Mercury: A Geo-Aware Mobility-Centric Messaging System

Teja Kommineni
teja.kommineni@utah.edu

Kirk Webb
kwebb@cs.utah.edu

School of Computing
University of Utah
50 S. Central Campus Drive
Salt Lake City, UT - 84112

The target call for papers is MobiSys 2017:

`https://www.sigmobile.org/mobisys/2017/cfp.php`

ABSTRACT

Intelligent Transportation Systems require a communication conduit for the agents in the ecosystem. A variety of safety- and consumer-service messages are useful, even critical for coordinating the ongoing state of mobile endpoints. Communicating telemetry between these endpoints, along with data from other devices in the environment, allows them to work together and improve safety. Moreover, it is important for centralized services to have a global view and provide information based on this vantage point. To this end, we have developed Mercury, an end-to-end messaging system with lightweight state overhead and publish/subscribe functionality. Mercury is designed to flexibly integrate into the mobile network environment under which it is deployed. Mercury's centralized Broker allows for data aggregation services and data analysis. The Adapter tunes to the particular network environment, allowing Client Endpoints a pathway to communicate. Concurrent processing and event-driven design are used to minimize latency to meet demanding low-latency messaging requirements.

1. INTRODUCTION

Intelligent Transportation Systems (ITS) [30] is an emerging area that includes many ideas and mechanisms for improving the safety, quality of experience, and communication capabilities of commuters. One particularly important aspect of ITS is vehicle-to-vehicle and infrastructure-to-vehicle communication. Vehicle area networks (VANETs) [16] have been introduced to meet the challenges of mobile network actors with rapidly changing associations arising from dynamic proximity to infrastructure and other vehicles. Together with mobile networking a la the 3GPP evolved packet system [13], robust communication approaches involving hybrid LTE and 802.11p [17] with dynamic multi-hop clustering have been proposed [27, 29, 31]. Considering safety and other types of information exchange in an ITS, applications and events have been identified that should be supported [11, 24], along with constraints such as latency for delivery and scope (radius). Publish-subscribe systems provide a means to efficiently distribute messages and events between infrastructure and mobile actors in an ITS. Such pubsub systems can support peer-to-peer and infrastructure-sourced messages at scale [23]. While research has been done in the areas of VANET communication and mobility-aware publish/subscribe systems, holistic compositions of these technologies is lacking.

Transportation safety issues are also considerable. The total number of vehicle sales in the United States averaged 15.43 million from 1993 up through 2015. The year 2015 saw a surge in sales, which climbed to 17.76 million vehicles and is expected to rise with the improving economy [2]. Increasing vehicle populations will likely result in increasing accident rates. A total of 32,675 people died in motor vehicle crashes in 2014 [1]. We argue that technology advances should be used to mitigate increasing casualties and ensure safe journey. A key focus in this paper is using state-of-art 5G [3] networks to capture vehicle telemetry and pass along safety events to vehicles that are relevant to their surroundings.

The setting in which any transportation messaging system operates requires reliable, fast paced communication. However, the latency incurred by present solutions is not adequate for time-sensitive interactions (e.g., for real-time inter-vehicle notifications). Although there are architectures [27] that use the 802.11p communication paradigm for low latency, they lack good trust mechanisms. Such ad-hoc networks are also often unstable, requiring frequent communication path adjustments that result in losses and overhead. 3GPP-style mobile networks provide advantages in stability, reachability, and security.

This paper introduces Mercury; a integration of pubsub systems and client endpoint mechanisms for low-latency message transport. Part of the holistic vision that Mercury espouses is mobility-specific geographic areas of interest (AOI). These areas map to slices of physical locality that are relevant to particular events. For example, a traffic accident and resulting congestion are relevant to vehicles en route to the accident location, back past potential egresses to alternate routes. Mercury takes into account location-specific context to calculate the relevant dynamic set of vehicles for message transmission.

This paper makes the following contributions:

- A holistic end-to-end messaging system design.

  The design of Mercury highlights the mechanisms by which publish/subscribe systems and mobile endpoints can be realized in 4G or 5G mobile network systems. Broker aggregation services communicate through Adapters to Client Endpoints at the edge. Mercury's design accommodates flexible deployment scenarios.

- A prototype implementation and evaluation of the Mercury ITS messaging system.

  We implemented a prototype of the Mercury components and evaluated it on the PhantomNet [5] testbed. This prototype makes use of OpenEPC [12] version 5, including its simplified emulated radio access network (RAN) environmnet. We drive the evaluation of Mercury with our own mobility simulator, and report on the former's functional and latency characteristics.

The remainder of this paper is organized as follows: Section 2 covers background, including putting Mercury into context within related work, introducing mobile networking environment concepts, and describing the vehicular endpoing context. Section 3 provides an overview of the design goals and components of Mercury, while section 4 delves into the details of the design. Section 5 looks at how Mercury can be positioned within different mobile networking deployments. Section 6 briefly discusses the Mercury prototype implementation. We cover our vehicle simulator and provide an evaluation of the prototype in section 7. Section 8 discusses observations resulting from the design and implementation of Mercury, and where we think it should go next. Finally, section 9 concludes.

2. BACKGROUND

The mobile network ecosystem is complex, and host to a number of related efforts in providing context-specific messages between endpoints in an Intelligent Transportation System environment. Before we discuss the design and implementation of Mercury, we couch it in the context of prior work. In addition, we provide backround on the 3GPP 4G Evolved Packet System [13], which is an important and widely used mobile networking ecosystem that we position Mercury within.

2.1 Related Work

There has been plenty of attention paid to effecient and reliable delivery of messages within VANETs. Much of this focuses on multi-hop clustering and hybrid use of evolved packet system RAN (LTE). The VMaSC [27], MDMAC [29], and NHop [31] systems attempt to form stable mobile 802.11p clusters, using the LTE network to bridge between disconnected clusters. These solutions largely ignore the details of the mobile core network, glossing over questions of component placement and the resulting effects on latency. Moreover, these systems are complimentary to Mercury in that they can be used to reduce LTE resource contention and improve reliable transfer of messages. Such integration would however come at the cost of additional complexity, failure modes, and latency due to additional network path segments.

From the publish-subscribe perspective, there is a large volume of prior work on traditional pubsub mechanisms [16] [21] [18] [4]. This work is complementary to ours since it focuses on the useful aspects of pubsub, which we largely wish to reuse. There has also been work on pubsub systems focused on mobile endpoints. The MoPS [23] publish-subscribe system scales efficiently for large numbers of clients and deals well with changing broker association. Mercury could replace the Kafka system used in the prototype with MoPS to better target mobile endpoints. However, MoPS does not include the area of interest concept, which would continue to be handled by the Mercury broker. A paper by Pongthawornkamol et al [25] looks at pubsub in the context of ad hoc wireless networks. However, this work only performs simulations of mechanisms, and does not consider a larger top-down vantage point (important for coordination in a large ITS).

