Edge-Assisted Camera Selection in Vehicular Networks

Ruiqi Wang and Guohong Cao
Department of Computer Science and Engineering
The Pennsylvania State University
E-mail: {ruw400, gxc27}@psu.edu

Abstract—Camera sensors have been widely used to perceive
the vehicle surrounding environments, understand the traffic con-
dition, and then help avoid traffic accidents. Since most sensors
are limited by line of sight, the perception data collected through
individual vehicle can be uploaded and shared through the edge
server. To reduce the bandwidth, storage and processing cost, we
propose an edge-assisted camera selection system that only selects
the necessary camera images to upload to the server. The selection
is based on the camera metadata which describes the coverage
of the cameras represented with GPS locations, orientations, and
field of views. Different from existing work, our metadata based
approach can detect and locate camera occlusions by leveraging
LiDAR sensors, and then precisely and quickly calculate the real
camera coverage and identify the coverage overlap. Based on the
camera metadata, we study two camera selection problems, the
Max-Coverage problem and the Min-Selection problem, and solve
them with efficient algorithms. Moreover, we propose similarity
based redundancy suppression techniques to further reduce the
bandwidth consumption which becomes significant due to vehicle
movements. Extensive evaluations demonstrate that the proposed
algorithms can effectively select cameras to maximize coverage
or minimize bandwidth consumption based on the application
requirements.

I. INTRODUCTION

Today’s advanced driver assistance systems rely on various
sensors to perceive the vehicle’s surrounding environments,
allowing them to understand traffic conditions and prevent acci-
dents. However, the perception data collected by sensors
from a single vehicle has limitations because most sensors
are restricted by line of sight. Consider the example shown
in Fig. 1, the pedestrian behind truck B is crossing the road,
while car A is passing through the intersection. Car A can
not see the pedestrian because its view is blocked by truck B
which is waiting to turn at the intersection. In this scenario,
only relying on sensors on car A is not able to detect the
crossing pedestrian, and hence resulting in an accident.

This accident can be avoided by sharing the collected traffic
information among vehicles. A specific way of information
sharing is to offload the perception data to the edge server. The
edge server collects and analyzes information from vehicles
and then builds a real-time map that tracks all objects in traffic
[1], [2]. Based on the map, the edge server can send messages
to vehicles about the environment beyond their line of sight.
Specifically, in Fig. 1, the crossing pedestrian can be observed
by the cameras on car D, which can upload the information
about the crossing pedestrian to the edge server. Then, the
edge server can send a message to A and help A avoid such
a possible accident.

There has been some research on edge-assisted information
sharing [1], [2], and they generally follow two approaches
and both face some challenges. One is to upload all camera
images to the server, but this may cause network congestion
especially when many vehicles exist in a busy intersection.
In the other approach, vehicles perform some local object
detection, and only upload detected objects to the edge server;
however, the object detection may not be accurate due to the
resource limitation on vehicles as detailed in [3]. Moreover,
only uploading detected objects may cause a lack of generality
[4], for example street reconstruction needs more details about
the street view rather than just detected objects.

To address these problems, we propose an edge-assisted
camera selection system that only selects the necessary camera
images to upload. The selection is based on the camera meta-
data which describes the coverage of the cameras represented
with GPS locations, orientations, and field of views. Similar
ideas have been studied in a different context in [5]–[7].
However, they do not consider occlusions and object (vehicle)
movements, and then the calculated camera coverage may not
be accurate when directly applied to vehicular networks. We
use the point cloud generated by LiDAR sensors to detect and
locate occlusions, which can precisely and quickly calculate
the real camera coverage under resource constraints.

Based on the camera metadata, we study two camera
selection problems. First, due to the resource constraints (e.g.,
bandwidth), the number of selected cameras should be limited.
To have a better understanding of the traffic, the area covered
by the selected cameras should be as large as possible. Hence,
we study the Max-Coverage problem, i.e., how to select a given
number of cameras to maximize the total coverage. On the other hand, cameras coverage may have overlaps, resulting in redundancy. To optimize resource utilization, including bandwidth, storage, and processing capabilities, we study the Min-Selection problem, which selects the minimum number of cameras to ensure that the total coverage is above a given requirement.

The paper has the following main contributions. First, we propose an edge-assisted camera selection system that only selects the necessary camera images based on the camera metadata. Different from existing work, our metadata based approach considers camera occlusions by leveraging LiDAR sensors. Second, we study two camera selection problems, the Max-Coverage problem and the Min-Selection problem, and solve them with efficient algorithms. Moreover, we propose similarity based redundancy suppression techniques to further reduce the bandwidth consumption which becomes significant due to vehicle movements. Third, extensive evaluations demonstrate the effectiveness of the proposed camera selection algorithms and the similarity based redundancy suppression techniques.

II. CAMERA METADATA

To accurately represent the camera coverage with metadata, we need to detect occlusions in vehicular networks. Thus, we first introduce techniques for occlusion detection and then present the details about camera metadata.

