Data Sharing in Virtual Edge Computing using Coded Caching

Gurjashan Singh Pannu*, Seyhan Ucar†, Takamasa Higuchi†, Onur Altintas† and Falko Dressler*
*School of Electrical Engineering and Computer Science, TU Berlin, Germany
†InfoTech Labs, Toyota Motor North America R&D, CA, U.S.A.
{pannu, dressler}@ccs-labs.org,
{takamasa.higuchi, seyhan.ucar, onur.altintas}@toyota.com

Abstract—**Multi-access edge computing (MEC) has been identified as a powerful concept for offloading computational tasks and for storing popular data in close proximity of end users; avoiding frequent communication to a back-end cloud server. In the context of vehicular applications, similar functionality can be provided by vehicles collaboratively offering storage and computational resources on-board, i.e., a virtual MEC concept. Data management in a virtual edge is particularly challenging due to the high degree of mobility. Coded caching is a concept to store data on distributed systems in form of fragments. When needed, these fragments are transmitted to the requesting node in a coded form so that the total number of transmissions is reduced (i.e., optimizing for reduced download times and reduced resource utilization). In this paper, we introduce a new protocol for data sharing among vehicles participating in virtual edge computing using coded caching. Our results show that coded caching improve the efficiency of data sharing by up to 50% in a virtual edge computing environment.**

I. Introduction

Connected car applications like fleet management, in-car entertainment systems, emergency calling, and navigation have been identified as the fastest growing application type for the upcoming years [1]. These applications require a reliable connectivity among themselves or to back-end cloud systems – often referred to as vehicle-to-everything (V2X) communication [2]. Some of these applications are capable of generating vast amounts of data which needs to be processed, and can also download large volume of data from the back-end cloud systems. In an ideal world, these applications can benefit from the multi-access edge computing (MEC) architecture [3], [4] in terms of offloading compute tasks or reduced latency due to popular data cached at MEC servers. However, the installation of MEC infrastructure at a certain geographic location depends on several factors like cost of running the infrastructure and revenue generated. Despite research on finding optimal locations for MEC servers [5]–[7], the actual deployment of MEC is still very limited.

Meanwhile, the concept of vehicular clouds evolved to provide edge computing services [8], [9]. Conceptually, this is now known as virtual edge [10]. A virtual edge can be realized by a group of vehicles collaboratively offering their on-board resources like computational power and storage. In locations where edge computing infrastructure is limited or unavailable, the vehicular applications can still benefit from the virtual edge created by the vehicles themselves.

Virtual edge computing comes with challenges of its own. Its member nodes, i.e., vehicles are highly mobile. The number of participating nodes could vary tremendously due to their frequent join and leave operations in a virtual edge. Furthermore, the time duration of their participation in a virtual edge is also variable. Consequently, the distributed resource pool in a virtual edge always keeps on changing. Under such constraints, maintaining data in the mobile nodes of a virtual edge, as well as accessing the data is very challenging and an utmost important task for the proper functioning of a virtual edge. Pannu et al. [11] proposed a novel protocol to maintain the data in a virtual edge. According to the protocol the vehicles may prefetch data from a virtual edge along its route even before joining it. The data may be prefetched from the vehicles which are leaving the virtual edge. This helps prevent the data from getting lost, as the vehicle which is soon joining a virtual edge bring back the data belonging to it.

To support proper functioning of the edge services, vehicles in a virtual edge may try to access different data contents which are cached by other members [12]. In this paper, we address the question of how a car can efficiently access data stored in the virtual edge. A naïve solution to tackle the data access problem is to have a requesting vehicle send a control message which includes a list of data content IDs that it wants to access. The vehicles which have the requested data contents in their cache then transmit the data to the requesting vehicle. If a number of requests originate at the same time, the wireless channel can easily be congested. We propose an optimized approach using coded caching. The main idea of coded caching [13] is to use the information about data contents available in the local cache of users, and the current on-going requests to broadcast multiple data contents coded together as a single transmission. Coded caching assumes the network to support multicast or broadcast communication, which is the case for V2X communication technologies such as ITS-G5 or C-V2X [2]. If the user has one part of the coded data, the other part can be decoded. As the data transferred is a coding of multiple data contents, several users can decode different contents from it in a single transmission, thus helping reduce the channel congestion.

The scenario of virtual edge computing is different from a typical coded-caching scenario. In our case, the vehicles in a virtual edge are the users having partial data contents in the local cache, as well as the servers offering the data to other virtual edge members. In addition, the topology of the system is highly dynamic, which constantly changes the cached data among the users. Our protocol is inspired by the coded caching technique and builds upon the existing virtual edge related protocol [14] aiming to maintain data within a virtual edge. In short, virtual edge member vehicles exchange beacons which contain information about the data which they have, as well as the missing data which they need to access. Based on this information, each member vehicle computes possible combinations of data chunks which could be coded together and transmitted in order to serve multiple data access requests with a single transmission. To the best of our knowledge, this is the first approach of using coded caching in the scope of virtual edge computing. Our results show that using our protocol, data present in a virtual edge can be accessed more efficiently by virtual edge members.

Our contributions can be summarized as follows:

- We discuss the concept of coded caching in the scope of virtual edge computing;
- We present a new protocol that allows vehicles to access distributed data cached in a virtual edge using coded caching concepts; and
- We evaluate the performance of the designed protocol using the number of transmissions and the data access latency as key metrics.