Location-aware messaging, or geo-routing, in vehicular networks has been studied fairly extensively [6]. Nevertheless, we find that most work has only proposed and simulated mechanisms. Furthermore, many studies look at ad hoc networks and fine-grained positioning of endpoints within vehicle clusters. While such mechanisms may be helpful for real-time collision avoidance, we argue that they tend to be overly complex and unnecessarily constrain the communication domain to clusters of endpoints versus a central system with a global view. The argument against centralized systems is frequently that mobile networks are overloaded. This may be true in instances of particularly high device concentration (e.g. music festivals), but we found no studies showing a general lack of available RAN resources outside of such crowded contexts. Furthermore, in an ITS environment, vehicle populations and anticipated growth could be used to properly size capacity.

The related work outline above focuses on particular aspects of messaging. To the best of our knowledge, no messaging system targeted at mobile endpoints simultaneously incorporates aspects we consider crucial to a holistic messaging platform for Intelligent Transportation Systems. Such a system should enable a global view of endpoints to facilitate coordinated decision-making with complete data. It should take advantage of mobile network environment mechanisms (e.g. eMBMS [20]) to reduce overhead and latency. It should consider the placement of components within the mobile network and the impact of this placement. Finally, an ITS messaging system should provide a location-aware addressing mechanism. The Mercury messaging system is designed with all of these aspects in mind.

2.2 Mobile Networking Ecosystem

Mercury is primarily framed in the context of the 3GPP 4G and emerging 5G mobile networking architectures. Therefore, we provide some background on these systems. The vast majority of mobile carriers utilize these Evolved Packet Systems (EPS), making them an especially relevant environment in which to operate a mobile messaging system. Note that Mercury is also amenable to other mobility-friendly network architectures, such as MobilityFirst [26], but we focus the discussion in this paper on the 3GPP EPS. The 4G system has been in active deployment since 2008 [9], and has undergone a number of revisions. Also in play in some deployment scenarios (see section 5) are software defined infrastructure concepts that are expected to be prominent components in the upcoming 5G EPS [3].

The 4G EPS includes the following key service functions relevant to Mercury: Mobility Managment Entity (MME), Home Subscriber Service (HSS), Serving Gateway (SGW), Packet Data Network Gateway (PDN-GW or more commonly PGW), evolved NodeB (eNodeB), and User Equipment (UE). We will briefly describe the role of each of these components, and their relationships with one another and with Mercury. The Mercury architecture diagram in figure 1 shows Mercury components in the context of a 4G EPS.

We will briefly cover the 4G components next. **User Equipment (UE)** typically refers to end user devices such as mobile phones, tablets, and 4G radio equipped laptops. The Mercury Endpoint component runs on these. The **Mobility Management Entity (MME)** is the 4G control plane function responsible for tracking the live (dynamic) session state for UEs. The **Home Subscriber Service (HSS)** is essentially a database of user (subscriber) information. The **Serving Gateway (SGW)** is the first data tunnel anchor point that UE sessions connect through (GTP tunnels). The **Packet Data Network Gateway (PGW)** act as the egress point for a large number of UE data bearers (GTP tunnels); they fan out to multiple SGWs. **Evolved NodeB (eNodeB)** devices are the wireless access points of the 4G EPS. They bridge the radio access network (RAN) through which the mobile endpoints (UEs) directly communicate with the evolved packet core (EPC). GTP tunnels are established for each UE between the eNodeB it is associated with and an upstream SGW. The eNodeB also initiates session setup and default data bearer establishment when UEs attach, acting as a proxy for UE to MME control plane signalling. eNodeBs covering adjacent cells coordinate through the MME and possibly with one another to accomplish handover as endpoints move.

2.3 The Vehicle Environment

The environment of a vehicle endpoint is highly relevant context for drivers and passengers. We argue that traffic, safety issues, and proximity to resources are key to decision making in this setting. Therefore, related telemetry such as position, speed, and direction should be relayed for analysis. Vehicle *position* is almost universally measured using GPS [22]. Although known radio access points can be used for rough localization, such positions are often too coarse-grained. In fact, a better system would make use of computer vision and/or static environmental sensors to extract lane position, relative distances to other vehicles, etc. Our work uses GPS coordinates and leaves more fine-grained placement techniques to future work. *Speed* is to be collected by vehicle speedometers, or through monitoring of accelerometers such as are found in most smart phones. Finally, to acquire *direction*, digital compass readings can be gathered from smart phones and are also available in many onboard vehicle systems. We expect that environmental sensors will grow in sophistication and accuracy through the proliferation of self-driving cars and ITS.

3. DESIGN OVERVIEW

Delivering messages, important and casual, to mobile users and endpoints that are interested in them is the overarching premise of this work. Our vision for realizing an end-to-end mobile message delivery service design principles revolve around four central goals:

- Relevant content

  The service should provide mechanisms to target groups of endpoints which have explicit or implied interest in messages. A message may be important because an end user specifically asked for the content based on its attributes (nearby gas prices). Alternatively, a message may be deemed relevant for the endpoint because it is related to an emergent event (vehicle accident ahead).

- Robust, low overhead, low latency communication

  Message intent drives content delivery requirements. For example, different types of emergency service messages have been identified for VANETs, each having distinct latency requirements [11]. The service should strive to minimize latency to provide on-time delivery with headroom for outlier delays. Messages should be categoriezed according to their relative importance and processed accordingly (e.g., emergency info before consumer content).

- Flexible deployment

  Adoption of the service is bolstered by adaptability to different mobile networking environments. Such accomodation allows for deployment into LTE networks with different geographic EPC service placements and degrees of maleability. The service should allow for centralized metropolitan area integration (CloudRAN) and distributed edge deployment (peer-to-peer eNodeB).

- Reuse of effective technologies

  It is our contention that an end-to-end service should not supplant existing mechanisms useful for achieveing its composition. Indeed, it is counterproductive to introduce new service components that induce unnecessary changes and capital investments. The messaging service should strive to work alongside existing mobile network protocols and services. Only where existing mechanisms do not provide key functionality or do not give adequate service levels should changes be introduced. Such changes should be as minimal and transparent as possible to foster compatibility and ease of adoption. On the other hand, considering less constrained future mobile network architectures [3, 28] is also important.

We next describe the components of the Mercury messaging system, along with how they fit into the 3GPP 4G mobile networking architecture.

3.1 Mercury Components

Mercury is comprised of four essential components: message broker, publish/subscribe system, message adapter, and endpoints. These components are visible in the architecture diagram in figure 1. This diagram shows the Mercury components in the context of a 4G mobile network (the latter will be described after Mercury is covered).

3.1.1 Message Broker

Mercury's message broker is the brain center of the messaging system. It processes all incoming messages and sends these to relevant endpoints (via Kafka). The broker calculates Areas of Interest (see section 4.2) for certain message types (e.g., emergency notifications). Data analysis (aggregated decisions, client statistics, etc.) occur at the broker. It does not concern itself with how to get the messages to the target endpoint(s); that job falls to the Mercury message adapter.

3.1.2 Publish/Subscribe System

Mercury needs a pubsub system that can serve a high number of events with low latency. This is because of the environment in which it operates. We have many agents/endpoints at any given point that interact with the system and for them to effectively respond to a situation we need to dissipate the messages quickly. The mobility of the clients creates the potential for intermitent loss of connectivity with the pubsub system which has to be re-established. Among the options considered, Apache Kafka had the best combination of low-latency and message queueing for recovering lost messages.

At a high-level Kafka gives the following guarantees: Messages sent by a producer to a particular topic partition will be appended in the order they are sent. A consumer instance sees records in the order they are stored in the log. For a topic with replication factor N, kafka will tolerate up to N-1 server failures without losing any records committed to the log.

