Mercury: A Geo-Aware Mobility-Centric Publish/Subscribe System

Teja Kommineni and Kirk Webb

School of Computing, University of Utah

November 25, 2016

Abstract

Write me.

Introduction

Intelligent Transportation Systems (ITS) [?] 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) [?] 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 [?], robust communication approaches involving hybrid LTE and 802.11p [?] with dynamic multi-hop clustering have been proposed [7] [9] [10]. Considering safety and other types of information exchange in an ITS, applications and events have been identified that should be supported [?], 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 [6]. While research has been done in the areas of VANET communication and mobility-aware publish/subscribe systems, holistic compositions of these technologies is lacking. This paper introduces Mercury; a 5G-based [?] integration of pubsub systems and VANETs for rapid and reliable 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 such relationships to calculate the relevant dynamic set of vehicles for message transmission.

This paper makes the following contributions:

- Integration of publish/subscribe systems with VANETs using 5G mechanisms.

  The design of Mercury highlights the mechanisms by which publish/subscribe systems and VANETs can be combined in a 5G ecosystem. Message interception toward the pubsub broker, and injection toward the mobile vehicles is accomplished using SDN techniques over control and data paths in the EPS. Injected messages make use of multicast slots in the LTE radio access network (RAN) for efficient dissemination. We also show how concepts such as CloudRAN [3] and mobile edge computing [?] can further enable our system to operate within tighter latency bounds.

- Introduction of practical methods for determining areas of interest.

A road database is annotated with ingress and egress points, and boundaries for vehicle route placement. Vehicles report their positions as part of the telemetery collected by the message distribution broker. Together with context-specific details for event types (accident, obstruction in road, lane closure, emergency vehicle approaching, etc.), regions are dynamically computed.

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

  We implemented a prototype of Mercury and evaluated it on the PhantomNet [1] testbed. This prototype makes use of OpenEPC [5] in the core network and Open Air Interface (OAI) [**?**] at the edge (RAN). We drive the evaluation of Mercury using an adaptation of the SUMO [2] mobility model, and report on the former's scaling and latency characteristics.

The remainder of this paper is organized as follows: Section 2 covers the design of the Mercury message handling system. Section 3 discusses the implementation of Mercury, including integration of the MoPS pubsub system and *TAP*, our mechanism for interposing on EPS mobile control signalling and data flows for message delivery. Section 4 discusses our experimentation setup in Phantomnet and section 5 presents the results obtained. Section 6 covers related work, and section 7 concludes.

Design

Delivering messages, critical 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 **X** 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 [**?**]. 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).

- Message trust and integrity

  Messaging is vulnerable to various attacks, particularly if peers are used as transits such as in 802.11p multihop clusters. A combination of PKI, message integrity checks, and centralized vetting should be employed to curtail abuse and instill trust.

- 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 [?] [8] 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.

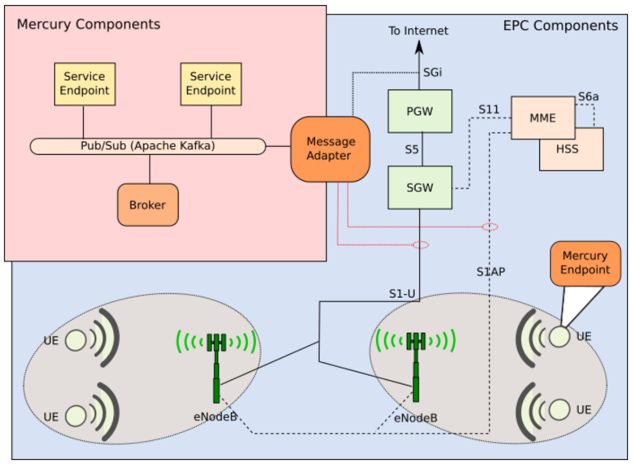

Figure 1: Mercury architecture diagram (in 4G mobile network context)

0.1 Mercury Components

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

- Publish/Subscribe System

  Mercury makes use of the Apache Kafkapub-sub system. Much work has been done on pubsub systems, and it is not the aim of this work to provide novel pubsub mechanisms. Apache Kafkaprovides ... **discuss desirable attributes, such as scaling, performance, community support, and rich subscription capabilities, and how they are relevant**.

