Data Offloading in Vehicular Networks

Project-II (CS47006) report submitted to

Indian Institute of Technology Kharagpur

in partial fulfilment for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Yarlagadda Srihas

(18CS10057)

Under the supervision of

Prof. Arobinda Gupta

Computer Science and Engineering

Indian Institute of Technology Kharagpur

Autumn Semester, 2021-22

May 3, 2022

DECLARATION

I certify that

(a) The work contained in this report has been done by me under the guidance of my supervisor.

(b) The work has not been submitted to any other Institute for any degree or diploma.

(c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

(d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date: May 3, 2022	(Yarlagadda Srihas)
Place: Kharagpur	(18CS10057)

COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

KHARAGPUR - 721302, INDIA

CERTIFICATE

This is to certify that the project report entitled "**Data Offloading in Vehicular Networks**" submitted by **Yarlagadda Srihas** (Roll No. 18CS10057) to Indian Institute of Technology Kharagpur towards partial fulfilment of requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering is a record of bona fide work carried out by him under my supervision and guidance during Autumn Semester, 2021-22.

Date: May 3, 2022
Place: Kharagpur

Prof. Arobinda Gupta
Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur - 721302, India

Contents

List of Figures

List of Tables

Abbreviations

RSU	**R**oad **S**ide **U**nit
V2V	**V**ehicle to **V**ehicle
V2I	**V**ehicle to **I**nfrastructure
DSRC	**D**edicated **S**hort **R**ange **C**ommunications
LTE-V2X	**L**ong **T**erm **E**volution **V**ehicle to **E**verything
OBU	**O**n **B**oard **U**nit
VANET	**V**ehicular **A**d-hoc **NET**work

1 Introduction

With increasing demand for data-hungry applications in vehicle infotainment systems, as well as the requirement of connected-vehicle based safety systems, it is important to ensure that the networks used to support such systems are managed efficiently. Cellular networks - which form the backbone of mobile internet - are likely to be utilized by these new applications developed for vehicles, leading to increased congestion. Hence, it is necessary to look at methods to reduce the direct usage of cellular networks for data delivery, and the anticipated adoption of vehicular network technology offers an avenue to solve this problem.

Vehicular networks typically include road-side units (RSUs) and vehicle on-board units (OBUs), which enable both vehicle-to-vehicle (V2V) connections and vehicle-to-infrastructure (V2I) connections. Heterogeneous vehicular networks, in particular, utilize various technologies like DSRC, LTE-V2X etc. in addition to cellular links to implement V2V and vehicle to RSU communication. The cellular data offloading problem for heterogeneous vehicular networks deals with the usage of these technologies to decrease the data transfer demand on cellular networks. Specifically, when data is required to be transferred between an external entity (like a server) and a vehicle that is part of a vehicular ad-hoc network (VANET), data offloading seeks to decrease the usage of cellular bandwidth by utilizing vehicle-to-RSU and inter-vehicle communication. The cellular channel may be used for sending/receiving control information, but the overall objective is to minimize its usage.

The data offloading problem can be classified into two major sub-problems - network selection and caching. The network selection problem refers to the problem of deciding whether to use the vehicular network or directly use the cellular network for retrieval/upload of data. The caching problem refers to the selection and identification of specified vehicles/RSUs as replicas for data, so that the data requests of certain vehicles can be served by a replica. Cache replacement policies are also a part of the caching problem, and must be addressed due to the finite availability of storage.

2 Related Works

The existing works on the data offloading problem can be categorized into two parts – solutions to the network selection problem and solutions to the caching problem.

Solutions to the Network Selection Problem: In solutions to the network selection problem, the choice of network is often based on the quality-of-service requirements of different applications. QoS requirements include acceptable delay, bandwidth requirements, tolerance for missing packets/errors etc. The selection decision is informed by the availability of the data in nearby vehicles and/or RSUs, as well as network factors like traffic and packet delay.

Push-and-track based algorithms [1], [2] are one class of solutions to the network selection problem. A set of seed vehicles are selected, either by a centralized controller or by a distributed mechanism, to disseminate the required data. This selection is made using data such as the location, speed and path of the vehicles, as well as anticipated proximity to RSUs or other vehicles. Some variants [3] use a contact graph to model this vehicular network, which is updated based on these parameters.

Other solutions include the application of a suitable clustering algorithm in order to identify gateways/cluster heads that will be responsible for the cellular data transfer on behalf of surrounding vehicles. Each vehicle sends/receives data from its nearest gateway instead of directly from the cellular network. The clustering algorithm may be implemented centrally [4] using the location details of the various vehicles, or may also be implemented in a distributed fashion [5] using broadcasting techniques.

Some solutions of the network selection problem focus on associating each network with a cost at the vehicle level, which enables choice of the cellular or vehicular network on the basis of connection status and various QoS requirements. This cost is used as a metric to determine which network a particular vehicle should use at a given time for data transfer.

Solutions to the Caching Problem: The caching problem focuses on storing frequently requested data in replicas, which are specified vehicles or RSUs. The choice of replicas is informed by proximity to requesting vehicles, grouping based on content similarity etc.

The solutions to the caching problem generally include tagging of various content using keywords, or other similar schemes to group content based on the interests of various consumers [6], [7]. The cache replicas are then chosen for these groups on the basis of various factors like proximity to the consumers of that type of content, and the data is directly dispatched to them using the cellular network. The replicas then disseminate the content to the subscribers.

Other solutions use probabilistic estimates to determine whether or not caching is required at a particular vehicle/RSU. Standard statistical distributions are used to model contact durations, and an optimization problem is formulated and solved. The preference for caching some types of files over others may also be decided on the basis of such estimates. Examples of such works include [8], [9].

It can be observed that works concerned with optimal network selection do not consider the effects of finite storage, cache replacement policies etc., while those concerned with caching do not consider the mobility information of the various vehicles. A caching policy that depends on the actual pattern of V2V contacts would likely lead to better replica placement, and thereby more data savings. Moreover, there is potential to consider network quality and congestion for caching decisions, rather than limiting its applicability to network selection.

3 The Data Offloading Problem

The problem consists of multiple vehicles travelling on their predetermined routes, and RSUs placed at fixed locations. The vehicles make requests for data items, which can be served (i) by a nearby vehicle that has a portion of the data item, (ii) by a nearby RSU that has spare capacity, or (iii) directly from the cellular network, through cellular base stations. The vehicles may pass through regions having varying cellular connectivity over their entire journey. Under this initial framework, we make the following assumptions:

1. RSUs are connected to the external world with a high speed backbone network, and hence transferring any data item to a RSU takes negligible time.

2. RSUs have infinite storage. Basically, this assumption, along with the previous one, means that all the data items can be seeded into the RSUs and any vehicle can download any data item from any RSU on its path. This is assumed for simplicity at this stage. However, all vehicles will still not be able to download everything from RSUs because of the constraints on the capacity, transmission rate, and contact time.

3. For each data item, a fixed number of vehicles are initially seeded with that data item at no cost. The choice of the vehicles will depend on the algorithm. Note that even after the initial seeding, a vehicle that is not a seed originally can get the data through other sources and act as a seed, and a vehicle that is a seed may no longer become a seed as it may choose to remove the data item from its cache based on its replacement policy if the cache is full. This is captured by the set of variables w_{ikt}.

4. The storage for data items requested by a vehicle is separate from its cache and does not occupy any space in the cache, unless it is placed in the cache as per policy. Partially received data items may be cached, but in such a case the entire received portion must be made available to other vehicles.

5. A vehicle caching a data item may transfer it to another vehicle if it has a larger portion of the item than the receiving vehicle at the time of contact.

6. A vehicle requests for a specific data item only once. However, it can have more than one request open at the same time for distinct data items.

7. For any request, a vehicle only transfers the part of the data it does not have from another vehicle, RSU, or cellular base station (i.e., it does not get any part of the data more than once).

3.1 Problem Formulation

We first formally specify the inputs of the problem. Let the total time be divided into T time slots, numbered from 1 to T.

- **Network Quality:** Let $\mathcal{Q} = \{q^c_1, q^c_2, q^c_3, \ldots, q^c_k\}$ denote k possible cellular network qualities that a vehicle may be in. Let the data rate for transfer between the vehicle and the base station when the network quality is q^c_i be $d^c(q^c_i)$.

- **Vehicles:** Let $\mathcal{V} = \{v_1, v_2, v_3, \ldots, v_n\}$ be the set of vehicles, where $|\mathcal{V}| = n$. The caching capacity of a vehicle v_i is s^v_i. Let the network quality of a vehicle v_i at time t be denoted by $q^v_{i,t}$. The mean transfer rate between two vehicles is α^v. The vehicle v_i starts at time t^s_i, and reaches its destination at time t^f_i.

- **RSUs:** Let $\mathcal{R} = \{r_1, r_2, r_3, \ldots, r_m\}$ be the set of RSUs, where $|\mathcal{R}| = m$. The mean data transfer rate between a vehicle and RSU is β^r when the vehicle is in the range of an RSU, and the maximum number of vehicles that can connect to an RSU at any one time is γ^r.

- **Data Items:** Let $\mathcal{D} = \{d_1, d_2, d_3, \ldots, d_p\}$ be the set of data items that a vehicle can request for. For each $d_i \in \mathcal{D}$, the size of the data is l^d_i, and its lifetime is T^d_i.

