Universidade de Aveiro
2016

Departamento de Eletrónica, Telecomunicações e Informática

Francisco José Pires Vaz

VNMS: Sistema de mensagens para redes veiculares

VNMS: Vehicular Network Messaging System

Universidade de Aveiro
2016

Departamento de Eletrónica, Telecomunicações e Informática

Francisco José Pires Vaz

VNMS: Sistema de Mensagens para redes veiculares

VNMS: Vehicular Network Messaging System

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Professor Doutor José Maria Fernandes, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e coorientação da Professora Doutora Susana Sargento, Professora Associada com Agregação do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri

presidente	**Prof. Doutor Joaquim João Estrela Ribeiro Silvestre Madeira** Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro
vogais / examiners committee	**Prof. Doutor Rui Pedro de Magalhães Claro Prior** Professor Auxiliar do Departamento de Ciências de Computadores da Faculdade de Ciências da Universidade do Porto
	Prof. Doutor José Maria Amaral Fernandes Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

agradecimentos

Quero agradecer aos meus pais por todo o apoio que me deram ao longo destes anos. Por acreditarem em mim, pela paciência que tiveram e por terem estado ao meu lado sempre que precisei. Sem eles eu não teria conseguido chegar a onde estou hoje e tudo o que tenho atualmente devo a eles.

Quero agradecer também ao professor José Maria Fernandes e à professora Susana Sargento por todo o apoio, conhecimento, sabedoria e experiência quem me forneceram durante o meu percurso académico e no desenvolvimento deste trabalho.

palavras-chave

Redes veiculares, VANET, comunicação utilizador-a-utilizador, comunicação utilizador-a-infraestrutura, comunicação veículo-a-veículo, Android, RabbitMQ, Beacon, Web Server.

resumo

Com conceitos como a internet das coisas a surgir e a tornarem-se cada vez mais populares, criar ligações entre veículos foi apenas um próximo passo lógico, formando assim as redes ad hoc veiculares. Estas redes são um caso particular das redes móveis ad hoc, nas quais os veículos se ligam uns aos outros de uma forma espontânea. Acrescentar aos veículos a capacidade de comunicarem uns com os outros faz surgir uma abundância de possibilidades. Contudo, atualmente já existem diversas aplicações que fazem uso destas redes; no entanto, a maioria destas aplicações estão mais diretamente relacionadas com a ccomunicação entre veículo e não entre utilizadores. Soluções como o REINVENT fornecem a capacidade de expedir mensagens através de uma VANET utilizando *smartphones*, contudo falta-lhe uma camada lógica capaz de suportar a expedição de mensagens de utilizador para utilizador.

A nossa contribuição, o Sistema de Mensagens para Redes Veiculares (VNMS), permite a troca de mensagens entre utilizadores numa VANET. Com a implantação de um quadro de avisos virtual nos nós da VANET, com uma camada de reencaminhamento de mensagens e um *naming service*, fornece aos utilizadores a capacidade de trocarem mensagens entre si sem a necessidade de informação ou serviços da VANET. Os nós do VNMS atuam como agregadores de mensagens, providenciando repositórios locais de mensagens de utilizadores e reencaminhamento sobre a rede para o utilizador alvo, i.e., o nó ao qual o utilizador está ligado. Na perspetiva do utilizador, este pode usar os serviços do VNMS de uma forma transparente através de uma aplicação Android – foi criada uma aplicação de *chat* que usa a VANET como prova de conceito.

keywords

Vehicular Networks, user-to-user communication, user-to-infrastructure communication, vehicle-to-vehicle communication, Android, RabbitMQ, Beacon, Webserver

abstract

With concepts like *the internet of things* currently cropping up and getting more popular, connecting vehicles with each other was just a logical step, originating the vehicular ad-hoc networks (VANETs). VANETs are a particular case of Mobile ad-hoc networks (MANETs) in which vehicles connect with other vehicles in ad-hoc mode and evolving topologies. By enhancing vehicles with the ability to communicate with each other, an abundance of capabilities arises. However, currently most applications using VANETs are focused on the vehicle to vehicle communications, and not on vehicles users, either drivers or passengers. Previous work like REINVENT provided a solution capable of dispatching messages through VANETs using standard smartphones; however, it lacked a logical layer to support user to user logical message brokering.

Our contribution, the Vehicular Network Messaging System (VNMS), allows user to user message exchange on VANET. By deploying virtual bulletin boards (VBBs) in VANETs nodes, a layer of message forwarding, and user naming service, it provides users the ability to exchange messages without the explicit need of any VANETs specific information or service. VNMS nodes act as brokers for user messages, providing local user message repositories and VANETs routing to targeted user(s) i.e. its VANET node. From the user perspective, it is possible to use VNMS services transparently using Android mobile application – we implemented a VANETs enabled chat application as proof of concept.

I. Contents

II. List of figures

1 Introduction

Vehicles are a leading solution used to transit either people or goods from one place to another. With so many vehicles circulating in the roads and the expanding popularity of concepts like “The Internet of Things” [1] [2], it is only natural that these vehicles will be integrated on a network. Amidst all these concepts and ideas, the vehicular networks (VANETs) were created.

VANETs connect nearby vehicles and provide them with the means to communicate among them, without the need of accessing to the internet, just like in any standard LAN without using an internet service provider (ISP). These communications between vehicles are titled vehicle-to-vehicle communications (V2V).

VANETs deployment implies that vehicles incorporate an appliance called On-Board Unit (OBU) capable of wireless communication. The connectivity between the OBUs in vehicles relies on Dedicated Short Range Communications (DSRC) technology instead of the typical Wi-Fi because it is the only available technology in the near-term that offers the latency, accuracy and reliability needed for active safety [3].

Although VANETs do not depend on external ISPs, they often use immobile nodes that can act as gateways (internet access), data repositories, and permanent connection to other geographically isolated network clusters – the Road-Side Units (RSU) provide vehicle-to-infrastructure (V2I) communications support using the same networking solution as OBUs. In Figure 1 we can see the types of connections that a VANET can have.

By augmenting vehicles with the ability of being able to communicate with one another, an abundance of capabilities comes to surface that can be used to service the users of these vehicles. The types of practical uses that can benefit from this unique type of communication can vary from commercial, traffic orientation, safety, productive and infotainment applications.

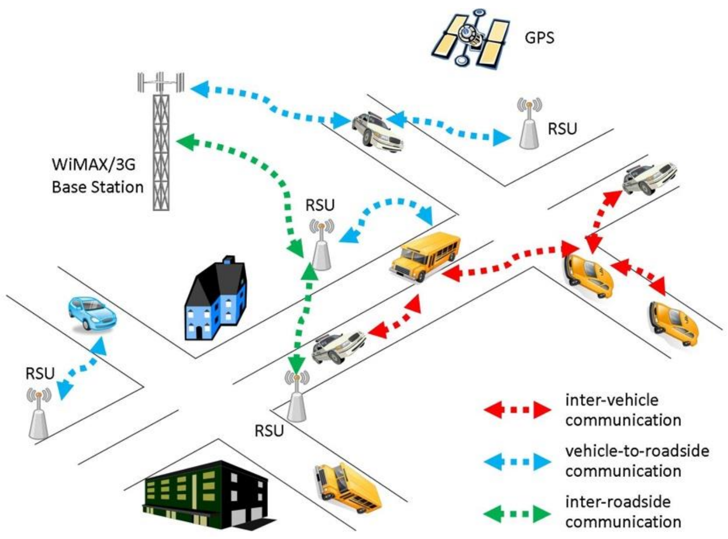

GPS
WiMAX/3G Base Station
RSU
RSU
RSU
RSU
inter-vehicle communication
vehicle-to-roadside communication
inter-roadside communication

Figure 1 – VANET’s types of communications [4]

However, OBUs and RSUs act as access point to other devices and do not provide any user interface. End users then connect to the vehicle OBU via smartphones, tablets, and laptops. These devices, typically, do not support DSRC technology, meaning that they cannot connect directly to the VANET. In this context several works have been made to allow end users to make use of VANETs resources transparently namely REINVENT [5] and VANESS [6]. Figure 2 shows a photo of one of the OBUs used in the development of this system.

By enhancing the vehicles with the means to communicate among them, and the users with the means to connect to these vehicles, the possibility of user-to-user (U2U) communication is created, and with it a number of possible applications arise.

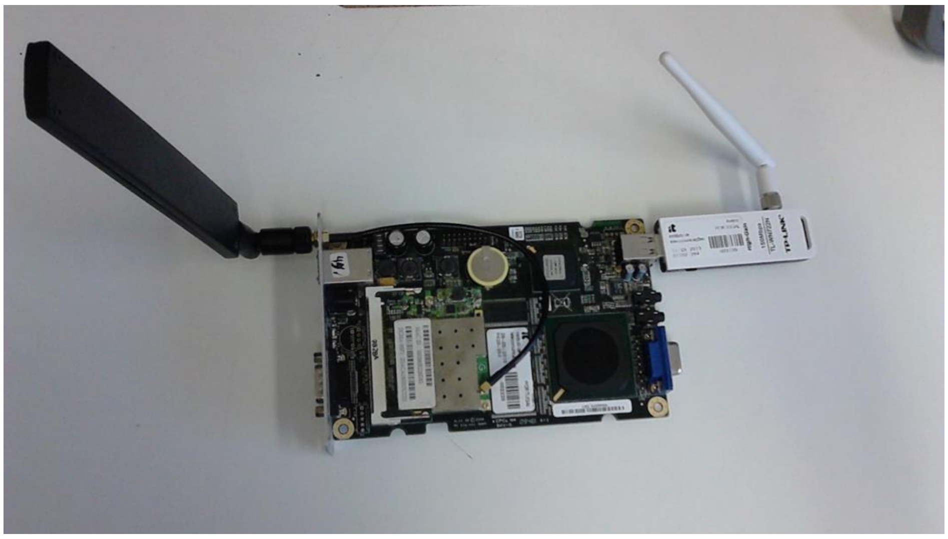


Figure 2 – On Board Unit

1.1 The Problem

Despite the technical possibilities that allow communications in VANETs, there is still an absence of support for user level applications for the vehicle users themselves, being either drivers or passengers. Some efforts exist, namely REINVENT which provides a system capable of dispatching messages to a VANET in multiple platforms using a messaging solution. However, it lacks a logical layer to support message brokering between users.

There is still a divide between the VANET and the intra-vehicular network that an OBU can establish, and in the logical scope of U2U communication throughout a VANET regardless of the shifting topologies, network conditions, etc. This divide becomes more perceptible when we envision public transportation. In public transportation, when users (passengers) want to perform a travel they will, in all likelihood, use more than one vehicle before reaching their destination, meaning that in order to do user-wise communications, the VANET would have to know in which vehicle a given user is connecting to the network. Furthermore, this U2U communication problem becomes more complex as vehicles, such as buses, can carry multiple passengers, meaning multiple users in the same vehicle. Given all these considerations, it becomes evident that plain V2V communications are not capable enough to solve the U2U communication problem.

1.2 Solution

The solution implemented, Vehicular Network Messaging System (VNMS), is an extension of REINVENT, by providing a user messaging broker level in the VANETs nodes (OBUs and RSUs). While maintaining the user level abstraction at the application level, it extends the VANET nodes with a message routing and storage ability making them act as virtual bulletin boards (VBBs), where users can post messages in and read from. VNMS supports a distributed user naming service to allow U2U message routing – a distributed evolution of the prior centralized solution VANESS.

1.3 Motivation

VNMS goal and motivation is to provide vehicle users with an easy way to communicate amongst them and doing so without requiring external services (e.g. internet access). Additionally, VNMS also creates virtual bulletin boards (VBBs) where users can post and read messages. VNMS provides users with a smartphone application, similar to any other chat application, where they can add friends, send text messages, and access the VBBs, post messages on the VBBs for his friends and get messages that were posted there, all of this using only the VANET.

The idea behind the chat application is pretty simple: a user can send and receive messages allowing them to have conversations, just like any regular chat application. The VBB is, however, a more complex concept. In this VBBs, users can store messages with expiration dates; furthermore, these VBBs are geographically scattered throughout the area where the VANET operates, and can only be accessed if the users are in its vicinities, thus creating time and space relative messages. The idea behind this is that users can leave a "*post*" in one of these VBBs, in a given area and with an expiration time, so that if any of their friends happens to be in that area during the time that the message is valid, they will receive that message.

1.4 Dissertation outline

This dissertation is divided in 6 chapters.

The current chapter provides an introduction to the environment in which the system operates, the obstacles derived from this environment and a proposed solution to work around them.

Chapter 2 aims to disclose an overview of the current state of vehicular networks. It describers the characteristics, pros and cons, and methodologies that are currently being deployed and studied to make use of their advantages as well as to overcome the shortcomings introduced by them.

Chapter 3 describes the system architecture of the VNMS. It details the various system components, the various features from which they are comprised and how these components communicate with each other.

Chapter 4 describes the implementation of the system components. This chapter expresses a more in depth view of the components and how they work internally.

Chapter 5 presents results of the system being used and achieving its desired goals.

Chapter 6 recaps the work done and what was learned during the development of this system. It also describes new features that the system could make use of.

2 State of the art

Ad-hoc networks are networks formed extemporaneously as nodes/devices connect to it. These networks have two types of topologies, they can be either heterogeneous, when some part of the network is structured and has gateways, or they can be homogeneous, when all nodes that build the network have the same capabilities and responsibilities.

Mobile ad-hoc networks (MANETs) are homogeneous ad-hoc networks created by mobile devices that connect directly with each other.