Within Mercury we need pub-sub system for communication of different entities. This requirement directly arises from the need for each vehicle to send and listen to particular types of messages. Pub-Sub systems are elegant for these activities we can publish and subscribe to different events. In our system a vehicle(UE/Client), Message Broker publish and subscribe to events. This is when we have started for a quest of an effective pub-sub system. The table**insert the comparison table for pub-sub systems** shows the comparison of different pub-sub systems. Below we describe the differences between pub-systems in detail and different design decisions that went in for choosing a pub-sub system for mercury.

Mercury needs a pub-sub system that can serve high number of events with low latency. This is because of the environment in which mercury operates. We have many UE/clients at any given point that interact with the system and for them to effectively respond to a situation we need to dissipate the message at the earliest. Making the things even harder our clients are mobile which means they could intermittently loose connection with the pub-sub system which has to be re-established. Apache ActiveMQ[**?**] is one of the oldest player in this space. Though, ActiveMQ messaging is reliable it's performance forms a bottle neck to our system. RabbitMQ[**?**] is a widely used message broker with huge documentation. It is written in Erlang which is suitable for distributed applications, as concurrency and availability are well-supported. However, It relies on synchronous delivery of messages that decreases the numbe r of events being served. ZeroMQ[**?**] is another class of pub-sub systems that is designed for high throughput/low latency scenarios. Yet, most of the features will have to be implemented by ourselves combining various pieces of the framework. Redis is an in-memory data structure store, used as database, cache and message broker. Redis needs to have as much memory as there are messages in flight making it more attractive for short living messages and where we don't have a huge consumer capacity so as to not run out of memory.

Finally, we have crossed upon Apache Kafka. It is a pub-sub system written by linkedin and is under open source license of Apache. Apache Kafka addresses many of these problems mentioned above. It is very fast and the number of messages/events it serves are way more than any traditional pub-sub system. It can handle bursts without any out of memory issues. Also, It guarantees event ordering. This helps us in going back in time and reading a message that has been previously published to Kafka. We can seek old messages by remembering the offset. In a mobile environment this is very useful as we could see frequent disconnections. All these attractive features made us choose Kafka as the pub-sub system for mercury. Below, we write briefly about different concepts of Kafka.

- Apache Kafka

  Apache Kafka 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 consumedusing a configurable retention period.

  Kafka has a beautiful concept called Consumer Groups. Consumer Groups can be defined as set of consumers subscribed to single topic and each consumer is assigned a different partition. We can have multiple consumer groups listening to same topic or different. This concept of forming Consumer Groups helps kafka serve advantages of both message queue and a message-broker. Message Queues and message -broker systems have both strengths and a weaknesses. The strength of queuing is that it allows us to divide the processing of data over multiple consumer instances, which lets you scale your processing. But, queues can't be multi-subscribedonce one process reads the data it's gone. Publish-subscribe allows you broadcast data to multiple processes, but has no way of scaling processing since every message goes to every subscriber. The consumer group concept in Kafka generalizes these two concepts. As with a queue the consumer group allows you to divide up processing over a collection of processes (the members of the consumer group). As with publish-subscribe, Kafka allows you to broadcast messages to multiple consumer groups. The advantage of this model is that every topic has both these propertiesit can scale processing and is also multi-subscriberthere is no need to choose one or the other. This is something which we can take advantage of to scale Message Broker horizontally.

  Kafka has four main APIs: The Producer API allows an application to publish a stream records to one or more Kafka topics. Producers publish data to the topics of their choice. The Consumer API allows an application to subscribe to one or more topics and process the stream of records produced to them. The Streams API allows an application to act as a stream processor, consuming an input stream from one or more topics and producing an output stream to one or more output. The Connector API allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. We use Producer API and Consumer API extensively in our implementation.

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.

Discuss how Mercury integrates with the pubsub.

- 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 Apache Kafka). It also coordinates messaging system sessions with the endpoints. The broker calculates Areas of Interest (see section **??**) for certain message types (e.g., emergency notifications). Messages are vetted by the broker to prevent spoofing, enforce authorized use, cull abuses (flooding, other DoS), and perform system data analysis (endpoint reputation, problems with coverage, etc.).

  The broker forwards vetted and scoped messages to the pubsub system for transmission to appropriate endpoints. The broker does not concern itself with how to get the messages to the target endpoint(s); that job falls to the Mercury message adapter.

  The message broker is horizontally scalable in a data center environment. The implementation section discusses how a centralized database and distributed in-memory caching is used to accomplish this.

- Message Adapter

  The Mercury message adapter is the conduit through which pubsub messages flow through to 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.