A. Occlusion Detection

The coverage of a camera can be modeled as a sector, where the center is the location of the camera, the radius is the maximum range of the camera, and the field of view (FoV) is an angle that shows how wide the camera can cover. However, in the real scenario, the camera view can be blocked by objects such as buildings or vehicles, and these blocked areas are not visible to the cameras. For example, Fig. 2a shows a scenario at an intersection. In the image, cars A, B, C and D block different part of the camera view and nothing behind them are covered. Thus, to show the real coverage of the camera, these occlusions should be detected and the blocked areas should be removed.

Local object detection. One natural solution to identify occlusion is by running object detection algorithms on the collected camera image. As shown in Fig. 2a, by detecting the vehicles, the occlusions can be detected and the blocked areas can be removed. However, most object detection models are based on deep neural networks which are computational intensive and time consuming. As illustrated in Table I, a typical object detection algorithm Faster R-CNN [8] takes more than 300 milliseconds to detect objects from a single $1242 \times 375$ image on NVIDIA Jetson TX2. With such a long delay, it is impossible to generate real-time traffic map.

As another solution, we can use light-weight object detectors such as the Single Shot Detector (SSD) [9] to detect objects. Although the light-weight detector has low delay (i.e., 43ms), it sacrifices accuracy and some important objects may be missed. As shown in Fig. 2b, by applying SSD on the camera image, two vehicles in red boxes are detected, but others are missed. Then, the area blocked by undetected vehicles will be considered as being covered by the camera, which is not correct.

LiDAR. Instead of performing inaccurate local object detection on camera images, we propose to identify occlusions with LiDAR. LiDAR sensors are widely equipped on modern vehicles that help perceive the driving environment and have $360^\circ$ FoV. LiDAR sensors emit lasers which reflect on the surface of objects, and by measuring the reflected lasers, the LiDAR sensors obtain the direction and the distance of the objects and generate point cloud of the environment around the vehicle. Therefore, points in the LiDAR data describe different objects, and they are associated with precise coordinates in the LiDAR coordinate system, which makes LiDAR sensors the best choice to locate occlusions and measure the distances between these occlusions and the vehicles.

The raw data from a LiDAR sensor contains over one million points, but only part of them are related to occlusions. To reduce the computational complexity, the following preprocessing steps are applied to remove less useful points. First, to detect occlusions that block the camera view, we only need points within the camera’s FoV. By intersecting the camera view with the point cloud, we extract points that are covered by the camera. Consider the point cloud in Fig. 3a, the red sector shows the coverage of the camera, and all points in that sector are within the camera’s FoV. Second, the point cloud in the camera’s FoV contains a huge number of points from the ground plane, colored in green in Fig. 3a. Since these ground points should not be considered as occlusions, we use the Random Sample Consensus (RANSAC) algorithm [10] to identify and remove them. Then, only points above the ground remain, and they form several independent point clusters. Each occlusion corresponds to a cluster, and the width and density of the cluster indicate the size of the occlusion. Tiny objects are narrow and have sparse points, while large objects like vehicles are wide and have dense points. Since occlusions of narrow objects like utility poles are not that important, we ignore them by removing clusters with low point density. As illustrated in Fig. 3b, after these preprocessing steps, there are still four point clusters, each of which indicates an occlusion caused by a vehicle.

To compare the performance of LiDAR with the object detection approaches (Faster R-CNN and SSD), we randomly select 1,000 camera images and point cloud data from the KITTI dataset [11], and use them to evaluate how these three approaches perform on a machine equipped with NVIDIA Jetson TX2. The result is shown in Table I, and the performance is measured in terms of running time and recall. We use the metric of recall because it shows the percent of true positive detection over all existing occlusions. Each

<table>
<thead>
<tr>
<th></th>
<th>LiDAR</th>
<th>SSD</th>
<th>Faster R-CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>77.21%</td>
<td>25.17%</td>
<td>83.16%</td>
</tr>
<tr>
<td>Time (ms)</td>
<td>12</td>
<td>43</td>
<td>318</td>
</tr>
</tbody>
</table>

TABLE I: Comparisons of occlusion detection techniques.
occlusion corresponds to a blocked area which may contain important information like pedestrians, and missing occlusions leads to false negatives which considers the blocked area as being covered and ignores the potential risk in that area. Thus, to measure the accuracy of different approaches, we prefer detecting as many occlusions as possible while maintaining the false negative as low as possible.

As can be seen from the table, Faster R-CNN achieves 83.16% recall [12], which is the highest among all these three approaches, but it has the longest running time 318 ms per image. With such a long delay, the traffic scenario may quickly out of date due to the fast moving vehicles. We do not use SSD which has the lowest recall of 25.17%. LiDAR can identify occlusions within 12 ms, which is much faster than SSD and Faster R-CNN, and it has comparable recall to Faster R-CNN. Therefore, we prefer using LiDAR to detect and locate occlusions.

B. Camera Metadata

After identifying the occlusions, we need to represent the real coverage of the camera. As shown in Fig. 3b, four occlusions split the camera coverage into 5 small sectors. Each sector is represented by a 3-tuple \((r_i, \phi_i, \theta_i)\), illustrated in Fig. 3c, where \(r_i\) denotes the radius of the sector. \(\phi_i\) indicates the orientation of each sector, and \(\theta_i\) is the central angle of the sector.

