# Live Synthesis of Vehicle-Sourced Data Over 4G LTE

Wenlu Hu
Carnegie Mellon University
wenlu@cmu.edu

Ziqiang Feng
Carnegie Mellon University
zf@cs.cmu.edu

Zhuo Chen
Carnegie Mellon University
zhuoc@cs.cmu.edu

Jan Harkes
Carnegie Mellon University
jaharkes@cs.cmu.edu

Padmanabhan Pillai
Intel Labs
padmanabhan.s.pillai@intel.com

Mahadev Satyanarayanan
Carnegie Mellon University
satya@cs.cmu.edu

## ABSTRACT

Accurate, up-to-date maps of transient traffic and hazards are invaluable to drivers, city managers, and the emerging class of self-driving vehicles. We present LiveMap, a scalable, automated system for acquiring, curating, and disseminating detailed, continually-updated road conditions in a region. LiveMap leverages in-vehicle cameras, sensors, and processors to crowd-source hazard detection without human intervention. We build a real-time simulation framework that allows a mix of real and simulated components to be tested together at scale. We demonstrate that LiveMap can work well at city scales within the limits of today's cellular network bandwidth. We also show the feasibility of accurate, in-vehicle, computer-vision-based hazard detection.

## CCS CONCEPTS

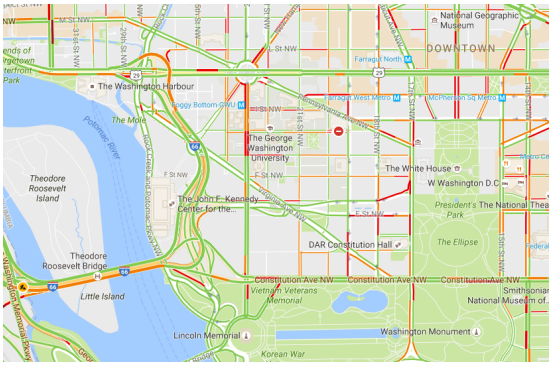
•**Networks** → **Application layer protocols; Mobile networks;** *Network performance modeling; Network experimentation;* •**Software and its engineering** → **Distributed systems organizing principles;** *Middleware; Operating systems; Client-server architectures;* •**Information systems** → **Sensor networks; Mobile information processing systems;** *Geographic information systems;*

## KEYWORDS

Vehicular Systems; Automotive Systems; Maps; Cloudlet; Edge Computing; Cloud Computing; Situational Awareness; Driverless Cars

## 1 Introduction

Every day, millions of drivers benefit from real-time synthesis of GPS location data that is periodically transmitted by participating vehicles (Figure 1). In this paper, we examine future extensions of this concept to provide fine-grain, deep-zoom details about road conditions and hazards such as *"Dead deer in left lane at GPS location (x,y), here is an image;"* or, *" Fog detected at GPS location (x,y), visibility down to 30 feet, here is a short video clip."* Receiving map overlays with such details in near real-time could greatly improve the situational awareness of many stakeholders such as driverless vehicles, drivers, road-maintenance crews, emergency personnel, and law enforcement officers [23]. We look to a future when such reports can be algorithmically-generated, without human assistance, from video cameras and other sensors on a vehicle.




Green: normal Orange: slow Red: very slow

**Figure 1: Traffic Overlay on Google Maps**

This paper focuses on *LiveMap,* a scalable mobile information system that synthesizes vehicular update streams in real-time. Our informal scalability goal is tens of thousands of vehicles over a county-sized coverage area. For the foreseeable future, 4G LTE offers the most plausible wide-area Internet connectivity from a moving vehicle. The demand for this resource is intense, and its spectrum-limited supply is scarce [9]. Hence, the crucial requirement for our system is to be frugal in terms of wireless transmission. Peak bandwidth demand as well as total volume of data transmitted should be minimized, while offering timely synthesis.

A simple implementation strategy would be to ship the video from moving vehicles over 4G LTE to the cloud or a regional data collection point for real-time video analytics and generation of map overlays. Unfortunately, this is not scalable in terms of wireless network usage. For example, consider rush hour in Manhattan over the two-block by two-block coverage area of a small cell. Using an average vehicle size of 5 m and a separation of 5 m, such an area can accommodate roughly 400 vehicles. If each vehicle streams SD video at 3.0 Mbps (Netflix's estimate [17]), the total aggregate uplink demand in the cell would be 1.2 Gbps. This clearly exceeds the stated capacity of 500 Mbps for LTE Advanced technology [16]. Streaming HD or 4K video rather than SD video would improve the quality of the video analytics and expose finer-grain features, but would worsen the bandwidth problem. Non-urban areas have much lower density of vehicles, but their cells are larger.

Scalability can be greatly improved by performing video analytics on board each vehicle at an edge computing device called a


*MSWiM '17, November 21–25, 2017, Miami, FL, USA*

IBSN 978-1-4503-5162-1/17/11.
DOI: https://doi.org/10.1145/3127540.3127543

*cloudlet* [24]. Only the extracted information, encoded in a standardized format such as XML or JSON, needs to be shipped over 4G LTE to the data collection center. For example, analysis of a single video frame that is multiple megabytes in size may result in output that is just a few hundred bytes in size. This reduction in bandwidth demand by 4 to 5 orders of magnitude is crucial for scalability. In practice, the system would require an image or short video clip to be uploaded for each detected event, for further analysis and confirmation purposes. Even with this, the bandwidth savings from processing in the vehicle over streaming all video to the regional data collection point will be tremendous. Of course, a critical requirement is that a vehicle's cloudlet be powerful enough to transform continuous real-time input from its video and other sensors into a low-bandwidth semantic update stream. This is a reasonable assumption with today's computing technology.

We focus on scalability issues in this paper, ignoring broader issues such as privacy, incentive structure for participation in LiveMap, and the HCI issues involved in optimally delivering synthesized output to different types of consumers. In this paper, we first describe the real-time simulation framework we have built to allow us to run and test a mix of real and simulated instances of system components at scale. We then describe how we model LiveMap components in our simulation framework, followed by an evaluation of bandwidth consumption for different upload policies. Finally, we demonstrate the feasibility of in-vehicle hazard detection by building detectors to find deer and potholes.

## 2 Background and Related Work

Research on vehicular communication and computing spans nearly two decades. A major theme has been *vehicle to vehicle (V2V)* use cases. These focus on transient information (lifetimes of milliseconds to seconds), whereas LiveMap involves persistence of sensor information and map information over timescales of minutes to hours to days. Second, the response times involved differ by two or more orders of magnitude: a few milliseconds or less for V2V use cases, versus LiveMap's best-effort response times of hundreds of milliseconds to a few seconds to detect and report an observation. Thus, LiveMap is "near real-time" rather than "hard real-time."