2.1 Vehicular Ad-hoc Networks (VANETs)

VANETs are a form of MANETs, they use the same principles as a MANET but have some particular characteristics. They are comprised of mobile nodes (the vehicles themselves) but can additionally encompass fixed nodes located on the roadside that provide connectivity and support to passing vehicles. Their main goal is to provide vehicle-to-vehicle and vehicle-to-infrastructure communications. These are peer-to-peer multi hop networks, where all nodes have the same authority.

The architecture of VANETs falls within three categories, they can be either pure cellular/WLAN, pure ad hoc, or hybrid [7].

In pure cellular/WLAN networks, the vehicles connect to the network by means of cellular gateways or by immobile WLAN access points located on the roadside. Vehicles do not communicate between them directly and do it so by usage of the structured network infrastructure. The limitation of this type of architecture is the need of a widely deployed infrastructure of cellular towers and/or wireless access points. Cellular coverage is widely accessible nowadays, but there are still some blind spots that can be induced by obstacle shadowing, furthermore a considerable cellular network usage can be costly. In addition, covering a wide geographical area with WLAN access points can also be a costly deed.

Pure ad-hoc networks are structureless meaning that they do not require additional infrastructures such as cellular towers or WLAN access points. In these networks, vehicles can only communicate with other vehicles so, in order to achieve full connectivity, they will not only be the clients of the network but will also be gateways aiming to provide data routes to other vehicles.

Hybrid networks results of combining pure cellular/WLAN networks with ad hoc networks. Vehicles are provided with the ability to connect to the cellular network, to the WLAN access points and to other vehicles [8]. This hybrid approach can provide an improved coverage but will also cause new problems implementing mechanisms to provide a seamless transition of the different types of communications amongst the different wireless systems.

To create a VANET there are several aspects that need to be accommodated, this is because of the underlying unique characteristics these networks have [9] [10]:

- **Highly dynamic topology –** Due to the speed of vehicles and as they move in or out of range of each other, transmission paths are created and severed constantly. This results in a network with ceaseless topology changes.
- **Scalability -** As a result of the highly dynamic topology, VANETs need to be able to operate seamlessly with both a small number of nodes or with a large number of nodes.
- **High quality network technology –** As a result of the dynamic topology, the network technology used to transmit data between nodes must be very dependable. It needs to be able to quickly establish connections, have a low latency to transmit data with very low delay, and be highly reliable [11].
- **Security and privacy –** As VANETs are mostly public networks, it needs to provide safe message authentication and privacy.

Vehicles however do not comprise, by default, the means to connect to a VANET by themselves. For a vehicle to be able to connect to a VANET, it needs to be equipped with a specific hardware device called on board unit (OBU). Usually these OBUs are micro computers with network devices and can additionally include other peripherals such as GPS modules, thermometers, etc. They are responsible for retrieving information from the vehicle itself and be able to transmit that information. For an OBU to be able to achieve communication and to accommodate all the requirements a VANET as pertaining its network communications, standard wireless interfaces cannot be used. The devices used for this type of communications operate on the dedicated short-range communications (DSRC) frequency using WAVE technology. The wireless access in vehicular environments (WAVE) is significantly different from the Wi-Fi and cellular wireless network environments. The more prominent standards for DSRC/WAVE networks are the IEEE802.11P and IEEE1609 [3].

By augmenting vehicles with the ability to communicate with one another, an abundance of capabilities that can be used to service the users of these vehicles comes to surface as can be seen in Figure 3. The types of practical uses that can benefit from this unique type of communication, can vary in different types of applications:

- **Active road safety –** Automated applications can transmit data to nearby vehicles (e.g. speed, location, heading), process data received by other vehicles and thus maintaining a state of awareness of its surroundings. Additionally, by communicating to the road side units, they can obtain information about the environment (e.g. road hazards, accidents, weather conditions).

- **Traffic orientation and fleet management –** By reporting the vehicles location to some centralized system, these networks can be used to monitor the roads traffic state and therefore use this information to dispatch recommended route diversions in case of road congestions.
- **Infotainment –** Enabling vehicles with the capability of communicating between them, the ability of the vehicle users to communicate with each other also arises. This creates the possibility of creating localized entertainment and social applications.

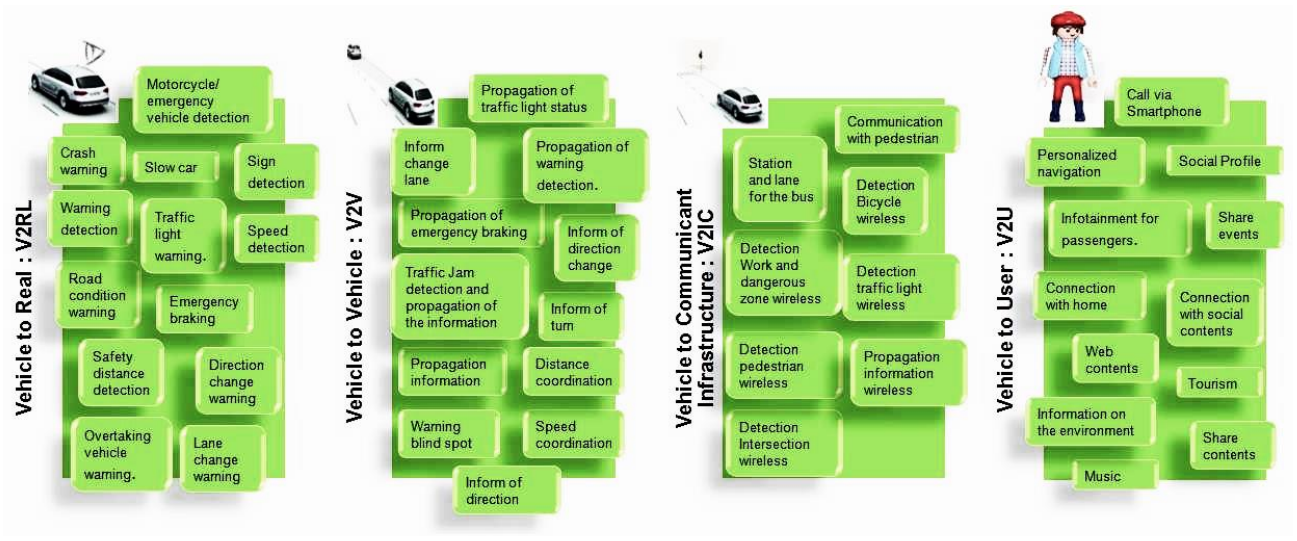

Vehicle to Real : V2RL
Motorcycle/ emergency vehicle detection
Crash warning
Slow car
Sign detection
Warning detection
Traffic light warning.
Speed detection
Road condition warning
Emergency braking
Safety distance detection
Direction change warning
Overtaking vehicle warning.
Lane change warning
Vehicle to Vehicle : V2V
Propagation of traffic light status
Inform change lane
Propagation of warning detection.
Propagation of emergency braking
Inform of direction change
Traffic Jam detection and propagation of the information
Inform of turn
Propagation information
Distance coordination
Warning blind spot
Speed coordination
Inform of direction
Vehicle to Communicant Infrastructure : V2IC
Communication with pedestrian
Station and lane for the bus
Detection Bicycle wireless
Detection Work and dangerous zone wireless
Detection traffic light wireless
Detection pedestrian wireless
Propagation information wireless
Detection Intersection wireless
Vehicle to User : V2U
Call via Smartphone
Personalized navigation
Social Profile
Infotainment for passengers.
Share events
Connection with home
Connection with social contents
Web contents
Tourism
Information on the environment
Share contents
Music

Figure 3 – VANET applications (adapted from [12])

2.2 Vehicular Ad-hoc Networks Routing

Due to the unique characteristics that define a VANET, creating an efficient and reliable protocol presents a hefty task. The highly dynamic topology where transmission paths are created and severed constantly require that these protocols be robust and resilient. VANET routing protocols are usually classified in five categories: topology based routing, position based routing, cluster based routing, geo cast routing and broadcast routing [8] [13].

Topology based routing is the simplest form of routing. These protocols make use of the links information to perform packet forwarding, they branch in two different approaches to achieve this. They can either be proactive or reactive. The *proactive* approach is a table driven approach and works similarly to routing protocols used in the internet. Nodes periodically broadcast packets with their knowledge of the surrounding topology to other nodes, and use this information to create routing tables that indicate the next hop node to which they will forward the information to reach the desired destination. In this approach, nodes also contain information of unused paths, meaning that part of information they share on the periodical broadcasts is meaningless. As nodes have to periodically broadcast information and as more nodes congregate, this can have a severe impact on the network bandwidth. In *reactive approach* no knowledge of the network topology is maintained, nodes only open a route when they need to forward data to another node and will only keep information of the routes that are currently being used. In this approach, when nodes need to learn a route to another node, they initiate a discovery process similar to the one from the proactive approach in order to attain a path, thus having an increased latency when a new route needs to be discovered. When a path is found, the discovery process ends and the nodes can now communicate.

Position based routing, also known as geographic based routing, makes use of a node's position to discover a path to that node. This method is mainly used in cases where the destination position is known like in a sensor networks with centralized data aggregators. In this approach, nodes need only to know the location of other nodes in a single hop distance, therefore only single hop broadcasts are necessary. As nodes send their location periodically, neighbouring nodes can maintain their routing tables updated, and as only the information of single hop distance nodes is maintained, this approach has a high scalability and introduces a minimal overhead on the network which results in a low bandwidth usage [14].

Cluster based routing operates by assembling a group of nearby nodes into overlapping clusters. Each cluster will elect one of its nodes as the cluster's head whose duty is to manage the intra and inter-cluster communications. Cluster's head nodes appoint gateway nodes that are located in the overlapping area and use them to communicate between them. By doing this the cluster head node can attain the membership information of other clusters. This means that inter-cluster routes are found by flooding the network through gateway nodes with route requests. Additionally, they attain the intra cluster membership information by periodically sending single hop distance "hello messages" and subsequently creating intra node routes. In this type of routing the protocols responsible for cluster formation, cluster head and cluster gateway selection have a great relevance in managing the VANETs, which will have a highly dynamic cluster formation that can be very computing intensive [13] [15].

Geo cast routing is based on location wise multicast routing. It consists of broadcasting a message to all nodes belonging to a well-defined geographical area called zone of relevance (ZOR). To achieve this there are several approaches that geo cast routing protocols use with different characteristics e.g. path strategy, scalability, message complexity and robustness. The least complex of these approaches is the simple flooding in which, when a node receives a message that it never received before and that originates from its ZOR, it will simply broadcast it to all surrounding nodes. More complex approaches are based on direct flooding by defining forwarding zones which comprise a sub set of all the network nodes. When these approaches fail to deliver the message they either increase their forwarding zone or fall back to simple flooding [16] [17].

Broadcast routing is achieved by sending a packet to all nodes in the network and the messages need to reach vehicles beyond transmission range. To achieve this multi hops packets are used. It is a very reliable method, but it requires a high use of bandwidth and nodes receive duplicate messages very often.

2.3 REINVENT

REINVENT [5] [18] is a system created with the aim of simplifying the communication between VANETs and mobile applications. It creates an abstraction layer that mobile applications make use of, allowing them to use the VANET resources without the need to delve further into the abstraction layers that define network communications. REINVENT's architecture is depicted in Figure 4.

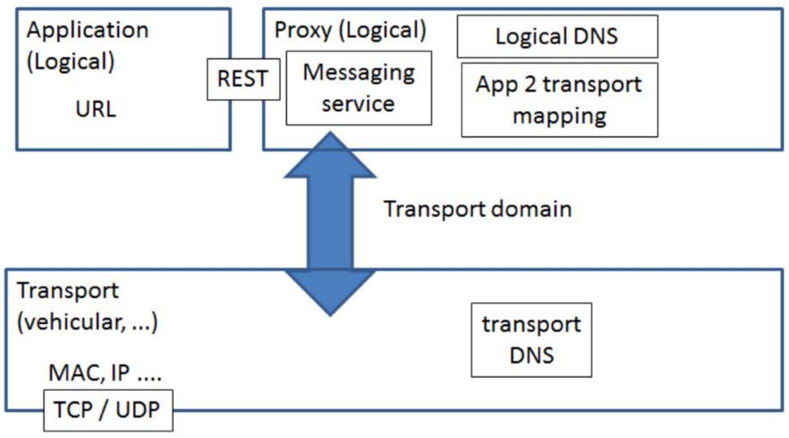


Figure 4 – REINVENT conceptual architecture [5]

To achieve this, REINVENT employs the Representational State Transfer (REST) architecture – an architectural style for designing networked applications [19]. REST operates on a client-server basis that uses basic HTTP to communicate and provides support for the CRUD (create, read, update, delete) operations. It separates the user interface from the processing segment making it possible for the user interface to be set up on different platforms using a different system architecture, and also improving the scalability of the servers as they are detached from the user interface. The communications on REST are stateless meaning that requests from a client to a server must contain all the information needed for the server to understand the request. REST is used on REINVENT to provide an abstraction between the user application and the modules (servers).

Communications between the clients and servers are made using a message-oriented middleware (MOM). MOM is a software framework that yields the capability of sending and receiving messages. Software MOM enables applications and devices to communicate with each other, and work as a larger application; despite the requirements that clients must connect to their servers, it can be used for peer-to-peer communication between two clients through these servers. The interactions models used by this middleware can be described by either synchronous or asynchronous interactions. Synchronous communications require that the client must block its execution thread when making a call, and for it to wait the call termination before resuming its execution. Asynchronous communications allow the caller to continue its execution upon making a call, and to achieve this, it requires a message broker to handle the requests.