We introduce a new data structure called camera metadata to represent the true coverage of the camera. Specifically, the metadata of a camera consists of the GPS location, the camera ID, and the sectors represented by 3-tuples. For example, the camera metadata in Fig. 3c should be: \(\{L, id, (r_1, \phi_1, \theta_1), (r_2, \phi_2, \theta_2), (r_3, \phi_3, \theta_3), (r_4, \phi_4, \theta_4), (r_5, \phi_5, \theta_5)\}\).

Based on the metadata, the camera coverage can be calculated. Consider the graph in Fig. 4 which is constructed based on the metadata of the vehicle cameras. It represents the scenario shown in Fig. 2a where vehicles are represented by \(A, B, C\) and \(D\). Each vehicle has a front camera, the GPS information in the camera metadata locates the camera in the graph, and the geometric information in 3-tuples specifies the area covered by the camera. The camera coverage is represented in different colors, and the current vehicle’s camera taking the picture is represented by black. As shown in the figure, these cameras have large overlaps, and uploading all these images to the edge server could waste a large amount of bandwidth, storage and processing capability. Thus, we present two camera selection algorithms which only select the necessary camera images based on the metadata in the following two sections.

III. ACHIEVING MAXIMUM COVERAGE

In this section, we consider the scenario that the edge server has a limited bandwidth which can only support the uploading of \(k\) camera images. We formulate the Max-Coverage problem and then propose a greedy algorithm to solve it.

A. Problem Statement

**Definition 1 (Max-Coverage).** Given a set of \(n\) cameras \(C = \{c_1, c_2, \cdots, c_n\}\) and the camera metadata of each camera. Our goal is to select \(k\) cameras such that the total coverage area is maximized.

**Theorem 1.** This Max-Coverage problem is NP-hard.

**Proof.** We prove the NP-hardness of the Max-Coverage problem via a reduction from the weighted maximum coverage problem [13]. In the weighted maximum coverage problem, there is a collection of sets, and each element in a set has a weight. The problem is to select a sub-collection of \(k\) sets such that the total weight of elements in the union of the sub-collection is maximized.

For any instance of the weighted maximum coverage problem, we can construct an instance of the Max-Coverage problem. Given a set of \(n\) cameras \(C = \{c_1, c_2, \cdots, c_n\}\), the coverage of these cameras are partitioned into a set of small, non-overlapping regions, \(O = \{o_1, o_2, \cdots\}\). Each \(o_i\) is a region with a weight of its area, and each camera \(c_i\) covers multiple regions, and the coverage forms a set \(O_{c_i} \subseteq O\). The collection, \(\{O_{c_1}, O_{c_2}, \cdots, O_{c_n}\}\), constructs the set collection in the weighted maximum coverage problem. The number \(k\) in the Max-Coverage problem is the same as the \(k\) in the weighted maximum coverage problem, corresponding to the size of the sub-collection.
A solution $S$ to this instance of the Max-Coverage problem is the $k$ selected cameras that maximizes the coverage area. When regions are seen as weighted elements, $S$ maximizes the total weight of elements. Thus, $S$ is also a solution to the weighted maximum coverage problem. This completes the reduction and hence the proof. \hfill \square

B. The Max-Coverage Algorithm

Since the Max-Coverage problem is NP-hard, we propose a greedy algorithm as shown in Algorithm 1 to solve it. Starting with an empty set of selected cameras $S = \emptyset$, our algorithm selects cameras from the cameras set $C$ and adds them to $S$ round by round. In each round, the algorithm chooses the camera $c_i$ with the largest coverage and moves it to the selected cameras set $S$ from the cameras set $C$. After selecting the camera $c_i$, the area covered by this camera is removed. This algorithm repeats the above steps until $k$ cameras are selected.

For example, as shown in Fig. 5, there are three vehicles A, B and C with five cameras, $C = \{c_1, c_2, c_3, c_4, c_5\}$. Each camera has a 100° FoV and 30 meters range. Cameras $c_2$ and $c_4$ are blocked by vehicle C and camera $c_5$ is blocked by vehicle B. The coverage of these cameras is partitioned into regions, $O = \{o_1, o_2, \cdots, o_{11}\}$.

With $k = 3$, our Max-Coverage algorithm will select three cameras in following steps. In the first round, camera $c_1$ is selected and added to $S$ because it has the largest coverage. Then, the coverage of $c_1$ is removed, including the region $o_2$ which overlaps with camera $c_3$, and the coverage area of camera $c_3$ should be updated by subtracting the area of $o_2$. Since the coverage has arcs, the calculation of the exact overlap area is hard. We approximate arcs with line segments such that the coverage and overlaps can be handled as polygons. After using polygons to represent $c_1$'s coverage and $c_3$'s coverage, the Greiner-Hormann clipping algorithm [14] is applied to extract the region $o_2$, and the area of $o_2$ can be calculated in $O(m)$ time where $m$ is the number of lines on the polygon [15]. In the second round, the regions covered by the remaining cameras are as follows: $O_{c_2} = \{o_4\}$, $O_{c_3} = \{o_3\}$, $O_{c_4} = \{o_6, o_8, o_9, o_{11}\}$ and $O_{c_5} = \{o_5, o_7, o_9, o_{10}\}$. Since $c_4$ covers the largest area, it is selected and added to $S$. Then, all regions in $O_{c_4}$ are removed, the overlapped region $o_9$ is extracted, and the area is subtracted from $c_5$’s coverage. In the third round, $c_2$ is selected and the algorithm terminates.