II. Related Work

Multi-access edge computing [3], [15] is one of the key network architecture concepts in 5G/6G networks. In brief, the idea is to deploy computational, storage infrastructure in close vicinity to end users. Being close to the users, the user can offload computational tasks [16], as well as requesting popular contents with minimal delays [17]. Particularly for vehicular networks, Coll-Perales et al. [7] investigated the performance for estimated MEC deployments. Chen et al. [12] discuss caching data at the edge based on geographic location, and data popularity.

In the absence of physical edge infrastructure, virtual edge can offer edge services virtually [8], [10]. Virtual edge is a small group of vehicles cooperating with each other and offering their computational and storage resources on-board in a distributed fashion. It can be stationary or mobile. Stationary virtual edge is usually setup at intersections, as it is in line-of-sight of many streets, and more vehicles can participate. Higuchi et al. [18] studied the feasibility using vehicular mobility traces for the city of Luxembourg. In order to provide edge services virtually, efficient data handling is crucial as vehicles join and leave the virtual edge very frequently [14].

Efficient transfer of partial data contents has been well researched in the information theory. Maddah-Ali and Niesen [13] proposed the coded caching scheme where users prefetch selected data contents during low network traffic hours. The server has information about the pre-cached contents in the local user cache. Using this information, the server encodes multiple data contents and broadcasts it to the users, which can decode the desired data using their data from the local cache. It has been shown that the benefits of coded caching are manifold [19]–[22]. On one hand, there is local gain thanks to the data contents locally cached. On the other hand, there is also global gain attributed to the encoded broadcasts for data delivery.

The initial proposal of coded caching was a centralized approach, which needed a central server to assist the careful placement of data contents in the user cache [13]. Maddah-Ali and Niesen [19] showed that it is even possible to fill up the user cache in a decentralized manner without any assistance of a central server, also showing performance of the system quite close to optimal results. Niesen and Maddah-Ali [20] considered non-uniform popularity of the data being requested by users, and grouped the data with similar priorities for filling up the user cache. Their results showed better performance than the highest popularity first (offline equivalent of least frequently used scheme) uncoded scheme for efficient data delivery.

One of the most challenging problems in coded caching is subpacketization. Generally, each data content is split into smaller parts. The users prefetch only a fraction of these parts into their local cache. As the number of users participating in the coded caching environment increases, the data needs to be split into further smaller parts in order to get the gains from coded transmissions. This makes the number of parts to increase exponentially. In our scenario, decentralized caching becomes particularly challenging as users can join or leave a virtual edge at any time. The users are not expected to know about the contents cached by other users as the server coordinates the cache placement for future coded transmissions. These problems have recently been addressed by Bayat et al. [22]. They extended the work in [13] to a two hop wired-wireless networks including one server connected via fronthaul links to a layer of access points and base stations which further communicate with the end-users wirelessly. Their approach tackles asynchronous user stream sessions, subpacketization, user mobility, scalability to large cellular networks with many users. These latest advancements make coded caching a potential technique for making virtual edge system operate more efficiently. To the best of our knowledge, coded caching techniques have not been applied to the virtual edge computing till date.

III. Coded Caching – A Primer

Coded caching was introduced by Maddah-Ali and Niesen [13] to reduce the network load and to improve latency problems over a shared wireless medium. The scheme consists of two steps:

- partial content caching in the local cache of end-users and
- coded multicasts to serve multiple unique data contents requests by different end-users.

The partial data contents are cached when network load is low; and coded multicasts help saving bandwidth during high network load.

Table I
CODED CACHING PERFORMANCE DEPENDING ON REQUESTS, CACHED CONTENTS, DATA TRANSFER NECESSARY FROM THE REMOTE SERVER.

Request by V_1	V_1 cache content	Request by V_2	V_2 cache content	TX	Channel load
A	-	A	-	A, A (unicast)	2
A	-	B	-	A, B (unicast)	2
B	-	A	-	B, A (unicast)	2
B	-	B	-	B, B (unicast)	2
Average load					2
A	A	A	B	A (unicast)	1
A	A	B	B	-	0
B	A	A	B	B, A (unicast)	2
B	A	B	B	B (unicast)	1
Average load					1
A	a_1, b_1	A	a_2, b_2	$a_1 \oplus a_2$ (multicast)	0.5
A	a_1, b_1	B	a_2, b_2	$a_2 \oplus b_1$ (multicast)	0.5
B	a_1, b_1	A	a_2, b_2	$b_2 \oplus a_1$ (multicast)	0.5
B	a_1, b_1	B	a_2, b_2	$b_2 \oplus b_1$ (multicast)	0.5
Average load					0.5

To understand this concept, let us consider the simple example shown in Figure 1. We have two users (vehicles) V_1 and V_2 and two data contents A and B of the same size at a remote server. Both data contents are split into two equal parts, i.e., A into a_1 and a_2, and B into b_1 and b_2. In the first phase of coded caching, partial data contents are fetched by the users when the network is not congested. Consider both vehicles have a cache capacity equal to the size of one data content. Vehicle V_1 fetches partial data contents a_1 and b_1, while vehicle V_2 fetches partial data contents a_2 and b_2. If vehicle V_1 requests A, it is missing only a_2 as a_1 is already present in the cache. If it requests B, it is again missing only a half of the data content b_2 as b_1 is present in the cache. Similarly, if V_2 requests A, the missing data is a_1 and if it requests B, the missing data is b_1.