Closely related to V2V use cases is the entire body of wireless networking research on *vehicular ad hoc networking (VANET)*. LiveMap does not rely on VANET technologies, but instead relies on the widely-used 4G LTE technology. The unique challenges of using LTE for vehicular use cases have been discussed by Araniti et al [6], and their insights apply to our work.

*Driverless vehicles* are another hot topic in vehicular research. Many of the V2V safety use cases mentioned above are relevant to driverless vehicles. Rapid sensing and actuation for collision avoidance are essential for any driverless vehicle. At the same time, proactive actions based on detailed map information are (whenever possible) better than reactive just-in-time actions. As Autor points out [7], "A Google car navigates through the road network primarily by comparing its real-time audio-visual sensor data against painstakingly hand-curated maps that specify the exact locations of all roads, signals, signage, and obstacles." The creation of these detailed maps, which change rapidly over time, is a large hidden cost of driverless vehicular technology. LiveMap could cheaply and continuously crowd-source the creation of these detailed maps.

Closest in spirit to LiveMap are commercial map services such as Waze [28]. LiveMap can be viewed as a Waze-like system that automates the sensing, reporting, and synthesis of events. Instead of relying on human input, LiveMap is based on sensor data that is locally processed to generate map update reports. It would be simple to extend LiveMap to also allow human input.

Independent of vehicular contexts, there is a huge body of work on data aggregation in sensor networks [12, 15, 26].That work has tended to focus on small low-cost sensors where the dominant constraint is the energy cost of sensing, processing and transmission. In contrast, energy usage for processing and transmission is only a minor consideration in LiveMap. Relative to the energy consumed in accelerating and maintaining a vehicle and its occupants at highway speeds, the energy used by LiveMap is modest. Bandwidth demand on 4G LTE is the dominant theme for LiveMap.

## 3 Simulation Framework

### 3.1 Goals and Requirements

Our intention is to build a prototype implementation of the LiveMap system and test it at scale. As it is impractical to actually implement, deploy, and connect even a small set of vehicles with cameras and computation, and drive them around a city, we instead rely on realistic simulations to provide scale. However, we would like to be able to plug in a few instances of real, implemented components, and have them interact with the large number of simulated components. This will allow us to both run the system at large scale and to test the actual implemented application code. Our primary focus is on system scalability, so accuracy of sensing models or low-level details of the network are less critical. Based on these considerations, we derive the following requirements:

- simulate vehicles and applications that communicate with fixed infrastructure
- support real maps and realistic traffic patterns
- allow interfacing with real implementations of system components by executing in real time
- support county- or city-scale simulations

The rest of this section details the simulation system that we have designed and implemented to meet these requirements.

### 3.2 Vehicle Simulation

We began our investigations with the Veins [25] system, which is intended for the study of connected vehicular systems. It provides accurate modeling of vehicular communication networks, and includes models for V2V communication and for LTE. Furthermore, it also provides a straightforward way to run custom application logic on each simulated vehicle, a key need for our work.

Veins itself is built on top of SUMO [8] and Omnet++ [27]. SUMO is an open-source vehicle simulation framework that is widely used to study traffic patterns and smart vehicle coordination, and has been shown to realistically simulate traffic patterns on maps of real cities. Omnet++ provides full network stack simulation and is used to provide accurate models of connectivity and communication among vehicles and to fixed infrastructure.

Although Veins is functionally well-suited to our goals, it suffers from performance issues. Its architecture separates the SUMO and

Omnet++ components into different processes, thus incurring inter-process communication overhead at each simulation step. Further, the network is modeled much more precisely than we need. Hence, we also investigate using SUMO alone, extending it with just the features we need.

### 3.3 Maps and Traffic Patterns

SUMO and, by extension, Veins have excellent support for using real maps in simulations. SUMO provides a tool, NETCONVERT, to convert the map data from OpenStreetMap [18] to the SUMO format. OpenStreetMap is a crowd-sourced map of the world, open and free to the public. Although this converting process is not perfect, e.g., the locations and changing cycles of traffic lights are guessed, it does allow simulations on almost any real road network.

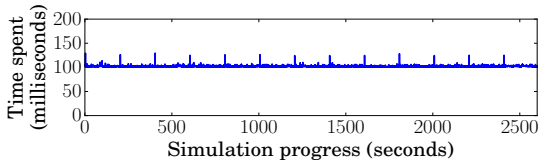
In addition to the maps, we also need realistic traffic patterns as input models to the simulator. The largest publicly available input model is the TAPAS Cologne dataset [5]. It describes traffic in the city of Cologne, Germany for a whole day, derived from observed traveling habits and information about the infrastructure of the area. Rush hours in this dataset have up to 14,000 vehicles on the road at the same time, providing us with a clear scalability target.

### 3.4 Real-time Simulation

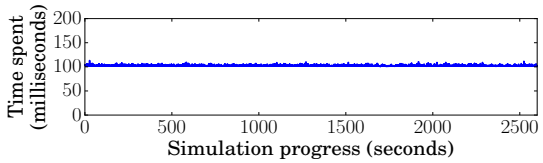
A key goal of our work is to mix real and simulated components together. As a consequence, we need a simulation system that can run in real time. In other words, the simulated time step equals the real-world elapsed time for that step. Both SUMO and Veins are designed for offline simulations. To interface the simulated world to the real components, we ensure that the simulation time of SUMO is synchronized with real, wall-clock time by adding a high-precision sleep to SUMO simulation steps. This modification adds just enough sleep to the end of each simulation step to allow wall-clock time to catch up to simulation time.

Of course, this works only when the simulation step takes no more time to execute than the corresponding real world time period. How well does this hold true? When running Veins with a few hundred cars and a time step of 100 ms, we note that most steps take just a few milliseconds to execute. However, some take significantly longer, more than 100 ms, and violate the requirement that elapsed time be less than simulated time. The largest spikes are due to writing of logs, $O(n^2)$ heartbeat messaging, and synchronized introduction of new vehicles into the simulation. We reconfigure Veins to eliminate these issues, and also pin the simulation process to a dedicated processor core to reduce context switching. However, more subtle, periodic spikes persist, as shown in Figure 2(a). After investigation, we determined these are due to the way the simulator loads input data — every 200 simulation steps, it loads inputs for the next set of steps. By preloading all inputs, we finally eliminate these spikes, as shown in Figure 2(b). Note that these plots reflect our modification that introduces a high-precision sleep after every step that executes too quickly, bringing the step time up to the desired value. The ideal curve would be a straight line at 100 ms.

