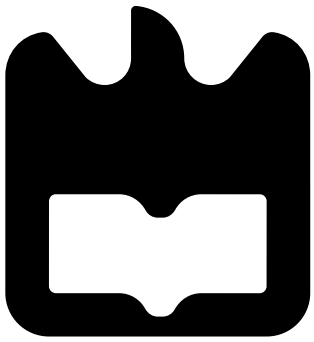
Universidade de Aveiro
Departamento de
Electrónica, Telecomunicações e Informática
2016

Pedro Manuel
Reverendo Cirne

Gestão segura de rotas numa VANET

Secure management of routes in a VANET



Pedro Manuel
Reverendo Cirne

Gestão segura de rotas numa VANET

Secure management of routes in a VANET

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 Doutor André Zúquete, Professor Auxiliar, e coorientação da Doutora Susana Sargento, Professora Associada com Agregação, ambos do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president	**Professor Doutor Rui Luís Andrade Aguiar** Professor Catedrático da Universidade de Aveiro
vogais / examiners committee	**Professor Doutor Pedro Miguel Alves Brandão** Professor Auxiliar da Universidade do Porto - Faculdade de Clências
	Professor Doutor André Ventura da Cruz Marnoto Zúquete Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática de Aveiro (Orientador)

agradecimentos A todos os que me deram amor, tempo, coragem, exemplos, reconhecimento, apoio... Sozinho estaria muito longe e bem diferente!

Resumo

As redes veiculares (VANETs) são um caso específico de redes *ad hoc* onde os nós são veículos. VANETs têm vindo a surgir nos últimos anos e é expectável que venham a desempenhar um papel importante no futuro para um grande número de aplicações. O roteamento é essencial para qualquer rede *ad hoc*, consequentemente, as estratégias de segurança para proteger o roteamento das VANETs devem ser consideradas essenciais. Nesta tese apresentamos: (1) TROPHY (Trustworthy VANET ROuting with grouP autHentication keYs), um conjunto de protocolos para autenticar mensagens de roteamento numa VANET, capaz de proteger as informações de roteamento distribuídas em condições de tempo altamente restritas; (2) `loop` (loop over orderly phases), um simulador interativo para testar e validar TROPHY juntamente com um protótipo de um KDC (Key Distribution Center). Os nós autorizados recebem recursivamente novas mensagens que lhes permitem atualizar o seu material criptográfico e manter as chaves de autenticação atualizadas na rede. Essas mensagens são construídas da forma a que qualquer nó que seja identificado como perdido ou fisicamente comprometido não seja capaz de executar a atualização, ficando assim excluído do processo de roteamento. Devido ao uso do KDC, uma entidade central, onde todo o material criptográfico é armazenado, incluímos um mecanismo para recuperar de qualquer acesso físico não autorizado e divulgação de todo esse material de uma só vez, sem exigir a intervenção humana na configuração dos dispositivos.

Abstract

Vehicular ad hoc networks (VANETs) are a specific case of *ad hoc* networks where nodes are vehicles. VANETs have being emerging in the last few years and are likely to play a major role in the future for a wide number of applications. Routing is essential for any *ad hoc* network, thus security strategies for protecting VANETs' routing must be considered essential. In this thesis we present: (1) TROPHY (Trustworthy VANET ROuting with grouP autHentication keYs), a set of protocols to authenticate routing messages in a VANET, under highly restrictive time conditions, capable of protecting the distributed routing information; (2) `loop` (loop over orderly phases), an interactive simulator for testing and validating TROPHY along with a prototype of KDC (Key Distribution Center). Authorized nodes recursively receive new messages that allow them to refresh their cryptographic material and keep the authentication keys updated across the network. These messages are built in a way that any node pinpointed as lost or physically compromised will not be able to perform the refreshment using them, and so, is excluded from the routing process. Due to the use of a KDC, a central entity, where all the cryptographic material is stored, we included a mechanism to recover from any unauthorised physical access and disclosure of all that material at once, without requiring the need of human intervention on devices' re-setup.

Contents

List of Figures

List of Tables

Acronyms

AP	Access Point
CRL	Certificate Revocation List
DoS	Denial of Service
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
GPS	Global Positioning System
IP	Internet Protocol
KDC	Key Distribution Center
LLC	Logical Link Control
MAC	Message Authentication Code
MANET	Mobile *Ad Hoc* Network
OBU	On Board Unit
OSI	Open Systems Interconnection
PSID	Provider Service Identifier
RSSI	Received Signal Strength Indication
RSU	Road Side Unit
SB2RP	Service-Based Layer-2 Routing Protocol

TCP	Transmission Control Protocol
UDP	User Datagram Protocol
V2D	Vehicle to Device
V2I	Vehicle to Infrastructure
V2V	Vehicle to Vehicle
VANET	Vehicular *Ad Hoc* Network
VU	OBU and RSU
WAVE	Wireless Access in Vehicular Environment
WSM	WAVE Short Message
WSMP	WAVE Short Message Protocol
WSN	Wireless Sensor Network
XOR	bitwise exclusive-or

Chapter 1

Introduction

For the things we have to learn before we can do them, we learn by doing them.

— Aristotle

Vehicular *Ad Hoc* Networks (VANETs) are a class of *ad hoc* networks that inherit most of the characteristics of Mobile *Ad Hoc* Networks (MANETs), but still, with enough differences to define a new category by itself [18, 41].

VANETs may have hundreds or thousands of nodes equipped with wireless communication capabilities. Most of the nodes are devices carried by vehicles, On Board Units (OBUs), and a relatively small part of the nodes are devices statically placed near roads, with added equipment for wired communication capabilities, Road Side Units (RSUs).

By their very singular nature and requirements, when compared with MANETs [6] [5], academic research community have given an increasingly amount of attention to the topic. Those characteristics have also lead to the creation of VANETs' specific solutions, such as the Wireless Access in Vehicular Environment (WAVE) architecture [1].

VANETs have being emerging in the last few years and are likely to play a major role in the future for a wide number of applications, ranging from hard real-time to delay tolerant applications [18].

Routing is essential for any *ad hoc* network, thus security strategies for protecting VANETs' routing must be considered essential. However, in the VANETs' routing context, it is not possible to apply classical techniques, such handshake-based authentication protocols [18], and so, some common security problems, such as availability, secrecy and integrity, do not have a widely accepted approach for a solution.

The widely accepted WAVE architecture defines a set of complementary and interdependent protocols to operate in a VANET environment [1, 29]. The IEEE 1609.2 [3] standard defines methods and formats of secure messages. For routing services, the IEEE 1609.2 standard may lead to service deficiencies [49], mainly due to message validation delays caused by asymmetric cryptography; therefore, we considered that such standard does not fit the requirements to authenticate routing messages in a VANET.

1.1 Objectives

The main objective of this work was to extend an existent VANET routing protocol where security aspects were not considered, Service-Based Layer-2 Routing Protocol (SB2RP), protecting the routing strategy from external attackers with the aim to modify or destroy it. During the development of this work, the routing protocol was being used on a real VANET, in Porto, with more than 600 OBU and RSUs (VUs). Since it was developed to operate under strict limitations related with time, it was our primary goal to avoid time consuming operations and delayed decisions.

The devices in operation are relatively limited, when compared with an average personal computer. Such devices do not have specialized co-processors for cryptographic operations, and the impact of asymmetric cryptographic operations is not negligible. It was part of our objective to explore the use of symmetric cryptographic primitives for the common and critical operations, avoiding, if possible, the use of asymmetric ones.

Although the connectivity of a VANET, specially OBUs' connectivity, can be complemented with other technologies, such as cellular networks, we focus on making our solution only dependent of the *ad hoc* network.

We assumed that there are 2 types of candidates for participating in the routing process, authorized and non-authorized ones. Within the non-authorized we have all the external devices, as well any device previously authorized but after some date pinpointed as lost or physically compromised. Non-authorized devices shall not interfere on the routing process. Authorized are all these that were initially prepared to operate in the VANET. It was assumed that all the authorized devices work properly. Furthermore, we did not pretend to detect any faulty behaviour on authorized devices. Moreover we included in our list of goals a mechanism to exclude any authorized device once pinpointed as lost, physically compromised, or otherwise improper for operation by any other reason.

We seek to authenticate all routing messages and defeat any adversary with the aim to

modify or destroy the routing process and full capabilities to eavesdrop, store, modify and transmit an arbitrary number of valid messages. Moreover, we seek to provide mechanisms to recover from attackers with the power to get physically access to one or more devices and all the cryptographic material stored on it, as well as mechanisms to recover from the disclosure of all cryptographic material stored on a central server.

It was out of the scope of this work to study alternative routing algorithms or mechanisms to detect faulty behaviors on authorized devices, as well all the topics related with the secrecy of the messages and attacks on the physical layer (e.g. jamming).

It was also out of the scope of this work to study updates of the long-term public keys used to authenticate messages sent from a central server to the VUs.

1.2 Contributions

The main contribution of this work is the specification and validation of TROPHY (Trustworthy VANET ROuting with grouP autHentication keYs), a set of protocols to authenticate routing messages in a VANET, under highly restrictive time conditions, capable of protecting the distributed routing information. Furthermore, the solution is able to recover, with minimal human interaction, from attacks to the physical infrastructure of the central key server that may lead to the disclosure of all the cryptographic material in use.

The demand for such a novel solution was originated due to a real scenario, present in a VANET in Porto city, operated by Veniam. The VANET has currently more than 600 devices, with well defined characteristics, and its routing protocol was built to be fast and reliable in this very specific scenario. We present a solution that honors the real and well-defined requirements of the VANET, and does not destroy the original routing strategy by excessively consuming time or resources.

As part of our solution for routing authentication, we included a mechanism to exclude any node at any time. This is specially important due to the non-controlled environment where the devices are operating, and the real possibility of any of the nodes to get stolen, lost, damaged or becoming suspicious. Due to the use of a central entity, where all the cryptographic material is stored, we also included a mechanism to recover from any unauthorised physical access and disclosure of all that material at once, without requiring the need of human intervention on devices' re-setup.

A secondary contribution is the development of `loop` (loop over orderly phases), an

interactive simulator for testing and validating TROPHY along with a prototype of Key Distribution Center (KDC). The simulator, interacting with the KDC, makes use of real data collected from the VANET to validate the set of protocols under a realistic scenario of high mobility. With the simulator and that real data it was possible to measure the impact of different options and strategies in a controlled environment, before the field implementation.

1.3 Document Structure

The rest of the document will unfold as follows:

Chapter 2 contains information to give to the reader the context needed to proceed along the document.

Chapter 3 has a description of different works related with this thesis.

Chapter 4 describes the architecture of our solution.

Chapter 5 presents the implementation details.

Chapter 6 aggregates the results of the performed tests.

Chapter 7 summarizes our work.

Chapter 2

Context

...packets will arrive before they are sent.

— Robert M. Hinden, *RFC 6921*

From the vast range of topics related with secure management of routes we focused on the authentication of the routing messages exchanged among a group of users. Moreover, we focused on the authentication of messages of an existent and validated routing protocol.

Generically, the authentication of messages can be obtained based on the inclusion of auxiliary information computed with two different cryptographic primitives: symmetric or asymmetric ones. When using symmetric primitives the auxiliary information is known as Message Authentication Code (MAC) while in case of asymmetric ones is known as digital signature.

The authentication methods inherit from the used primitives some important characteristics, namely: the demand of computational power and the use of a single key or a pair of distinct keys to create and validate it.

2.1 VANET

VANETs are a class of *ad hoc* networks made out of vehicles equipped with devices with wireless communication capabilities, OBUs. Besides OBUs, VANETs also have a relatively small part of stationary nodes with extra wired connectivity, RSUs. Due to the mobile nature of vehicles that carry the OBUs, VANETs are considered a subclass of MANETs, with some distinct characteristics, namely: high node speed, highly volatile topology, high fragmentation probability and potentially large scale.

Our work is based on Veniam's VANET and its characteristics, currently deployed in Porto city, Portugal. In our scenario, both OBUs and RSUs are devices with the same hardware and exactly the same computational power. Alternative scenarios, with computationally powerful RSUs, were not considered.

Besides the *ad hoc* network, the devices (VUs) in a VANET may have access to other types of wireless communication infrastructures, such as cellular networks and WiFi networks. In our work, we did not make use of wireless communication capabilities other than the ones provided by the *ad hoc* network. Nevertheless, the VANET is not isolated from the outside world, due to the existence of the wired connections in the RSUs. The RSUs can directly use any service placed in the internet and the OBUs may have indirect access to the same services based on the *ad hoc* infrastructure.

The decision of avoiding the use of cellular networks was an additional challenge. The reason was exclusively the operational costs associated to a solution that, for hundreds, possibly thousands of devices, would rely on cellular networks to exchange information, even if sporadically.

Regarding the use of WiFi networks, such as 802.11g, it was not considered due to the current absence of configured Access Points (APs) available to serve OBUs.

2.2 OBUs and RSUs (VUs)

The VUs in operation are embedded linux systems with limited resources, when compared with an average computer. The VUs have an `ARMv7` single-core processor at 800MHz and 256Mb of memory. Those characteristics, specially the ones related with the processor, are important for the cryptographic operations.

To evaluate the performance of the VUs, regarding different cryptographic algorithms, we used the facilities provided by the `OpenSSL` project[1]. Since the `OpenSSL` components were available in VUs, we used `openssl speed`[2], a tool specially designed to test the performance of different cryptographic algorithms on the hosts. The tests were done under low load and the representative results are shown in Table 2.1.

[1] https://www.openssl.org/
[2] https://wiki.openssl.org/index.php/Manual:Speed(1)

Algorithm	Operation	Time
RSA 2048	Sign	53.3ms
	Verify	15.1ms
ECDSA 224	Sign	1.4ms
	Verify	5.7ms
DSA 2048	Sign	15.0ms
	Verify	18.1ms
AES-128-CBC 1024Bytes	Encrypt	54.2μs
HMAC(MD5) 1024Bytes	Digest	9.9μs

Table 2.1: Performance of cryptographic algorithms in VUs

2.3 Wireless Access in Vehicular Environment (WAVE)

The WAVE architecure has as goal to support low-latency communications among vehicles, Vehicle to Vehicle (V2V), between vehicles and the stationary infrastructure, Vehicle to Infrastructure (V2I), and between vehicles and hand-held devices, Vehicle to Device (V2D) [1].

2.3.1 WAVE standards and protocols

The WAVE architecure is built based on a stack of standards. A device is considered a WAVE device if it is at least conformant with the core WAVE architecure standards, namely: IEEE 802.11p, IEEE 1609.4 and IEEE 1609.3 (shown together with IEEE 1609.2, in Figure 2.1). Although the WAVE architecure defines more standards, all of them are out of the context of this work, so, they will not be considered. Each standard defines the interaction between multiple protocols which individually perform their operations.

IEEE 802.11p: Specifies the medium access control related with the wireless connectivity among vehicles, infrastructure and devices. It is an amendment to IEEE 802.11 with extensions to add support for vehicular environment.

IEEE 1609.4: Specifies extensions to the IEEE 802.11p, useful to the vehicular environment but not included on IEEE 802.11, such as physical channel coordination and MAC-layer readdressing for supporting pseudonyms.

IEEE 1609.3: Specifies networking and transport services essential for the operation of a WAVE system.

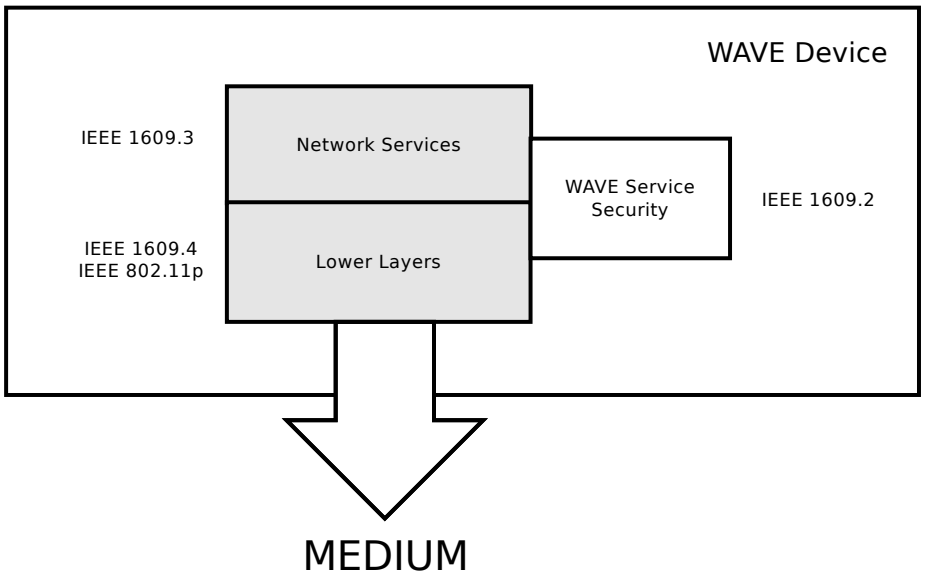


Figure 2.1: Stack of WAVE standards

IEEE 1609.2: Specifies transversal security mechanisms accessible to the other WAVE standards.

The aggregation of the different protocols, associated to the correspondent standard is shown in Figure 2.2. The IEEE 1609.3 includes the well known Logical Link Control (LLC), Internet Protocol (IP), User Datagram Protocol (UDP), and Transmission Control Protocol (TCP) protocols. Side by side with the TCP/IP and TCP/UDP stacks, sharing the lower layer, is placed the specific WAVE Short Message Protocol (WSMP).

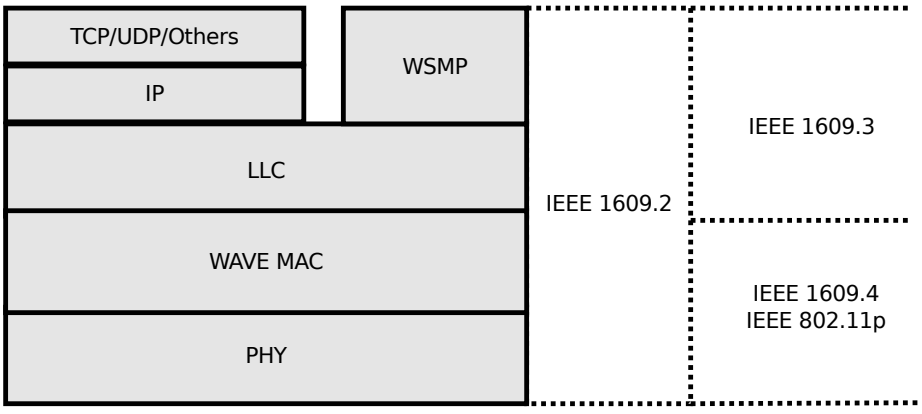


Figure 2.2: Protocol stack of the WAVE and inter-relation among the standards

The WSMP is an optimized protocol to send messages, WAVE Short Messages (WSMs), in a wireless vehicular environment. This optimized protocol is interchangeable and dis-

tinguishable from the IP-based on the LLC protocol. The WSMs have a variable length, dependent on the physical layer [2]. Using the IEEE 802.11p the length value is defined between 1 and 2302 bytes, with a default of 1400 bytes, shared by the variable length WSMP header and the WSM data.

According to the WAVE architecure, the WSM data can be sent as plain data without taking in consideration security aspects, or can be sent encrypted or signed based on security services defined on IEEE 1609.2 standard [3]. The IEEE 1609.2 standard relies on asymmetric algorithms based on elliptic curves, such as Elliptic Curve Digital Signature Algorithm (ECDSA) [20] for signing and Elliptic Curve Integrated Encryption Scheme (ECIES) [28] for encrypting. Along with the algorithms are defined all the auxiliary mechanisms needed for the distribution and management of the keys, the certificates and the Certificate Revocation Lists (CRLs).

When compared with the Open Systems Interconnection (OSI) model, the topmost 3 layers have no correspondents on the WAVE architecure while the lowest 4 layers have following correspondents:

Physical Layer Directly maps to the WAVE physical layer;

Data Link Layer Includes both LLC and WAVE medium access control;

Transport and Network Layer Merged by the WSMP.

2.3.2 WAVE application-services

WSMs are composed by WAVE application-services, hereafter known as WAVE applications, contextualized with the wireless vehicular environment. The WAVE applications are allowed by the WSMP to directly control physical parameters such as selecting the channel and the transmitter power.

The WAVE architecure defines a WAVE application independently and completely decoupled from the application layer of the OSI model as well as from the concept of application associated to a computer program. According to the IEEE Guide for WAVE [1]: *"An application-service is a service involving an exchange of data, generally provided by a higher layer entity (e.g., an application) on one WAVE device to a similar entity on another WAVE device, using WAVE communications."*.

The WAVE applications, along with the data messages to be sent, need to provide a Provider Service Identifier (PSID) associated to the message and the address of the

destination. The destination is the MAC (Medium Access Control) address of a neighboring device or the broadcast address (all bits at 1).

On the reception of both unicast and broadcast WSMs, the WSMP delivers them to the applications that registered interest on such message, based on the PSID. If the PSID associated to the received WSM is not claimed by any WAVE application, the message can be discarded.

2.4 Service-Based Layer-2 Routing Protocol (SB2RP)

From the point of view of an *ad hoc* network, the routing strategy in VANETs has to deal with some new challenges, even when compared with MANETs [56, 25]. The routing strategy in use was developed to face all the new challenges found during the deployment. The routing algorithm is used with the aim to connect the OBUs to the RSUs and vice-versa (RSUs have IP-based communication with an infrastructure).

SB2RP [12] is a WAVE application associated to the OSI data link layer (layer 2) that implements a periodic (distance-vector) routing algorithm to route between the infrastructure and the VANET (communication to and from the infrastructure). Besides the operation on the OSI layer 2, SB2RP has direct access to the routing tables of IP, associated to the OSI Layer 3. The routing service is identified by a PSID and the messages are broadcast according to the WSMP in periods of 100 milliseconds. The relevant information for this work exchanged by each VU is: the Global Positioning System (GPS) information, the local time, and the routing information.

Each VU stores the information received on the SB2RP messages for each of the neighbors in a auxiliary table. Each time a new SB2RP message is received any older information related with the same origin is replaced. All the entries on the auxiliary table are removed if older than 1.5 seconds.

The IP routing tables are updated, conditionally to the neighbors in a range of 3 hops, in periods of 1 second, based on the information stored on the auxiliary table. This is possible due to a direct relation between the identification of the VU and the IP address of that VU.

SB2RP does not have into consideration any security aspects. After the validation of the routing messages, SB2RP may use the authenticated information to filter them based on the contained time, GPS position or any other parameter, and update the routing tables.

# Ops	% CPU
17	100
8	50
4	25

Table 2.2: Number of ECDSA verifications per 100 milliseconds and CPU usage

The algorithms proposed on the IEEE 1609.2 standard are not compatible with the operation mode of SB2RP. This incompatibility can be deduced by taking as reference the time needed for the relevant cryptographic operations in VUs (Section 2.2) and the 100 milliseconds period to exchange routing information. Due to the time related constraints, this incompatibility is extended to all the strategies based on similar asymmetric cryptographic primitives.

The ECDSA is the algorithm with faster verifications and takes 5.7 milliseconds for each one. During all the time taken by verification of a signature, the CPU is near if not at 100%. In these conditions, the number of verifications per 100 milliseconds, is shown in Table 2.2.

The use of 100% of the CPU, even 50%, to validate routing messages is not acceptable, and still it would just be possible to validate 17 or 8 messages per 100 milliseconds. If the use of 25% of the CPU is acceptable or not is irrelevant, since 4 validations per 100 milliseconds are not enough to have a functional routing algorithm. Moreover, due to the difference of time that an ECDSA takes to be validated and the generation of a bogus message, the SB2RP would be highly susceptible to Denial of Service (DoS) attacks.

Chapter 3

Related Work

Curiouser and curiouser!

— Alice

During the development of this thesis we looked for other works that faced similar problems. Since we are on a very specific context, VANETs, this content was the starting point. Based on some similar characteristics that VANETs have with MANETs and Wireless Sensor Networks (WSNs), we also looked for ideas on these fields.

We end up investigating solutions, from the late 90s, initially proposed for a quite distinct problem, multicast over wired connections. Although in a different context, they have in common some similarities regarding the management of cryptographic material in a large group of users.

We analysed all the different contexts where each of the proposed solutions were applied, and identified the ideas possible to be considered and the ones that according to our scenario, expectations and goals would have no benefit.

3.1 Security on VANETs

The WAVE architecture [1] allows the transmission of secure messages based on the 1609.2 standard [3], which for signing relies on asymmetric signatures according to the ECDSA. Under the highly restricted time conditions of communications and of the available hardware, the time needed to produce and validate the asymmetric signatures is relatively high when compared with the period for the transmission of beacons (see Chapter 2). We did not follow the 1609.2 standard since the use of ECDSA to sign each of the

routing message would be a serious limitation to the number of processed messages: the signature of a percentage of them would end up on a non-deterministic behavior and the production of one signature for groups of more than one routing messages would create critical time and contextual dependencies between the mobile nodes. Nevertheless, we use the ECDSA to authenticate, in the VUs, some messages produced by a central server. In this case, the time needed to produce the signature is not critical, since it is done sporadically by a distinct and powerful machine, neither it is the time needed to validate the signature in VUs, due to the sparse periods of time this occurs.

With focus on broadcast authentication, Perrig et al. [37] [36] proposed a solution where asymmetry is obtained based on time synchronization and successive disclosure of keys in well determined periods of time. In short, the sender first sends a message with a MAC, and later discloses the key and starts using a new one. With the message, the MAC and later the key, any receiver can validate the MAC. Assuming time synchronization between the participants, with the use of one-way functions, it is possible to ensure that a disclosed key is not a valid key to create new MACs. Some variations were created for more specific scenarios, such as WSNs [39] and VANETs [49], but none in the routing context. The variation proposed by Studer et al. [49] has in common with our solution the goal to avoid the digital asymmetric signatures associated to the WAVE architecture due to time related constrains. However, this solution needs a large number of keys being stored in each device, the same number of keys as the number of devices in the VANET. Moreover, due to the use of a pre-computed chain of values resultant from a one-way function, all the devices need to be loaded periodically with a new chain and a public starting point of the chain of all the other devices. In our case, we have a small number of keys stored in each device, and there is no need for periodic, global and synchronized re-setups in all the devices.

Ahmed et al. [4] and Wagan et al. [51] presented two distinct solutions based on the geographical division of the region accessible by the OBUs into smaller areas. The division can be recursive, so, the smaller areas can be, by their own, divided into even smaller areas. Both proposals, also due to time related constrains, have as goal to avoid the digital asymmetric signatures associated to the WAVE architecture. In each area there is one or more RSUs or a group leader elected among OBUs, responsible for the registration and the delivery of the correspondent keys and certificates to the new OBUs entering the area. Both solutions make use of symmetric cryptography to speed up the message authentication for high priority emergency messages [4] or for communications inside a

small area [51]. Besides the use of symmetric cryptography, in an abstract way, there is another common idea between these two proposed architectures and our own: the use of a hierarchy to facilitate the management of the VANET, regarding the distribution of new cryptographic material. However, instead of groups associated to a geographical area, we have groups based on a virtual geographical independent hierarchy where OBUs are free to be dispersed among the available region and use a common symmetric key for all the region. Instead of a dependence on the geographical area we have a dependence on time, so, the key associated to the authentication of routing messages is different from time to time and not from local to local.

3.2 Group keys

A group key is a cryptographic key shared among a group of users. All users with access to the group key can use it to encrypt or authenticate (and perform the respectively reverse operation) all the transmitted messages among them via an insecure channel, following an one-to-many or a many-to-many paradigm. The topic of group keys started to receive attention from network researchers in the late 1990s, due to the envision of future needs related with multicast services based on group communications model [19, 48, 52, 54], e.g. teleconference and information services.

According to the architecture in use, group key management is currently classified according to 3 main approaches [42, 55, 16, 43, 23, 46]:

Centralized: There is a single entity responsible for the management of the group.

Decentralized: There are multiple entities responsible for the management of subgroups of the main group.

Distributed or contributory: There is no central nor decentralized entity. The members of a group are responsible for the management. They can contribute to the computations or elect (be elected by) any of the peers to perform such task.

Considering our well defined scenario, the centralized approach is the one that best fits our requirements. We will focus our analysis on the different approaches on the centralized category.

Under a centralized architecture, the creation and revocation of keys are under a single entity, a KDC. A group key distribution protocol is used to secretly make available to all the members of a group the needed key, or keys.

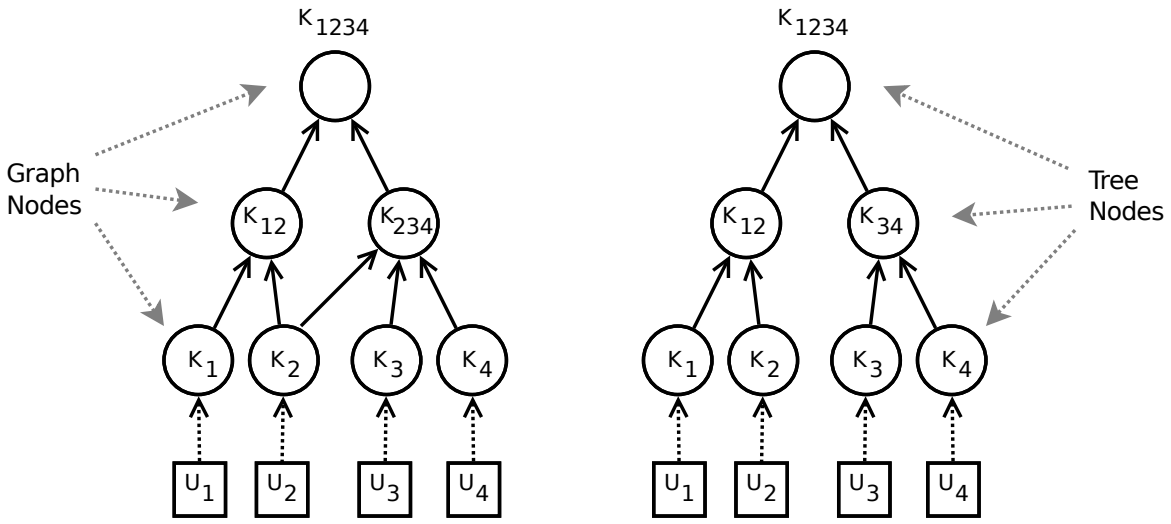


Figure 3.1: Graph based and tree based hierarchy

Relatively to the centralized category we do not focus on schemes based on pairwise keys, where the KDC and each of the V members of the group share an individual secret and the distribution of a new key relies on the V rekeying messages [19].