Consider V_1 requires A and V_2 requires B. Based on the partial data cached by the users, the server encodes a message containing $a2$ and $b1$ by bitwise XOR. The resulting $X = a_2 \oplus b_1$ is transmitted by the server as a multicast to both vehicles V_1 and V_2. On successful reception of X, V_1 is able to decode missing content a_2 by using b_1 from its cache as $a_2 = X \oplus b_1$. Similarly, V_2 gets missing b_1 by using a_2 from the cache. As the transmitted data is bitwise XOR of the two partial data contents, the size of the result remains equal to the size of a partial data content. Additionally, two or more requests can be served by a single transmission (depending on the requested data and the partial data present in the user cache).

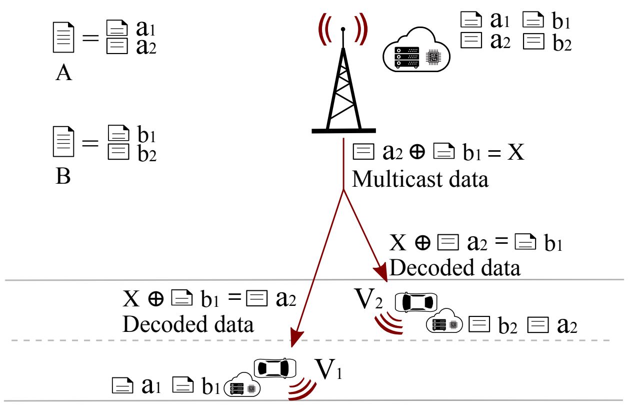


Figure 1. Example setting with two data contents and two vehicles, which do not yet cache any data content.

To show the benefits of coded caching, Table I lists different combinations of requests which can originate from vehicles V_1 and V_2. It covers three cases:

- when vehicles do not cache any data and requests are served as a unicast from the remote server,
- when vehicles cache a complete data content and requests are served as a unicast from the remote server, and
- when vehicles cache partial data contents and requests are served as an encoded multicast of missing partial data contents.

In the first case, vehicles do not cache any data. So, every data request needs a subsequent data transfer from the server. In this case, average load on the shared wireless medium is 2 data transfers. In the second case, vehicles pre-fetch a random data content into their local cache. When the requested data is already in the local cache, there is no need for data transfer over the wireless medium. On average, there is 1 data transfer necessary. The prefetching gain is generally referred to as *local gain*. In the third case, the vehicles do prefetch partial data contents. Now, the average load on the wireless channel becomes 0.5 data transfers. As the receiving vehicles already have a part of the encoded data, they XOR their locally available data content part with the received encoded data content part to get the missing data content part. The further gain to reduce the load on the shared channel comes from the data available at cache of other users and coded multicasts.

This gain is referred to as *global caching gain*. In order to maximize the global gain, the prefetching the cache becomes very critical as the server has to decide which partial data contents can be encoded together to fulfill multiple requests.

To generalize, suppose there are K vehicles, N data contents, and M per-vehicle cache slots. The local gain can be obtained as

$$K \times (1 - M/N). \tag{1}$$

The global gain can be calculated as

$$\frac{1}{1 + K \times M/N}. \tag{2}$$

Thus, the overall gain through coded caching is

$$K \times (1 - M/N) \times \frac{1}{1 + K \times M/N}. \tag{3}$$

The normalized aggregate (global) size of cache can be quantified as

$$t = KM/N. \tag{4}$$

Obviously, coded caching is a promising approach to transfer data efficiently during the peak network traffic hours.

IV. Data Downloading in a Virtual Edge

In this paper, our focus is on a virtual edge formed at an intersection. The member nodes of the virtual edge are vehicles which are assumed to be equipped with a GPS device to know their current location and the route they are following. We also assume that the vehicles have information about the location of virtual edges established along their route [14]. Each virtual edge has several data contents distributed among different member vehicles.

As a novel concept, these data contents are fragmented into smaller parts, and it is possible that different fragments of a data content are present at different member vehicles. The scenario in virtual edge is appropriate for benefiting from coded caching scheme, where partial data contents are available in the local cache of users. Generally, we try to minimize the data transfers from the backend remote servers. If requests could be served by other virtual edge members, then the data transfer takes place among the members locally without relying on the backend server.

A. An Example Scenario

Figure 2 shows an example virtual edge set up at an intersection. Vehicle A has a partial data content Y_2 in its cache and requests partial data content X_1. Vehicle B has partial data content X_1 in its cache, and requests partial data content Y_2. Vehicle C has both data contents X_1 and Y_2 available in its cache.

The requests of A and B can be served individually, i.e., A sending partial data content Y_2 to B, and B sending partial data content X_1 to A. Alternatively, C can send X_1 to A and Y_2 to B. Using coded caching, however, C now encodes X_1 and Y_2 and broadcasts $X_1 \oplus Y_2$. Vehicle A decodes the received data to retrieve $X_1 = (X_1 \oplus Y_2) \oplus Y_2$ and vehicle B decodes the received data to retrieve $Y_2 = (X_1 \oplus Y_2) \oplus X_1$.