As the execution time of a step depends on the size of the simulation, this limits the scale of real-time simulation. Figure 3 compares the execution time of a single simulation step to the real-world time it represents for Veins as the number of vehicles is increased. The




(a) Periodic spikes in Veins execution time




(b) Consistent execution times with fully pre-loaded input data

**Figure 2: Execution Time Variability in Veins**

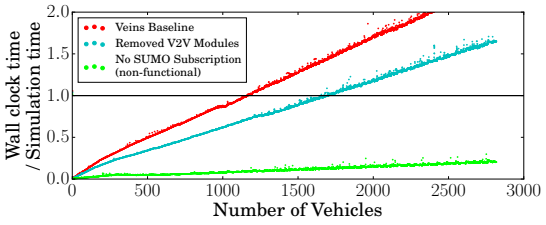



**Figure 3: Scalability of Veins**

Y axis presents execution time of each simulation step, normalized by the step size (100ms). Real-time simulation can only be achieved when this ratio is below 1, as shown by the horizontal line. As we can see, Veins can run in real-time for up to 1200 vehicles on a 3.6 GHz CPU (turbo clock rate). As Veins is largely sequential, increasing the number of CPU cores does not improve scalability.

As LiveMap does not involve V2V communications, we modified Veins to eliminate V2V-related functionality, in the hope of increasing scalability. This configuration supports real-time simulation of up to 1700 vehicles. This disappointing result shows that Veins will likely not suffice for our goal of county-scale simulations.

Further investigations showed the bottleneck to be due to transferring data between the SUMO simulation and the rest of Veins, which run in separate processes. To evaluate this bottleneck, we disabled the inter-process data transfer, using dummy data instead. The third curve in Figure 3 is for this non-functional system and indicates that real-time simulation at significantly higher scales is possible without the multi-process architecture imposed by Veins.

### 3.5 Scaling to County-size Inputs

To allow real-time simulation with more than 1700 vehicles, we must forego Veins and Omnet++, and instead use SUMO alone. We retest scalability with SUMO alone, plotting the execution time of a simulation step relative to the length of time that the step represents as the number of vehicles is increased. Figure 4 shows that SUMO can simulate up to 20,000 vehicles while maintaining real-time performance on the same 3.6 GHz CPU (turbo clock rate) as in the Veins experiments. We note that this does not include any LiveMap code or network communication. With careful implementation of application logic for the vehicles, and use of real

networking between the simulated vehicles and external components, we are able to scale SUMO-based, real-time simulations of LiveMap to 14,000 simultaneous vehicles needed for the Cologne dataset, for most LiveMap configurations. For some variants that require more processing time, we slightly increase the step size to keep the simulation real-time.

Further scaling of SUMO is limited by its architecture. The SUMO core simulation is single-threaded, and prior attempts to parallelize it have not been very successful [14]. A faster simulation model provided by SUMO, the mesoscopic model, only outputs aggregated information at the road level and is not useful for LiveMap, where the locations of individual vehicles along roads are important. Despite these limitations, we are able to use SUMO to demonstrate LiveMap at the scale of a city.

### 3.6 Simulating In-vehicle Application Logic

We extend SUMO to add support for custom application logic that is run on each vehicle. The application logic for LiveMap implements a model for sensing of road hazards, an application-level data cache, protocols to maintain the cache, and logic to decide when and what updates based on sensed hazards should be sent. Our SUMO extension permits a single application callback method, which is invoked once per vehicle, per simulation step. This requires the application logic to use a polled, event-driven style, and to explicitly keep track of state of activities across simulation steps. To avoid slowing the simulation, the application method is required to return quickly, and avoid long running computations or blocking calls. To support slow operations and blocking calls (e.g., network communication operations), our system provides a means of deferred execution – the operations are queued and executed in the background by a worker thread pool. It is up to the application method to check if the deferred operation has completed in a future invocation. Finally, we use multiple threads to run the application callbacks concurrently. These implementation choices add some complexity and introduce some nondeterminism into the simulation, but ensure that real-time performance is minimally impacted.

For network communications in our simulation, we use an actual wired connection instead of network models to avoid their impact on the real-time performance. This setup can be seen as an upper bound of the cellular network and may have an impact on simulation accuracy. As shown in Section 5, this impact is very small in terms of the metrics important to LiveMap.

## 4 Modeling LiveMap

### 4.1 LiveMap Components

LiveMap is a distributed sensing and aggregation system intending to provide situational awareness across a region. It primarily consists of a large number of vehicles with multiple sensors, typically cameras, and significant compute capability in the form of in-vehicle cloudlets. These vehicles observe, detect, and report anomalies and hazards. For this paper, all of the vehicles and corresponding in-vehicle cloudlets are simulated, but the LiveMap software is real.

We refer to the coverage area of a single instance of LiveMap as its *zone.* A centralized entity called the *zone cloudlet* is responsible for the zone. We use the term *in-vehicle cloudlet* in the rest of the paper to distinguish in-vehicle cloudlets from zone cloudlets. A zone cloudlet fuses inputs from all in-vehicle cloudlets in its zone, curates the data to ensure quality control, enforces security and privacy policies, and selectively disseminates the synthesized knowledge to participants. A zone cloudlet may be physically replicated for survivability, and standard failover protocols can be used to create a high-availability LiveMap service for each zone. A working prototype zone cloudlet has been implemented. It interacts with the simulated in-vehicle cloudlets over network connections.

How large a zone should LiveMap target? It is within a single zone that LiveMap offers the best situational awareness — i.e., the most up to date and timely sharing of information across participating entities. While it is tempting to consider the entire planet as one giant zone, there are many reasons why smaller zones are advisable. In particular, the granularity and resolution of detail of synthesized information has to be fine enough to base the second-to-second actions of driverless vehicles. For a vehicle traveling at 70 mph (roughly 100 feet per second), hazards as small as a one-foot pothole or an even smaller rock are worthy of attention over the many hundred feet of the road that will be covered in the next few seconds. At such a fine spatial and temporal granularity, with the end-to-end latency of today's networking technologies as a guide and the speed of light as a lower bound, it is hard to see how to create a single zone that spans the entire planet. What appears feasible is a federation of many smaller zones. Across that federation, the spatial and temporal granularity of knowledge propagation may be significantly lower than within a single zone. Even if observers outside a zone can "zoom in" to details within that zone, there will be significant lag in seeing updates. Our intuition which is validated through the results presented later is that a city-sized or county-sized coverage area is feasible today, and the focus of our simulation work. As the end-to-end latency of networking technologies improves, and as our experience with LiveMap implementation matures, it is conceivable that a typical zone may expand to a medium-sized US state.