3.1.3 Message Adapter

The Mercury message adapter is the conduit through which pubsub messages flow through to endpoints. The adapter coordinates sessions with client endpoints. It forwards client reports and other content to and from the pubsub. It also maintains pubsub topic subscriptions for the endpoints. The prototype implementation described in this paper targets the 4G EPC, but it is possible to target other mobile networking architectures with different adapters.

3.1.4 Endpoints

Mercury endpoints are the producers and consumers of most content in the messaging system. Client Endpoints run as applications on ITS-equipped vehicles. They report in with telemetry, such as their current position and speed. Client endpoints also subscribe to particular consumer messages identified by specified attributes.

Service Endpoints may also be centralized services that connect to the pubsub, running at the same location as the Broker, and using the Broker's services. Examples include emergency notification processors, and consumer information aggregators (gas prices).

3.2 Mercury Messages

Messages are the principle and only communication mechanism in Mercury. There are two high-level flavors: control and content. Control messages include session handling (between broker and endpoints), broker to message adapter communication, and subscription management (endpoint to pubsub/broker). Content messages are those relevant to applications running on endpoints, such as emergency alerts and consumer information (e.g. gas prices). Periodic session maintenance messages are sent between the mobile endpoints and the Mercury Adapter to update status (location) and track liveness.

Content messages use structured message attributes to signal category information. This allows content producer and consumer endpoints to steer messages to one another based on interests. Most messages (including client reports) flow through the pubsub system to the broker. This allows the Broker to perform data aggregation and analysis (triggers based on thresholds, for example).

Message destination addressing in Mercury includes endpoint, content, and area of interest targetting. Endpoint addresses are primarily used for session setup and teardown. Content-specific addressing makes use of subscription attributes to deliver messages. AOI addresses are unique to the mobile environment. AOI bounds are computed by the broker. These bounds form the destination address. Message adapters check bounds to determine if their downstream area coverage is relevant before passing along, and endpoints check their position relative to the bounds. In this way, Mercury allows messages to be "area multicasted." As we discuss in section 4, our prototype restricts itself to simple bounding models (radius). Complex spatial reasoning for improving relevancy is left for future work.

4. DESIGN DETAILS

Each Mercury component runs as a self-contained applica-

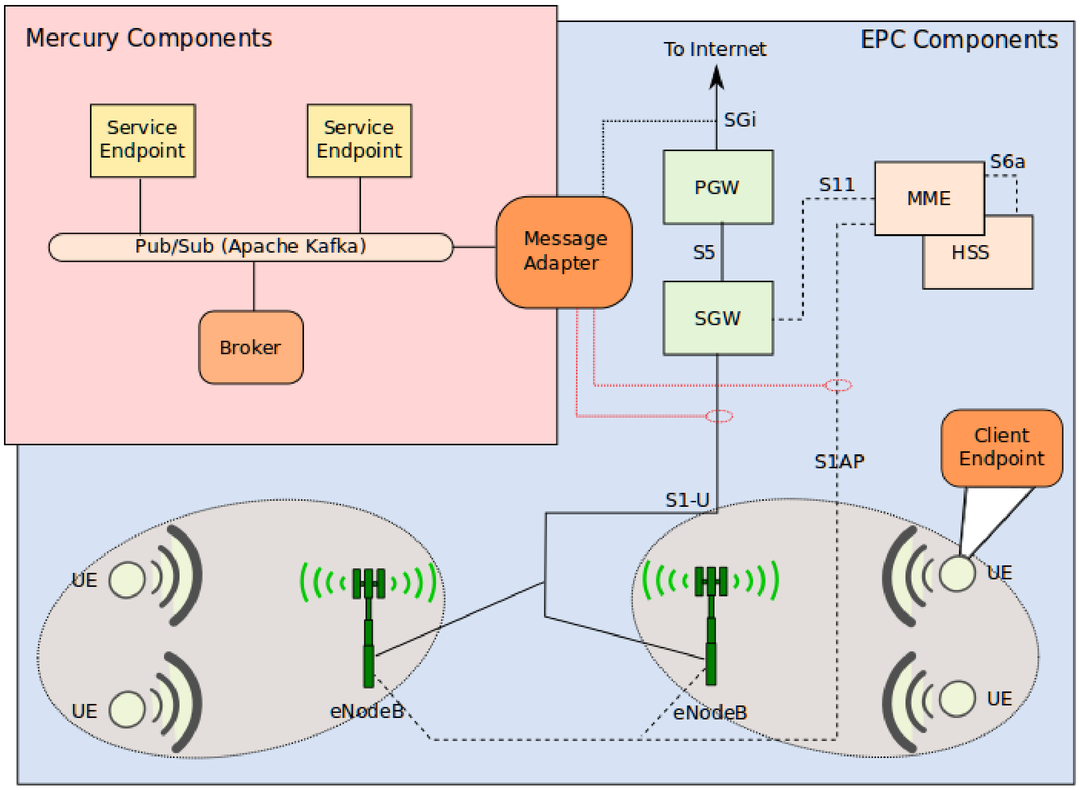

Figure 1: Mercury architecture diagram (in 4G mobile network context)

tion. The Broker and Adapter(s) communicate over unmodified Kafka. Client Endpoints communicate with the Adapter over UDP. Client-side applications using Mercury communicate over the local loopback with the Client Endpoint process. Service Endpoints use Kafka to communicate with the Broker and talk to Clients via Adapter(s).

4.1 The messages and events

The Mercury Broker we have implemented handles different types of events. Emergency Event, This is an event that is published by a vehicle when it senses an emergency situation. Any message indicating such an event is immediately processed by the Mercury broker and sends back an area of interest (AOI) that has to be alerted for this event. Moving Objects Event, This event indicates that there is an object that is crossing a road. For such events, whenever a predicate calculated by scheduler evaluates to true we inform the vehicles in the AOI calculated. There are other types of events such as Collision, Obstacle, Congestion and Blocked. All, these events are handled similarly but they differ in two things. One is the predicate function and the second is the frequency at which the schedulers run. Our system also supports an other type of event called Area Of Interest. This is different from the aoi which is calculated by Mercury broker. In this type of event the user is interested in knowing about the different events happening at a particular location. Whenever an event of type area Of interest is received by the Mercury broker. It interacts with different handlers and determines the bin for each event type in which the requested aoi falls into. Then runs a predicate on these bins and learns about the traffic conditions in that area which is communicated back to the user.

Messages must fit into a single UDP packet (maximum of 64K). Each is self contained, with required identifiers, addresses, and message payload. There are two top-level types of messages in the Mercury system: session and pubsub. Session messages are transmitted between Client Endpoints and associated Adapters. Pubsub messages can essentially be exchanged between any Mercury components. Table 4.1 shows the fields present in Mercury Session and PubSub messages. Each Mercury Address contains a destination and source address. The source address can be an Adapter, a specific Client, a Service, or the Broker. The destination address can be any of those listed for source, and additionally may be a broadcast (all clients, or Area of Interest geo-address). Each message contains key/value pairs in the payload which are interpreted by specific applications. For example, emergency alerts published by the Broker may be consumed by an alert display application at the Client Endpoint. Client Report message contain mobile telementry information: GPS location, direction, and speed. These reports have semantic meaning to Mercury, and so are tracked by the Client Endpoint, Adapter, and Broker components. In our prototype implementation, the Broker has had emergency event mes-

Field	Description
UUID	Unique identifier for message.
Session ID	Identifies session between Client and Adapter.
Message Type	Distinguishes intent (to/from Broker, PubSub, etc.)
Source Address	Client ID, Service ID, Broker, or Adapter.
Dest Address	Client ID, Service ID, Broker, Adapter, or AOI.
Topic	Indicates topic message goes to or came from (PubSub only).
KeyVal	Generic key-value *pairs*. Telemetry and other data is encoded here.

