Bus on the edge: Continuous monitoring of traffic and infrastructure

Canbo Ye (https://orcid.org/0000-0002-8011-5881)
Christoph Mertz (https://orcid.org/0000-0001-7540-5211)
Mahadev Satyanarayanan (https://orcid.org//0000-0002-2187-2049)

FINAL RESEARCH REPORT - December 31, 2020

Contract # 69A3551747111

Contents

1. Introduction

Data is essential to maximize mobility and make the transportation system resilient. Examples are real time schedules, traffic conditions, weather conditions, or infrastructure inspection. This project is the first phase in a multi-year effort to develop an efficient data collection platform that can monitor complete routes on an hourly basis. The idea is to make use of transit buses that travel on main roads on a regular basis. They are often already equipped with cameras, GPS, and IMU and contain an electronics cabinet that houses recording equipment and the automatic annunciation system. Because of bandwidth and storage constraints, it is impractical to send all this data to the cloud. Instead, we added edge computer to such a system, i.e. we converted the recording box to a computer that can analyze the video streams live to detect relevant events and send only these detections over a cellular network to a central location. Data that is interesting but not time critical can be saved and uploaded via wifi at the end of the day. Potentially relevant data can be saved for a set amount of time locally and analyzed or uploaded upon request.

The data collected by such a system can be used for event detection, traffic modeling and infrastructure monitoring and thereby provide input for up-to-date detailed maps of roads and traffic. Up-to-date detailed maps are one essential component of autonomous driving, but they are also needed for traffic management, planning, and infrastructure maintenance. Other examples are traffic counts, counting of parked cars, observations of road construction, pothole detection, detecting landslide precursors, measuring snow cover, or observing crossing of wildlife.

The next sections give a detailed description of this BusEdge platform. In section 2, an overview of the system architecture is given. This is followed by the implementation details in section 3. Section 4 will explain how to achieve flow control with the help of the Gabriel [1] framework. The hardware description for our experiments is given in section 5. We will show an example application built upon the BusEdge platform in section 6 for demonstration. Finally, we will discuss the work that needs to be done in the next phase in section 7.

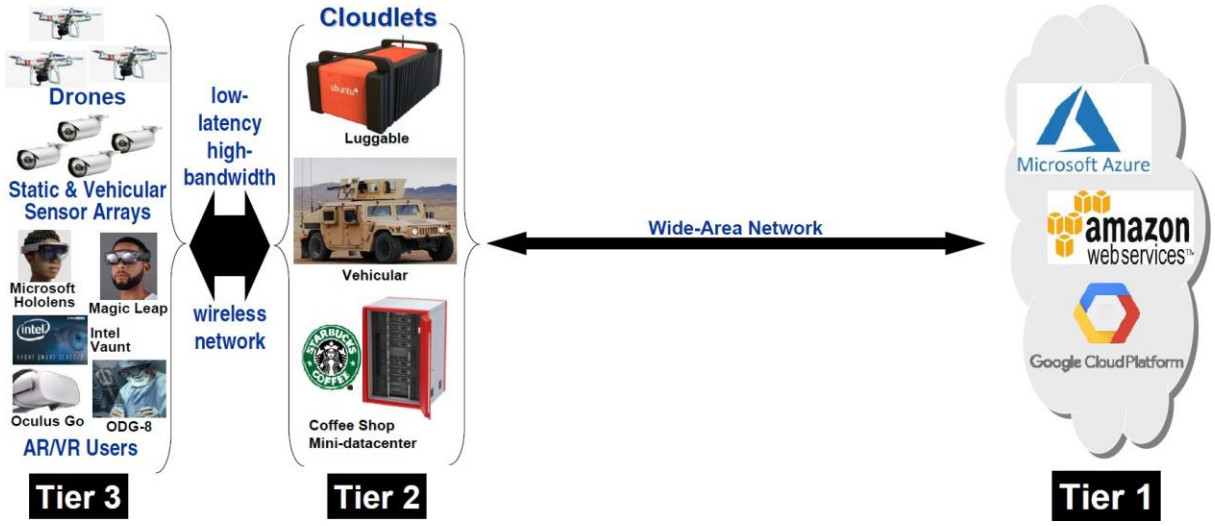

Figure 1: Tiered Model of Computing [2]