Since we approximate cameras coverage with polygons and calculate the overlap area based on those polygons, we need to know the accuracy. To find a good tradeoff between accuracy and calculation time, we randomly sample 10 cameras on a road and calculate all overlapping areas using Greiner-Hormann clipping algorithm [14] and using the equations in [15]. We calculate the exact area by using lines to replace arcs every 0.1° which is small enough. Then, we evaluate how the degree of arcs affect the accuracy and time. We repeat the experiment 100 times on an Intel Core i7 3.80GHz machine, and the result suggests a 5° interval which has an area error of 0.05% and 1.7 milliseconds running time. Thus, in this paper, arcs are replaced by lines every 5°, for example $c_1$'s arc has a 100° FoV and it will be approximated by 20 lines.

The following theorem shows the approximation ratio of our algorithm.

**Theorem 2.** Let $A(\cdot)$ denote the coverage area of the selected cameras. Let $S_{our}$ be the set of $k$ cameras selected by our algorithm, and let $S^*$ be the $k$ cameras selected by the optimal solution. Then, $A(S_{our}) > (1 - \frac{1}{e})A(S^*)$

**Proof.** As we have proved in Theorem 1, the $k$ selected cameras construct a solution to the weighted maximum coverage problem. The total weight of the solution is maximized if and only if the coverage area of selected cameras is maximized. The solution obtained by the greedy algorithm covers at least $(1 - \frac{1}{e})$ of the optimal weight [13], hence the coverage area of selected cameras is lower bounded by $(1 - \frac{1}{e})A(S^*)$. \hfill \square

IV. ACHIEVING MIN-SELECTION

In this section, we first formulate the Min-Section problem and then propose an algorithm to solve it.

A. Problem Statement

**Definition 2 (Min-Selection).** Given a coverage requirement $T$ and a set of $n$ cameras $C = \{c_1, c_2, \cdots, c_n\}$ with their camera metadata. Our goal is to find a minimum selection of cameras whose coverage area is no less than $T$.

**Theorem 3.** The Min-Selection problem is NP-hard.

**Proof.** The hardness of the Min-Selection problem can be seen by a reduction from the set cover problem [13]. In the set cover
problem, there is a collection of sets and a universal set. The problem is to identify a minimum number of sets from the collection to cover all elements in the universal set.

For any instance of the set cover problem, we can construct an instance of the Min-Selection problem. Given \( n \) cameras \( \mathcal{C} = \{c_1, c_2, \cdots, c_n\} \), the coverage of these cameras is partitioned into non-overlapping regions \( O = \{o_1, o_2, \cdots\} \), and each camera \( c_i \) covers a set of regions, \( O_{c_i} \subseteq O \). \( \{O_{c_1}, O_{c_2}, \cdots, O_{c_n}\} \) is the collection of sets in the set cover problem. The coverage requirement \( T \) is satisfied when the area of some regions is no less than \( T \), and these regions form the universal set in the set cover problem.

A solution \( S \) to this instance of Min-Selection problem is the minimum selection of cameras whose coverage area is no less than the coverage requirement \( T \). When the coverage requirement is converted to a universal set, the solution \( S \) selects the minimum number of sets from \( \{O_{c_1}, O_{c_2}, \cdots, O_{c_n}\} \) that cover all elements in the universal set. Thus, \( S \) is also a solution to the set cover problem. This completes the reduction and hence the proof.

**B. The Min-Selection Algorithm**

Since the Min-Selection problem is NP-hard, we propose a greedy algorithm as shown in Algorithm 2 to solve it. The greedy algorithm starts with an empty set of selected cameras \( S = \emptyset \), and adds one camera each round. In each round, the algorithm selects the camera \( c_i \) with the largest area of coverage and adds it to the selected set \( S \). Then, regions that are covered by this selected camera are removed. After each round, the total coverage area of the selected cameras is calculated. If the area is larger than the coverage requirement \( T \), the algorithm terminates, and the set \( S \) is the minimum selection; otherwise, the algorithm continues.

An example can be seen in Fig. 5. Given 5 cameras \( c_1, c_2, c_3, c_4 \) and \( c_5 \) in Fig. 5, some of them should be selected to satisfy the coverage requirement. Let the coverage requirement can be satisfied by covering regions \( \{o_1, o_2, o_3, o_4, o_5, o_6, o_9, o_{10}, o_{11}\} \). Our algorithm first selects camera \( c_1 \) since it has the largest coverage area, and regions \( o_1 \) and \( o_2 \) are removed. In the second round, camera \( c_4 \) which covers the largest area \( O_{c_4} = \{o_6, o_8, o_9, o_{11}\} \) is selected, and all its coverage are removed. Then, to satisfy the coverage requirement, regions \( \{o_3, o_4\} \) should be covered. Therefore, cameras \( c_2 \) and \( c_3 \) are selected before the algorithm terminates.

