{max,high}$ and $S_{i,j}^{max,med}$ from $\Phi_{j,i}^{grant}$;
    if $(H_i = H_j)$ **then**
        Calculate $T_{i,j}^{c}$ using Eq. (2);
    else if $(H_i \neq H_j)$ **then**
        Calculate $T_{i,j}^{c}$ using Eq. (3);
    end if
    Calculate $\mathcal{S}_{i,j}^{max}$ using Eq. (4);
    if $(\mathcal{S}_{i,j}^{max} \geq (S_{i,j}^{max,high} + S_{i,j}^{max,med})$ **then**
        / Offload all the allowed data. */*
        Offload S_i^{high} and S_i^{med} to Vehicle $j \mid S_i^{high} = S_{i,j}^{max,high}$ and $S_i^{med} = S_{i,j}^{max,med}$;
        Update $S_i^{high} \leftarrow S_i^{high} - S_{i,j}^{max,high}$;
        Update $S_i^{med} \leftarrow S_i^{med} - S_{i,j}^{max,med}$;
    else if $(\mathcal{S}_{i,j}^{max} \leq S_{i,j}^{max,high})$ **then**
        / Offload high priority data equal to the allowed data size. */*
        Offload S_i^{high} to Vehicle $j \mid S_i^{high} = \mathcal{S}_{i,j}^{max}$;
        Update $S_i^{high} \leftarrow S_i^{high} - \mathcal{S}_{i,j}^{max}$;
    else if $(\mathcal{S}_{i,j}^{max} < S_{i,j}^{max,high} + S_{i,j}^{max,med})$ **then**
        / Offload high and medium priority data equal to the allowed data size. */*
        Offload S_i^{high} to Vehicle $j \mid S_i^{high} = S_{i,j}^{max,high}$;
        Offload S_i^{med} to Vehicle $j \mid S_i^{med} = \mathcal{S}_{i,j}^{max} - S_{i,j}^{max,high}$;
        Update $S_i^{high} \leftarrow S_i^{high} - S_{i,j}^{max,high}$;
        Update $S_i^{med} \leftarrow S_i^{med} - (\mathcal{S}_{i,j}^{max} - S_{i,j}^{max,high})$;
    end if
end if

5.5.1. Processing of Data Offloading Reply from RSU

Part I of the algorithm shows the procedure when Vehicle i receives data offloading reply from RSU k. In this case, the RSU k has already calculated the actual size of each priority data that Vehicle i should offload to it in the granted set $\Phi_{j,k}^{grant}$. Hence, the Vehicle i fetches the granted sizes of high, medium and low priority data and offloads each priority data equal to its respective granted size to the RSU k. Finally, it updates the sizes of each priority data.

5.5.2. Processing of Data Offloading Reply from Neighboring Vehicle

Part II of the algorithm shows the procedure when Vehicle i receives data offloading reply from a neighboring Vehicle j. In this case, the Vehicle j has informed the maximum allowed data sizes of high and medium priority data that Vehicle i can offload to it in the granted set $\Phi_{j,i}^{grant}$ (instead of the actual data sizes that Vehicle i should offload, as in the case of RSU). Hence, the Vehicle i first fetches the maximum allowed data sizes of high and medium priority data (i.e., $S_{i,j}^{max,high}$ and $S_{i,j}^{max,med}$) to offload from granted set $\Phi_{j,i}^{grant}$, and calculates the offloading capacity $\mathcal{S}_{i,j}^{max}$ using Eq. (4), i.e., the maximum allowed data size that it can offload to Vehicle j within the connectivity time.

If the offloading capacity $\mathcal{S}_{i,j}^{max}$ within the connectivity time $T_{i,j}^{c}$ matches or exceeds the sum of sizes of maximum allowed high and medium priority data informed by Vehicle j (i.e., $S_{i,j}^{max,high}$ and $S_{i,j}^{max,med}$), it means that Vehicle i can offload the maximum data informed by Vehicle j, hence it offloads its high and medium priority data (S_i^{high} and S_i^{med}) to Vehicle j equal to the maximum allowed sizes of high and medium priority data ($S_{i,j}^{max,high}$ and $S_{i,j}^{max,med}$), respectively. Subsequently, it updates the data sizes of its high and medium priority data.

If the offloading capacity $\mathcal{S}_{i,j}^{max}$ is less than or equals the size of maximum allowed high priority data $S_{i,j}^{max,high}$ informed by Vehicle j, the Vehicle i only offloads its high priority data equal to the maximum allowed data size $\mathcal{S}_{i,j}^{max}$ within the connectivity time $T_{i,j}^{c}$. It then updates the size of its high priority data.

Otherwise, if the offloading capacity $\mathcal{S}_{i,j}^{max}$ within the connectivity time $T_{i,j}^{c}$ is less than the sum of data sizes of maximum allowed high and medium priority data informed by Vehicle i (i.e., $S_{i,j}^{max,high}$ and $S_{i,j}^{max,med}$), the Vehicle i offloads its high priority data equal to the size of maximum allowed high priority data $S_{i,j}^{max,high}$ informed by Vehicle j and offloads its medium priority data equal to the size of difference of offloading capacity $\mathcal{S}_{i,j}^{max}$ within the connectivity time $T_{i,j}^{c}$ and the size of maximum allowed high priority data $S_{i,j}^{max,high}$ informed by Vehicle j (i.e., $S_i^{med} = \mathcal{S}_{i,j}^{max} - S_{i,j}^{max,high}$). Subsequently, it updates the data sizes of its high and medium priority data.

5.6. Flow Chart

Figure 2 presents a flow chart for explaining the association and flow of algorithms. We hope this will help the global understanding of DIVINE.

5.7. Examples

We present illustrative examples to better understand the procedures of V2I and V2V data offloading.

5.7.1. V2I Data Offloading

An example of V2I data offloading is presented in Figure 3 in which the red Vehicle i, into our consideration of data offloading, contains multiple types of

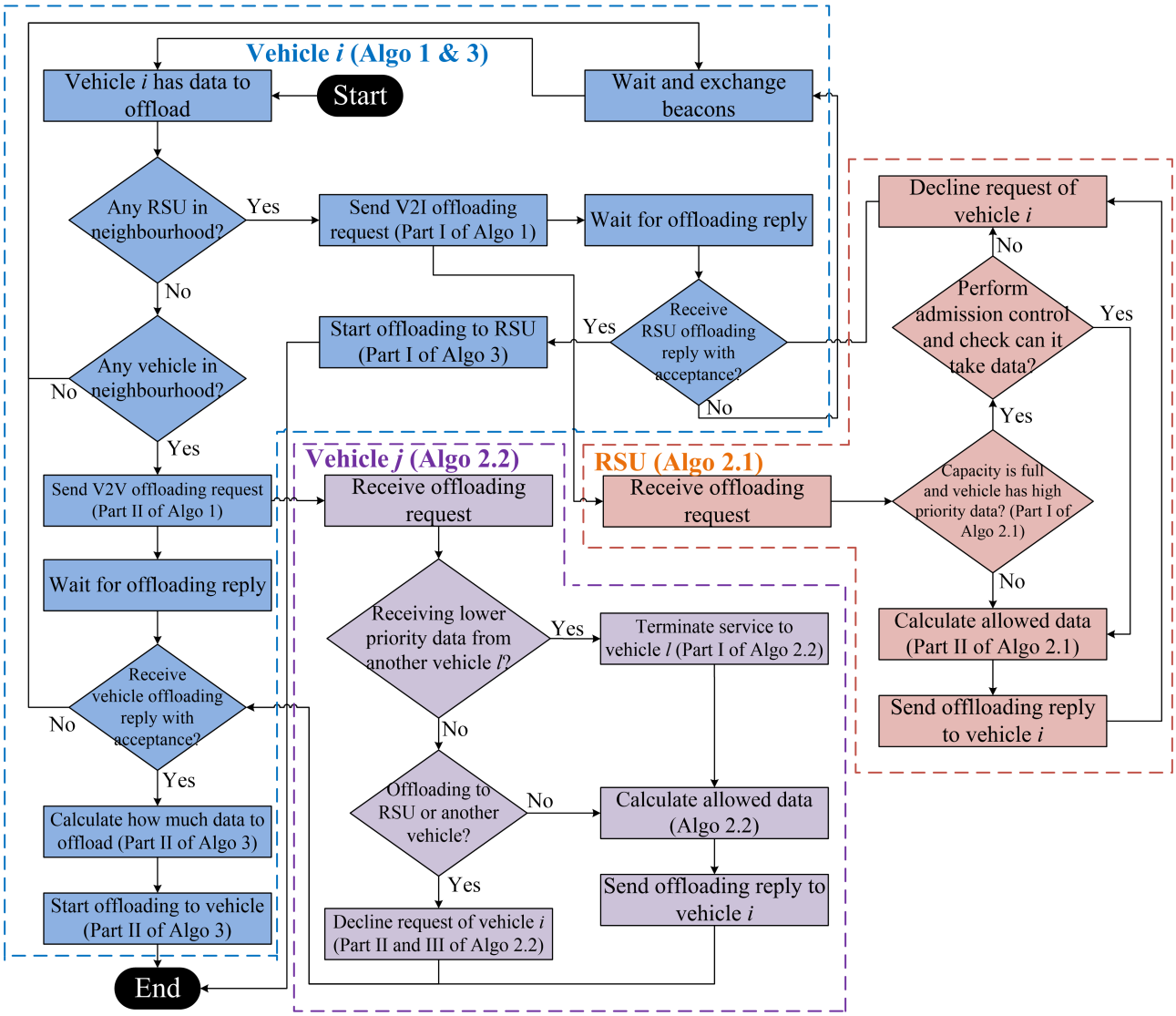


Figure 2: Flow chart.

priority data to offload and it has RSU k_1 in its coverage. The red Vehicle i executes Algorithm 1 for the node selection to request data offloading. Assuming the speed of red Vehicle i to be $V_i = 22$m/s (i.e., average vehicles speed on highways) and data rate of red Vehicle i to be $r_i = 0.75$MBps (i.e., 6Mbps using QPSK modulation scheme), following the Part I of algorithm, the red Vehicle i sends a data offloading request to RSU k_1 containing its parameters (speed and data rate) and the sizes of each priority data to offload for obtaining data offloading permission.

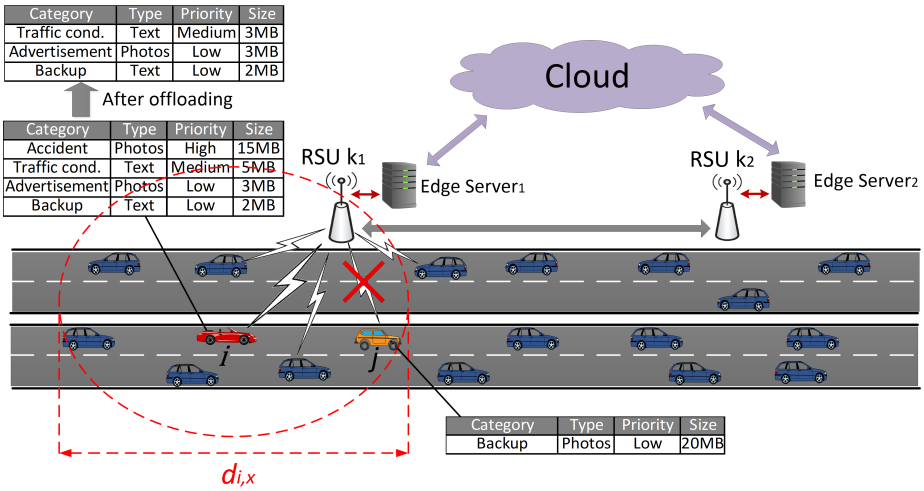


Figure 3: An example of processing of data offloading request at RSU.