- **Vehicle-to-Vehicle Contacts:** Let $X^{(t)}$ denote a symmetric matrix of dimension $n \times n$ where, for each position (i, j), element $X^{(t)}_{i,j} \in \{0, 1\}$ and $X^{(t)}_{i,j} = 1$ indicates that vehicles v_i and v_j are in contact range at time t.

- **Vehicle-to-RSU Contacts:** Let $Y^{(t)}$ denote a matrix of dimension $n \times m$ where, for each position (i, j), $Y^{(t)}_{i,j} \in \{0, 1\}$ and $Y^{(t)}_{i,j} = 1$ indicates that vehicle v_i and RSU r_j are in contact range at time t.

- **Requests:** Each vehicle v_i makes a sequence of requests for data items. Let $req^v_{i,j,t}$ denote a request made by the vehicle v_i for a data item d_j at time t. The request can be served by a combination of data transfer between v_i and another vehicle, an RSU, or a cellular base station.

For a given instance of the problem, we are required to find out the transfers each vehicle must make at each time instant to/from the other agents. Additionally, we also need to know which data items are placed in each vehicle's cache. Thus, the output of the problem is defined by the following sets of indicator variables:

- w_{ikt}, for all vehicles $v_i \in \mathcal{V}$, all data items $d_k \in \mathcal{D}$, and all time t, where w_{ikt} is 1 if vehicle v_i stores its obtained portion of d_k in its cache at time t, or 0 otherwise.

- x_{ijkt}, for all vehicles $v_i, v_j \in \mathcal{V}, i \neq j$, all data items $d_k \in \mathcal{D}$, and all time t, where $x_{ijkt} = 1$ if vehicle v_i transfers part of data item d_k from vehicle v_j at time t, 0 otherwise.

- y_{ijkt}, for all vehicles $v_i \in \mathcal{V}$, all RSUs $r_j \in \mathcal{R}$, all data items $d_k \in \mathcal{D}$, and all time t, where $y_{ijkt} = 1$ if vehicle v_i transfers part of data item d_k from RSU r_j at time t, 0 otherwise.

- z_{ikt}, for all vehicles $v_i \in \mathcal{V}$, all data items $d_k \in \mathcal{D}$, and all time t, where $z_{ikt} = 1$ if vehicle v_i transfers part of data item d_k using the cellular network at time t, 0 otherwise.

The main goal of the problem is to minimize the access to the cellular network. However, a trivial solution for that is to not transfer anything (so no request is satisfied). Hence, we formulate the problem as maximizing the number of requests satisfied (requested data item downloaded in its entirety within time T) while minimizing the total amount of data (over all requests, satisfied or not) transmitted using the cellular network.

Let us define an indicator variable $I_{i,k,t}$ which is set to 1 if $req^v_{i,k,t}$ is satisfied, 0 otherwise. The request will be satisfied if the total data transferred by the vehicle v_i for data item d_k from any other vehicle, RSU, or the cellular network is equal to the size of d_k. The time within which $req^v_{i,k,t}$ has to be satisfied is given by $T^{req}_{i,k,t} = \min(t + T^d_k, T)$. Finally, we use f_{ikt} to represent the amount of the data item d_k that a vehicle v_i holds at time t.

$$I_{i,k,t} = \begin{cases} 1 \text{ if } & (\sum_{v_j \in \mathcal{V}} \sum_{\tau=t}^{T^{req}_{i,k,t}} x_{ijkt}) \times \alpha^v + (\sum_{r_j \in \mathcal{R}} \sum_{\tau=t}^{T^{req}_{i,k,t}} y_{ijkt}) \times \beta^r + \sum_{\tau=t}^{T^{req}_{i,k,t}} (z_{ikt} \times d^{cell}(q^v_{i,t})) = l^d_k \\ 0 & \text{otherwise} \end{cases}$$

Then the problem can be formulated as:

$$\text{maximize} \sum_{v_i \in \mathcal{V}} \sum_{d_k \in \mathcal{D}} \sum_{t=1}^{T} I_{i,k.t}$$

$$\text{minimize} \sum_{v_i \in \mathcal{V}} \sum_{d_k \in \mathcal{D}} \sum_{t=1}^{T} z_{ikt}$$

subject to the constraints:

- A vehicle v_i can transfer a data item d_k from another vehicle v_j only if they are in range and v_j has a larger portion of the data item in its cache.

$$x_{ijkt} = 1 \implies (X^{(t)}_{i,j} = 1 \wedge w_{jkt} = 1 \wedge f_{jkt} > f_{ikt}) \quad (1)$$

- A vehicle v_i can transfer a data item d_k from a RSU r_j only if they are in range.

$$y_{ijkt} = 1 \implies Y_{i,j}^{(t)} = 1 \tag{2}$$

- The number of vehicles that can transfer data from a RSU at any time t is less than or equal to γ^r.

$$\sum_{v_i \in \mathcal{V}} \sum_{d_k \in \mathcal{D}} y_{ijkt} \leq \gamma^r, \;\; \forall r_j \in \mathcal{R}, \; \forall t \in T \tag{3}$$

- The total amount of data across all data items cached in a vehicle v_i at any time t is less than or equal to s_i^v.

$$\sum_{d_k \in \mathcal{D}} (w_{ikt} * f_{ikt}) \leq s_i^v, \;\; \forall v_i \in \mathcal{V}, \; \forall t \in T \tag{4}$$

- All transfers start after a vehicle's start time and finish before it reaches its destination.

$$x_{ijkt} = 0, y_{ijkt} = 0, z_{ijkt} = 0 \;\; \forall t < t_s^i, \; t > t_i^f \tag{5}$$

For this work, we do not consider the storage constraint (4) from the general problem described above, and thereby we assume that all received data items are cached — thus the cache replacement policy is not needed.

3.2 Model Simulation

The optimization problem proposed above is a mixed-integer problem, known to be NP-hard. A constrained form of the problem where $f_{ikt} \in \{0,1\}$ was created in LP format and solved using the Gurobi mixed-integer optimizer (Simplex, branch-and-bound with cutting planes). It was observed that for a random input on average, either the time taken to arrive at an optimal solution was very long or the model was unable to run at all due to space constraints. The model size and observed solution time for different input sizes is summarized in Table 1.

TABLE 1: Variation in optimization model size and runtime

No. of vehicles	No. of constraints	Model file size (GB)	Runtime (s)
10	356,962	0.85	15295.85
20	1,394,008	3.8	Out-of-memory
30	2,947,816	8.7	Out-of-memory
No. of data items	**No. of constraints**	**Model file size (GB)**	**Runtime (s)**
1	356,962	0.85	15295.85
5	1,193,410	4.2	Out-of-memory
10	2,238,970	8.4	Out-of-memory
Distance (km)	**No. of constraints**	**Model file size (GB)**	**Runtime (s)**
5	175,300	0.22	716.46
10	356,962	0.85	15295.85
20	719,678	3.5	$>$ 32000.00

Note: Unless otherwise mentioned, the default number of vehicles is 10, number of data items is 1 and distance is 10 km. Testbench: Lenovo ThinkPad-T480 (4-core, 16GB RAM)

Thus, the optimization model is infeasible for realistic input sizes, providing motivation for the development of a heuristic algorithm.

4 Period Based Scheduling

As a simplifying assumption, we consider the data lifetime of each data item to be equal to the total time T. We also consider that a vehicle may send/receive data to/from only a single agent at a time, and finally we consider that the cellular network is used only when a vehicle cannot complete its request by T, thus we schedule only V2V and RSU transfers (the cellular coverage quality is therefore not considered). The following is a summary of the notations used in the following discussion:

- For each RSU r_j, let the set of $\langle vehicle, item\rangle$ pairs for vehicles in range of r_j at time t be denoted by ψ_t^j. Each pair represents an open (unfulfilled) request by a vehicle for a particular data item.
- Let the set of RSU transfers for an RSU r_j at time t be denoted by ϕ_t^j. Each RSU transfer $x^j \in \phi_t^j$ has three components — i) *id*, the vehicle receiving data from r_j; ii) *item*, the data item being transferred; and iii) t_0, the time at which the transfer will end.
- Each RSU r_i is associated with γ^r slots, each of which represent a simultaneous transfer to a vehicle. The *next period* of a slot s^j, i.e. the next time instant at which it must be scheduled, is denoted by a vector element $p_{next}^j[s^j]$.
- The set of $\langle vehicle, item\rangle$ pairs corresponding to seed vehicles that have an item from the start time $t = 0$ is denoted by $\mathcal{V}_{seed}$. The set of $\langle vehicle, item\rangle$ pairs $\mathcal{V}_{new}$ is used to denote vehicles whose transfers must be updated, and is an argument to the updateTransfers function.
- The vector F represents a state of storage values, $\forall v_i \in \mathcal{V}, d_k \in \mathcal{D}$. Each element $F[v_i, d_k]$ has two components — i) *strg*, the stored amount of item d_k in v_i, i.e f_{ikt}; and *time*, the time corresponding to the stored amount.
- The vector Q represents the request times, $\forall v_i \in \mathcal{V}, d_k \in \mathcal{D}$. The request time $Q[v_i, d_k]$ is set to t if $req_{i,k,t}^v = 1$, or INT_MAX otherwise.