Excluding pairwise key schemes, all the rest of the centralized approaches fall into a hierarchical organization, based on graphs, and trees as a specific case (See Figure 3.1). On this type of organization, group members, at the lowest level, have a collection of keys: all the keys from the path from the lowest node until the root node. In Figure 3.1 the group key is k_{1234}. The other keys are used to distribute the new keys. For example, if ciphered with k_{1234}, a new key is available for all the members, but when ciphered with k_{12} it is just available to u_1 and u_2.

Graphs in which two vertices are connected by more than one path are rarely used in literature, and when they are, the authors end up removing some paths, ending up with best results based on the resultant trees [54].

We started with a tree hierarchy and, at an early stage of our work, we explored non-tree graphs and concluded that, for our requirements, tree-based hierarchies have also clearly advantages to graphs in terms of management due to the homogeneous and simplified structure.

Tree-based hierarchies were independently proposed by Wong et al. [54] and Wallner et al. [52]. Before these proposals, according to Wong et al. [54], *"no one has addressed the need for frequent key changes and the associated scalability problem for a very large group"* with a globally shared group key. These initial works have been proposed with wired

connections, multicast and a one-to-many communication paradigm in mind. This is a crucial difference, since the characteristics of *ad hoc* networks, VANETs in special, create additional challenges [16, 13]. Moreover, since we are using the group key to authenticate messages between members and not to guarantee the secrecy of any data, we can relax our needs on the so-called backward secrecy [44, 24].

Wong et al. [54] formalizes 3 types of strategies to update the keys on members:

User-Oriented: Where each member receives a specific single message with the new keys, ciphered with a key held by the members.

Key-Oriented: where each of the new keys are ciphered individually, and so, members may need more than one message.

Group-Oriented: where all the messages are concatenated in a single message, sent to all members, that will later select the needed keys.

Key-Oriented and Group-Oriented imply that all members need to receive more than one message or a relatively big message, respectively, thus they are not suitable for our scenario. We followed a hybrid model, using an User-Oriented strategy in a minor number of cases, and an approach that takes the best of the User-Oriented and the Key-Oriented strategies. The majority of the key updates are performed by building a relatively small number of ciphered messages, where all members will find one that can deciphered, and use the content to update (refresh) all keys. An update method, based on a refreshment, called synchro-difference, was firstly proposed by Zhu [57], where instead of distributing all new keys, a single key update is distributed and used by the members to produce all the new keys. The distributed key update contains a common difference between the old key and the new one to be produced.

Taking the tree in the right hand size of Figure 3.1 as example, if u_4 gets compromised, in a User-Oriented strategy, u_1 and u_2 would receive the same message with the new k_{1234} ciphered with the k_{12}, and u_3 receives a distinct message with both k_{1234} and k_{34} ciphered with k_3. In a Key-Oriented strategy, u_1 and u_2 would receive the same message, but u_3 needs 2 messages: one with the new k_{34} ciphered with k_3 and another with the new k_{1234} ciphered with the new k_{34}. With a synchro-difference, u_1 and u_2 would receive some random value ciphered with k_{12}, and u_3 would receive the same random value ciphered with k_3. Each of the new keys is produced based on a bitwise exclusive-or (XOR) of the old key and the received random value.

Canetti et al. [11] presented a different way to reduce the overhead of messages in the network. Instead of a key update used to generate all new keys, an one-way function is used, $H(x)$, to give to the different members the results of different number of chained iterations over the one-way function. Each of the members can then compute all the keys and only the keys it is entitled to have. Using the tree on Figure 3.1 as example, to exclude u_1 it is created a random value for k_1 (that can be used to re-include u_1), u_2 receives $k_{12} = H(k_1)$ ciphered with k_2 and both u_3 and u_4 receive $k_{1234} = H(H(k_1))$ ciphered with k_{34}. The authors give only as example the exclusion of only one member, although it is possible to generalize the strategy and conclude that the exclusion of multiple members at once can only be endured by the successive exclusion of one at a time. Based on a synchro-difference we can exclude multiple members at once.

Sherman et al. [45] extended the idea of Canetti et al. [11] from an one-way function to one-way trees, also know as Merkle trees [30]. A solution based on one-way trees allows multiple exclusions at a time, but needs the management (and distribution) of an extra value associated to each of the nodes, a so-called blind key. The blind key is built upon the secret key of the sibling of each node. By sibling we mean the node that shares the same ascendant. This enables each node to calculate the common ascendant, without breaking the secrecy of its sibling's secret key.

Perrig et al. [38] presented a work focused on *"secure media broadcast over the internet"* where the KDC and all members have synchronized time and where both can, in some occasions, independently update the keys in well-defined periods of time. This can be seen as a virtual distribution protocol of the new keys, where no messages need to be sent between entities. In common with this idea we have the notion of periodic updates, but in our case, with explicitly distribution of new messages. The idea of synchronization with no messages was initially attractive and considered, but we concluded that an epidemic propagation of the key updates would be a better way to fit our requirements, since with it we are not dependent on scheduled events that may occur in a different context in the future.

Dini et al. [17] proposed the use of one-way functions to provide self-authenticated key-oriented updates for efficient key revocation, avoiding the use of digital signatures in update messages. A key k can be obtained by $k = H(k_{nxt})$, where $H(x)$ is an one-way function and k_{nxt} is the next key received on a ciphered message. With this approach each member can test the newly received key, that was delivery ciphered but without any type of signature. However, to the best of our knowledge, this cannot be applied along with

our method where we send a difference to all keys of the next generation. Thus, using Key-Oriented updates to benefit from this strategy was not considered worthy.

Perrig et al. [38], Ng et al. [31] and Park et al. [35] proposed different hierarchical schemes based on unbalanced and dynamic balanced trees, with the aim to constantly optimize the structure according to the volatile number of members. In contrast, we use a static and permanently balanced tree, prepared to accommodate a pre-defined large number of members. Instead of rearranging the tree structure each time new members join or leave the group we use a strategy, with negligible effect in both storage and processing needs, to identify the impact of these actions on the pre-accommodated nodes and considered only their existence on a needed basis. Thus, we end up with the same benefits as if the tree was being constantly re-balanced.

3.3 Simulation tools for VANETs

Under the VANETs context there are two distinct categories of simulators to be considered: microscopic traffic simulators and network simulators. The microscopic traffic simulators [26, 15, 34, 10, 14, 32, 21] generate traces of vehicles and possible other entities while the network simulators [22, 50, 9, 33] use those traces to simulate the communications between entities.

With a few exceptions [53, 27] that implement both types of simulators side by side, the two types are decoupled and independently developed from each other. Other simulators that integrate one network simulator and one microscopic traffic simulator [47, 40], provide a bridge between them and give some extra facilitates to deal with VANETs specific scenarios.

Since we had the possibility to use data-sets of real behavior (including the position and the listened neighbors with the correspondent signal's strength) of almost 500 devices, with 24 hours of information, all the tools for modelling artificial positions and states of mobile entities or communications among them were not considered interesting.

Since both real mobility and network characteristics are embedded in the data-sets, we implemented a simple simulator to take the best out of these 2 different sets of available information.

Chapter 4

Architecture

Give me six hours to chop down a tree and I will spend the first four sharpening the axe.

— Anonymous

Our architecture has 4 entities: a KDC, its human operator, the OBUs and the RSUs. We may indistinctly mention both OBUs and RSUs as a single entity, the VUs.

We designed our architecture to operate on a private VANET with a number of VUs ranging from a few to some thousands units.

Our solution is based on a central entity, a KDC, under the same control of the VANET, responsible for creating, updating, and periodically initiating the distribution of messages needed to maintain the routing authentication process valid across the network.

The KDC was prepared to interact with a human operator, to setup the network as well as to update some parameters based on decisions taken during its operation.

All the orders produced by the KDC are initially sent in messages to one or more RSUs and then disseminated by all of them to all the others, following a best effort, epidemic approach.

Symmetric cryptography is used to authenticate messages exchanged between VUs. Asymmetric cryptography is used to sign contents produced by the KDC.

Along with the authentication mechanism conceived for routing messages, we designed a strategy to exclude any node (or set of nodes) from the routing process at any time.

In the upcoming sections we describe in detail TROPHY (Trustworthy VANET ROuting with grouP autHentication keYs), a set of protocols based on group authentication keys capable of protecting the routing strategy of a VANET from external attackers with the

aim to modify or destroy it.

4.1 Interaction between entities

The human operator can at any time include or exclude new VUs, as well as to reconfigure some network-wide parameters on the KDC. After the initial setup of VUs by a human operator, the routing processed is initiated on those VUs.

The routing messages exchanged between VUs, hereafter referred as beacons, are authenticated with a MAC created with the **routing key**, ρ, a symmetric key shared by all VUs.

The routing key is periodically refreshed by the KDC, to anticipate its potential cryptanalysis, given the high number of MACs that may be created with the same key during the VANET operation. Furthermore, if one or more VUs get pinpointed as lost or physically compromised, the human operator can immediately launch a key refreshment to prevent the exploitation of a possible leaked routing key. The refreshment process is also used to exclude the compromised VUs thereafter from the VANET's routing (the VUs to exclude are not involved in the key refreshment process).

When we mention ρ, we mention the routing key without taking in consideration a specific refreshment associated to a time interval. The multiples refreshments of ρ create a sequence of values identifiable as $\rho(1), \rho(2), \rho(3), \ldots, \rho(t-1), \rho(t), \rho(t+1)$.

The time for the key updates in the VANET is controlled by the KDC, under the form of time intervals with subsequent indexes. The time intervals are associated to the real time of the KDC and each of the intervals may have a different duration. Each VU uses the most recent index to compare itself with the rest of the VANET, e.g. if a VU with $\rho(t)$ receives an update message associated to $t+2$, it can deduce that exists in the VANET one or more messages associated to $t+1$. If otherwise we had updates with absolute time-stamps, the VUs could not deduce if there would be any other intermediate updates between them unless there was a well defined period for the KDC to release new updates. However, strictly periodic updates would not allow arbitrary update decisions, and so, are not desirable. For this reason, even with the absolute time present in beacons and available for routing decisions based on minimum clock synchronization between VUs, the update of ρ in the VANET is not dependent on it.

The VUs may have different versions of ρ as their most recent routing key. Each VU will always use the most recent version of ρ to create and validate MACs. Beacons are only

used to update the routing information if they have a valid MAC computed with the most recent version of ρ present on the receiver. Beacons with valid MACs computed with older versions of ρ are only used to trigger the update process in already updated neighbors.

4.1.1 Epidemic propagation of key refreshments

The refreshment of ρ is performed based on a **refreshment update**, $r(t)$, sent by the KDC to the VANET. Such updates are signed by the KDC with its private, asymmetric key and all the VUs can validate the signature with the correspondent public key that was initially preloaded. The signed refreshment update, together with some auxiliary information, forms a **refreshment message**. The signature of refreshment messages by the KDC is the only part of the process requiring asymmetric cryptography.

Refreshment messages are associated to a specific time interval, t, and always have as first destination the RSUs. For the same time interval t, it may exist one or more distinct refreshment messages and each message carry this distinct number on it. RSUs periodically contact KDC, and receive the new messages, if any. From RSUs, the refreshment messages are redistributed to all OBUs within beacons.

After receiving a refreshment message, OBUs can redistribute it to all neighbor OBUs, in the exactly same way the RSUs initially did. This way, we have an epidemic propagation of the refreshment messages where VUs do not have to trust on each other to receive and validate contents created by KDC. Moreover, we have a process completely controlled by KDC (which decides when updates need to occur) whereas most of the work is delegated to VUs, following a best effort approach.

In Figure 4.1 we have a basic scenario to illustrate the epidemic propagation of the refreshment messages, associated to $r(1)$, and its use to update $\rho(0)$ to $\rho(1)$ in RSUs and OBUs.

The epidemic key refreshment is a chained process where VUs can only update its cryptographic material to the last state t, based on the previous state $t-1$.

The use of a chained process, instead of a direct replacement, has one main vantage: the same refreshment message can be used to refresh different keys at the same time on different VUs without breaking the secrecy of such keys. This is specially useful since the routing key ρ is not the only symmetric key used in the VANET.

In groups of VUs that were not isolated from the VANET, each VU has the required routing key to create MACs for its own beacons and to validate the ones received from neighbors. After an isolation period, a VU may be required to engage on a specific key

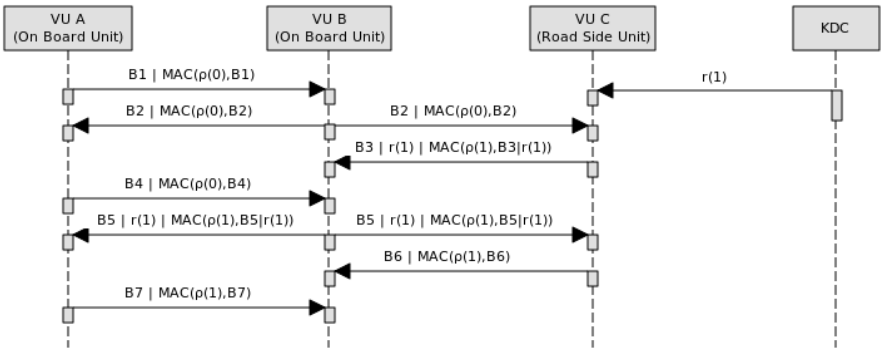


Figure 4.1: Epidemic propagation of the refreshment messages

update before being able to participate again on the routing process with its neighbors. Such key update is supported by updated neighboring VUs, based on an **historical cache** of the last refreshment messages stored on such neighbors. The **size of the cache**, h, is measured in time intervals and is controlled by the KDC. Any updated VU can contribute to the update of a peer, by sending cached information previously received.

If none of the neighbors has the needed refreshment messages on the historical cache, the isolated VU remains unable to participate on the routing process. This calls for an alternative method to deal with this extreme situation.

4.1.2 Historical cache

Since ρ has multiple versions, along with the beacon and the correspondent MAC, is sent the time interval, t, associated to ρ, for enabling the receiver to identify which version of ρ was used in the MAC.

The receiver of a beacon can get the difference between the time interval on the beacon and its own state (time interval of its last cryptographic material). When receiving a beacon from an outdated neighbor and the difference is in the range of the historical cache, the MAC is verified with the old version of ρ associated to the candidate historical cache entry (without this verification, the VUs could be fooled with bogus beacons). If the outdated beacon is valid according the outdated state of the neighbor, the VU will search in its historical cache for the needed refreshment message to update the neighbor. If the message is found, it will append such message to its next beacon. If a VU detects more than one outdated neighbor that need different refreshment messages, it will randomly select only one.

To update the entire VANET from $t-1$ to t, the KDC can create one or more messages. If there is only one refreshment message for a specific time interval t, any VU on state t can update any other from state $t-1$ to t, sending the message that was used to perform its own update from $t-1$ to t.

With more than one refreshment message, a VU on state t may not be able to update some neighbor on state $t-1$, unless it has all the messages created by the KDC for that update. Each refreshment message has the information of the number of distinct messages created for the same time interval. Due to the broadcast nature of communications, when refreshment messages are attached to beacons to update some specific neighbor, they will be received by all the neighbors of the transmitter. This way, any VU can verify if any of the refreshment messages are missing on its cache, and load them in such case. With more cached entries, VUs have a higher chance to contribute, in the future, to the global key update of the VANET.

To improve the propagation of all the distinct messages across VANET, each VU includes in its beacons the state, either complete or not, of its historical cache. If all the VUs are updated, but some have their historical cache incomplete, neighbors can detect the situation and append to its next beacon a refreshment message from its cache, selected randomly. Based on the incompleteness of the neighbors, VUs can append refreshment messages to a limited percentage of the transmitted beacons, instead of doing it for all of them. This percentage, the **basal refreshment rate**, brr, is defined by the KDC and is the same for all VUs. Along with a new refreshment, the KDC can redefine brr.

The refreshment messages are only included in the historical cache after the validation of the signature produced by the KDC.

4.1.2.1 Out of order reception of refreshment messages

As a complementary mechanism (useful only for OBUs) to the historical cache, we include an out-of-order buffer for the reception of the refreshments messages. This allows the OBUs to validate and save some refreshment messages, that are not useful at the moment but are expected to be in a near future. For instance, an OBU in state $t-2$ may first receive a refreshment message to update from $t-1$ to t before the message to update from $t-2$ to $t-1$.

Based on the updates that occur in the cryptographic material of each OBU, the messages in the out-of-order buffer are directly sent to the historical cache. The messages are not distributed back to VANET directly from the out-of-order buffer because OBUs can

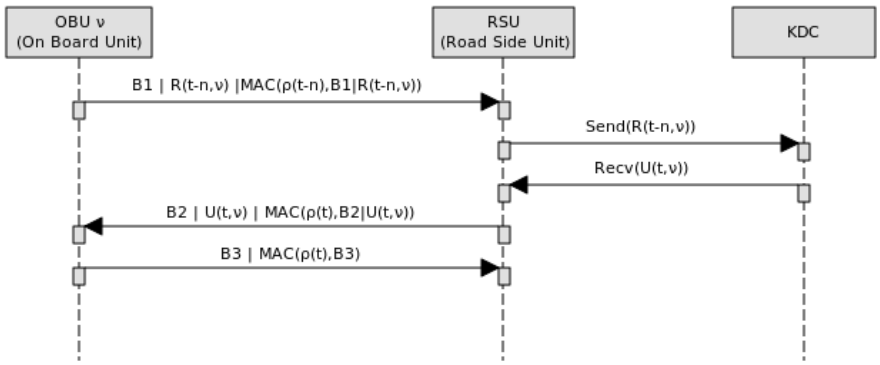


Figure 4.2: Recovering from isolation period

not validate the MAC on the messages coming from neighbors in some future state (the ones that could use such messages).

4.1.3 Fallback after a period of isolation

If one VU gets isolated or powered off, it may loose one or more refreshment messages. Furthermore, the VU can be explicitly isolated by the KDC. Mainly because of the first case, a fallback method is necessary, since it is desirable that the VU reengages on the routing process. On the second case, the isolation must be honored, and the fallback must only work after some explicit decision that cancels the previous state of isolation.

The RSUs can directly exchange messages with the KDC at any time, so, if allowed, an RSU can easily recover the last state of the cryptographic material of the VANET. The OBUs, on the other hand, do not have such possibility.

If an OBU cannot be updated by the content on the historical cache of its neighbors, it can use an RSU as proxy to request a specific message to synchronize its subset P with the values present in KDC. The requests, **sync message requests**, contain the last time interval known by the OBU, and 2 MACs, one created with ρ and the another one created with a key, ϵ, shared only by the KDC and the OBU. The first MAC can be used to filter requests on RSUs. The second is used to authenticate the OBU on the KDC. The replies, **sync message replies**, contain all the keys for the VU, encrypted with a ϵ and signed by the KDC.

In Figure 4.2 we illustrate this protocol, where neither the OBU nor the KDC have to trust on the intermediate RSU. After a period of isolation, and upon detecting an RSU, an OBU ν appends to its beacon a sync request, R_ν, that will be sent to the KDC by an

RSU. After receiving a sync reply, $U_\nu(t)$, the RSU will append it to its next beacon. The sync reply will be available to all the neighbors of the RSU, but can only be decrypted by the OBU ν. All the neighbors of the RSU can resend $U_\nu(t)$, appending it to a beacon. If ν gets out of range of the RSU during the time to process the reply on the KDC, it may still be able to receive $U_\nu(t)$ by an intermediate OBU. The resending process only occurs when $U_\nu(t)$ cames from an RSU and not when it comes from an OBU.

In the case of RSUs, the process is similar, but simpler, since there is no need of any intermediate entity. RSUs can do their own update requests and this is not visible on the neighbors, there is no impact on transmitted beacons.

When receiving a beacon with a sync request appended and a MAC created with an older ρ, the RSU may or may not have ρ. The verification of the MAC by the RSU is not mandatory, it may be only performed if when instructed by the KDC (in case of it starts receiving invalid sync requests). In this case, if the old ρ is not present in the RSU, it may be sent back to it.

4.1.4 Human operator and VANET

The human operator can control the VANET through the KDC. Some operational parameters can be redefined to better fit new requirements:

default refreshment period: Although refreshments can be sent to VANET at any time, we enforce a minimum period if no action is taken by the operator.

historical cache size: h, the limit for the old refreshments stored on VUs (see Section 4.1.2). This value is always sent along with all the new refreshment messages.

basal refreshment rate: brr, the percentage of beacons that can carry a refreshment message for updating neighbors with an incomplete cache (see Section 4.1.2). This value is always sent along with all the new refreshment messages.

And some actions can be taken to manage the VANET:

Include new VUs: The cryptographic material is loaded from the KDC to the device. This is transparent to the rest of the VANET and will not trigger any new refreshment.

Exclude compromised VUs: The exclusion of VUs immediately triggers a new refreshment of the VANET keys (see Section 4.2.2).

Resurrect compromised VUs: In the KDC a new exclusive key ϵ is generated and stored, and then loaded into the device (see Section 4.2.2).

Reset all keys on VANET: In case of all the VANET distribution keys in the KDC got disclosed (a major disaster!) new ones will be generated. Moreover, a reset message is immediately sent to VANET that will allow all the VUs to recover without any need of human intervention on its hardware (see Section 4.2.2.1). This recover is only possible if the private key of the KDC has not being compromised.

4.2 Cryptographic material

With the exception of an asymmetric key pair belonging to the KDC, all the other keys used in the VANET are symmetric keys. The private key of the KDC is used to sign all the contents sent to the VANET. The public key is preloaded on all its VUs.

On Section 4.1 we mentioned a key, the routing key ρ, known by all VUs, used to create and validate MACs of routing beacons. We also mentioned a key, the exclusive key ϵ, shared by each of the VUs and KDC, used to secretly exchange messages between the KDC and each VU.

The exclusive keys of all VUs form the **set of exclusive keys** E. The components of E are initialised with random values (seeds) and keep the same value in all the different time intervals:

$$E(t+1) = E(t) \tag{4.1}$$

To securely distribute the refreshment messages across the VANET, we use n auxiliary group keys, which form the **set of group keys** G.

The components of G are associated to a time interval t:

$$G(t) = \bigcup_{i=0}^{n-1} g(t)_i \tag{4.2}$$

For $t = 0$, all the components of G are initialised with random values (seeds). The refreshment of all components of G from t to $t+1$ is based on a chaining process dependent of the refreshment update $r(t+1)$:

$$G(t+1) = \bigcup_{i=0}^{n-1} g(t)_i \oplus r(t+1) \tag{4.3}$$

The two previously mentioned sets, E and G, are both made out of symmetric keys randomly initialised (with seeds). Between the two sets, the only difference is the fact they are different or the same in distinct time intervals (see Equation 4.1 and Equation 4.3). The components of the 2 sets of symmetric keys are together know as the **set of distribution keys** S.

$$S = E \cup G \tag{4.4}$$

The components of the set S may be used indistinctly but always have its origin well identified. The origin is specially important to maintain the constraint related with the transition between time intervals:

$$S(t+1) = E(t) \cup \bigcup_{i=0}^{n-1} g(t)_i \oplus r(t+1) \tag{4.5}$$

Each VU ν has an ordered subset of S, the subset P_ν, with d distribution keys. Each subset P_ν is distinct from any other. The subset P_ν is built based on $p(\nu)$ (detailed in the upcoming Section 4.2.1), that selects the elements of S associated to ν.

$$P(t)_\nu = \bigcup_{\forall j \in \{p(\nu)\}} s(t)_j \tag{4.6}$$

Since the components of S have an identifiable origin, the same happen with the components of $P(t)_\nu$. Moreover, it is assured by $p(\nu)$ that in each subset $P(t)_\nu$:

1. There is one, and only one, component with origin in E;

2. There is one, and only one, common component in all the different subsets.

By (1.) we honor the need for the existence of one symmetric key shared only between a VU and the KDC, the exclusive key ϵ_ν. By (2.) we associate the routing key $\rho(t)$ to the set G and consequentially S. The previously introduced routing key $\rho(t)$ is a special component of the group keys. Although the routing key has a distinct purpose from the group keys (routing authentication and distribution of key updates, respectively), the same key can interchangeably be used for the two purposes.

Similarly to Equation 4.5, all subsets are independently refreshed:

$$P(t+1)_\nu = \epsilon_\nu \cup \bigcup_{\forall p \in P(t)_\nu \setminus \epsilon_\nu} p \oplus r(t+1) \tag{4.7}$$

On each VU, the ordered subset P_ν can be represent as an array of e elements. The index 0 contains the routing key, which is the same in all VUs, $\rho(t)$. The index $e-1$ contains a distinct key in all the subsets, ϵ_ν. All the group keys on P_ν (indexes from 0 to $e-2$) are always associated to the same time interval.

Besides all the keys of S, KDC stores an ordered **set of images** I of a non-stored **set of auxiliary values** A. The set I has the same size as E, there is one and only one value associated to each VU. Based on randomly generated auxiliary values, α_ν for each VU ν, using a cryptographically secure one-way function, $H(x)$, I is computed as:

$$I = \bigcup_{\forall \alpha \in A} H(\alpha) \tag{4.8}$$

Each VU ν has access to one and only one value of A associated, α_ν. After the creation of I each of the auxiliary values on A is loaded into the correspondent VU, and discarded from the KDC, that only stores I.

4.2.1 Setup

The distribution keys S are arranged in a complete binary tree structure, where the number of leafs, V, is the maximum number of VUs supported by the KDC. Hereafter, we will assume that the number of VUs is equal to $V = 2^v$, for any $v \in \mathbb{N}^+$, unless otherwise stated.

Each component of S is associated to a node of the binary tree. On the root node we have the routing key ρ, on each leaf we have a different exclusive key ϵ (all the elements of E are in the leaf level).

The binary tree defines successive binary partitions of the VANET. For each partition is given a new and distinct key. Figure 4.3 illustrates a scenario for a VANET with a maximum of 4 VUs.

All the VUs in the VANET are associated to $s_0 = \rho$ (partition $P0$). Then, the VUs are successively divided in 2 partitions with the same size, until each of the partitions having just one VU.

In our illustrative case (see Figure 4.3), we obtain 1 partition with 4 VUs ($P0$), 2 partition with 2 VUs ($P1$ and $P2$) and 4 partitions with 1 VU ($P3$, $P4$, $P5$ and $P6$).

Starting on the leafs of the tree, that represent a partition with only one VU, e.g. , we give to that VU the key associated to its leaf ($s_3 = \epsilon_0$) and all the keys associated to all the ascendant nodes, namely s_1 (or s_2) and $s_0 = \rho$.

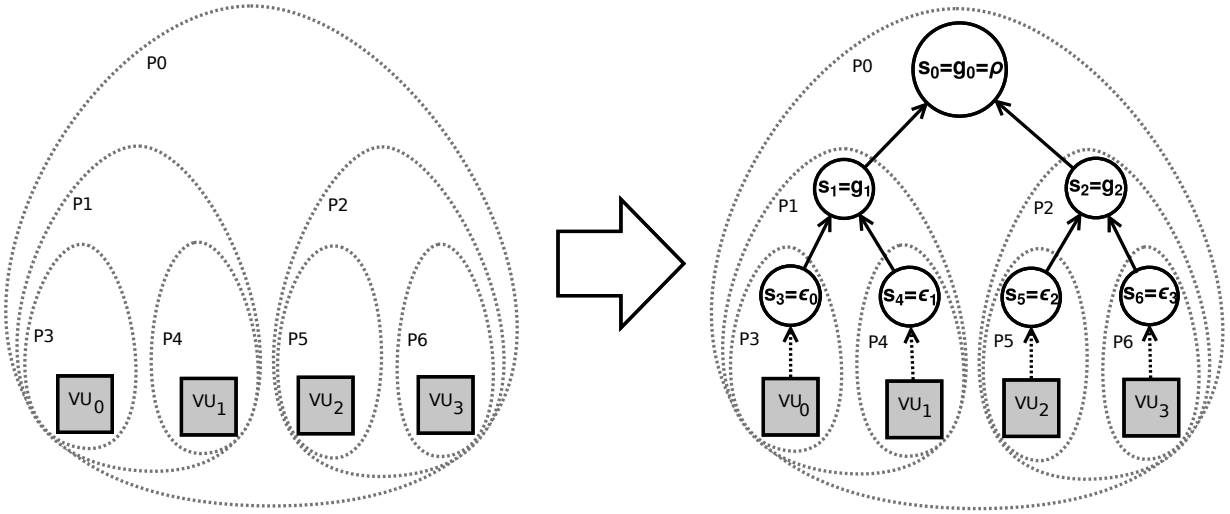


Figure 4.3: Organization of the distribution keys S by the KDC on a binary tree for a maximum of 4 VUs

Index in P	VU_0	VU_1	VU_2	VU_3
0	$s_0=g_0=\rho$			
1	$s_1=g_1$		$s_2=g_2$	
2	$s_3=\epsilon_0$	$s_4=\epsilon_1$	$s_5=\epsilon_2$	$s_6=\epsilon_3$

Table 4.1: The elements on the index of all the subsets P in a VANET with a maximum of 4 VUs

---

Since the binary tree is a complete binary tree, with V leafs, allowing up to V VUs in the VANET, the number of elements of S, t, is:

$$t = n + V = V - 1 + V = 2 \cdot V - 1 \tag{4.9}$$

The number of distribution keys stored in each VU is:

$$e = \log_2(V) + 1 \tag{4.10}$$

With 4 VUs, the set of distribution keys contains $t = 7$ keys (3 group keys and 4 exclusive keys), and each of the 4 subsets contains $e = 3$ elements. The 4 subsets are shown on Table 4.1.