Upon reception of the data offloading request, the RSU k_1 executes Algorithm 2.1. Assuming the coverage of red Vehicle i with RSU k_1 on lane x in which the red vehicle i is moving to be $d_{k_1,x} = 500$m, registration time $\tau_{i,k}$ and waiting time $T^{wait}_{i,v2i}$ to be negligible (in this example for simplicity), the RSU k_1 calculates its connectivity time $T^c_{i,k}$ with red Vehicle i and the maximum offloading capacity $\mathcal{S}^{max}_{i,k}$ using Eqs. (1) and (4). Suppose that the current load ϑ_{k_1} at RSU k_1 is above the maximum tolerable load $\vartheta^{max}_{k_1}$, hence the RSU k_1 performs admission control by following Part I of the algorithm. It stops servicing an orange vehicle j that is offloading only low priority data in order to allow red Vehicle i to offload its high priority data. Subsequently, following the calculations in Part II, the RSU k_1 grants the red Vehicle i to offload 15MB of high priority data and 2MB of medium priority data and sends the granted set in data offloading reply.

Finally, upon receiving the data offloading reply from RSU k_1, the red Vehicle i executes the Algorithm 3 of data offloading decision and following the Part I, it offloads its data equal to the granted data sizes to RSU k_1. The figure shows both the initial data before offloading and the remaining data after offloading.

For the sake of simplicity, in Figure 3 and other following figures, we present the coverage of Vehicle i with RSU on lane x to be $d_{*,x}$, where $*$ represents the vehicle and x represents the lane on which the vehicle into our consideration is moving.

5.7.2. V2V Data Offloading

Two examples of V2V data offloading are presented in Figs. 4 and 5 in which the red vehicle (Vehicle i), into our consideration of data offloading, has no RSU in its coverage but has two neighboring vehicles: an orange vehicle (Vehicle j_1) heading on the same direction and a black vehicle (Vehicle j_2) heading on the

opposite direction. In Figure 4, the red vehicle i selects a neighboring orange vehicle j_1 heading on the same direction, while in Figure 5, the red vehicle i selects a neighboring black vehicle j_2 heading on the opposite direction for V2V data offloading. Let us take the following assumptions for the sake of simplicity: the red Vehicle i has not recently sent data offloading requests to orange Vehicle j_1 and black Vehicle j_2 (i.e., $nb_i^{sent} = \emptyset$), the connectivity time of red Vehicle i with both neighboring Vehicles j_1 and j_2 is sufficient (i.e., $T^c_{i,j_1} \geq \omega^{min,c}_{v2v}$ and $T^c_{i,j_2} \geq \omega^{min,c}_{v2v}$), the coverage of each vehicle is $d_{*\in\mathcal{N}} = 500$m, the speeds of red Vehicle i and black Vehicle j_2 are $V_i = V_{j_2} = 22$m/s, the speed of orange Vehicle j_1 is $V_{j_1} = 27$m/s, the data rate of each vehicle is $r_{*\in\mathcal{N}} = 0.75$MBps (i.e., 6Mbps using QPSK modulation scheme) and the registration time τ_{i,j_*} and waiting time $T^{wait}_{i,v2v}$ to be neglible. According to Part II of Algorithm 1, the red Vehicle i selects a neighboring vehicle to request data offloading based on the least expected time to reach the RSU for neighboring vehicle using Eq. (5) and its connectivity time with neighboring vehicle. Hence, in Figure 4, the red Vehicle i selects orange Vehicle j_1, while in Figure 5, the red Vehicle i selects black Vehicle j_2. Subsequently, it sends data offloading requests to them for obtaining data offloading permission and starts waiting for data offloading reply from them.

For the next step of processing of data offloading request in this example, we will use the term Vehicle j instead Vehicle j_1 and Vehicle j_2. On receiving data offloading request, the Vehicle j executes Algorithm 2.2. Suppose that the Vehicle j is neither receiving offloaded data from another Vehicle l, nor offloading its data to RSU or other vehicle. In Part IV of the algorithm, since vehicles only offload high and medium priority data in V2V data offloading, the Vehicle j calculates the maximum size of high and medium priority data that it can allow Vehicle i to offload. For the sake of simplicity, in Figure 4 and 5, we keep the original data of both orange vehicle j_1 and black vehicle j_2 to be the same. The Vehicle j estimates the offloading capacity (i.e., the maximum possible size that it can offload to RSU in the future) to be S_j^{max}=17MB. The size of Vehicle j's high priority data is S_j^{high}=7MB, while the requested size of Vehicle i's high priority data is S_i^{high}=15MB. Hence the Vehicle j grants the red Vehicle i to offload the maximum of 10MB of high priority data and sends the granted set in data offloading reply.

Finally, on receiving a data offloading reply, the red Vehicle i executes the Algorithm 3 of data offloading decision and following the Part II, it recalculates the connectivity time using Eq. (2) or (3) depending on the heading direction of neighboring Vehicle j and the maximum offloading capacity $\mathcal{S}^{max}_{i,j}$ using Eq. (4). In the case of Figure 4, the red Vehicle i offloads all the granted 10MB of high priority data to orange Vehicle j_1, while in the case of Figure 5, the red Vehicle i offloads 8.5MB of high priority data to black Vehicle j_2. Note than 8.5MB is equal to half of the data in Figure 3 in which the red Vehicle i offloaded its data to RSU. This is because in this scenario, both vehicles are mobile having the same speed, therefore the connectivity time is half of the connectivity time with RSU in which only one vehicle was mobile and RSU was stationary.

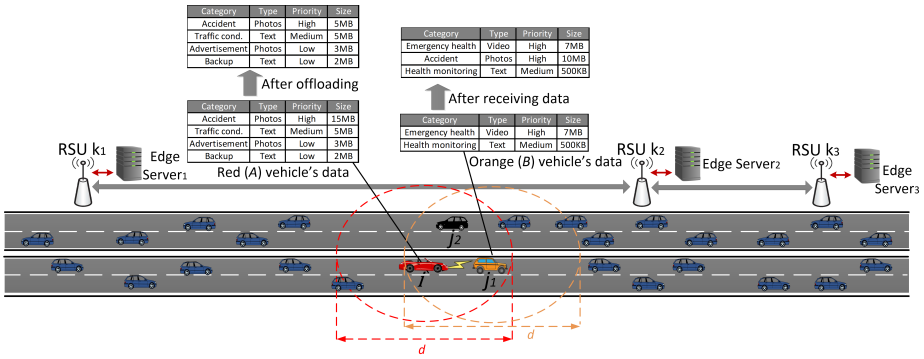


Figure 4: An example of V2V data offloading on the same heading direction.

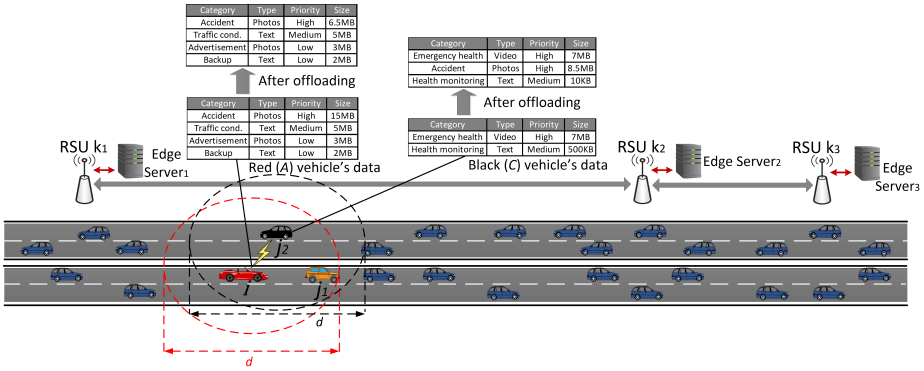


Figure 5: An example of V2V data offloading on the opposite heading direction.

6. Analysis of DIVINE Overhead

This section investigates the overhead of DIVINE, specifically the control messages overhead and time complexity. The control messages overhead M is the number of messages exchanged between the vehicles and RSUs and among the vehicles. If a vehicle sends a control message to its neighbor (RSU or vehicle), the message overhead is incremented by one [21]. The time complexity T is the number of time slots to enable a vehicle to start offloading its data, specifically, registering it with an RSU or neighboring vehicle for data offloading [22]. We assume discrete time and that the transmission of a message by a vehicle/RSU and ACK reception requires one time slot [21].

In DIVINE, time and complexity overheads are generated by three functions: beacons exchange, data offloading request and data offloading reply.

6.1. Beacons Overhead

Beacons are broadcast in each predefined interval by vehicles and RSUs so that vehicles can learn about their neighboring nodes (RSUs and vehicles). We

Table 2: Summary of DIVINE overhead

Overhead type	Message overhead	Time complexity
Beacons	$D \times r_b \times (\|\mathcal{K}\| + \|\mathcal{N}\|)$	D
Data offloading request	$[1, \|nb_i\|]$	$[T_{scan} + 1 + T^{wait}_{i,j \in \mathcal{K},\mathcal{N}}$, $T_{scan} + \|nb_i\|(1 + T^{wait}_{i,j \in \mathcal{K},\mathcal{N}})]$
Data offloading reply	$[1, 1 + \|\mathcal{N}_{ac}\|]$	$[1, 1 + \|\mathcal{N}_{ac}\|]$
Total	$[D \times r_b \times (\|\mathcal{K}\| + \|\mathcal{N}\|) + 2$, $D \times r_b \times (\|\mathcal{K}\| + \|\mathcal{N}\|) + \|nb_i\| + \|\mathcal{N}_{ac}\| + 1]$	$[D + T_{scan} + T^{wait}_{i,j \in \mathcal{K},\mathcal{N}} + 2$, $D + T_{scan} + \|nb_i\|(1 + T^{wait}_{i,j \in \mathcal{K},\mathcal{N}}) + \|\mathcal{N}_{ac}\| + 1]$

denote the number of beacons transmitted by a vehicle/RSU per time slot as rate r_b and the network lifetime is equal to D time slots. The overhead incurred in broadcasting the beacons per time slot at rate r_b by the total number of vehicles and RSUs is $|\mathcal{K}| \times r_b$ and $|\mathcal{N}| \times r_b$, respectively. To sum up, the message overhead of total beacons is $M = D \times r_b \times (|\mathcal{K}| + |\mathcal{N}|)$ and the time complexity is $T = D$ time slots.