- The vector χ^v represents vehicular transfers, $\forall v_i \in \mathcal{V}, t$ in $\ldots 1 \ldots T$. Each transfer $\chi^v[v_i, t]$ has five components — i) *id*, the vehicle other than v_i involved in the transfer; ii) *isrecv*, a boolean value representing whether v_i is the receiver; iii) *type*, the type of transfer, either $V2V$ or RSU; iv) *item*, the data item being transferred; and v) *amnt*, the amount being transferred in that time instant.

- The two vectors D_{lost} and D_{gain} represents amounts of a data item lost and gained since the last call to updateTransfers, $\forall v_i \in \mathcal{V}, d_k \in \mathcal{D}$.

- In the score calculation, the weight given to data that can fully satisfy the request of a contacted vehicle is denoted by w_{sat}; the weight is w_{nsat} otherwise.

- Let the default scheduling period at an RSU be denoted by the constant p. Let the threshold value for updateTransfers be $\delta^r \in [0, 1]$, a fraction representing the required change in amount of a data item before updateTransfers is called.

The general idea of the algorithm is to schedule periods of time to vehicles arriving at an RSU. Vehicles are assigned a score on the basis of the amount of data they can transfer to other vehicles in the future, and the highest ranked vehicle is chosen for data transfer over the current period, whose default length is the constant value p.

Consider the scenario where, say, there are 5 vehicles $\{v_1 \ldots v_5\}$, each with an open request for 2 data items $\{d_1, d_2\}$, that are in range of an RSU r_j at a particular time instant t and none of them are already receiving data from the RSU. Scores must be calculated for each pair $\langle v_i, d_k \rangle$. The method of score calculation is illustrated as follows: suppose that the score is being calculated for $\langle v_1, d_1 \rangle$. We compute t_0, i.e. the end of the current scheduling period, taking the length of the period to be the minimum of a) the default p and b) the duration needed to complete v_1's request. Over the interval $[t_0, T]$ we consider the contacts between v_1 and the remaining vehicles given by the contact matrix $X^{(t)}$. If, for instance, v_2 has already secured $l_1^d/2$ amount of data, it would require a further $l_1^d/2$ amount to satisfy its request. For each contact between v_1 and v_2, we calculate the amount of data that can be transferred. This is checked against updated storage values (f) that are calculated factoring in the new RSU transfer. If the total contact duration in the above time interval is sufficient to transfer $l_1^d/2$ amount of data and thereby satisfy v_2's request, we add $w_{sat} * l_1^d/2$ to the score for the pair $\langle v_1, d_1 \rangle$. Otherwise the score is incremented by the maximum possible data transfer weighted by w_{nsat} (indicating a partially completed request). This is repeated for the other vehicles $v_3 \ldots v_5$ to get the score of the pair $\langle v_1, d_1 \rangle$. Thus through the score we prioritise vehicles that can i) help complete the most requests in the future and ii) transfer the most data to other vehicles.

The pair with the highest score, say $\langle v_{best}, d_{best} \rangle$, is selected for the current period (upto time t_0) and is added to list of transfers for the RSU (ϕ_t^j). Then, we clear all vehicular transfers $\chi^v[v_{best}, t]$ to/from v_{best} over the interval $[t, t_0]$ — giving priority to the RSU transfer from r_j. We then update the vehicular transfers $\chi^v[v_k, t]$ for all vehicles v_k that are contacted by v_{best} over $[t_0, T]$, considering that v_{best} receives data till the end of the scheduling period (through a call to updateTransfers).

Further, some vehicles v_k may receive a significant amount of data — and we must consider whether they can act as a cache for other vehicles. On the other hand, there may be other vehicles v_l that lose data that was previously promised — this may occur when vehicular transfers are cleared in favour of RSU transfers, for instance. Both of these cases can have a knock-on effect on other vehicles, and thus we must keep track of them and update their transfers regularly. In order to do this, we define a threshold amount $\delta^r * d_m$ (for each data item d_m) as the minimum amount of net change required to necessitate a call to updateTransfers. Whenever there is a net change above this threshold, the $\langle vehicle, item \rangle$ pair is added to $\mathcal{V}_{new}$ and we update transfers recursively. Thus the updateTransfers function plays the dual role of adding new V2V transfers, and cancelling any transfers if they do not agree with new storage values.

This procedure of choosing vehicles for RSU transfer is repeated until the RSU reaches its capacity γ^r or there are no more vehicles left that are not already on the

RSU transfer list ϕ_t^j. We then schedule transfers for the remaining RSUs, processing them in increasing order of congestion (estimated by $|\psi^j|$) in a bid to relieve RSUs that have high loads. Processing over all time instants, the algorithm finally yields the outputs — ϕ_t^j and $\chi^v[v,t]$ ($\forall t$ in $1 \dots T, \forall v \in \mathcal{V}$), where ϕ_t^j gives the allocation of RSUs to vehicles and $\chi^v[v,t]$ gives the vehicular transfers at each time instant. Cellular transfer is assumed to be used to complete any requests that are still pending at T.

The complete logic is split into four parts that are described with their accompanying pseudocode below. Algorithm 1 is used to calculate the score for a $\langle vehicle, item \rangle$ pair at an RSU. Algorithm 2 is used to get the updated storage value of a vehicle given its vehicular transfer vector. Algorithm 3 is used to update the vehicular transfers of affected vehicles, given a set of $\langle vehicle, item \rangle$ pairs $\mathcal{V}_{new}$. Algorithm 4 utilizes algorithms 1, 2 and 3 to allocate the RSUs to the vehicles.

Notes: When the storage value of vehicles is affected by a change in vehicle transfers and the magnitude of this change is below the threshold δ^r, the scores may be inaccurate — the data held by receiving vehicles v_k is incorrectly estimated while calculating scores. However, for a reasonably small value of the threshold δ^r and the default period length p the impact on the scores, and therefore on the data savings, should be low.

Moreover, after the algorithm is run there may be some illegal transfers (where source has less data than the receiver) which have not been cancelled due to the same reason. This necessitates a cleanup operation in order to remove such transfers. This cleanup function has been implemented to generate the results below, but the pseudocode has been omitted for brevity due to a significant overlap with the updateTransfers function.

Algorithm 1 − **score**$(v_i, t_0, t_1, r_j, d_k, F, Q, \chi_v)$:

```
Inputs: vehicle $v_i$, RSU transfer start time $t_0$, RSU transfer end time $t_1$, RSU $r_j$, Data item $d_k$, Set of $\langle strg, time \rangle$ pairs $F$ ($\forall v \in \mathcal{V}, d \in \mathcal{D}$), Set of request times $Q$ ($\forall v \in \mathcal{V}, d \in \mathcal{D}$), Vehicle transfers $\chi_v$
Outputs: Score for given $(v_i, d_k)$
1: $\chi_v^{temp} \leftarrow \chi_v$
2: $F^{temp} \leftarrow F$
3: Initialize $D[v]$ to 0 $\forall v$ in $\mathcal{V} \setminus v_i$
4: $\mathcal{C} \leftarrow \phi$
5: for $t$ in $t_0 \ldots (t_1 - 1)$ do
6:     Update $\chi_v^{temp}[v_i, t]$ with RSU transfer from $r_j$
7: for $t$ in $t_1 \ldots T$ do
8:     if $\chi_v^{temp}[v_i, t] \neq \phi \wedge \chi_v^{temp}[v_i, t].isrecv = false \wedge \chi_v^{temp}[v_i, t].item = d_k$ then
9:         $v_r \leftarrow \chi_v^{temp}[v_i, t].id$
10:        updateStorage($\chi_v^{temp}[v_i], F^{temp}[v_i, d_k], t$)
11:        updateStorage($\chi_v^{temp}[v_r], F^{temp}[v_r, d_k], t$)
12:        if $(F^{temp}[v_i, d_k].strg - F^{temp}[v_r, d_k].strg) > \chi_v^{temp}[v_i, t].amnt$ then
13:            $a^{new} \leftarrow \min(F^{temp}[v_i, d_k].strg - F^{temp}[v_r, d_k].strg, \alpha^v)$
14:            Add $(a^{new} - \chi_v^{temp}[v_i, t].amnt)$ to $D[v_r]$
15:            Update $\chi_v^{temp}[v_i, t].amnt$ and $\chi_v^{temp}[v_r, t].amnt$ to $a^{new}$
16:    else if $\chi_v^{temp}[v_i, t] = \phi$ then
17:        for each vehicle $v_j \in \mathcal{V}, v_j \neq v_i$ do
18:            if $Q[v_j, d_k] \leq t \wedge X_{i,j}^{(t)} = 1 \wedge \chi_v^{temp}[v_j, t] = \phi$ then
19:                updateStorage($\chi_v^{temp}[v_i], F^{temp}[v_i, d_k], t$)
20:                updateStorage($\chi_v^{temp}[v_j], F^{temp}[v_j, d_k], t$)
21:                if $(F^{temp}[v_i, d_k].strg - F^{temp}[v_j, d_k].strg) > 0$ then
22:                    $\mathcal{C} \leftarrow \mathcal{C} \cup \{v_j\}$
23:        if $\mathcal{C} \neq \phi$ then
24:            $v_{sel} \leftarrow \text{argmin}_v \{l_k^d - F^{temp}[v, d_k].strg\}$ over $v \in \mathcal{C}$
25:            Add $\min(F^{temp}[v_i, d_k].strg - F^{temp}[v_{sel}, d_k].strg, \alpha^v)$ to $D[v_{sel}]$
26:            Update $\chi_v^{temp}[v_i, t]$ and $\chi_v^{temp}[v_{sel}, t]$ with V2V transfer from $v_i$ to $v_{sel}$
27:        $\mathcal{C} \leftarrow \phi$
28: for each vehicle $v_j \in \mathcal{V}, v_j \neq v_i$ do
29:     if $F^{temp}[v_j, d_k].time < T$ then
30:         updateStorage($\chi_v^{temp}[v_j], F^{temp}[v_j, d_k], T$)
31:     if $F^{temp}[v_j, d_k].strg \geq l_k^d$ then
32:         Add $w_{sat} * (D[v_j])$ to $score$
33:     else
34:         Add $w_{nsat} * (D[v_j])$ to $score$
35: return $score$
```

Algorithm 1 Description: The idea of score calculation is to consider the transfers that can be made by a vehicle if it is chosen to receive data from an RSU.