**Theorem 4.** Given \( n \) cameras to be selected, let \( S_{\text{our}} \) be the set of cameras selected by our Min-Selection algorithm, and let \( S^* \) be the cameras selected by the optimal solution. Then, \( |S_{\text{our}}| \leq O(\ln n)|S^*| \)

**Proof.** Since the coverage of \( n \) cameras is partitioned into non-overlapping regions, the number of regions whose area is larger than the coverage requirement is bounded by \( O(n^2) \) [14], [16]. The minimum set of cameras that cover these regions implies the solution of the set cover problem which covers \( O(n^2) \) elements. According to the approximation ratio of greedy algorithm for set cover problem in [13], the number of cameras selected by our algorithm is upper bounded by \( O(\ln n)|S^*| \).

**V. SIMILARITY BASED REDUNDANCY SUPPRESSION**

After cameras are selected, images taken by these selected cameras should be uploaded to the edge server. Since a camera may take multiple images per second (i.e., 30 frames per second for video), transmitting all of them will consume a large amount of bandwidth. The newly captured image may be similar to the previous one because vehicles cannot move too far in a short time (i.e., 33ms) and hence the changes in the images are negligible. The similarity of the consecutive images contains significant redundancy while the redundancy provides less useful information about the traffic. Thus, we can further reduce the bandwidth consumption by not uploading these similar images, and the edge server can reuse the previous uploaded image for analysis. Specifically, whenever a new image is taken, the system determines whether the image should be uploaded or not by measuring its similarity with the previous one. If the similarity is larger than a threshold \( \alpha \), it will not be uploaded and the server should use the previous image. Otherwise, the new image is uploaded.

A natural solution to implement this idea and estimate the similarity is to use the Intersection over Union (IoU) metric which measures the overlap of the camera coverage. Let \( C_{\text{pre}} \) denote the previous camera coverage, and let \( C_{\text{new}} \) denote the new coverage of the same camera. Then, the IoU is calculated as follow, \( \text{IoU} = (C_{\text{pre}} \cap C_{\text{new}}) / (C_{\text{pre}} \cup C_{\text{new}}) \).

With the camera metadata, IoU can be easily obtained by comparing the new camera metadata with the previous one. However, IoU may fail to make correct decisions in some cases. For example, suppose \( \alpha = 0.9 \). In Fig. 6, a camera takes two images while waiting at an intersection. Fig. 6a is the first frame taken at time \( t_1 \), and Fig. 6b is the second frame taken at time \( t_2 \). If the IoU is calculated as follows, \( \text{IoU} = (C_{\text{pre}} \cap C_{\text{new}}) / (C_{\text{pre}} \cup C_{\text{new}}) \) which provides new information, and this image should be uploaded. However, as shown in Fig. 6c, the camera coverage has large overlap, and IoU is 0.96 which is larger than the \( \alpha = 0.9 \). Thus, the second frame will not be uploaded, which is a wrong decision. As another example shown in Fig. 7, two images have nearly identical views, but they have low IoU of 0.82 because the

**Algorithm 2 The Min-Selection Algorithm**

<table>
<thead>
<tr>
<th>Input:</th>
<th>Camera set ( \mathcal{S} ), coverage requirement ( T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>Selected camera set ( S )</td>
</tr>
<tr>
<td>1: while ( A(S) &lt; T ) and ( S \neq \emptyset ) do</td>
<td></td>
</tr>
<tr>
<td>2: Find a camera ( c_i ) from ( \mathcal{S} ) that covers the largest area</td>
<td></td>
</tr>
<tr>
<td>3: ( S \leftarrow S \cup c_i )</td>
<td></td>
</tr>
<tr>
<td>4: ( S \leftarrow S \setminus c_i )</td>
<td></td>
</tr>
<tr>
<td>5: Update the coverage by removing the overlap ( (\bigcup_{c_j \in S} O_{c_j}) \cap O_{c_i} )</td>
<td></td>
</tr>
<tr>
<td>6: end while</td>
<td></td>
</tr>
<tr>
<td>7: return ( S )</td>
<td></td>
</tr>
</tbody>
</table>
camera location changes. Since the IoU is lower than \( \alpha \), the image will be uploaded even though it is redundant.

To address this problem, we use perceptual hashing (pHash) [17]–[19] to measure the similarity of the two images. The pHash method first converts the camera image to a \( 32 \times 32 \) gray-scale thumbnail. Then, Discrete Cosine Transform (DCT) is applied to transform pixels in the thumbnail from the spatial domain to the frequency domain, and obtain a \( 32 \times 32 \) coefficient matrix. Let \( \mathbf{F} \) denote the coefficient matrix, and let \( \mathbf{T} \) denote the thumbnail. Each element of the coefficient matrix \( F_{x,y} \) is calculated as follow.

\[
F_{x,y} = \sum_{i=0}^{32} \sum_{j=0}^{32} I_{i,j} \cos \left( \frac{(2i + 1) \pi}{2 \times 32} x \right) \cos \left( \frac{(2j + 1) \pi}{2 \times 32} y \right)
\]

The pHash only considers the low frequency features which corresponds to the \( 8 \times 8 \) matrix from the upper left corner of the coefficient matrix, and the \( 8 \times 8 \) matrix is converted to a vector of length 64 (8\( \times \)8). Next, the pHash compares the vector with its mean value to build the 64-bit fingerprint. If the value in the vector is larger than the mean, the corresponding bit in the fingerprint is 1; otherwise, the bit is 0. Then, given two fingerprints \( \mu \) and \( \nu \), the similarity can be calculated with the Hamming distance as follows, which counts the number of bits where the corresponding values are different.

\[
\text{Similarity} = \frac{1}{64} \cdot \| \{ i : \mu_i \neq \nu_i, i = 1, 2, \ldots, 64 \} \|
\]

For the example shown in Fig. 6, the pHash similarity of the two images is 0.84, compared with \( \alpha = 0.9 \) the new image in Fig. 6b will be uploaded as expected. On the other hand, the two similar images in Fig. 7 have a higher pHash similarity of 0.98, and then the new image will not be uploaded.