2. System Architecture

When designing the architecture of the BusEdge platform, there are several requirements that need to be considered. The first is the scalability of the system, which means that it should be straightforward to apply and extend the platform to multiple buses and different functionalities. The second point is the importance of low latency, which represents the ability to handle the most up-to-date data so that we could be responsive to different urgent events. The third is the ability to cache data locally and revisit it if needed.
Before diving into the architecture design of the BusEdge platform, it is worth mentioning the concept of *cloudlet* and the corresponding tiered model. The tiered model shown in Figure 1 represents a hierarchy of increasing physical size, compute power, energy usage, and elasticity. Tier-3 represents IoT (Internet of Things) and mobile devices, which stands in our case for the bus edge computer. Tier-2 represents cloudlet, which is a new architectural element that arises from the convergence of mobile computing and cloud computing. It is built on standard cloud technology and has similar functionality with the cloud service, but it is much closer to the associated mobile devices. The cloudlets could provide high compute power with low latency and high bandwidth due to network proximity. In our experiments, we use a cloudlet server situated on the Carnegie Mellon University campus.

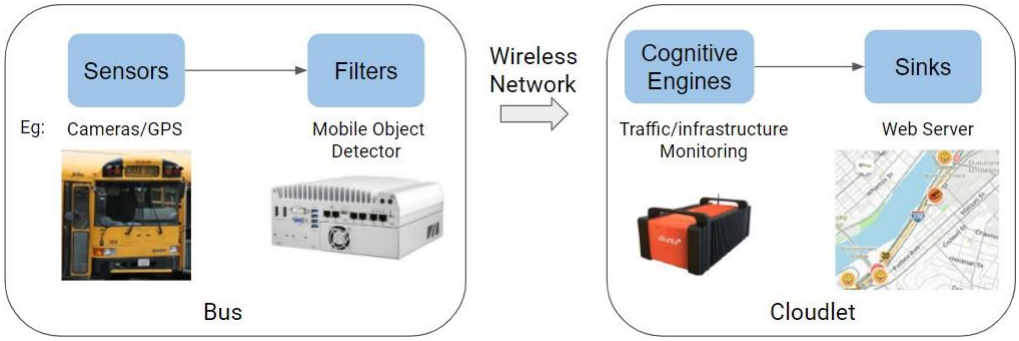

Figure 2: The major components of the BusEdge platform

As is shown in Figure 2, the BusEdge platform contains four major components: *Sensors* and *Early-Discard Filters* on the bus side as well as *Cognitive Engines* and *Sinks* on the cloudlet side.

Sensors are the data sources on the bus, which usually include video streams from multiple cameras, GPS, accelerometer and other vehicle status read from the CANBUS. First, these sensor inputs will be preprocessed and then published to different Early-Discard Filters for refinement.
Early-Discard Filters represent the data refinement components running on the edge computer. Different tasks could share the same filtering node or have different ones. Each filter is mutually independent and will send its outputs to the specific Cognitive Engine on the cloudlet for further analysis.
Cognitive Engines are the computer vision modules running on the cloudlet to handle and analyze the filtered message from bus clients. Each Cognitive Engine will register one type of filtered source with a given source name before launching. The flow control between the bus clients and the cloudlet server is managed by the Gabriel framework [1], which will be introduced in section 4.
Sinks represent the final components on the cloudlet to collect all the results from different Cognitive Engines and do the data analysis and visualization. This could be an OpenStreetMap [3] server showing all the detection results on a map or some sophisticated software or toolbox for data analytics and visualization like Tableau or Kibana.