Instead of a tree with V leafs, we represent our tree as a **main tree** where on each of the leafs, main leafs, we have an **inner tree** with the same size. Moreover, the main tree

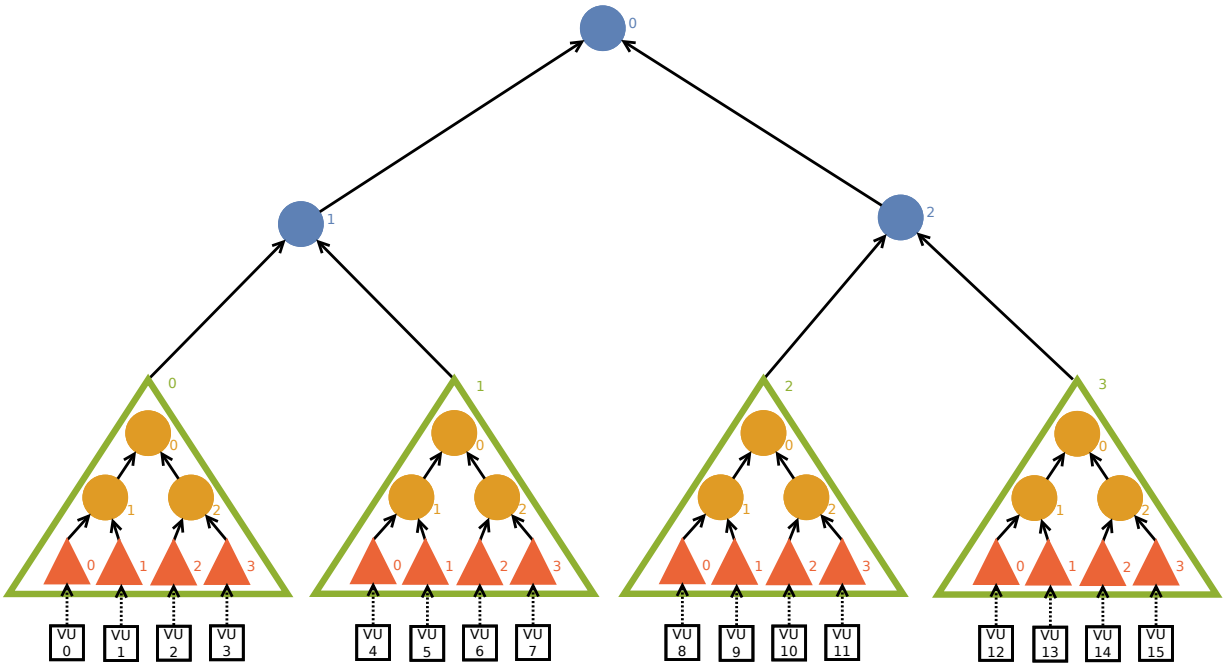


Figure 4.4: Main tree (blue and green) and inner trees (orange and red) supporting up to 16 VUs. Branch nodes are represented as circles and tree nodes as triangles. Each of the main leafs (green triangles) have a complete inner tree.

---

and the inner tree have the same number of leafs, l.

$$l = \sqrt{V} \tag{4.11}$$

We use the same indexes for both branch nodes and leafs of the trees. By branch nodes we mean all the nodes that are not leafs. In a complete binary tree with l leafs, there are $l - 1$ branch nodes. Figure 4.4 illustrates such division for $V = 16$ and $l = 4$: the main tree is composed by 3 branch nodes in blue and 4 leafs in green; each of the leafs of the main tree contains an inner tree composed by 3 branch nodes in orange and 4 leafs in red; the 16 VUs are assigned to the 4 leafs of the 4 inner trees.

It is possible to distinguish all the branch nodes with indexes from 0 up to $l - 2$. Similarly it is possible to distinguish all the leafs with indexes from 0 up to $l - 1$. Based on this, for each tree we can use a variable length code for indexes of both branch nodes and leafs, with indexes of branch nodes taking half of the size of indexes of leaf nodes. If the first part is between 0 and $l - 2$ (different from $l - 1$), the index refers to a branch node, and there is not a second part. If the first part is equal to $l - 1$ (an invalid index for branch nodes), indicates that there is a second part, with the index of the leaf.

Addressed Node		Index			
		Main		Inner	
		Branch	Leaf	Branch	Leaf
	Main Branch Node	0..2			
	Inner Branch Node	3	0..3	0..2	
	Inner Leaf Node	3	0..3	3	0..3

Table 4.2: Variable length code for distribution keys

To distinguish between branch nodes and leafs of a single tree we have a code with 1 or 2 parts. Since we have a main tree and multiple inner trees, to distinguish all the nodes (all the components of S), we have a left-to-right variable code with 1 (branch nodes of main tree), 3 (branch nodes of the inner tree) and 4 (leaf nodes of the inner tree) parts. We do not use the leaf nodes of the main tree alone by their own, so there are no code with 2 parts. For the example presented in Figure 4.4, the variable length codes of the distribution keys are shown in Table 4.2.

Under all the circumstances, the distribution keys associated to the branch nodes of main tree are the most used, and the less used are the ones associated to leaf nodes of inner trees, thus, we have a shorter identification for the most used indexes (see Section 4.3.1.1). Another benefit of the use of a tree of trees is is the possibility to explore solutions based on tables of $l = \sqrt{V}$ entries instead of V entries (see Section 4.3.1.2).

Although the indexes of the distribution keys associated to VUs (the set E) are composed by 4 parts, to distinguish among VUs, only 2 of those parts contain meaningful information: the indexes of both leafs. Thus, to strictly identify VUs we use an index made out of the concatenation of leaf's indexes. To all the V VUs, this is coincident with a linear indexing from 0 to $V - 1$ on all inner leafs.

For a VU ν, its index is composed by $b = \log_2(V)$ bits, where the **most significative half**, ν^h, is the index of the main tree leaf and the **lowest significative part**, ν^l, the index of inner tree leaf.

The ascendant of any binary tree node, x, excluding the root node, is given by:

$$a(x) = \lfloor \frac{x-1}{2} \rfloor \tag{4.12}$$

With $A(x)$ defined as a function that returns the set of all ascendants of node x, $B = l - 1$, "|" representing concatenation, all the indexes of the subset P_ν, the path from

ν up to the root node, is given by:

$$p(\nu) = B \mid \nu^h \mid B \mid \nu^l \cup A\left(\nu^h + B\right) \cup \bigcup_{\forall x \in A\left(\nu^l + B\right)} \left(B \mid \nu^h \mid x\right) \tag{4.13}$$

In Figure 4.4, from left to right, VUs take the indexes $0, 1, 2, 3, 4, \ldots, 11, 12, 13, 14, 15$, all expressed with 4 bits, coincident with the concatenation of the 2 bits of the correspondent main leaf (orange triangle) and inner leaf (red rectangle).

Again in Figure 4.4, for the leftmost VU $\nu = 0$, $p(0)$ returns the set composed by index of VU, the indexes of main branch nodes and the indexes of inner branch nodes, respectively:

$$B \mid \nu^h \mid B \mid \nu^l = 3|0|3|0 \tag{4.14}$$

$$A\left(\nu^h + B\right) = A(0+3) = A(3) = \{0, 1\} \tag{4.15}$$

$$\bigcup_{\forall x \in A\left(\nu^l + B\right)} \left(B \mid \nu^h \mid x\right) = \bigcup_{\forall x \in A(0+3)} (B \mid 0 \mid x) = \{3|0|0, 3|0|1\} \tag{4.16}$$

All the VUs are loaded with the correspondent subset of distribution keys, P, associated to the most recent time interval present in the VANET. The KDC can at any time, send messages to VANET with orders to refresh, reset or freeze P. Based on such orders, it is possible exclude, re-include and possible add new VUs.

4.2.2 Manipulation and synchronization

The KDC performs 2 types of operations on distribution keys, S, refreshments on the set of group keys G and replacements of all S (E and G) and I.

The refreshment of G from t to $t+1$ is based on a random value, the **refreshment update** $r(t+1)$. All the components of G in the state t are refreshed with $r(t+1)$ (see Equation 4.5). With the refreshment update $r(t+1)$, each of the VUs can perform the same operation in the correspondent subset P_ν, (see Equation 4.7), and keep itself updated with the KDC.

To secretly distribute $r(t)$, the KDC uses one or more distribution keys (either a group key or a exclusive key) from $S(t-1)$ to encrypt it and produce the correspondent **refreshment message**. Each of the refreshment messages sent to the VANET have an encrypted

value of $r(t)$, thus each VU with the correct distribution key on the respective subset P_ν is able to decrypt it.

To refresh all the VANET from t to $t+1$, the KDC only needs to send one message, with $r(t+1)$ encrypted with $\rho(t)$. As ρ is known by all VUs, the refreshment update can be obtained by all.

Multiple messages, encrypted with different distribution keys, are used to exclude one or more VUs. If one or more VUs got pinpointed as compromised or lost, part of the distribution keys is considered compromised. One of the compromised keys is always ρ, since it is known by all VUs. In this scenario, neither ρ nor any of the distribution keys present in the subsets P_ν of the compromised VUs can be used to encrypt $r(t+1)$.

With the exception of all the distribution keys associated to the compromised VUs, all the others can be safely used as key to encrypt $r(t+1)$ and produce refreshment messages. Some of those distribution keys are used to produce the minimal number of refreshment messages, which are enough to update all but the VUs to exclude. The process of selecting the distribution keys is described in Section 4.3.1.1. On the KDC, once the excluded VUs are marked as compromised all future requests based on the fallback method (see Section 4.1.3) are ignored.

The other type of operation performed by the KDC is the replacement of the exclusive keys on E and images on I, used to re-include any of the compromised VUs. To re-include a VU ν that was compromised at the time interval t_c, the exclusive key ϵ_ν is replaced by a new random value (seed) associated to the most recent time interval, t_r. Besides that, a new α_ν is randomly generated and the image, ι_ν, in the set I is replaced by a new value $\iota_\nu = H(\alpha_\nu)$. After this, the same device, or a different one, can safely take the identification of ν and be loaded with the new value of ϵ_ν, associated to t_r, along with the α_ν (α_ν is discarded from KDC). The existence of a seed with a time interval t_r greater than t_c indicates that the VU is resurrected.

Since the compromised version of VU was explicitly excluded from the refreshment process, the newly resurrected VU, with the new ϵ associated to t_r, inherits the old context and needs to use the fallback method (see Section 4.1.3) to synchronize its subset of distribution keys P. Since the exclusive key ϵ_ν was replaced, the KDC can recognize the messages produced by the resurrected instance of ν, encrypted with the new ϵ_ν, and provide the needed data.

4.2.2.1 Recovering from the disclosure of all the distribution keys

If the KDC is compromised, all the distribution keys S (including ρ and all the exclusive keys ϵ) are available to an attacker. This means that the use of the distribution keys does not guarantee the secrecy of any new data distributed with them. By compromising the KDC an attacker also have access to the set of images, I, but this is not relevant.

To reset all the keys in all the VUs, the KDC sends a **reset message**, a special refreshment message (without refreshment key and with the number of distinct messages defined as 0), associated to state $t + 1$. At the same time, the KDC replaces all the distribution keys S (both E and G) by new ones associated to $t + 2$.

On state $t+1$ all the VUs are excluded from the chained process (none of them can use the refreshment message), but allowed (informed!) to reset the subset P_ν to $t+2$. The reset is performed based on the fallback method (see Section 4.1.3), with an extra step, that will guarantee the secrecy of the new values being distributed. All the VUs will secretly receive their correspondent subset P_ν with the new keys, previously created on state $t+2$.

The extra step relies on the set of images I and the correspondent set of auxiliary values A. The set I is stored in the KDC, so also compromised, but the auxiliary values are distributed along the VUs. For each VU ν, the KDC can receive each α_ν, encrypted with its asymmetric public key, compute $H(\alpha_\nu)$ and compare it with the correspondent ι_ν of I.

Assuming the secrecy of the private key of the KDC, a compromised KDC can recover from the disclosure of all the distribution keys stored on it, because it is possible to distinguish a sync request made by a VU and a sync request made by the attacker that holds E but not the distributed A. The reply from the KDC to each VU ν, instead of being encrypted with the disclosed key ϵ_ν is in this case, encrypted with the now compromised α_ν.

The inclusion of α_ν encrypted with the KDC public key creates an overhead on the fallback method. Although this step is essential to recover from the disclosure of all the distribution keys, it does not need to be present on the majority of the requests, that will be made by VUs to recover from a period of isolation. Since the case of recovering from the disclosure of all the distribution keys is easily distinguishable from the case of synchronization after a period of isolation, (the first is triggered by a reset message), the inclusion of the encrypted value of α on sync requests, as well as the use of α instead of ϵ to decrypt the sync replies, is only part of the protocol when recovering from the disclosure of all the distribution keys, and never when synchronizing after a period of isolation.

In case of the KDC being compromised along with some of the VUs, the recovering must be done in two steps: first recover from the disclosure of all the distribution keys and them the exclusion of the compromised VUs.

4.2.3 Dealing with an arbitrary number of VUs

We assumed until now that the number of VUs is $V = 2^n$ for some positive n. This is a strong assumption and cannot be guarantied on a real scenario. When handling a smaller number of VUs, assuming that the VANET is always composed by the maximum number would imply that some refreshment messages would be created and sent to VANET but not used by any of the VUs. This would have a negative impact since VUs would have to process and store useless messages as well as consume bandwidth to exchange them.

In order to have V as a real maximum and not necessarily the effective number of VUs, without suffering from the overhead of a big number V when handling a small number of VUs, the KDC flags all the nodes as used or not used. Based on that, it is possible to send messages only to the nodes that are effectively in the VANET, without maintaining a tree with an arbitrary and mutable number of leafs. Using a big tree and flagging its unused nodes, when compared with the use of a tree with the minimum number of nodes, has in fact an overhead. This is explained in detail in the upcoming Section 4.3.1.1. Nevertheless this overhead is extremely small and the alternative, always keep the balanced, would be much heavier.

Figure 4.5 illustrates the case of only 4 VUs assigned on our toy example introduced before, a VANET with 16 VUs. This is one of the many possible arrangements. Any different arrangement will result on different nodes flagged as unused. From the point of view of messages sent to the VANET, we will always have the same number. However, since the nodes in the main tree have a shorter identification that the one in inner trees (see Section 4.2.1) we can enforce the use of the main tree nodes to obtain shorter messages.

4.2.3.1 Optimum distribution of VUs in the tree

For an arbitrary number of VUs, a, the best distribution is obtained by any arrangement that maximizes the minimum distance, m, between any two consecutive VUs.

$$m = \frac{V}{2^{\lceil \log_2(a) \rceil}} \tag{4.17}$$

In the example presented in Figure 4.5 the maximal minimum distance between con-

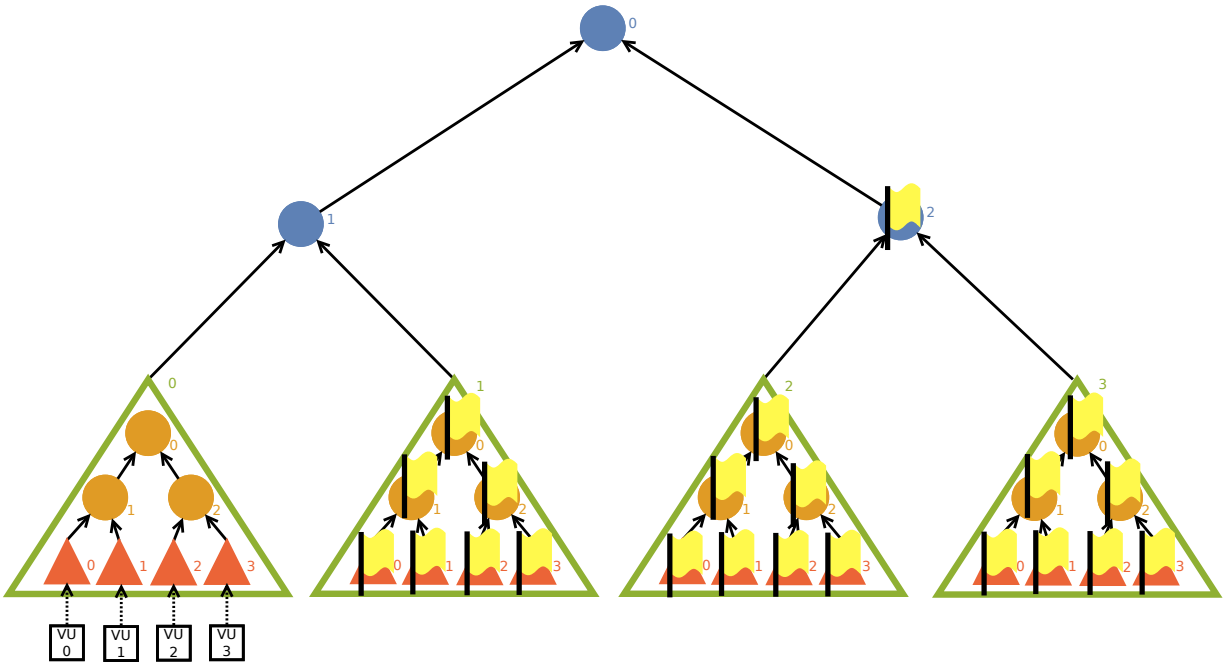


Figure 4.5: Unused nodes flagged in a non-optimal distribution with 4 members in a VANET with a maximum of $V = 16$

secutive VUs is 1, and not the maximum possible to obtain, $4 = \frac{16}{4}$.

One of the distributions that maximizes m for case of 4 VUs is shown in Figure 4.6. In this case each VU is virtually attached to the an upper point of the tree.

During the inclusion or exclusion of VUs, the flagged nodes may change. If the number of VUs is reduced by half or more, it is not guaranteed that the distribution keeps optimal. In the case of a big reduction of VUs, it may be advised to reindex the active ones. We did not consider the need for reindex the VUs, since on our real scenario such variation is not present neither expected.

4.3 Secure routing

To enforce the security of the routing protocol all the previous routing messages (beacons) are replaced with a secure version, which ensures its integrity. The routing information is the same of the base protocol.

Instead of a message solely with the routing information, VUs sends a message with:

- routing information;
- time interval, t, of the most recent routing key ρ;

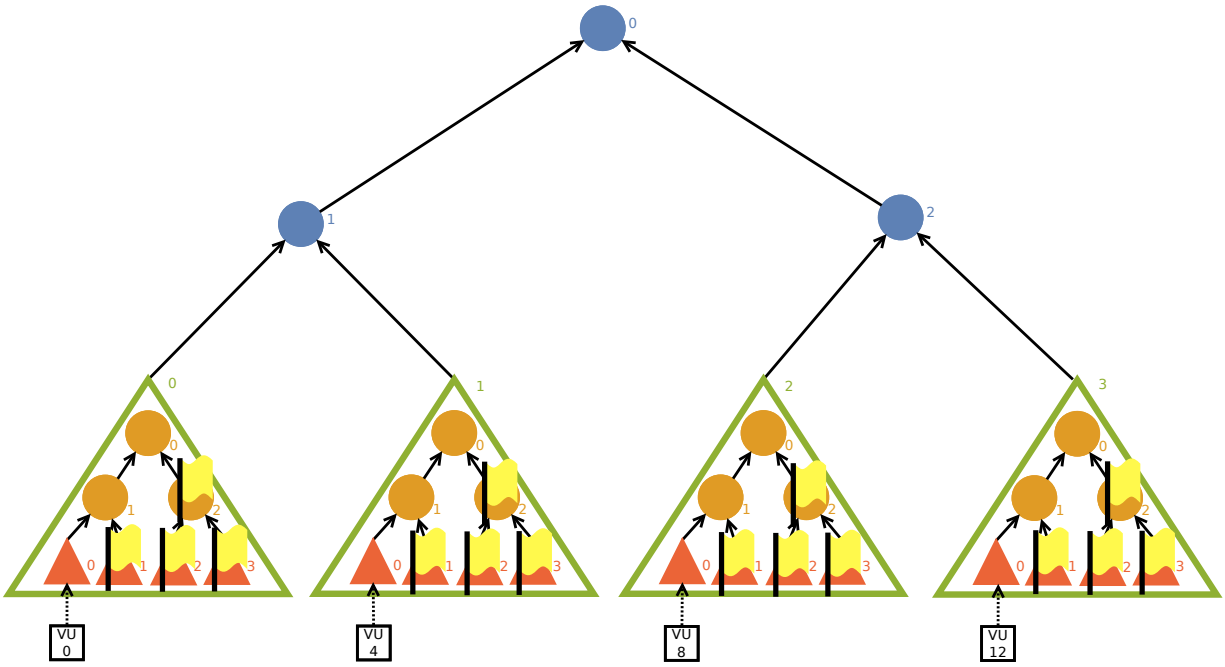


Figure 4.6: Unused nodes flagged with the dispersion of 4 members on a maximum of $V = 16$ possibilities. The minimum distance between consecutive VUs is maximized ($m = 4$).

- extra payload;
- state of the historical cache (see Section 4.1.2);
- MAC computed with $\rho(t)$.

Assuming the secrecy of $\rho(t)$ across VUs, the routing process is protected against external attackers, due to the MAC. Since this assumption is not strong enough for a long period of operation, we included a mechanism to modify ρ, and implicitly, if needed, exclude any of the VUs in operation. To support the key modification, beacons can carry an extra payload, either **refreshment messages** or **sync messages**.

Both type of messages are associated to a well defined time interval, and both carry some auxiliary control information, for this specific time interval: the number of distinct refreshment messages produced by KDC (see Section 4.3.1.1), the size of the historical cache and the basal refreshment rate, respectively, h (see Section 4.1.1) and brr (see Section 4.1.2).

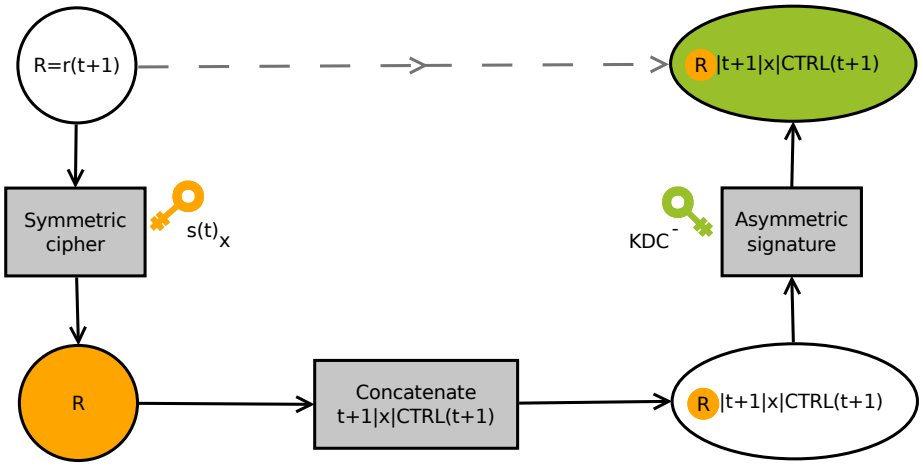


Figure 4.7: Wrapping of the refreshment update $r(t+1)$

4.3.1 Refreshment messages

The refreshment messages are built by the KDC, according to Figure 4.7, with the goal of secretly distributing a refreshment update $r(t+1)$ to all or part of the VUs.

The refreshment messages are made out of:

- $r(t+1)$ encrypted with an element of $S(t)$;
- identification of the element of $S(t)$ used to encrypt $r(t+1)$;
- value of t;
- control information (number of distinct refreshment messages, brr, h);
- KDC signature.

4.3.1.1 Selection of the distribution keys

To distribute $r(t+1)$ to the entire VANET, is produced only one refreshment message, with $r(t+1)$ encrypted with ρ. Since ρ is a common entry in all the subsets P, this key is the best candidate to encrypt $r(t+1)$. This is enough to protect the VANET against external attackers, unless such attackers had access to ρ.

To exclude one or more VUs, the KDC uses the binary tree structure to select a minimum set of safe distribution keys. All the distribution keys present on the subset of any of the VUs to exclude are considered compromised keys. The minimum set of safe keys is

built based on all the siblings, not flagged as unused (see Section 4.2.3), of nodes representing the compromised keys. If there is more than one VU to exclude, from the previous group of selected siblings are excluded the ones that are compromised.

After generating multiple refreshment messages for $t+1$, the KDC can start again using ρ as the only distribution key, for $t+2$. This happen because all of the excluded VUs were unable to receive $r(t+1)$ and update their state to $t+1$, thus, are unable to decrypt any future refreshment messages, and more important, unable to create MACs with the most recent $\rho(t+1)$ present in VANET.

With V VUs, the number of distribution keys used to create the refreshment messages (sent to VANET) to exclude $c > 0$ VUs at once, in the worst case, is give by:

$$m_w = \lfloor c \cdot (\log_2(V) - \log_2(c)) \rfloor = \lfloor c \cdot \log_2 \left(\frac{V}{c} \right) \rfloor \ , \ \forall \ 0 < c \leq V \tag{4.18}$$

$$\text{Max}\left(m_w\right) = \frac{V}{2} \tag{4.19}$$

On our toy example introduced before, a VANET with 16 VUs, the exclusion of one VU is illustrated in Figure 4.8 and the exclusion of 2 VUs in Figure 4.9, both identified with a red cross. On black with a red circle around we have the compromised elements of S, with a dashed blue circle (only Figure 4.9) we have the sibling nodes that are also compromised and with a continuous blue circle the sibling nodes that represent the minimum set of distribution keys needed to refresh all the VANET but the VUs to exclude. In both cases, half of the remaining elements are updated with the same number of messages equal to the number of elements to exclude, 1 and 2, respectively.

The worst case, when excluding c VUs, occurs when all the tree nodes at level $\lceil l_c = \log_2(c) \rceil$ (assuming the root node as level $l = 0$) have at least 1 compromised VU in the descendant leafs (see Figure 4.10). If $c > 0$ and one or more levels at l_c (or higher) just have non-compromised VUs, it means that the keys at theses levels can be used to produce refreshment messages (with a high number of VUs served by those messages). With at least 1 VU associated to all the nodes at l_c, all the nodes from level 0 to l_c are compromised (no refreshment messages need to be considered from them). The needed refreshment messages are associated to the bottommost $l_b = \log_2(V) - \log_2(c)$ levels. The nodes on the l_b bottommost levels can be seen as c pseudo-detached trees (each one with the pseuso-root node at level l_c). When $c = 2^b \leq V$ for any $b \in \mathbb{N}^+$ each pseudo-detached tree has associated 1 and only one compromised node, and Equation 4.18 can be simplified

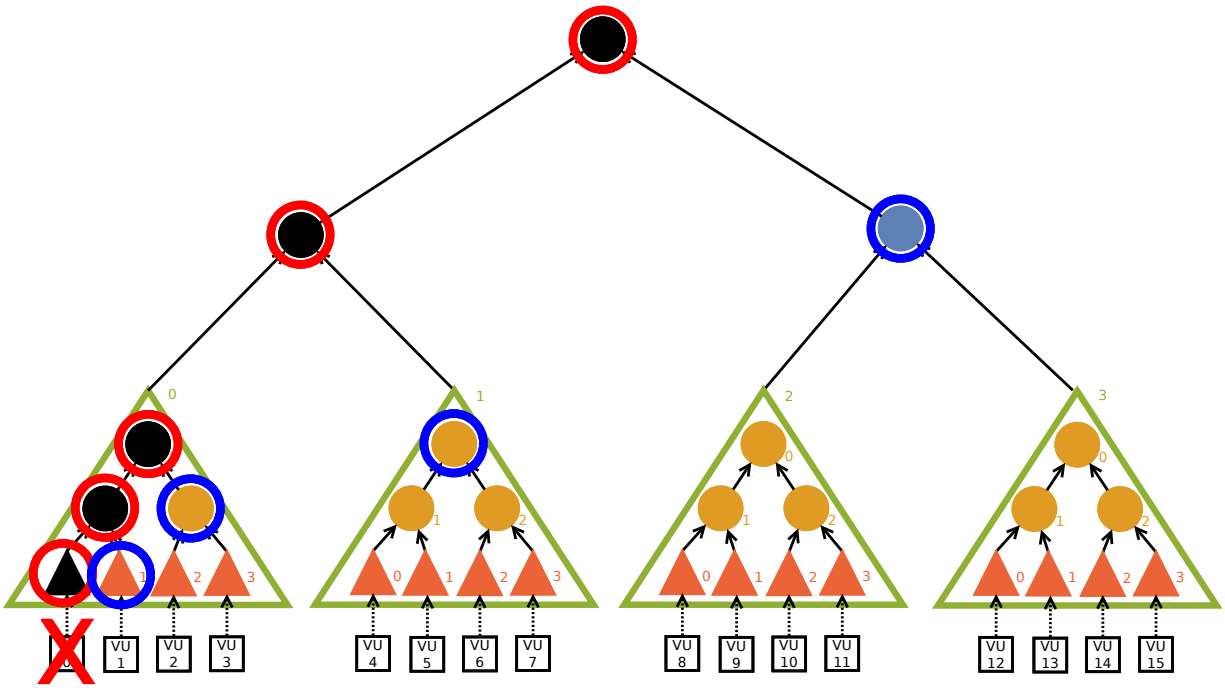


Figure 4.8: Selection of the distribution keys during the exclusion of 1 VU (leftmost). The compromised keys are on black with a red circle around. The selected distribution keys are with a blue circle around.

---

removing the floor operator ($\lfloor\rfloor$). With the floor operator ($\lfloor\rfloor$) we have a generalization for any $c \in \mathbb{N}^+$.

From the m_w messages created when excluding c elements, in the worst case, to update half of the remaining elements are used c of those messages (the ones associated to the level $l_c + 1$).