6.2. Data Offloading Request Overhead

Initially, each vehicle has no knowledge about its neighbors (RSUs and vehicles) and it listens to beacons to discover the neighbors. We denote such a scanning period as T_{scan}. Hence, each vehicle must wait at least T_{scan} time slots before sending a data offloading request. We denote the number of neighbors of a vehicle as $|nb_i|$. After T_{scan} period, a vehicle i sends a data offloading request to a neighbor and waits for $T^{wait}_{i,j \in \mathcal{K},\mathcal{N}}$ duration to receive a data offloading reply. It takes at least $T = T_{scan} + 1 + T^{wait}_{i,j \in \mathcal{K},\mathcal{N}}$ time slots and $M = 1$ message if the first neighbor accepts its data offloading request, and at most $T = T_{scan} + |nb_i|(1 + T^{wait}_{i,j \in \mathcal{K},\mathcal{N}})$ time slots and $M = |nb_i|$ messages if the last neighbor accepts its data offloading request.

6.3. Data Offloading Reply Overhead

When an RSU or a vehicle receives a data offloading request, it processes it and sends a data offloading reply (accepting or declining the request) back to the vehicle. An RSU or a vehicle may need to perform admission control. We denote the average number of vehicles that an RSU/vehicle has to stop servicing during admission control as $|\mathcal{N}_{ac}|$. For data offloading reply, it takes at least $T = 1$ time slot and $M = 1$ message if admission control is not required. Otherwise, it takes at most $T = 1 + |\mathcal{N}_{ac}|$ time slots and $M = 1 + |\mathcal{N}_{ac}|$ messages.

6.4. Total Overhead

The total overhead is summarized in Table 2. To sum up, the total overhead of DIVINE is based on beacons exchange, data offloading request and data offloading reply. The lower bound of total DIVINE overhead for control messages overhead is $M = D \times r_b \times (|\mathcal{K}| + |\mathcal{N}|) + 2$ and time complexity is

$T = D + T_{scan} + T^{wait}_{i,j \in \mathcal{K},\mathcal{N}} + 2$. The upper bound of total DIVINE overhead for control messages is $M = D \times r_b \times (|\mathcal{K}| + |\mathcal{N}|) + |nb_i| + |\mathcal{N}_{ac}| + 1$ and time complexity is $T = D + T_{scan} + |nb_i|(1 + T^{wait}_{i,j \in \mathcal{K},\mathcal{N}}) + |\mathcal{N}_{ac}| + 1$.

7. Performance Evaluation, Results and Discussions

In this section, we evaluate the performance of DIVINE through extensive simulations and present simulation setup and parameters, performance metrics, results and discussions.

7.1. Simulation Setup and Parameters

DIVINE is implemented in network simulator OMNeT++ 5.5.1 [23] with Veins 5.0 [24] and SUMO 1.7.0 (Simulation for Urban Mobility) [25] frameworks. SUMO is used to create the scenarios and mobility of vehicles, while OMNeT++ and Veins are used to simulate the vehicular communications using IEEE 802.11p standard. We built a 30km two-way highway scenario in SUMO having three lanes on each direction without intersections. The vehicles are equally distributed on each lane and each vehicle is departed at a random time and location. The generated scenario is imported into OMNeT++ using Veins framework. The RSUs are uniformly distributed besides the highway. Each vehicle and RSU periodically send beacons with an interval of one second. The number of vehicles varies from 50 to 200, the number of RSUs varies from 10 to 100, the maximum speed of vehicles varies from 15m/s to 30m/s and the maximum RSU capacity varies from 2,000Mb to 10,000Mb. The default number of vehicles is 100, number of RSUs is 30, the vehicles' maximum speed is 25m/s (i.e., the average vehicles speed at highways) and the maximum RSUs' capacity is 10,000Mb. We assume that while sending beacons, the RSU uses the same transmit power as of vehicles. Hence, if vehicles can receive beacon from RSU, then RSU can also receive beacon from vehicles. However, RSU can use higher transmission power to broadcast some emergency messages to vehicles in order to cover larger area, which is not the scope of this work. Since we are not dealing with networking and MAC layer issues, therefore after a vehicle is informed about the amount of data it can offload to RSU or neighboring vehicle, then every time it receives a beacon from RSU or neighboring vehicle to whom it is offloading its data, it considers the received beacon as acknowledgement and thus, calculates the amount of data that could be offloaded since the reception of last beacon using its speed and data rate. Subsequently, it updates its application data size. This helps to evaluate the full performance of our data offloading scheme without having impact of external factors (such as collision). The simulation parameters are summarized in Table 3.

7.2. Performance Metrics

The performance metrics for DIVINE are as follows:

Table 3: Simulation parameters and values.

Parameter	**Value**
Area	30km long highway
Number of RSUs	10-100 (default 30)
Number of vehicles	50-200 (default 100)
Maximum vehicles' speed	15-30 m/s (default 25m/s)
Vehicles depart position	Random
Vehicles depart time	Random
Simulation time	2000 seconds
Simulation runs	20
Maximum RSUs' capacity	10000Mb
$\gamma_k^{max,med}$	75%
$\gamma_k^{max,low}$	50%
Ω_{RSU}^{arr}	10 seconds
$\omega_{v2v}^{min,c}$	10 seconds
Vehicle data	6000Mb
Data rate	6Mbps
Transmission power	10mW
Transmission range	357m
Communication technology	IEEE 802.11p
Beacon interval	1 second

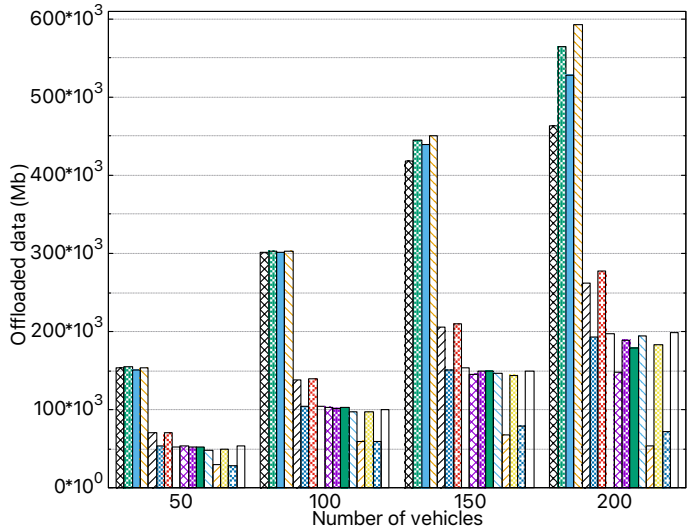
- *Offloaded data* is the amount of vehicles' total, high, medium and low priority data that has successfully been offloaded to RSUs. Higher amount of offloaded data is preferable.
- *Average offloading delay* is the average time required for all the vehicles to offload their total, high, medium and low priority data. Lower average offloading delay is preferable.
- *Maximum offloading delay* is the maximum offloading delay for all the vehicles to offload their high, medium and low priority data [26]. Lower maximum offloading delay is preferable.
- *Running time* is the running time of the network in which all the vehicles can offload their total, high, medium and low priority data [26].

7.3. Comparison Schemes

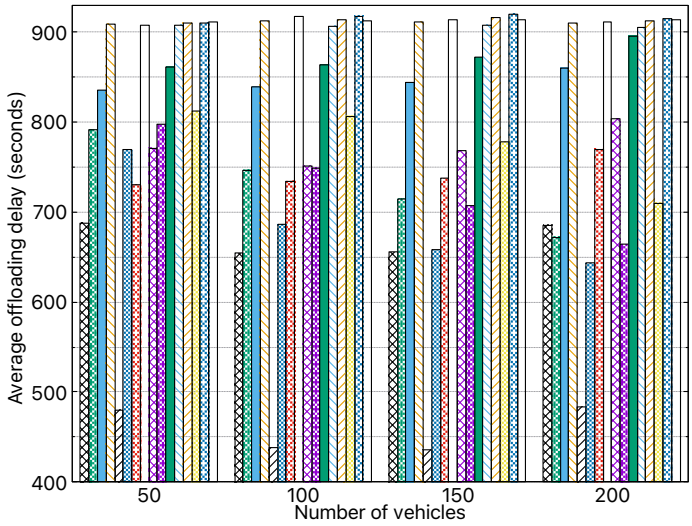
DIVINE is compared with three schemes. Firstly with DOVE, a baseline of DIVINE without QoS consideration. Secondly with V2I-Q, a V2I data offloading scheme with QoS provisioning [17]. Thirdly with V2I, a traditional V2I data offloading that has also been used for comparison in the literature [3].

7.4. Performance Evaluation

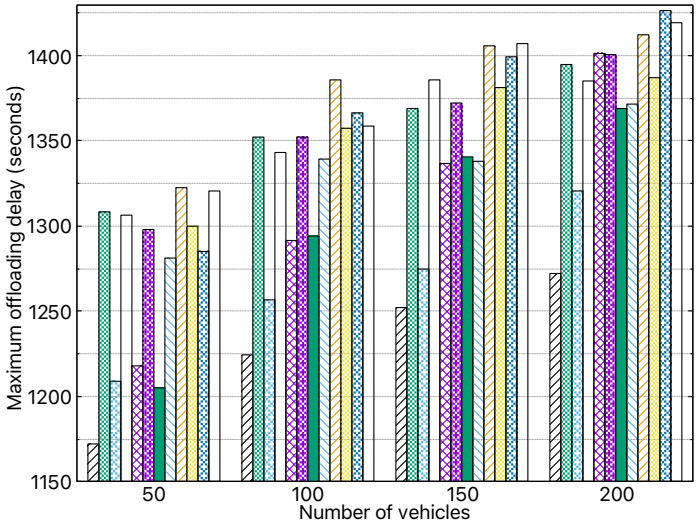
This section evaluates the performance under the effects of number of vehicles (network density) [27, 28], vehicles' maximum speed [29, 30], number of RSUs [31] and maximum RSUs' capacity.



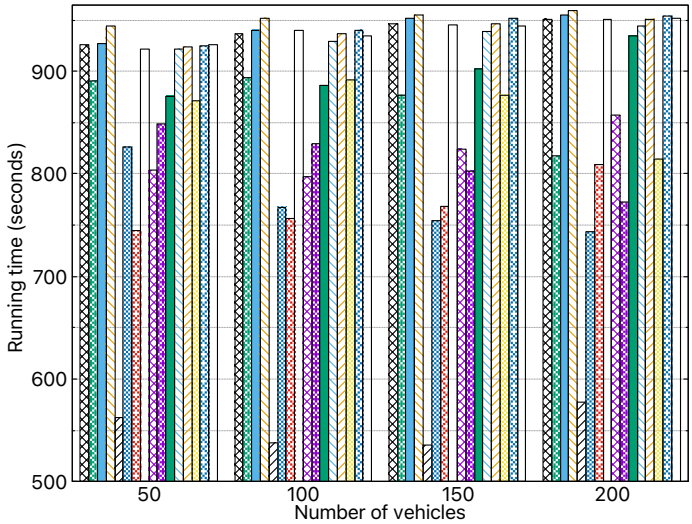
(a) Amount of offloaded data.