## VI. PERFORMANCE EVALUATIONS

In this section, we first use a case study to show the effectiveness of the proposed solution, and then evaluate its performance with extensive simulations.

### A. Evaluation Setup

Existing datasets for autonomous driving, like KITTI [11], only collect perception data from a single vehicle. However, the evaluation of information sharing requires data from multiple vehicles simultaneously. Thus, we collect data from multiple vehicles and evaluate the performance of our system in CARLA [20], an open source simulator for autonomous driving. CARLA provide several high-definition (HD) maps for different towns and cities. Sensors data, like camera images and LiDAR point cloud, can be obtained from multiple vehicles simultaneously.

We select two intersections in CARLA and simulate dense traffic scenarios by randomly spawning vehicles. Each vehicle is equipped with a LiDAR sensor and two cameras, a front camera and a rear camera. Each LiDAR sensor keeps scanning surroundings at 30 frames per second (fps), and it generates 1,500,000 points for each frame. Each camera has a 100° FoV and a maximum range of 30 meters, and it generates 1280 \( \times \) 720 RGB images at 30 fps.

In the evaluation, we compare the performance of our solution (i.e., Max-Coverage and Min-Selection) with a random selection algorithm, which selects cameras randomly. We repeat the random selection algorithm 10 times for each simulation scenario, and the results are averaged.

### B. Case Study

We first use a case study to demonstrate the effectiveness of our solution. In this case study, the traffic of an intersection at frame \( T \) is shown in Fig. 8a. There is a red boundary showing the road area that should be covered, and only coverage within the boundary is calculated. This is because the camera coverage outside the boundary does not help obtaining the traffic information. For any vehicle camera, its coverage sector will be cut smaller due to the occlusion of the boundary, i.e., the sector outside of the boundary will be blocked. To evaluate the performance of our algorithm, we use the coverage ratio, which is the area covered by the selected cameras divided by the total area within the boundary. If there is no confusion, we will use coverage and coverage ratio interchangeably.

Within the boundary, there are 10 vehicles and some pedestrians. Since each vehicle carries a front camera and a rear camera, there are 20 cameras in total. The pedestrian marked with the yellow box is not visible to vehicle 1 since the pedestrian is in the area blocked by vehicle 2. Therefore, a good selection of the cameras should maximize the coverage and the selected cameras should cover the blocked area. In this case study, we solve the Max-Coverage problem. We assume that the bandwidth can support the uploading of \( k = 5 \) camera images, and demonstrate that our Max-Coverage algorithm selects 5 cameras which have much better coverage, i.e., a good view of the intersection area and covering the vehicles and pedestrians as many as possible.

The random selection algorithm may not perform well. For example, as shown in Fig. 8b, the blue areas represent the coverage of five randomly selected cameras, which only covers 43.1% of the area. The crossing pedestrian behind vehicle 2 is not covered by any camera in the random selections, and hence the random selection algorithm fails to eliminate the risk of potential traffic accidents. Then we show the selection result...
of our Max-Coverage algorithm in Fig. 8c where the green areas represent the coverage of the five selected cameras. The selected five cameras cover 70.8% of the road area within the red boundary, which is only 8% less than the best achievable coverage (i.e., 78.5%) of all 20 cameras. Besides, the crossing pedestrian is covered by vehicle 5’s front camera.

To better understand the effectiveness of our system, in Fig. 9, we show the front camera views of vehicles 5 and 7 selected by our algorithm. These two images cover the intersection from different aspects and provide a good view of the traffic. Based on these images, the edge server can detect the 10 pedestrians highlighted with bounding boxes. The crossing pedestrian mentioned before (in Fig. 8a), shown in the yellow bounding box, is successfully detected in vehicle 5’s front camera. Thus, the edge server can send a warning message about the crossing pedestrian to vehicle 1 and help it avoid the traffic accident. On the other hand, the cameras selected by the random selection algorithm may fail to provide useful information and uploading these images is a wast of resource.

We quantify the computational time of the Max-Coverage algorithm on a machine equipped with an Intel Core i7 3.80GHz and an NVIDIA RTX 3080. The mean computational time is 13ms, which is more than enough to support 30fps. We also use the brute force algorithm to find the optimal solution for camera selection. The complexity of the brute force algorithm is $O(n^k)$ where $n$ is the total number of cameras and $k$ is the number of selected cameras. In this case study, $n=20$, $k=5$. The computation time is 8.005 seconds, which is too long to support 30fps. Its coverage can reach 74.0%. Compared to the 70.8% coverage achieved by our Max-Coverage algorithm, this small increase of 3.2% is not significant.