In lines 1-2, we first make copies of the vehicular transfers vector (χ_v^{temp}) and storage values (F^{temp}). The temporary transfers for the given vehicle v_i are updated assuming data transfer from the RSU r_j (Lines 5-7). Then for each time instant till the time horizon T, we first look for existing vehicular transfers from v_i to other vehicles. If there is such a scheduled transfer to vehicle v_r, we update the storage values F_{temp} based on the temporary transfers and check whether a larger amount of data can be transferred now (Lines 8-15). We keep track of this data by updating the data transfer vector $D[v_r]$ and the temporary transfers χ_v^{temp}. If there is a time instant where no transfer is currently scheduled, we find candidate vehicles that have requested for the item, are in contact and are free for transfer (Lines 18-22). These are added to $\mathcal{C}$. From $\mathcal{C}$, the vehicle that is closest to completing its request is selected (v_{sel}), and $D[v_{sel}]$ and the temporary transfers are updated. (Lines 24-26). Finally, to compute the score we check the amount $D[v_j]$ for every receiving vehicle v_j — if it is enough to satisfy the request we apply a weight w_{sat}, else we use w_{nsat} and add it to the score (Lines 28-32). The score is returned.

Algorithm 2 – **updateStorage**($\chi_v[v_i], F[v_i, d_k], t_0$):

```
   Inputs: List of vehicle transfers $\chi_v[v_i]$ for vehicle $v_i$, $\langle strg, time\rangle$ pair $F[v_i, d_k]$
   (for corresponding $v_i \in \mathcal{V}, d_k \in \mathcal{D}$), time instant for update $t_0$
   Updates: $\langle strg, time\rangle$ pair $F[v_i, d_k]$
1: for $t$ in $F[v_i, d_k].time \dots t_0$ do
2:     if $\chi_v[v_i, t].item = d_k \wedge \chi_v[v_i, t].isrecv = true$ then
3:         Add $\chi_v[v_i, t].amnt$ to $F[v_i, d_k].strg$
4: if $F[v_i, d_k].strg > l_k^d$ then
5:     $F[v_i, d_k].strg \leftarrow l_k^d$
6: $F[v_i, d_k].time \leftarrow t_0$
7: return
```

Algorithm 2 Description: The updateStorage algorithm updates the storage values $\langle strg, time\rangle$ for a given vehicle v_i, data item d_k, its transfers $\chi_v[v_i]$ and an update time t_0. For each time instant between the current value of $F[v_i, d_k].time$ and t_0 we add the received transfer amount (if any) to the vehicle storage, capping at the length of the data item. Finally, the *time* value is updated to t_0.

Algorithm 3 − **updateTransfers**($\mathcal{V}_{new}, F, Q, \chi_v, t_{cur}, D_{lost}, D_{gain}$)

```
1:  Inputs: Set $\mathcal{V}_{new}$ of $\langle vehicle, item\rangle$ pairs to update transfers, Set of $\langle strg, time\rangle$ pairs
    $F$ ($\forall v \in \mathcal{V}, d \in \mathcal{D}$), Set of request times $Q$ ($\forall v \in \mathcal{V}, d \in \mathcal{D}$), Vehicle transfers $\chi_v$,
    current time instant $t_{cur}$
2:  Updates: Vehicle transfers $\chi_v$, Set of $\langle strg, time\rangle$ pairs $F$
3:  Initialize $F^{temp}$ with $\langle 0, 0\rangle$ $\forall v \in \mathcal{V}, d \in \mathcal{D}$
4:  $\mathcal{C}_{pair} \leftarrow \phi$
5:  for each vehicle $v_i$ s.t. $\exists$ $\langle v_i, d\rangle \in \mathcal{V}_{new}$ do
6:      Set $D_{lost}[v_i, d]$ to 0 $\forall d \in \mathcal{D}$, $D_{gain}[v_i, d]$ to 0 $\forall d$ s.t. $\langle v_i, d\rangle \in \mathcal{V}_{new}$
7:      for $t$ in $1 \ldots T$ do
8:          if $\chi_v^{temp}[v_i, t] \neq \phi \wedge \chi_v^{temp}[v_i, t].type = V2V$ then
9:              $v_j \leftarrow \chi_v^{temp}[v_i, t].id$
10:             $d_k \leftarrow \chi_v^{temp}[v_i, t].item$
11:             updateStorage($\chi_v[v_i], F^{temp}[v_i, d_k], t$)
12:             updateStorage($\chi_v[v_j], F^{temp}[v_j, d_k], t$)
13:             $a_{diff} \leftarrow F^{temp}[v_i, d_k].strg - F^{temp}[v_j, d_k].strg$
14:             if $\chi_v^{temp}[v_i, t].isrecv = true$ then
15:                 $a_{diff} \leftarrow a_{diff} * (-1)$
16:             if $a_{diff} \leq 0$ then
17:                 if $\chi_v^{temp}[v_i, t].isrecv = false$ then
18:                     Add $\chi_v^{temp}[v_i, t].amnt$ to $D_{lost}[v_j, d_k]$
19:                 Clear $\chi_v^{temp}[v_i, t]$ and $\chi_v^{temp}[v_j, t]$
20:             else
21:                 $a_{new} \leftarrow \min(\alpha^v, a_{diff})$
22:                 if $\chi_v^{temp}[v_i, t].isrecv = false \wedge a_{new} > \chi_v^{temp}[v_i, t].amnt$ then
23:                     Add $a_{new} - \chi_v^{temp}[v_i, t].amnt$ to $D_{gain}[v_j, d_k]$
24:                 else if $\chi_v^{temp}[v_i, t].isrecv = false$ then
25:                     Add $\chi_v^{temp}[v_i, t].amnt - a_{new}$ to $D_{lost}[v_j, d_k]$
26:                 Update $\chi_v^{temp}[v_i, t].amnt$ and $\chi_v^{temp}[v_r, t].amnt$ to $a^{new}$
27:                 continue;
28:         else
29:             continue;
30:         if $\chi_v^{temp}[v_i, t] = \phi$ then
31:             for each data item $d_k$ s.t $\langle v_i, d_k\rangle \in \mathcal{V}_{new}$ do
32:                 for each vehicle $v_j \in \mathcal{V}, v_j \neq v_i$ do
33:                     if $Q[v_j, d_k] \leq t \wedge X_{i,j}^{(t)} = 1 \wedge \chi_v^{temp}[v_j, t] = \phi$ then
34:                         updateStorage($\chi_v^{temp}[v_i], F^{temp}[v_i, d_k], t$)
35:                         updateStorage($\chi_v^{temp}[v_j], F^{temp}[v_j, d_k], t$)
36:                         if $(F^{temp}[v_i, d_k].strg - F^{temp}[v_j, d_k].strg) > 0$ then
37:                             $\mathcal{C}_{pair} \leftarrow \mathcal{C}_{pair} \cup \{\langle v_j, d_k\rangle\}$
38:             if $\mathcal{C}_{\surd\dashv\rangle\triangledown} \neq \phi$ then
39:                 $\langle v_{sel}, d_{sel}\rangle \leftarrow$ argmin$\{l_c^d - F^{temp}[v_c, d_c].strg\}$ over $\langle v_c, d_c\rangle \in \mathcal{C}_{pair}$
40:                 Update $\chi_v^{temp}[v_i, t]$ and $\chi_v^{temp}[v_{sel}, t]$ with V2V transfer from $v_i$ to $v_{sel}$
41:                 Add $\chi_v^{temp}[v_i, t].amnt$ to $D_{gain}[v_{sel}, d_{sel}]$
42:             $\mathcal{C}_{pair} \leftarrow \phi$
43:             if $t = t_{cur}$ then
44:                 $F \leftarrow F^{temp}$
45:     $\mathcal{V}_{new} \leftarrow \mathcal{V}_{new} \cup \{\langle v_n, d_n\rangle \mid ||D_{gain}[v_n, d_n] - Dlost[v_n, d_n]|| > \delta^v * l_n^d\}$
46:     $\mathcal{V}_{new} \leftarrow \mathcal{V}_{new} \setminus \{\langle v_i, d\rangle \mid \langle v_i, d\rangle \in \mathcal{V}_{new}\}$
47: return
```

Algorithm 3 Description: The updateTransfers function is called to update vehicle transfers for a set V_{new} of $\langle vehicle, item\rangle$ pairs. Specifically, we check for new transfers where the *vehicle* acts as a source of the data item *item* for other vehicles, while for existing transfers we check whether the data amount can be updated (for all data items). We initialize a temporary storage vector (F^{temp}) to be used to keep track of newly transferred data, and a candidate set $\mathcal{C}_{pair}$ (Line 3). We loop over each unique vehicle v_i appearing in a pair of V_{new}, first resetting the values of data lost/gained for v_i ($D_{lost}[v_i, d], D_{gain}[v_i, d]$) since the last call to updateTransfers. $D_{gain}[v_i, d]$ is reset only for data items appearing in a pair with v_i in V_{new} while $D_{lost}[v_i, d]$ is reset for all data items, in accordance with the types of transfers checked in this function (Lines 4-6).