(b) Average offloading delay.



(c) Maximum offloading delay.



(d) Running time.

Figure 6: Effects of number of vehicles.

Table 4: Summary of results of varying number of vehicles (cf. Figure 6).

Num of vehicles	Scheme	Offloaded data (Mb)	Avg offloading delay (sec)	Max offloading delay (sec)	Running time (sec)
50	DIVINE Total	154025	688	–	926
	DOVE Total	154609	792	–	890
	V2I-Q Total	151457	835	–	927
	V2I Total	154313	909	–	944
	DIVINE High	70902	480	1172	562
	DOVE High	53557	769	1308	826
	V2I-Q High	70708	730	1209	745
	V2I High	52458	908	1306	922
	DIVINE Medium	53225	771	1218	804
	DOVE Medium	51918	797	1298	848
	V2I-Q Medium	51954	861	1205	876
	V2I Medium	47539	908	1281	922
	DIVINE Low	29898	910	1323	924
	DOVE Low	49134	812	1300	872
	V2I-Q Low	28795	910	1285	924
	V2I Low	54316	911	1321	926
100	DIVINE Total	300903	654	–	936
	DOVE Total	303029	746	–	894
	V2I-Q Total	301616	839	–	940
	V2I Total	303283	912	–	952
	DIVINE High	137772	439	1224	538
	DOVE High	104073	686	1352	768
	V2I-Q High	140288	734	1257	756
	V2I High	104416	918	1343	940
	DIVINE Medium	103224	752	1291	798
	DOVE Medium	101533	749	1352	829
	V2I-Q Medium	102575	864	1294	886
	V2I Medium	98076	906	1339	929
	DIVINE Low	59908	914	1386	936
	DOVE Low	97423	806	1357	892
	V2I-Q Low	58752	917	1367	940
	V2I Low	100791	912	1359	934
150	DIVINE Total	418318	655	–	946
	DOVE Total	444415	715	–	877
	V2I-Q Total	439392	844	–	951
	V2I Total	450977	912	–	955
	DIVINE High	205174	436	1252	535
	DOVE High	151324	658	1369	755
	V2I-Q High	210216	737	1275	768
	V2I High	154081	914	1386	945
	DIVINE Medium	145351	768	1336	824
	DOVE Medium	149040	707	1372	803
	V2I-Q Medium	149618	872	1340	902
	V2I Medium	146916	908	1338	939
	DIVINE Low	67794	916	1406	946
	DOVE Low	144050	779	1382	877
	V2I-Q Low	79557	920	1399	951
	V2I Low	149980	914	1407	944
200	DIVINE Total	463206	685	–	951
	DOVE Total	564476	672	–	817
	V2I-Q Total	527768	860	–	955
	V2I Total	592146	910	–	960
	DIVINE High	261420	484	1272	578
	DOVE High	192685	643	1395	744
	V2I-Q High	276915	770	1321	809
	V2I High	197877	911	1385	950
	DIVINE Medium	147674	804	1401	857
	DOVE Medium	189061	664	1401	772
	V2I-Q Medium	179284	895	1369	935
	V2I Medium	195093	905	1371	944
	DIVINE Low	54112	912	1412	951
	DOVE Low	182729	709	1387	814
	V2I-Q Low	71568	915	1427	954
	V2I Low	199176	913	1419	952

7.4.1. Effects of Number of Vehicles

This section evaluates the performance for varying number of vehicles from 50 to 200. Figure 6(a) presents the amount of offloaded total, high, medium and low priority data by DIVINE, DOVE, V2I-Q and V2I. The amount of offloaded data increases with the increasing number of vehicles. This is very natural because with the increasing number of vehicles, the amount of data also increases in the network. DIVINE is able to offload similar amount of total offloaded data as other schemes. However, when there are 200 vehicles,

DIVINE offloads slightly lower amount of total data than others. This is because the threshold values for the maximum allowed medium and low priority data ($\gamma_k^{max,med}$ and $\gamma_k^{max,low}$) are reached. V2I-Q is able to offload slightly more data than DIVINE when number of vehicles are 200 because due to the higher number of transmissions in DIVINE, some beacons broadcast by RSUs might be collided, causing the vehicles to miss the opportunity of V2I data offloading. DIVINE and V2I-Q are able to offload the highest and similar amount of high priority data than DOVE and V2I, thanks to QoS considerations, however at the cost of offloading lesser amount of low priority data. All schemes offloaded similar amount of medium priority data. Overall, DIVINE offloads the highest amount of high priority data similar to V2I-Q and almost equal amount of medium priority data at the cost of offloading lesser amount of lower priority data.

Figure 6(b) presents the average offloading delay of total, high, medium and low priority data. For all the varying number of vehicles, DIVINE outperforms all other schemes and achieves much lower average offloading delay, fulfilling our objective of offloading the data as soon as possible to RSUs. DIVINE achieves the lowest offloading delay for total and high priority, thanks to QoS provisioning and V2V data offloading. However, V2I-Q achieves slightly higher offloading delay for total and high priority data. This is because of not taking advantage of V2V data offloading. For DOVE, the average offloading delay of total and high priority data is even lower than V2I-Q, it shows the significance of V2V data offloading, i.e., even without QoS provisioning, V2V data offloading can significantly help to reduce the offloading delay. V2I performs the worst for all types of data having almost similar average offloading delay. This is because it neither considers QoS provisioning nor V2V data offloading. For medium priority data, DIVINE performs almost similarly to DOVE. V2I-Q performs worse than DIVINE and DOVE but better than V2I. For low priority data, DOVE achieves lower average offloading delay because of no QoS provisioning (i.e., traffic classification, overload control and admission control). DIVINE and V2I-Q perform similar to V2I because they only offload low priority data once high and medium priority data are offloaded. Overall, DIVINE achieves the lowest offloading delay for total, high and medium priority data but gets higher offloading delay for low priority data equivalent to V2I-Q and V2I.

Figure 6(c) presents the maximum offloading delay for high, medium and low priority data. DIVINE achieves the lowest maximum offloading delay for high priority data. The maximum offloading delay of all types of data for DIVINE is almost similar to V2I-Q. This is because the maximum offloading delay corresponds to the cases of vehicles when V2V data offloading in DIVINE is either not possible or not preferred. Hence, such vehicles offload their data using V2I data offloading, exhibiting the similar maximum offloading delay as V2I-Q. DOVE and V2I cause higher maximum offloading delay for all high, medium and low priority data than DIVINE and V2I-Q because of no QoS provisioning. Overall, DIVINE achieves the lowest maximum offloading delay for high priority data and for medium priority data, its maximum offloading delay is similar to V2I-Q and better than DOVE and V2I.

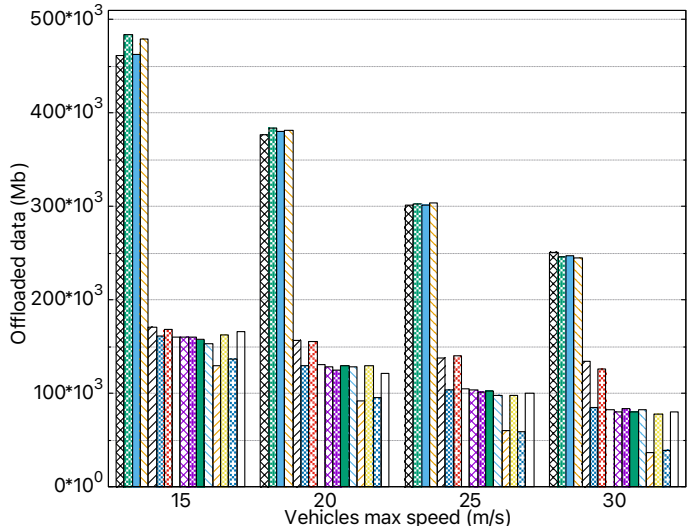
Figure 6(d) presents the running time for total, high, medium and low priority data. The total running of all the schemes is similar which is very natural because in order to offload all the data, each scheme requires similar amount of time. DIVINE achieves the lowest running time for the high priority data due to QoS provisioning and V2V data offloading at the expense of higher running time for low priority data (as discussed in Figure 6(a)). For medium priority data, it achieves similar running time as DOVE. DOVE has similar running time for high, medium and low priority data due to the lack of QoS provisioning, however, because of V2V data offloading, its running time is lower than V2I that exhibits the highest running time for high, medium and low priority data. Finally, V2I-Q achieves similar running time for high priority data as DOVE and higher running time for medium priority data as compared to DIVINE and DOVE because of lack of V2V data offloading. For low priority data, its running time is similar to DIVINE because of the same reason (i.e., offloading high and medium priority data first). Overall, as usual, DIVINE performs the best for high and medium priority data, and for low priority data, it performs similar to V2I-Q and V2I but not better than DOVE because DOVE achieves lesser running time for low priority data at the expense of higher running time for high and medium priority data.

The results for varying number of vehicles are also presented in Table 4 for better readability.

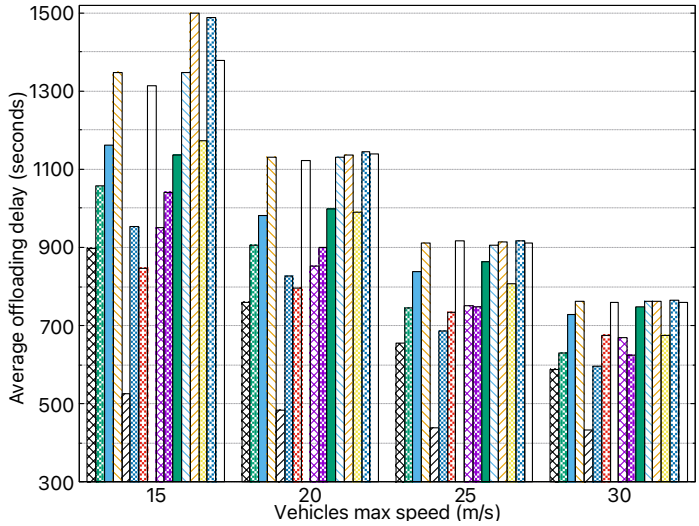
7.4.2. Effects of Vehicles' Maximum Speed

This section evaluates the performance for varying vehicles' maximum speed from 15m/s to 30m/s. Figure 7(a) presents the amount of offloaded total, high, medium and low priority data by DIVINE, DOVE, V2I-Q and V2I. The amount of offloaded data reduces with the increasing vehicles' speed because with the increasing vehicles' speed, the connectivity time of vehicles with RSUs reduces. All schemes offload similar amount of total offloaded data. When the speed is 15m/s, all the schemes offload almost similar amount of high, medium and low priority data. This is because with lower speed, all vehicles are able to offload all the data they contain, however there will be difference in offloading delay which we will see next. As the speed keeps increasing, DIVINE and V2I-Q offload more high priority data at the cost of offloading lesser amount of low priority data because of QoS provisioning, as compared to DOVE and V2I. DOVE and V2I offload similar amount of high, medium and low priority data because of lack of QoS provisioning (i.e., no differentiation between types of data).