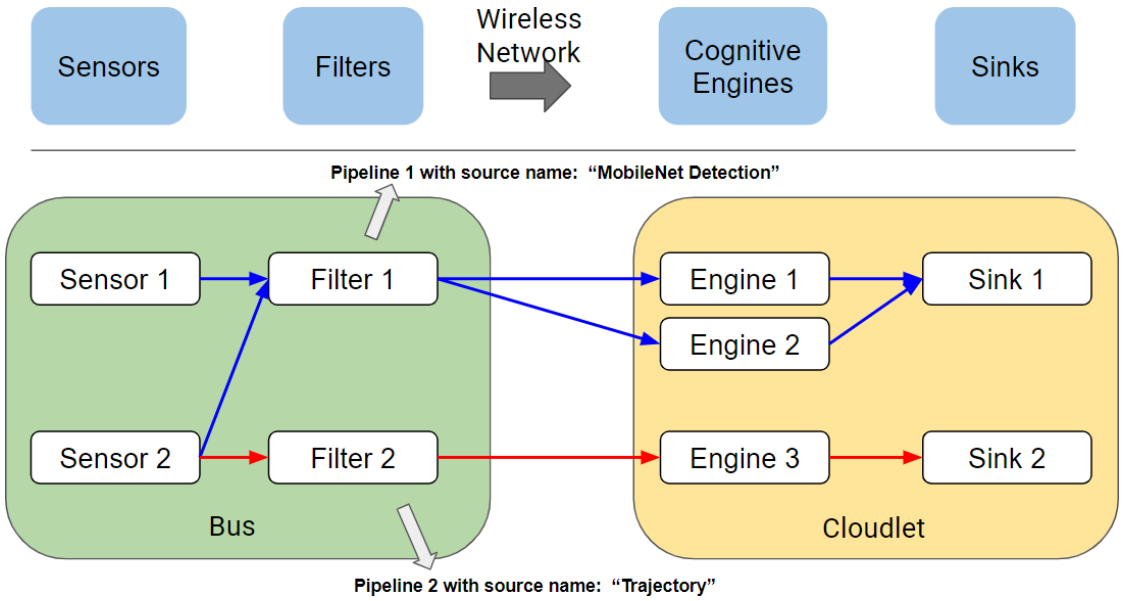

Figure 3: An example of the BusEdge pipelines

3. Implementation Details

A typical BusEdge pipeline consists of all four components mentioned above, but the details could vary from task to task. Figure 3 shows a use case where we implement two pipelines on the BusEdge platform, which are distinguished by the color of the arrows. Each pipeline should register its own source name. For one BusEdge pipeline with a given source name, there will be only one early-discard Filter. For each early-discard filter, however, data could be captured by different sensors and its output could be transmitted to one or multiple cognitive engines on the cloudlet. Besides, every message from one source should have the same type of data, and this type should not change. As is shown in Figure 3, pipeline 1 with the source name "MobileNet Detection" uses inputs from two sensors and sends filtered messages to two different cognitive engines, while pipeline 2 with the source name "Trajectory" has only one sensor input and one cognitive engine. In other words, the source name is the keyword to define a BusEdge pipeline. The motivation for this design is that we want to reuse filtered data to save bandwidth and this also allows cognitive engines to know what to expect in input frames and what results the client expects back if needed.

Modular design is another important feature of the BusEdge platform to realize scalability. The flow control of the pipeline is transparent to the developers and they only need to develop their own early-discard filter and cognitive engine. Each early-discard filter and cognitive engine could run in their own Docker container both on the bus client and the cloudlet server. They could also choose an existing filtered source if they share the same targets with some other tasks. In general, when the developer wants to deploy code onto the BusEdge platform, all that needs to be done is connecting the Docker container with the Gabriel server on the cloudlet and register an existing source or create a new early-discard filter on the bus. We isolate the functionality modules including cognitive engines and early-discard filters into different Docker containers because we want to enable rapid functionality expansion and avoid dependency conflicts.