Table 1: Message fields found inside individual Mercury messages.

sage handling folded into it's logic. Therefore, the prototype Broker also interprets safety messages sent via pubsub topics such as "Collision" and "Object_Hazard". We envision that such message handling would ultimately be moved to an external Service Endpoint application.

4.2 Areas of Interest

Mercury uses Area of Interest geo-addressing to target specific sets of Client Endpoints. When the Broker wishes to send a message that is specific to a location context (e.g., a warning triggered from multiple collision reports), it embeds the desired geo-address as the destination, and sends this along with the message toward the Clients. This message is picked up from the pubsub by the Adapters. These check their coverage area and forward the message along to Clients within the defined geo-address that they serve (or drop if there is no overlap). When an AOI-addressed message is received by a Client Endpoint, it checks its current location to determine if the message applies, and processes if so.

4.3 Mercury broker

Mercury Broker handles events in two steps: In the first step events published are passed on to the respective handlers where we have a filter that determines the different locations from which the feeds are coming in for that specific event.This will give us the messages published at different locations for an event.Each location can be considered as a bin. We maintain for each bin the count of messages published, the center point and the radius for that bin. Whenever a message comes into a bin it carries along the coordinates from which this event has been published. We apply a simple mean with the existing center point to determine the new center point. Thus, at every instance we keep updating the center point and radius.In the second step we have schedulers for each event that get triggered at equal intervals based on the event type. These schedulers empty the bins of their respective event and run a predicate on them which is a simple Boolean formula that checks for a condition. A predicate is usually defined over some aggregation function expressed on messages; when the predicate evaluates to true, the Mercury broker publishes an event to the system with the center point and the radius which we have calculated for that bin. These are used to determine the area of interest(AOI), AOI is the region that is determined by the Mercury broker and constitutes all the clients that have to be alerted about a situation.

4.4 Pubsub service

Within Mercury we require a publish-subscribe system for communication between different entities. This requirement directly arises from the need for each vehicle to send and listen to particular types of messages. Pubsub systems are well-suited for these activities as we can publish and subscribe to different events. In our system components such as Client Endpoints and the Message Broker publish and subscribe to events. Given this communication pattern, we developed the set of requirments we need in a publish-subscribe mechanism and surveyed available options.

As mentioned in section 3, we selected Apache Kafka as our pubsub component. It is a distributed streaming platform which lets users to publish and subscribe to streams of records. Kafka is run as a cluster on one or more servers. The Kafka cluster stores streams of records in categories called topics. Each record consists of a key, a value, and a timestamp. A topic is a category or feed name to which records are published. In our case it is the event type. Topics in Kafka can be multi-subscribed; that is, a topic can have one or more consumers that subscribe to the data written to it. In our implementation all the values written to a topic are listened by Message Broker. But, we could extend this to different service end points listening on the same topic.

For each topic, the Kafka cluster maintains a partitioned log. Each partition is an ordered, immutable sequence of records that is continually appended to a structured commit log. The records in the partitions are each assigned a sequential id number called the offset that uniquely identifies each record within the partition. In our system we create a single partition for each topic. As within a partition order is maintained we can traverse through records using the offset of the record. Each partition is replicated across a configurable number of servers for fault tolerance. The Kafka cluster retains all published records whether or not they have been consumed using a configurable retention period.

4.5 Mobile network messaging adapter

The Mercury Adapter follows a multi-threaded, event-driven execution pattern. It is highly modular, with clear separation between client state tracking, network-specific address mapping, and transport functionality. This allows new modes of core network communication to be added alongside (or in place of) existing mechanisms. Different threads of execution handle pubsub, UDP communication, and scheduled tasks (e.g., session checking); all are coordinated through the main thread.

Mercury is designed so that any number of Adapters can connect to the Kafka system. Each will send client telemetry reports and pubsub messages along. A single broker message sent via pubsub reaches all connected Adapters, which filter as necessary. For example, an Adapter will drop messages with Area of Interest destinations that do not overlap with its coverage area. These coverage areas are configured by the operator.

Endpoint identification/tracking The Adapter implements client session tracking. When a client comes online, it goes through a lightweight handshake with the Adapter. An INIT message is sent to the adapter, which immediately responds with an ACKNOWLEDGE message. The client includes a telemetry report in its INIT message so that the Adapter and Broker have immediate knowledge of its current state. Each client is provided with a session ID token that must be used in subsequent messages. The Adapter sends out periodic HEARTBEAT messages to clients so that they can measure liveness in the absence of other messages.

The Adapter maintains a small amount of state for each client. This includes its unique ID, current mobile telemetry (no history), session ID, and simple statistics (timestamp for last message received, message count). The Adapter also maintains a mapping from the client's ID to its mobile core network address. Changes in this address are automatically detected and the mapping updated. The Adapter treats periodic client reports as heartbeats, and resets corresponding individual timers as these are received. Sessions are pruned after not receiving a report for three times the reporting interval, or after receiving an explicit CLOSE message from the client.

Unicast UDP transport In the prototype implementation, we use UDP unicast packets for all communication between Adapter and Client Endpoints. Ideally we would use eMBMS broadcasts for AOI and broadcast (e.g. heartbeat) messages, and we plan to do so in future work. Each UDP packet is a self-contained Mercury message. The contents are encoded using Google's Protocol Buffers [7] for performance and space efficiency. For AOI destination addresses, the Adapter calculates the set of clients to transmit a message to and unicasts to each individually. For broadcasts, it transmits to all clients with valid sessions.

Pubsub passthrough Messages from established clients that are intended for the pubsub system are directly published to the indicated topic by the Adapter. Likewise, messages received from the Broker via pubsub are sent to specific endpoints or broadcast to all clients handled by the Adapter (as appropriate).

4.6 Client Endpoint mechanisms

The Mercury Client Endpoint connects to the Adapter over UDP. It establishes a session using a simple handshake; an INIT message is sent to the adapter, which it replies to with an acknowledgement. The client will continue trying to connect until it receives the acknowledgement, backing off linearly with random jitter to prevent synchronization with other clients (prevent in-cast overload). The client will also try to reestablish its session if it does not receive a heartbeat from the Adapter after three heartbeat intervals (tunable). Messages are encoded using Google's Protocol Buffers.

The client gathers movement telemetry via the local sensors of the device it is running on and sends these along periodically as report messages to the Adapter. As mentioned, the Adapter uses these reports to measure the liveness of the connected clients. Some random jitter is added initially to the report interval to again avoid synchronization with other clients.

The client listens over the loopback interface for connections from local applications. Similar to what the Adapter does for its Client Endpoints, a Client Endpoint relays subscription requests and messages from applications through to the pubsub/broker. Protocol Buffers are also used between clients and applications for encoding the messages.