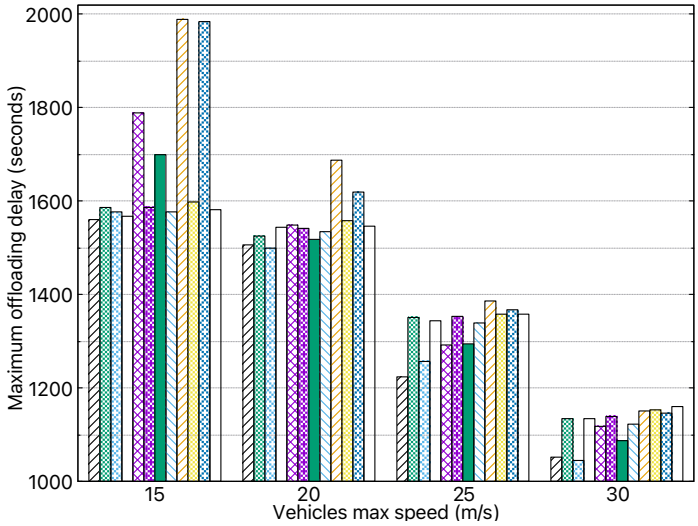
Figure 7(b) presents the average offloading delay of total, high, medium and low priority data. For all the varying speed, the average offloading delay reduces because with the increasing speed, vehicles meet RSUs sooner (although for shorter duration, causing lesser amount of offloaded data, as seen in Figure 7(a)) and hence, they can offload their data to RSUs earlier. DIVINE outperforms all other schemes and achieves much lower average offloading delay. DIVINE has the lowest offloading delay for total and high priority data, thanks to QoS provisioning and V2V data offloading, however at the cost of higher offloading delay for low priority data. V2I-Q achieves slightly higher offloading delay



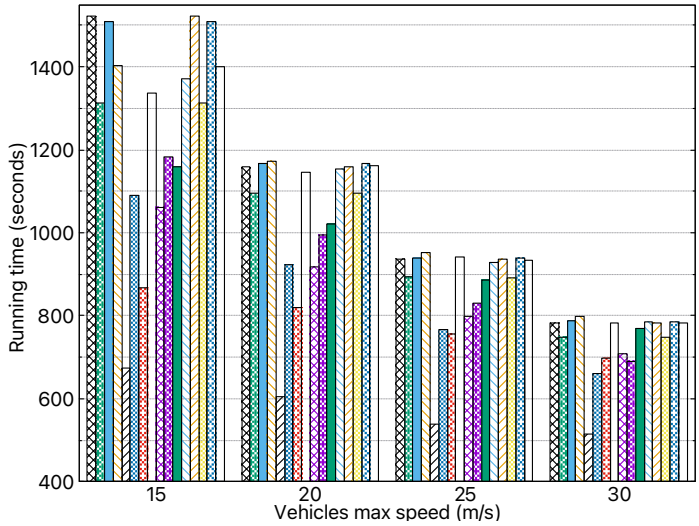
(a) Amount of offloaded data.



(b) Average offloading delay.



(c) Maximum offloading delay.



(d) Running time.

Figure 7: Effects of vehicles' maximum speed.

Table 5: Summary of results of varying vehicles' speed (cf. Figure 7).

Vehicles speed	Scheme	Offloaded data (Mb)	Avg offloading delay (sec)	Max offloading delay (sec)	Running time (sec)
15	DIVINE Total	460734	898	–	1522
	DOVE Total	483758	1056	–	1312
	V2I-Q Total	462138	1161	–	1510
	V2I Total	479165	1347	–	1402
	DIVINE High	170702	525	1560	673
	DOVE High	161001	952	1585	1090
	V2I-Q High	167936	846	1576	869
	V2I High	159971	1314	1567	1336
	DIVINE Medium	160434	951	1788	1062
	DOVE Medium	159766	1040	1587	1182
	V2I-Q Medium	157429	1137	1699	1159
	V2I Medium	153252	1348	1578	1370
	DIVINE Low	129598	1499	1989	1522
	DOVE Low	162991	1173	1599	1312
	V2I-Q Low	136774	1487	1983	1510
	V2I Low	165942	1378	1581	1401
20	DIVINE Total	376744	759	–	1159
	DOVE Total	383571	906	–	1095
	V2I-Q Total	380024	982	–	1166
	V2I Total	380936	1131	–	1173
	DIVINE High	157083	484	1506	604
	DOVE High	129697	828	1525	924
	V2I-Q High	155173	797	1500	819
	V2I High	130786	1123	1543	1145
	DIVINE Medium	128215	853	1548	919
	DOVE Medium	124709	899	1541	996
	V2I-Q Medium	129576	999	1517	1021
	V2I Medium	128280	1131	1535	1154
	DIVINE Low	91446	1136	1687	1159
	DOVE Low	129165	991	1558	1094
	V2I-Q Low	95275	1144	1620	1166
	V2I Low	121870	1139	1546	1161
25	DIVINE Total	300903	654	–	936
	DOVE Total	303029	746	–	894
	V2I-Q Total	301616	839	–	940
	V2I Total	303283	912	–	952
	DIVINE High	137772	439	1224	538
	DOVE High	104073	686	1352	768
	V2I-Q High	140288	734	1257	756
	V2I High	104416	918	1343	940
	DIVINE Medium	103224	752	1291	798
	DOVE Medium	101533	749	1352	829
	V2I-Q Medium	102575	864	1294	886
	V2I Medium	98076	906	1339	929
	DIVINE Low	59908	914	1386	936
	DOVE Low	97423	806	1357	892
	V2I-Q Low	58752	917	1367	940
	V2I Low	100791	912	1359	934
30	DIVINE Total	250733	588	–	783
	DOVE Total	246398	631	–	748
	V2I-Q Total	246862	729	–	788
	V2I Total	244990	761	–	798
	DIVINE High	134156	434	1052	514
	DOVE High	84817	595	1135	661
	V2I-Q High	126595	675	1045	698
	V2I High	82205	759	1135	782
	DIVINE Medium	80234	669	1118	708
	DOVE Medium	83367	624	1140	690
	V2I-Q Medium	80646	747	1088	769
	V2I Medium	82139	763	1122	786
	DIVINE Low	36343	761	1150	783
	DOVE Low	78215	675	1153	747
	V2I-Q Low	39621	764	1147	786
	V2I Low	80645	759	1160	782

for total and high priority data because of not taking the advantage of V2V data offloading. For DOVE, the average offloading delay of total and high priority data is even lower than V2I-Q which shows the significance of V2V data offloading. V2I performs the worst for all types of data (i.e., total, high, medium and low priority) and achieving similar average offloading delay for each varying speed. This is because of neither considering the QoS provisioning nor V2V data offloading. For medium priority data, DIVINE performs almost similar to DOVE, while V2I-Q performs worse than DIVINE and DOVE but better than V2I. For low priority data, DOVE achieves lower average offloading

delay because of no QoS provisioning. DIVINE and V2I-Q perform similar to V2I because they only offload low priority data once high and medium priority data are offloaded.

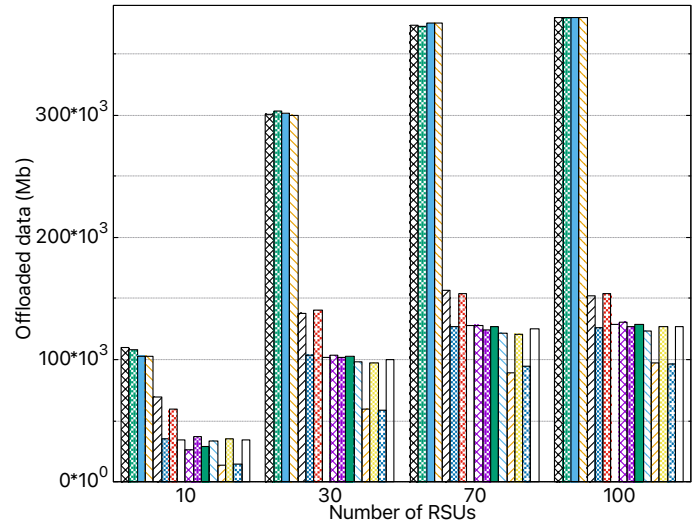
Figure 7(c) presents the maximum offloading delay for high, medium and low priority data. With the increasing speed, the maximum offloading delay reduces because of the same reason as explained for Figure 7(b). When the vehicles' maximum speed is 15m/s and 20m/s, DIVINE and V2I-Q have higher maximum offloading delay for low priority data. This is the cost of offloading high priority data faster (as seen in Figure 7(b)). When the vehicle's maximum speed exceeds 25m/s, DIVINE and V2I-Q have lower maximum offloading delay for high priority data due to QoS provisioning. For medium and low priority data, all the schemes exhibit similar maximum offloading delay. To recall, the maximum offloading delay for DIVINE and V2I-Q is similar because the maximum offloading delay is the case when V2V data offloading is not possible/preferred for some vehicles, hence only V2I data offloading is used for such vehicles.

Figure 7(d) presents the running time for total, high, medium and low priority data. For the varying vehicles' maximum speed, the total running of all the schemes is almost similar (except 15m/s speed) because in order to offload all the data, each scheme requires similar amount of time. When the vehicles' maximum speed is 15m/s, DIVINE and V2I-Q take slightly longer time because as we have seen in Figure 7(c), DIVINE and V2I-Q require more time for offloading low priority data that increases the total running time. DIVINE achieves the lowest running time for the high and medium priority data due to QoS provisioning and V2V data offloading at the expense of higher running time for low priority data. DOVE has similar running time for high, medium and low priority data due to a lack of QoS provisioning, however, because of V2V data offloading, its running time is lower than V2I that exhibits the highest running time for high, medium and low priority data. V2I-Q achieves similar running time for high and medium priority data as DOVE. For low priority data, its running time is similar to DIVINE because of offloading high and medium priority data first. Overall, as expected, DIVINE performs the best for high and medium priority data, and for low priority data, it performs similar to V2I-Q and V2I but not better than DOVE because DOVE achieves lesser running time for low priority data at the expense of higher running time for high and medium priority data.

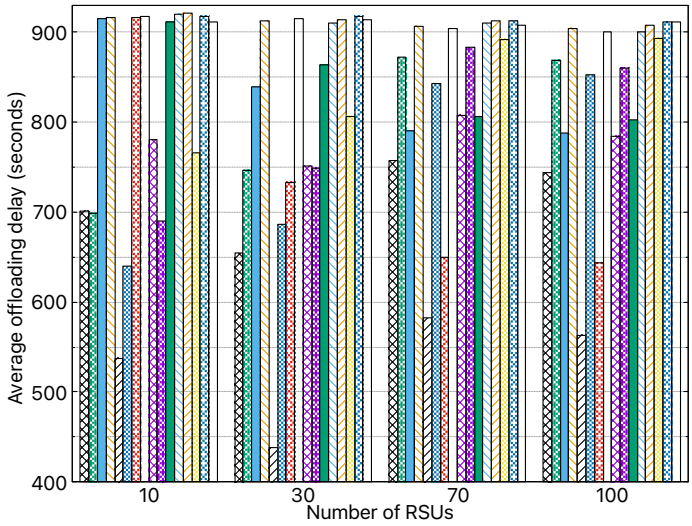
The results for varying vehicles' speed are also presented in Table 5 for better readability.