For each time instant, we first check existing V2V transfers. The storage values are updated based on the current transfers, and we check the difference in vehicle storage values (Lines 9-13). If the source vehicle now has less data than the receiver, we cancel the transfer and clear it from the transfer list of the receiver and the source, D_{lost} is updated accordingly (Lines 16-19). If the transfer is still possible but with a different data amount, the *amnt* value of the transfer is updated. D_{lost} or D_{gain} is updated depending on whether the new amount is lesser or greater than before (Lines 20-27).

If there is no transfer scheduled, we prepare a list of candidate vehicles for transfer at this instant. For each data item d_k that appears with the current vehicle v_i in $\mathcal{V}_{new}$, we look for vehicles v_j that are in range, have requested for d_k and are free for transfer, and add $\langle v_j, d_k\rangle$ to a set of candidates $\mathcal{C}_{pair}$ (Lines 30-37). Of the candidate pairs in $\mathcal{C}_{pair}$, we select the vehicle closest to completing its transfer. χ^v and D_{gain} are updated for the selected vehicle (Lines 30-41). The actual storage values F are updated to F^{temp} when the time instant matches the supplied t_{cur}. Finally, V_{new} is updated: all $\langle vehicle, item\rangle$ pairs such that the data lost/gained (as per D_{lost}, D_{gain}) is greater than a threshold are added, and pairs in which the current vehicle v_i appears are removed.

Algorithm 4 Period-based Scheduling

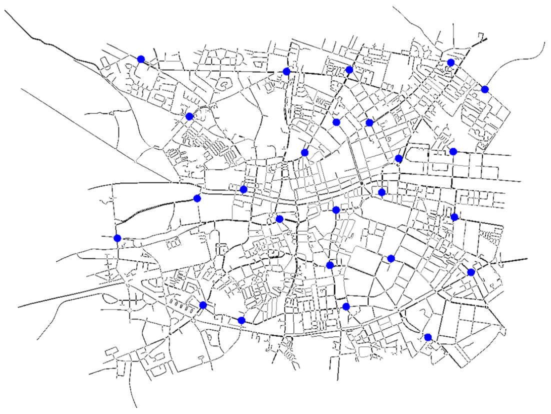
```
1: Initialize $F[v_i, d_k]$ to $\langle l^d_k, 0\rangle$ $\forall\langle v_i, d_k\rangle \in \mathcal{V}_{seed}$, $\langle 0, 0\rangle$ otherwise
2: Initialize $Q[v_i, d_j]$ to $t$ if $\exists$ $t$ s.t. $req^v_{i,j,t} = 1$, INT_MAX otherwise
3: $\mathcal{V}_{new} \leftarrow \mathcal{V}_{seed}$, Initialize $D_{lost}[v_i, d_k]$ and $D_{gain}[v_i, d_k]$ to 0 $\forall v_i \in \mathcal{V}, d_k \in \mathcal{D}$
4: **updateTransfers**($\mathcal{V}_{new}, F, Q, \chi_v, 0$)
5: **for** $t$ in $1 \ldots T$ **do**
6:     $\mathcal{R}_{sched} \leftarrow \phi$
7:     **for** *each* RSU $r_j$ **do**
8:         $\phi^j_t \leftarrow \phi^j_{t-1}$
9:         **for** *each* RSU transfer $x^j \in \phi^i_t$ **do**
10:            **if** $x^j.t_0 = t$ **then**
11:                $\phi^j_t \leftarrow \phi^j_t \setminus \{x^j\}$
12:        $sched \leftarrow false$
13:        **if** $\exists s^j$ in $1 \ldots \gamma^r$ s.t. $p^j_{next}[s^j] = t$ **then**
14:            $sched \leftarrow true$
15:        **for** *each* vehicle $v_i$ s.t. $(\exists\ \langle v_i, d\rangle \in \psi^j) \wedge (Y^{(t)}_{i,j} = 0)$ **do**
16:            $\psi^j \leftarrow \psi^j \setminus \{\langle v_i, d\rangle \mid \langle v_i, d\rangle \in \psi^j\}$
17:        **for** *each* vehicle $v_i \in \mathcal{V}$ **do**
18:            **if** $Y^{(t)}_{i,j} = 1 \wedge (x^j.id \neq v_i,\ \forall x^j \in \phi^i_t)$ **then**
19:                **for** *each* data item $d_k \in \mathcal{D}$ **do**
20:                    **if** $Q[v_i, d_k] \leq t \wedge \langle v_i, d_k\rangle \notin \psi^j$ **then**
21:                        **updateStorage**($\chi_v[v_i], F[v_i, d_k], t$)
22:                        **if** $F[v_i, d_k].strg < l^d_k$ **then**
23:                            $\psi^j \leftarrow \psi^j \cup \{\langle v_i, d_k\rangle\}$
24:                            $sched \leftarrow true$
25:        **if** $sched = true$ **then**
26:            Insert $r_j$ into $\mathcal{R}_{sched}$ in increasing order by $|\psi^j|$
27:    **for** *each* RSU $r_j \in \mathcal{R}_{sched}$ in *order* **do**
28:        **for** *each* slot $s^j$ in $1 \ldots \gamma^r$ s.t. $p^j_{next}[s^j] \leq t$ **do**
29:            $\mathcal{C}_{score} \leftarrow \phi$
30:            **for** *each* pair $\langle v_i, d_k\rangle \in \psi^j$ **do**
31:                $t_0 \leftarrow \min(t + ceil[(l^d_k - F[v_i, d_k].strg)/\beta^r], t + p, T)$
32:                **for** $t_c$ in $t \ldots t_0$ **do**
33:                    **if** $Y^{(t)}_{i,j} = 0 \vee (\chi_v[v_i, t_c] \neq \phi \wedge \chi_v[v_i, t_c].type = RSU)$ **then**
34:                        $t_0 \leftarrow t_c$
35:                        **break;**
36:                **if** $t_0 > t$ **then**
37:                    $\mathcal{C}_{score} \leftarrow \{x^j(id = v_i, item = d_k, t_0 = t_0)\}$
38:            $x^j_{best} \leftarrow \text{argmax}_{x^j}\{\textbf{score}(x^j.id, t, x^j.t_0, r_j, x^j.item, F, R, \chi_v)\}$ over $x^j \in \mathcal{C}_{score}$
39:            $\phi^j_t \leftarrow \phi^j_t \cup \{x^j_{best}\}$
40:            $\psi^j \leftarrow \psi^j \setminus \{\langle x^j_{best}.id, d\rangle \mid \langle x^j_{best}.id, d\rangle \in \psi^j\}$
41:            **for** $\tau$ in $t \ldots (x^j_{best}.t_0)$ **do**
42:                **if** $\chi_v[v_i, t] \neq \phi$ **then**
43:                    $v_r \leftarrow \chi_v[v_i, \tau].id$
44:                    Clear $\chi_v[v_i, \tau]$ and $\chi_v[v_r, \tau]$, update $D_{lost}[v_r, \chi_v[v_i, \tau].item]$
45:                Update $\chi_v[v_i, \tau]$ with RSU transfer from $r_j$
46:            $\mathcal{V}_{new} \leftarrow \mathcal{V}_{new} \cup \{\langle x^j_{best}.id, x^j_{best}.item\rangle\}$
47:            **updateTransfers**($\mathcal{V}_{new}, F, Q, \chi_v, t$)
48:            $p^j_{next}[s^j] \leftarrow x^j_{best}.t_0$
```

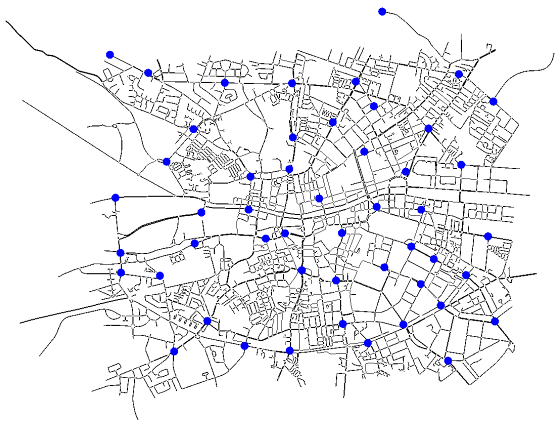
Algorithm 4 Description: Lines 1-2 initialize the storage values (F) and the request time vector Q based on the given seed vehicles and list of requests. V_{new} is populated with pairs from V_{seed}, and updateTransfers is called to account for V2V transfers from seed vehicles (Lines 3-4). Then, for every time instant we schedule transfers at RSUs. For each RSU r_j, we first remove completed transfers from the list of transfers (ϕ_t^j) (Lines 8-11). Then, we assess whether the RSU needs scheduling at this time instant, reflected by the boolen value *sched*. If there is any slot whose next period p_{next}^j has arrived, a transfer is complete and therefore *sched* is set to true. Scheduling may also be needed if a new vehicle is added to the set ψ^j of in-range $\langle vehicle, item \rangle$ pairs. For this, we look for vehicles v_i in contact with r_j that are not already on the transfer list (Lines 17-18), update their storage for each requested data item d_k (19-21), check if the request has already been completed and if the pair is already in ψ^j. If not, the pair $\langle v_i, d_k \rangle$ is added to ψ^j, and *sched* is set to reflect that a new pair has been added. Note that a previous pair may be added again after completing its transfer as ϕ_t^j has already been updated to exclude completed transfers. If *sched* is true, the RSU r_j is added to the ordered list of RSUs $\mathcal{R}_{sched}$, where they are stored in increasing order of congestion ($|\psi^j|$).