In Figure 4.11 is shown the number of messages created to exclude a different number of VUs, according to Equation 4.18, with $V = 16$. The maximum number is obtain for $c = 8$, where all the non-compromised VUs will receive a different message. The worst case occurs when the minimal distance between two consecutive VUs to exclude is maximized, all the even (or odd) indexes. On the other hand, if the 8 VUs to exclude are placed on the 8 leftmost (or rightmost) positions, the best case occurs and only one message will be needed.

With a tree structure capable of handling up to $V = 2^v$ VUs, for any $v \in \mathbb{N}^+$, for an arbitrary number $a \leq V$ of VUs, with the optimal distribution described in Section 4.2.3.1

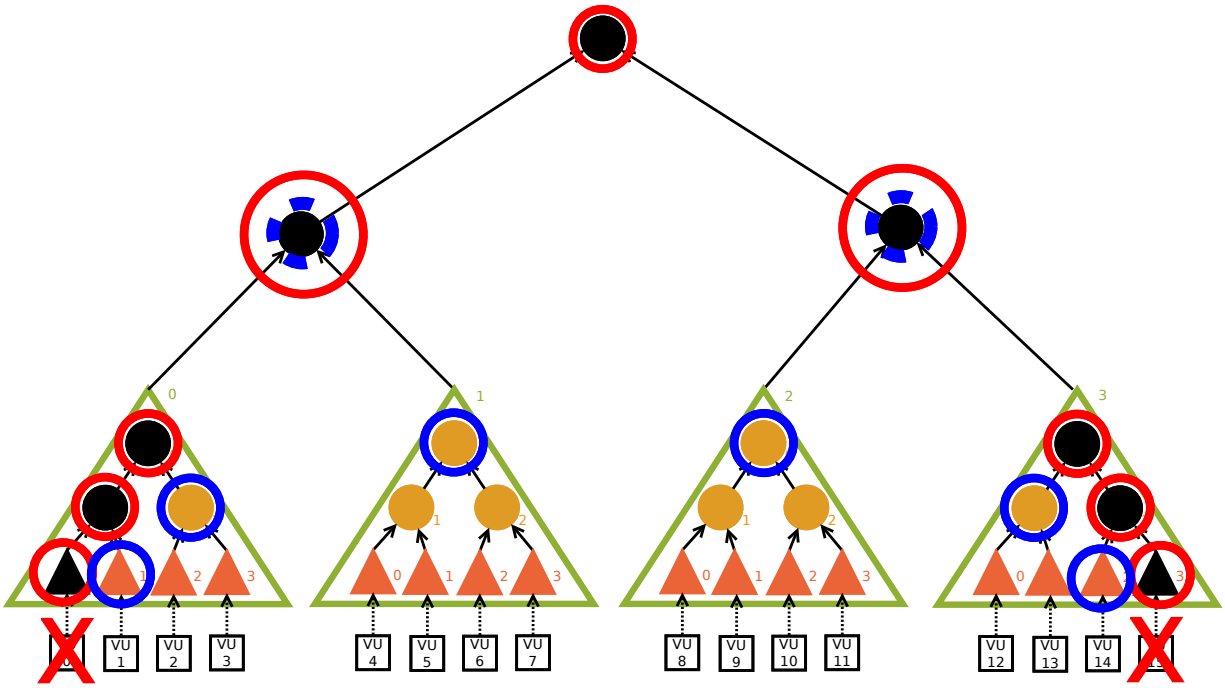


Figure 4.9: Selection of the distribution keys during the exclusion of 2 VUs (leftmost and rightmost, one of the worst cases). The nodes with a dashed blue circle around were candidates if excluding just 1 VU. When excluding 2 VUs the nodes with a dashed blue circle are also compromised and not used.

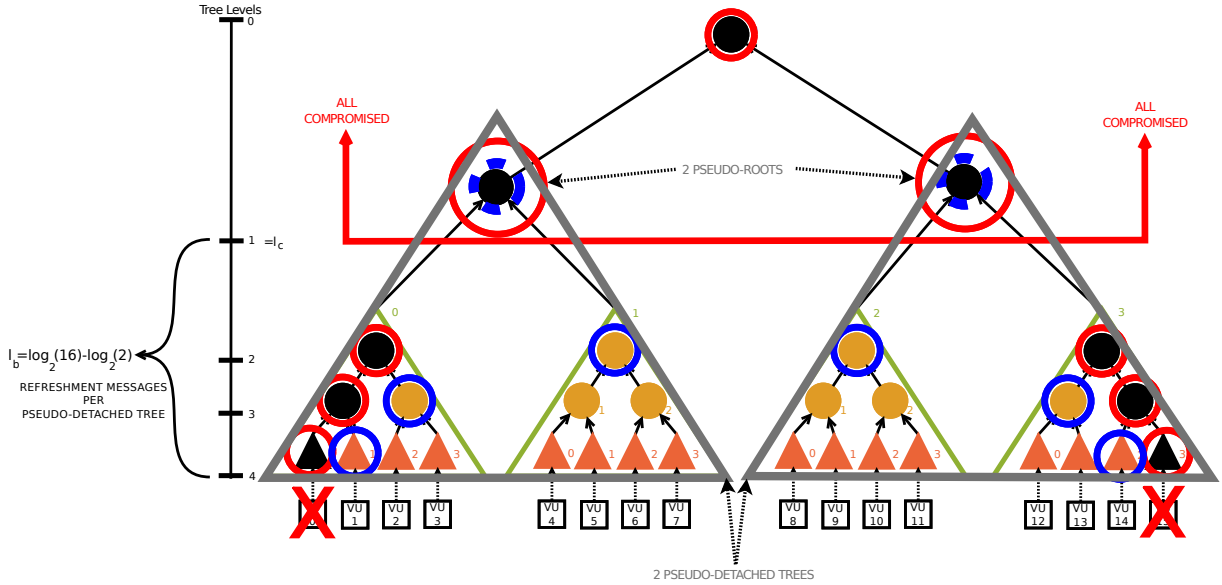


Figure 4.10: Detailed analysis when excluding 2 VUs and its relation with Equation 4.18

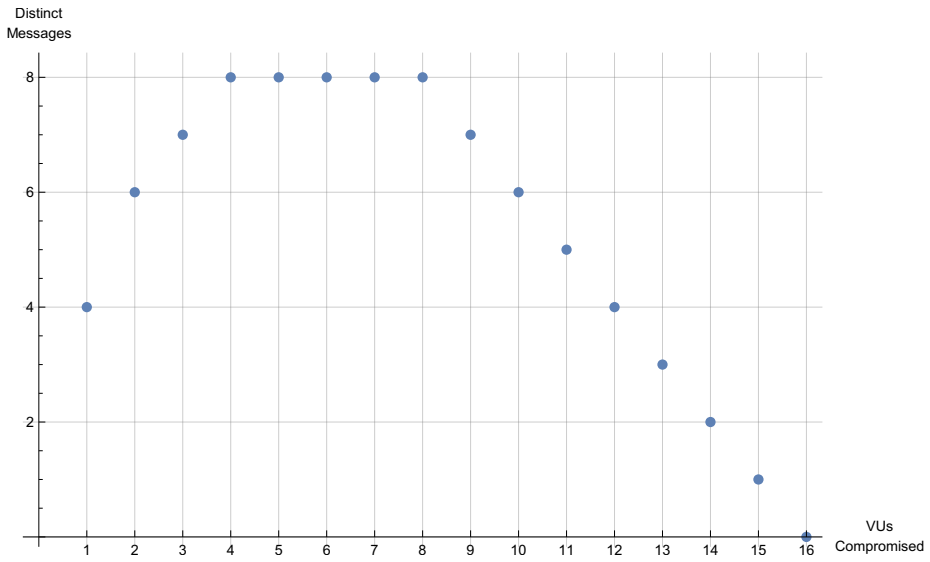


Figure 4.11: Messages created to exclude VUs in a VANET with 16 VUs (worst case)

and the existence of flagged nodes, the Equation 4.18 has a different, but similar formula:

$$m_w = \lfloor c \cdot (\lfloor \log_2(a) \rfloor - \log_2(c)) \rfloor + \min\left(c \cdot \log_2(x), y\right) \ , \forall\ 0 < c \leq a \leq V \tag{4.20}$$

$$x = \lceil \frac{2^{\lceil \log_2(a) \rceil}}{a} \rceil \tag{4.21}$$

$$y = a - 2^{\lfloor \log_2(a) \rfloor} \tag{4.22}$$

With x as the maximum number of VUs associated to the tree nodes on level $l_a = \lfloor \log_2(a) \rfloor$ of the tree structure in use (assuming the root node as level $l = 0$) and y as the number of the tree nodes that have this maximum number (see Figure 4.12). The x value is always 1 when $a = 2^b$ for any $b \in \mathbb{N}^+$ or 2, otherwise. Due to this fact, $\log_2(x)$ is always 0 or 1 and has a small impact on m_w.

When $a = 2^b \leq V$, for any $b \in \mathbb{N}^+$, we get $x = 1$ and $y = 0$ so the Equation 4.20 falls into the Equation 4.18, with a taken the place of V (the number of refreshment messages depends on the number of VUs in use a and not on the maximum V).

In the example shown in Figure 4.12, when excluding 3 VUs, the worst case occurs in any of the following associations:

$$(\mathrm{VU}_0 \oplus \mathrm{VU}_2) \bigwedge (\mathrm{VU}_4 \oplus \mathrm{VU}_6) \bigwedge (\mathrm{VU}_8 \oplus \mathrm{VU}_{10}) \tag{4.23}$$

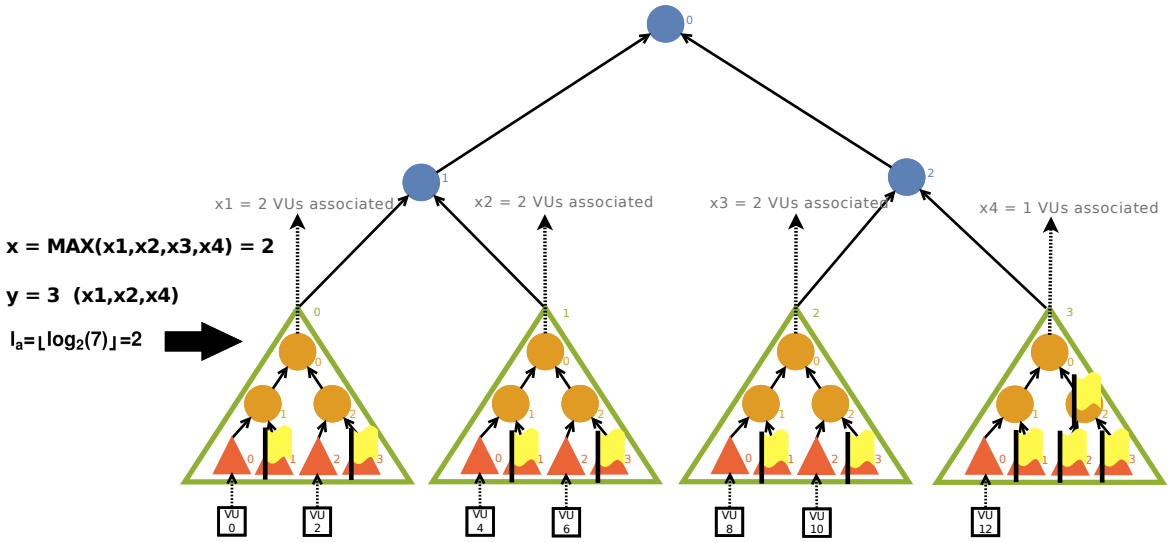


Figure 4.12: Example of the parameters x and y for generalization of the worst case when flagging the unused nodes (Equation 4.20), with $a = 7$. Since the unused nodes are flagged neither x nor y is dependent of V.

These associations yield 4 refreshment messages (the Equation 4.20 with $c = 3$, $a = 7$, $x = 2$ and $y = 3$), 1 for VU_{12} and 3 others for each of the non-compromised VU of each *"XOR pair"*. Any other possibility yields a smaller number of messages.

The only impact of the strategy of flagging the unused nodes (see Section 4.2.3) instead of balancing the tree is given by the difference between Equation 4.18 (assuming a balanced tree with V leafs for V VUs) and Equation 4.20 (assuming a tree with arbitrary number of leafs, possibly flagged to fit a VUs): 1 extra refreshment message per excluded VU (limited to the difference given by y) in the worst case (that is not the most probable). Since the tree balancing is not a trivial nor an overhead-free task (the new relations among VUs created in the KDC must be reflected in the VUs themselves), we considered the use of flagged nodes the best approach.

4.3.1.2 Epidemic propagation of refreshment messages

The refreshment messages are initially sent to the RSUs and then disseminated by all of them to all the others, following a best effort, epidemic approach. The success of such dissemination depends on a fast and efficient detection of the neighbors and the selection of the right refreshment message, by all VUs.

The VUs are associated to the leafs of the binary tree and the refreshment messages to all the branch nodes as well as to the inner leafs. Thus, the association between VUs and

the useful messages can be obtained using $A(x)$ defined as a function that returns the set of all ascendants of leaf x (based on successive iterations on Equation 4.12).

The refreshment messages associated to the topmost branch nodes are the most commonly used in the VANET. To keep all the previous VUs on VANET, only one refreshment message is sent, associated to the root of the main tree, while when excluding one or more VUs, the refreshing messages associated to the topmost nodes of the trees have a higher coverage of VUs, so will be often needed and transmitted. Thus, with a list of all the ascendant branch nodes associated to a neighbor, starting a search from the topmost branch nodes is the best strategy to find any useful refreshment message for a neighbor.

With the division of a binary tree into the **main tree** and the **inner trees** (see Section 4.2.1) the indexes of the VUs are composed by the indexes of both main and inner leafs. The ascendant branch nodes of the main tree are the topmost ones, so they should be searched first. Due, with the division into main and inner tree, its possible to first search a smaller group of ascendants (the main branch nodes) and then, when needed, a second group of ascendants (the inner branch nodes).

In a VANET with V VUs, instead of single group with $h = \log_2(V)$ ascendants, that has to be calculated or stored for each possible neighbor, we have 2 groups of $\frac{h}{2}$ ascendants. Due to the fact that it is more probable to find an useful refreshment message associated to the first group of $\frac{h}{2}$ ascendants in the main tree, the $\frac{h}{2}$ ascendants in the inner tree do not always need to be calculated or accessed. This strategy gives us advantages either if the indexes of ascendant branch nodes are calculated each time a neighbor is detected or if the indexes are precalculated and stored.

When using precalculated data, since the indexes are the same for the main tree and each of the inner trees (see Figure 4.4), according to Equation 4.11 we only store the ascendants for a tree with $l = \sqrt{V}$ instead of V leafs. As the indexes of the VUs are composed by the indexes of both main and inner leafs (see Section 4.2.1), the same information for the indexes of the ascendants is used for the main and the inner branch nodes, which drastically reduces the information being stored.

The selection of the refreshment messages to update a neighbor can be accelerated if each of the refreshment messages associated to branch nodes become referenced by the respective descendant leafs nodes. This makes the time of a search for a refreshment message constant, at the price of an increase in memory. The division into main and inner trees can also be explored to reduce the size needed to have a constant search time, or to only apply the constant search to the main tree (which has a higher probability to contain

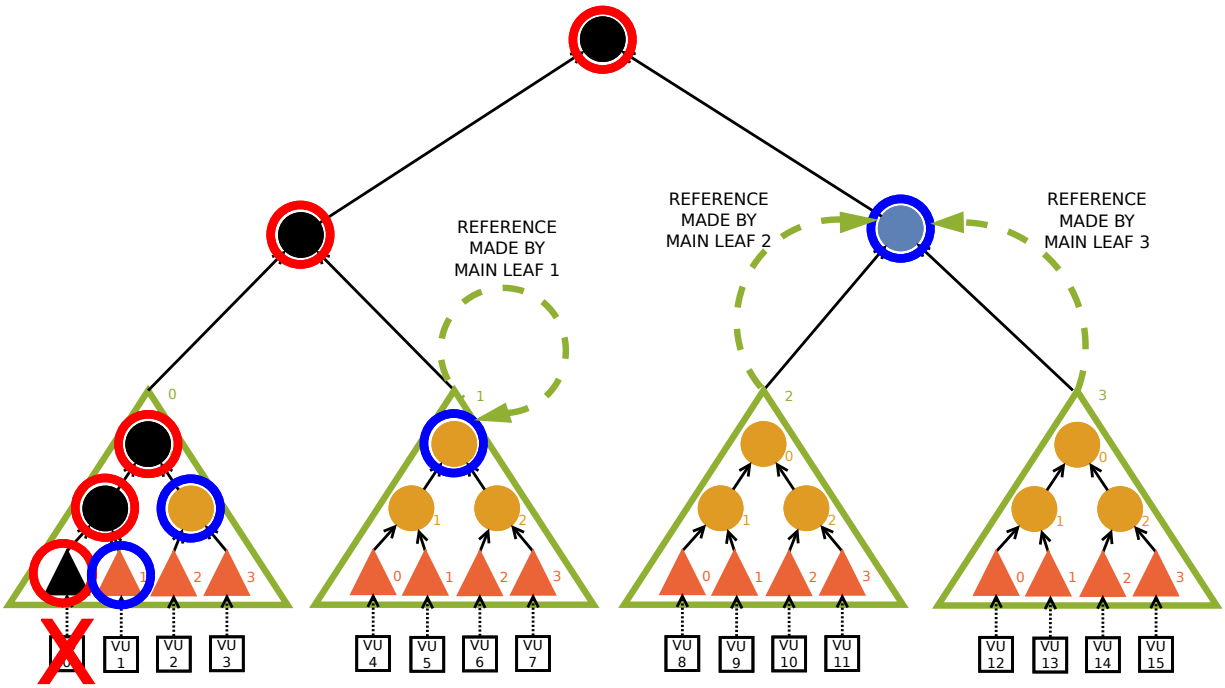


Figure 4.13: Constant time search of refreshment messages using references from the main leafs to the correspondent refreshment messages.

---

an useful refreshment message).

Figure 4.13 shows how the use of such division allows, with 3 references, to have a constant search that covers 12 VUs, the rightmost ones that in groups of 4, share the same ν^h (most significative bits). The shared ν^h is the index of the main leaf with the reference to the needed refreshment messages, so, 2 out of 4 messages are immediately available to 12 possible neighbors just by directly accessing to 3 out of 4 well-defined positions. For the 3 leftmost leafs, it can be used the same strategy, or the previously described search across all the nodes of the inner tree (not the main tree!).

4.3.2 Sync messages

The sync messages are exchanged in 2 different directions, from VUs to KDC and vice-versa, as requests and replies, respectively.

From VUs to KDC, request messages are made out of:

- VU identification, ν;
- last time interval, t, know by the VU (the current state of P_ν);

- auxiliary key, α_ν, encrypted with the public key of the KDC (this is only present after the reception of a reset message, see Section 4.2.2.1);
- MAC of the previous content computed with the last exclusive key, ϵ_ν;
- MAC of the previous content computed with the last routing key, $\rho(t)$.

While the replies, are made out of:

- the ordered subset P_ν encrypted with ϵ_ν or α_ν (see Section 4.2.2.1);
- time interval of the encrypted P_ν;
- control information;
- KDC signature.

Chapter 5

Implementation

I would find it harder if I had to spend all my time learning how to use somebody else's routines.

— Donald Knuth, *Coders at Work*

We built `loop` (loop over orderly phases) an interactive simulator for both OBUs and RSUs, with a graphically visualization, that communicates with a KDC, according to a realistic model of a VANET. We also built a prototype of the KDC and prepared it to accommodate up to 2^{16} (65536) VUs.

The simulator was fed with 24 hours of real data samples, collected from Veniam's VANET on Porto city, containing 396 OBUs and 50 RSUs. All VUs send periodic routing messages (beacons) in intervals of 100 milliseconds, informing all the neighbors about their current state. The data samples have the logged state of all nodes in periods of 2 seconds, including their geographic position and a list of the neighbors listened during the 2 seconds, along with the correspondent beacon signal strength. The area used by OBUs was about 21.1 by 18.5 kilometers.

Based on the lists of the neighbors per VU, we simulated the transmission and reception of beacons in periods of 100 milliseconds and enforced their authentication. Furthermore, we simulated the connections made by the RSUs to the KDC and appended to the beacons, on a needed basis, the refreshment messages produced by the KDC as well as the requests for sync messages produced by OBUs and the respective replies produced by KDC.

5.1 Definition of parameters

To accommodate up to 2^{16} VUs we have the same number of leafs in the KDC, and both main and inner trees have $l = 256$ leafs (see Equation 4.11). The set of distribution keys S has $n = 131071$ elements (see Equation 4.9) and each VU stores $e = 17$ of those elements on the correspondent subset P_ν (see Equation 4.10).

With $l = 256$ the index (identification) of each VU has 2 bytes and the index of the distribution keys has from 1 to 4 bytes (see Equation 4.13): 1 byte to the branch nodes of the main tree; 3 bytes to the branch nodes of the inner trees; 4 bytes to the leafs of the inner trees (where VUs are associated).

For the maximum number of VUs, in the worst case, the number of messages produced to exclude multiple VUs at once is shown in Figure 5.1. Although the need for exclusion of thousands VUs at once is theoretically possible, it is not expected in our scenario, so, for a reasonable number of compromised VUs, up to 256 out of 2^{16} at once, a more detailed information is shown in Figure 5.2. The best case occurs when all the VUs to exclude have successive indexes.

The exclusion of 256 VUs in the best case, with $l = 256$, is the exclusion of all VUs on the same inner tree, and so, to update the remaining ones are needed 8 messages associated to the 8 branch nodes of the main tree; for the worst case, is the exclusion of 1 VU on each of the 256 inner trees, and so, to update the remaining ones are needed 8 messages associated to the 8 branch nodes on each of the 256 inner trees; to update half of the VANET, when 1 VU is compromised on each of the 256 inner trees, 1 message will update half of the remaining VUs on each of the inner trees, and so, 256 messages will update half of the VANET.

With a KDC able to handle up to 2^{16} VUs, on our practical case, with 446 VUs, we have $m = 128$ (see Equation 4.17), which means that each inner trees (with $l = 256$ leafs) has associated a maximum of 2 VUs, more precisely, 190 inners trees have 2 VUs and 66 have 1 VU ($190 \cdot 2 + 66 = 446$).

The RSUs check for new information in the KDC in periods of 1 minute. The OBUs, when not able to update their cryptographic material to the last state of the VANET based on their neighbors, wait up to 5 minutes to start using the fallback method based on the RSUs (see Section 4.1.3). The replies from KDC take between 400 milliseconds and 4 seconds to be sent back to RSUs.

The size of the historical cache (see Section 4.1.2) is dynamically defined by the KDC on each new refreshment message, in the simulator we defined the limits between 1 and

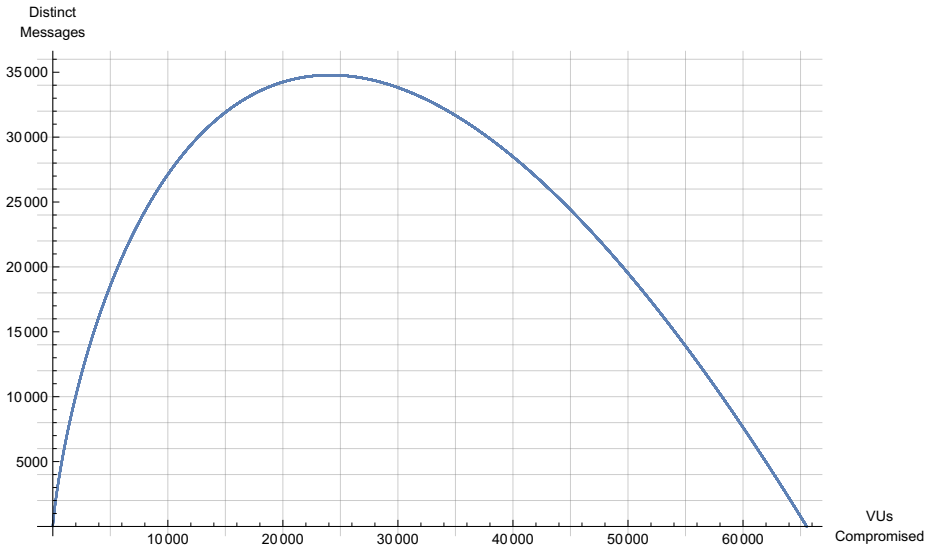


Figure 5.1: Maximum number of messages produced by KDC to exclude VUs at once

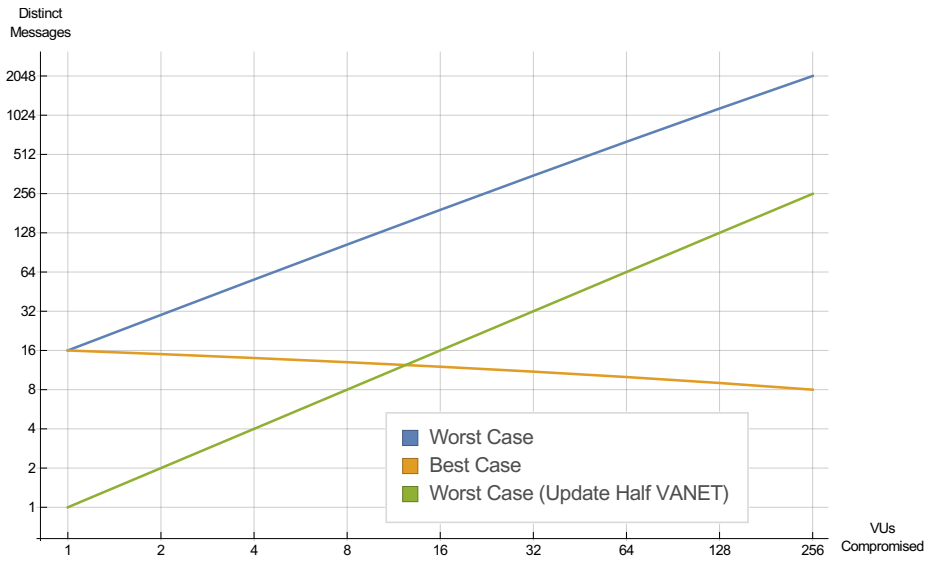


Figure 5.2: Messages produced by the KDC to exclude up to 256 VUs at once

32. The size of the out of order buffer for the refreshment messages listen by VUs is 8.

For the symmetric cipher we chose the AES128, so the symmetric keys have 16 bytes. We assumed the use of HMAC-MD5 as the MAC function, with an output of 16 bytes, and the use of ECDSA, as the asymmetric signature algorithm, more precisely ECDSA-224 with a signature of 56 bytes and a public key with 28 bytes. These technologies were chosen based on the security currently provided, $\geq$ 112 bits of security strength [8], and the current recommendations for the near future [7], up to 2030.

With the best expectations being to use the protocol without any update up to 2030, we use 3 bytes to define the time intervals (the number of minutes up to 2030 is less than 2^{24}).

5.1.1 Size of the messages

Based on the previously defined values, the routing beacons (see Section 4.3) will have extra 20 bytes, plus the size of the optional refreshment or sync messages:

- size of the previous routing information;
- 3 bytes for the time interval;
- 1 byte to advertise the state of the historical cache (7 bits left);
- 16 bytes for the MAC.

The refreshment messages (see Section 4.3.1) have from 80 to 83 bytes:

- 16 bytes for the encrypted value of r;
- 1 to 4 bytes to the identification of the key (see Section 4.2.1);
- 3 bytes for the time interval;
- 4 bytes for the control information: 2 for the number of distinct messages, 1 for the historical cache, 1 for the basal refreshment rate;
- 56 for the KDC signature.

The requests for sync messages sent by the OBUs (see Section 4.3.2) have 37 or 293 bytes:

- 2 bytes for the identification of the VU;

- 3 bytes for the time interval;
- 256 bytes for the encrypted α (only when recovering from the disclosure of all keys in the KDC);
- 16 bytes for the MAC computed with ϵ;
- 16 bytes for the MAC computed with ρ.

While the replies of sync messages sent by the KDC have 338 bytes:

- 272 ($16 \cdot 17$) for the encrypted subset of P_ν;
- 3 bytes for the time interval of P_ν;
- 4 bytes for the control information: 2 for the number of distinct messages, 1 for the historical cache, 1 for the basal refreshment rate;
- 56 for the KDC signature.

5.2 Simulator

We built `loop` (loop over orderly phases[1]), a simulator using C++11[2] and OpenCV[3] to support a visualization of the simulation.

We implemented all the cryptographic operations related with the distribution keys (see Section 4.2): the VUs can receive beacons (see Section 4.3) in periods of 100 milliseconds, and react according to the content of such beacons. Since we were not interested in modeling the impact of fake information in the VANET, we did not implement the asymmetric operations related with the signature of the KDC neither did we included MACs on the transmitted beacons (we only verify the information of time, and assume that a valid MAC exists according to this information). Nevertheless, we considered the time that asymmetric operations need and limited them in time, according to a realistic scenario. At the end of each simulation all the symmetric keys stored in each VU are validated against the values stored in KDC.

The simulator has 3 different threads of execution: 1 for the visualization; 1 for the interaction with a user (see Section 4.1.4); and 1 for the simulation itself. The visualization

[1] The name nicely describes what it does!
[2] https://isocpp.org/wiki/faq/cpp11
[3] http://opencv.org/

and the interaction only offer extra output and input features, and may not be needed for all the use cases. Thus, they can be activated and deactivated as needed. The only thread that is always running is the simulation thread.

The kernel of the simulation is a loop over different phases, repetitively over the time, where each of the phases is responsible for a well defined task, and according to this, contact with the KDC (see Section 5.3.2). The granularity on the time being simulated is 1 millisecond and the total time of the simulation (in this case, 24 hours) is divided into 2 types of periods: macro periods and inner periods.

The macro period is associated to the periodicity of the input data. On each macro period we have different information (such as position and number of neighbors) for the VUs. The inner period is associated to the period of the transmission and reception of beacons in the routing protocol. We worked with macro periods of 2 seconds and inner periods of 100 milliseconds.