### 4.2 Vehicle-Zone Interactions

Figure 5 shows the interactions between an in-vehicle cloudlet and its currently-associated zone cloudlet. All of the data streams shown are implemented over TCP connections. In a real deployment, these will be secured using standard SSL/TLS mechanisms. The in-vehicle cloudlet performs edge analytics on external sensor inputs (e.g., video cameras, possibly multiple per vehicle) and internal sensor readings (such as speed, engine performance parameters, occupant alertness, etc.). These edge analytics transform the high data rate of raw sensor data into a semantic update stream of much lower bandwidth. Several decentralized transmission control mechanisms, described in Section 5.3, can be used to determine whether a specific update is likely to be redundant because of reports from other vehicles. The updates deemed redundant are suppressed, while the rest are transmitted to the zone cloudlet (arrow ① in Figure 5).

The transmission control mechanism in a vehicle may sometimes be too aggressive. Some data deemed redundant by an in-vehicle cloudlet may, in fact, be valuable to the zone cloudlet. From time to time, a zone cloudlet may explicitly request more information or ask for confirmation of an observation from another vehicle

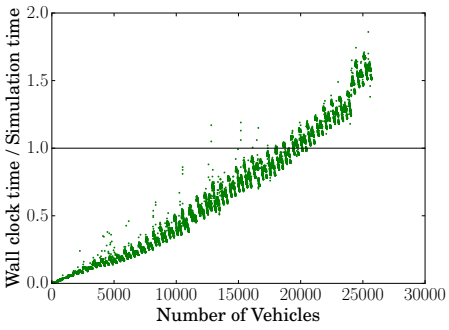



**Figure 4: SUMO Scalability**


**Figure 5: Vehicle-Zone Interactions**

(arrow ② in Figure 5). This request is an implicit hint to reduce the throttling of updates by a in-vehicle cloudlet. Each in-vehicle cloudlet caches data from the zone cloudlet. The communication to maintain cache consistency is shown as arrow ③ in Figure 5.

Raw sensor data is buffered in local storage at the in-vehicle cloudlet for a finite period of time. Retention is valuable if a need arises later to re-process the data with fresh analytics, or to drill down for more details. For example, if a public service alert is issued for a lost child, it may be valuable to search for the child's face and clothing in the retained video data from vehicles that recently passed through relevant neighborhoods. At an average of 3 GB per hour for HD video [17], almost two weeks of video can be stored in a modest 1 TB disk, that costs only $50 today. In Figure 5, arrow ④ corresponds to these ad hoc interactions between the zone cloudlet and in-vehicle cloudlet. An authorization mechanism and policy to determine who can present such requests will be needed in a real-world implementation. More details on how vehicle-zone interaction is implemented in our prototype can be found in Section 5.1.

## 4.3 Synthetic Hazard Generation

To the best of our knowledge, there are no existing large datasets of road hazards from which we can mine sophisticated statistical patterns. Therefore we create our own synthetic road hazard event generator. We model events with the following assumptions and constraints. First, we assume different types of road hazards (e.g., "dead deer" vs. "car accident") are independent and happen according to type-specific probabilities. Thus, for example, we can configure hazard profiles with a large number of disabled cars, but just a few deer sightings. We also assume that events happening on different road segments are independent. Furthermore, in each unit of time, the number of events of a particular type that happen on a road segment follows a Poisson distribution. Thus, the probability that $k$ events happen is $P(k) = e^{-\lambda}\frac{\lambda^k}{k!}$, where $\lambda$ is the expected number of events in one unit of time. We modify this slightly so that the expected number of events on a road segment depends on its "area," which is determined by its length and the number of lanes. Therefore, a three-lane highway will in average have 3 times the number of events as a similar length single-lane road. Finally, we constrain particular hazard types to only happen on specific types of roads, e.g., a "car accident" cannot happen on a cycleway.

Our hazard generator takes two files as input: a *hazard profile*, containing statistical parameters and constraints, and an OpenStreetMap file of a region. It also takes in the starting and ending times of a simulation and a time unit. By default, we use a time unit of one second. It then produces a trace log of road hazards, each with a timestamp, coordinates, and duration.

We would like the same hazard profile to be applicable to any map without modification, so that we can generate hazard traces for two different cities with similar statistical characteristics. Hence the parameters and constraints specified in the hazard profile are independent of the map (e.g., New York or Cologne). For a certain hazard type $e$, we specify a parameter $\beta_e$, and a list of road types it is allowed to occur on. On each road segment, its Poisson parameter is $\lambda_e = \beta_e \cdot area$. For each time unit, we generate events for each road segment and each hazard type independently. The final outputs are merged and sorted by time for playback.

## 4.4 Sensing Model

The simulated vehicles in our system execute the complete LiveMap application logic and protocol explained in Section 4.2. They do not, however, execute real video analytics code to detect hazards. To use real analytics code, the simulation framework would need to generate realistic camera views, including realistic portrayals of synthetic hazards and other vehicles in the system, for each vehicle at each timestep, and then execute the relatively expensive analytics algorithms to detect the hazards. This would be impractical, and greatly limit the scalability of the simulation.

Since we are primarily interested in testing scalability of the the LiveMap system, and not accuracy of analytics, we instead use a simple, fast sensing model to decide when simulated vehicles detect hazards. We assume each vehicle is equipped with multiple cameras that can view in all directions. The in-vehicle cloudlet is assumed to perform computer vision algorithms to detect events of interest (e.g., stopped cars, obstacles, etc.). The feasibility of such analytics is demonstrated in Section 6. Our model approximates sensing with omnidirectional cameras: within a configurable radius (50 m default), any hazard is assumed to be seen and detected. At each simulation step for each vehicle, the list of currently active

hazards from the generated trace is consulted, and a sublist of active hazards within the detection radius of the vehicle's current position are returned by the simulated analytics. Other application logic decides if these need to be reported to the zone cloudlet, according to a variety of policies explored in Section 5.3.

## 5 Zone Cloudlet Prototype and Evaluation

To study our ideas on improving scalability and bandwidth use of automobile-based sensing, we have implemented a prototype of a zone cloudlet that serves a large collection of in-vehicle cloudlets at city or county scales. Although we test this server prototype with simulated in-vehicle cloudlets, the design and implementation of the server is fully independent of the simulation framework. The same server prototype can serve a large collection of real vehicles without modification. This feature is valuable when we extend the simulation to include real vehicles in the future.