7.4.3. Effects of Number of RSUs

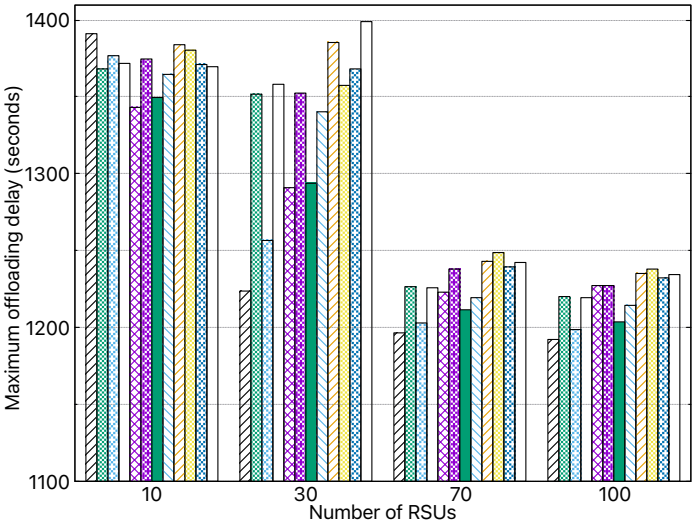
This section evaluates the performance for varying number of RSUs from 10 to 100. Figure 8(a) presents the amount of offloaded total, high, medium and low priority data by DIVINE, DOVE, V2I-Q and V2I. The amount of offloaded data increases with the increasing number of RSUs. This is because with the increasing number of RSUs, vehicles get more opportunity to offload their data. When there are 10 RSUs, all schemes offload very lesser amount of data because there are not enough RSUs to receive the data of vehicles. When



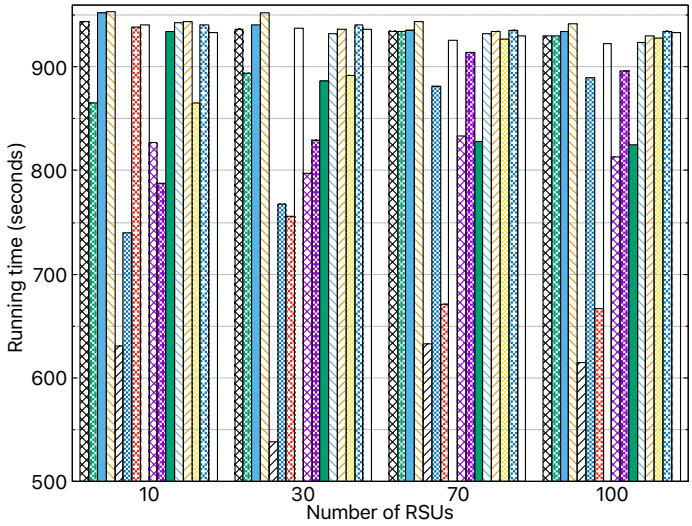
(a) Amount of offloaded data.



(b) Average offloading delay.



(c) Maximum offloading delay.



(d) Running time.

Figure 8: Effects of number of RSUs.

Table 6: Summary of results of varying number of RSUs (cf. Figure 8).

Num of RSUs	Scheme	Offloaded data (Mb)	Avg offloading delay (sec)	Max offloading delay (sec)	Running time (sec)
10	DIVINE Total	109800	701	–	944
	DOVE Total	107704	699	–	865
	V2I-Q Total	102675	915	–	952
	V2I Total	102800	916	–	953
	DIVINE High	69686	537	1391	631
	DOVE High	35445	639	1368	740
	V2I-Q High	59157	915	1377	938
	V2I High	34660	918	1372	941
	DIVINE Medium	26624	780	1344	827
	DOVE Medium	36759	689	1375	787
	V2I-Q Medium	29203	912	1350	934
	V2I Medium	33872	920	1365	942
	DIVINE Low	13490	921	1384	944
	DOVE Low	35500	766	1380	865
	V2I-Q Low	14315	917	1371	940
	V2I Low	34269	911	1370	933
30	DIVINE Total	300903	654	–	936
	DOVE Total	303029	746	–	894
	V2I-Q Total	301258	839	–	940
	V2I Total	299998	913	–	952
	DIVINE High	137772	439	1224	538
	DOVE High	104073	686	1352	768
	V2I-Q High	140258	733	1257	756
	V2I High	101774	915	1358	937
	DIVINE Medium	103224	752	1291	798
	DOVE Medium	101533	749	1352	829
	V2I-Q Medium	102415	864	1294	886
	V2I Medium	97856	909	1340	932
	DIVINE Low	59908	914	1386	936
	DOVE Low	97423	806	1357	892
	V2I-Q Low	58584	917	1368	940
	V2I Low	100368	913	1399	936
70	DIVINE Total	373298	757	–	934
	DOVE Total	372297	871	–	934
	V2I-Q Total	375074	791	–	935
	V2I Total	375211	907	–	944
	DIVINE High	156493	582	1197	633
	DOVE High	127058	843	1227	882
	V2I-Q High	153548	649	1203	671
	V2I High	128316	903	1226	926
	DIVINE Medium	127936	807	1223	833
	DOVE Medium	124083	883	1238	914
	V2I-Q Medium	127084	806	1211	828
	V2I Medium	121637	909	1220	932
	DIVINE Low	88870	912	1243	934
	DOVE Low	121156	892	1249	927
	V2I-Q Low	94443	913	1240	935
	V2I Low	125259	907	1243	929
100	DIVINE Total	380045	744	–	929
	DOVE Total	379462	868	–	929
	V2I-Q Total	379302	787	–	934
	V2I Total	379402	904	–	942
	DIVINE High	152049	563	1192	615
	DOVE High	126060	852	1220	889
	V2I-Q High	154237	644	1199	666
	V2I High	129075	900	1220	922
	DIVINE Medium	130831	785	1227	813
	DOVE Medium	126608	860	1227	896
	V2I-Q Medium	128511	802	1204	824
	V2I Medium	123704	901	1215	923
	DIVINE Low	97166	907	1235	929
	DOVE Low	126794	893	1238	928
	V2I-Q Low	96554	912	1233	934
	V2I Low	126624	911	1234	933

the number of RSUs exceed 70, the amount of all types of offloaded data for all the schemes do not change because at this point, there are sufficient RSUs to receive offloaded data from all the vehicles and there is no benefit of adding more RSUs in our considered scenario. DIVINE and V2I-Q offload the highest and similar amount of high priority data as compared to DOVE and V2I, thanks to QoS considerations, however as expected, at the cost of offloading lesser amount of low priority data. Their offloaded amount of medium priority data is similar to DOVE and V2I. Due to the lack of QoS consideration, DOVE and V2I offload similar amount of high, medium and low priority data.

Figure 8(b) presents the average offloading delay of total, high, medium and low priority data. For all the varying number of RSUs, DIVINE outperforms all other schemes and achieves much lower average offloading delay. Similar to Figure 8(a), when the number of RSUs exceed 70, the average offloading delay for all the schemes do not change because at this point, all vehicles can offload their data and hence, there is no effect of adding more RSUs. DIVINE has the lowest offloading delay for total and high priority data, however at the cost of higher offloading delay for low priority data. V2I-Q achieves slightly higher offloading delay for total and high priority data because of not taking the advantage of V2V data offloading. DOVE achieves even lower average offloading delay of total and high priority data as compared to V2I-Q due to the exploitation of V2V data offloading. As expected, V2I performs the worst for all total, high, medium and low priority data and it achieves similar average offloading delay because of neither considering the QoS provisioning nor V2V data offloading. For medium priority data, DIVINE performs even better than DOVE, while V2I-Q performs worse than DIVINE and DOVE and similar to V2I. For low priority data, only DOVE achieves slightly lower average offloading delay when number of RSUs are less than 70. However, beyond this point, all the schemes have similar average offloading delay of low priority data.

Figure 8(c) presents the maximum offloading delay for high, medium and low priority data. With the increasing number of RSUs, the maximum offloading delay reduces because vehicles find RSUs earlier and can offload their data faster. When there are 10 RSUs, DIVINE has the highest maximum offloading delay for high priority data. Since there are very few RSUs, such maximum offloading delay is recorded when there is higher delay for some vehicles caused by V2V data offloading for offloading high priority data in DIVINE. When the number of RSUs increases, DIVINE achieves the least maximum offloading delay for high priority data. When the number of RSUs reach 70, beyond this point, the maximum offloading delay for all schemes stays same. DIVINE and V2I-Q achieve the lowest maximum offloading delay for high priority data due to QoS provisioning. For medium priority data, DOVE achieves the lowest maximum offloading delay, while for low priority data, all the schemes exhibit similar maximum offloading delay. Even though DIVINE has slightly higher maximum offloading delay for medium priority data, however as seen in Figure 8(b), it has the least average offloading delay for medium priority data.

Figure 8(d) presents the running time for total, high, medium and low priority data. For the varying number of RSUs, the total running of all the schemes is almost similar because each scheme requires similar amount of time to offload all their data. DIVINE achieves the lowest running time for the high priority data due to QoS provisioning and V2V data offloading. DOVE has similar running time for high, medium and low priority data due to lack of QoS provisioning, however, its running time is lower than V2I due to the exploitation of V2V data offloading. V2I that exhibits the highest running time for high, medium and low priority data. V2I-Q achieves slightly higher running time for high and medium priority data as DIVINE. Overall, as usual, DIVINE performs the best for high and medium priority data, and for low priority data, it performs similar

Table 7: Summary of results of varying RSUs' capacity (cf. Figure 9).