5. DEPLOYMENT SCENARIOS

A key aspect of the Mercury design is adaptability to different mobile network deployments and architectures. Endpoints, broker, and pubsub components are all mobile-environment agnostic. Mobile endpoint identity is specific to the Mercury messaging system. One of the message adapter's primary functions is to separate the concerns of the messaging system with the routing of messages inside the mobile network. Mercury requires a low-latency vantage point near the mobile network edge. Message adapters, pubsub, and broker all need to reside at this vantage point. Such proximity allows for message delivery to meet timing requirements. Distance to endpoints incurs a tradeoff between convenience of deployment and ability to meet timing needs. We show deployment versus distance/latency requirements in table 5.

Latency Tolerance	**Examples**	**Ref**
High (> 1 second)	Consumer info, Congestion	[11, 24]
Medium (< 50 msec)	Lane change assistance	[?]
Low (< 10 msec)	Vehicle control loop	[15]

Table 2: Latenct tolerances for different types of vehicular environment messages.

The message adapter can utilize different techniques to interface with a mobile network deployment. In the simplest case, it can interact with endpoints via routed network addresses (e.g. IPv4 or IPv6). Alternatively, it may transparently interpose on encapsulated communication channels to inject/extract messages (e.g. GTP bearers). Another option is for the message adapter to operate in concert with

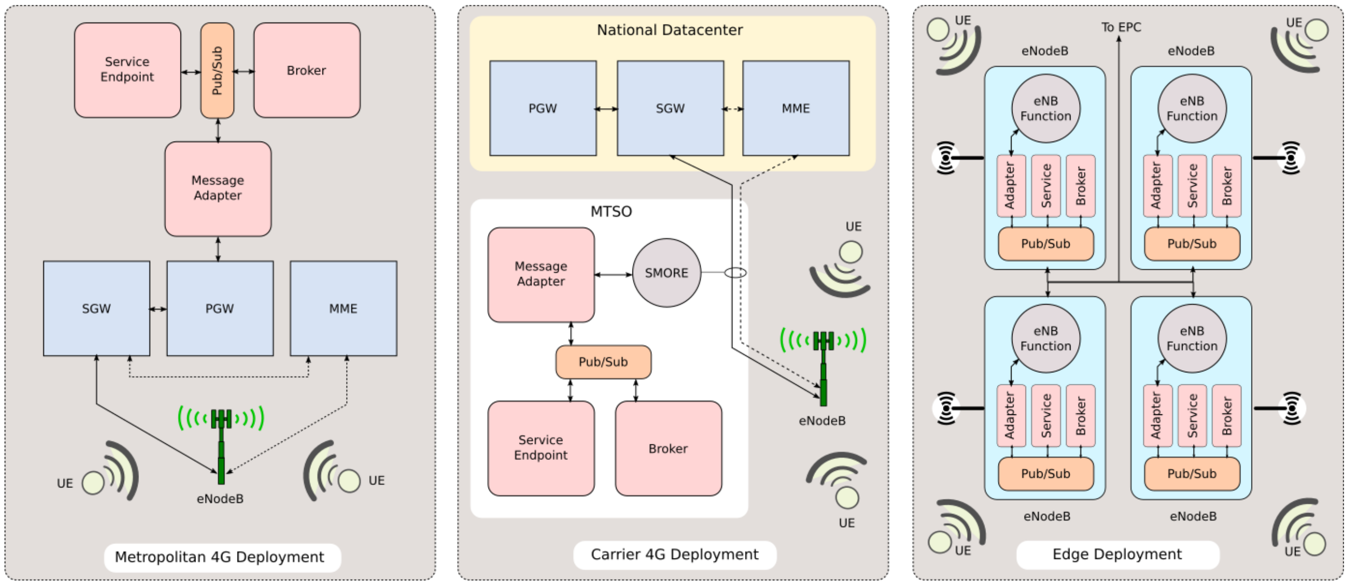

Figure 2: Mercury deployment scenarios.

the wireless access point right at the edge. It could do this by transparently interposing on the access point's data plane uplink, or via an integrated side channel added into the access point software. We next present three plausible deployment scenarios, each of which is shown in figure 2.

5.1 Metropolitan area 4G deployment

Consider a city investing in Intelligent Transportation System infrastructure. Resources include a data center within the city and wireless coverage via eNodeBs (particularly along major vehicle corridors). The 4G EPC components reside in the local data center, including the PGW. Given a maximum cabling distance from the data center to any eNodeB of around 100 kilometers, one way speed of light latency is bounded to less than 5 milliseconds. LTE one-way wireless first-hop target latency is 5 ms, but may be more like 10 ms [19] due to UE and eNodeB processing overhead. Further switch/router hops within the data center should not appreciably add to the latency. Egress through SGW/PGW can add another 5 ms of one-way delay. Total mobile network infrastructure RTT is thus bounded to 40 ms. This leaves 10 ms for Mercury components to process messages for low-latency vehicle-to-vehicle coordination (within 50 ms).

In this deployment, the Mercury message adapter, pubsub, and broker all reside alongside the PGW in the local metro data center. The message adapter and mobile endpoints communicate using native network addresses (i.e. IPv4 or IPv6). Mercury components communicate via the data center pubsub deployment. The endpoints can obtain the address of their associated message adapter using DNS, DHCP options, or multicast query.

5.2 Existing mobile carrier 4G network deployment

In an existing 4G mobile carrier network, there may not be a centralized upstream network egress location in a particular metro area of interest. Further, PGW nodes are typically deployed at a handful of national data centers. Total one-way latency to egress through a PGW in such deployments is often over 50 ms [14, Chapter 7]. Installing Mercury at these national data centers would impair its ability to meet SLAs for some classes of low-latency messages (see table 5). Many mobile carriers, however, concentrate regional connectivity at a Mobile Telephone Switching Office (MTSO). Typical one-way latency from eNodeB to MTSO is 10 ms [10]. With LTE RAN latency added, the one way latency between MTSO and UE is about 20 ms.

Deployed into a MTSO, Mercury message adapters can interpose on GTP data bearers for UEs in the same way that SMORE [10] does. Alternatively, lower-capacity (thus cheaper) combined SGW/PGW functions may be deployed into the MTSO. This would allow the provider to setup dedicated bearers for services such as Mercury in a MTSO location. Using these options, Mercury can operate with slightly higher RTT compared to metropolitan data center deployments. However, the additional latency may not meet the latency requirements for some UE-to-UE (vehicle-to-vehicle) applications.

5.3 Mobile edge 4G/5G deployments

Mercury can be altered to run at the network edge, on the mobile wireless access points. If a deployment includes the ability for running services on the eNodeB (or equivalent) access points, then Mercury can operate using peer-to-peer semantics. All components of Mercury would run on each eNodeB, and some service endpoints as well. The pubsub is extended to form a connectivity mesh between adjacent eNodeBs. Mercury's scope at each eNodeB is smaller, and so the amount of processing is reduced. When a message destination is an area of interest, the local Mercury broker determines whether parts of this area lie outside of its coverage. If so, it sends such messages along (via pubsub mesh) to neighboring eNodeBs. These handle their portion of the endpoints in the AOI. Arbitrary consumer content messages are also sent along to potential subscribers via the pubsub mesh. This deployment scenario allows for very low-latency messaging between nodes connected via the same eNodeB (potentially under 20 ms RTT considering RAN latency). Inter-eNodeB communication latency depends on the communication path between eNodeBs. This could be only a handful of milliseconds if there are metro-area-local paths.