For each RSU on $\mathcal{R}_{sched}$, we loop over each slot till we find the one whose next period has arrived (Lines 27-28). Then for each pair in range (ψ^j), we calculate t_0, the time instant up to which it would get the RSU if it were scheduled. (Lines 31-35). A candidate transfer corresponding to each pair is added to a set $\mathcal{C}_{score}$. We calculate scores for each candidate transfer using Algorithm 1 as described above, and choose the transfer x_{best}^j with the highest score. x_{best}^j is added to the list of transfers of the RSU, and its corresponding pairs are removed from ψ^j to exclude it from consideration till t_0. From the current time instant till t_0 we update the chosen vehicle with the RSU transfers, clearing any vehicular transfers that may have been scheduled and updating D_{lost} accordingly. Finally, the pair $\langle x_{best}^j.id, x_{best}^j.item \rangle$ are added to V_{new}, and updateTransfers is called to account for further transfers to contacted vehicles. The next period time p_{next}^j is updated for the scheduled slot.

5 Simulation Setup

Road Network: The **S**imulation of **U**rban **Mo**bility (SUMO) [10] tool was used to generate vehicle positions from traffic flows. The road network used for the traffic simulation was an urban map covering an area of 3.5x5 km in the city centre of Dublin, Ireland, with 435 intersections. The sumo configuration file, road network, traffic sensor positions and vehicle demand data were sourced from a recent work on autonomous vehicle safety [11].


(A) 25 RSUs


(B) 50 RSUs

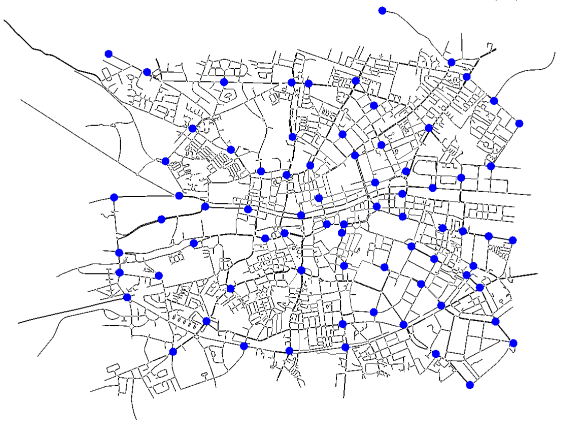

(C) 75 RSUs

FIGURE 1: RSU positions in the road network.

RSU Positions: The positions corresponding to the SCATS sensor locations were extracted from the SUMO configuration file. As these sensors are spread fairly uniformly across the area, they offered good candidates for RSU positions. However there were several sensor locations in close proximity, and thus k-means clustering was used to extract RSU positions that were sufficiently separated to minimize overlapping. Three 'k' values were used for three different numbers of RSUs — 25, 50 and 75. The sensor location nearest to each mean was taken as an RSU location. The RSU positions for each density are mapped in Figure 1.

Traffic loads: The traffic flow data represents readings from the Dublin SCATS dataset averaged over several months, as detailed in [11]. For the purpose of this simulation, the vehicle model corresponding to Scenario A (all human drivers) was used from the same work. The SUMO simulation was sampled over a duration of 150 seconds (18000-18150s) to yield a low vehicle demand with $\approx$ 250 vehicles. Then, the traffic was scaled using SUMO's *--scale* option to generate traffic for the medium and high loads with $\approx$ 500 and $\approx$ 750 vehicles respectively.

Requests: Each simulation was run with 2 data items. The requests were generated as follows — half of the vehicles were chosen randomly to request the first data item and the remaining vehicles were chosen to request the second data item. Of each group of vehicles, 20 percent of vehicles were chosen to request for both the data items. The same random seed was used across test cases for comparability. Thus, the request pattern involved roughly equal number of requests for each data item, with 20 percent of vehicles requesting both items.

6 Results

We have evaluated the performance of the algorithm by simulating over some initial scenarios. In all the inputs considered, the map used was the same as described in the previous section. Three values of $|\mathcal{V}|$ (number of vehicles) were used for testing — 250 and 500 and 750. Two data items were considered for each test case, with sizes 300Mb and 500Mb. Seed vehicles were not included in any of the test cases ($\mathcal{V}_{seed} = \phi$). The remaining parameters such as number of RSUs, vehicle rate α^v, RSU capacity γ^r and period length p were varied across test cases. The values used are summarized in Table 2. Using the request pattern as described in the previous section, the number of requests for each data item and the total data demand for each value of $|\mathcal{V}|$ is summarized in Table 3.

Table 2: Values of constants used

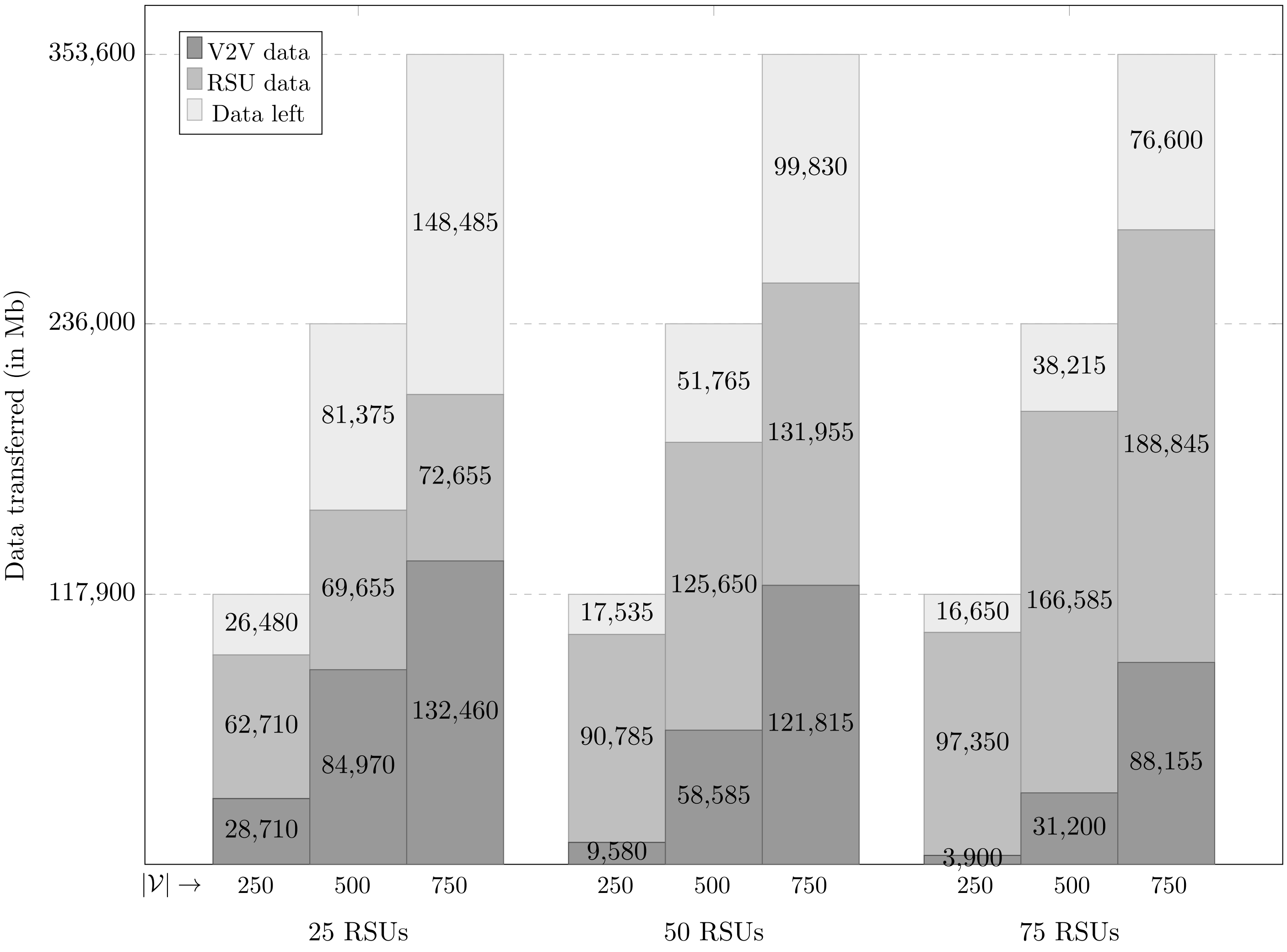
Quantity	Value
RSU range	300m
β^r	10 Mbps
Vehicle-to-vehicle range	300m
w_{sat}	1.5
w_{nsat}	1
δ^r	0.4
l_1^d	300Mb
l_2^d	500Mb

Table 3: Number of requests and total data demand

Number of vehicles	Number of requests			Data demand (Mb)		
	d_1	d_2	Total	d_1	d_2	Total
250	148	147	295	44,400	73,500	117,900
500	295	295	590	88,500	147,500	236,000
750	442	442	884	132,600	221,000	353,600

Number of RSUs: The algorithm was run for each value of $|\mathcal{R}|$ — 25 RSUs, 50 RSUs and 75 RSUs — with a vehicle transfer rate α^v of 5Mbps, i.e. half of the RSU transfer rate β^r. The default scheduling period length p was set to 10s, and the maximum number of concurrent transfers γ^r for RSUs was set to 2. The data transferred through RSU-to-vehicle and vehicle-to-vehicle communication is plotted in Figure 2, and the number of requests satisfied is summarized in Table 4.