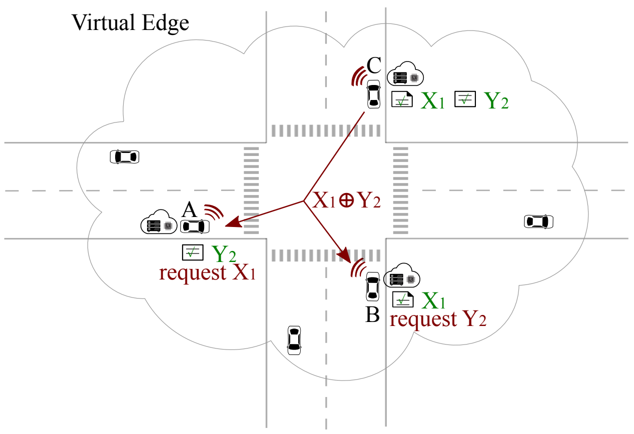
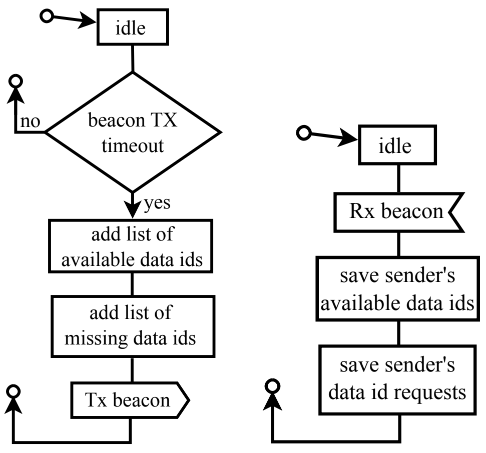


Figure 2. Example scenario showing a virtual edge at an intersection. Vehicle A needs to access data fragment X_1, and has data fragment Y_2 available in its cache. Vehicle B has X_1 in its cache, but needs to access Y_2. If vehicle C has both X_1 and Y_2 in cache. C broadcasts a bitwise XOR of X_1 and Y_2, both vehicles A and B can decode the data fragments they need using the data from their local cache and the received encoded data.


idle
no
beacon TX timeout
yes
add list of available data ids
add list of missing data ids
Tx beacon
idle
Rx beacon
save sender's available data ids
save sender's data id requests

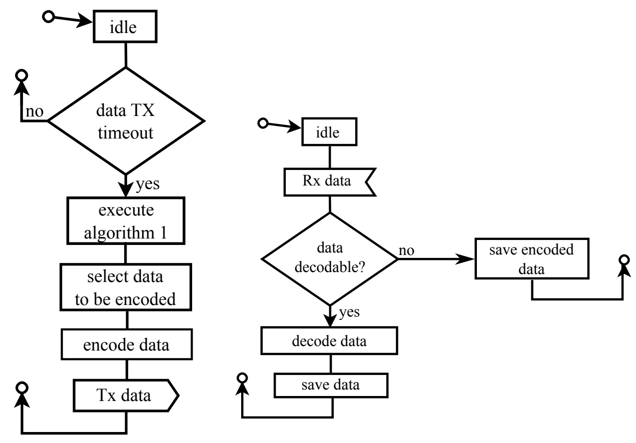
(a) Control beacon TX procedure (b) Control beacon RX procedure

Figure 3. Transmission and reception procedure of control beacons.

B. Coded Caching Protocol

Each vehicle maintains a virtual edge management database. It contains information about all the other virtual edge members. Our coded caching protocol defines two types of data exchange among the virtual edge member vehicles, i.e., control beacons and encoded data. Control beacons are periodically sent by each vehicle. They include information such as cached data fragments and missing data fragments. The procedures to send and receive control beacons are shown in Figure 3. Each vehicle stores a list of available and missing data contents per sender in its virtual edge management database to prepare for the data exchange.

The procedures for data transmission to serve requests is shown in Figure 4a. The vehicles need to know which data contents can be encoded together for serving multiple missing data requests in a single data transmission. To identify the



(a) Data TX procedure (b) Data RX procedure

Figure 4. Transmission and reception procedure of encoded data.

possible encoding combinations, each vehicle periodically executes Algorithm 1.

The input to the algorithm is the list of data contents owned by the vehicle, the list of data contents owned by other vehicles, as well as their data requests. All of the information is obtainable from the virtual edge management database. Line 2 in the algorithm implies pre-calculation of all the possible data duplets (k_i, k_j) from the list of data contents available in the cache. In line 4 to 10, the vehicle aims to find those duplets such that each one of them is available in the cache of two different vehicles, and each vehicle requests the other data from the duplet. In line 11, the algorithm returns a list of all possible data duplets which can be encoded and transmitted by the vehicle to serve data requests.

Once the vehicle has prepared the list, the next step is to select one of the duplets for encoding. For selection, there can be several heuristics which can be applied, e.g., time past since the data request, or number of vehicles requesting the same

Algorithm 1 Finding candidate data fragments to be encoded

```
Input: $\mathbb{V}$, set of virtual edge member vehicles other than me
Input: $\mathbb{D}_v \forall v \in \mathbb{V}$, set of data locally accessible by vehicle $v$
Input: $\mathbb{R}_v \forall v \in \mathbb{V}$, set of data requested by vehicle $v$
Input: $\mathbb{K}$, set of data in my local cache
 1: $\mathbb{E} \leftarrow \phi$, empty result set to contain encoding duplets
 2: $\mathbb{X} \leftarrow \{(k_i, k_j) \mid k_i, k_j \in \mathbb{K}\}$, all possible duplets of
    elements of $\mathbb{K}$
 3: while $\mathbb{X} \neq \phi$ do
 4:     if $\exists\, x_i, x_j \in \mathbb{X} \mid x_i \in D_{v_1} \wedge x_i \in R_{v_2} \wedge x_j \in D_{v_2} \wedge x_j \in$
        $R_{v_1}$ then
 5:         $\mathbb{E} \leftarrow \{(x_i \oplus x_j)\} \bigcup \mathbb{E}$
 6:         $\mathbb{X} \leftarrow \mathbb{X} \setminus \{(x_i, x_j)\}$
 7:     else
 8:         break
 9:     end if
10: end while
11: return $\mathbb{E}$
```