Message brokers use message queues to forward the requests to their designated destination and these queues tend to work on a first-in first-out (FIFO) approach, meaning that messages will be processed in order of arrival. Message queues can operate using two different models: the point-to-point (PTP) model and the publish/subscribe. On the PTP model, queues are used to send messages to a single consumer and messages are removed from the queue upon consumption. Additionally, even though the PTP model is used to communicate to a single consumer, multiple clients can publish messages on this type of queues. The publish/subscribe method allows the communication amongst one client and many consumers, and the communication amongst many clients and many consumers. Consumers subscribe to the desired channels and topics, and clients publish the messages on these channels and topics. The message broker will be responsible for forwarding these messages to the desired recipients. REINVENT uses the Advanced Message Protocol [20] (AMQP) as the application layer for the MOM.

2.4 VANESS

VANESS [6] [21] is a naming service to support U2U communications in a VANET, with or without access to an ISP, focused on mobile devices. It is able to map a user to its access point either by using only the VANET, by making use of an internet connection available to that access point, or by using an internet connection available to some other VANET node. It is deployed over three main components: the smartphone application, the OBUs, and on the web. On the smartphone application, it is used for communication and authentication purposes. On the OBUs it will allow messages to be exchanged between devices and the VANET, and between vehicles. On the web it will deploy a centralized naming service responsible for data management, and to allow distinct VANETS to communicate if both have internet access.

VANESS is an extension of REINVENT, and expands it by appending two modules, namely VNS (VANET naming service), and VnAuth (VANET Authentication). VNS is responsible for mapping the users to the vehicles, more precisely to the unit through which a user connects to the VANET. VnAuth provides a device-centralized login and signup method that is used by the smartphone application, allowing VANESS to know the users connected to the system, the association of the user to the device, and subsequently the unit to which they are connected. Figure 5 depicts the VANESS architecture.

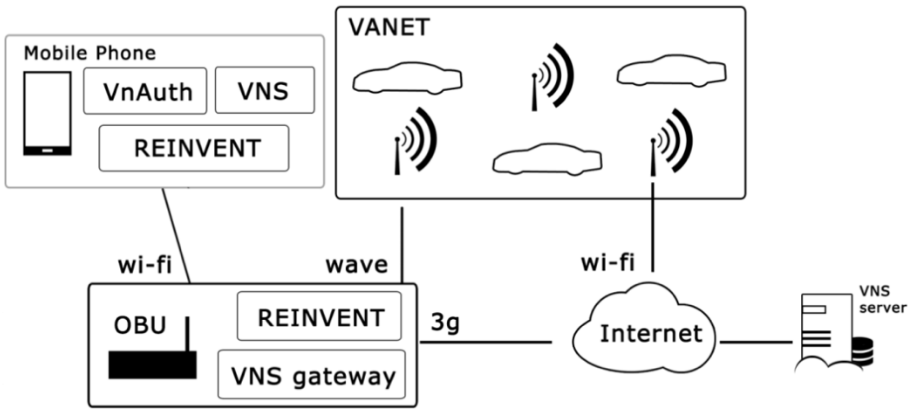


Figure 5 – VANESS architecture [6]

REINVENT is not capable of knowing to which vehicle a user is connected, and therefore is unable to forward him messages. VNS is internal to VANESS and transparently used by a developer creating applications. As VANESS extends REINVENT, for an application to send a message to a specific user, it will do it using REINVENT only, which internally will use the VNS module to forward it to the rightful destination. The way VANESS operates is that when a message is sent through REINVENT, a request is made to the VNS module with the destination username, and in turn, VNS will reply with the unit's ID to which the destination is connected. After learning the unit's ID, REINVENT pairs this ID to the message and sends it through the

VANET for it to reach the desired recipient, which will then be received by the user’s mobile application. After doing this, the sender’s mobile application will also gain knowledge of the addressee unit which will keep in cache.

VNS has several ways of mapping a user to a unit. Firstly, it will try to contact the web server; as this is a centralized resource it is likely to have the most reliable information. It can achieve this by directly contacting it through the unit’s internet access, and if not available, another unit can perform this request and provide the answer. VNS accesses this web server by means of a REST web service it deploys. If the web server is not available, the units themselves will translate the request, but only if the addressee is connected to them in which case they will provide the answer. Lastly, the smartphone application maintains in cache the locations of the recently contacted users; in case any of the previous cases fails, this information will be used. The mapping information is kept updated through the units by means of the VnAuth module which every time a mobile application connects to a unit, will revise the information. However, VANESS does not implement any mechanism to keep track if a user disconnects from the system either manually or by getting out of communication range, meaning that if this happens, the system will still keep information of the last known location of a given user, and will make use of it to forward messages to him.

3 Vehicular Network Messaging System

Vehicular Network Messaging System (VNMS) is the main contribution of the current work. The main goal of VNMS is to deploy a messaging system capable of user-to-user, and user-to-infrastructure communication. VNMS is deployed over VANET nodes (both OBUs and RSUs), allowing users, through the smartphone application, to use the VANET to communicate with other users over the VANET transparently i.e. without explicitly handling VANET specific services or information such as the vehicle identification or VANET topology.

VNMS provides users the ability of posting messages in the VANET nodes that are scattered throughout the network. Users can define the time during which the message is deemed valid, and the visibility of the message (i.e. who can read a given message). These messages can be posted (sent) to specific users, to a group of users, or can be posted in public groups that anyone can read if they are subscribers of this given public group. This functionality works as virtual bulletin boards for location wise and time sensitive messages.

VNMS mobile application uses REINVENT to support smartphone communication with VANET/VNMS nodes.

3.1 Architecture

VNMS is based on previous work of MSc, namely REINVENT [5] and VANESS [6], but its goal is to extend their functionalities.

REINVENT's main concept is to abstract the transport resources behind a Representational State Transfer (REST) interface with a module that encloses the support of messaging system already coupled with the transport layer specifications. This module incorporates the messaging service that encapsulates the application messages in the specific transport layer protocol [5]. REINVENT however has some limitations that needed to be addressed for VNMS to achieve its goals, namely that it only supports one user per unit, and that it does not implement a reliable data transmission system in the case that users disconnect from the unit, resulting in message losses. VNMS extends REINVENT by adding communication channels that support multiple users and an acknowledge mechanism to ensure that no data is lost.

VANESS can be described as a naming service to support U2U communication in a heterogeneous scenario comprising both VANET and traditional web. Its goal is to ease the use of VANETs' communication resources on mobile applications working as a naming service resource transparent to developers, making exchanges of messages between users immediate to any software running on a mobile device. It handles the translation/mapping between the user's alias/username and the transport level needed information [6]. VANESS uses REINVENT as communication system between applications and the VANET, meaning that it suffers from the same limitations of REINVENT. Furthermore, VANESS uses an external centralized naming service that resides on the internet; VNMS replaces it with an intra-network distributed naming service deployed on the VANET nodes, eliminating the need for internet access. It also expands it to support multiple users connected to the same node.

The VNMS system architecture features three key components:

- **Mobile application** as the end user segment, through which users are serviced. This application enables the user to send and receive messages to other users and to access the VBB for posting and reading messages.
- **Roadside units** (RSUs) are stationary units located throughout the geographical area where the VANET operates. These units can be considered as hotspots to which the vehicular part of the network connects. In addition, these units serve as repositories where the VBB messages are stored.
- **On board units** (OBUs) are located on vehicles and are responsible for exchanging messages between users and the vehicular network. These units can connect directly to the RSUs or through other OBUs.

As depicted in Figure 6, smartphones connect to the units via Wi-Fi interfaces and vehicular units communicate with other units through the WAVE interface. VNMS uses a client-server architecture, in which the smartphone applications are the clients and vehicular units deploy the servers.

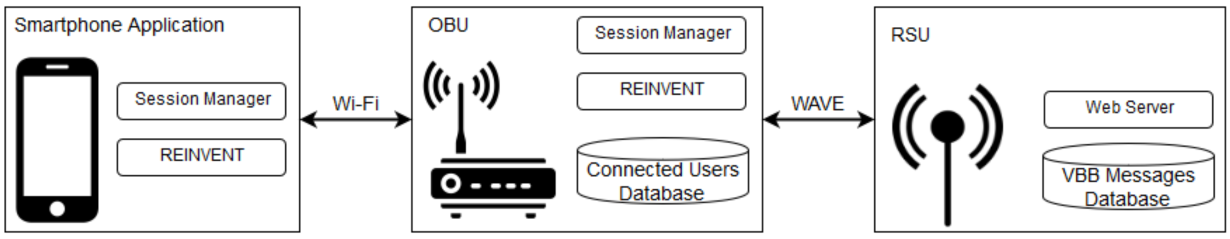


Figure 6 - VNMS components and connections

3.2 VANET nodes (Units)

Vehicular units, both OBUs and RSUs, are responsible for exchanging messages between themselves and the smartphone application. They provide the access point to which users connect to the network and system. Additionally, units also communicate with other units establishing the VANET.

They act as network switches in the sense that they forward messages to the endpoint devices connected to them, but also act as routers in the sense that they are responsible for finding a way to deliver messages to users connected to different units. Units are comprised of several modules responsible for tackling specific problems. Figure 7 depicts the modules that the units deploy.

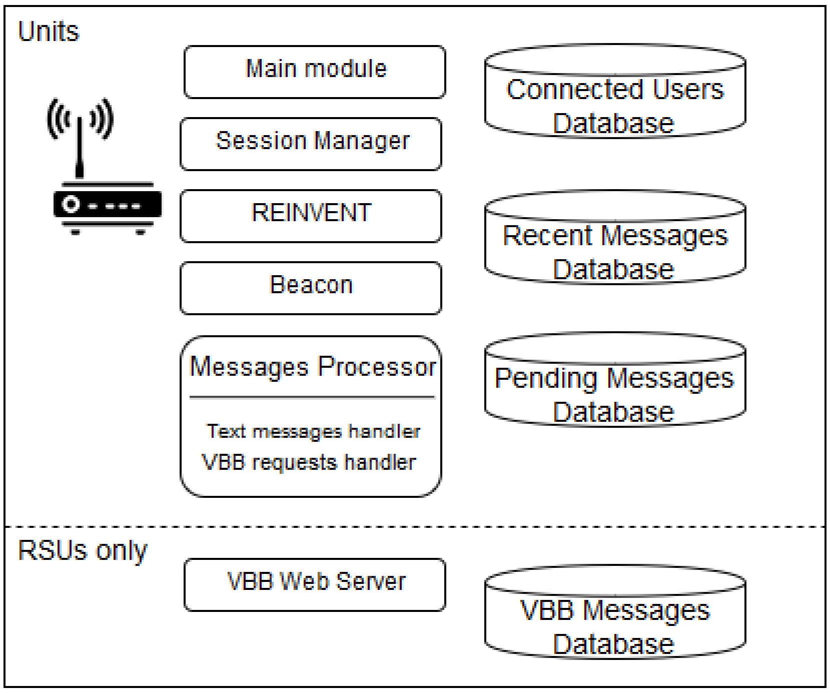


Figure 7 – Units components

The **main module** sets up all the environment for the other modules to operate and initialises them. In addition to this, it also deploys the listener responsible for attaining messages broadcasted in the VANET. After receiving a message from the VANET, it forwards them to the message processor.

The **message processor** is responsible for identifying the type of messages that arrive from the VANET and for sending them to the respective handler. The text messages handler goal is to forward text messages do their respective recipients. The VBB request handler processes the VBB requests made by users, and will take different actions depending on the type of unit where it resides. A more detailed explanation of how it works will be made on the following chapter 3.4.3.

The **session manager** keeps track of the users connected to the unit. It listens for requests made by the smartphone application, whether they are subscriptions or drafts. This module is one of the key extensions made to REINVENT as it allows the system to support multiple users connected to the same unit and to observe the state of said users i.e. if still connected or not. To achieve this, this module makes use of the **connected users database** that resides in the unit. This per unit database is the evolution of VANESS and works as the intra-network distributed naming service that was described before. Its role and functionalities are more thoroughly explained in the following chapter 3.4.1

The **beacon** module is responsible for periodically sending "hello messages" announcing the unit's presence to other units that are in communications range, and thus, within the capability of communicating with each other. The reason for this module is because units have no innate way of knowing when they get in communications range of other units. This module allows units to become aware of other units as they get in communications range.

The **pending messages database** contains messages that the unit received but could not deliver to their addressees. The **recent messages database** contains a list of the most recent messages that were received through the VANET. It has a fixed size, meaning that when it is full and a new message arrives, it will discard the oldest message and save the recently received one. Its goal is to prevent messages from looping through the network. A more comprehensive use and function of these databases is detailed in the chapter 3.4.4.

Additionally, so that RSUs can deploy the VBB themselves, they need to contain additional modules to yield these functionalities. RSUs contain the **VBB messages database**, and deploy a **web server** that runs a RESTful web service to provide an interface to access the resources on this database.

The **log** module, even though not necessary for the system to operate, is deployed in all units. The goal of this module is to maintain a database for all the messages that traverse a unit for possible statistical studies. It compiles all the text messages, VBB requests, session related requests and beacon messages. All these messages are stored in its full form and paired with the GPS coordinates from where the unit was located when the given message was logged by this unit.

3.3 Smartphone application

The smartphone application connects directly to the units that comprise the network. In the physical layer they communicate by means of their Wi-Fi interfaces. In the application layer it makes use of the REINVENT messaging system. This application acts as a client to the servers deployed by the units. On Figure 8 the components that comprise this application are depicted.

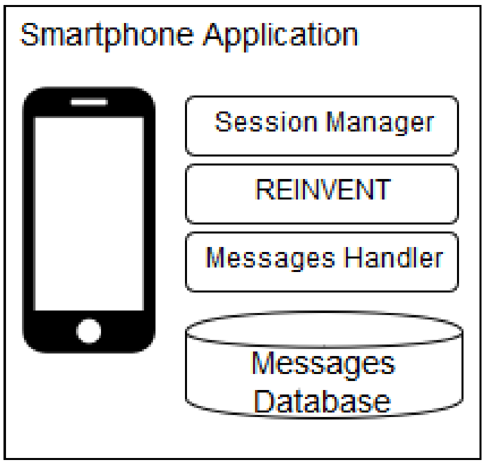


Figure 8 - Smartphone application components

The **session manager** is responsible for handling all the messages relative to establishing, maintaining, and ending a session with a unit. The **message handler** is responsible for sending the messages to the units, and for deploying the listeners pledged to receive the messages. Both the session manager and the message handler conduct their communications through REINVENT.