  The message adapter receives group membership updates from the broker. This allows it to map (groups of) endpoints to destinations within the mobile network. For example, an adapter's domain may span from a single eNodeB to multiple eNodeBs. The message adapter forwards messages toward endpoints (UEs) based where they are attached and what state they are in.

- Endpoints

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

  Endpoints may also be centralized services that connect to the pubsub, running at the same location (e.g., CloudRAN datacenter). Examples include emergency notification processors, and consumer information aggregators (gas prices).

0.2 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 from mobile endpoints toward the broker to update location and status. The broker likewise sends periodic heartbeat messages toward all endpoints. These messages ensure clients and broker stay in sync. Session handling also includes establishment and teardown message exchange. 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. All messages flow through both the pubsub system and the broker. The broker is always in the loop so that it can enforce sender and message authenticity, perform flow control (rate limiting, etc.), and compute metrics.

Messages have temporal and spatial relevancy. Each message type, including content messages, are valid within a bounded time window. Further, certain message types have strict delivery deadlines (emergency messages, inter-vehicle coordination). Such time bounds necessarily restrict how Mercury can be deployed, as we discuss in subsection **??** below.

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." Mercury's design vision includes using spatial analysis techniques [?] [?] to create and test bounds. As we discuss in the implementation section, the prototype restricts itself to simple bounding models (e.g., radius or current route). Complex spatial reasoning for improving relevancy is left for future work.

0.3 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 [?], but we focus the discussion in this paper on the 3GPP EPS. The 4G system has been in active deployment for **XXX** years, and has undergone a number of revisions. We leverage features up through revision 12 of the 4G EPC standards. Also in play in some deployment scenarios (see section **??**) are software defined infrastructure concepts that are expected to be prominent components in the upcoming 5G EPS **??**.

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 **??** shows Mercury components in the context of a 4G EPS. There are additional 4G functions in this figure that we will cover in more depth when discussing Mercury's design details.

- User Equipment (UE)

  User Equipment typically refers to end user devices such as mobile phones, tablets, and 4G radio equipped laptops. The class of UE devices also includes fixed-position equipment, such as alarm controllers, remote telemetry systems, and other upcoming 5G Internet of Things (IoT) devices. Particularly relevant to Mercury, UE devices also include ITS endpoints (vehicles and road network infrastructure).

- Mobility Management Entity (MME)

  The MME is the 4G EPS control plane function responsible for tracking the live (dynamic) session state for UEs. It manages session setup/release, location tracking, handover between eNodeBs, and other tasks. It coordinates interaction between UEs, eNodeBs, SGWs, and the HSS. UE authentication and authorization are handled by the MME. In some deployment scenarios, Mercury observes the MME signalling (S1-AP protocol) to track device status and association. Mercury also uses the MME to setup and manage eMBMS bearers for multicasting messages to endpoints.

- Home Subscriber Service (HSS)

  The HSS is essentially a database of user (subscriber) information. The MME obtains UE information from the HSS based on mobile network identifiers (typically the IMSI in 4G EPS). Authentication parameters, QoS service levels, phone system parameters, and other information are stored for UEs in the HSS. Mercury does not make direct use of the HSS, but we mention it because it is a key component of the EPC.

- Serving Gateway (SGW)

  The SGW is one of the two anchor points for connected UEs (the other being the PGW). The SGW maintains GTP tunnels (which carry UE network traffic) between itself and the downstream eNodeB, and also an upstream PGW. Handovers between eNodeBs within a particular area typically involve migrating the same SGW's downstream tunnel between the two. Handover can be seamlessly handled such that it is largely invisible to the UE; its IP address stays the same, and its connections remain open. The upstream GTP tunnel to the PGW stays in place across handovers. In some deployments, Mercury taps into the UE data tunnel between eNodeBs and the SGW to extract messages at a location closer to the edge (e.g., in a CloudRAN data center).

- Packet Data Network Gateway (PDN-GW, or PGW)

  PGWs are frequently deployed in centralized datacenters in mobile carrier networks. They act as the egress point for a large number of UE data bearers (GTP tunnels), and fan out to multiple SGWs. PGWs forward UE trafic along toward a target network, which is frequently the Internet, but may be a corporate network or IMS (for VoLTE calls). PGWs enforce QoS, usually based on the subscriber's plan with the carrier. They also do traffic accounting, reporting to billing and charging functions. As with the HSS, Mercury does not make direct use of the PGW. In some deployments, Mercury sends and receives endpoint messages immediately upstream of the PGW device.