Along with each simulation is produced a log file, that contains the important parameters used for its configuration, so that the experience can be identified and easily reproduced, and the important results collected during the simulation for additional analysis with external tools.

5.2.1 Phases of the simulation

On the simulation, we keep a distinct separation between the state of the simulator and the state of the VUs. By state of simulator we mean all the non-realistic state that only exists for convenience of the simulation e.g. the graphical interface. On the other hand, the state of the VUs represents the model of the real operations found in VANET e.g. transmission and reception of routing beacons.

Generically, each phase can have 4 different temporal points of execution, that can be used to better modeling a specific task:

1. once on macro period start
2. once on inner period start (never used!)
3. on macro period
4. on inner period

The first 2 points of execution are associated to the state of the simulator, thus, the state of the VUs is only accessible for reading operations and not allowed to be modified.

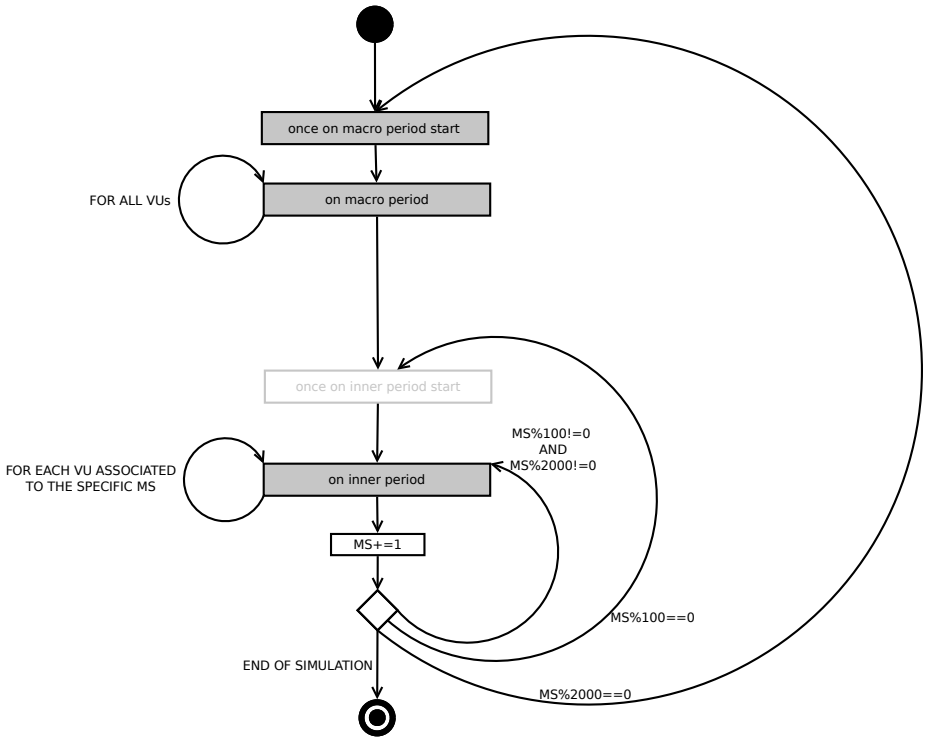


Figure 5.3: Points of execution of `loop` (with `"%"` representing the modulo operation and `"=="` and `"!="` boolean tests). The unused point of execution is shown in light gray. The macro period is 2000 milliseconds and the inner period 100 milliseconds.

The last 2 points of execution are VU's specific and executed for each VU with permission to modify its state (as well as permission to read all the other VUs' public state).

VUs are randomly distributed to one specific millisecond of the inner period (between 0 and 99) and keep that same position in time until the end of the simulation. The simulator jumps from millisecond in millisecond and for each phase, on a needed basis, it executes the code associated to the start of the periods as well as the code for VUs associated to that millisecond. The different points of execution are shown in Figure 5.3.

To guaranty the determinism of the execution of the phases, if two or more VUs are associated to the same millisecond of the simulation, running each of the phases on that millisecond should yield the same result independently of the order that operations are applied to each VU.

The generic phase is implemented on the simulator under a C++ class. Phases with a well defined task are classes with a relation of inheritance to the generic phase. After the initial setup, where VUs are associated to a millisecond of the inner period and a list of phases is created, the simulation runs based on successive calls to the members of the

```
for ms in simulation; do

    for p in phases; do

        if starting a macro period
            if p->hasWorkOnStartOfMacroPeriod()
                p->OnceOnMacroPeriodStart(&simulation data, ms, const &all VUs, ...)

        if starting a inner period
            if p->hasWorkOnStartOfInnerPeriod()
                p->OnceOnInnerPeriodStart(&simulation data, ms, const &all VUs, ...)


        for v in VUs associated to ms; do

            if first time on macro period
                if p->hasWorkOnMacroPeriod()
                    p->onMacroPeriod(&simulation data, ms, &v, const &all VUs, ...)

            if first time on inner period
                if p->hasWorkOnInnerPeriod()
                    p->onInnerPeriod(&simulation data, ms, &v, const &all VUs, ...)
        done

    done

done
```

Figure 5.4: Pseudo code of `loop`

classes that represent the points of execution (see Figure 5.4).

To properly model and simulate the important characteristics of the VANET, we identified 3 distinct tasks and implemented the correspondent C++ classes to be executed in loop during the simulation time. Those tasks are, in order:

1. Apply the effect of time (Proc)
2. Transmit information (Tx)
3. Receive information (Rx)

The different tasks have influence among each others, as described in Figure 5.5. In the figure, an arrow between tasks A $\rightarrow$ B, represents a relation of influence of A on B. The labels are used to identify the cases in the following explanations. Left to right arrows ($\rightarrow$) represent an immediate influence visible for the next phase, right to left ones ($\leftarrow$) represent delayed influence, only visible 1 or more inner periods later. Black arrows represent internal influence along the same VU, red ones represent influence across multiple VUs, and green ones represent the optional interaction (IN) and the optional visualization (OUT).

5.2.1.1 Apply the effect of time

The first phase has 2 points of execution, both associated to the macro periods, one to update the state of the VUs and another one to update the state of the simulator.

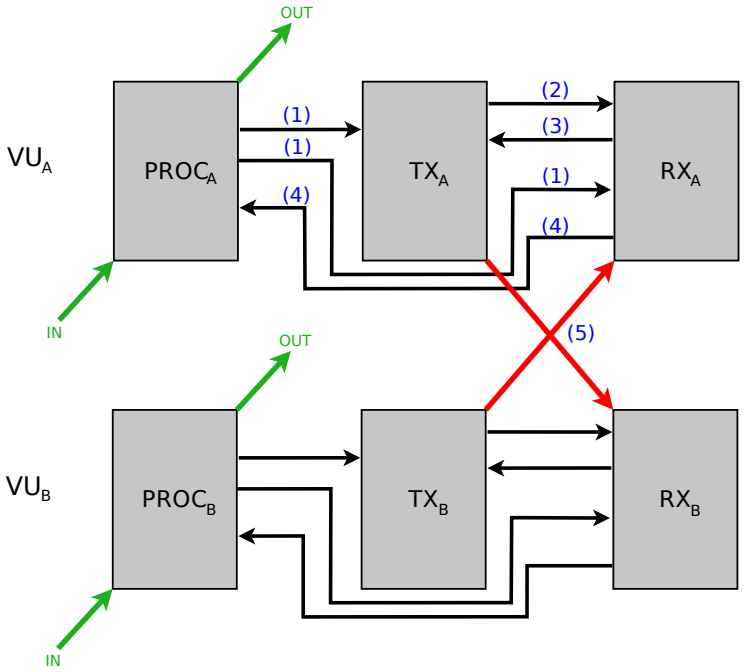


Figure 5.5: Influence of phases among each others

Updating the state of VUs consists on set them as online or offline (1). When the VUs are online, the position and the list of neighbors is also updated on a needed basis. Updating the state of the simulation consists on handling all the counters in use for logging purposes as well as to prepare a new visualization for the specific macro period. The visualization is dependent of the previous messages received (4).

5.2.1.2 Transmit information

The second phase has 2 points of execution for VUs, for both macro and inner period. In the inner periods of 100 milliseconds we model the transmission of routing beacons by all VUs. The other type of transmissions, the much less frequent requests for new information made by RSUs to KDC, are modeled on a needed basis in the macro periods.

The transmission of beacons is modeled making information available to all the other VUs, so, when needed, such information can be read (5). All the transmissions of a VU are influenced by the previous receptions (3). When the requests made by RSUs to KDC have a reply, such reply is received and stored, and all the effect of the received data on the simulation are handled by Rx (2).

5.2.1.3 Receive information

The third phase has 1 point of execution for VUs, in the inner period. At this point we model the reception of routing beacons and also, in case of RSUs, the reception of the data from the KDC (2). The reception of beacons is modeled by reading the messages on the correspondent neighbors at that time (5). The beacons read from neighbors represent the beacons received in the last 100 milliseconds, on a fifo order.

The received information handled on Rx, beacons from others (5) and KDC replies in the case of RSUs (2), may or may not contain signed refreshment or sync messages, that need to be validated with the KDC public key before being used. Since such validation is a time consuming operation, we limited its number in the inner periods (corresponding to 100 milliseconds) and delayed to the next inner period the missed validations. Based on the initial tests done on VUs' hardware, where each validation of an ECDSA signature takes approximately 6 milliseconds, we limited the number of validations (to a number randomly selected) between 4 and 7 per inner period.

The refreshment and the sync messages are used at this point by VUs to update the subset of distribution keys P (and the routing key ρ). After being validated, the received refreshment messages are cached (see Section 4.1.2) and will influence the future transmissions (3). The update on the subset of distribution keys P on VUs will influence the next visualizations (4).

Based on the received beacons we build, update and compare two routing tables according to SB2RP (see Section 2.4): one based on all the beacons and another one based only on the authenticated beacons.

In Figure 5.5, both VU_A and VU_B can be seen as associated to the same millisecond. The influence across multiple VUs needs to take in account the transmission time of the packet, so, if they are transmitting at the same time, this mutual influence is not instantaneous, as the red (left to right) diagonals arrows are suggesting.

Taking as reference that the transmission of 2KB needs approximately 1 millisecond when the physical channel (of 16Mb) is otherwise idle, we assume that the beacons will be transmitted within 1 inner period (100 milliseconds) under any circumstances, and so, in our simulation the instantaneous transmission of information have the same effect as a delayed transmission (with a delay smaller that 100 milliseconds) because the beacon will only be processed after the other beacons produced sooner (up to 100 milliseconds sooner). Delays greater than 100 milliseconds will be analyzed as communications over a lossy channel.

```
783,1423764002,41151180,-8609789,35,21,0,707,1,815,5,690,25,815,36,0,36,815,36,255,5,783,1423764004,41
151240,-8609789,33,14,0,99,1,815,9,690,26,815,32,0,32,815,32,255,10,512,2,799,1,0,1,799,1,783,14237640
06,41151272,-8609789,45,6,0,0,1,815,8,690,29,815,33,0,33,815,33,512,4,799,1,799,1,799,1,783,1423764008
,41151276,-8609533,45,0,0,185,1,815,5,690,31,815,32,0,32,815,32,512,10,783,1423764010,41151276,-860953
3,45,0,0,1412,1,815,5,690,28,815,33,0,33,815,33,512,10,783,1423764012,41151272,-8609533,45,0,0,939,1,8
15,4,690,29,815,34,0,34,815,34,783,1423764014,41151272,-8609533,45,0,0,99,1,815,5,690,29,815,34,0,34,8
15,34,512,2,783,1423764016,41151272,-8609533,45,0,0,60,1,815,5,690,29,815,34,0,34,815,34,512,4,783,142
3764018,41151276,-8609533,45,0,0,185,1,815,5,690,25,815,33,0,33,815,33,255,0,783,1423764020,41151276,-
8609533,45,0,0,1511,1,815,6,690,25,815,33,0,33,815,33,255,0,512,7,783,1423764022,41151292,-8609533,45,
4,300,945,1,815,8,690,24,815,33,0,33,815,33,799,2,512,2,512,11,799,2,783,1423764024,41151352,-8609533,
40,10,0,99,1,815,8,690,22,815,31,0,31,815,31,799,2,512,2,512,12,799,2,783,1423764026,41151420,-8609533
,29,15,0,0,1,815,5,690,17,815,28,0,28,815,28,512,11,783,1423764028,41151528,-8609533,346,19,130,389,1,
-1,5,690,16,815,12,0,12,815,12,512,10,783,1423764030,41151628,-8609533,324,23,485,1517,1,-1,5,690,15,8
15,10,0,10,815,10,512,9,783,1423764032,41151704,-8609789,312,21,269,0,1,-1,1,690,10,783,1423764034,411
51752,-8609789,304,17,900,1345,1,-1,5,690,9,512,2,815,3,815,3,815,3,783,1423764036,41151800,-8609789,2
```

Figure 5.6: Sample of data collected from VANET

5.2.2 Input data

The simulator uses as input different data-sets with 24 hours built based on real data collected for the different VUs. The collected data is made out of multiple text files for each VU (1 file for each continuous period of activity). Each file has a single line separated by commas (see Figure 5.6 for a sample), containing multiple groups of the following information:

- An unique identification of the VU;
- A global time stamp in seconds (linux epoch);
- A value representing latitude;
- A value representing longitude;
- 6 other values not used by the simulator, e.g. heading;
- The number of neighbors listened;
- A list of pairs of listened neighbors: identification, strength of the signal.

Without any known reason, some data entries where inconsistent with the rest of the data-set, either by having multiple times the same time stamp, by having successive positions of latitude and longitude that represent a movement above normal speed limits, or by having in the list of neighbors either an offline VU or one that is too far way.

We wrote a perl script to convert the raw data, excluding any inconsistent entry, limiting it to one distinct time stamp per VU, with a maximum speed of 180 kilometers per hour (based on two consecutive time stamps and respective positions) and a maximum distance

```
1
0 67312 87296 1 0 23 4
2 67308 87296 3 3 326 7 430 7 443 31 326 430 443
4 67296 87552 1 1 443 25 443
6 67252 87552 2 2 326 8 443 25 326 443
8 67232 87808 1 1 443 26 443
*
508 61616 105216 2 2 233 20 258 6 233 258
510 61500 105216 2 1 233 14 358 9 233
512 61356 105216 2 1 233 8 358 17 233
514 61276 105216 0 0
516 61020 104960 0 0
518 60840 104704 0 0
```

Figure 5.7: Sample of data used in `loop`

for the wireless communications. We built 2 data-sets with communication limited to 1 kilometer and 100 meters.

When the filtering of the data let us without any data at all for a specific time, we assumed that the VU was offline. A continuous offline period of arbitrary length is identified by a line with a single asterisk. All the used values were transformed and normalized: the VUs get continuous identification numbers starting at 0; latitude and longitude are transformed into cartesian coordinates. At the end, we have a single text file for each VU (see Figure 5.7 for a sample), with a header in the first line and a different line for each time stamp. The header only has a field to distinguish between RSUs from OBUs, the next lines can be an asterisk that represent an offline period or a bundle of information for each time stamp. The information contained in the bundle is:

- A normalized time stamp;
- cartesian coordinates;
- The number of neighbors listened;
- The number of neighbors that listen to the VU;
- A list of pairs of listened neighbors: identification, strength of the signal;
- A list of identifications of neighbors that listen the VU.

Besides being filtered, the produced data-set has the advantage of being perfectly human-readable. Envisioning that the simulation would result in complex interaction between VUs, we considered that an human-readable data-set would be an advantage to better understand what was happening and easily discard (or discover!) errors associated to the simulation process.

5.2.3 Configuration and interaction

We used two types of configuration: statically in the code and by command line arguments. All the values that have any type of control over the simulator and are possible to be modified were initially defined as C++ constants in a source file, `config.hpp`, inside a distinct namespace, `config`. During the development and evaluation all the constants that were being modify, implying the recompilation of the code, were converted into command line arguments, with default values equal to the previously defined constants values.

We ended up with 33 values statically defined in `config.hpp` and 12 possible command line arguments. On `config.hpp` we have parameters like the size of the symmetric keys and the dimensions of the visualization windows. The command line arguments are:

`-b` The base directory where multiple data-sets may be placed.

`-c` The file with identifications of the VUs (camouflage). The file contains a list of identifications, unique numbers between 0 and $2^{16} - 1$, as many as the number of the VUs in the simulation. This allows the distribution of the VUs to different leafs of the different inner trees (see Section 4.2.3).

`-d` The name of the data-set to use.

`-G` A flag to disable the graphical visualization.

`-h` The size of historical cache (see Section 4.1.2).

`-I` A file with commands to run in simulator (disables the interactive mode with the user).

`-k` The address and optionally the port of the KDC.

`-l` The name of the output log file.

`-m` The percentage of beacons to mute on the receptions' phase of all VUs (to simulate a lossy environment).

`-r` The period for the release of a new refreshments message. This period is not considered when exclusions are performed as an order to exclude 1 or more VUs will immediately release to the VANET the correspondent refreshment messages.

`-s` A seed to the C++ pseuso-random number generator. This allow us to test the same configurations with all the (pseudo-)randomly selected parameters modified, e.g. the

```
# define the basal refreshment rate after 16 hours
57600000 brr 50

# compromise some VUs at the same time
57600000 compromise 1 4097 8193 12289 16385 20481 24577
```

Figure 5.8: Sample of a file with commands for `loop`

distribution of the VUs across the inner period, the selection of the refreshment messages by the VUs and the number of asymmetric signature validations that can be done by inner periods.

`-u` An early stop to the simulation which will run up to the given number of milliseconds and not for all the time entries contained in the data-set.

More than an initial configuration, we allow the interaction with the simulation by one of two options: an interactive mode or a file with a predefined list of commands. In the interactive mode the commands are given to the simulator in a terminal-like interface, while in a file the commands are available, one per line, with the time (milliseconds) to be executed pre-appended (see Figure 5.8 for a sample). The commands available for any of the cases are:

`brr` to indicate to the KDC a new basal refreshment to be sent along the refreshment and sync messages. Takes a number (permillage) between 0 and 1000 (see Section 4.1.2).

`compromise` to indicate to the KDC that one of more VUs are compromised and need to be excluded. Takes a list of identifications (according to the `-c` option).

`info` to print the state of the simulation. Prints the current millisecond, the time of the simulation and the time spent, both in dd:hh:mm:ss format, and the speedup of the simulation.

`map` to change the map view (see Section 5.2.4). Takes the name of the view (`bigbrother` or `naive`) to shown up.

`pause` to pause and re-initiate the simulation.

`q` to print a random quote related with computer science and technology (useful while waiting for the last seconds of the simulation).

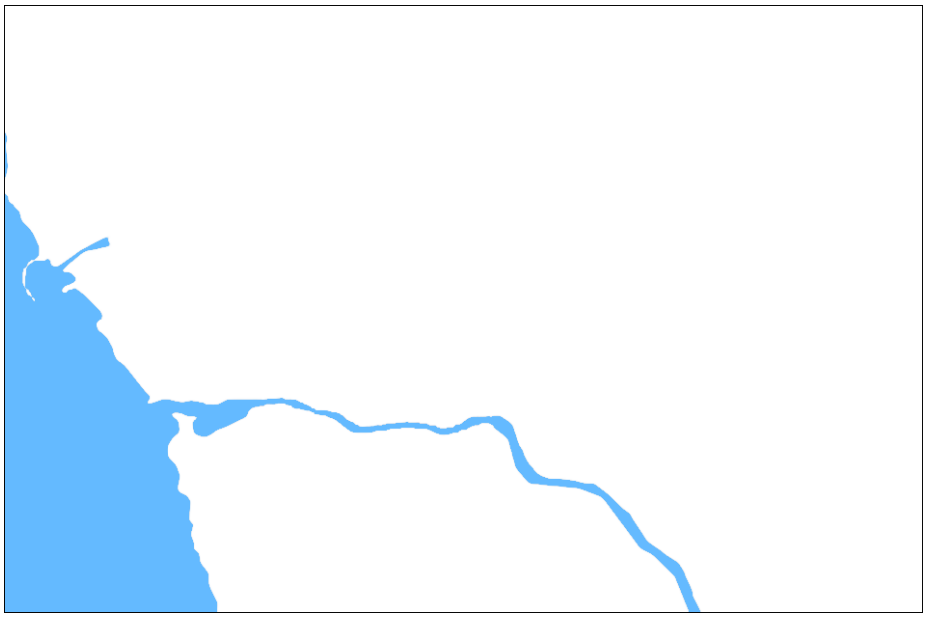


Figure 5.9: Base image used for the visualizations

`reset-keys` to simulate the disclosure of all the distribution keys in the KDC and initiate the recovering process (see Section 4.2.2.1).

`resurrect` to indicate to the KDC that any of the previously compromised VUs are not compromised anymore (see Section 4.2.2). Takes a list of identifications (according to the `-c` option).

5.2.4 Visualization

The visualization let us follow the simulation and easily understand what is happening at which moment. There are 2 different windows: 1 with a map showing the state of VUs and another one with some temporal parameters plotted.

The visualization is associated to the area that VUs use during the simulation. A base image to the visualization area can be initially loaded, to help the identification of the map. In our case, since the area of the simulation is the Porto city, we load a map with the atlantic ocean, the Douro river and Leixões harbor shown in blue, Figure 5.9.

At the beginning of the simulation, the data of all VUs is loaded into the simulator. During this process, the base image is populated with the cartesian coordinates that VUs have along the time. These coordinates will end up showing all the paths used by the VUs (see Figure 5.10).

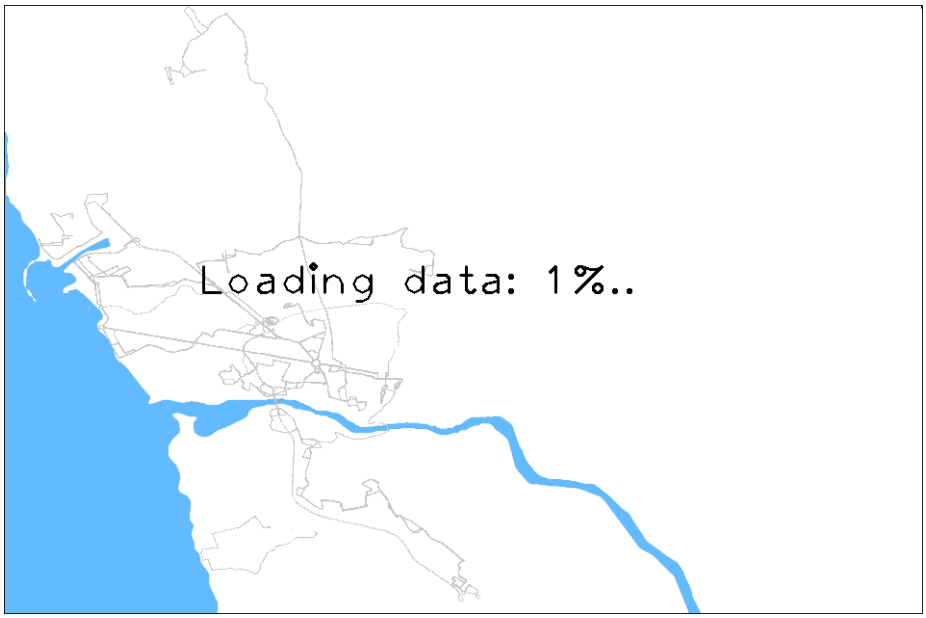


Figure 5.10: Loading the data for the simulation

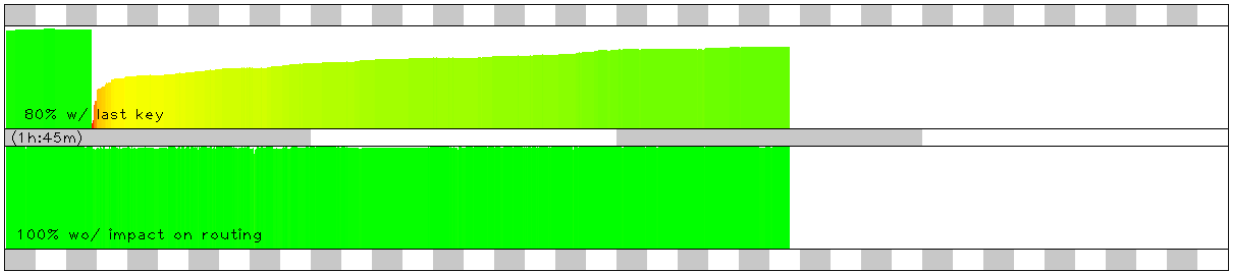


Figure 5.11: Parameters plotted during the simulation

After the loading of the data, the simulation starts, and a second window appear. This window shows the time of the simulation, and the current state regarding to 2 different variables: the percentage of active VUs with the last routing key and the percentage of routes not affected at all by the secure routing when compared with the insecure one.

Those two variables are shown as in Figure 5.11. The reddest parts are closer to 0 while the green ones are closest 100. The smaller gray bars represent 1 minute of the simulation time while the bigger ones represent 10 minutes.

During the simulation the map has 2 different views available, possible to selected with the `map` command (see Section 5.2.3). In both, the RSUs are shown in violet. The OBUs may change the color in which they are shown, according their state and the selected view. The map views are:

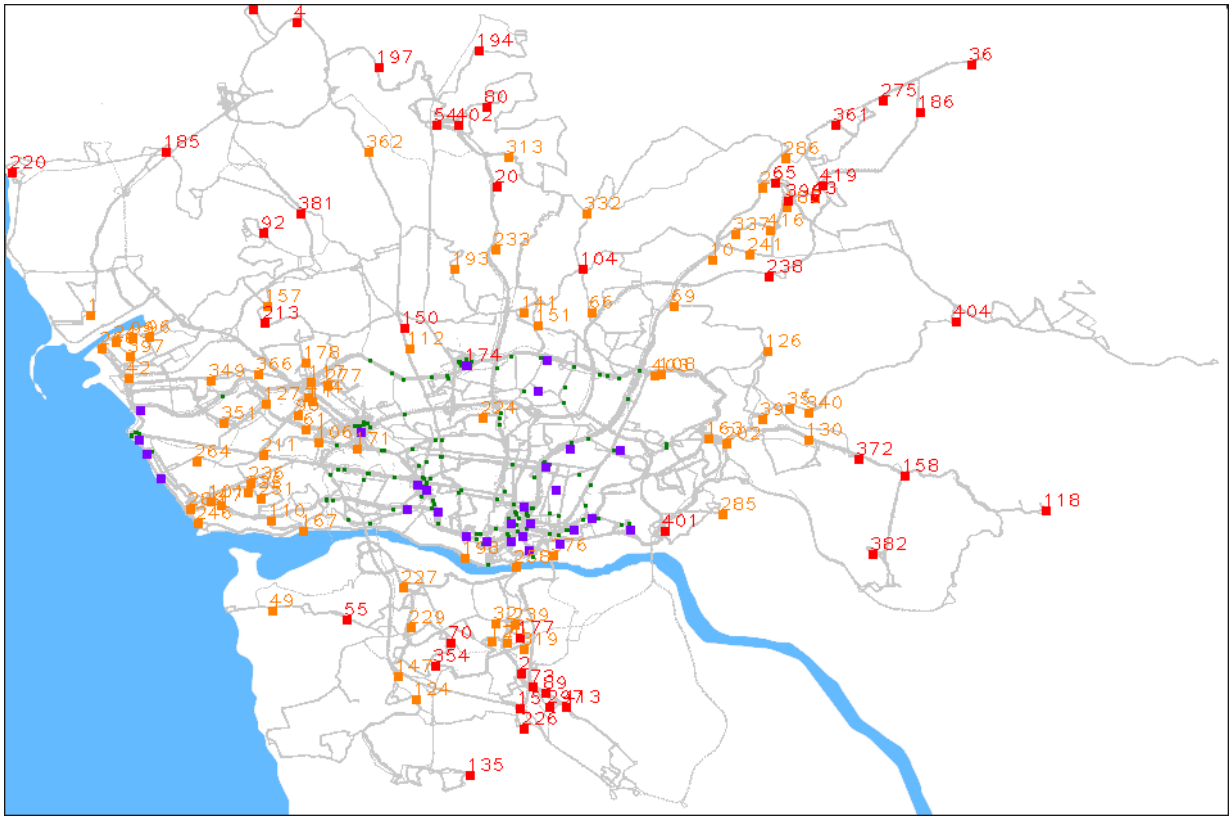


Figure 5.12: Map showing the big brother view

The big brother view: Selected with `map bigbrother`, shows the active OBUs in 3 possible colors, green, orange and red, according to the state of their cryptographic material when compared with the last refreshment message sent to the VANET (see Figure 5.12). In this view it is possible to analyze the epidemic propagation of the refreshment messages and the impact that RSUs have to the closest OBUs. The OBUs that have received the last refreshment message are shown in green. The other OBUs are shown in orange if their cryptographic material is within the limit of the historical cache (so they can be updated by some neighbor) or red if they are out of the range of the historical cache of the updated (green) OBUs and may need to use the RSUs to request an update message from the KDC or may possible receive a refreshment message from an orange one. The OBUs without the last refreshment message are highlighted: they are bigger and present their identification. This helps to diagnose why a particular OBU is not updated.

The naive view: Selected with `map naive`, shows the active OBUs in 4 possible colors, black, green, orange and red, according to the state of their cryptographic material when compared with their neighbors (see Figure 5.13). In this view it possible to analyze the evolution of the relations created between neighbors. The isolated OBUs, the ones that are not listened nor listen any other, are shown in black. The OBUs that listen one or more neighbors all with the cryptographic material in the same state (so they can produce and validate beacons among each other) are shown in green. The OBUs that are listening neighbors with cryptographic material in a different state are shown in orange or red, if they are the more or the less updated ones, respectively. Along with the red OBUs, is shown a red line connecting them to any neighbor that has a more updated cryptographic material. When listening or listened by neighbors with cryptographic material in a different state, the OBUs are bigger and showing their identification. This map highlights the situation where routing beacons may be discarded due to discrepancies on the keys being used by neighbor OBUs.