The **messages database** is a simple database were the messages sent and received (conversations) are stored, it's goal is to maintain a conversations log, just like any other chat application.

3.4 VNMS Protocols

As in any other communication protocol, VNMS uses a specific set of rules for sending messages from one node to another. Every message has an exact meaning intended to obtain a response from a set of pre-determined responses for that particular request. These protocols extend the REINVENT protocols to support its added features.

3.4.1 User session protocol

In order to connect to the system, the smartphone application needs to initially register with it. It is only after successfully registering that it will have full access to the system capabilities. Additionally, after being connected, keep alive messages are periodically exchanged in order to ensure that the application is still connected. On Figure 9 we can see the order and direction of the messages exchanged to achieve this.

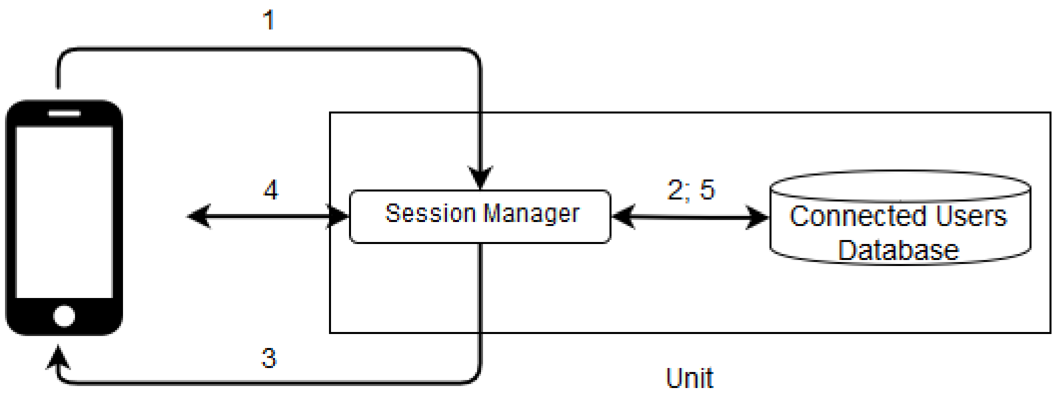


Figure 9 – Session protocol message exchanges

- ***Registering***

On Figure 9 we can see the flow of the messages exchanged for a user to register with a unit. Detailed information on the message formats and fields that comprise them can be found on chapter 8.1.5. The order and flow of the messages exchanged is the following:

1. The smartphone application sends a *subscribe* message to the unit containing user related information requesting to register. This (subscribe) operation is stateful, meaning that the application will be expecting a reply.
2. The unit adds an entry to the connected users database containing the *username* and a timestamp with the current system date.
3. After the entry has been successfully added to the database, the unit replies to the user's request with a simple acknowledge message notifying it that the registration has been completed. After this step the smartphone application becomes connected to the system.

Additionally, the user can request at any time to end the session by sending a *draft* message to the unit. After receiving such a message, the unit will delete the username's entry from the database and stops sending keep alive messages. This operation is *stateless*.

- ***Keep alive***

Keep alive messages are exchanges periodically between the smartphone application and the unit. These messages ensure the user is still connected. The order of these messages is the following:

4. Every 5 seconds, the unit sends a query to the smartphone application to check if it's still connected and waits 4 seconds for a response.
5. If the unit gets a reply it will update the timestamp of the *username* entry on the database.
5. If the unit waits 4 seconds and doesn't get a reply it will delete the *username* entry form the database.

Due to the time constraints, namely the 5 seconds period between queries and the 4 seconds the unit waits before considering that the query as timed out, it can take up to 9 seconds for a unit to realise that a user as disconnected.

3.4.2 Text Message sending protocol

This protocol describes the flow and the messages exchanged that the system uses to deliver a text message sent from one user to another. Although very similar, there are two distinct situations that can occur and difference resides on where the addressee is connected to i.e. if to the same unit or to a different unit. Detailed information of the message formats and fields that comprise them can be found on chapter 8.1.3.

- ***Text message to a user connected to the same unit***

On Figure 10 the order of operations and exchanged messages that occur when one user sends a text message to another user connected to the same unit is depicted.

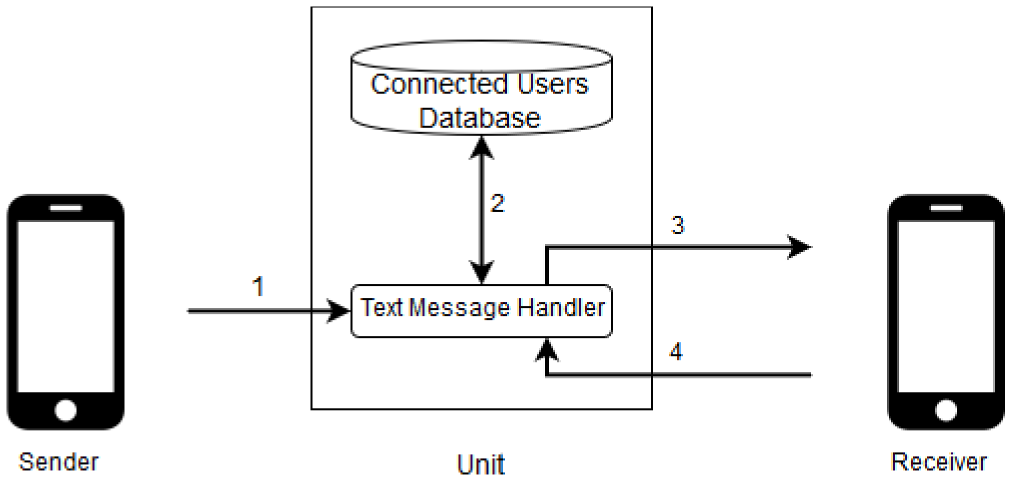


Figure 10 – Sending a text message to a user connected to the same unit.

1. The message sender sends a text message to the unit. This operation is stateless.
2. The text message module checks if the addressee is connected to if by searching for its entry on the connected users database.
3. After confirming that the addressee is connected, the unit will send the message to him. This is a stateful operation as the unit will wait for a confirmation that the message was received.
4. After receiving the text message, the application sends an acknowledge message to the unit notifying it was successfully received.

The flow described before is the best case scenario on which everything goes according to plan. The unit checks if the user is connected to it and if it is the message will be sent, however as described before in the keep alive protocol it can take up to 9 seconds for a unit to realise that a given user has disconnected. If a message for a disconnected user arrives during this period, the system will try to deliver it nonetheless. This will result in the unit not receiving the acknowledge message from the application. In this case the unit will treat the message as if to a user connected to a different unit and will take action accordingly, as is described in the next chapter.

- ***Text message to a user connected to a different same unit***

On Figure 11 the order of operations and exchanged messages that occur when one user sends a text message to a user connected to a different unit.

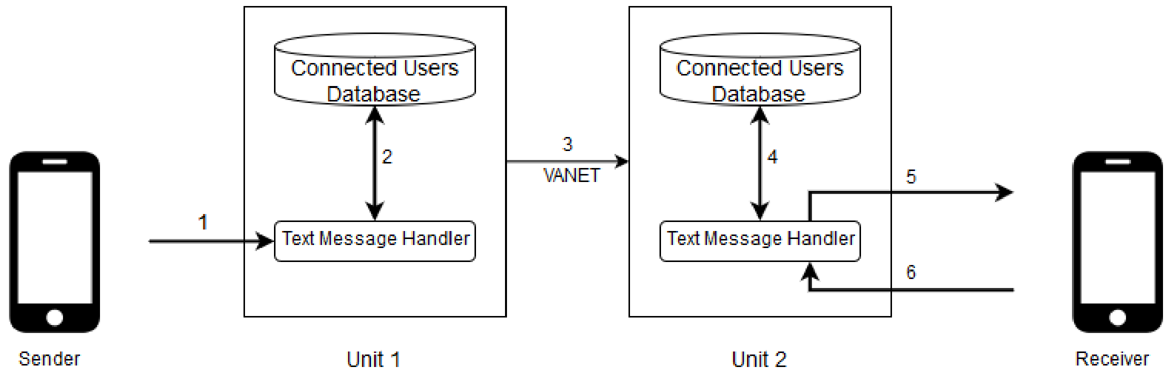


Figure 11 – Sending a text message to a user connected to a different unit.

1. The message sender sends a text message to unit 1. This operation is stateless.
2. Unit 1 will check if the addressee is connected to if by searching for its entry on the connected users database.
3. After ascertaining that the addressee isn't connected to it, unit 1 will broadcast the message through the VANET and save it on the pending messages database for future processing.

4. Unit 2 checks if the addressee is connected to if by searching for its entry on the connected users database.
5. After confirming that the addressee is connected, the unit will send the message to him. This is a stateful operation as the unit will wait for a confirmation that the message was received.
6. After receiving the text message, the application sends an acknowledge message to the unit notifying it was successfully received.

On step 3, it is stated the message is saved on the pending messages database for future processing. A detailed description of this processing is described on chapter 3.4.4.

3.4.3 Virtual Bulletin Board requests protocol

This protocol describes the flow of the messages destined to the RSUs' VBBs. These messages are requests to be made on the VBB. A user can make three types of requests, he can make a post request, a delete request or a get request. *Post* and *delete* requests do not trigger responses but *get* requests do, meaning that it will be necessary to deliver a response to the requester. Detailed information of the message formats and fields that comprise them can be found on chapter 8.1.4. On Figure 12 we can see the order and flow of message exchanges that occur when a request is made to a VBB.

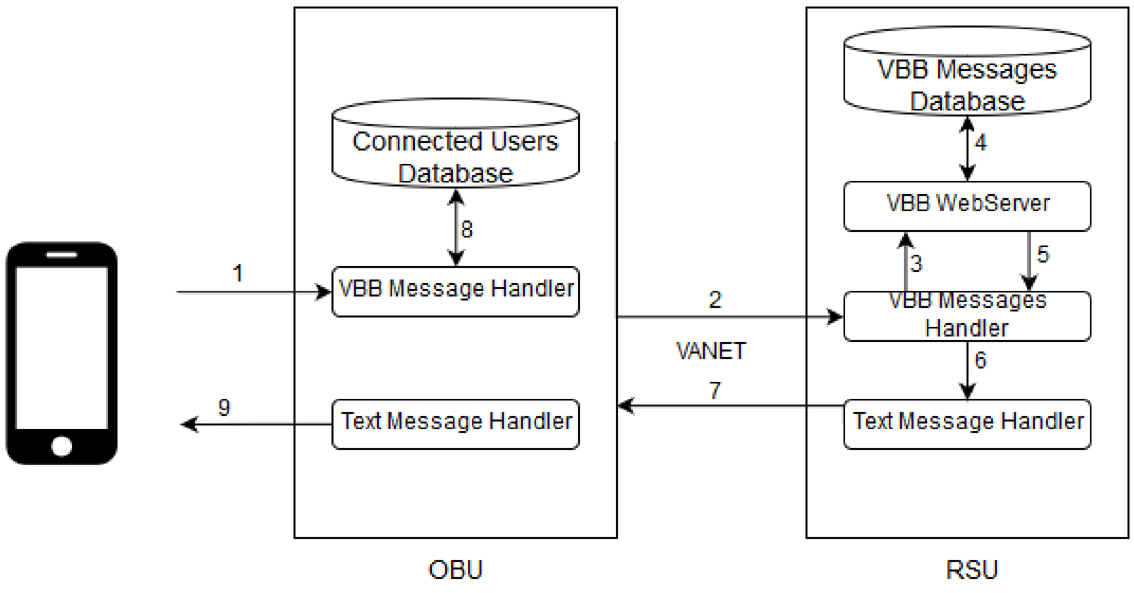


Figure 12 – Performing a request on a virtual bulletin board.

1. The requester sends a message to the unit. This message contains the request type as well as all the additional information relevant to the specific request.

2. OBUs will invariably forward VBB requests to the VANET in order for them to reach a RSU.

3. The RSU's VBB message processor will translate the request message into a REST request and will perform it on the unit's web server.

4. The web server executes the REST request and depending on its type will perform the adequate operation in the unit's VBB messages database. In case of a *post* or *delete* request, this flow end here and the operation is finished.

5. In case of a *get* request, the web server will send a reply to the message processor's request made in 3. This reply will be a list of messages that match the request criteria.

6. The message processor will convert all the messages in the list into modified text messages and will forward these messages to the text message module who will be in charge of delivering them to the requester. After this stage, as these messages are only but a modified version of regular text messages, the same methodology used to deliver regular text messages is used to deliver them as has been described in the previous chapter 3.4.2.

3.4.4 Message Retransmission and VANET Forwarding

In all the previously described protocols we assumed that, when a message is sent through the VANET, it would indubitably arrive to another unit and/or subsequently to the desired recipient. A message would be sent through the VANET and another unit would process it accordingly. Furthermore, as described before, sometimes messages were stored in the pending messages database for future processing. In order to assure that these messages are delivered to their recipients, a complex mechanism takes place in the units. Figure 13 depicts all the unit's modules that play a role in this course of actions.