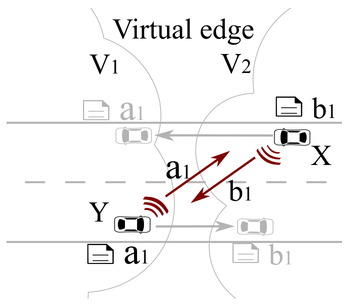


Figure 5. The vehicle X prefetches a_1 from the vehicle Y which belongs to V_1 even before joining virtual edge V_1. Similarly, Y prefetches b_1 which belongs to V_2 and joins V_2 to prevent b_1 from getting lost in V_2.

data. In our study, we take an average of the time since the data requests were made for both data contents in a duplet. The duplet which has the maximum average is selected for transmission to avoid any data request from starving. If multiple duplets have the same average time since request, the vehicle selects a random duplet among them to break a tie.

Figure 4b shows the procedure followed by a vehicle on receiving an encoded data. If the received data is decodable, i.e., one of the data contents encoded in the received data is present in the cache, then the vehicle decodes the message. If the data is not decodable, the vehicle saves the encoded data in cache. This encoded data can later be used to decode other data contents if necessary.

As vehicles add received data contents in their cache, they update the list of required and available data contents. This updated list is transmitted again in their subsequent control beacons. As there are continuous updates in the cache, the requests which have already been served are removed from the virtual edge management table to avoid repetitive transmission of the same data contents.

C. Combination with Data Recovery Techniques

Vehicles in a virtual edge may join and leave the system frequently. If the vehicle leaving the virtual edge is having the last copy of a data content available in the virtual edge, the data is lost. As shown in Figure 5 the lost data can be recovered by allowing the neighboring virtual edge vehicles to cooperate with each other [14].

Based on the knowledge of the trajectory being followed by the vehicles, they prefetch data belonging to the virtual edge which they will be joining in the near future. The prefetching is done from the vehicles which have recently left that virtual edge, or will be leaving soon. As a result, the data which might get lost due to the leaving vehicles is brought back into the virtual edge by the vehicles which have recently joined or will join soon.

The design of our protocol to access data cached by other virtual edge members is very similar to the data recovery protocol so that the two protocols can operate in combination for the efficient virtual edge operations.

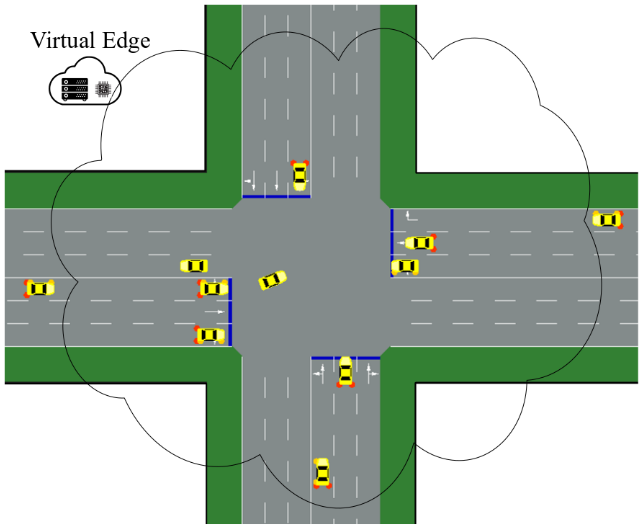


Figure 6. Screenshot of the SUMO scenario showing vehicles moving close to the intersection. A virtual edge is set up around the intersection with a radius of 200 m.

V. Performance Evaluation

To evaluate the performance of our coded caching-based data management protocol for virtual edge computing, we used vehicular networking simulation toolkit Veins [23], which makes use of the road traffic simulator SUMO [1] and the network simulator OMNeT++ [2]. For performance comparisons, we compared the coded approach with a baseline as follows:

- *No coding:* the vehicles do not encode the data to serve multiple requests at the same time. Data to be transmitted is selected based on the time since the first request for the data content. The longest waiting time gets the highest priority.
- *Coded caching:* two data contents are encoded together following Algorithm 1. Protocol details have been described in Section IV-B.

As the vehicles are moving, there are frequent vehicle join and leave operations in the virtual edge, which may lead to data loss due to the leaving vehicles. This may negatively impact the performance. Therefore, we also evaluate the coded caching protocol in combination with our previous work helping to recover data by exchanging data between leaving and joining vehicles [14].

A. Simulation Setup

We configured a virtual edge at an intersection with a radius of 200 m as shown in Figure 6. To understand the performance of accessing data under different conditions, we investigated different configurations. The number of data contents in a virtual edge were configured to be 10, 20, 30, 40 and 50. Size of each data content is configured to be 10 kByte. Each of these data contents was fragmented into blocks of 1 kByte. In coded caching, the cache placement phase is very critical as the encoding of data contents for mulicast depends on the data contents available in the cache. For the cache placement phase, we use a redundancy factor, i.e., the number of vehicles that initially cache each data fragment, set to 2, 3 and 5. All relevant simulation parameters are summarized in Table II.