5.3 KDC

We built a prototype of the KDC using Python3[4] and SQLite3[5] to persistently store all the data in use. The implementation is based on a TCP/IP server, following the threaded

[4]https://www.python.org/
[5]https://sqlite.org/

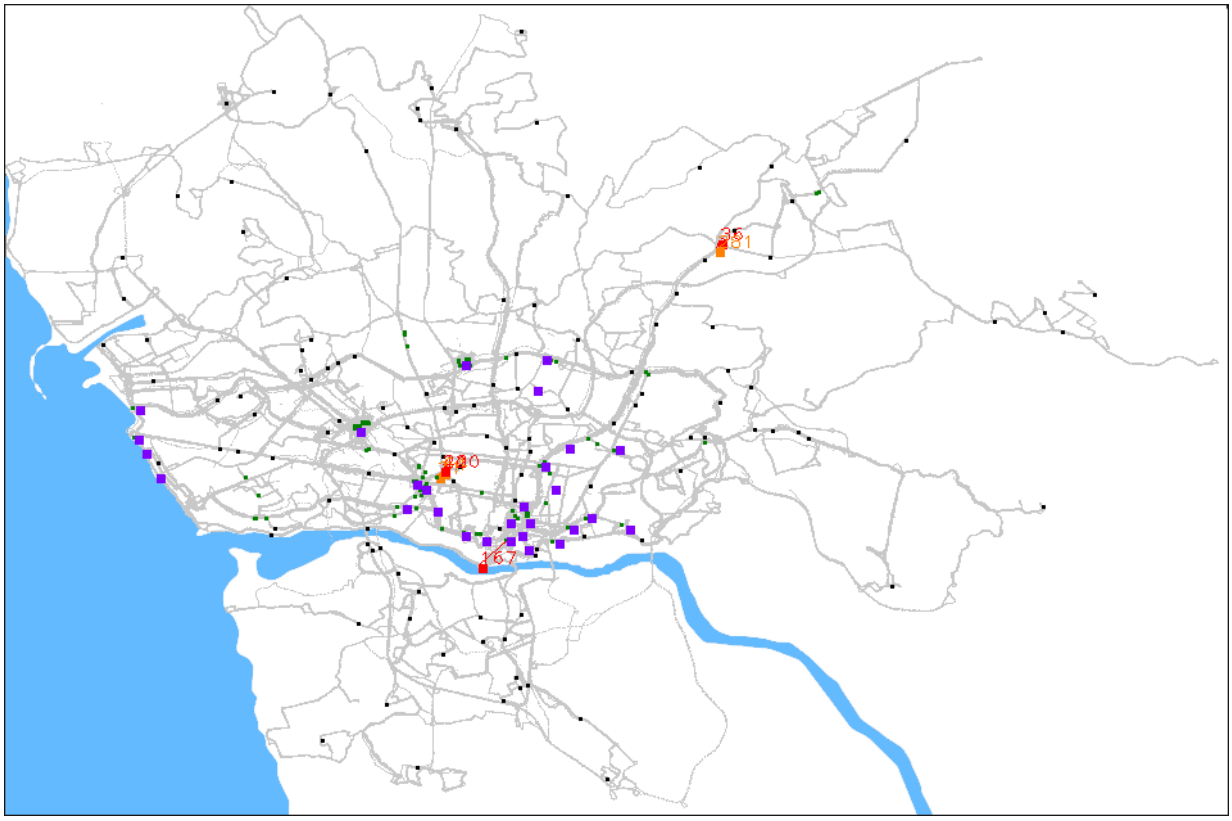


Figure 5.13: Map showing the naive view

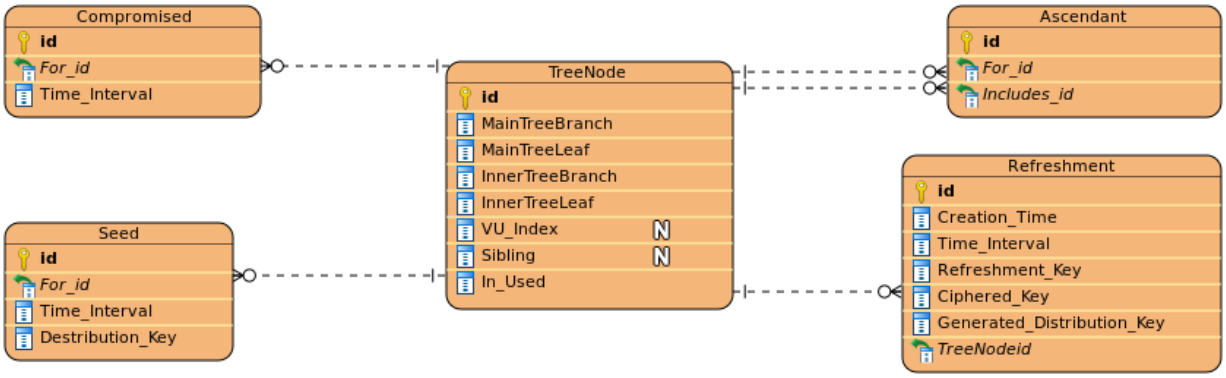


Figure 5.14: Database model of the KDC

model, that receives and handles the requests made by the simulator as they were made by the real RSUs. Since the KDC has to interact with the simulator, some facilities were included for the start and the end of each simulation.

We mention the KDC as a prototype because the development was from the beginning focused on testing and evaluating the different new ideas proposed. We did not try to achieve any kind of performance that would be possible with the use of a non-interpreted language or a database with support for a higher level of concurrency.

5.3.1 Database

Although being a prototype, we did not relax the approach related with the data manipulation, so, the running process of the KDC holds no context of the simulation, and all the mutable state is stored and retrieved as needed from the database.

With built a data model that falls into the star scheme, shown in Figure 5.14. On the center of the model we have the `TreeNode` table that is referenced by all the other tables. We focus on speed up and simplification of the read operations, at the cost of some redundancy and denormalization, e.g. the `Refreshment` table may have multiple entries, one per refreshment, with the same stored value of `Creation_Time`, `Time_Interval` and `Refreshment_Key`.

The database has 5 tables:

`TreeNode` stores all the nodes of the trees, for the leafs of the inner trees, the identification of the associated VU is also stored. All the nodes are created during the initialization of the database and the only mutable column is the one used to mark the node as used or not used.

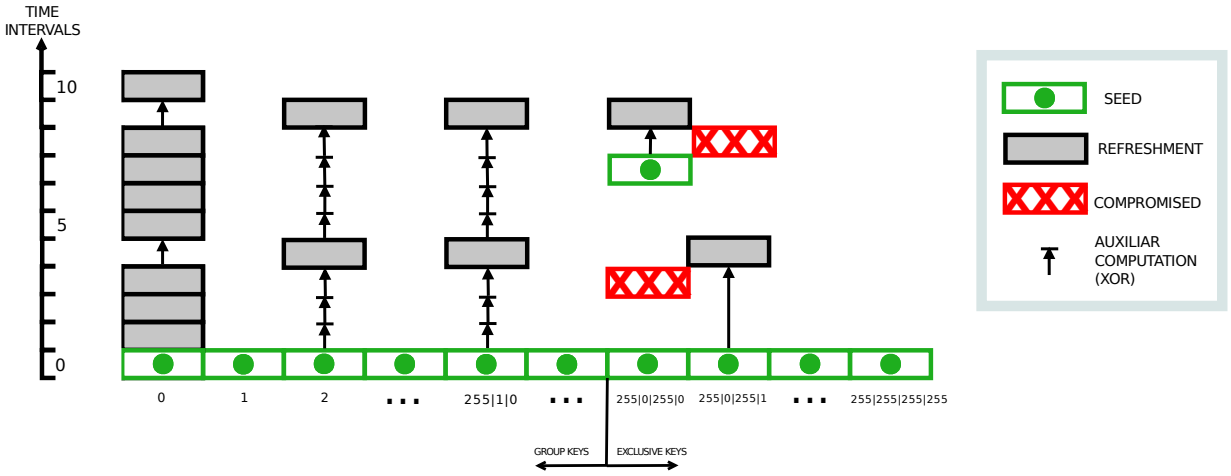


Figure 5.15: Relation between entries of the multiple tables

`Ascendant` has the information about all ascendants for all the VUs. This table is created during the initialization of the database and not modified anymore.

`Refreshment` stores the information used to built each of the refreshments messages sent to VANET. For each time interval, are inserted the same number of entries as the number of refreshments sent to VANET.

`Compromised` stores the compromised VUs in a time interval. A VU, in `TreeNode`, is considered compromised if there is an entry in this table, and there is no entry on `Seed` with a greater time interval.

`Seed` contains the initial distribution key for each of the nodes of the trees. During the initialization of the database ($t = 0$) is created one entry for each of the nodes presented in `TreeNode`. New entries can be added (for $t \neq 0$) to re-include any of the VUs previously compromised. All the seeds need to be explicit loaded in the VUs.

An example of the relation between entries of the multiple tables is shown in Figure 5.15, where each box represent a stored entry in the correspondent table. At the bottom of the image we have all the initial seeds for the $n = 131071$ elements of the distribution keys. The identification of the nodes follows the division in the main and the inner trees (see Section 4.2.1).

The transition to $t = 1$, $t = 2$ and $t = 3$ is done with a single refreshment message at the time. Each refreshment message contains the correspondent refreshment key $r(t)$ (random value), the encrypted value (with the previous distribution key) that is sent to

the VANET, and the newly resultant distribution key (the previous distribution key XOR with $r(t)$).

During the interval $t = 3$, the VU_0 is pinpointed as compromised and this immediately triggers the production of refreshment messages to update all the VANET but VU_0. Assuming that all the 2^{16} VUs are active, according to Figure 5.2 are generated 16 refreshment messages. These 16 messages contain the same refreshment key $r(4)$, encrypted with different distribution keys. To obtain the needed distribution keys, some auxiliary computations (XORs) are done, based on the stored refreshment keys $r(1)$, $r(2)$ and $r(3)$ and the stored seeds for $t = 0$. With all the 16 values of the distribution keys for $t = 3$, 16 refreshment messages are produced, stored and sent to VANET.

In the time interval $t = 4$, VU_0 is out of the chained refreshment process, with no possession of ρ_4, and so, is safe to use a single refreshment message again at $t = 5$. Although, since ρ_4 is not stored in the database, it has to be computed from the stored values of ρ_3 and $r(4)$. After the computation of ρ_4 the needed refreshment message is produced and the chained process follows with a single refreshment message per transition until $t = 8$, inclusive.

During the interval $t = 7$ a new seed is stored for VU_0, meaning that it was re-included and allowed to participate again in the routing process. This has no impact on the chained refreshment process neither on the creation of refreshment messages, but since the newly created seed was also loaded in the VU_0, this resurrected VU can now use the fallback method to get an update version of subset P_0 (see Section 4.1.3).

During the interval $t = 8$, the VU_1 is pinpointed as compromised and the consequences are similar to the exclusion of VU_0 at $t = 3$, with the difference of being used the last seed available for VU_0, and not the initial (compromised). The VU_1 is marked as compromised until a possible new seed being created.

The use of the auxiliary computations appears as a consequence of the lack of information immediately available in the database. To avoid such computations it would be needed to write the result of the refreshment of all the $n = 131071$ distribution keys in the database. Apply the refreshments only to the compromised keys, and not to all, would reduce the number of insertions in the database, but would imply an increase of complexity on storage and manipulation.

Since the auxiliary computations are the same performed to create the replies to the fallback method (see Section 4.1.3), after a long period of service, any iteration over all the missed values to create the complete chain could lead to a significant number of com-

putations. This can be efficiently solved by periodically calculating and storing all the distribution keys, associated to the same time interval, as seeds for this time interval.

5.3.2 Interactions with the simulator

The KDC reacts according to the requests made by the simulator. There are two requests specific from the simulator, and not existent in a realistic environment: a signal to start the simulation, and an option to validate the data at the end of the simulation. At the start of the simulation all the values from the previous simulation, if any, are flushed. To validate the data at the end of the simulation the KDC returns the values of the subset of the distribution keys, P_ν, in the same time interval that the VUs reach the end of the simulation.

The other interactions are divided into two groups; the ones performed by the human operator are the following:

`initialSetup:` takes the identification of the VU marks all the ascendants nodes of the VU as used and returns the correspondent subset P_ν;

`setBasalRateRefreshment:` receives and stores the new basal refreshment rate;

`reportCompromised:` receives a list of VUs to mark as compromised;

`resurrect:` receives a list of compromised VUs and re-includes them;

`resetKeysOnKDC:` creates new entries on table `seeds` and forces all the VUs to request the new subset P_ν. Initiates the part of the protocol described in Section 4.2.2.1.

The other group of interactions is formed by the ones available to RSUs:

`getRefreshment:` checks the existence of new refreshment and send all if they existent;

`directCommunication` receives a request according to the fallback method and sends a proper reply (see Section 4.1.3).

Chapter 6

Analysis of results

The purpose of computing is insight, not numbers.

— Richard Hamming

To validate the proposed solution as well as the implementation of the needed components, we simulated different scenarios and analyzed the results.

The tests related with the performance of VUs were already mentioned in Section 2.2. Theses results were taken as intrinsic characteristics and highly influenced our work from the very beginning.

The analysis of the simulation results is preceded by the analysis of the data-sets and a theoretical evaluation of the maximum number of messages, their occurrence and impact, under the conditions of a fully populated VANET (according to the KDC limit of 2^{16} VUs).

6.1 Input data

Based on the collected data we built 2 data-sets (see Section 5.2.2). We built a collection of scripts to mine the data and extract meaningful information from the data-sets.

The difference between the 2 data-sets is only the distance for which the VUs can receive and send beacons. The data-set where communications are limited to 100 meters have about 23% less received beacons when compared with the data-set with communications up to 1 kilometer. The cumulative percentage of received beacons by distance is shown in Figure 6.1.

The number of active OBUs is variable along the day, with a maximum of 396, as shown in Figure 6.2. Since the collection of the base data started at 18h:00m and the behavior

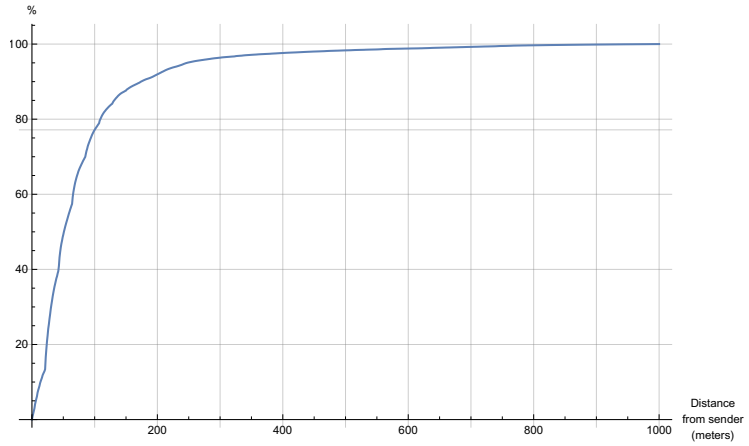


Figure 6.1: Cumulative percentage of received beacons by distance

of the OBUs is different depending on the time of the day, for convenience, we labeled the x-axis according to the real time of the day.

The 396 OBUs, along with the 50 RSUs, were organized in the KDC according to the best distribution, i.e. equally dispersed along the 2^{16} positions available. In the KDC, the nodes of the tree unused and not associated to any VU are automatically marked as unused (see Section 4.2.3).

We implemented an extra behavior in the KDC, uniquely for the test scenarios: the unused $(2^{16} - 396 - 50)$ positions for the non-existent VUs were used as *ghosted* devices with impact in the VANET if excluded. In a real scenario, if a VU does not exist it cannot be excluded. Allowing this in the test scenarios was a benefit, since we do need to lose the contribution of one or more of the 446 VUs to simulate the exclusion of compromised VUs.

While active, the OBUs may be isolated from the VANET. We measured the average time that OBUs are isolated, in minutes per hour. This time is dependent of the data-set in use, for communications limited to 1 kilometer it is shown in Figure 6.3. For communications limited to 100 meters the results followed the same pattern, with the values of isolation time increasing from 0 to 10 minutes.

When active, the VUs are exchanging beacons with the neighbors. Since the communications are based on broadcasted messages, without any mechanism to ensure the reception of the messages, the communication created among VUs may not be bidirectional: a VU, ν_1, may be able to listen a neighbor, ν_2, while ν_2 cannot listen ν_1. Since our solution relies on a epidemic propagation of messages according to the needs of the neighbors, the bidirectional communications are important and the presence of unidirectional communications may have a negative impact.

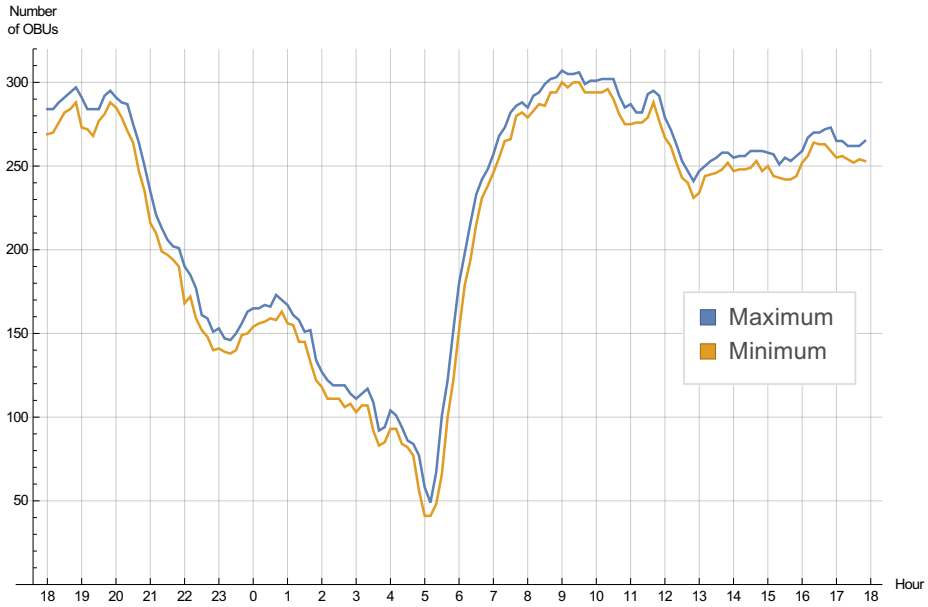


Figure 6.2: Number of active OBUs in the data-sets.

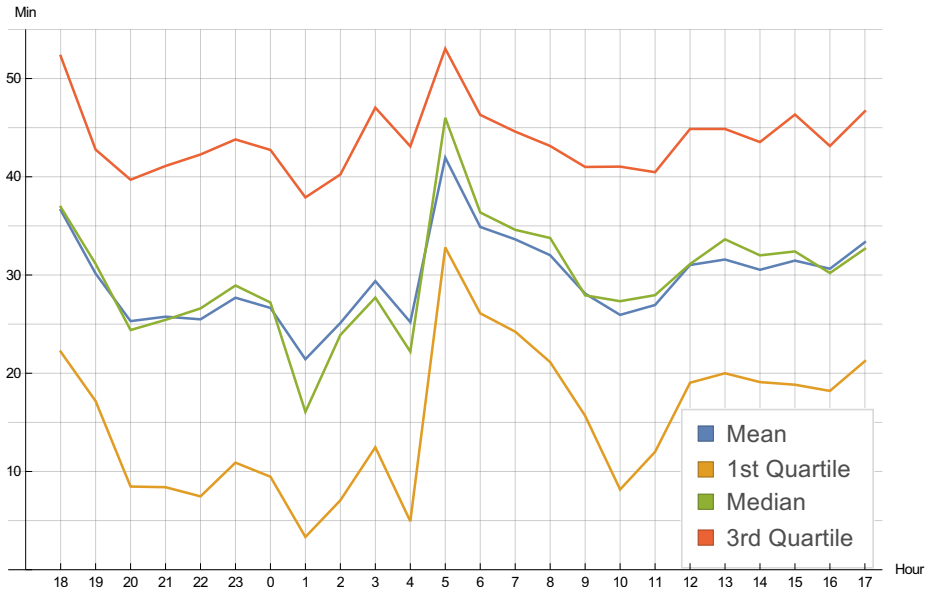


Figure 6.3: Isolation time of active OBUs when communicating up to 1 kilometer

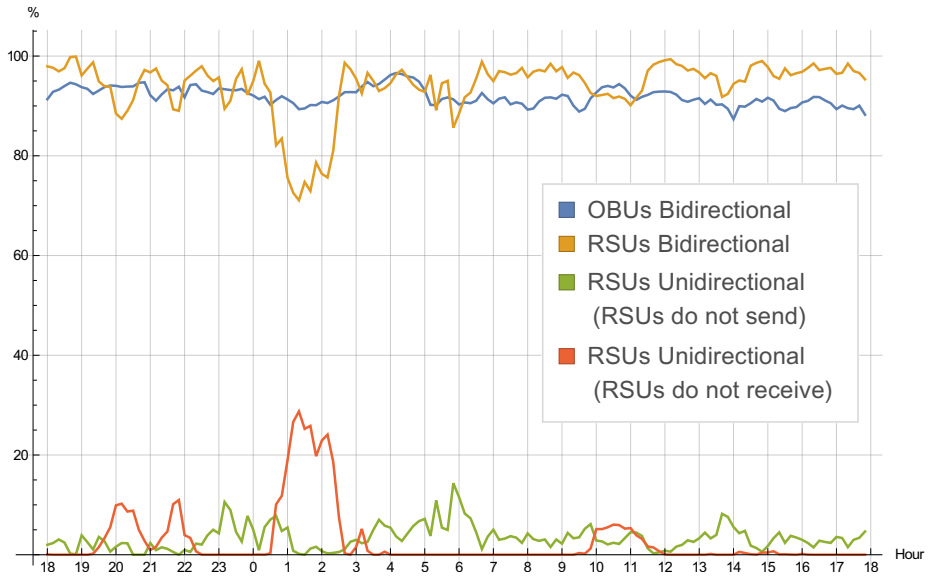


Figure 6.4: Characteristics of the connections in the VANET

The percentage of bidirectional connections created by OBUs and RSUs as well as the discriminated unidirectional connections for RSUs (can listen but cannot be listened or the opposite) are shown in Figure 6.4. It is important to notice that the OBUs can exchange beacons with RSUs and other OBUs while the RSUs only exchange beacons with OBUs. The percentage of unidirectional communications found in the OBUs is between 5% and 10%. In the case of RSUs there is some variation, between 0 and 30%.

The time since the last direct connection with an RSU made by the OBUs is shown in Figure 6.5.

The analysis of Figure 6.2, Figure 6.3 and Figure 6.4 suggests that approximately from 23h:00m to 6h:00m the VANET may be susceptible to communication problems. The problems may be caused by 3 different reasons, that individually occur for a relatively small period of time, but all in continuous time window, namely:

- A small number of active OBUs, specially from 4h:00m to 5h:00m (see Figure 6.2);
- A high number of OBUs isolated for a large period of time, around 5h:00m. More than 75% of the active OBUs are isolated more than 30 (non-continuous) minutes during 1 hour (see Figure 6.3);
- A high percentage of unidirectional communicating between RSUs and OBUs, from 1h:00m to 2h:30m and around 23h:00m and 6h:00m (see Figure 6.4).

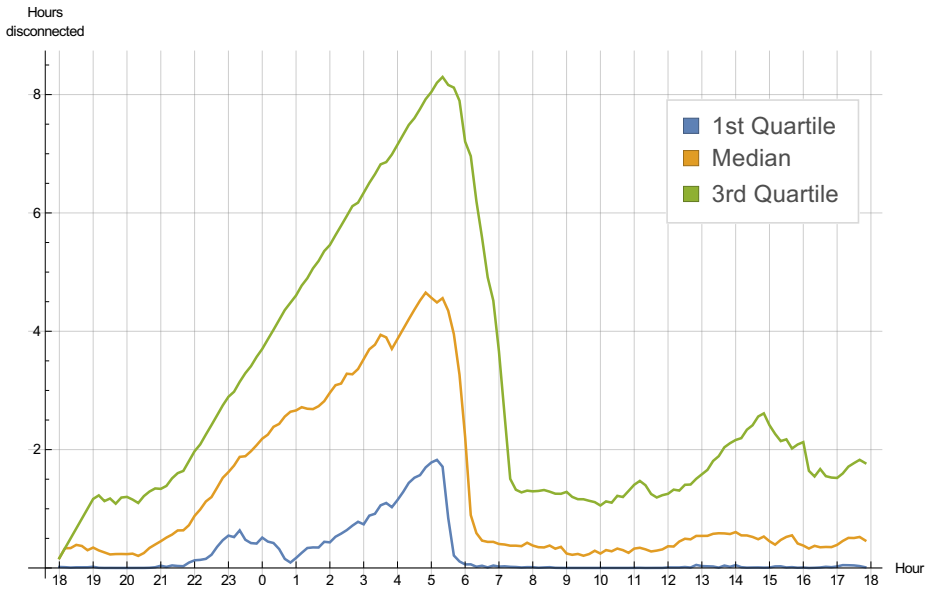


Figure 6.5: Time since the last direct connection with an RSU

The area covered by the VANET is shown in Figure 6.6. The routes followed by OBUs are in gray, while the position of the 50 RSUs available is a violet square.

A more detailed view of the routes used by OBUs is shown in Figure 6.7. In this figure it is possible to see the average number of seconds that each position is occupied by an OBU. The impact of the same OBU stopped in the same area during 2 seconds is the same as 2 different OBUs crossing the area. We took as reference the number of seconds (max) on the position more occupied by OBUs during the 24 hours and compared it with all the others positions. To highlight the relative difference between position we used $\frac{\log(x)}{\log(\max)}$ instead of $\frac{x}{\max}$, otherwise the map would be mostly red and would offer little visual information.

Another detailed view is shown in Figure 6.8. In this figure it is possible to see the positions where more and less beacons are received, for the data-set with communications limited to 1 kilometer. We measured the position of the OBU for each of the received beacon, during the 24 hours, in periods of 100 milliseconds. We took as reference the positions with more beacons received, max, and compared it with all the others positions using $\frac{\log(x)}{\log(\max)}$, where at least 1 beacon was received ($x > 0$). In this view, some of the routes disappeared, which means that no beacons were received during the 24 hours.