As cellular bandwidth is the scarce resource in the system, we use this prototype to explore ways to lower the bandwidth needed. We seek quantitative answers to the following questions:

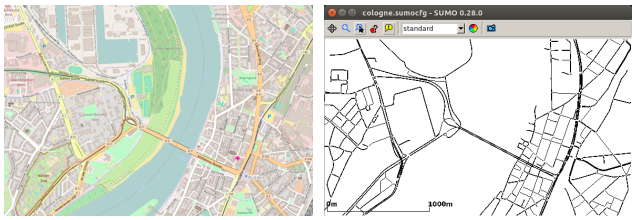
- How much bandwidth do LiveMap updates consume?
- How effective are different bandwidth throttling policies?
- What is the bandwidth-accuracy tradeoff?
- How well do caching and cache-based policies perform?
- How close can we get to the theoretical lower bound?

### 5.1 Zone Cloudlet Implementation

The zone cloudlet service is a typical TCP server written in Python. The service listens for incoming communications from vehicles, and sends messages to concurrent handlers for processing. Although there may be tens of thousands of active vehicles, only a small fraction of them will be in communication with the zone cloudlet at the same time, limiting the level of concurrency needed. These handlers update an in-memory Redis database [3] of the current state of the world. Each current road hazard is stored as an entry in the database, and an index of its location is created to facilitate fast search. We use the Gevent library [1] to provide a coroutine-based concurrency implementation underneath a thread-like API. Since the service is I/O bound, the coroutine approach works well. When the workload fully utilizes one CPU core, we spawn multiple processes of the same server and use a HAProxy load balancer [2] to coordinate them. We separate the information flow into two phases, data acquisition and data dissemination, and study them separately. Data transferred for acquisition is mostly on the uplink and data for data dissemination is mostly on the downlink.

The most naïve approach for data acquisition is to let vehicles report every road hazard they observe to the zone cloudlet. This approach provides the most accurate map (high coverage and low staleness), but transfers the most bytes and consumes the most bandwidth. We call this approach `upload-all` and use it as a baseline. The other extreme is the unattainable but ideal `oracle-driven` approach, where each hazard is reported exactly once, and on the earliest observation. This gives a lower bound on the bandwidth demand of any approach that provides full coverage.

We design and implement three other data acquisition approaches. The first is a probabilistic approach called `upload-X%`. With this approach, whenever a vehicle observes a hazard, it throws a die to decide whether or not to report this hazard. The upload happens X% of the time, statically configured across all vehicles. The second approach, `throttle-by-traffic`, uses a dynamic upload probability that is inversely proportional to the vehicle density in a 50m x 50m grid cell. The zone cloudlet tracks the traffic density of each cell, based on reports from the vehicles as they enter a new grid cell, and sends this aggregated information back to the vehicles.




(a) OpenStreetMap (b) Extracted SUMO road network

**Figure 6: Parts of the Map in the Cologne Dataset**

The third approach is deterministic. The `throttle-by-cache` approach requires vehicles to maintain a cache of the live map of the surrounding area. The cache contains a subset of the map grid cells where each cell is 50m x 50m. When vehicles observe a hazard, they consult their cache and only upload new observations. To keep the cache up to date, vehicles refresh their cache when leaving the cached area, or when they receive an invalidation callback from the zone cloudlet indicating the cache may be stale. The callback mechanism is implemented on top of a modified Paho MQTT library [13], a low-bandwidth publish-and-subscribe system for IoT applications. We modify its asynchronous I/O multiplexing mechanism from `select` to `poll` to work for the scale of our experiments. The Pub/Sub channels correspond to grid cells. The vehicles subscribe to channels related to the cells in their cache, and the zone cloudlet publishes "cache invalid" messages to the corresponding channels when appropriate.

In addition to maintaining a live map, the zone cloudlet disseminates the acquired information to vehicles. If the network supports broadcast messages, the most efficient way to disseminate data is to broadcast a message to all vehicles when the zone cloudlet first learns about a road hazard. If broadcast is not available, a substitute can be emulated broadcast: the zone cloudlet sends one message to each vehicle for every road hazard. Dissemination can also be done with callback caching. With callback caching, the vehicles do not know about all hazards on the map, but they know the ones in their surroundings. With the `throttle-by-cache` acquisition option, callback caching is automatically assumed.

### 5.2 Experimental Setup

We run our zone cloudlet services and vehicle simulation framework in two virtual machines (VMs) on the same physical host, emulating near-perfect networking between vehicles and the zone cloudlet. The tradeoff between fidelity and scalability/practicality is discussed in more detail in Section 3. The host machine is a server with two Intel®Xeon®E5-2699 v3 processors (2.30 GHz, turbo 3.6 GHz, total of 36 cores, 72 hyper threads) and 128 GB memory. The zone cloudlet VM is configured with 4 GB memory and 8 VCPUs, and

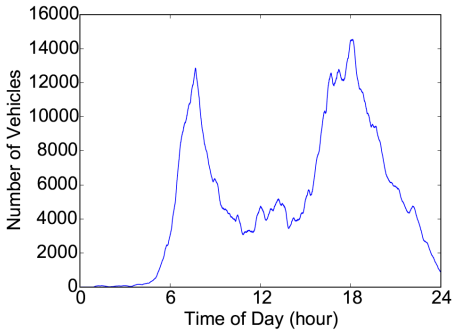



**Figure 7: Traffic Statistics of TAPAS Cologne Dataset**

| | |
|---|---|
| Map area | 1110 sq km |
| Vehicle total | 462,000 vehicles |
| Peak traffic | 14,000 vehicles |
| Median trip length | 11 minutes |

**Figure 8: Summary of the Cologne Dataset Statistics**

the simulation VM is configured with 8 GB memory and 32 VCPUs. These are ample resources for our experiments.

We use the TAPAS Cologne dataset (introduced in Section 3.3) for our experiments unless otherwise specified. Figure 6 shows the OpenStreetMap excerpt as well as the extracted road network for SUMO corresponding to this dataset. As suggested by the dataset provider, we reduce traffic demands to 30% of the realistic value to avoid city-wide traffic jams. This is a limit of the dataset itself and the current traffic simulation technology. Even with this reduction, it is still the largest available dataset to the best of our knowledge. Figure 7 shows the number of vehicles in the simulation as the simulation progresses. The rush hours are 6 am to 8 am and 4:30 pm to 8pm. The number of vehicles peaks at 7 am with around 13,000 vehicles and again at 6 pm with around 14,000 vehicles. Figure 8 shows a summary of statistics about this dataset.

We measure the bandwidth consumed, bandwidth efficiency, and the accuracy of the live map constructed. Bandwidth efficiency is measured by *duplication*, the percentage of messages that repeat previously-reported information. A few (two to three) messages for a particular hazard are useful in helping the zone cloudlet verify crowd-sourced information and resolve conflicts. Thus, a good duplication value may be between 50% and 67%, while a much higher one means a waste of resources. LiveMap accuracy is described by hazard *coverage*, the percentage of road hazards that are reported to the zone cloudlet, and information *staleness*, the average latency between when a road hazard appears and when the zone cloudlet receives the first report about it.