**Dealing with vehicle movements:** Due to vehicle movements, our system has to update the traffic condition at a rate of 30fps. Since there is significant redundancy in consecutive images, we propose redundancy suppression techniques to further reduce the bandwidth consumption. As shown in Fig. 8c and Fig. 8d, due to the short time difference (33ms), vehicles will not move too far and hence the camera coverage is similar. By using our redundancy suppression techniques, the bandwidth consumption can be reduced by 42.0%. Specifically, the gray areas in Fig. 8d indicate the camera images that are redundant (i.e., vehicle 3’s and 5’s front cameras).

### C. Simulation Results

We conduct extensive experiments to evaluate the performance of our algorithms at a larger scale. We spawn 30 vehicles at intersections to simulate the busy traffic. Since some of the vehicles may not participate in our system due to hardware limitations or other reasons, we also evaluate how the number of participating vehicles ($p$) affects the performance.

1) **Results on Max-Coverage:** In this part, we evaluate the performance of our Max-Coverage algorithm (“Ours”) and compare it with the random selection algorithm (“Random”). It is impossible to obtain the optimal selections in practice, so we show the performance upper bound with the **Best Achievable** coverage, which is the total coverage that can be achieved by using all vehicle cameras.

We first study how the number of selected cameras ($k$) affects the coverage. Fig. 10a shows the coverage of different algorithms as a function of $k$, when there are 20 participating vehicles ($p$), the camera range $r$ is 30 meters, and the FoV is 100°. As shown in the figure, the coverage of our algorithm and the random selection algorithm increases when more cameras are selected, and our algorithm always outperforms random selection. Since each vehicle has two cameras, there are 40 cameras in total and the best achievable coverage is obtained with all these 40 cameras. When $k=2$, which means that only two cameras are selected, our algorithm achieves about 0.40 coverage, and it underperforms the best achievable which has the coverage of 0.89. As the number of selected cameras increases to 14, our algorithm can achieve similar coverage with the best achievable, but with much less amount of wireless bandwidth, i.e., transferring 14 camera images instead of 40.

Fig. 10b compares three algorithms as the number of participating vehicles changes. When $p$ increases, there are more selection choices in our algorithm and the best achievable, and
Fig. 10: Simulation results on Max-Coverage problem.

Fig. 11: Simulation results on Min-Selection problem.

Thus their coverage also increases. For example, when \( p = 20 \) (vs. \( p = 5 \)), our algorithm can select the best 8 cameras from these \( 20 \times 2 \) (vs. \( 5 \times 2 \)) cameras to achieve a coverage of 0.82 (vs. 0.68), while the best achievable will select all these \( 20 \times 2 \) (vs. \( 5 \times 2 \)) cameras to achieve a coverage of 0.89 (vs. 0.71). On the other hand, in random selection, when \( p \) increases from 5 to 10, the coverage is decreased. This can explained as follows. When \( p = 5 \), eight cameras are randomly selected from these \( 5 \times 2 \) cameras and hence the chances of selecting the cameras with better coverage is higher. When \( p = 10 \), eight cameras are randomly selected from these \( 10 \times 2 \) cameras and hence the chances of selecting the cameras with better coverage is reduced. As the number of participating vehicles continues to increase, the coverage will not change too much since chance of selecting the good cameras does not change too much.

Fig. 10c compares the three approaches when the camera range changes. As shown in the figure, the coverage increases as the camera range increases. Our algorithm always outperforms the random selection algorithm, and approaches to the best achievable as the camera range increases.

In Fig. 10d, we fix the camera range as 30 meters, and \( k = 8 \), and compare the coverage of these three approaches when the FoV changes from 80° to 120°. As shown in the figure, the coverage increases as the FoV increases. Our algorithm always outperforms random selection, and comparable to the best achievable.

2) Results on Min-Selection: In this subsection, we evaluate the performance of our Min-Selection algorithm, and compare it with the random selection algorithm, which keeps selecting cameras randomly until the coverage requirement is satisfied. The coverage requirement is represented by the coverage ratio, and the performance is measured by the number of selected cameras.

Fig. 11a illustrates the number of cameras selected to satisfy the specified coverage requirement in both algorithms, when there are 20 participating vehicles (\( p \)), the camera range \( r \) is 30 meters, and the FoV is 100°. As shown in the figure, both algorithms require more cameras to cover a larger area, and Min-Selection consistently outperforms random selection. For example, to reach 80% coverage, random selection requires 21 cameras, but Min-Selection only needs 8 cameras.

To ensure a larger coverage at the intersection, we set the coverage requirement as 80%, and study how the number of participating vehicles affects the number of selected cameras. As shown in Fig. 11b, Min-Selection generally requires much less number of cameras than random selection, except when \( p = 5 \). This is because the coverage requirement is too high and even the best achievable can not reach. As shown in Fig. 10b, even when all the \( 5 \times 2 \) cameras are selected, the coverage can only reach 0.71. As a result, our algorithm has to select all cameras, which has same result as the random selection. As the number of participating vehicles increases, e.g., \( p \geq 10 \), our algorithm can carefully select the minimum number of cameras to reach the required coverage. On the other hand, the random selection algorithm chooses cameras randomly, and its disadvantage becomes more visible when there are more participating vehicles.