Based on Figure 6.8 it is evident that in the center of the map, the number of beacons received is much higher than in the periphery. This was expected since the center of the map is the area with RSUs and the area with an higher occupation of OBUs (see Figure 6.7). The bad communication in the periphery, completely absent in some routes, is a

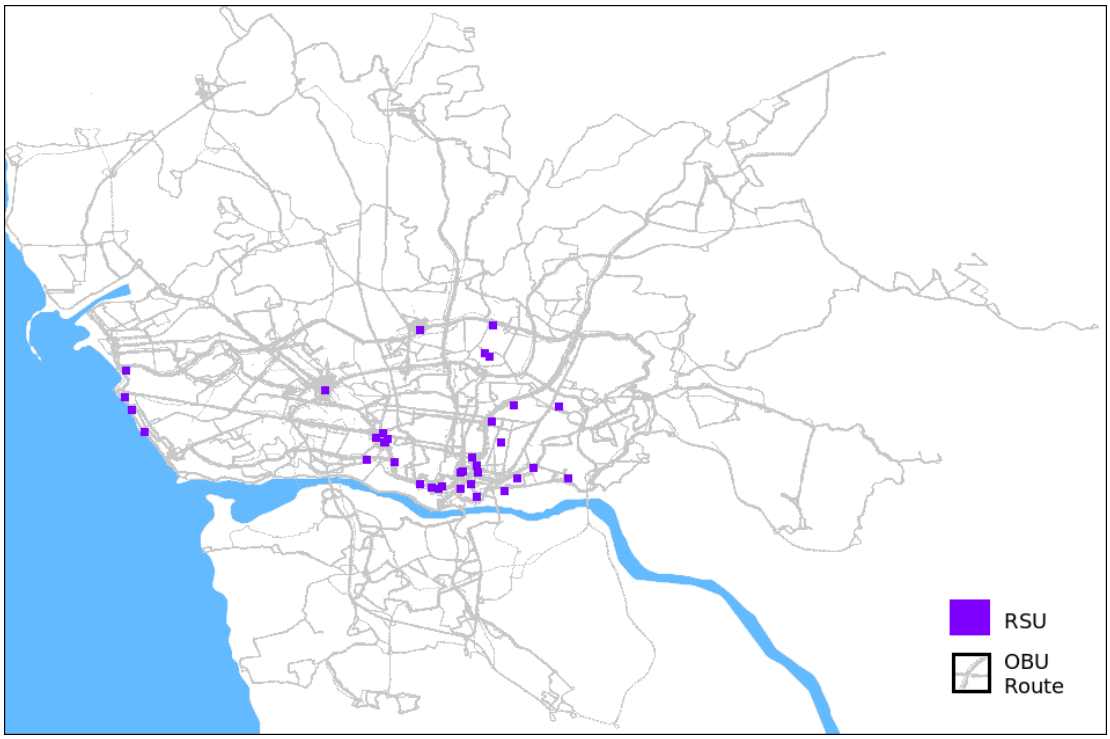


Figure 6.6: Area of the VANET

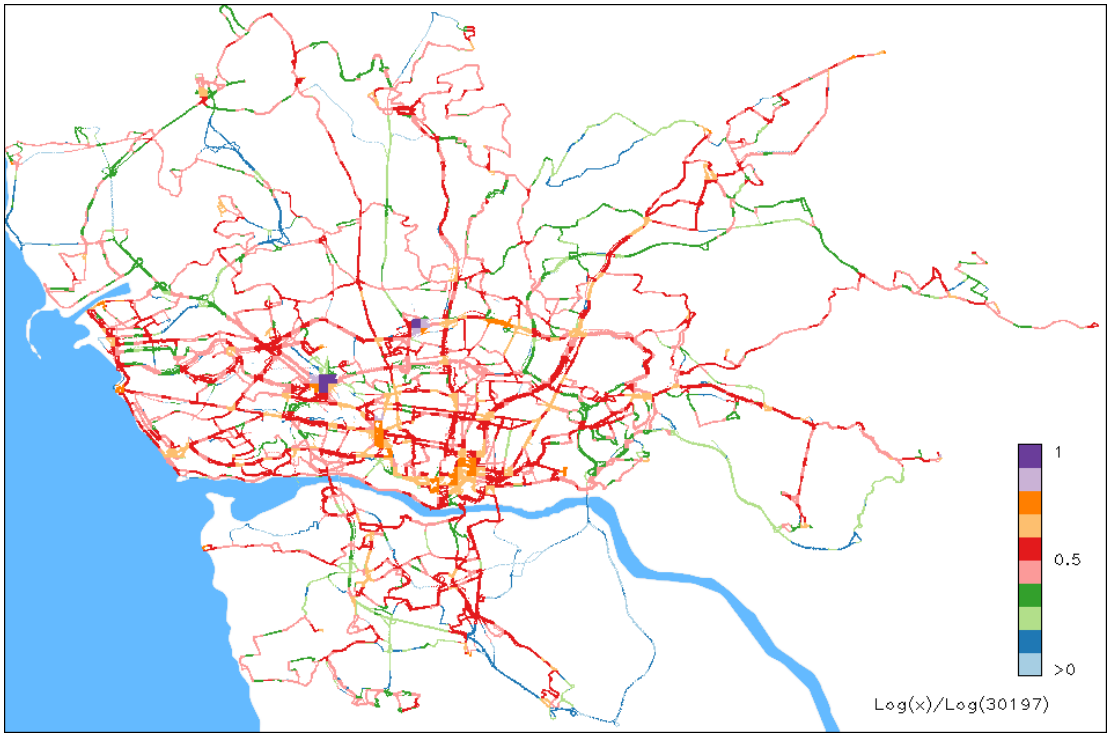


Figure 6.7: Relative density of OBUs

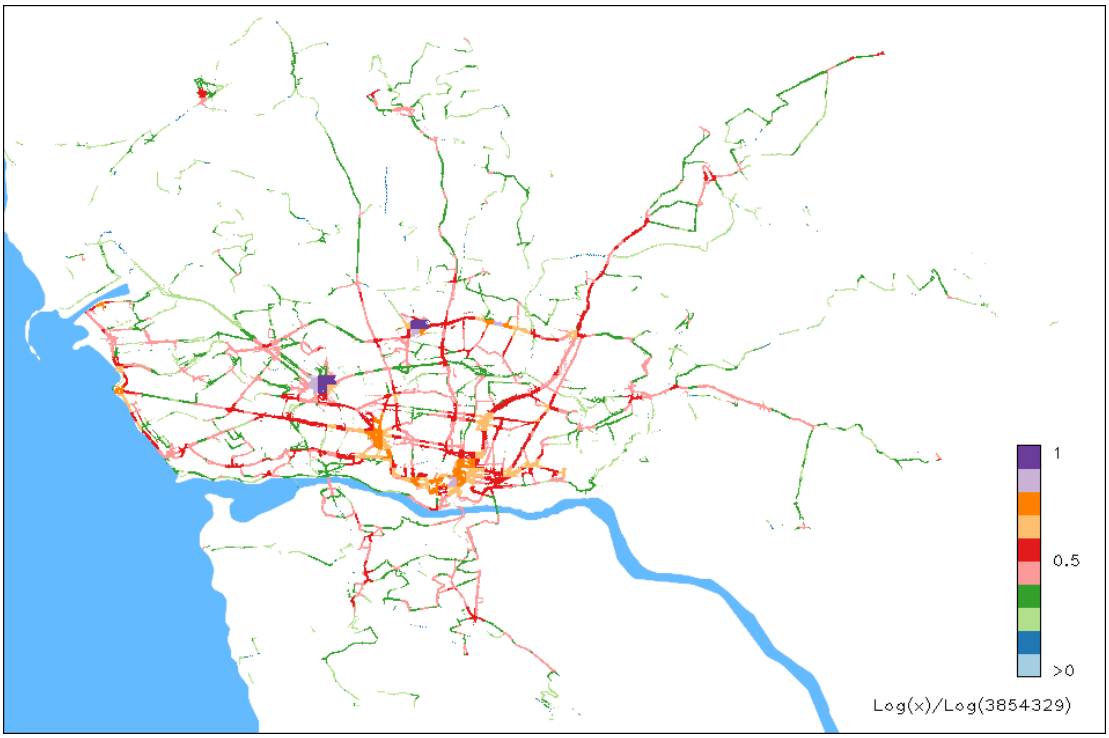


Figure 6.8: Relative density of received beacons

relevant fact, since we are dependent on the communication among OBUs to exchange the refreshment messages. Comparing with the the usage of the peripheral routes in Figure 6.7, it is expected that the OBUs using those routes will need more time to update than the ones closer to the center.

Figure 6.7 and Figure 6.8 have both 2 dark violet areas near center. Those small areas are the 2 places where the maximum of both maps are found, and represent 2 stop points where OBUs stay when not moving around the city.

6.2 Refreshment messages: number and occurrence

In Equation 4.18 we defined the number of distribution keys, in the worst case, used to create the refreshment messages needed to exclude c elements. We also mentioned that, when excluding c elements, in the worst case, to update half of the remaining ones, are used c refreshment messages. In Figure 5.2 is shown the behavior of our solution, regarding the number of messages sent to VANET, to update all and half of the remaining nodes.

To test and validate this definitions we implemented the process related with the exclusion of elements and selection of the respective nodes of the tree. This was done with

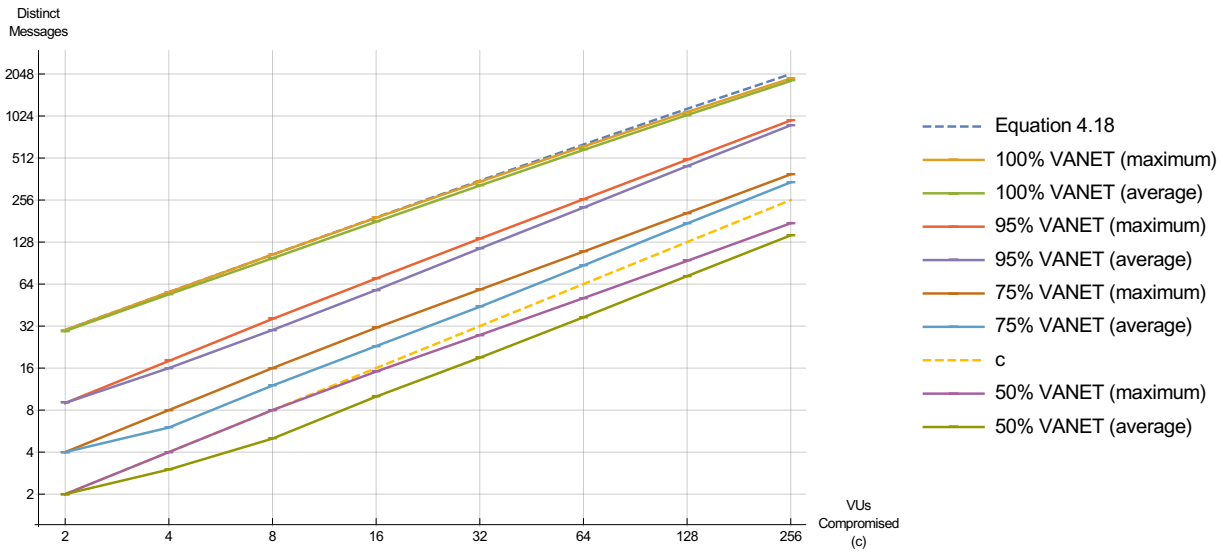


Figure 6.9: Number of messages to exclude VUs (Experimental results)

a perl script, that randomly compromises c VUs out of 2^{16}, and strictly follows the rules defined in Section 4.3.1.1:

1. Select the siblings of all the compromised tree nodes;

2. Exclude any sibling that is already compromised.

After this, we calculate the number of refreshment messages and the number of VUs that use each of the refreshment messages associated to each of the tree nodes.

We selected different numbers of VUs to compromised, $c \in \{2, 4, 8, 16, 32, 64, 128, 256\}$. For each of the numbers to analyze, we run 100000 tests: each test consists in randomly compromise c VUs and measure the respective impact. For each group of 100000 tests we computed the maximum and the average number of messages to update 100%, 95%, 75% and 50% of the remaining VUs.

We repeated the process to obtain the maximum and the average values 100 times and computed the 95% confidence interval (less than 1 for all the cases). The total number of executions was $8 \cdot 100000 \cdot 100 = 80 \cdot 10^6$.

The results are shown in Figure 6.9, among with the 2 theoretical values (dashed lines) to update all and half the VUs, in the worst case.

Figure 6.9 reveals that the defined values to the worst cases start to slightly diverge from the experimental results, specially for $c \geq 16$. This indicates that the predicated

c	$P(m_w)$
2	0.500008
4	0.0937586
8	0.00240429
16	1.13631×10^{-6}
32	1.81409×10^{-13}
64	3.32095×10^{-27}
128	8.26383×10^{-55}
256	4.37064×10^{-110}

Table 6.1: Occurrence of the worst case when generating refreshment messages

worst case is highly unlikely to happen. A more careful analysis, based on the probability of the worst case, is coincident with such results.

Since the worst case happens when each of the compromised VUs is uniquely associated to different tree nodes in the higher possible level[1], when randomly compromising c VUs out of 2^{16} the probability of the worst case is given by:

$$P(m_w) = \prod_{i=1}^{c} \frac{\frac{2^{16}}{c} \cdot (c - i + 1)}{2^{16} - i + 1} \tag{6.1}$$

The result of $P(m_w)$ for the different values of c is shown in Table 6.1, where it is possible to see the different magnitude of the occurrence of the worst case before and after $c = 16$, coincident with the experimental results in Figure 6.9.

In a VANET with 2^{16} VUs, the exclusion of 256 VUs will mostly result on $\cong$ 1900 messages to update all the remaining VUs and $\cong$ 143 messages to update half, rather than the previously predicated worst case of 2048 and 256 messages, respectively.

Figure 6.9 also reveals the relation between the number of messages to update 95% and the entire VANET, that is always, at least, twice smaller. This indicates that even when excluding an high number of VUs, even without all the refreshment messages cached, an updated OBU can with high probability contribute to the update of an high number of neighbors.

The number of refreshment messages is heavily dependent on the binary tree used in the KDC. We analysed others n-ary trees at an early stage of this work. Without loss of generality, we consider now the case of 1 exclusion in a VANET with a high number of

[1]When c=2, the worst case happen when there is 1 compromised VU in the first half (0 - 32767) and another in the second half (32768 - 65535). It means that for c=2 the total space is divided in c=2 continuous partitions with the same size and there is only one compromised VU in each partition. The same happen for c>2, c partitions with the same size and only 1 compromised VU per partition.

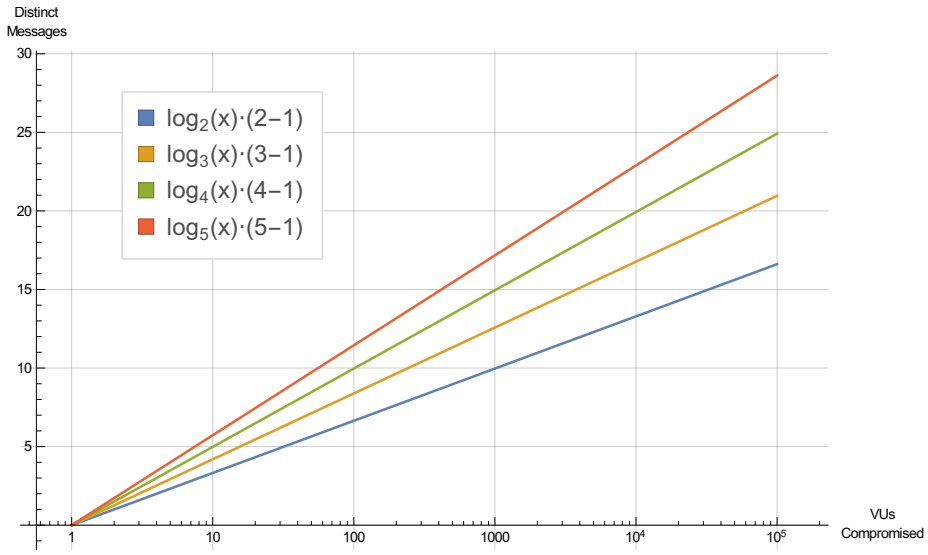


Figure 6.10: Number of refreshment messages produced to exclude 1 VU for different n-ary tree structures.

VUs where the KDC has as base a complete n-ary tree ($n \geq 2$) with V leafs.

Using a complete n-ary tree, the number of nodes with the same ascendant is n, while the depth of the tree is $\log_n(V)$. When excluding 1 VU there is 1 path with compromised nodes from the leafs level up to the root node and the number of non-compromised siblings for each of the compromised node in each level of the path is $n - 1$.

The exclusion of 1 VU consists on creating 1 refreshment message for each of the non-compromised sibling of all the compromised nodes, so, the number of refreshment messages is: $m_w = \log_n(V) \cdot (n - 1)$.

For a binary tree ($n = 2$) and $c = 1$, this is coincident with Equation 4.18. For the general case, the binary tree is the best structure and yields the best results since it minimizes the expression: $\log_n(V) \cdot (n - 1)$.

The results for $n = 2$ up to $n = 5$ are shown in Figure 6.10. Although not shown, higher values of n will follow the growing pattern, on the limit we have $n = V$ (and a refreshment message per VU!).

6.3 Test scenarios

We created multiple scenarios to analyze and validate the implemented solution. Since we have the intention to test the solution, we looked for the lower bounds of use, and the respective parameters. This goal led to some stressful configurations that are not expected

to find equivalent in a real context, but still, are useful for the evaluation.

On each scenario we tested different parameters and in same cases, based on the results, we fixed and carried on with the best parameters to some other test scenarios. We ran each of the individual configuration 50 times, with different values of seed (1 to 50) to the pseuso-random number generator of the simulator (see Section 5.2.3). The shown results are the average of the 50 executions.

We evaluated the variation of 8 distinct parameters, namely:

1. Percentage of beacons received with a valid MAC;

2. Percentage of lost routes due to security (assuming routes between OBUs and RSUs with a maximum of 3 hops, according to Section 2.4);

3. Percentage of equivalent routes (same route to same destination, with or without security);

4. Percentage of secure routes with more hops;

5. Percentage of secure routes with same number of hops but using a path with lower quality (based on Received Signal Strength Indication (RSSI));

6. Total number of refreshments messages cached in the active VUs (not normalized);

7. Percentage of beacons transmitted with the overhead of a refreshment message;

8. Percentage of beacons transmitted with the overhead of a sync message.

We started by evaluating the parameters on different time intervals to refresh the routing key, first with normal communication conditions and after with some adverse communication conditions. Then we tested the impact of the exclusion of VUs, first excluding 1 VU periodically and after excluding 16 VUs at once. Finally, we analyzed the behavior of our solution recovering from the disclosure of all the the symmetric key stored on KDC.

All the presented results were obtained with simulation running in less than 15 minutes on a personal computer with a CPU[2] with 4 cores, 8GB of ram and a HDD.

[2]Intel(R) Core(TM) i5-3317U CPU @ 1.70GHz

6.3.1 Fixed periods of refreshment

In this scenario we tested different periods to distribute a single refreshment message with the aim to simply update the routing key, ρ. Using the data-set with communications up to 1 kilometer, we tested periods of 10 minutes, 1 hour and 2 hours. All the tests were initially done allowing the OBUs to cache the refreshment messages to the last 8 time intervals (see Section 4.1.2). Based on the results with the period of 10 minutes, we inspected the impact of the variation on the historical cache, setting it to 32. The results of the 4 configurations are shown in Figure 6.11.

Globally, we see that the overhead of the messages associated to the secure routing (see Section 4.3) is only present in a small percentage of the transmitted beacons (2.5% maximum in `g)` and 0.27% in `h)`). Moreover, there a relation of approximately $\frac{1}{10}$ for the two types of transmitted messages, with the bigger ones (the sync messages) being the less frequent.

The number of refreshment messages cached in the VANET (see Figure 6.11 `f)`) is mostly independent of the period of refreshment, converging after an initial period where the caches are not fully populated. As expected, the strong dependency is associated to the number of active VUs (see Figure 6.2) clearly visible on the minimum around 5h:00m when less OBUs are active.

The higher percentage of lost routes in all the tests are coincident with the time window where it was expectable to have communication problems (see Section 6.1), so from 1h:00m to 6h:00m the VANET is clearly more susceptible to the chosen parameters.

Regarding the percentage of valid beacons, the period of 2 hours has the best results, with values close to 100%. Consequently, the routes created, based on the received beacons, have minor changes: the percentage of routes lost is residual, always lower than 0.25%; most of the routes are the same, always greater than 97.5%; the routes that are not equivalent, are replaced by alternatives with more hops or lower quality.

The percentage of valid beacons associated to the other periods of refreshments (10 minutes and 1 hour) is highly affected by the limit imposed on the historical cache. For the period of 1 hour caching the last 8 messages corresponds to 8 hours while for the period of 10 minutes corresponds to 1h:20m (8 time intervals) and 5h:20m (32 time intervals). After filling the cache, all these cases have associated a variation on the percentage of valid beacons, meaning that VUs do not keep the refreshment messages needed to update the neighbors. For the 2 hours period, after the filling of the historical (16 hours) there is no observable variation.

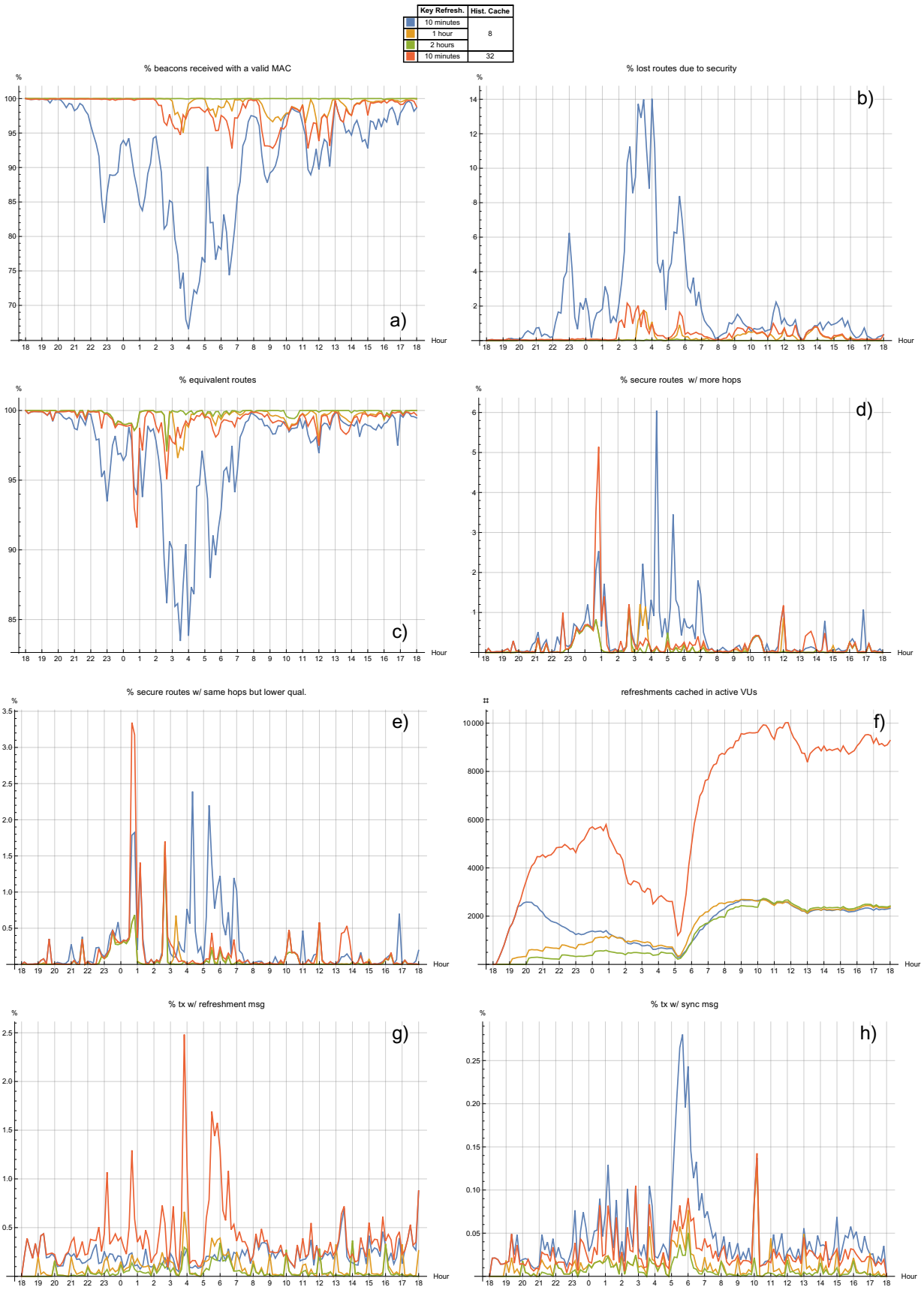


Figure 6.11: Results for fixed periods of refreshment

A period of 10 minutes combined with a limit of 8 time intervals for the historical cache yields the worsts results. With a period of 10 minutes, the higher percentage of valid beacons obtained with a limit of 32 time intervals in the historical cache, is coincident with a higher percentage of overhead caused by refreshment messages and a lower percentage of overhead related with the transmission of sync messages. This is coincident with the activation of a high number of OBUs, at 5h:10m (see Figure 6.2) with bidirectional connections with RSUs (see Figure 6.5). Although OBUs have contact with RSUs, the epidemic propagation of cached messages reduces the number of sync messages (see Figure 6.11 `g)` and `h)`).

The results associated to a period of 10 minutes and a limit of 8 time intervals for the historical are expected since any OBU offline during more than 1 hour and 20 minutes will be out of the routing process until direct contacting the KDC. Besides that, the OBUs using the periphery of the map for more than 1h:20m are tendentially in risk of being isolated and out of the routing process, due to the paths with few or no reception of beacons on it (see Figure 6.7 and Figure 6.8).

6.3.1.1 Adverse communication conditions

In this scenario we tested the behavior of our solution under adverse communication conditions, limiting the propagation of the refreshment and the sync messages in space and time. The limitation in space is done using the data-set with communications up to 100 meters, instead of 1 kilometer, with about 23% less of received beacons (see Section 6.1). The limitation in time is done by explicitly discarding 90% of the received beacons in all the VUs.

Both conditions are artificial and not likely to be found in real environments, at least not continuously for a wide area during a long period of time, as we are testing. Nevertheless, the behavior is indicative for less aggressive and unexpected conditions.

We discarded 90% of the received beacons with the aim to disassociate our solution from a possible deviation of the real conditions caused by the period of collection of the input data. The input data was collected in period of 2 seconds but the beacons are sent in periods of 100 milliseconds. This means that during the 2 seconds a VU may have listened from each neighbor between 20 to 1 beacons. Although it is not likely to happen the case of only 1 beacon be listened nor only just a few of them, we can not guaranty the same for the case of 19 or other numbers close to 20.

In the simulation we are always assuming that during the 2 seconds there are 20 listened

beacons per neighbor. By discarding 90% of them we are disassociating our requisites for the propagation of refreshment messages from any possible deviation from the real conditions. Moreover, discarding 90% of beacons let us analyze the behavior of our solution under the conditions of a lossy channel, were the delivery of messages to maintain the secure routing may be delayed.

We took as reference the period of refreshment of 2 hours without any limitation. We tested the limitation in space, in time and both together. Although the case without any limitation is already present in the previous scenario, for easy comparisons, we re-include it among the 3 tested configurations shown in Figure 6.12.

Globally, we see that all the analyzed parameters suffer little impact.

Inspecting the results in more detail, we see that the applied limitations on communication led to a higher, although small, percentage of lost routes, specially in the moment of releasing a new refreshment message to VANET. This is not surprising since the delayed transmission of refreshment messages will cause a delayed update on some OBUs than can not be used in the multi-hop routes by the updated ones and vice-versa.

The periodic spikes on the percentage of overhead associated to the refreshment messages are caused by the demand of the newly released refreshment messages. The coincident periodic spike on the percentage of overhead associated to the sync messages is not caused by a higher demand, but rather by repetitions of the requests and consequent retransmission of replies: since the beacons can only carry one type of message at once, the transmission of more refreshment messages (near the RSUs) will imply that same of the sync messages are discarded (and new requests need to be done).

Before 6h:00m, not coincident with the releasing of a new refreshment message, there is a negative spike on the percentage of the equivalent routes, when the communications are only limited to 100 meters. Although this specific situation have not been predicated, the behavior is not unexpected, since when the range of communication is smaller, the OBUs are more dependent of the neighbors to reach the distant destinations, so more likely to be affected by the state of more neighbors. Moreover it is coincident with a higher percentage of unidirectional communications (see Figure 6.4). The alternative routes found in the VANET led to an higher increase of the percentage of routes with more hops rather than the ones with the some number of hops but lower quality, by the same reason.

With the obtained results we can conclude that our solution, namely the epidemic propagation of the refreshment messages, can still operate normally with a less frequent exchange of key management information among VUs.

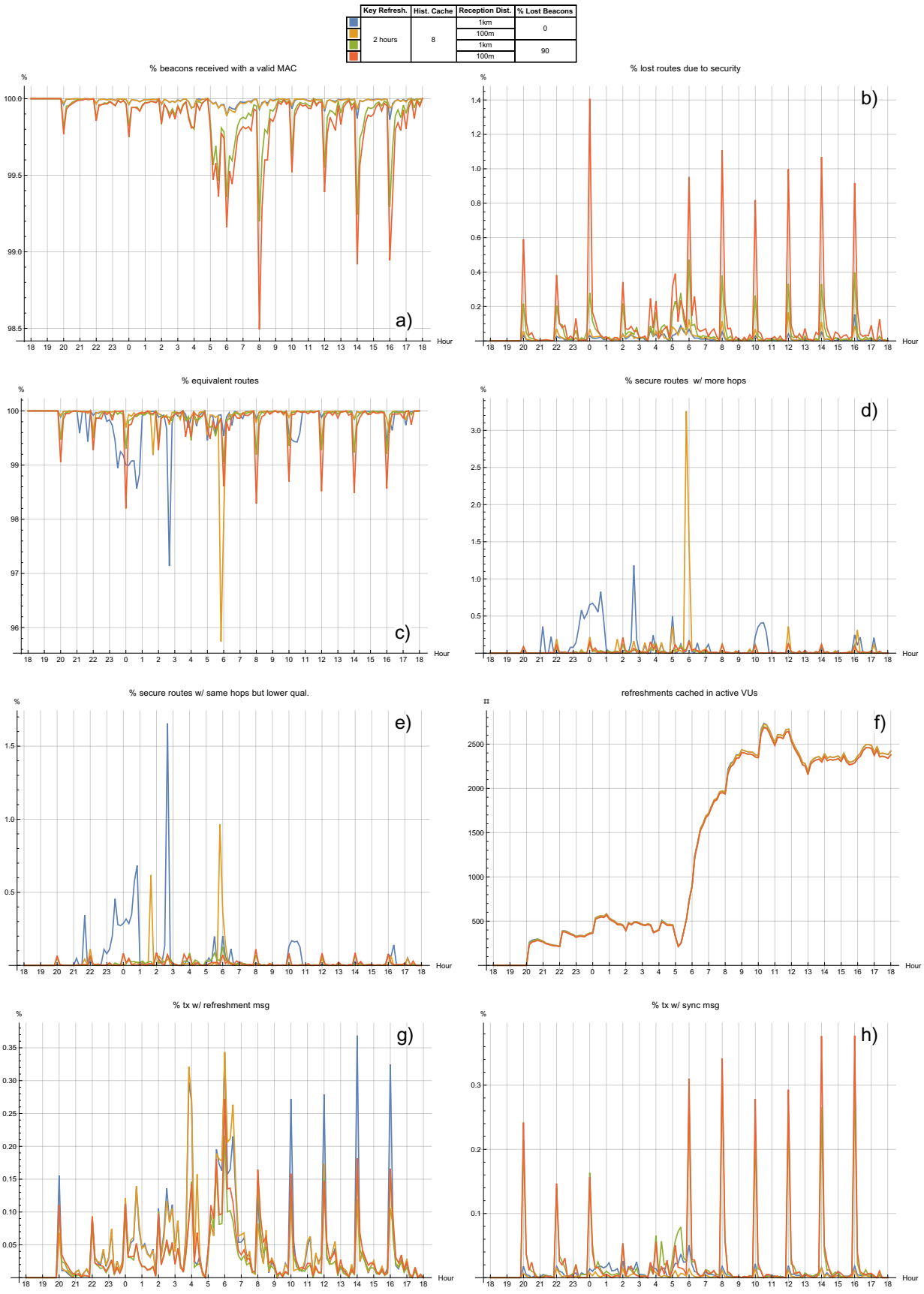


Figure 6.12: Results for adverse communication conditions

6.3.2 Successive exclusions of a single VU

In this scenario we tested the impact of successive exclusions of a single VU. Based on the results presented in Section 6.3.1, we used the period of 2 hours and all the same parameters, data-set and size of the historical cache, to successively exclude 1 VU. In did the first exclusion at the 2 hours of simulation time, so in the 24 hours of the data-set we did 11 exclusions. From the 11 exclusions we forced the worst case in 10 of them and used the other 1 to compare the difference. As mentioned in Section 6.1, we used non-occupied places in the KDC and did not exclude VUs for which we have data.

Each 2 hours we forced the update of the routing key, but now with an explicitly exclusion of one VU at the time. The instructions to the exclusion of the VUs were given to the simulator by file with commands (see Section 5.2.3).

Since we are excluding VUs, more than one refreshment message will be generated. With multiple refreshment messages, a VU may be updated but not in condition of update its neighbors. This case was predicated and led to the definition of the basal refreshment rate, *brr*, (see Section 4.1.2).