A 2.0 MB video file is submitted with each hazard report, equivalent to approximately 10 seconds of SD video or 2.4 seconds of HD video. [17]. Experiments are run three times with different random seeds, and median results presented. The simulation step is set to 200 ms to account for the computing time of LiveMap. Fidelity is slightly sacrificed for scalability and practicality of the simulation.

## 5.3 Bandwidth Saving - Acquisition

Figure 9 shows how the different approaches perform. The baseline approach, `upload-all`, has the best map accuracy and largest

| Approach | Peak BW (Mbps) | Bytes Sent (GB) | Duplication | Coverage | Staleness (minutes) |
|---|---|---|---|---|---|
| upload-all | 1362 | 77 | 98% | 100% | 1.9 |
| upload-50% | 657 | 38 | 97% | 95% | 2.6 |
| upload-10% | 295 | 7 | 90% | 73% | 4.1 |
| throttle-by-traffic[a] | 809 | 63 | 98% | 98% | 2.1 |
| throttle-by-cache[b] | 1037 | 17 | 89% | 100% | 2.0 |
| oracle-driven | 153 | 2 | 0% | 100% | 1.9 |

[a]The number of traffic update messages is very large. 16% of the messages are not sent because of the limited number of threads in the simulation.
[b]The simulation step is relaxed to 500ms to account for more computing time.

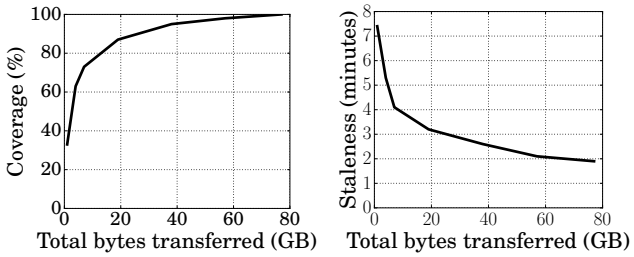
**Figure 9: Bandwidth Saving Techniques for Data Acquisition (Cologne Scenario)**

resource consumption by definition. The 1.9-minute staleness is mostly due to the time between when a hazard appears and when the first vehicle passes the area and notices it. The `upload-50%` approach lowers the peak bandwidth and bytes transferred approximately by half, at the cost of 3% on coverage, and 0.7 minute of staleness. `upload-10%` further reduces the peak bandwidth and bytes transferred to 10% of that of the baseline. 73% of the road hazards still get reported with 4.1-minute staleness. By controlling the upload probability in `upload-X%` approaches, we can tune this simple approach to fit different network bandwidth budgets with modest sacrifice of map accuracy.

The `throttle-by-traffic` approach has near-perfect hazard coverage and staleness. This approach significantly reduces peak bandwidth, but not the total bytes transferred. Typically peak bandwidth is required when many vehicles are in the area when a hazard appears, and multiple vehicles simultaneously report it. With `throttle-by-traffic` because of high vehicle density, these vehicles upload with only a small probability, thus significantly reducing peak bandwidth. As peaks do not occur very often, the total bytes transferred mostly depends on the other situations where the vehicle density is smaller. Whenever the vehicle density is below 50 vehicles per cell per hour, vehicles report all detected hazards. So this approach does not significantly save bytes transferred.

The `throttle-by-cache` also has near-perfect coverage and staleness. It reduces the total bytes transferred to 22% of that of the baseline, including the extra bytes needed to fetch and maintain cache. A hazard that has already been reported by another vehicle is not likely to be reported again. However, this does not reduce peak bandwidth significantly. When a hazard appears in a high-traffic area, multiple vehicles may report it before there is time for the information to appear in their caches, contributing to peak bandwidth and duplication. Although `throttle-by-cache` has the least duplication other than oracle, it is still significant at 89%.

The `throttle-by-traffic` and `throttle-by-cache` approaches are both very useful because of their high accuracy, as shown by coverage and staleness. They are also efficient in reducing peak bandwidth and total bytes transferred separately. A combination of them might be able to reduce both peak bandwidth and bytes transferred at the same time, while providing high accuracy. We are exploring this possibility in continuing studies.




(a) Coverage Vs. bytes transferred (b) Staleness Vs. bytes transferred

**Figure 10: `Upload-X%` Tradeoff (Cologne Scenario)**

| Technique | Bytes Transferred |
|---|---|
| Broadcast | 61 KB |
| Emulated broadcast | 567 MB |
| Callback caching | 524 MB |

**Figure 11: Bandwidth Saving Techniques for Data Dissemination (Cologne Scenario)**

## 5.4 Sensitivity to Acquisition Parameters

Some of the above approaches have tunable parameters that may affect the performance of the approach, such as the upload probability in `upload-X%` and the inverse proportion coefficient in `throttle-by-traffic`. These parameters serve as tuning knobs of the tradeoff between resources consumed (characterized by peak bandwidth and bytes transferred) and map accuracy (characterized by coverage and staleness). When these parameters change, it is straightforward to expect peak bandwidth and bytes transferred to change approximately proportionally. But it is difficult to predict how other metrics change, as the relationship is not linear.

We take `upload-X%` as an example to study this tradeoff. We experiment with varying values of upload probability and present the tradeoff curve in Figure 10. As we lower the upload probability, fewer bytes are transferred and fewer hazards are covered, and it takes longer for the zone cloudlet to learn about hazards. The first half of the bytes can be saved at a low cost of coverage and staleness. To further reduce the bytes transferred to a quarter of its original value, the staleness has to increase from 1.9 minutes to 3.3 minutes and coverage decreases from 100% to 87%. Further reduction in the number of bytes transferred comes at an even larger cost. The "knees" of the curves suggest a good operating range between 10 GB and 40 GB transferred, which corresponds to an upload probability between 10% to 50%. In this range, coverage is between 74% to 96% and staleness ranges from 2.6 to 4.1 minutes.

## 5.5 Bandwidth Saving - Dissemination

Figure 11 shows the performance of the data dissemination approaches. Surprisingly, similar numbers of bytes are transferred for callback caching and emulated broadcast. This is due to coarse-grain cache invalidation; if any cells change, vehicles will refresh the whole cache. These design choices reduce the number of messages sent, but transfer extra bytes for refreshing up-to-date portions of the cache. If we make caching granularity smaller and have vehicles only refresh the invalid cells, we will able to reduce this overhead significantly. If caching granularity is too small, the system will suffer from frequent inefficient small fetches of data. We will investigate the optimal cache granularity in continuing studies.

| Approach | Peak BW (Mbps) | Bytes Sent (GB) | Duplication | Coverage | Staleness (minutes) |
|---|---|---|---|---|---|
| upload-all | 2077 | 61 | 99% | 100% | 1.3 |
| upload-10% | 235 | 6 | 95% | 84% | 3.6 |
| throttle-by-traffic[a] | 792 | 61 | 99% | 100% | 1.3 |
| throttle-by-cache | 1186 | 3 | 74% | 100% | 1.3 |

[a] 2% of the messages are not sent due to limited number of threads in the simulation.