[1]https://www.eclipse.org/sumo/

[2]https://omnetpp.org/ for V2X communication

Table II
Simulation parameters.

Parameter	Value
Channel	5.89 GHz
Transmission power	20 mW
Bandwidth	10 MHz
Data rate	6 Mbit/s
Building size	400 m × 400 m
Virtual edge radius	200 m
Manhattan Grid	3 × 3
Number of data contents per virtual edge	10, 20, 30, 40 and 50
Data size	10 kByte
Fragments per data	10
Initial data fragment redundancy	2, 3 and 5
Control beacon interval	1 s
Data transfer interval	1 s
Vehicle density (avg. number in a virtual edge)	10, 20 and 30
Repetitions per configuration	5
Simulation duration	1200 s

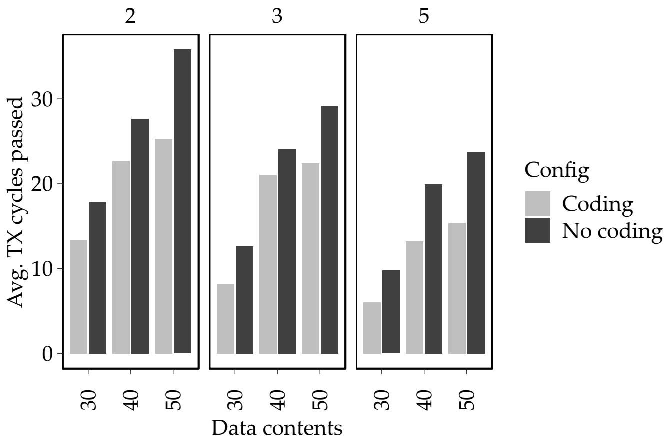


Figure 7. Average number of data transmit cycles (cf. Figure 4a) passed until a complete data content was available in the cache of a vehicle. Data is shown for a vehicle density of 20 vehicles in the virtual edge. The labels on top of facets represent the data redundancy factor.

B. Data Access Time

One of the most important metric to measure the performance of the designed protocol is data access time. To quantify this metric, we record the number of data transmit cycles (cf. Figure 4a) passed until a vehicle was able to access all fragments of a data content. Figure 7 shows the average number of such transmit cycles for which a vehicle had to wait until it was able to receive all fragments of the requested data. The labels on the top of facets represent the fragment redundancy factor, i.e., each data fragment is cached by how many vehicles initially. As clearly seen, the coded caching approach significantly reduces the data access time. The main

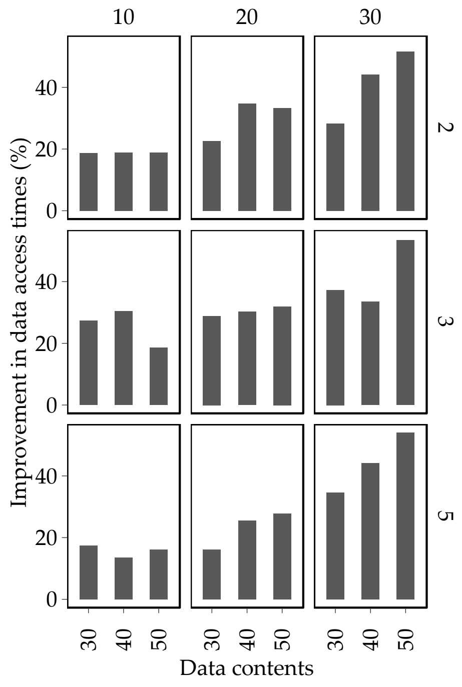


Figure 8. Improvement in the data access times when using the data recovery algorithm presented in [14]. The labels on top of facets indicate the number of vehicles in the virtual edge, while the numbers on the right represent the data redundancy factor.

reason for this effect is that the vehicles are able to serve multiple data requests with a single transmission.

Another pattern that we observe from this plot is that as redundancy factor increases, the access time decreases. We observe the decrease in both cases, i.e., with and without coding. However, the improvement is much more prominent when the coded data are exchanged. When a particular data fragment is available in the cache of multiple vehicles, the possible data encoding combinations increase. Thus, more data requests can be served by the vehicles. Although it increases the computational load to solve a larger combinatorial problem, some promising solutions have been proposed in the literature to reduce the complexity [24].

Figure 8 shows the improvements in the data access times when we use the designed protocol along with data recovery algorithm proposed for virtual edge (cf. Section IV-C). The labels on the top represent average vehicles present in the virtual edge, while the labels on the side represent data redundancy factor. As the data is also being recovered continuously irrespective of the mobility of the vehicles, the vehicles continue to serve the requests of other member vehicles. The data access times using using coded caching can achieve an improvement of up to 50 %.