A second realization of this deployment scenario is via a 5G CloudRAN environment [8]. Here eNodeBs are split into remote radio heads (RRH) connected to base band units (BBUs). The latter are located in nearby compute aggregation locations; RRH and BBU functions can only be about 25 km apart in order to maintain the closed control loop between them. BBU functionality in a compute location serves multiple RRH units. Mercury components would be deployed into the compute locations in this case. Multiple such compute locations may serve an area. Mercury could be deployed to each with a peer-to-peer pubsub mesh setup between them. Similar very low latency messaging is possible in this scenario with a Mercury message adapter working with the BBUs at its location. An additional advantage is that the very low latency messaging extends to the (larger) area covered by all associated RRH units.

6. IMPLEMENTATION

We next briefly cover some of the essential implementation details of the Mercury prototype. Section 4 discusses our selection of Kafka for the pubsub component. Google's Protocol Buffers were used for efficient on-the-wire binary encoding of messages. Such encoding is used on all non-pubsub communication paths. The Mercury prototype was written in python in about 2,200 lines of code. We chose Python because it was easy to rapidly produce a working system in this high-level langauge, and because it has good support for parallel processing. Integrations exist for Python for the other technologies used. Concurrent threads are employed to capture and buffer incoming messages from both the pubsub and UDP point-to-point channels. An internal scheduler was implemented to perform periodic tasks, and supports randomized offsets.

The prototype implementation targets the centralized PGW deployement scenario where the core Mercury components (Adapter, Broker, and pubsub) are positioned immediately on the other side of a standard 3GPP data bearer egress point. This deployement scenario affords low latency overhead when EPC core components are located in a datacenter near the network edge. We covered this deployement scenario in section 5.1. Note that we did not take advantage of any specific aspects of the mobile environment in our prototype, which is left as future work.

7. EVALUATION

We have performed functional and performance (latency) evaluations of our Mercury prototype. The components operate functionally as expected, tracking location and responding to area of interest events where they are verified to be in the defined region. Micro-benchmarks show that the system can operate down to 10ms RTT from baseline single message testing, with a slowly growing response that begins to ramp up more significantly once the input message rate overtakes the maximum processing rate of the components. We believe that introducing more concurrency will both lower the nominal RTT even for moderate incoming message rates, and push out the maximum processing rate.

7.1 Vehicle Mobility Simulator

To verify Mercury, we have written a simulator that mimics real-time vehicular movement. The Simulator provides two main functionalities. Firstly, to add vehicles to the system. Secondly, to emulate an event at a given location. Mercury Simulator has been implemented as follows: We define an area with in the co-ordinate system in which the vehicles keep moving. This area depends upon the number of vehicles a user wants to initialize. Each vehicle is associated with a unique identifier, x and y co-ordinates that indicate its location in the system and speed at which it is moving. Each vehicle on being initialized binds themselves to a port and periodically sends reports to the Message Adapter about its location. While sending a report, we take the vehicles present speed and its location to determine its new location. Which in turn is updated on the vehicle and communicated to the Message Adapter. This is done for each vehicle and thus we have a simulated vehicle environment. One point to be noted here is since it is a bounded region we take care that all the vehicles move within this enclosed area.

Using simulator, we can emulate an event by specifying the (x,y,r) 3-tuple. All the vehicles enclosed within the region covered by a circle centered at (x,y) with radius r sense this event. We have implemented this as follows: Whenever a simulate event function is invoked with 3-tuple. Since we have list of all the vehicles within the system for each vehicle we verify if it falls within the region specified by the 3-tuple. If yes, we send a message to the Message Adapter on the port that vehicle is bound.

7.2 Functional Evaluation - Area of Interest

Using the Message Simulator we have verified the functionality of the system. To do this we have initially set up ten vehicles in the simulated ecosystem. Then mocked an event collision at location (5,2) with a radius of 2 as in figure 3. All the vehicles within this region have sensed the event and published a message to the Mercury. We have taken care that the number of vehicles that sense the event and publish the messages to Message Adapter will be greater than the threshold needed for the Message Broker to trigger an Area of Interest message. The AOI message from Message Broker is represented as bigger circle in the figure 4.

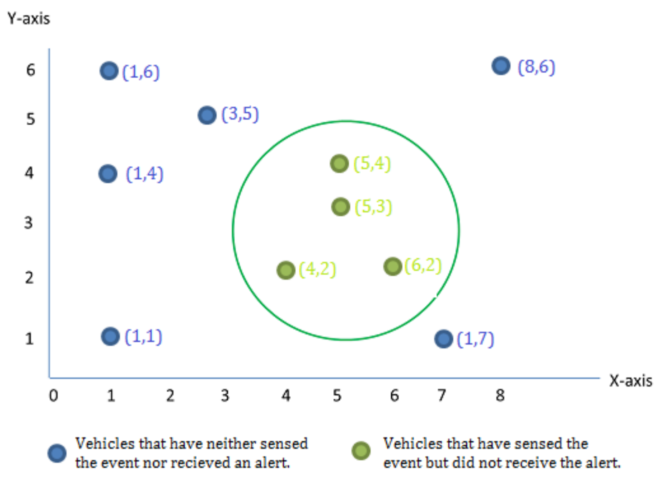

Figure 3: Mercury event simulation.

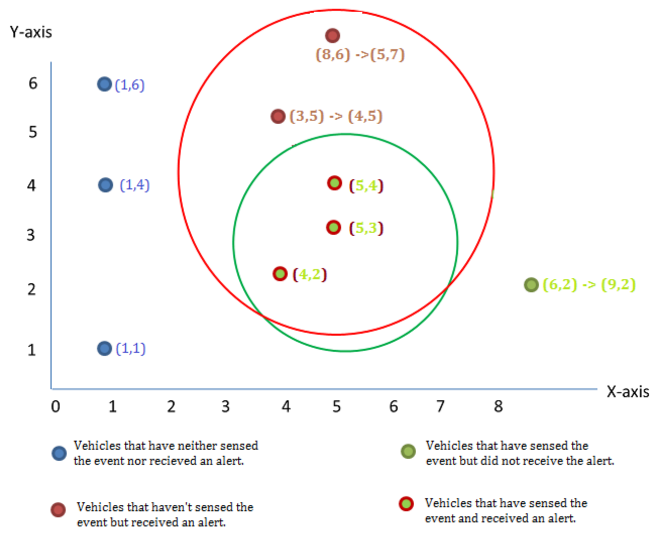

Figure 4: Mercury Area Of Interest.

By the time AOI message from Message Broker has reached the vehicles, we have observed the vehicle which has sensed the event at location (6,2) has moved away from the region. This vehicle didn't receive any alert from Mercury. There were two other vehicles that are at location (3,5) and (8,6) and have moved into the AOI calculated by Message Broker both these vehicles received the message. And the three of four vehicles which have sensed the event were still falling within the Message Broker's AOI and have received an alert.