**Figure 12: Bandwidth Saving Techniques for Data Acquisition (Luxembourg Scenario)**

## 5.6 Sensitivity to Input Traffic Model

To verify the generalizability of our previous results, we run the same experiments on another input model, the Luxembourg SUMO Traffic Scenario [10]. It features a map of the Luxembourg City and traffic in this area for a whole day. The traffic patterns are synthesized with the SUMO ACTIVITYGEN tool, which takes detailed demographics data as an input. The dataset provides four variants of the traffic model with different mobility models and traffic light models. We choose the variant with the most traffic and run the experiments from 6 am to 9 am. The peak traffic is around 5200 vehicles at 8 am.

Figure 12 shows that the results are similar to those from the Cologne experiments. `upload-10%` reduces peak bandwidth and bytes transferred to 10% of their baseline values as expected. 84% of the hazards still get reported to the zone cloudlet, which is higher than that in the Cologne scenario, and it takes 2.3 minutes longer than the baseline for the zone cloudlet to learn about hazards. The `throttle-by-traffic` approach has perfect coverage and staleness, and significantly reduces peak bandwidth to 32% of the baseline. This reduction is much bigger than in the Cologne scenario. On the other hand, bytes transferred are not saved, unlike in the Cologne case. These differences may be due to the different traffic patterns of the two scenarios. Despite the differences, this approach is effective in significantly reducing peak bandwidth in both scenarios. `throttle-by-cache` still has the perfect coverage and staleness as expected. Comparing to the baseline, only 5% of the bytes are transferred. Peak bandwidth is also reduced by half.

Experiments on more input models can give us more insights. Unfortunately, to the best of our knowledge, there are no other large-scale per-vehicle traffic datasets publicly available. However in general, we believe `throttle-by-traffic` will consistently be more effective at reducing peak bandwidth, and `throttle-by-cache` will be more effective at reducing total bytes transferred. We plan to study hybrid approaches that combine these ideas.

# 6 Feasibility of In-vehicle Hazard Detection

Computer-vision based video analytics to detect road hazards is a critical component of LiveMap. Such analytics need to be fast enough to run on the in-vehicle cloudlets, yet provide reasonably good accuracy. False positives will result in unnecessary updates

| Dataset | # of Images | Labeled by |
|---|---|---|
| Google Deer Dataset | 340 | us |
| ImageNet Deer Dataset | 691 | ImageNet |
| Google Pothole Dataset | 34 | us |
| ImageNet Pothole Dataset | 267 | us |

**Figure 13: Summary of Training Datasets**

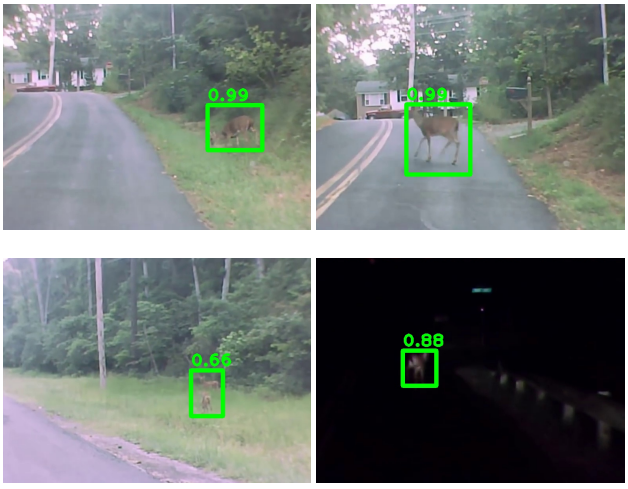



**Figure 14: Examples of Deer Detection Results**

and video transmitted to the zone cloudlet, wasting precious bandwidth. On the other hand, with too many false negatives, hazards are not detected and the system becomes useless. In this section, we demonstrate the feasibility of in-vehicle hazard detection by implementing a fast, reasonably accurate system for detecting deer and potholes.

Our implementation uses the state-of-the-art, neural-network-based Faster R-CNN [22] algorithm. To generate training data, we manually labeled deer and potholes in the videos and images found from the web. We employ the transfer learning approach [19] to reduce the total number of hand-labeled training images and total amount of training time needed for our detector. Figure 13 summarizes the datasets we used for training.

Running on a machine with a modern NVIDA Tesla K40 GPU, our detectors are able to operate at 7 frames per second (FPS). This confirms that today's computing technology is able to process video streams fast enough for LiveMap. The two proof-of-concept hazard detectors described below are representative of what is achievable today. Any computer vision work that improves road hazard detection will complement our work. When better detectors are available (for example, those created by entities such as RoadBotics [4] and Lost And Found [20]), they can be easily plugged into LiveMap to improve the accuracy and speed of hazard detection.

## 6.1 Example: Deer Detection

Obtaining appropriate data for training the object detector is a non-trivial task. A simple online search often returns images that do not have the right camera view for a vehicle mounted camera. This could lead to low detection accuracy. Therefore, we had to manually filter images to find ones with the right views (e.g., dashcam views) before including them in the training dataset. Our training data for deer detection comes from two sources. We first searched for "deer on road" in Google Image Search, and manually selected and annotated 340 valid images (Figure 13). We then included 691 annotated deer images from ImageNet [11]. Videos from YouTube did not provide good training data, as many of them are compilation videos deer-car collisions, without normal poses and views of deer.

In the precision-recall curve of a 10-fold cross-validation, the area under the curve of our detector is 87.8%. This is comparable to the reported accuracy of state-of-the-art object detection work today [22]. Figure 14 shows the detection results on example frames from a 2-minute YouTube dashcam video [21]. The full video with annotations of detection results can be found at https://youtu.be/_GrP42359z8.