- Evolved NodeB (eNodeB)

  eNodeBs are the wireless access points of the 4G EPS. They bridge the radio access network (RAN) through which the UEs directly communicate with the evolved packet core (EPC). GTP tunnels are established for each UE device between the eNodeB it is associated with and an upstream SGW. The eNodeB also initiates session setup and default data bearer establishment when UE devices attach. It acts as a proxy for UE to MME control plane signalling (Non-Access Stratum (NAS) over S1-AP). eNodeBs covering adjacent cells coordinate through the MME and possibly with one another to accomplish handover as UEs move. In the distributed peer-to-peer deployment scenario, Mercury components run directly on the eNodeBs.

0.4 Deployment Scenarios

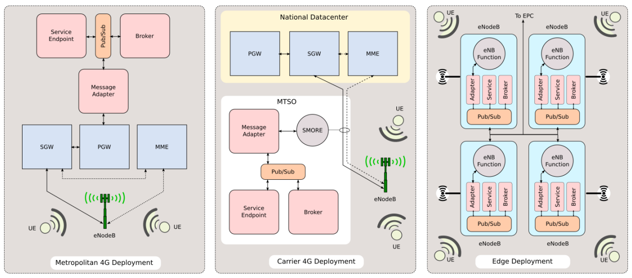

Figure 2: Mercury deployment scenarios.

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. The message adapter's primary function 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 SLAs. 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 **??**.

ADD DEPLOYMENT VS. LATENCY TABLE

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 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 present next three plausible deployment scenarios.

- 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 [**?**] 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.

- 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 [**?**]. Installing Mercury at these national data centers would impair its ability to meet SLAs for some classes of low-latency messages (see table **??**). 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 [**?**]. 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 [4] 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.

- 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 [3]. 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.

Implementation

Each Mercury component runs as a self-contained application. The Broker and Adapter(s) communicate over unmodified Apache 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 Apache Kafkato communicate with the Broker and talk to Clients via Adapter(s). Figure **??** shows our prototype implementation, including labeled communication channels. Note that we chose the centralized (via-PGW) deployment scenario as our target for the prototype implementation. All Mercury components are written in Python, aside from the unmodified Apache Kafkapubsub software.

FIXME - Add implementation diagram.

- The messages/events - types, constraints (size), attributes, etc.

  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 **??** 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 message 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.

- 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.

- Mercury broker

  – Our Approach

    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.

- Pubsub service

  FIXME - Add a brief discussion on the pubsub setup/config.

- Mobile network messaging adapter

  - Overview

    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 Apache Kafkasystem. 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 [**?**] 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).
  - Trust enforcement
    FIXME - Move to "discussion" section.

- UE/client mechanisms
  FIXME - FILL IN!

Evaluation

- Discuss evaluation approach.
  - Do evaluation in PhantomNet using OpenEPC with emulated RAN.
  - Use vehicle mobility model, SUMO, to drive realistic mobility scenarios.
  - Use SUMO output (position, primarily), to trigger handover.
- Types of evaluations to perform.
  - Functional: Endpoints connect and are tracked properly (handover).
  - Functional: Areas of Interest are interpreted properly.
  - Scaling: Run simulated/emulated scenarios with dozens of endpoints.
  - Scaling: Increase message/sec load and observe system response times.
  - Trust: Try to forge and alter messages (MITM)
  - Trust: Try to manipulate system state ("i.e., game the system").

Discussion

- Lessons/insights extracted from design/implementation/evaluation.
- Future work.
- Limitations of approach.

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 [7], MDMAC [9], and NHop [10] systems attempt to form stable mobile 802.11p clusters, using the LTE network to bridge between disconnected clusters. **INSERT INFO ON CMGM.** These systems are complimentary to Mercury in that they can be used to reduce LTE resource contention and improve reliable transfer of messages.

NEED MORE PUBSUB CITATIONS.

The MoPS [6] publish-subscribe system scales efficiently for large numbers of clients and deals well with changing broker association. Mercury could replace the Emulab pubsub system used in the prototype with MoPS to help it scale better. MoPS does not include the area of interest concept, which would continue to be handled by the Mercury broker.

Conclusion

A conclusion on the marvelous Mercury messaging mechanism goes here.

References

[1] 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.

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

[3] Checko, A., Christiansen, H. L., Yan, Y., Scolari, L., Kardaras, G., Berger, M. S., and Dittmann, L. Cloud ran for mobile networksa technology overview. *IEEE Communications surveys & tutorials 17*, 1 (2015), 405–426.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.