Through this experiment we have verified that all the vehicles within the AOI of Message Broker have received an alert irrespective of sensing the event and those vehicles which are out of the AOI did not receive it.

7.3 Latency Micro-benchmark

In order to assess the latency response of Mercury under load, we performed a micro-benchmark involving sending messages at progressively higher rates, end-to-end. To do this, we developed an client-side application, "echorate", that sends timestamped messages to the Broker. The Broker responds to these immediately, embedding the timestamp provided (similar to ICMP ping). The "echorate" app takes rate and total message count as arguments. For this evaluation, we took baseline measurements at 1 message per second over a one minute period, then at rates of 10 messages/second up through 1,000 messages/second at roughly logarithmic rate increments. For each increment we increase the count such that the total test target run time is 10 seconds.

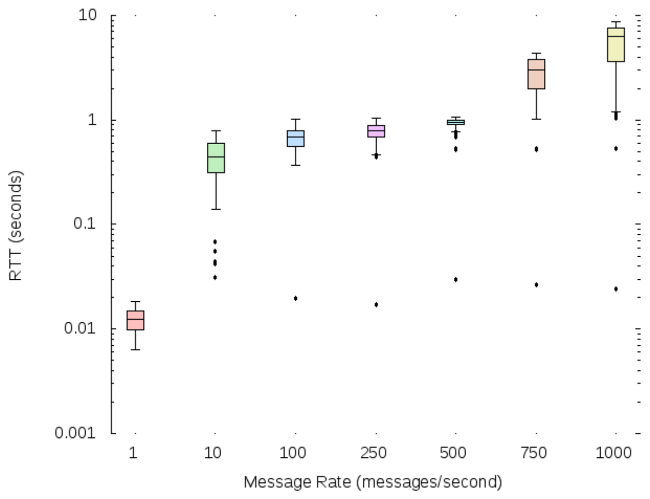

Figure 5: Round trip time (RTT) as a function of message rate.

The results are presented as a box-and-whisker plot in figure 5. Message rate appears as discrete values along the x-axis. Round trip time is plotted using a log scale on the y-axis. The dots indicate outliers. This graph shows that the minimum RTT our prototype manages is about 10 milliseconds. However, even at a rate of 10 messages/second, the RTT jumps by an order of magnitude. The response stays fairly flat at around 1 second through a rate of 500 messages/second. Somewhere between this latter rate and 750 messages/second, the rate begins a linear climb. This trend is a result of the Mercury components falling behind in message processing. As the quartile spread indicates at these higher rates, the RTT slowly creeps up as message processing continues to lag behind the incoming message rate.

These results indicate that the prototype needs additional work to improve its throughput and responsiveness in order to meet the demand of realistic workloads while maintaining latency-sensitive message requirements. Our prototype already uses parallel processing inside individual components to queue incoming messages. However, message processing and sending is single-threaded. We believe that the key to increasing throughput is to parallelize these tasks. Decreasing the RTT baseline message rate can be accomplished by reimplementing Mercury in a lower-level language where individual processing time contributions are easier to manage and processing overhead is generally lower. Using real-time scheduler directives would likely also help. We leave such optimizations as future work.

8. DISCUSSION

The design of Mercury, the possible deployment scenarios it can accommodate and the evaluation of prototype implementation of Mercury clearly suggest that Mercury is a fit for both present and future Intelligent Transportation Systems. Yet, there are few areas where we can further improve to make Mercury more appealing.

At present, Mercury uses AOI as a main primitive in delivering messages to the clients. Mercury Broker determines the AOI and this is processed by the Message Adapter to deliver messages to respective Clients. The AOI that we are implementing or envisioned in this paper is based on the circle centered at a point with a given radius. Clearly, when we are using the circular region as metrics there could be vehicles which are not affected by the event yet receive the alert message from Mercury because they fall inside this region. This is where we think AOI calculation should also add another dimension to its measurement, namely direction. This comes from the basic concept that an event happening on one side of the road may or may not affect the vehicles moving on the other side of the road. Looking at the directions we can also detect collisions between vehicles using the reports they send to the system. Hence, this is an interesting future work area.

Presently, the Message Adapter doesn't implement any significant security measures when dealing with the messages that are moving in and out of the system. We only check if the sender of the message has a session with Mercury and then forward it to the Mercury Broker. The verification that is done here can be further supplemented by other methods such as self-certifying endpoint IDs and signed messages (to prevent spoofing), and data analysis at the Broker looking for anomalies (to prevent gaming). Without such protections, miscreants could exploit and/or spoof messages without sensing an event or in any other anticipated way that could generate profits to them or cause harm to others.

The Message Broker receives all sensed events from the Message Adapter and performs computation on them to determine AOI. Here, we can see that Message Broker depends on the incoming messages to figure out whether an incident/event is happening. It doesn't have any other way to know about an event except for the incoming messages. But, if we have a close look at Mercury we also have another source of information; the reports that are periodically sent by vehicles. We can make the Message Broker intelligent enough to look at these reports and detect an event. It could further decrease the latency in the system and we could make more informed decisions a step before a threshold of vehicles get to sense this event and report it to the system. For example, all vehicles which are in a region experiencing congestion move at a slow speed. Looking at the reports from vehicles, we could detect the slowing down of vehicles in any region and send a congestion alert to relevant vehicles (via AOI).

Section 5 mentions a handful of different deployment scenarios. We would like to extend Mercury into these different scenarios and measure its performance. In particular, we would like to make use of eMBMS for broadcasting AOI and heartbeat messages to Clients. We would also like to explore the edge deployment scenario. This scenario has the greatest potential for minimizing latency, and also has interesting fault tolerance properties and challenges. Keeping a mesh of agents running at each eNodeB bypasses issues in the core network, but means manging message routes and maintaining healthy paths.

Finally, there are a number of things we can do to improve the performance of Mercury. We could increase the performance by increasing parallelism in the system such as scaling Message Adapter and Message Broker. We could also take advantage of the consumer groups concept in the Apache Kafka that help us in instantiating multiple consumers on a single topic. This way all the messages that come to a single topic can also be processed by multiple consumer instances giving us high throughput. We can also take advantage of the retention period in the Kafka to go back in time and receive any alert if missed. This could probably be an interesting option to look at because vehicles may tend to lose connection to intermittently and when they establish connection again with the system they might be interested in listening those events that it missed.

9. CONCLUSION

We have presented Mercury, an end-to end mobility centric messaging system that improves safety within the vehicular environment. We have looked at the architecture and different components of Mercury. We have discussed the different deployment scenarios and how Mercury can fit in each of them. We also analyzed the need of a pubsub system in this environment and our choice to use Kafka for the implementation. We evaluated our prototype Mercury implementation in a 4G deployment context. A vehicle simulator was written to test functional correctness, and a synthetic echo application was used to evaluate rount trip time latency performance. We conclude that the prototype is functionally accurate, but requires additional parallelism and tuning to meet latency requirements for realistic workloads. We have also considered how adding a direction based metric to this system would make it more precise and how we can incorporate more trust into Mercury so that it is not misused.

10. REFERENCES

[1] Insurance institute for highway safety. `http://www.iihs.org/iihs/topics/t/general-statistics/fatalityfacts/overview-of-fatality-facts`, 2015.

[2] Trading economics. `http://www.tradingeconomics.com/united-states/total-vehicle-sales`, 2015.