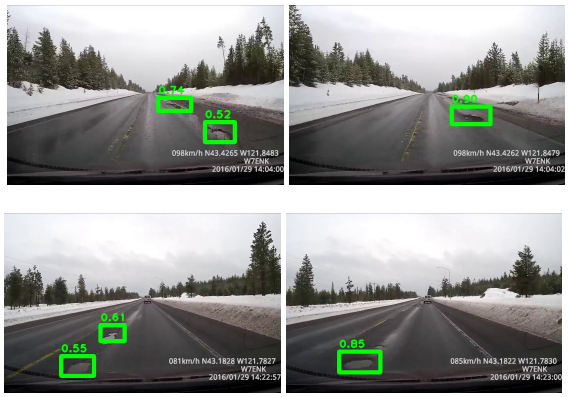



**Figure 15: Examples of Pothole Detection Results**

## 6.2 Example: Pothole Detection

Using a similar procedure as we followed for deer detection, we obtain several thousand images of potholes from ImageNet and Google. However, potholes are much harder to detect than deer, due to their greater variation in shape, and change in appearance with distance and viewing angles. The potholes at a distance can also be really small, only a few pixels in each dimension. This required us to perform a more careful screening of our raw dataset based on viewing angle, finally resulting in 267 images from ImageNet and 34 images from Google (Figure 13). To help reduce false positive rate with this fairly small set of images, we included the deer images as negative samples in our training set for the pothole detector.

Our trained pothole detector is sensitive to the viewing angle and distance to the pothole. So on still images, it typically only detects a subset of potholes. However, it performs well on YouTube dashcam videos like in Figure 15. A full video with annotated potholes can be found at https://youtu.be/U7_QAVbiF8U. Although some potholes may not be detected at a distance, they will likely be caught when the vehicle moves closer, leading to a hazard report in subsequent frames. In the video mentioned above, 913 unique potholes appear and 74% of them are detected in at least one frame. In addition, of the reported potholes, 75% are true positives.

## 7 Conclusion

A live, continuously-updated map overlayed with road conditions and hazards can provide the situational awareness needed to enable self-driving vehicles, empower human drivers and optimize city services. We have proposed LiveMap, an automated approach to this goal that employs in-vehicle processing of video and sensor data to detect road conditions, and uses a central zone cloudlet to manage, aggregate, and disseminate a unified view onto regional conditions. We have shown that LiveMap can scale to city or county scales within the limits of today's 4G LTE network bandwidth. We have also demonstrated the feasibility of in-vehicle computer-vision-based hazard detection.

Our evaluations of LiveMap are based on a novel mixed simulation framework that allows real implemented components and simulated ones to operate together. This effectively provides us the best of both worlds, allowing us to test real components and code, at system scales only practical in simulation. A key necessary requirement is that our simulation framework executes in real time. We are able to meet the real-time requirements at city scale simulation. Looking ahead, further scaling is limited by the single-threaded architecture of the core SUMO traffic simulator. To scale real-time simulation to hundreds of thousands of vehicles, significantly faster processor cores or an efficient multi-threaded implementation of SUMO will be needed. Finally, to better understand how LiveMap performs in the real world, we hope to deploy real vehicles instrumented with cameras, and in-vehicle cloudlets, and to construct a fully-operational instance of LiveMap.

## Acknowledgements

This research was supported by the National Science Foundation (NSF) under grant number CNS-1518865. Additional support was provided by Intel, Google, Vodafone, Deutsche Telekom, Verizon, Crown Castle, NTT, and the Conklin Kistler family fund. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the view(s) of their employers or the above-mentioned funding sources.

## REFERENCES

[1] Gevent. http://www.gevent.org/.

[2] HAProxy. http://www.haproxy.org/.

[3] Redis. https://redis.io/.

[4] RoadBotics. https://www.roadbotics.com/.

[5] TAPAS Cologne Scenario. http://sumo.dlr.de/wiki/Data/Scenarios/TAPASCologne, Accessed May 11, 2017.

[6] G. Araniti, C. Campolo, M. Condoluci, A. Iera, and A. Molinaro. LTE for vehicular networking: a survey. *IEEE Communications Magazine*, 51(5):148–157, May 2013.

[7] D. H. Autor. Why Are There Still So Many Jobs? The History and Future of Workplace Automation. *Journal of Economic Perspectives*, 29(3):3–30, Summer 2015.

[8] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz. Sumo–simulation of urban mobility: an overview. In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011.

[9] R. N. Clarke. Expanding mobile wireless capacity: The challenges presented by technology and economics. *Telecommunications Policy*, 38:693–708, 2014.

[10] L. Codeca, R. Frank, and T. Engel. Luxembourg sumo traffic (lust) scenario: 24 hours of mobility for vehicular networking research. In *Vehicular Networking Conference (VNC), 2015 IEEE*, pages 1–8. IEEE, 2015.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[12] H. R. Dhasian and P. Balasubramanian. Survey of data aggregation techniques using soft computing in wireless sensor networks. *International Journal of Information and Computation Technology*, 3(3):167–174, 2013.

[13] Eclipse. Paho. https://www.eclipse.org/paho/, Accessed May 18, 2017.

[14] K.-H. Kastner, R. Keber, P. Pau, and M. Samal. Real-time traffic conditions with sumo for its austria west. In *Simulation of Urban MObility User Conference*, pages 146–159. Springer, 2013.

[15] V. Kumar and S. Madria. Secure Data Aggregation in Wireless Sensor Networks. In T. Hara, V. I. Zadorozhny, and E. Buchmann, editors, *Wireless Sensor Network Technologies for the Information Explosion Era*, pages 77–107. Springer, Berlin, Heidelberg, 2010.

[16] LteWorld. LTE Advanced: Evolution of LTE. http://lteworld.org/blog/lte-advanced-evolution-lte, August 2009. Retrieved: 2016-1-11.

[17] Netflix Help Center. How can I control how much data Netflix uses? https://help.netflix.com/en/node/87. Accessed November 26, 2016.

[18] OpenStreetMap Foundation. OpenStreetMap. https://www.openstreetmap.org, Accessed May 19, 2017.

[19] S. J. Pan and Q. Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, October 2010.

[20] P. Pinggera, S. Ramos, S. Gehrig, U. Franke, C. Rother, and R. Mester. Lost and found: detecting small road hazards for self-driving vehicles. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 1099–1106. IEEE, 2016.

[21] W. Quinn. Deer on the road dec 2015. https://www.youtube.com/watch?v=JPSQUkT8rZY.

[22] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[23] M. Satyanarayanan. Edge Computing for Situational Awareness. In *Proceedings of the 23rd IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN 2017)*, Osaka, Japan, June 2017.

[24] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4), 2009.

[25] C. Sommer, R. German, and F. Dressler. Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis. *IEEE Transactions on Mobile Computing*, 10(1):3–15, January 2011.

[26] A. Tripathi, S. Gupta, and B. Chourasiya. Survey on Data Aggregation Techniques for Wireless Sensor Networks. *International Journal of Advanced Research in Computer and Communication Engineering*, 3, July 2014.

[27] A. Varga and R. Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[28] Waze. http://waze.com, Accessed Nov 22, 2016.