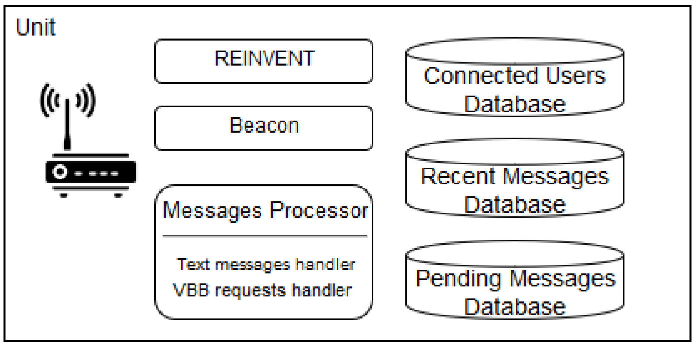


Figure 13 – Message retransmission and VANET forwarding modules

When the message processor receives a message from the VANET, it starts by checking if the message is on the recent message database. If the message is found there, it means that the unit recently received it ergo, it will take no action by simply discarding it. If the message is not found there, it will insert it on that database and will clear it to be processed. The main goal of the recent messages database is to disencumber the system and avoid processing repeated messages.

After the message has been cleared for processing, the message processor module forwards it to that message type specific module, and depending on this module, it will have a distinct treatment.

VBB requests are treated with a *best effort* approach, meaning that when a request arrives to the unit, it will simply be broadcasted through the VANET. Text messages are, in the ideal scenario, processed as described in chapter 3.4.2; however there are multiple reasons by which the message addressee cannot be reached. To circumvent this problem and ensure that the message is delivered, when a unit receives a text message and fails to deliver it to the addressee, either by him not being connected to that given unit or because of him failing to acknowledge the reception of that message, the unit will store that message in the pending messages database.

The pending messages database will contain all the text messages that arrived to a unit and that given unit did not deliver to its addressee. As the name describes, these are pending messages to be delivered, however messages on a data base will not deliver themselves. There are two events that can trigger a unit's attempt to deliver these messages:

- **Addressee connecting to the unit –** When a user connects to a unit, this unit will check the pending messages database for messages destined to that user and will try to send them to him. Messages will be deleted from the pending messages database only if the recipient acknowledges their reception.
- **Another unit in range of transmission –** This case is where the beacon module comes into play. When a unit receives a beacon message from another unit, it will broadcast all the messages in the pending messages database to the VANET with the intent of delivering them to other units to which the user might be connected, so that they can forward these messages to other units.

In any of these cases, when a message is retrieved from the pending messages database, its validity field (described in chapter 8.1.2) will be checked to ascertain if the message is still valid. Only if the message is still valid it will be transmitted, if the message is deemed invalid it will be deleted from the database and will be discarded. Furthermore, on the case that another unit comes in range, the broadcast will occur after the beacon message is received; however, to prevent a network message flooding and reduce bandwidth usage, this broadcast will not happen every time a unit receives a beacon message from another unit. When a unit receives a beacon message, the beacon module will check when was the last time that it received a beacon message from that same unit. The pending messages broadcast will only occur if a determined time has elapsed since the last time it received a beacon from that other specific unit.

In this scenario it is possible that a user changing vehicles, and thus changing the unit from which he connects to the system, to connect to a unit that contains previously received messages in its pending messages database. In this case it falls down to the smartphone application to handle the duplicated messages and discard them, which will be able to do by making use of the message ID.

4 VNMS Implementation

In this section, using the VNMS blueprint architecture, we describe in detail the main decisions and implementation solutions namely new contributions (Virtual Bulletin Board, user messaging support) and evolutions of the previous VANET related work REINVENT and VANESS (message protocols, naming services), the VNMS implementation reference. In the end, we focus on VNMessenger, the smartphone application that relies on VNMS for message exchange in VANETs.

4.1 RabbitMQ as message broker

RabbitMQ is a messaging system that enables software applications to connect and exchange data between them. It supports messages over a variety of messaging protocols and can be deployed using a diverse collection of programing languages, allowing it to be used in different platforms with different architectures [22]. The way RabbitMQ works is by creating message exchanges where the messages are published, and by configuring these exchanges to work the way its intended, they will send the messages through message queues to the desired recipients. The methods used by VNMS are [23]:

- **Publish/Subscribe –** This method provides the ability to send the same message to an assortment of users at the same time. The exchange used on this method is a *fanout* type exchange, meaning that after a message is published on it, it will create a message queue for every user that subscribed to it and it will forward the same message to all of those users through those queues.
- **Routing –** This method provides the ability to selectively send a message to a given listener. The exchange used on this method is a *direct* type exchange. These type of exchanges require additional information to properly send the message to the correct listener. When a listener subscribes to a *direct* type exchange it will have to provide a *binding key* in which the listener identifies itself. Additionally, when a publisher publishes a message on a *direct* type exchange it will have to designate the destination's *binding key*. When the message is published on the exchange with a *binding key*, the exchange will create a message queue to the user that subscribed to it using the same *binding key* and will send the message through it to that determined listener only.

- **RPC (Remote Procedure Calls) –** RPC is a protocol that one application can use to request a service to another application (usually located in another machine), and after the request being processed, the caller will receive a response to the request it made. This method provides the ability for callers to execute requests on the servers and seamlessly receive a response to this request. To achieve this, initially, the caller creates an anonymous and exclusive call-back queue through which it will receive the reply to the request. Afterwards the caller publishes the request on the RPC queue describing on it the call-back queue and the correlation id, this id is used to unequivocally identify the request. The server to which the request was made is listening for requests in the RPC queue, after receiving a request, the server processes it and when the process has been made, it uses the call-back queue to deliver the result. This result is paired with the correlation id the caller defined. Lastly the caller receives the result to the request on the call-back queue and by use of the correlation id it will associate the response to its respective request.

4.2 Extending REINVENT

VNMS uses REINVENT as the basis for its communication system; however, the limitations of REINVENT needed to be addressed so that VNMS could achieve its goals. The main limitations imposed by REINVENT were the inability to support a multiple number of users connected to the same unit and the inability to keep track of the users' states (connected/disconnected). REINVENT uses the RabbitMQ's framework to support the messaging service, and its limitations were imposed due to the fact that it only used simple message queues made for communications with a single recipient. VNMS extends REINVENT by adjusting its communication channels and creating some additional ones. Figure 14 depicts the new and refurbished communication channels that were added to the system. These queues and exchanges cover the communication between units and the smartphone application.

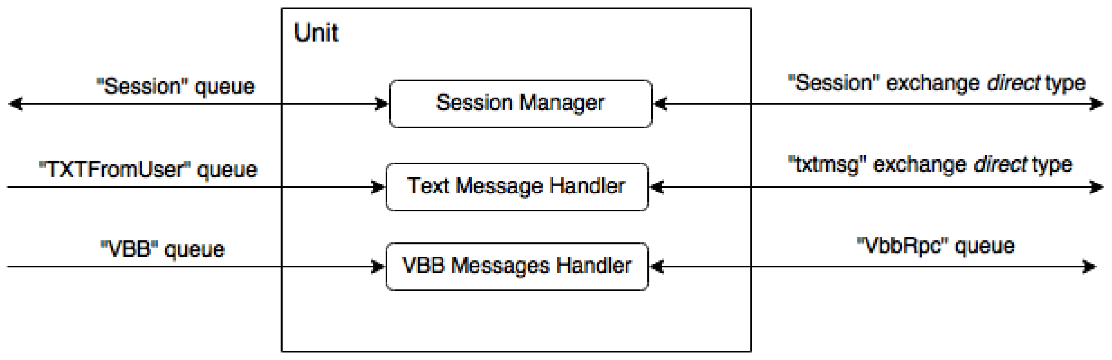


Figure 14 – Queues and exchanges deployed by the units

- **"Session" message queue –** this queue is where the unit will be listening for session related requests sent by the smartphone application. In this queue it will receive session *subscribe* and *draft* RPCs. This is just a basic queue as messages that are sent through it are only destined to the unit, as an analogy, it can be considered as a "well known port" to which the smartphone applications send their session related requests.
- **"Session" exchange –** this exchange is used by the unit to keep tabs on the users' states i.e. if they are still connected or not. It's through this exchange that the unit performs the periodical keep alive RPCs on the smartphone application. It's a *direct* type exchange because it's used to send messages to specific users.
- **"TXTFromUser" message queue –** this queue is where the unit receives text messages to be sent to users. These messages can be received from the smartphone application, and additionally, text messages that arrive to a unit over the VANET will also be sent to this text message module by means of this queue. This is just a basic queue as messages that are sent through it are only destined to the unit, as an analogy, it can be considered as a "well known port" to which the smartphone applications send outgoing text messages.
- **"txtmsg" exchange –** this exchange is used to send text messages to the users. Text messages will be sent to the users as RPC as the unit will be expecting a reply confirming that the message was received. It's a *direct* type exchange because it's used to send messages to specific users.
- **"Vbb" message queue –** in this queue, units will be listening for VBB request messages. Messages can arrive to this queue originating the smartphone application or from the VANET. This is just a basic queue as messages that are sent through it are only destined to the unit, as an analogy, it can be considered as a "well known port" to which the smartphone applications send the VBB requests.
- **"VbbRPC" message queue –** this queue is used to perform requests pertaining the VBB that require a response. It's through this queue that the smartphone application performs the operations to check the VBB messages file version and to obtain this messages file.

The communication between units in VNMS is the same as in REINVENT. It uses the Wave Short Message Protocol (WSMP), a service defined in the IEEE 1609.3 that allows requests from higher layers, such as applications, for sending messages over the air by MAC Address in both unicast and broadcast. REINVENT uses the control channel and sends messages with the Provider Service Identifier (PSID) 80-01. To receive messages, the unit's main module starts a daemon that listens for incoming Wave Short Messages (WSM) with the defined PSID using the command:

```
uswmp receiveWSM psid 80-01
```

By opening a pipe with this command, it allows the unit’s main module to read from the Standard Output (stdout) of the daemon where it will write the received WSMs. After receiving a message, the module will parse it into a message that can be interpreted by the message processor. Sending WSMs is achieved by issuing the command:

```
usmp sendWSM psid 80-01 amount 1 msg [message body]
```

4.3 Unit’s application

The unit’s client is responsible for instantiating and starting all the modules that it requires to run. It starts by establishing a connection to all the databases it deploys. In case these databases don’t exist, whether it be because they were deleted or it’s the first time the client is running, it will create them first and then it will connect to them.

Afterwards, the client will instantiate all the modules required: VBB requests handler, text messages handler, beacon, message processor, and the session manager. All of these modules except the message processor will have their own thread of execution as they need to be kept running simultaneously and continuously.

The VBB requests handler, the text messages handler, and the session manager deploy their respective queues and exchanges, the listeners responsible for retrieving these messages, and the call-backs responsible for processing them. The beacon module will start a periodic thread that emits the beacon messages. It also instantiates the dictionary were the timestamps of when it received a beacon from another unit are stored.

Finally, the unit starts the daemon responsible for attaining the messages from the VANET. To send these messages to the respective modules, it will invoke the message processor passing it the received message.

4.4 Units' databases

Units contain several databases, namely the: connected users database, pending messages database, recent messages database, and in the case of RSUs, the VBB messages database. The beacon dictionary could also be treated as one but as it is internal to that specific module it will not be considered in this section.

- **Connected users database** – stores a list of the users connected to the unit. Its entries contain the username and a timestamp detailing the last time they responded to a keep alive request.
- **Pending messages database** – stores all the messages that the unit didn't deliver to the addressee. Its entries contain the username of the addressee, an internal message id, and the message itself.
- **Recent messages database** – this is not a database per se, but was implemented as a circular buffer. When it full and receives a new message it will discard the oldest one.
- **VBB messages database** – this is where the messages posted on a RSU's VBB will be stored. A more detailed overview of this database can be found on the following chapter.

4.5 Virtual Bulletin Board

The VBBs are deployed on the RSUs and run as a separate application to the unit's client. These VBBs are comprised of a database where the messages are stored, and of a web server that exposes a web service to interact with the database.

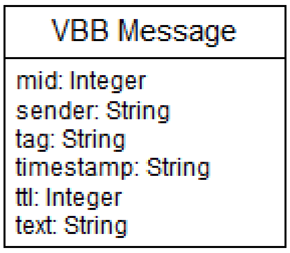


Figure 15 - Class diagram of a VBB message entry

Figure 15 depicts a class diagram of the records that can be stored in the database. Its attributes are:

- **sender –** the username of who posted the message;
- **mid –** the message identifier in conjunction with the sender attribute form a unique identifier for the message.
- **tag -** used to identify the destination of the message. If a message is destined to a single user, the *tag* field will be the destination's *username*; however, the system provides the functionality of sending a message to a group of users. To achieve this,

instead of the destination's *username*, the *tag* field will be a group ID known to all the members of a group. When a user wants to retrieve messages sent to it on a given VBB, it will make a GET request using its *username* as the *tag*. If a user wants to retrieve the messages from a group, it will use the group ID as the *tag*. Detailed information of these message requests can be found on chapter 8.1.4.

- **timestamp –** a timestamp with the time of when the message was sent.
- **ttl (time to live) –** this field is used to define the expiration date of the message. It stores a value in minutes. When combined with the timestamp the system can determine if the message is still valid.
- **text** – this field will contain the message itself. These messages however will be encoded using a known format that will be discussed further.