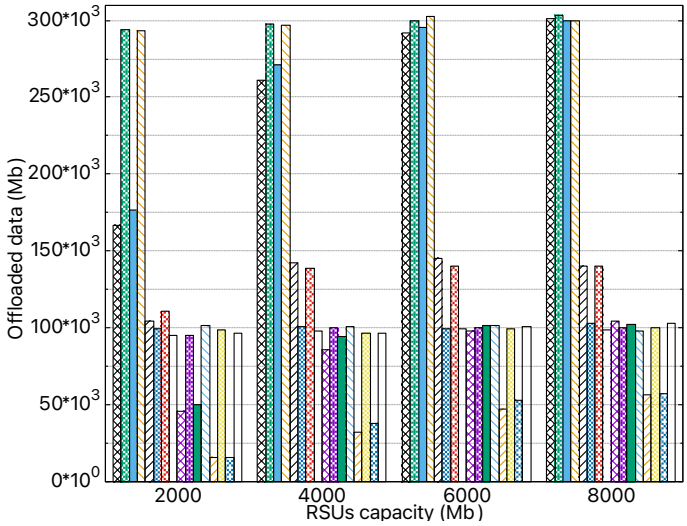
RSU capacity	Scheme	Offloaded data (Mb)	Avg offloading delay (sec)	Max offloading delay (sec)	Running time (sec)
2000	DIVINE Total	167018	701	–	937
	DOVE Total	293837	691	–	807
	V2I-Q Total	176469	906	–	949
	V2I Total	293199	913	–	953
	DIVINE High	104681	513	1356	600
	DOVE High	99874	669	1352	758
	V2I-Q High	110668	888	1375	910
	V2I High	95271	914	1337	936
	DIVINE Medium	46216	807	1372	843
	DOVE Medium	95066	691	1358	780
	V2I-Q Medium	50157	912	1350	934
	V2I Medium	101435	915	1338	938
	DIVINE Low	16122	914	1390	937
	DOVE Low	98897	712	1348	797
	V2I-Q Low	15644	917	1371	940
	V2I Low	96492	909	1345	932
4000	DIVINE Total	260952	677	–	937
	DOVE Total	297719	721	–	856
	V2I-Q Total	271035	854	–	942
	V2I Total	296522	910	–	953
	DIVINE High	142463	468	1236	564
	DOVE High	100653	693	1355	771
	V2I-Q High	138526	753	1257	775
	V2I High	98443	913	1363	935
	DIVINE Medium	86241	795	1351	839
	DOVE Medium	100056	708	1338	794
	V2I-Q Medium	94470	890	1322	913
	V2I Medium	101291	907	1343	930
	DIVINE Low	32248	915	1380	937
	DOVE Low	97011	764	1391	851
	V2I-Q Low	38039	917	1371	940
	V2I Low	96788	910	1348	932
6000	DIVINE Total	291665	664	–	939
	DOVE Total	299821	743	–	888
	V2I-Q Total	295324	840	–	940
	V2I Total	302205	912	–	948
	DIVINE High	145594	454	1248	556
	DOVE High	99657	694	1355	775
	V2I-Q High	140179	734	1257	757
	V2I High	99602	909	1340	931
	DIVINE Medium	98421	767	1299	816
	DOVE Medium	100326	731	1337	810
	V2I-Q Medium	102007	867	1295	890
	V2I Medium	101442	914	1357	936
	DIVINE Low	47650	917	1365	939
	DOVE Low	99838	809	1379	888
	V2I-Q Low	53137	917	1371	940
	V2I Low	101161	913	1357	935
8000	DIVINE Total	301100	659	–	937
	DOVE Total	303437	752	–	885
	V2I-Q Total	299994	839	–	940
	V2I Total	299836	912	–	951
	DIVINE High	140312	446	1252	545
	DOVE High	103019	707	1353	783
	V2I-Q High	140142	734	1257	756
	V2I High	98709	916	1359	939
	DIVINE Medium	104283	754	1286	802
	DOVE Medium	99944	740	1349	817
	V2I-Q Medium	102230	864	1294	887
	V2I Medium	98139	912	1339	935
	DIVINE Low	56505	914	1376	937
	DOVE Low	100474	809	1388	885
	V2I-Q Low	57622	917	1371	940
	V2I Low	102987	908	1368	931

to V2I-Q and V2I but not better than DOVE because DOVE achieves lesser running time for low priority data at the expense of higher running time for high and medium priority data.

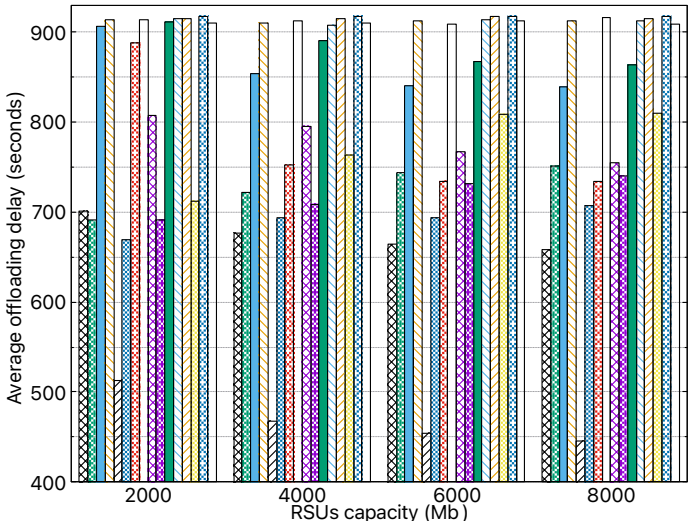
The results for varying number of RSUs are also presented in Table 6 for better readability.

7.4.4. Effects of RSUs Capacity

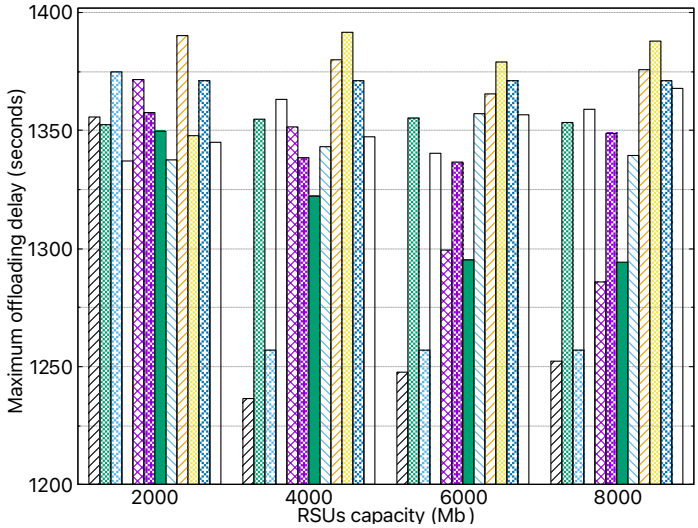
This section evaluates the performance for varying RSUs' capacity from 2000Mb to 8000Mb. Figure 9(a) presents the amount of offloaded total, high,



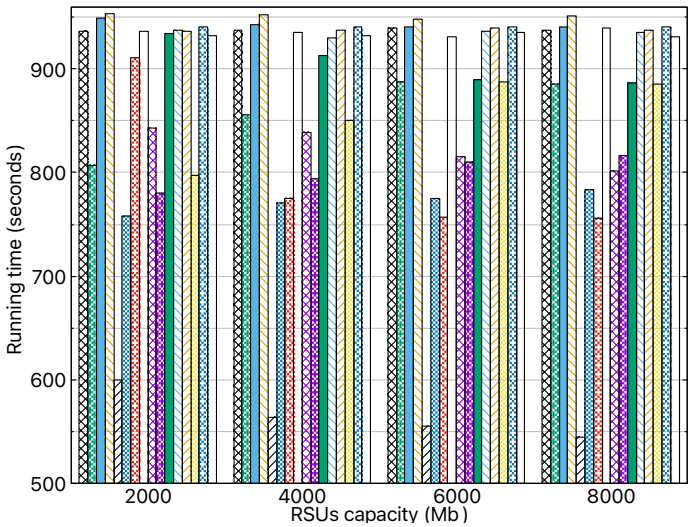
(a) Amount of offloaded data.



(b) Average offloading delay.



(c) Maximum offloading delay.



(d) Running time.

Figure 9: Effects of RSUs' capacity.

medium and low priority data by DIVINE, DOVE, V2I-Q and V2I. When the RSUs' capacity is 2000Mb, DIVINE and V2I-Q offload lesser amount of total data because the capacity of RSUs is insufficient and the threshold values of maximum allowed medium and low priority data ($\gamma_k^{max,med}$ and $\gamma_k^{max,low}$) restricted them to offload higher amount of medium and low priority data. Within such RSUs' capacity, the maximum amount of high priority data that DIVINE and V2I-Q can offload is similar to DOVE and V2I. However, as the capacity of RSUs keeps increasing, DIVINE and V2I-Q take its advantage and start to offload more high priority data, however at the cost of offloading lesser amount of low priority data. When the RSUs' capacity exceeds 6000Mb, there is no much difference in the amount of offloaded data by DIVINE and V2I-Q because this is the optimal RSUs' capacity to offload all the data of vehicles in the network without getting overloaded in our current scenario. As usual, DOVE and V2I offload similar amount of high, medium and low priority data because of lack of QoS provisioning.

Figure 9(b) presents the average offloading delay of total, high, medium and low priority data. For all the varying RSUs' capacity, DIVINE outperforms all other schemes and achieves much lower average offloading delay for total and high priority data. When the RSUs' capacity exceeds 6000Mb, there is very negligible effect because as seen in Figure 8(a), 6000Mb is the optimal RSUs' capacity to successfully offload all the data without getting overloaded in our scenario. As expected, V2I-Q achieves slightly higher offloading delay for total and high priority data because of not taking the advantage of V2V data offloading, while DOVE achieves lower average offloading delay of total and high priority data as compared to V2I-Q due to the exploitation of V2V data offloading. V2I performs the worst as always for all total, high, medium and low priority data resulting in similar average offloading delay due to the lack of QoS provisioning and V2V data offloading. For medium priority data, DOVE performs the best, while DIVINE performs the second best. V2I-Q performs worse than DIVINE and DOVE and similar to V2I. For low priority data, only DOVE achieves slightly lower average offloading delay.

Figure 9(c) presents the maximum offloading delay for high, medium and low priority data. For all the varying RSUs' capacity, DIVINE outperforms all other schemes and achieves much lower maximum offloading delay for high and medium priority data, except when the capacity of RSUs is 2000Mb because this is the case of insufficient RSUs' capacity in which some vehicles offload their data only using V2I data offloading. When the RSUs' capacity exceeds 6000Mb, there is very negligible effect because as seen in Figs. 8(a) and 8(b), 6000Mb is the optimal RSUs' capacity to successfully offload all the data without getting overloaded in our scenario. Hence, the maximum offloading delay is similar. V2I-Q achieves slightly higher offloading delay for high and medium priority data than DIVINE åbecause of not taking the advantage of V2V data offloading, while DOVE and V2I have similar maximum offloading delay for high, medium and low priority data. For low priority data, all schemes exhibit almost similar maximum offloading delay.

Figure 9(d) presents the running time for total, high, medium and low pri-

ority data. For the varying RSUs' capacity, the total running of all the schemes, except DOVE, is almost similar. Since DOVE does not differentiate between the data, therefore, using V2V data offloading, it offloads all its data without considering QoS provisioning and gets lower total running time. DIVINE achieves the lowest running time for the high priority data due to QoS provisioning and V2V data offloading. DOVE has similar running time for high, medium and low priority data due to lack of QoS provisioning. V2I results in the highest running time for high, medium and low priority data. V2I-Q achieves higher running time for high priority data than DIVINE and higher running time for medium priority data than DIVINE and DOVE. All schemes, except DOVE, have the similar running time for low priority data. Overall, DIVINE performs the best for high and medium priority data.

The results for varying RSUs' capacity are also presented in Table 7 for better readability.