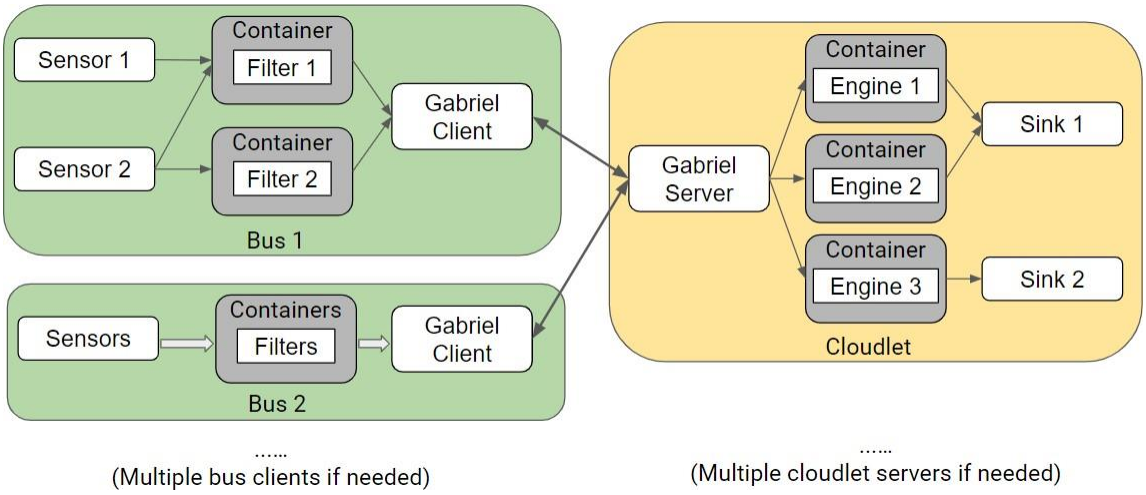

Figure 4: The system architecture details of BusEdge

Figure 4 illustrates the system architecture details of a typical BusEdge use case. We could observe some other features beyond what we have mentioned above. On the one hand, the system could be expanded to multiple bus clients if needed, which further improves the scalability of the system. On the other hand, the communication between the bus clients and cloudlet server is bidirectional, making it possible for the bus clients to react to the analysis results from the server. This also enables the update of early-discard filter during online processing. Some possible vehicle-cloudlet interaction contents are illustrated in Figure 5. Besides that, there are two components we missed in Figure 4 namely the Gabriel client and Gabriel server. They are the two major components of the Gabriel framework to realize flow control, which will be discussed in section 4.

Figure 5: Vehicle-cloudlet interactions [4]

Furthermore, the implementation details of the bus client is shown in Figure 6. We utilize ROS [5] to manage the data acquisition, distribution and storage. The advantages of using ROS are that it provides us with a series of convenient data preprocessing packages and its publisher-subscriber-based messaging protocol fits our modular architecture very well. It will be very convenient for a developer in robotics community to expand or update the system using ROS because of its popularity and wide application. Another detail to note from Figure 6 is that not

all the early-discard filters are encapsulated into the Docker containers. This is because some simple filter nodes are provided for common usages or experiments, such as sending the GPS trajectory at a given rate or sending preprocessed images at fixed intervals, which are not necessary to be isolated into containers.

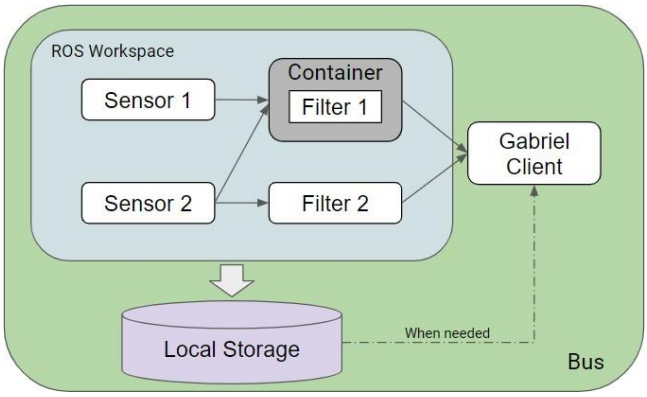

Figure 6: The bus architecture details

It is also worth noting that in addition to real-time processing, we will also store all recent sensor data locally because these raw data could be valuable for many tasks. The sensor inputs are recorded in the rosbag format. We can download these locally cached data when the bus has returned to the garage and is connected to WIFI. It is also possible to revisit a fraction of previously filtered data during online processing. A typical use case is when we deploy an incrementally learning object detector, it will be beneficial if we could reexamine the discarded piles with an updated model from time to time.

4. Flow Control Based on Gabriel

Gabriel is a flow control framework initially designed for wearable cognitive assistance using cloudlets in [1]. Two key ideas were embodied in the design of Gabriel: (a) the ability to use disparate legacy code bases to speed up the development of applications and (b) an end-to-end flow control mechanism for timely delivery of requests even during network congestion. These two ideas are exactly in line with the needs of the BusEdge platform, making it a reasonable choice to apply the flow control mechanism of Gabriel to our project. We will explain the flow control mechanism of Gabriel and also illustrate its role in the BusEdge platform.

As is shown in Figure 4, there is a Gabriel client on the bus side to gather refined data from multiple early-discard filters, and a Gabriel server on the cloudlet side to distribute messages to different cognitive engines. The wireless communication between the Gabriel client and server is achieved via the WebSocket Protocol. Gabriel will manage the flow control for every individual pipeline as is listed in Figure 3. The fundamental target of Gabriel flow control is to avoid starvation or saturation for each pipeline and enable the engines to process the most up-to-date data.