C. Data Access Success

In our performance evaluation, we consider data requests to be successful only when all of the data fragments for a certain request are received. Even a single missing fragment results in a failure. To obtain deeper insights on the received

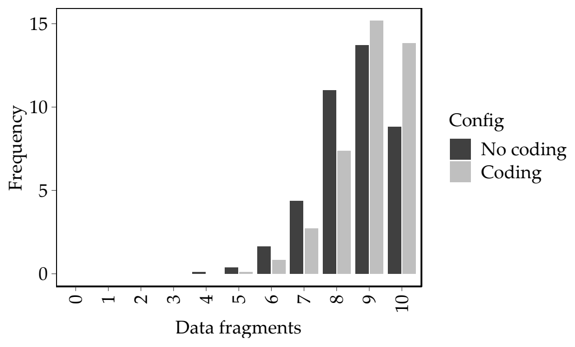


Figure 9. Histogram of successfully received data fragments (without the additional data recovery algorithm). We only show data for a redundancy factor of 5 and a vehicle density of 20 vehicles in the virtual edge.

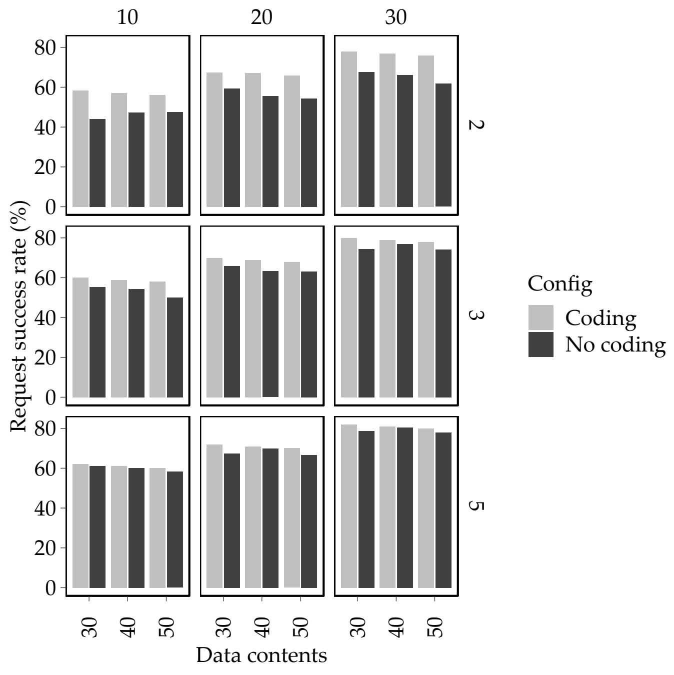


Figure 10. Successfully served data requests when using the data recovery algorithm presented in [14]. The labels on top of facets indicate the average number of vehicles in the virtual edge, while the numbers on the right represent the data redundancy factor.

fragments, we also looked at the number of fragments received by vehicles.

Figure 9 shows a histogram over the number of fragments received successfully for a data content (without the additional data recovery algorithm). As similar trends were observed in all configurations, we only present results for a data redundancy factor of 5 and a vehicle density of 20 vehicles in virtual edge. The bar corresponding 10 data fragments shows the completely successful data requests. We can observe that we had more successful data requests when the data is transferred using the coding approach. An interesting observation is that all vehicles received at least 4 fragments of their requested data. In case of coded caching, the number is significantly shifted to a larger number of successfully received contents.

When adding the data recovery algorithm, we expect an even higher number of successful requests. The results are shown in Figure 10. An interesting observation here is that for a lower data redundancy factor, the difference between successfully served requests with coding compared to no coding is higher. Thanks to the data recovery algorithm running concurrently with the data accessing protocol, increasing the redundancy of data improves the ability of vehicles to retain (or recover) a larger number of data contents in the virtual edge. As a result, the difference between overall successful data requests in the case of no coding and coding approaches decreases. However, the performance of the coding approach still remains better because of the efficient use of the shared wireless medium.

Comparing Figure 8 and Figure 10, our protocol helps in accessing the data available in the cache of other virtual edge members. Even in scenarios where vehicles have fewer copies, it is beneficial to transfer data using the coding technique.

VI. Conclusion

In this paper, we presented a coded caching-based protocol designed specifically for a virtual edge and it aims to allow vehicles to access distributed data from other member vehicles efficiently. Coded caching has first been suggested for efficient multicast-based communication from a single to distributed clients. We explored the capabilities of coded caching in this vehicular networking scenario, in which wireless communication inherently supports broadcast or multicast communication. According to our protocol, vehicles use the information about data contents in their local cache, as well as missing and cached data contents of other vehicles to compute possible data encoding combinations to fulfill the data requests of other vehicles. As the data is transmitted in an encoded form, a single data transfer may serve two data requests, thus saving the channel bandwidth.

We evaluated the performance of our protocol in a simulation scenario. We studied different performance metrics like data access time, total transmissions by a vehicle, and successfully served data requests. The results show that using our protocol based on coded caching, vehicles can access data up to 50 % faster while consuming less bandwidth on the shared wireless medium. Thus, internal operations of a virtual edge which involve data sharing among its members can benefit significantly using our designed protocol.

References

[1] "Cisco Annual Internet Report (2018–2023)," Cisco, White paper, Mar. 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[2] C. Sommer and F. Dressler, *Vehicular Networking*. Cambridge University Press, 2014.

[3] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, Mar. 2017.

[4] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A Survey on Mobile Edge Networks: Convergence of Computing, Caching and Communications," *IEEE Access*, vol. 5, pp. 6757–6779, Jun. 2017.

[5] Y. Li and S. Wang, "An Energy-Aware Edge Server Placement Algorithm in Mobile Edge Computing," in *IEEE International Conference on Edge Computing (EDGE 2018)*, San Francisco, CA: IEEE, Jul. 2018.