[3] 5GPPP. 5G Vision - The 5G Infrastructure Public Private Partnership: the next generation of communication networks and services. `https://5g-ppp.eu/wp-content/uploads/2015/02/5G-Vision-Brochure-v1.pdf`, 2015.

[4] ARANITI, G., CAMPOLO, C., CONDOLUCI, M., IERA, A., AND MOLINARO, A. Lte for vehicular networking: a survey. *IEEE Communications Magazine 51*, 5 (2013), 148–157.

[5] BANERJEE, A., CHO, J., EIDE, E., DUERIG, J., NGUYEN, B., RICCI, R., VAN DER MERWE, J., WEBB, K., AND WONG, G. Phantomnet: Research infrastructure for mobile networking, cloud computing and software-defined networking. *GetMobile: Mobile Computing and Communications 19*, 2 (2015), 28–33.

[6] BILAL, S. M., BERNARDOS, C. J., AND GUERRERO, C. Position-based routing in vehicular networks: A survey. *Journal of Network and Computer Applications 36*, 2 (2013), 685–697.

[7] BUFFERS, P. GoogleâĂŹs data interchange format, 2011.

[8] CHECKO, A., CHRISTIANSEN, H. L., YAN, Y., SCOLARI, L., KARDARAS, G., BERGER, M. S., AND DITTMANN, L. Cloud ran for mobile networksâĂŤa technology overview. *IEEE Communications surveys & tutorials 17*, 1 (2015), 405–426.

[9] CHEN, Y., DUAN, L., AND ZHANG, Q. Financial analysis of 4g network deployment. In *2015 IEEE Conference on Computer Communications (INFOCOM)* (2015), IEEE, pp. 1607–1615.

[10] CHO, J., NGUYEN, B., BANERJEE, A., RICCI, R., VAN DER MERWE, J., AND WEBB, K. Smore: software-defined networking mobile offloading architecture. In *Proceedings of the 4th workshop on All things cellular: operations, applications, & challenges* (2014), ACM, pp. 21–26.

[11] CONSORTIUM, C. V. S. C., ET AL. Vehicle safety communications project: Task 3 final report: identify intelligent vehicle safety applications enabled by dsrc. *National Highway Traffic Safety Administration, US Department of Transportation, Washington DC* (2005).

[12] CORICI, M., GOUVEIA, F., MAGEDANZ, T., AND VINGARZAN, D. Openepc: A technical infrastructure for early prototyping of ngmn testbeds. In *International Conference on Testbeds and Research Infrastructures* (2010), Springer, pp. 166–175.

[13] DAHLMAN, E., PARKVALL, S., AND SKOLD, J. *4G: LTE/LTE-Advanced for Mobile Broadband*, 1st ed. Academic Press, 2011.

[14] GRIGORIK, I. *High Performance Browser Networking: What every web developer should know about networking and web performance*. " O'Reilly Media, Inc.", 2013.

[15] HANSSON, H. A., NOLTE, T., NORSTROM, C., AND PUNNEKKAT, S. Integrating reliability and timing analysis of can-based systems. *IEEE Transactions on Industrial Electronics 49*, 6 (2002), 1240–1250.

[16] HARTENSTEIN, H., AND LABERTEAUX, L. A tutorial survey on vehicular ad hoc networks. *IEEE Communications magazine 46*, 6 (2008), 164–171.

[17] JIANG, D., AND DELGROSSI, L. Ieee 802.11 p: Towards an international standard for wireless access in vehicular environments. In *Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE* (2008), IEEE, pp. 2036–2040.

[18] KIM, H.-Y., KANG, D.-M., LEE, J.-H., AND CHUNG, T.-M. A performance evaluation of cellular network suitability for vanet. *World Academy of Science, Engineering and Technology, International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering 6*, 4 (2012), 448–451.

[19] LANER, M., SVOBODA, P., ROMIRER-MAIERHOFER, P., NIKAEIN, N., RICCIATO, F., AND RUPP, M. A comparison between one-way delays in operating hspa and lte networks. In *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2012 10th International Symposium on* (2012), IEEE, pp. 286–292.

[20] LECOMPTE, D., AND GABIN, F. Evolved multimedia broadcast/multicast service (embms) in lte-advanced: overview and rel-11 enhancements. *IEEE Communications Magazine 50*, 11 (2012), 68–74.

[21] MIR, Z. H., AND FILALI, F. Lte and ieee 802.11 p for vehicular networking: a performance evaluation. *EURASIP Journal on Wireless Communications and Networking 2014*, 1 (2014), 1.

[22] MISRA, P., AND ENGE, P. *Global Positioning System: Signals, Measurements and Performance Second Edition*. Lincoln, MA: Ganga-Jamuna Press, 2006.

[23] NASIM, R., KASSLER, A. J., ANTONIC, A., ET AL. Mobile publish/subscribe system for intelligent transport systems over a cloud environment. In *Cloud and Autonomic Computing (ICCAC), 2014 International Conference on* (2014), IEEE, pp. 187–195.

[24] PAPADIMITRATOS, P., DE LA FORTELLE, A., EVENSSEN, K., BRIGNOLO, R., AND COSENZA, S. Vehicular communication systems: Enabling technologies, applications, and future outlook on intelligent transportation. *IEEE Communications Magazine 47*, 11 (2009), 84–95.

[25] PONGTHAWORNKAMOL, T., NAHRSTEDT, K., AND WANG, G. The analysis of publish/subscribe systems over mobile wireless ad hoc networks. In *Mobile and Ubiquitous Systems: Networking & Services, 2007. MobiQuitous 2007. Fourth Annual International Conference on* (2007), IEEE, pp. 1–8.

[26] RAYCHAUDHURI, D., NAGARAJA, K., AND VENKATARAMANI, A. Mobilityfirst: a robust and trustworthy mobility-centric architecture for the future internet. *ACM SIGMOBILE Mobile Computing and Communications Review 16*, 3 (2012), 2–13.

[27] UCAR, S., ERGEN, S. C., AND OZKASAP, O. Multihop-cluster-based ieee 802.11 p and lte hybrid architecture for vanet safety message dissemination. *IEEE Transactions on Vehicular Technology 65*, 4 (2016), 2621–2636.

[28] VENKATARAMANI, A., KUROSE, J. F., RAYCHAUDHURI, D., NAGARAJA, K., MAO, M., AND BANERJEE, S. Mobilityfirst: a mobility-centric and trustworthy internet architecture. *ACM SIGCOMM Computer Communication Review 44*, 3 (2014), 74–80.

[29] WOLNY, G. Modified dmac clustering algorithm for vanets. In *2008 Third International Conference on Systems and Networks Communications* (2008), IEEE, pp. 268–273.

[30] ZHANG, J., WANG, F.-Y., WANG, K., LIN, W.-H., XU, X., AND CHEN, C. Data-driven intelligent transportation systems: A survey. *IEEE Transactions on Intelligent Transportation Systems 12*, 4 (2011), 1624–1639.

[31] ZHANG, Z., BOUKERCHE, A., AND PAZZI, R. A novel multi-hop clustering scheme for vehicular ad-hoc networks. In *Proceedings of the 9th ACM international symposium on Mobility management and wireless access* (2011), ACM, pp. 19–26.