The web server is a separate application to the unit's client. It runs a web service that provides a RESTful interface to interact with the database and achieves this by exposing 3 methods:

- **Get messages** – used to get messages destined to a user or group.
  - **URL:** `/webserver/getMessages?tag=:tag`
  - **Method:** `GET`
  - **URL Params:** `tag=[string]`
  - **Success Response:** the responses to this requests will either be an empty string in case there are no messages destined to the requester or it will be a list containing all the messages. The items on this list are strings separated by the "`|`" (pipe) character and comprised of a message identifier, the timestamp from when the message was sent, the user name of the sender and finally the message itself. In case there are more than one message, they will be separated by the char sequence "`<p>`".
    - Ex:

```
    10|162601144735|chico|12<p>

    12|12|162601144735|joao|10#cantina#10m<p>

    162601144735|toni|16#friend
```

- **Add messages –** used to store messages on the database.
    - **URL:** `/webserver/addMessage?mid=:mid&sender=:sender&tag=:tag&timestamp=:timestamp&ttl=:ttl&text=:text`
    - **Method:** `PUT`
    - **URL Params:**
        - `mid=[integer]`
        - `sender=[string]`
        - `tag=[string]`
        - `timestamp=[integer]`
        - `ttl=[interger]`
        - `text[string]`
    - **Success Response:**
        - `Message stored.`

- **Delete messages –** used to delete a message.
    - **URL:** `/webserver/removeMessage?mid=:mid&sender=:sender`
    - **Method:** `DELETE`
    - **URL Params:**
        - `mid=[integer]`
        - `sender=[string]`

As mentioned before, the messages stored on this database will be encoded. Furthermore, this VBB only supports the storage of pre-defined messages that can contain some editable fields. This restriction is imposed by the WSMP and the limited size of the messages that it can send. VNMS uses a set of pre-defined messages that need to be known to all the system users. These messages are distributed by the system in a JSON file with the following structure:

```
{
   "version" : 3,
    "messages": [
        {
            "ID": 10,
            "text": "I will be at # for the next #.",
            "efields": 2
        },
         {
            "ID": 12,
            "text": "Hello #! Hit me up for #. I can be found at #.",
            "efields": 3
        },
         {
            "ID": 13,
            "text": "Good Morning.",
            "efields": 0
        },
         {
            "ID": 14,
            "text": "Hello #!",
            "efields": 1
        }
    ]
}
```

When the smartphone application connects to a unit it will verify the version of its message file and the version of the file on the unit. If the unit contains a newer version, the application will retrieve it. This file needs to be maintained and updated throughout the system in order to ensure a correct behaviour. With this predefined format, messages can be sent using less data and containing more information. A message encoded in this format will only contain the ID of the message and the values of the editable fields. To decode these messages, it is a simple matter of looking for its ID in the file and concatenating its editable fields with the text of the message on the file.

Ex:

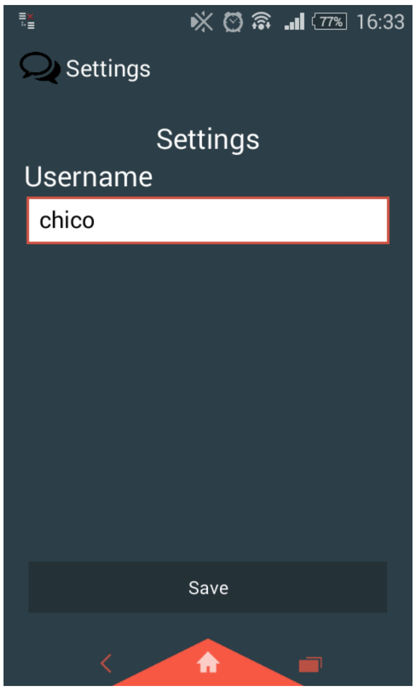
- Encoded: `10#bar do deti#half hour.`

  Decoded: `"I will be at bar do deti for the next half hour."`
- Encoded: `13`

  Decoded: `“Good Morning.”`

4.6 Smartphone application - VNMessenger

To use VNMS, a smartphone application was developed for the Android platform called VNMessenger. When executed for the first time, this application stars by prompting the user to insert a username as is depicted on Figure 16. This username is what the application will use to register in the system.


16:33

Settings

Settings

Username

chico

Save

Figure 16 – VNMessenger prompting for the username

After saving the username, the application goes to its main activity[1]. From henceforth the first run, the application will always start in this main activity that is shown on Figure 17.

[1] An Activity is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map. Each activity is given a window in which to draw its user interface [31].

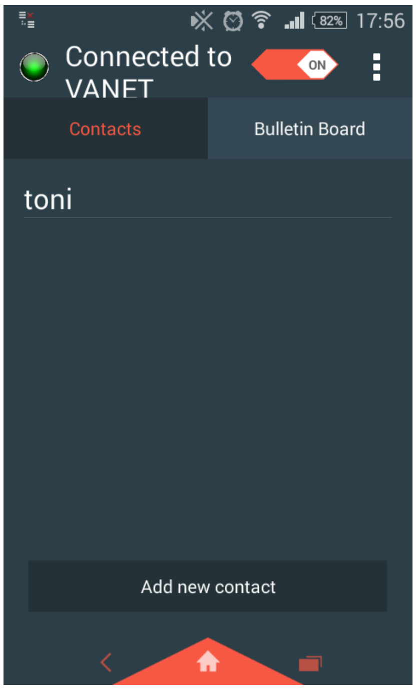
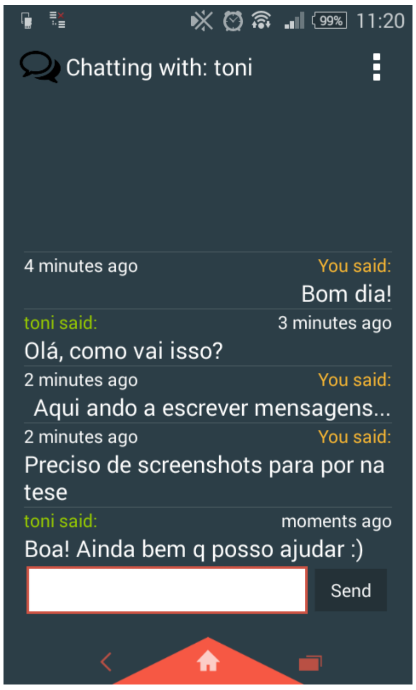


Figure 17 – Main activity

Here the user can set his state (online or offline), can view its contacts and its VBB subscriptions (a subscription is either a contact or a group).

In this activity the user has the ability to switch between the contacts tab where he can send text messages to its contacts or the bulletin board tab where he can perform operations on the VBB. In the contacts tab the user can add new contacts so that he can send messages to him. These contacts will also be added to the bulletin board subscriptions list so that the user can post messages in the VBB directed to that given contact only.


Chatting with: toni

4 minutes ago You said:
Bom dia!

toni said: 3 minutes ago
Olá, como vai isso?

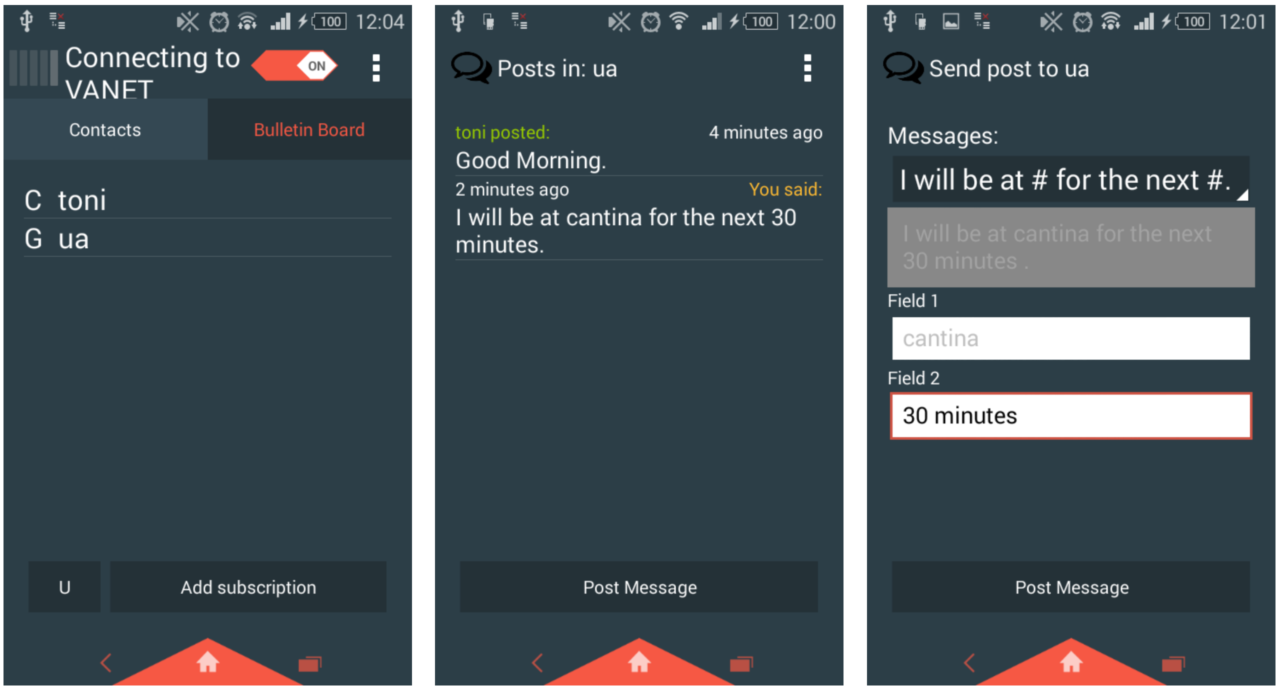
2 minutes ago You said:
Aqui ando a escrever mensagens...

2 minutes ago You said:
Preciso de screenshots para por na tese

toni said: moments ago
Boa! Ainda bem q posso ajudar :)

Figure 18 – VNMessenger chat activity

By tapping a contact name on the contacts list, the application will move to the chat activity and the user will be able to send messages to it, and view the messages that received from it.


Connecting to VANET

C toni
G ua

Posts in: ua

toni posted: 4 minutes ago
Good Morning.

2 minutes ago You said:
I will be at cantina for the next 30 minutes.

Send post to ua

Messages:
I will be at # for the next #.
I will be at cantina for the next 30 minutes .

Field 1
cantina

Field 2
30 minutes

Figure 19 – Bulletin board activities (subscriptions list, group posts, and posting a message)

On the bulletin board tab, the user has the ability of adding groups as subscriptions as shown on Figure 19. These groups are to be shared between users, and any user in this group can post and read messages to and from. By tapping on one of these subscriptions, the user can view the messages stored on that group and can post a message on it. These are the predefined messages described on the previews chapter. The user can choose which message he wants to post on the dropdown menu and depending on the number of editable fields, the application will display text boxes where he can insert the text. A preview of the message is also available.

4.7 Messages log

The messages log deployed on the units creates a database where all the messages that traverse a unit are stored, being text, VBB requests, session control, and beacon messages. All the log entries in this database are paired with the GPS coordinates of the unit location upon the log of the given message.

Messages are stored in this database directly by the unit's modules; however, to access this database knowledge, an additional web service that deploys a RESTful web service can be deployed on the units. This web service is not needed for logging purposes and can be executed only for the knowledge retrieving purpose and shut down right after, as this is not considered a frequent operation and thus alleviating the unit's processing unit.

The webserver runs a web service that provides a RESTful interface to interact with the database and achieves this by exposing the method:

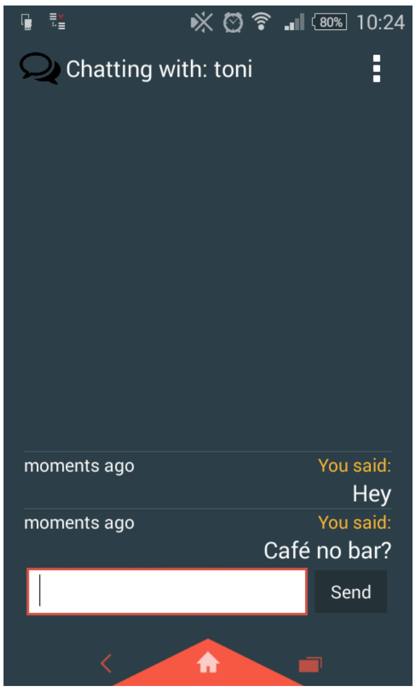
- **Get messages** – used to get messages destined to a user or group.
  - **URL:** `/webserver/getLog?tag=:tag`
  - **Method:** `GET`
  - **URL Params:** `tag=[string]`
  - **Tag:** the tag field can have the string values: `messages`, `session`, `vbbrequests`, or `beacon` and depending on it, it will return the corresponding entries.
  - **Success Response:** the responses to this request will either be empty in the case that there are no messages logged or a JSON file containing all the messages relative to that specific request.

No GUI application is available to access this database, but it can easily be accessed through any web browser if the device is connected to the unit.

5 Results

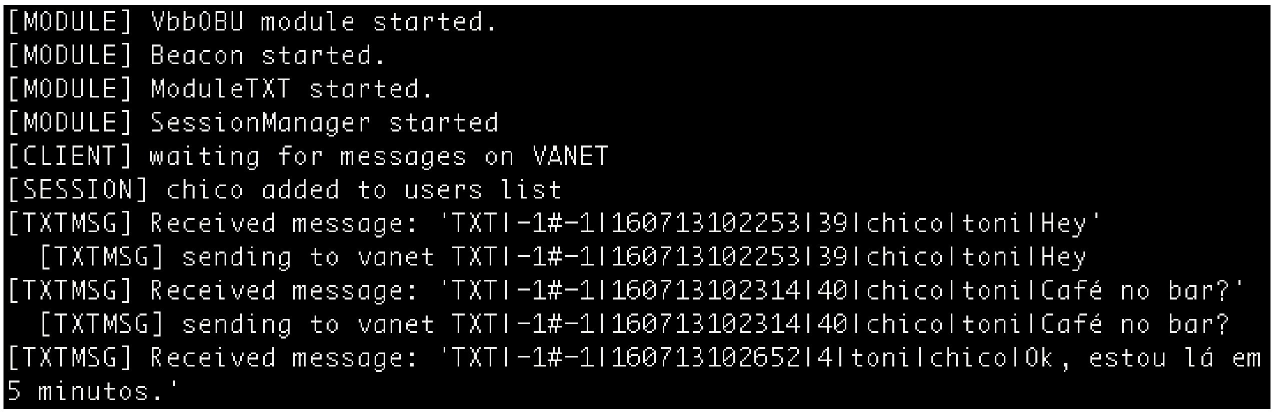
The focus of VNMS was on the ability to exchange user to user messages in the VANETs. In this section we use a scenario of message exchange between two users in different vehicles (i.e. OBUs) to track the VNMS flows of message exchange.