FIGURE 2: Data transferred v/s number of RSUs



As expected, increasing the number of RSUs corresponds to a significant increase in the amount of data transferred, and number of requests satisfied. The amount of data transferred was ≈ 60% of the total in the worst case (with 750 vehicles and 25 RSUs), and ≈ 86% in the best case (with 250 vehicles and 75 RSUs). An

TABLE 4: Number of requests satisfied (varying RSUs)

Test case	Requests satisfied			Total requests satisfied (within 10%)
	d_1	d_2	Total	
250 vehicles, 25 RSUs	84	74	158	176
250 vehicles, 50 RSUs	109	109	218	229
250 vehicles, 75 RSUs	111	110	221	232
500 vehicles, 25 RSUs	121	45	166	214
500 vehicles, 50 RSUs	182	175	357	383
500 vehicles, 75 RSUs	214	213	427	441
750 vehicles, 25 RSUs	140	16	156	246
750 vehicles, 50 RSUs	221	147	368	432
750 vehicles, 75 RSUs	285	252	537	570

interesting result can be observed comparing the cases $(|\mathcal{V}| = 750, |\mathcal{R}| = 25)$ and $(|\mathcal{V}| = 750, |\mathcal{R}| = 50)$ — the V2V transfers decreased by a small percentage (37.4% to 34.45%) while the RSU transfers increased significantly (20.5% to 37.3%). This indicates that the increased RSU density helped the overall utilization. We also observe diminishing returns when adding a large number of RSUs as the V2V transfers begin to be affected.

Vehicle rate: The V2V transfer rate α^v was increased to 10 Mbps, making it equal to the RSU transfer rate β^r. $|\mathcal{R}|$ was set to 50, and the remaining parameters had the same values as described in the previous section. The results are summarized in Figure 3 and Table 5.

In terms of total data transferred, it was observed that increasing V2V transfer rate led to an improvement — the amount of data left was smaller in all cases in line with expectations. Additionally, the improvement observed was positively correlated to the number of vehicles as expected, the percentage improvement in total data transfer was 1.06%, 4.84% and 7.82% respectively for each increasing value of $|\mathcal{V}|$. It can also be observed that the increase in V2V transfers was accompanied by a (smaller) decrease in RSU transfers in each case.

FIGURE 3: Data transferred v/s vehicle rate α^v

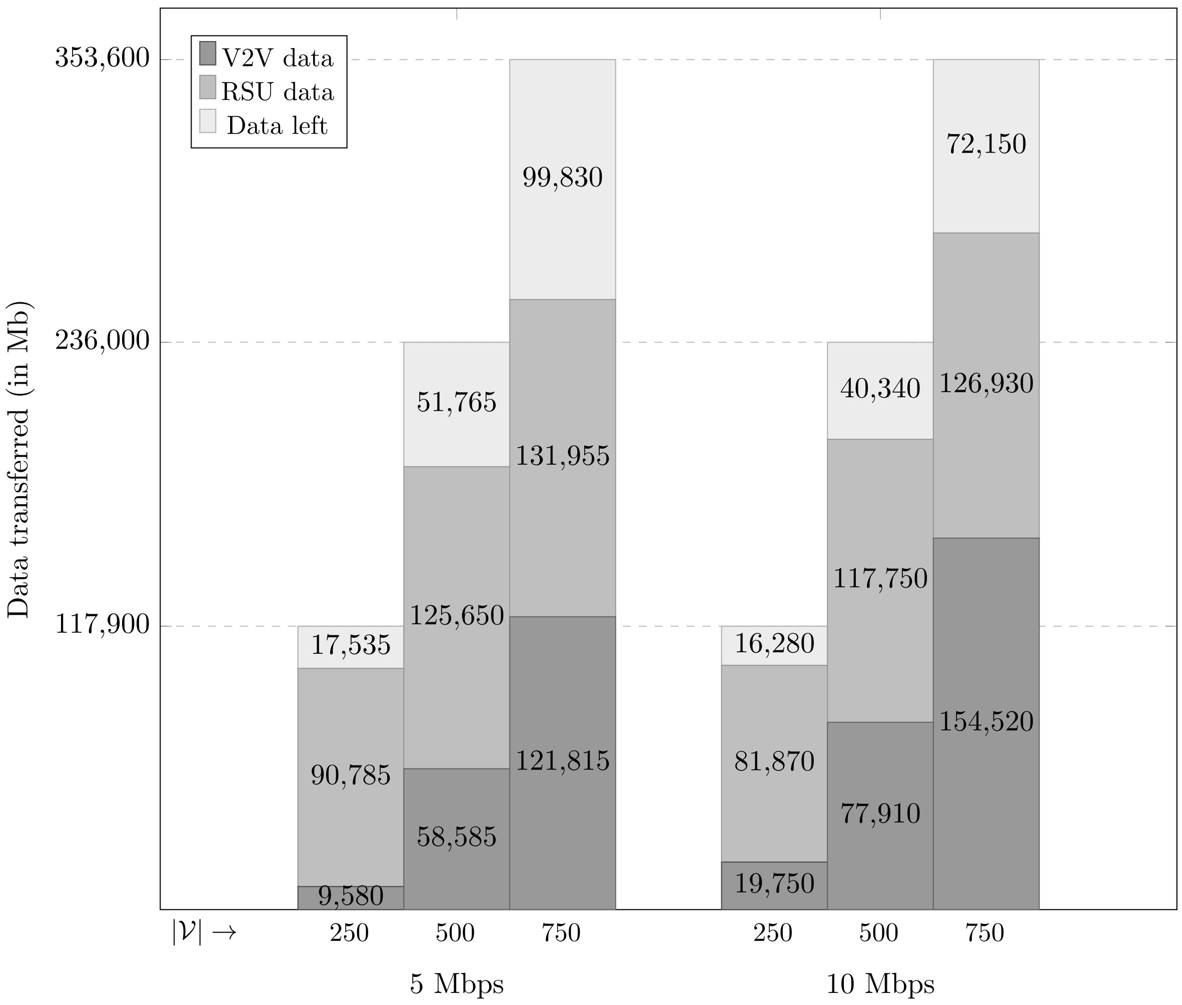


TABLE 5: Number of requests satisfied (varying V2V transfer rate α^v)

Test case	Requests satisfied			Total requests satisfied (within 10%)
	d_1	d_2	Total	
250 vehicles, 5 Mbps	109	109	218	229
250 vehicles, 10 Mbps	92	109	201	232
500 vehicles, 5 Mbps	182	175	357	383
500 vehicles, 10 Mbps	74	175	249	357
750 vehicles, 5 Mbps	221	147	368	432
750 vehicles, 10 Mbps	52	153	205	449

On the other hand, we see that there is actually a decrease in the total number of requests satisfied. This indicates that increasing the vehicle rate leads to the data being spread more uniformly across the vehicles. This is further supported by the number of requests that were satisfied within 10% — this number either increased or had a smaller drop.

RSU capacity: The RSU capacity (γ^r) was doubled from 2 to 4, thus allowing more concurrent transfers at each RSU. The remaining parameters were kept the same as the previous section with $|\mathcal{R}|$ set to 50 and $\alpha^v = 5$ Mbps. The results are summarized in Figure 4 and Table 6.