7.4.5. Summary and Insights

To summarize, DIVINE overall performs the best by offloading the highest amount of high priority data with the least offloading delay and the least running time, however, at the cost of lower performance for low priority data (i.e., lesser amount of offloaded data, higher offloading delay and higher running time). DOVE offloads similar amount of data with similar offloading delay and similar running time for all types of data, however it has the higher maximum offloading delay. V2I-Q performs similar to DIVINE for the amount of offloaded data, however it incurs higher offloading delay and higher running time. V2I offloads lower amount of high priority data, but higher amount of medium and low priority data (similar to DOVE), but it has the highest average and maximum offloading delays, and the highest running time. In our current scenario, 60 RSUs, each with a capacity of 6000Mb is the optimal setting to achieve the best performance in terms of higher amount of offloaded data, lower average offloading delay, lower maximum offloading delay and lower running time. Moreover, although we target a highway scenario, however our evaluation also covers a rural scenario. Because the rural scenario has lower number of RSUs deployed (evaluated in Section 7.4.3), lower number of vehicles (evaluated in Section 7.4.1) and higher vehicles' speed (evaluated in Section 7.4.2). In future, we plan to consider more specific urban and rural scenarios into consideration.

8. Conclusion and Future Work

In this paper, we proposed DIVINE, an efficient data offloading scheme for vehicular networks with QoS provisioning. We modeled the connectivity of vehicles with RSU, with other vehicles heading on the same or opposite direction, the offloading capacity, and the expected time to reach the next RSU. We provisioned the QoS using three QoS functions: traffic classification, overload control and admission control. We have provided detailed data offloading procedure and algorithms of DIVINE, i.e., how a vehicle selects a node (RSU/vehicle) for

requesting data offloading, how the data offloading request is processed at RSU and vehicles, and finally, how a requested vehicle processed data offloading reply to start data offloading. In order to better understand DIVINE, we presented illustrative examples. The simulation results confirmed that DIVINE outperforms other schemes by offloading more high priority data with lower average offloading delay, the maximum offloading delay and the running time.

In future, we plan to extend this work in several dimensions. We plan to consider urban and rural scenarios that will consider more crossing roads and heterogeneous traffic. In such scenarios, estimated contact time between vehicles and time to reach an RSU will have to be revisited. We will investigate how to avoid offloading redundant data (e.g., same accident data captured by multiple vehicles) by enabling coordination and collaboration among vehicles. We also plan to consider overlapping in the coverage of RSUs in which a vehicle does not necessarily need to establish connection with each RSU separately. We will consider to change the priority of data with time and consider an expiration timer of data. We plan to extend V2V data offloading in two ways. Firstly, by enabling a vehicle to receive data from multiple vehicles at a time. Secondly, if a vehicle is performing V2I data offloading, then enabling a vehicle to change its granted low and medium priority data with high priority data, and its granted low priority data with medium priority data of requested vehicle. Finally, we plan to tune the threshold values of maximum allowed medium and low priority data.

Acknowledgment

This work was partially supported by CPER DATA and ELSAT projects.

References

[1] H. Zhou, H. Wang, X. Chen, X. Li, S. Xu, Data Offloading Techniques Through Vehicular Ad Hoc Networks: A Survey, IEEE Access 6 (2018) 65250–65259. doi:10.1109/ACCESS.2018.2878552.

[2] H. Lin, S. Zeadally, Z. Chen, H. Labiod, L. Wang, A Survey on Computation Offloading Modeling for Edge Computing, Journal of Network and Computer Applications 169 (2020) 102781. doi:10.1016/j.scitotenv.2020.139795.

[3] C.-M. Huang, S.-Y. Lin, Z.-Y. Wu, The k-hop-limited V2V2I VANET Data Offloading using the Mobile Edge Computing (MEC) Mechanism, Vehicular Communications 26 (2020) 100268. doi:10.1016/j.vehcom.2020.100268.

[4] S. Ancona, R. Stanica, M. Fiore, Performance Boundaries of Massive Floating Car Data Offloading, IEEE/IFIP Proceedings on 11th Annual Conference on Wireless On-Demand Network Systems and Services, (WONS) (2014) 89–96doi:10.1109/WONS.2014.6814727.

[5] X. Zhu, Y. Li, D. Jin, J. Lu, Contact-Aware Optimal Resource Allocation for Mobile Data Offloading in Opportunistic Vehicular Networks, IEEE Transactions on Vehicular Technology 66 (8) (2017) 7384–7399. doi:10.1109/TVT.2017.2668396.

[6] M. Lee, J. Song, J. P. Jeong, T. T. Kwon, DOVE: Data Offloading through Spatio-temporal Rendezvous in Vehicular Networks, 24th International Conference on Computer Communications and Networks, (ICCCN) (2015) 1–8doi:10.1109/ICCCN.2015.7288400.

[7] Y. Sun, L. Xu, Y. Tang, W. Zhuang, Traffic Offloading for Online Video Service in Vehicular Networks: A Cooperative Approach, IEEE Transactions on Vehicular Technology 67 (8) (2018) 7630–7642. doi:10.1109/TVT.2018.2837024.

[8] J. Feng, Z. Feng, A Vehicle-Assisted Offloading Scheme for Hotspot Base Stations on Metropolitan Streets, IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC) (2018) 1–6doi:10.1109/PIMRC.2017.8292366.

[9] C. Song, J. Wu, W. S. Yang, M. Liu, I. Jawhar, N. Mohamed, Exploiting Opportunities in V2V Transmissions with RSU-assisted Backward Delivery, IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) (2017) 271–276doi:10.1109/INFCOMW.2017.8116388.

[10] S. Guntuka, E. M. Shakshuki, A. Yasar, H. Gharrad, Vehicular Data Offloading by Road-Side Units Using Intelligent Software Defined Network, Procedia Computer Science 177 (2020) 151–161. doi:10.1016/j.procs.2020.10.023.

[11] Y. Wu, J. Zheng, Modeling and Analysis of the Uplink Local Delay in MEC-Based VANETs, IEEE Transactions on Vehicular Technology 69 (4) (2020) 3538–3549. doi:10.1109/TVT.2020.2984835.

[12] Y. Wu, J. Zheng, Modeling and Analysis of the Downlink Local Delay in MEC-Based VANETs, IEEE Transactions on Vehicular Technology 69 (6) (2020) 6619–6630. doi:10.1109/TVT.2020.2984835.

[13] Y. Wang, J. Zheng, N. Mitton, Delivery Delay Analysis for Roadside Unit Deployment in Vehicular Ad Hoc Networks with Intermittent Connectivity, IEEE Transactions on Vehicular Technology 65 (10) (2016) 8591–8602. doi:10.1109/TVT.2015.2506599.

[14] Y. Wang, J. Zheng, Connectivity analysis of a highway with one entry/exit and multiple roadside units, IEEE Transactions on Vehicular Technology 67 (12) (2018) 11705–11718. doi:10.1109/TVT.2018.2873706.

[15] G. Li, L. Boukhatem, J. Wu, Adaptive Quality-of-Service-Based Routing for Vehicular Ad Hoc Networks with Ant Colony Optimization,

IEEE Transactions on Vehicular Technology 66 (4) (2017) 3249–3264. doi:10.1109/TVT.2016.2586382.

[16] A. H. Sodhro, M. S. Obaidat, Q. H. Abbasi, P. Pace, S. Pirbhulal, A.-u.-h. Yasar, G. Fortino, M. A. Imran, M. Qarage, Quality of Service Optimization in an IoT-Driven Intelligent Transportation System, IEEE Wireless Communications 26 (2) (2019) 10–17.

[17] Y. Saleem, N. Mitton, V. Loscri, A Vehicle-to-Infrastructure Data Offloading Scheme for Vehicular Networks with QoS Provisioning, International Wireless Communications & Mobile Computing Conference (IWCMC). URL https://hal.inria.fr/hal-03188360/document

[18] F. Lyu, H. Zhu, N. Cheng, H. Zhou, W. Xu, M. Li, X. Shen, Characterizing Urban Vehicle-to-Vehicle Communications for Reliable Safety Applications, IEEE Transactions on Intelligent Transportation Systems (2019) 1–17doi:10.1109/tits.2019.2920813.

[19] W. Zhao, M. Ammar, E. Zegura, A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks, International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc) (2004) 187–198doi:10.1145/989459.989483.

[20] Quality of Service (in Mobile Networks), Last accessed: March 2021. URL https://www.tu-ilmenau.de/fileadmin/public/iks/files/lehre/mobicom/MCN-08-QoS.pdf

[21] C. Bettstetter, S. Konig, On the message and time complexity of a distributed mobility-adaptive clustering algorithm in wireless ad hoc networks, Proceedings of European Wireless Conference (2002) 128–134.

[22] I. I. Er, W. K. Seah, Clustering overhead and convergence time analysis of the mobility-based multi-hop clustering algorithm for mobile ad hoc networks, Journal of Computer and System Sciences 72 (7) (2006) 1144–1155.

[23] A. Varga, R. Hornig, An Overview of the OMNeT++ Simulation Environment, Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools)doi:10.4108/ICST.SIMUTOOLS2008.3027.

[24] C. Sommer, R. German, F. Dressler, Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis, IEEE Transactions on Mobile Computing 10 (1) (2011) 3–15. doi:10.1109/TMC.2010.133.

[25] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y. P. Flotterod, R. Hilbrich, L. Lucken, J. Rummel, P. Wagner, E. Wiebner, Microscopic Traffic Simulation using SUMO, Proceedings of IEEE Conference on Intelligent Transportation Systems (ITSC) (2018) 2575–2582doi:10.1109/ITSC.2018.8569938.

[26] J. Zhang, H. Guo, J. Liu, Y. Zhang, Task Offloading in Vehicular Edge Computing Networks: A Load-Balancing Solution, IEEE Transactions on Vehicular Technology 69 (2) (2020) 2092–2104. doi:10.1109/TVT.2019.2959410.

[27] R. S. D. Sousa, A. Boukerche, A. A. F. Loureiro, A Distributed and Low-Overhead Traffic Congestion Control Protocol for Vehicular Ad Hoc Networks, Computer Communications 159 (March) (2020) 258–270. doi:10.1016/j.comcom.2020.05.032.

[28] T. S. Gomides, R. E. De Grande, A. M. de Souza, F. S. H. Souza, L. A. Villas, D. L. Guidoni, An Adaptive and Distributed Traffic Management System using Vehicular ad-hoc Networks, Computer Communicationsdoi:10.1016/j.future.2020.02.017.

[29] A. J. Kadhim, S. A. H. Seno, Energy-efficient Multicast Routing Protocol based on SDN and Fog Computing for Vehicular Networks, Ad Hoc Networks 84 (2019) 68–81. doi:10.1016/j.adhoc.2018.09.018.

[30] M. H. Eiza, T. Owens, Q. Ni, Q. Shi, Situation-Aware QoS Routing Algorithm for Vehicular Ad Hoc Networks, IEEE Transactions on Vehicular Technology 64 (12) (2015) 5520–5535. doi:10.1109/TVT.2015.2485305.

[31] G. Premsankar, B. Ghaddar, M. D. Francesco, R. Verago, Efficient Placement of Edge Computing Devices for Vehicular Applications in Smart Cities, IEEE/IFIP Network Operations and Management Symposium (NOMS) (2018) 1–9doi:10.1109/NOMS.2018.8406256.