The scenario starts with the user “chico” connecting to the OBU. User “chico” sends two messages (as can be seen on Figure 20) to user “toni”. As “toni” is not connected to that unit, the messages are broadcasted into the VANET as is depicted on Figure 21.



moments ago You said:
Hey
moments ago You said:
Café no bar?

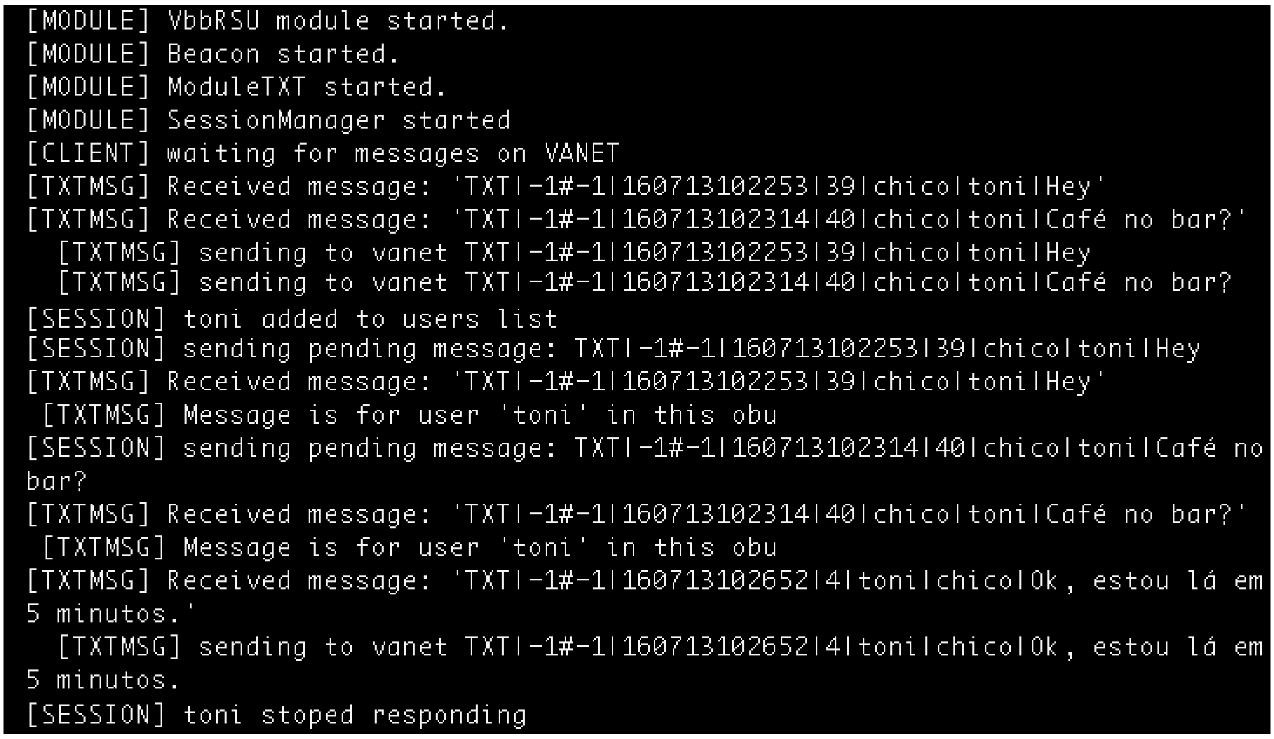
Figure 20 – User “chico” chatting with “toni”



```
[MODULE] VbbOBU module started.
[MODULE] Beacon started.
[MODULE] ModuleTXT started.
[MODULE] SessionManager started
[CLIENT] waiting for messages on VANET
[SESSION] chico added to users list
[TXTMSG] Received message: 'TXT|-1#-1|160713102253|39|chico|toni|Hey'
  [TXTMSG] sending to vanet TXT|-1#-1|160713102253|39|chico|toni|Hey
[TXTMSG] Received message: 'TXT|-1#-1|160713102314|40|chico|toni|Café no bar?'
  [TXTMSG] sending to vanet TXT|-1#-1|160713102314|40|chico|toni|Café no bar?
[TXTMSG] Received message: 'TXT|-1#-1|160713102652|4|toni|chico|Ok, estou lá em
5 minutos.'
```

Figure 21 – OBU output during chat.

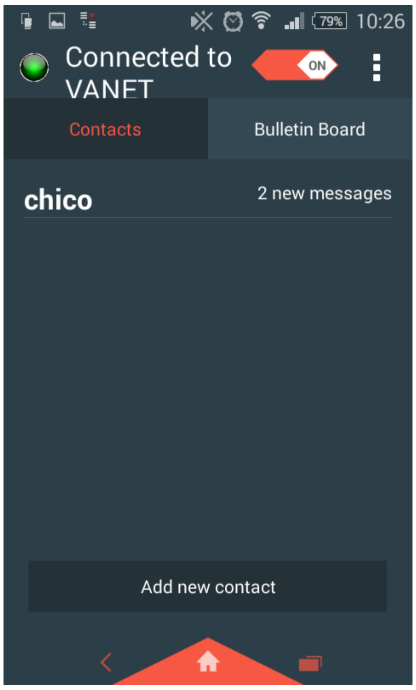
The messages arrive to another unit, in this case a RSU; however, user “toni” is not connected to it, and this unit will also broadcast these messages into the VANET as can be seen on Figure 22.

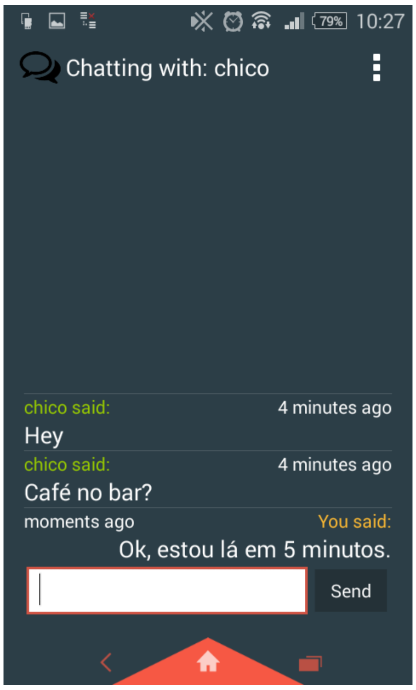


```
[MODULE] VbbRSU module started.
[MODULE] Beacon started.
[MODULE] ModuleTXT started.
[MODULE] SessionManager started
[CLIENT] waiting for messages on VANET
[TXTMSG] Received message: 'TXT|-1#-1|160713102253|39|chico|toni|Hey'
[TXTMSG] Received message: 'TXT|-1#-1|160713102314|40|chico|toni|Café no bar?'
  [TXTMSG] sending to vanet TXT|-1#-1|160713102253|39|chico|toni|Hey
  [TXTMSG] sending to vanet TXT|-1#-1|160713102314|40|chico|toni|Café no bar?
[SESSION] toni added to users list
[SESSION] sending pending message: TXT|-1#-1|160713102253|39|chico|toni|Hey
[TXTMSG] Received message: 'TXT|-1#-1|160713102253|39|chico|toni|Hey'
 [TXTMSG] Message is for user 'toni' in this obu
[SESSION] sending pending message: TXT|-1#-1|160713102314|40|chico|toni|Café no
bar?
[TXTMSG] Received message: 'TXT|-1#-1|160713102314|40|chico|toni|Café no bar?'
 [TXTMSG] Message is for user 'toni' in this obu
[TXTMSG] Received message: 'TXT|-1#-1|160713102652|4|toni|chico|Ok, estou lá em
5 minutos.'
  [TXTMSG] sending to vanet TXT|-1#-1|160713102652|4|toni|chico|Ok, estou lá em
5 minutos.
[SESSION] toni stoped responding
```

Figure 22 – RSU output during chat

Eventually, user “toni” connects to this RSU and as he connects, the unit sends him the pending messages it has for him. Its smartphone application notifies him of these new messages as can be seen on Figure 23. After reading the messages, “toni” replies to “chico”. As “chico” is not connected to this unit, the message is sent through the VANET and arrives to the unit where “chico” is connected as is depicted on Figure 21.





chico said: 4 minutes ago
Hey
chico said: 4 minutes ago
Café no bar?
moments ago You said:
Ok, estou lá em 5 minutos.

Figure 23 – User “toni” gets messages from user “chico”

As “toni” leaves the range of the unit to which he was connected (possibly to go to the bar) the unit realises he stopped responding and thus becomes aware that he disconnected.

6 Conclusions and Future work

The Vehicular Network Messaging System (VNMS) is the main contribution of the current work. VNMS allows user to user message exchange on a VANET. By deploying virtual bulletin boards (VBBs) in VANETs nodes, a layer of message forwarding and user naming service, it provides users the ability to exchange messages without the explicit need of any VANETs specific information or service. VNMS nodes act as user message brokers, providing local user message repositories and VANETs routing to targeted user(s) i.e. its VANET node. From the user perspective, it is possible to use VNMS services transparently using Android mobile application, VNMessenger. We implemented a VANETs enabled chat application as proof of concept.

As VNMS is the natural evolution of REINVENT and VANESS, it addresses some issues previous identified in these solutions.

First it provides a user level abstraction to REINVENT avoiding the user to deal with VANETs specific details – REINVENT implied using OBU’s VANET address explicitly for message routing and did not support message persistence – currently handled by VNMS VBBs.

VNMS evolves the VANESS naming solution. VANESS assumed a centralized naming service on the internet and distributed caching on the VANETs. This implied the system to be dependent on resources external to the VANET (the centralized name service in the internet) and to have a temporal consistency issue on local naming service replicas on VANETs nodes vs the centralized naming service. VNMS eliminates the need of a VANET external resource, as the VANETs nodes rely on no external resource, but does not solve completely the consistency issue – a partial solution is provided via message exchange between VANETs node (with some piggy backing) to update naming service user to VANET node mapping. Although not ideal, it ensures local consistency between connected VANETs nodes.

The system contains some limitations that must be attended before being used in a real world scenario. It needs to provide a mechanism for user registering to ensure that there are not multiple users with the same *username*; this is used as the *key* used to ascertain the location of a user. If multiple users using the same *username* were connected to the network, any message sent to one of them would be sent to all of them.

An authentication system needs to be added to the Virtual Bulletin Board. Currently any user can view any message posted on these boards, it only needs to know the group’s *tag* to do it.

Currently, the only way to use the system is through the Android application developed for this purpose and through a rudimentary java console application that was created for debugging and testing purposes. However, in order for it to reach a broader group of users, GUI applications for other platforms need to be created. The system was designed to have a high interoperability allowing that applications for other platforms can interact with it seamlessly.

VNMS implements a messaging protocol with well-defined messages. These messages are basic text strings and their headers are also strings. To reduce the message overhead, these headers could be encoded in binary and thus reducing their size significantly. Also using string compression algorithms would achieve a reduced bandwidth usage in the data segments of these messages.

The log system implemented on this solution was created solely for the purpose of studying the network usage, system loads and other statistical information that can help to understand the network better; however, currently it is logging the messages in its totality, headers and data segments alike. If the system were to be deployed on real world usage, this logging system would need to be adapted to log the relevant information needed and maintain the privacy of its users by removing any data that could compromise this.

7 References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications,” *IEEE Commun. Surv. Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[2] A. Whitmore, A. Agarwal, and L. Da Xu, “The Internet of Things - A survey of topics and trends,” *Inf. Syst. Front.*, vol. 17, no. 2, pp. 261–274, 2015.

[3] Y. J. Li, “An Overview of the DSRC / WAVE Technology,” *7th Int. Conf. Heterog. Netw. Qual. Reliab. Secur. Robustness*, pp. 544–558, 2012.

[4] J.-C. Kao, “Research interests.” [Online]. Available: http://www.cs.nthu.edu.tw/~jungchuk/research.html. [Accessed: 12-Jul-2016].

[5] F. F. de Oliveira, “REINVENT: accessing Vehicular Networks in Mobile Application,” Master’s Thesis, Universidade de Aveiro, 2013.

[6] P. J. C. dos Santos, “VANESS: DNS for nomadic users in vehicular networks,” Master’s Thesis, Universidade de Aveiro, 2014.

[7] K. Lee, L. Uichin, and M. Gerla, “Survey of Routing Protocols in Vehicular Ad Hoc Networks,” *Inf. Sci. Ref.*, pp. 149–151, 2010.

[8] R. Kumar and M. Dave, “A Comparative Study of Various Routing Protocols in VANET,” *Int. J. Comput. Sci.*, vol. 8, no. 4, p. 6, 2011.

[9] B. Paul, M. Ibrahim, and M. Abu Naser Bikas, “VANET Routing Protocols: Pros and Cons,” *Int. J. Comput. Appl.*, vol. 20, no. 3, pp. 28–34, 2011.

[10] V. Kumar, S. Mishra, and N. Chand, “Applications of VANETs: Present & Future,” *Sci. Res.*, vol. 05, no. 01, pp. 12–15, 2013.

[11] W. Fehr, “DSRC: The Future of Safer Driving.” [Online]. Available: http://www.its.dot.gov/factsheets/dsrc_factsheet.htm. [Accessed: 06-Jul-2016].