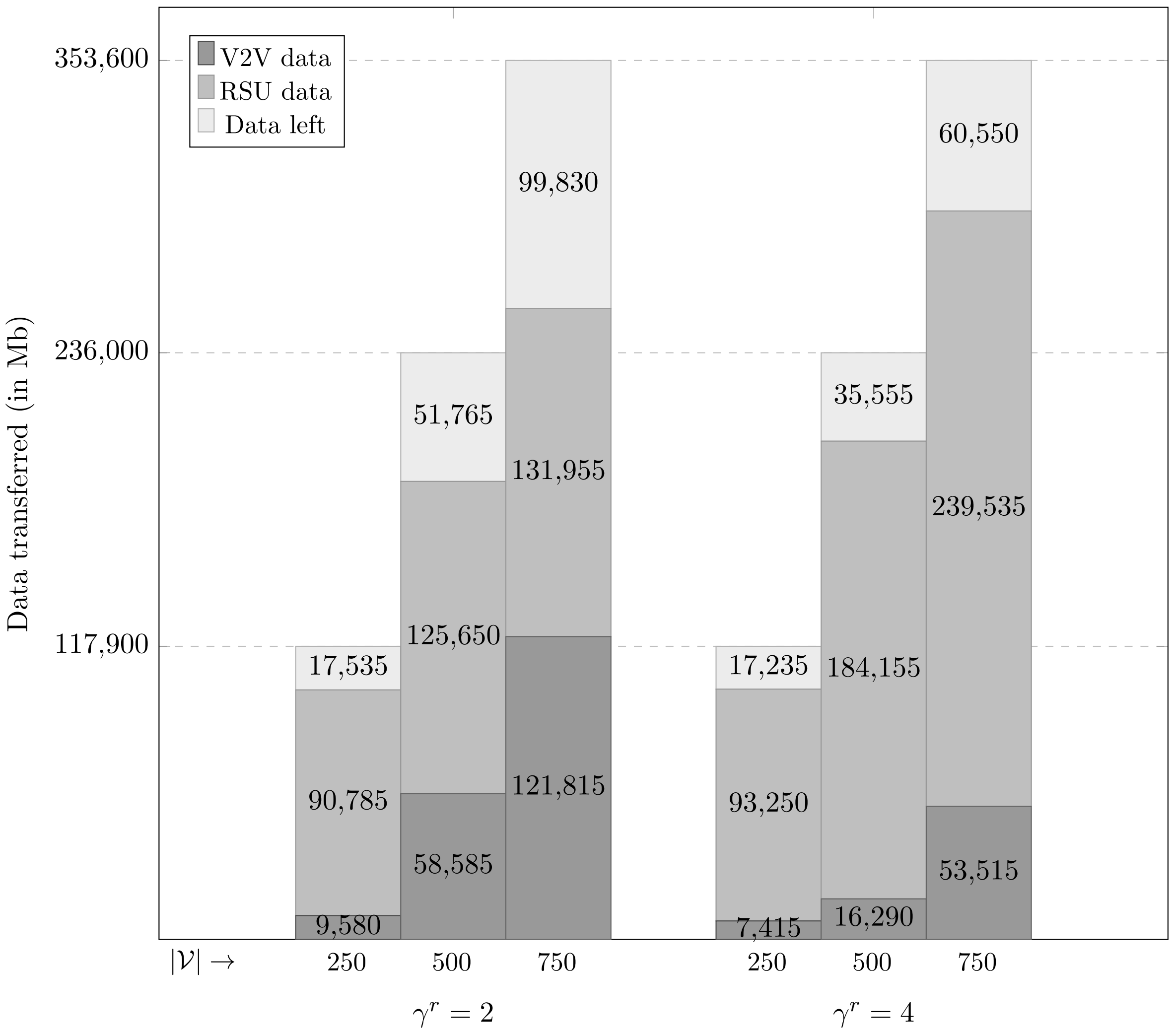
A large change in the proportion of RSU transfers was observed as expected. This led to a corresponding decrease in V2V transfers, while the total data transferred saw an improvement in all cases. The largest improvement in total data transfer ($\approx$ 11%) was observed with 750 vehicles, i.e. the case likely to have the largest contention at each RSU.

TABLE 6: Number of requests satisfied (varying RSU capacity γ^r)

Test case	Requests satisfied			Total requests satisfied (within 10%)
	d_1	d_2	Total	
250 vehicles, $\gamma^r = 2$	109	109	218	229
250 vehicles, $\gamma^r = 4$	111	110	221	231
500 vehicles, $\gamma^r = 2$	182	175	357	383
500 vehicles, $\gamma^r = 4$	217	222	439	464
750 vehicles, $\gamma^r = 2$	221	147	368	432
750 vehicles, $\gamma^r = 4$	300	306	606	655

A similar pattern was observed with the number of requests — with significant increases in the total number of requests satisfied across the test cases. This provides further support for the notion that a larger proportion of RSU transfers promotes more requests being completed, while a larger proportion of V2V transfers leads to a wider distribution of data as seen in the previous section.

FIGURE 4: Data transferred v/s RSU capacity γ^r



Default period length p : Finally, the effect of the default scheduling period was tested. A larger value of p would theoretically lead to longer and more continuous segments of RSU transfer to a particular vehicle before switching, at the expense of less accurate scores. For each test case, a period of 25s was used, and some of representative results are summarized in Table 7. All parameters other than the period length are the same as used the previous section.

It was observed that the number of requests satisfied decreased across the board upon increasing the period. This can be attributed to the fact that scores are calculated less frequently, and thus more data is given to vehicles that may no

TABLE 7: Requests satisfied (varying period length p)

Test case	Requests satisfied	Requests (within 10%)
250 vehicles, $p = 10s$	218	229
250 vehicles, $p = 25s$	221	229
500 vehicles, $p = 10s$	357	383
500 vehicles, $p = 25s$	305	372
750 vehicles, $p = 10s$	368	432
750 vehicles, $p = 25s$	302	410

longer be the highest scorers at a given time instant. Small net decreases to the overall data transfer ($\approx 3.6\%$ in the worst case of 750 vehicles) were observed, with slightly increased RSU transfers and decreased V2V transfers.

7 Future Scope of Work

In this thesis, we have addressed the problem of cellular data offloading in a vehicular network with multiple vehicles, RSUs and data items. A period-based scheduling algorithm has been proposed, which considers RSU congestion and computes RSU-to-vehicle transfers based on a heuristic score. Vehicle-to-vehicle transfers are scheduled with consideration for partial data item caches. The algorithm has been tested on a traffic map with real world data, and results on the total data transfer and number of requests satisfied have been presented. The areas identified for future expansions to this work are:

- Incorporate cellular downloads during the vehicle's journey. The proposed algorithm deals with the allocation of RSUs to vehicles to disseminate data, but it may be beneficial to use the cellular network to seed a vehicle early in its journey if it can contribute more to other vehicles on its path. The quality of cellular coverage and its impact on data rate may also be considered for caching decisions.

- Implement fairness/priority for the data items. While choosing a receiving vehicle for a V2V transfer, we select the ones nearest to completing their requests which may have an effect on fairness. There is scope to add data item fairness by modifying this selection criteria, or to implement a priority based ordering.

- Network level congestion may be considered in detail. In a multiplexed network with a common communication medium, the data transfer rates and number of simultaneous transfers may be affected by congestion. This can be an additional factor for caching decisions.

References

[1] John Whitbeck, Yoann Lopez, Jérémie Leguay, Vania Conan, and Marcelo Dias De Amorim. Push-and-track: Saving infrastructure bandwidth through opportunistic forwarding. *Pervasive and Mobile Computing*, 8(5):682–697, 2012.

[2] Filippo Rebecchi, Marcelo Dias de Amorim, and Vania Conan. Droid: Adapting to individual mobility pays off in mobile data offloading. In *2014 IFIP Networking Conference*, pages 1–9, 2014.

[3] Hong Yao, Deze Zeng, Huawei Huang, Song Guo, Ahmed Barnawi, and Ivan Stojmenovic. Opportunistic offloading of deadline-constrained bulk cellular traffic in vehicular dtns. *IEEE Transactions on Computers*, 64:1–1, 12 2015.

[4] Shucong Jia, Sizhe Hao, Xinyu Gu, and Lin Zhang. Analyzing and relieving the impact of fcd traffic in lte-vanet heterogeneous network. In *2014 21st International Conference on Telecommunications (ICT)*, pages 88–92, 2014.

[5] Abderrahim Benslimane, Tarik Taleb, and Rajarajan Sivaraj. Dynamic clustering-based adaptive mobile gateway management in integrated vanet — 3g heterogeneous wireless networks. *IEEE Journal on Selected Areas in Communications*, 29(3):559–570, 2011.

[6] Yong Li, Depeng Jin, Zhaocheng Wang, Lieguang Zeng, and Sheng Chen. Coding or not: Optimal mobile data offloading in opportunistic vehicular networks. *IEEE Transactions on Intelligent Transportation Systems*, 15(1):318–333, 2014.

[7] Xiangming Zhu, Yong Li, Depeng Jin, and Jianhua Lu. Contact-aware optimal resource allocation for mobile data offloading in opportunistic vehicular networks. *IEEE Transactions on Vehicular Technology*, 66(8):7384–7399, 2017.

[8] Luigi Vigneri, Thrasyvoulos Spyropoulos, and Chadi Barakat. Storage on wheels: Offloading popular contents through a vehicular cloud. In *2016 IEEE*

17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), pages 1–9, 2016.

[9] Weicheng Zhao, Yajuan Qin, Deyun Gao, Chuan Heng Foh, and Han-Chieh Chao. An efficient cache strategy in information centric networking vehicle-to-vehicle scenario. *IEEE Access*, 5:12657–12667, 2017.

[10] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*, pages 2575–2582. IEEE, November 2018.

[11] Maxime Gueriau and Ivana Dusparic. Quantifying the impact of connected and autonomous vehicles on traffic efficiency and safety in mixed traffic. In *23rd IEEE International Conference on Intelligent Transportation Systems*, 2020.