[6] F. Zeng, Y. Ren, X. Deng, and W. Li, "Cost-Effective Edge Server Placement in Wireless Metropolitan Area Networks," *Sensors, Special Issue on Edge/Fog/Cloud Computing in the Internet of Things*, Dec. 2018.

[7] B. Coll-Perales, M. C. Lucas-Estan, C.-H. Wang, J. Gozalvez, T. Shimizu, S. Avedisov, M. Sepulcre, T. Higuchi, B. Cheng, A. Yamamuro, and O. Altintas, "Impact of the MEC Location in Transport Networks on the Capacity of 5G to Support V2X Services," in *16th IEEE/IFIP Conference on Wireless On demand Network Systems and Services (WONS 2021)*, Virtual Conference: IEEE, Mar. 2021.

[8] F. Dressler, G. S. Pannu, F. Hagenauer, M. Gerla, T. Higuchi, and O. Altintas, "Virtual Edge Computing Using Vehicular Micro Clouds," in *IEEE International Conference on Computing, Networking and Communications (ICNC 2019)*, Honolulu, HI: IEEE, Feb. 2019.

[9] B.-J. Qiu, C.-Y. Hsieh, J.-C. Chen, and F. Dressler, "DCOA: Double-Check Offloading Algorithm to Road-Side Unit and Vehicular Micro-Cloud in 5G Networks," in *IEEE Global Communications Conference (GLOBECOM 2020)*, Taipei, Taiwan: IEEE, Dec. 2020.

[10] F. Dressler, C. F. Chiasserini, F. H. P. Fitzek, H. Karl, R. Lo Cigno, A. Capone, C. E. Casetti, F. Malandrino, V. Mancuso, F. Klingler, and G. A. Rizzo, "V-Edge: Virtual Edge Computing as an Enabler for Novel Microservices and Cooperative Computing," *IEEE Network*, vol. 36, no. 3, pp. 24–31, May 2022.

[11] G. S. Pannu, S. Ucar, T. Higuchi, O. Altintas, and F. Dressler, "Dwell Time Estimation at Intersections for Improved Vehicular Micro Cloud Operations," *Elsevier Ad Hoc Networks*, vol. 122, p. 102 606, Nov. 2021.

[12] J. Chen, H. Wu, P. Yang, F. Lyu, and X. Shen, "Cooperative Edge Caching With Location-Based and Popular Contents for Vehicular Networks," *IEEE Transactions on Vehicular Technology (TVT)*, pp. 10 291–10 305, Sep. 2020.

[13] M. A. Maddah-Ali and U. Niesen, "Fundamental Limits of Caching," *IEEE Transactions on Information Theory*, vol. 60, no. 5, pp. 2856–2867, May 2014.

[14] G. S. Pannu, F. Hagenauer, T. Higuchi, O. Altintas, and F. Dressler, "Keeping Data Alive: Communication Across Vehicular Micro Clouds," in *20th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2019)*, Washington, D.C.: IEEE, Jun. 2019.

[15] ETSI, "Mobile Edge Computing (MEC), Framework and Reference Architecture," European Telecommunications Standards Institute, Sophia Antipolis, France, GS MEC 003 V1.1.1, Mar. 2016.

[16] H. Guo, J. Liu, and J. Lv, "Toward Intelligent Task Offloading at the Edge," *IEEE Network*, vol. 34, no. 2, pp. 128–134, Mar. 2020.

[17] M. Emara, M. C. Filippou, and D. Sabella, "MEC-Assisted End-to-End Latency Evaluations for C-V2X Communications," in *European Conference on Networks and Communications (EuCNC 2018)*, Ljubljana, Slovenia: IEEE, Jun. 2018.

[18] T. Higuchi, J. Joy, F. Dressler, M. Gerla, and O. Altintas, "On the Feasibility of Vehicular Micro Clouds," in *9th IEEE Vehicular Networking Conference (VNC 2017)*, Turin, Italy: IEEE, Nov. 2017, pp. 179–182.

[19] M. A. Maddah-Ali and U. Niesen, "Decentralized Coded Caching Attains Order-Optimal Memory-Rate Tradeoff," *IEEE/ACM Transactions on Networking (TON)*, vol. 23, no. 4, pp. 1029–1040, Aug. 2015.

[20] U. Niesen and M. A. Maddah-Ali, "Coded Caching With Nonuniform Demands," *IEEE Transactions on Information Theory*, vol. 63, no. 2, pp. 1146–1158, Feb. 2017.

[21] S. P. Shariatpanahi, S. A. Motahari, and B. H. Khalaj, "Multi-Server Coded Caching," *IEEE Transactions on Information Theory*, vol. 62, no. 12, pp. 7253–7271, Dec. 2016.

[22] M. Bayat, K. Wan, and G. Caire, "Coded Caching Over Multicast Routing Networks," *IEEE Transactions on Communications*, vol. 69, no. 6, pp. 3614–3627, Jun. 2021.

[23] C. Sommer, R. German, and F. Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," *IEEE Transactions on Mobile Computing (TMC)*, vol. 10, no. 1, pp. 3–15, Jan. 2011.

[24] M. Cheng, J. Li, X. Tang, and R. Wei, "Linear Coded Caching Scheme for Centralized Networks," *IEEE Transactions on Information Theory*, vol. 67, no. 3, pp. 1732–1742, Mar. 2021.