Specifically, Gabriel's flow control is based on token mechanism. When the client sends a message to the server, this consumes a token for the source that produced this message. When the first cognitive engine finishes processing this message, the client gets back the token that was consumed sending the message. This ensures that messages are sent to the server at the rate that the fastest engine consuming messages from this source can process them. After a client consumes all of its tokens for a source, it will only send a new message from this source after it receives a token back from the server. This will help to avoid pipeline saturation and guarantee low latency.

5. Hardware Descriptions

In this section, we will illustrate some information of the hardware we utilize in our project, which include the sensors and devices installed on the bus. Figure 7 shows some pictures of the hardware on the bus.

Figure 7: Some pictures of the hardware on the bus. From left to right: bus edge computer; exterior camera; interior camera; GPS and network antenna.

Bus Edge Computer is the major computing and storage device on the bus to acquire data from the sensors and carry out the first step analysis. The technical specifications of the bus edge computer are listed in the Table 1.

Brand	Safety Vision
Model	RoadRecorder 9000
CPU	Intel Core i7-8700t @ 2.40GHz
RAM	16 GB
Storage	5 TB
Power Input	9-48V DC
Dimensions	289 × 118 × 250 mm

Table 1: Technical specifications of the bus edge computer.

Cameras. We install four waterproof exterior cameras at the four corners of the bus and one interior camera behind the windshield of the bus. Table 2 and Table 3 list the technical specifications of the cameras.

Brand	Safety Vision
Model	37 series IP camera
Image Sensor	1/2.8” CMOS
Highest Resolution	1920 × 1080
Focal Length	2.8mm, 4.0mm
Field of View	H: 84.0°, V:43.3°, D:99.4°
Video Compression	H.264, Motion JPEG
Infrared Illuminators	4
Maximum IR Distance	30 m
Water Ingress Protection	IP67

Table 2: Technical specifications of the four exterior cameras.

Brand	Safety Vision
Model	43 series IP camera
Image Sensor	1/3” CMOS
Highest Resolution	1920 × 1080
Focal Length	2.8mm
Field of View	H: 80.0°, V:44.0°, D:93.5°
Video Compression	H.264, Motion JPEG
Infrared Illuminators	10
Maximum IR Distance	15 m

Table 3: Technical specifications of the interior camera.

GPS and Network Antenna. We use Mobile Mark’s LTM501 Series Multiband MIMO (multiple-input-multiple-output) antenna, which contains five separate antennas, all in one compact antenna housing. The five antennas include two LTM/Cellular antennas, two dual-band WiFi antennas, and one GPS antenna.

6. Example Application: LiveMap for HD Map Update

Building a HD map and maintaining it periodically are very important for autonomous vehicles, but will be expensive if we use all of those sophisticated sensors like LiDAR and precision GPS and IMU to do the monitoring and update frequently. Moreover, we also want to be notified when something in the HD map has changed as soon as possible, e.g. new traffic signs and road markings or some other infrastructure. Based on the BusEdge platform, we build a pipeline to achieve the traffic sign monitoring for HD map update, and we name it “LiveMap.” The goal is to get a notification and update the HD map once an old traffic sign is removed or a new one is detected.