Fig. 11c shows the impact of camera range. As shown in the figure, our Min-Selection always selects fewer cameras than the random selection algorithm. For example, when the camera range is 20 meters, our algorithm only requires a third of the cameras compared with random selection. As the camera range increases, each camera has a larger coverage, and less cameras are needed. As shown in Fig. 11d, with a larger FoV, the view of each camera becomes wider and the coverage increases. As a result, fewer cameras are needed to satisfy the coverage requirement when FoV increases in both algorithms, while the Min-Selection always outperforms random selection.

3) Results on Redundancy Suppression: The proposed similarity based redundancy suppression techniques only depend on the selected cameras, and then they can be applied to all selection algorithms. Here, the evaluation is based on the
Max-Coverage algorithm with $k = 8$. We also fix the camera range as 30 meters and FoV = 100°. As a “Baseline”, cameras generate images at 30 fps, and then $8 \times 30$ images are uploaded to the edge server per second.

Fig. 12a evaluates the bandwidth requirement in pHash when the vehicle speed changes. As the vehicle speed increases, the camera view changes rapidly which results in low similarity between camera images. Hence more images will be uploaded and more bandwidth is needed. We can also see that a higher threshold $\alpha$ leads to more bandwidth consumption. Since there is an obvious jump in the bandwidth when $\alpha$ increases from 0.9 to 0.95, we set the threshold $\alpha=0.9$.

We then fix the vehicle speed as 30 km/hour, which is the average speed of vehicles in cities [21]. We compare the performance of using similarity metrics, IoU and pHash, with the baseline when different number of cameras are selected. As shown in Fig. 12b, pHash significantly outperforms IoU which also outperforms the baseline. In general, pHash reduces the bandwidth usage by 80% ~ 85% compared to the baseline.

VII. RELATED WORK

Our work is related to the field of information sharing in vehicular networks [22], and there has been considerable research on this topic [23]–[27]. Qiu et al. [28] designed a system which enables sharing visual information with neighboring vehicles to see objects blocked by previous vehicles. In [29], Wang et al. proposed to broadcast objects features obtained by LiDAR to nearby vehicles to enhance their perception of the environment. In [30], the authors designed a system that allows vehicles to share perception data about the surrounding environment with neighboring vehicles, and cooperatively merge the data into a better traffic view. However, these works are all based on vehicle-to-vehicle communication.

As mobile edge computing becomes popular, there is growing amount of research on information sharing through the edge server [2], [31], [32]. LiveMap [1] utilizes the edge server to gather and analyze camera images uploaded from vehicles to build a dynamic map of the road traffic. LiveMap proposes to adaptively offload the object detection tasks to the edge server such that the bandwidth and computational resources can be saved, and the offloading is scheduled among a few vehicles based on the coverage of their cameras. However, occlusions are not considered. Similarly, in [2], Liu et al. introduced a system which shares the locations of objects in the traffic through the edge server based on the camera images from vehicles. In [4], Zhang et al. proposed an edge-assisted system that allows multiple vehicles to share the raw data from LiDAR sensors. Different from these works, our system utilizes the LiDAR data to locate occlusions and reduce the bandwidth consumption by only selecting the necessary camera images for uploading based on camera metadata.

The camera selection problem studied in this paper is related to the area coverage problem in sensor networks, and this problem has been well studied in the past years [33]–[37]. In [33], Huang et al. studied the k-coverage problem in sensor networks, and they proposed a polynomial-time algorithm to determine whether an area is k-covered. In [37], He et al. proposed approximation algorithms that select the minimum number of cameras to achieve full-view coverage such that every point in a given area is covered from 360°. However, none of them considers occlusions and addresses the challenges of vehicle and camera movements.

Our work is inspired by previous studies on using metadata to represent the camera coverage. Specifically, in SmartPhoto [5], the authors proposed to quantify the quality of crowdsourced photos based on the camera metadata, which can be used to select photos to cover a target point. Later, Wu et al. [6] extended this work to select crowdsourced photos to cover a target area under various resource constraints. In VideoMec [7], the authors leveraged metadata to select crowdsourced videos under bandwidth and energy constraints. However, none of them considers the object occlusions when using the camera metadata. In contrast, we leverage LiDAR to identify occlusions and apply the concept of metadata to select camera images in vehicular networks considering the fast movement of the vehicles.

VIII. CONCLUSION

In this paper, we proposed an edge-assistant camera selection system which only selects the necessary camera images based on the camera metadata, to reduce the bandwidth, storage and processing cost at the edge server. We use LiDAR sensors to detect and locate occlusions, and represent the real camera coverage after removing blocked areas with camera metadata. Then, based on the camera metadata, we propose efficient algorithms to solve two camera selection problems, i.e., the Max-Coverage which selects a given number of cameras to maximize the total coverage, and the Min-Selection which selects the minimum number of cameras while ensuring that the total coverage area of these selected cameras is above a given requirement. Moreover, we propose the similarity based redundancy suppression technique to further reduce the bandwidth consumption which becomes significant due to vehicle movements. Extensive evaluations demonstrate that the proposed algorithms can effectively select cameras to maximize the coverage or minimize the bandwidth consumption based on the application requirements.

IX. ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under grant number 2125208.
REFERENCES