[12] M.-A. Lèbre, F. Le Mouël, E. Ménard, J. Dillschneider, and R. Denis, “VANET Applications: Hot Use Cases,” *HAL ArXiv ID 1407.4088*, 2014.

[13] U. Nagaraj and P. Dhamal, “Broadcasting Routing Protocols in VANET,” *Netw. Complex Syst.*, vol. 1, no. 2, 2011.

[14] R. S. Battula and S. Dutt, “A Review of Location-Based Geographic Routing Protocols for Wireless Sensor Networks,” *Int. J. Eng. Res. Technol.*, vol. 2, no. 6, pp. 1170–1174, 2013.

[15] K. Jahanbakhsh and M. Hajhosseini, “Improving Performance of Cluster Based Routing Protocol using Cross-Layer Design,” *Arxiv Prepr. arXiv0802.0543*, pp. 1–8, 2008.

[16] S. Kamboj and S. Chawla, “Geocast Routing in Vehicular Ad Hoc Networks :,” *Int. J. Comput. Sci. Inf. Technol.*, vol. 5, no. 4, pp. 5365–5370, 2014.

[17] C. Maihofer, “A survey of geocast routing protocols,” *IEEE Commun. Surv. Tutorials*, vol. 6, no. 2, pp. 32–42, 2004.

[18] F. F. de Oliveira, S. Sargento, J. Fernandes, and A. Cardote, “REINVENT: accessing Vehicular Networks in Mobile Application,” *IEEE Int. Symp. Comput. Commun.*, 2014.

[19] R. T. Fielding and R. N. Taylor, “Principled Design of the Modern Web Architecture,” *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115–150, 2002.

[20] “AMQP.” [Online]. Available: https://www.amqp.org. [Accessed: 19-Jul-2016].

[21] P. Santos, S. Sargento, and J. Fernandes, “VANESS: DNS for Nomadic Users in Vehicular Networks,” *2th Int. Conf. Mob. Ubiquitous Syst. Comput. Netw. Serv. (ACM MOBIQUITOUS 2015)*, 2015.

[22] “What can RabbitMQ do for you?” [Online]. Available: https://www.rabbitmq.com/features.html. [Accessed: 07-Jul-2016].

[23] “RabbitMQ Tutorials.” [Online]. Available: https://www.rabbitmq.com/getstarted.html. [Accessed: 07-Jul-2016].

[24] “Python.” [Online]. Available: https://www.python.org. [Accessed: 18-Jul-2016].

[25] “Introduction to Pika.” [Online]. Available: http://pika.readthedocs.io/en/0.10.0/. [Accessed: 18-Jul-2016].

[26] “CherryPy.” [Online]. Available: http://www.cherrypy.org. [Accessed: 18-Jul-2016].

[27] “PyDbLite.” [Online]. Available: http://pydblite.readthedocs.io/en/latest/. [Accessed: 18-Jul-2016].

[28] “Requests: HTTP for Humans.” [Online]. Available: http://docs.python-requests.org/en/master/. [Accessed: 18-Jul-2016].

[29] “Android Studio.” [Online]. Available: https://developer.android.com/studio/index.html.

[30] “prettytime.” [Online]. Available: http://www.ocpsoft.org/prettytime/. [Accessed: 18-Jul-2016].

[31] “Activities - Android Developers.” [Online]. Available: https://developer.android.com/guide/components/activities.html. [Accessed: 11-Jul-2016].

8 Appendices

8.1 Protocol messages

Messages are sent/received by components through all the methods described before. However, all the data forming these messages must abide the formats defined so that each module/component can retrieve information from them. Like in any network protocol, these messages are comprised of a header and a data segment. The header of the message contains the control information and the data segment contains the message itself. All the messages in this system are text strings, and each field is separated by the pipe '|' character.

The message header contains numerous information necessary for the system, so that it can deliver the message to the correct destination. Additionally, the header contains information regarding the validity of a message.

8.1.1 Timestamp

Most of the messages used in the system will have a field called the timestamp, this field contains the time at which the message was sent. This field is comprised by a 12 characters string that can be parsed and converted in a date with precision up to seconds.

This field format is the following:

`yyMMddHHmmss`

yy – Last two numbers of the year;

MM – Number of the month;

dd – Day of the month;

HH – Hour of the day

mm – minutes

ss – seconds

An example would be:

14h 47m 35s, 26th January 2016 which would be encoded as: `160126144735`

8.1.2 Message Validity Field

When a unit receives a message for a user not connected to it, it will broadcast the message through the VANET to any other unit in range, in addition, when a unit receives a beacon from other unit, it will likely broadcast that message again. With all these unconditional broadcasts, messages were fated to circulate the network *ad infinitum*. To phase out this problem, messages are complemented with a validity field.

The validity field describes how many times and/or for how long a message can be broadcasted before being deemed invalid. When a message is deemed invalid, a unit will simply discard it. This validity field is comprised by two components:

- **Hop limit –** An integer value. This field specifies a limit on the number of hops a message is allowed before being discarded. Before a message is broadcasted, this number is decremented in one unit. When this value reaches zero, a unit will not broadcast this message again.
- **Time limit –** An integer value. This field specifies the amount of time a message is deemed valid after being sent. If the amount of time in this field elapses after the message was sent (can be known through the validity field), the units will deem the message invalid and will discard it.

Both these fields can be used at the same time or independently, and it is only necessary that one of them reaches its limit for the message to be discarded. For a message to be sent ignoring one of these fields, instead of a positive value for hops or time limit, this field should have the value `-1`. These inner fields are separated by the '#' character. The supported formats are the following:

`-1#-1` – message won't have either a hops limit or a time limit. It will always be broadcasted through the units.

`10#-1` – message will hop trough units a maximum of 10 times before being discarded.

`-1#15` – message won't have hops limit and, will only be broadcasted for 15 minutes after being sent.

`10#15` – message will hop trough units a maximum of 10 times or for a maximum of 15 minutes. If any of these conditions reaches its limit, the message will be discarded.

8.1.3 Text messages from users to users

These are the messages that a user sends directly to other user. In addition to the timestamp and the validity field, these messages include additional information necessary for it to reach its destination:

1. **Message type**: In this case will be the string "`TXT`". This field is used define the type of message;
2. **Validity field**;
3. **Timestamp**;
4. **Message ID**: This message id is generated by the sender's smartphone application and it's unique to each message;
5. **Sender username**: The *username* of the sender;
6. **Destination username**: The *username* of the addressee;
7. **Message text**: The message itself.

```
TXT|10#15|162601144735|33|chico|toni|good afternoon!
```

Message from "chico" to "toni" containing the text "good afternoon!".

8.1.4 Virtual Bulletin Board Messages

Messages sent to the VBB need to be translated in REST requests that a RSU will make to its webserver. Depending on the type or request, additional parameters are required. The header of these messages is comprised by the following fields:

1. **Message type**: In this case will be the string "`VBB`". This field is used define the type of message;
2. **Validity field**;
3. **Timestamp**;
4. **Message ID**: This message id is generated by the sender's smartphone application and it's unique;
5. **Command**: Can be either of the strings: "`PUT`", "`GET`", "`DELETE`", "`CHECKFILEVERSION`" or "`GETMESSAGESFILE`".

Depending on the type of request (command), the fields that follow the previous header vary according to each case.

- PUT → this command is followed by 3 additional fields: the sender ID, the tag field and, the message text.
    - Ex: `VBB|-1#-1|162601144735|33|PUT|chico|ua|12`
    - This message was sent by the user “chico” to the group named “ua” and contains the encoded message 12.

- GET → this command is followed by 2 additional fields, the ID of the user requesting the messages and, the tag of the messages the user wants to get.
    - Ex: `VBB|-1#-1|162601144735|34|GET|chico|ua`
    - This message was sent by the user “chico” requesting to get the messages posted with the tag “ua”

- DELETE → this command is followed by 2 additional fields, the ID of the user requesting the deletion of the message and the id of the message to be deleted.
    - Ex: `VBB|-1#-1|162601144735|35|DELETE|chico|33`
    - This message was sent by the user “chico” requesting to delete the message with the id 33.

- CHECKFILEVERSION → this command is used to check the version of the pre-defined messages file on a unit.
    - Ex: `VBB|-1#-1|162601144735|35|CHECKFILEVERSION|chico`
    - This message was sent by the user “chico” requesting the version of the messages file.

- GETMESSAGEFILE → this command is used to check the version of the pre-defined messages file on a unit.
    - Ex: `VBB|-1#-1|162601144735|35|GETMESSAGEFILE|chico`
    - This message was sent by the user “chico” requesting the messages file.

When a user performs a GET request to a VBB, the messages he requested, will be sent to him as regular text messages described before, however it will have an additional field preceding the sender id field. This additional field is comprised by two inner fields: the first one is the keyword “`vbb`”, and the second one will be the tag of the message, these fields are separated by the “#” character. In the case were the message is from another user, the tag and the sender ID will be the same. In the case were the message is from a shared group, the tag will be the name of the group and the sender ID will be name of the user that posted that message.

Ex: `TXT|-1#-1|162601144735|55|vbb#ua|toni|chico|12`

In this case, the user “chico” received a message posted by the user “toni” in the group “ua”.

Ex: `TXT|-1#-1|162601144735|56|vbb#toni|toni|chico|12`

In this case, the user “chico” received a message posted by the user “toni” whose only destination was the user “chico”.

8.1.5 Session Control Messages

These messages are sent from the smartphone application to the unit it is connected to. As these messages are not sent through the VANET and only have relevance in the unit itself, they do not incorporate a validity field, or any other unnecessary information. These are quite simple messages containing only 3 fields:

1. **Message type**: In this case will be the string “`Session`”. This field is used define the type of message;
2. **The command**: Can be either “`Subscribe`” or “`Draft`”. `Subscribe` will be used to request that a session is established with the unit, and `Draft` will be used to terminate a session.
3. **Sender username**: The username of the user performing the request.

Ex: `Session|Subscribe|chico`

The user “chico” is requesting for a session to be established.

Ex: `Session|Draft|chico`

The user “chico” is requesting the termination of its session.

8.1.6 Beacon Messages

Beacon messages are periodically sent by units, to announce their presence to any other nearby units. These messages sent through the VANET, however they are not forwarded by any other units, meaning they do not incorporate the validity field. They are comprised of 3 fields:

1. **Message type**: In this case will be the string "`BEACON`". This field is used define the type of message;
2. **Unit ID**: an integer unequivocally identifying the unit that sent the message.
3. **GPS Coordinates**: A string containing the GPS coordinates from where the beacon was sent. This field is used only for logging purposes and currently as no use in the beacon system itself.

Ex: `BEACON|12|(40.633066, -8.659825)`

A beacon sent by unit 12 in the given coordinates.

8.2 VNMS dependencies

The unit's module created by VNMS is based on REINVENT which was created in its original form using Python [24]. Additionally, REINVENT uses RabbitMQ as a message broker. For the unit's module to be deployed and executed on a unit, this unit needs to have Python and RabbitMQ installed. The RabbitMQ server needs to be executed before the unit's application in order for it to be able to create its communication channels.

RabbitMQ is supported in Python by the use of Pika which is a pure-Python implementation of the AMQP 0-9-1 protocol that tries to stay fairly independent of the underlying network support library [25]. Currently, Pika supports Python 3.5.x versions, however when the development of VNMS started it did not. To ensure that the system works properly, any Python 2.7.x version must be installed as well as the Pika library. The system was not tested with Python 3.5.x so correct behaviour is not guaranteed if using it.

Some additional libraries were also used and need to be installed for the unit's module to execute. These libraries are:

- **CherryPy** – A minimalist python web framework [26]. Used to create the web servers and deploy the web services on them.
- **PyDbLite** – A fast, pure-Python, untyped, in-memory database engine, using Python syntax to manage data [27]. Used to implement the databases on the system.
- **Requests** – allows to send HTTP/1.1 requests, without the need to manually add query strings to URLs, or to form-encode POST data [28]. Used to create the requests to be made on the web servers.

The smartphone application was created using Android Studio [29] and requires two additional libraries for it to compile which are:

- **RabbitMQ Java Client** – to communicate with the RabbitMQ servers deployed on the units.
- **prettytime** – an open source time formatting library that creates human readable, relative timestamps [30]. Used to format the messages timestamps.

8.3 Unit configuration

VNMS's unit module contains several settings that need to be configured and defined for it to operate. To easily perform this, a configuration file system was implemented as well as a Python module with functions to read the values from this file. These values can be obtained in the application by use of the module `config.py` which defines functions to get them. A sample file would be:

```
board_id:12
n_recent_messages:50
client_timeout:4
client_probe_period:5
messages_json_path:/root/Gateway/vbb_messages.json
rsu_reply_max_hops:-1
rsu_reply_max_time:-1
beacon_period:5
retransmit_period:100
```

The fields contained in this file are:

- **`board_id`** – number unequivocally identifying a unit.
- **`n_recent_messages`** – maximum number of messages that the circular array of the recent messages database will store.
- **`client_timeout`** – number of seconds a unit waits for the reply of a keep alive message sent to a user before considering the user as disconnected.
- **`client_probe_peridod`** – number of seconds between the keep alive queries that a unit makes to the smartphone application.
- **`messages_json_path`** – absolute path to where the JSON file of the VBB predefined messages files is located.
- **`rsu_reply_max_hops`** – number of max hops a RSU will add to a modified text message from the VBB to be sent to a user.
- **`RSU_reply_peridod`** – time expiration date, in minutes, that a RSU will add to a modified text message from the VBB to be sent to a user.

- **`beacon_period`** – number of seconds between “hello” messages sent by the beacon module.
- **`retransmit_peridod`** – time in seconds that must elapse between beacon messages for a unit to retransmit the pending messages upon receiving a beacon message from the same other unit.