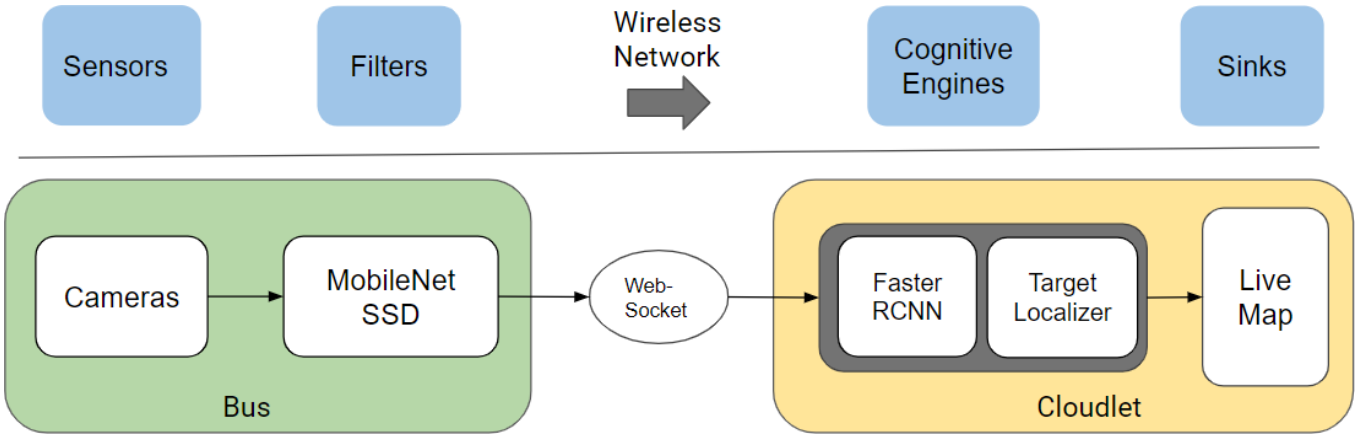

Figure 8: The pipeline of LiveMap

An illustration of the pipeline is shown in Figure 8. We have MobileNet- SSD [6, 7] as the early-discard filter; we use Faster RCNN [8] and a target localizer as the cognitive engines; we have a database and a web server at the end to show the final results.

Specifically, we utilize cameras installed on the transit buses to capture data and a light-weight detector is applied to filter out the image inputs. Those selected images will then be sent to the cloudlet server and fed to a more sophisticated traffic sign detector. With the detected bounding boxes, these images will be registered to a prebuilt

3D model to get the accurate positions of the detected signs. At the end, we could synthesize the results we get from the cognitive engines and the HD map database, to determine whether changes of traffic signs have taken place and mark the change on a map server built on OpenStreetMap. Figure 9 shows some example outputs of the whole pipeline, including results from the filter stage and the final visualization stage.

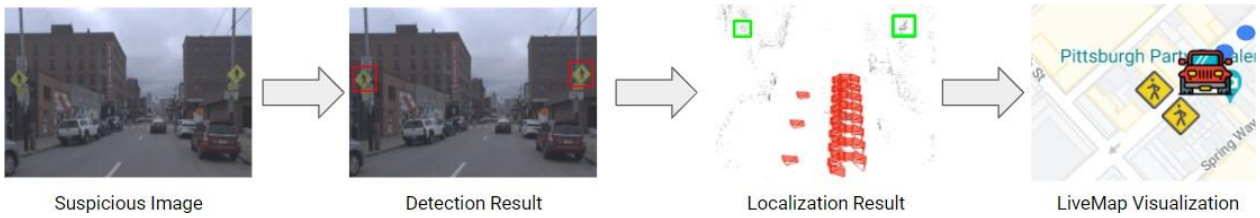

Figure 9: Example outputs of the LiveMap pipeline

7. Outlook for the next project phases

The hardware and software is ready to be installed on the bus. We are planning the installation in early 2021. First we will test the basic building blocks and functionality of the system while it is running live. Next, we will make improvements to the system so that the example application will run efficiently. This will ensure that we can build additional applications into the system. Finally we want to put this platform into a larger research context by combining it with data streams from stationary cameras. The goal is to have a city-wide system that can monitor traffic and infrastructure on many different time scales.

Bibliography

[1] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. *Towards wearable cognitive assistance*. In Proceedings of the 12th annual international conference on Mobile systems, applications, and services, pages 68–81, 2014.

[2] Junjue Wang, Ziqiang Feng, Shilpa George, Roger Iyengar, Padmanabhan Pillai,and Mahadev Satyanarayanan. *Towards scalable edge-native applications*. In Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, pages 152–165,2019

[3] Mordechai Haklay and Patrick Weber. *Openstreetmap: User-generated street maps*. *IEEE Pervasive computing*, 7(4):12–18, 2008.

[4] Wenlu Hu, Ziqiang Feng, Zhuo Chen, Jan Harkes, Padmanabhan Pillai, and Mahadev Satyanarayanan. *Live synthesis of vehicle-sourced data over 4G LTE*. InProceedings of the 20th ACM International Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems, pages 161–170, 2017.

[5] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[6] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. *Mobilenets: Efficient convolutional neural networks for mobile vision applications*. *arXiv preprint arXiv:1704.04861*, 2017.

[7] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. *SSD: Single shot multibox detector*. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[8] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. *Faster r-cnn: Towards real-time object detection with region proposal networks*. *arXiv preprint arXiv:1506.01497*, 2015.