We defined the basal refreshment rate to 10% and ran two more tests. In one we only changed the basal refreshment rate. In the other one, besides the basal refreshment rate we followed the configurations tested in Section 6.3.1.1 and changed the communication parameters limiting them in space up to 100 meters and discarding 90% of the received beacons. The results of the 3 tests are shown in Figure 6.13.

In all the results measuring percentages is evident the presence of periodic spikes, associated to the 2 hours period. The worst results are obtained under adverse communication conditions, although, even in this case, the impact of the exclusions tend to disappear after 1 hour and are not present at the time of the next exclusion.

Without the basal refreshment rate, *brr*, the performance is not good. First there is an evident inability to authenticate all the beacons. Second the beacons with the overhead of the refreshment message is extremely high, near 40% in some cases. This happens because the OBUs are instructed to update themselves and do nothing more regarding the other refreshment messages. Later, when the selfish OBUs find outdated neighbors, they do not have the messages needed by them. When listening an outdated neighbor and not in possession of the need refreshment messages, the OBUs still send a randomly refreshment message (this makes it possible to any of the neighbors, independently of their state, to validate the KDC signature and to discover that they are outdated). Although the OBUs are choosing randomly, they just own a few of the messages and will end up choosing

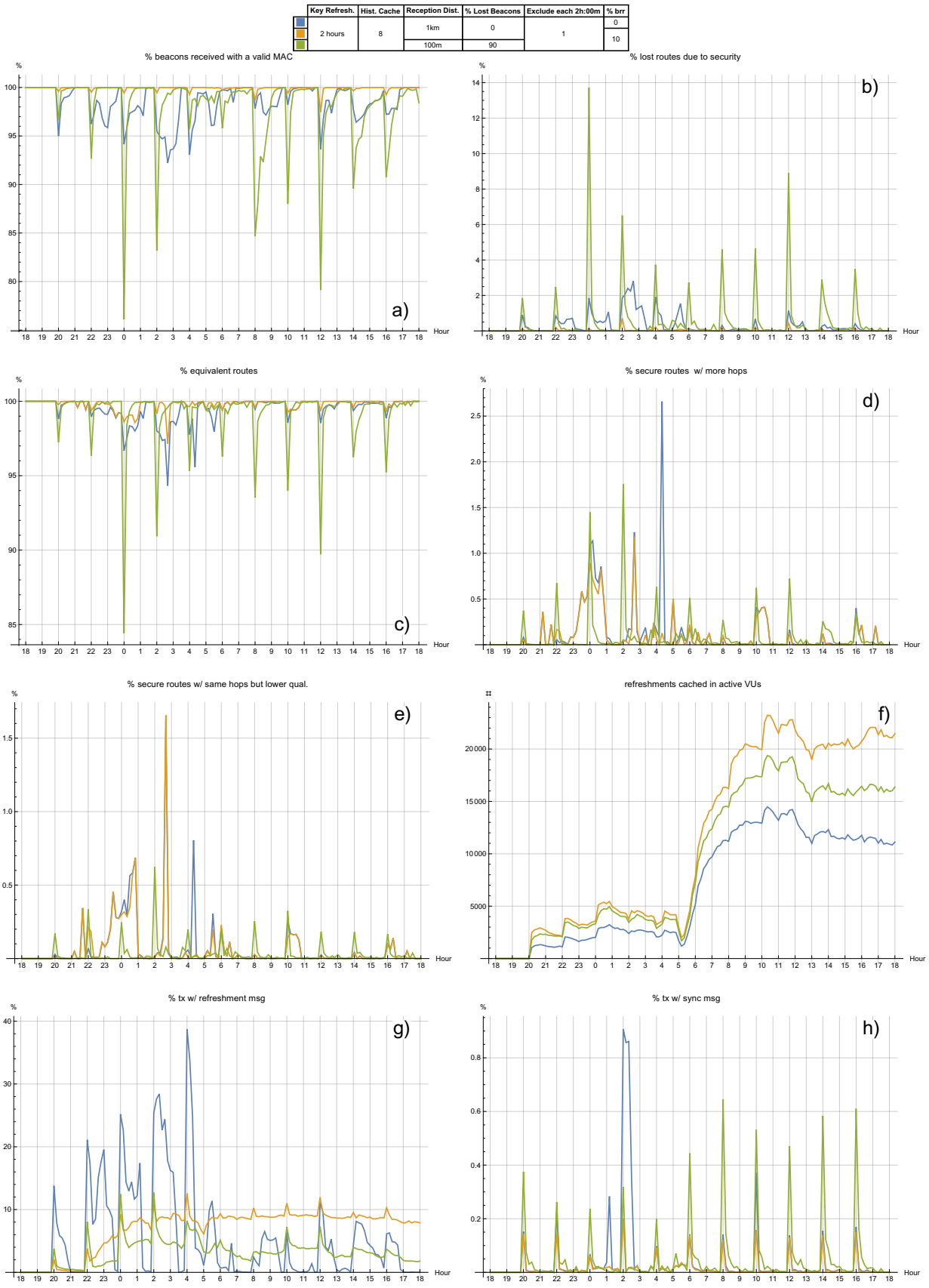


Figure 6.13: Results for successive exclusions of a single VU

randomly from an incomplete set of possibilities[3].

The case with the basal refreshment rate set (without limitations on communication) has clearly a good performance, with almost no impact on the percentage of valid beacons. The solution to improve the performance is associated a better dispersion of all the refreshment messages. This is visible in the number of total refreshment messages found in the VANET. Although the basal refreshment rate led to a blind distribution of refreshment messages without an explicit detection for its need, the percentage of overhead in the beacons is reduced.

At the end of the tests, the KDC produced 109 refreshment messages for the 11 exclusions. In detail, 10 exclusions generated 10 refreshment messages (the worst case according Equation 4.20) and 1 exclusion generated 9 refreshment messages.

6.3.3 Single exclusion of multiple VUs

In this scenario we tested the impact of a single exclusion of multiple VUs. Again, as in Section 6.3.2, based on the results presented in Section 6.3.1, we used the period of 2 hours and all the same parameters, data-set and size of the historical cache, to apply the exclusions.

We chose 16 out of the 446 as the number of VUs to exclude simultaneously. The 16 *ghosted* elements (see Section 6.1) were chosen in order to produce the maximum number of refreshment messages possible. The time for the exclusions was defined at the limit of the historical cache, after 16 hours of simulation. This time was chosen based on the evidence given by the smaller periods tested on Section 6.3.1 that after the limit of the historical cache the percentage of valid beacons may start to oscillate.

The logic of tests followed the one applied in Section 6.3.2, an initial test without a basal refreshment rate, another one with a basal refreshment rate of 5%, and a third with adverse communication conditions. The results of the 3 tests are shown in Figure 6.14.

In all the results measuring percentages there is a spike associated to the exclusion of the 16 elements. Based on the results of the previous 2 scenarios, this was expected.

The relation with the total number of refreshment messages cached in the VANET was, in the same way as in Section 6.3.2, positively influenced by the basal refreshment rate defined to 5%. The same happened for the percentage of overhead related with the refreshment messages.

[3]http://dilbert.com/strip/2001-10-25

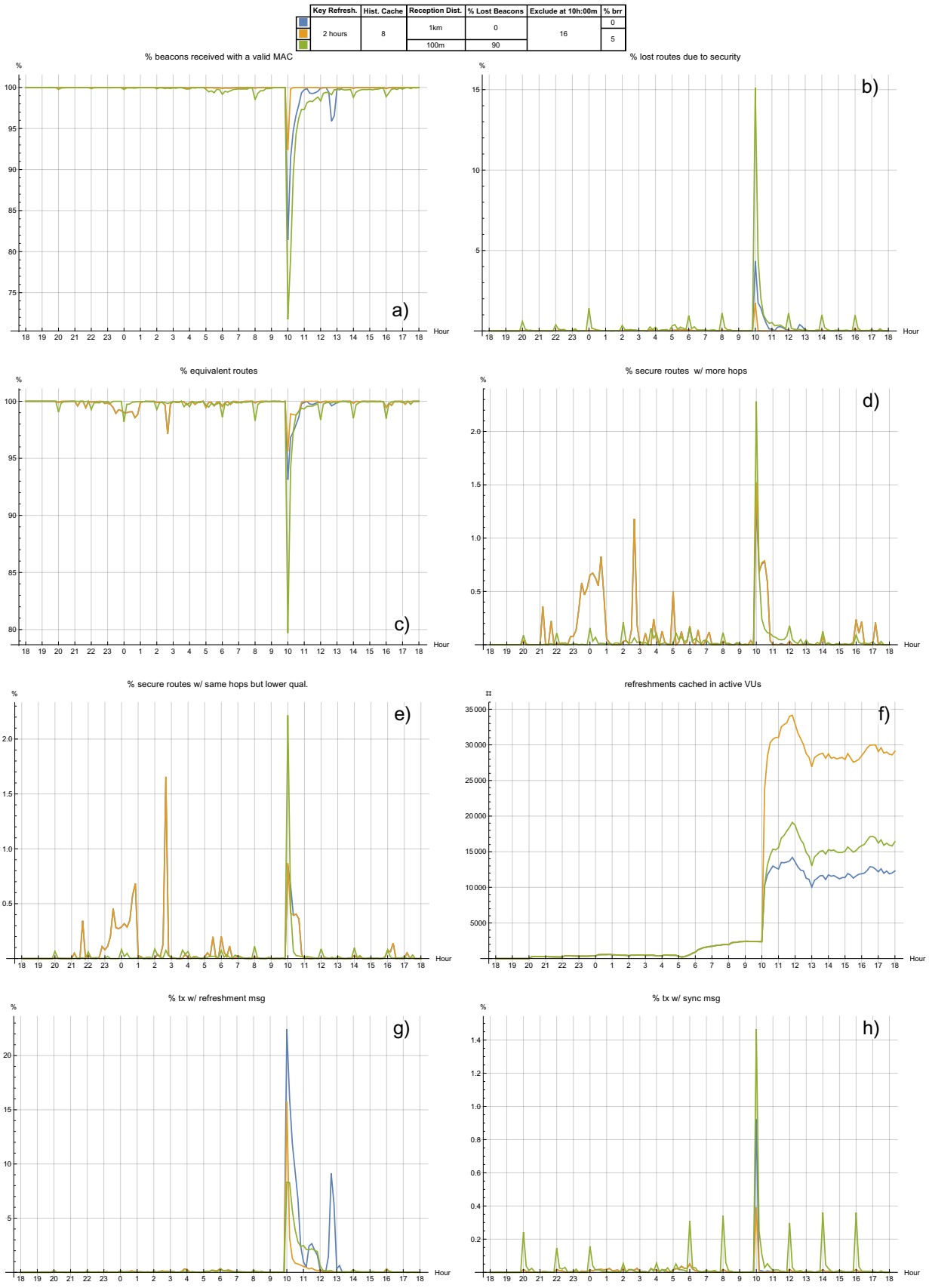


Figure 6.14: Results for single exclusion of multiple VUs

Without adverse communication conditions, the minimum percentage of valid messages in the VANET is 93% and in 10 minutes the network recovers as if no exclusion had been made. In the case of adverse communication conditions, these values are, respectively, 73% and 3 hours (and 2 small variations in the following releases of refreshment message).

At the final of the tests, the KDC produced 106 refreshment messages for the 11 exclusions. In detail, 96 refreshment messages to exclude the 16 VUs, the worst case according Equation 4.20 and 1 refreshment message for each of the other periodic releasing happening before and after, without exclusions.

6.3.4 Reset of all the keys on the KDC

In this scenario we tested the behavior of our solution recovering from the disclosure on all the key in the KDC (see Section 4.2.2.1).

We used the period of 2 hours to release refreshment messages and gave an order to reset all the keys in 2 different times: at the begin of the simulation and after 16 hours. We used 2 different reset instants aiming to differentiate the impact caused to the VANET.

As in the previous scenarios, we tested the same configurations with adverse communication conditions. The results of the 4 tests are shown in Figure 6.15.

Since all the OBUs are informed, by the reset message, to request new keys, using the sync messages (see Section 4.2.2.1), for all the tested scenarios, this is the one where the percentage of overhead related with the sync message is higher.

With the exception of the case with adverse condition of communication and the reset order at 10h:00m, all the others ended up recovering at the end of the simulation.

Whit adverse communication conditions the impact on the percentage of valid beacons is amplified, but still the percentage of routes lost is most of the time smaller than 5%.

With normal communication conditions, the line of the percentage of the valid beacons related with the reset at the middle of the simulation (10h:00m) closely follows the line of the order at the begin of simulation (18h:00m). Moreover, with normal communication conditions, the percentage of overhead related with the sync messages, essential for the OBUs to get the new keys, has little variation (with exception of the moment of the reset). This is a strong indication that the topology of the VANET influenced the process, and the OBUs are slowly, but constantly, contacting with the KDC. This is also according to the little and constant spikes present in the cases with adverse communications. These almost constant spikes happen due to the discarded messages that will force the repetition of the requests.

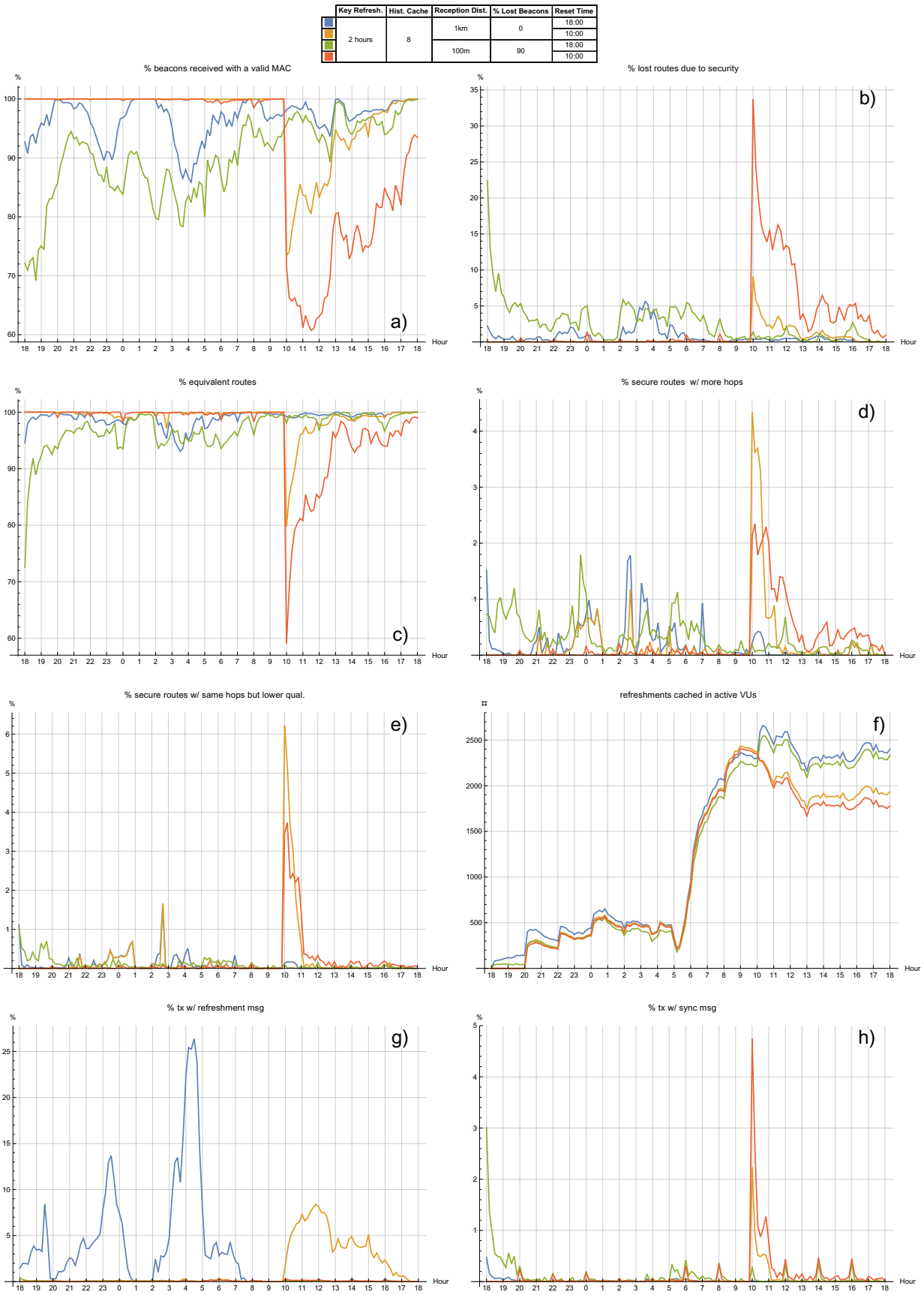


Figure 6.15: Results for reset of all the keys on the KDC

The number of refreshment messages in the last 8 hours of the simulation, tend to form 2 pairs with a small gap between them. The bottommost line of each pair represent the reset at the middle of the simulation (10h:00m) while the topmost one represent the reset at the beginning (18h:00m). At the end of the simulation, the cases for which the reset happened at the middle of the simulation still have the time interval related with the reset in their historical cache window: since the reset message is only cached by RSUs, the total occupation of the historical cache in the OBUs is smaller.

6.3.5 Global considerations

The previous scenarios with the correspondent variation of parameters gave important insights about the behavior of our solution under the different conditions found along the 24 hours of the data-set. It is evident that the different characteristics of the VANET found at different times have a strong impact on the evaluated parameters.

For the available data, we found the functional lower limits of the parameters, although, in different situations, those values are not guarantee to have the same impact. So, we do not have the goal to provide global best parameters, but rather indicative parameters among with a tool, guide lines and expected behavior to analyze new situations.

The analysis of other situations, is dependent on the collection of different data to (automatically) build new data-sets. The time for each simulation (15 minutes in the worst case in a computer with limited characteristics) is relatively small so a wide number of tests are possible in a short period of time.

We had available a different set of 24 hours of data collected from the same VANET (the same geographical area). However, the number of VUs available in this data-set is smaller, only 350 (instead of 446), and the time granularity of the collection is 5 seconds (instead of 2). We did not considered this second data-set on the detailed tests, since it has worst characteristics. Nevertheless we used it to confronting the results of some association of parameters with different conditions.

We built another data-set for `loop` (see Section 6.1) and analysed the percentage of valid beacons received and the percentage of lost routes. The results of 4 associations are presented in Figure 6.16. When compared with the correspondent ones of the previous scenarios, it is visible a small difference. Although limited to 2 days to analyse, we saw that the patterns and the relations are similar, but different, when operating in similar conditions.

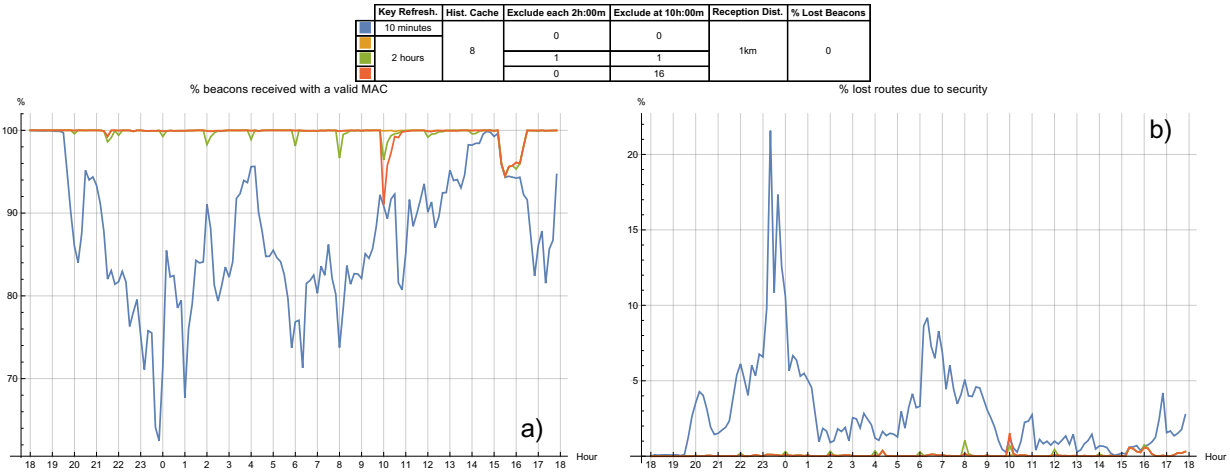


Figure 6.16: Representative results for a data-set with 350 VUs and time granularity of 5 seconds. The orange line is most of the time under the other lines. It is visible around 11h:00m in a).

Chapter 7

Conclusions

(he) must always keep in mind this concept of change and progression.

— Emil Zátopek

We presented TROPHY (Trustworthy VANET ROuting with grouP autHentication keYs), a set of protocols capable of maintaining the authenticity of routing messages in a VANET, `loop` (loop over orderly phases), an interactive simulator for testing and validating TROPHY along with a prototype of KDC. The solutions presented were designed in order to deal with restricted and well defined conditions found in a specific VANET, namely the Veniam's VANET, in Porto. To validate the architecture and the protocols we developed a simulator where we implemented the protocols and analyzed their behavior and impact taking advantage of real data collected from our target VANET.

We realised, at an early stage of our work, that traditional solutions using digital signatures that are usually applied in similar problems, but different contexts, are unsustainable in our case, mainly because we were dealing with a scenario were time and processing power were both restricted. Those exactly same restrictions led to the developed of a new, security-agnostic routing algorithm, the one we used as starting point, since none of the existent ones published in the literature were able to scale during the deployment of a real VANET. The development of the routing strategy was new and challenging by itself, so security aspects where postponed for future improvement stages.

Security mechanisms are intrinsically associated to an extra layer of overhead. We were aware of the emergence of such layer as well as the thin red lines that would make it sustainable in our context.

Although focused on *ad hoc* based technologies, we initially tackled our problem with the possibility of using different technologies and policies, even if more costly (such as cellular network connectivity or recalls of vehicles for some physical intervention). However, since the very beginning that none of those possibilities were considered elegant, practical of cost-effective. The initial effort for avoiding those alternative strategies, is now, at the end of the work, considered the main cause for the extremely satisfactory final results. Not only we ended up building a functional solution using solely the *ad hoc* infrastructure, and mainly its existing routing control plane, but we also achieved better results than the ones initially envisioned, specially the ones related with:

- The lower bounds of the key refreshment periods;
- The re-engagement of VUs that were offline for long periods of time;
- The exclusion of compromised VUs;
- The recovering from a major disaster that would be the disclosure of all the distribution keys held by the KDC.

7.1 Future work

Based on the developed work and all the interiorised experience, new ideas and improvements have been considered, but not implemented since they are largely beyond the scope of this thesis.

A logical following step is the implementation of the proposed protocol on the real devices. Along with the implementation and the first tests in real environment, the collection of periodically data (such as the routes available) along with the one already existent in the data-set, would be a tremendous advantage, since it would be possible to confront the real data and the simulation results and detect (and solve!) any possible mismatch.

Assuming the continuation of the developed work, both simulator and KDC have topics for improvements. The KDC prototype is more susceptible from being affected by the new ideas, so any modification or improvement, must consider such ideas.

The simulator was developed specially for the specific case presented in this thesis, although with modularity (for different scenarios) in mind. Currently, during the simulation, the information exchanged between VUs is generic (but focused on the scenario of this thesis), so new fields could be easily included, implementing new and different protocols.

However, although possible, this would lead to heavy dependencies and to an agglomeration of different information in the same place. In our opinion, a better approach implies the decoupling of each different situations (and respective data) among each other. This was not implemented, but the refactoring needed to achieve this level of modularity is expected to be minor.

Regarding the new ideas and improvements, those are related with 5 main topics:

Distribution of asymmetric keys: The distribution of asymmetric keys was not considered during this work. Based on the results obtained, the proposed solutions for the distribution of symmetric keys can be explored and extended to implement this improvement.

Possibility of cooperation between different KDCs: The possibility of temporally cooperation, without compromising any of the companies in the future time is a logical improvement. With this ability, OBUs of different companies could help each other upon some specific order from the respective KDC. Also, static nodes, such as sensors, could have the benefit of receiving cryptographic material when some OBU is crossing the area.

The use of alternative technologies: The current OBUs can connect to WiFi networks, such as 802.11g. The ability to use such extra technology can increase the number of places where OBUs can receive information. With this, the task of RSUs can be replaced by APs in remote areas. Alternatively, it is interesting the idea of using long-range, low throughput radio communication technologies, such as LoRA[1], to perform massive key updates in a VANET.

A mechanism for periodical collection: The decisions in the VANET may be improved based on the analysis of the previous conditions. Along with the implementation on the field, a mechanism to collect and analyze the impact of different orders may be considered essential.

A mechanism for intrusion detection: We built all the mechanisms to exclude compromised nodes after being pinpointed by any reason. According to the conditions faced by VUs, the detection of misbehaving VUs in the VANET may significantly improve the overall functionality of the network by allowing a fast exclusion of compromised VUs.

[1] https://www.lora-alliance.org

Bibliography

[1] IEEE guide for wireless access in vehicular environments (WAVE) - architecture. pages 1–78.

[2] IEEE standard for wireless access in vehicular environments (WAVE) – networking services. pages 1–160.

[3] IEEE standard for wireless access in vehicular environments (WAVE) – security services for applications and management messages. pages 1–240.

[4] K. J. Ahmed, M. J. Lee, and J. Li. Layered scalable WAVE security for VANET. In *MILCOM 2015 - 2015 IEEE Military Communications Conference*, pages 1566–1571.

[5] Neeraj Kumar Amit Dua. A systematic review on routing protocols for vehicular ad hoc networks. 1(1):33–52.

[6] Rasmeet S Bali, Neeraj Kumar, and Joel J. P. C. Rodrigues. Clustering in vehicular ad hoc networks: Taxonomy, challenges and solutions. 1(3):134–152.

[7] Elaine Barker. Recommendation for key management part 1: General.

[8] Elaine B. Barker and Allen L. Roginsky. Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths.

[9] Rimon Barr, Zygmunt J. Haas, and Robbert van Renesse. JiST: An efficient approach to simulation using virtual machines: Research articles. 35(6):539–576.

[10] G. Cameron, B. J. N. Wylie, and D. McArthur. PARAMICS-moving vehicles on the connection machine. In *Proceedings of Supercomputing '94*, pages 291–300.

[11] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: a taxonomy and some efficient constructions. In *IEEE INFOCOM '99. Eighteenth*

Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings, volume 2, pages 708–716 vol.2.

[12] Carlos Ameixieira and Filipe Neves. Service-based layer-2 routing protocol, veniam.

[13] Tzu-Chiang Chiang and Yueh-Min Huang. Group keys and the multicast security in ad hoc networks. In *2003 International Conference on Parallel Processing Workshops, 2003. Proceedings*, pages 385–390.

[14] David R. Choffnes and Fabián E. Bustamante. An integrated mobility and traffic model for vehicular wireless networks. In *Proceedings of the 2Nd ACM International Workshop on Vehicular Ad Hoc Networks*, VANET '05, pages 69–78. ACM.

[15] Hugo Conceição, Luís Damas, Michel Ferreira, and João Barros. Large-scale simulation of v2v environments. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, pages 28–33. ACM.

[16] Babak Daghighi, Miss Laiha Mat Kiah, Shahaboddin Shamshirband, Salman Iqbal, and Parvaneh Asghari. Key management paradigm for mobile secure group communications: Issues, solutions, and challenges. 72:1–16.

[17] Gianluca Dini and Ida Maria Savino. An efficient key revocation protocol for wireless sensor networks. In *Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, WOWMOM '06, pages 450–452. IEEE Computer Society.

[18] Richard Gilles Engoulou, Martine Bellaïche, Samuel Pierre, and Alejandro Quintero. VANET security surveys. 44:1–13.

[19] H. Harney and C. Muckenhirn. Group key management protocol (GKMP) specification.

[20] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. *An Introduction to Mathematical Cryptography*. Undergraduate Texts in Mathematics. Springer New York.

[21] J. Härri, F. Filali, C. Bonnet, and Marco Fiore. VanetMobiSim: Generating realistic mobility patterns for VANETs. In *Proceedings of the 3rd International Workshop on Vehicular Ad Hoc Networks*, VANET '06, pages 96–97. ACM.

[22] M. Ikeda, E. Kulla, L. Barolli, and M. Takizawa. Wireless ad-hoc networks performance evaluation using NS-2 and NS-3 network simulators. In *2011 International Conference on Complex, Intelligent, and Software Intensive Systems*, pages 40–45.

[23] B. Jiang and X. Hu. A survey of group key management. In *2008 International Conference on Computer Science and Software Engineering*, volume 3, pages 994–1002.

[24] Paul Judge and Mostafa Ammar. Security issues and solutions in multicast content distribution: A survey. 17:30–36.

[25] Pabitra Mohan Khilar and Sourav Kumar Bhoi. Vehicular communication: a survey. 3(3):204–217.

[26] Daniel Krajzewicz. Traffic simulation with SUMO – simulation of urban mobility. pages 269–293.

[27] Rahul Mangharam, Daniel Weller, Raj Rajkumar, Priyantha Mudalige, and Fan Bai. GrooveNet: A hybrid simulator for vehicle-to-vehicle networks. pages 1–8. IEEE.

[28] V. Gayoso Martínez, L. Hernández Encinas, and C. Sánchez Ávila. A survey of the elliptic curve integrated encryption scheme. 2(2):7–13.

[29] Mohamed Nidhal Mejri, Jalel Ben-Othman, and Mohamed Hamdi. Survey on VANET security challenges and possible cryptographic solutions. 1(2):53–66.

[30] Ralph Charles Merkle. Secrecy, authentication, and public key systems. AAI8001972.

[31] Wee Hock Desmond Ng, Michael Howarth, Zhili Sun, and Haitham Cruickshank. Dynamic balanced key tree management for secure multicast communications. 56(5):590–605.

[32] L. E. Owen, Yunlong Zhang, Lei Rao, and G. McHale. Traffic flow simulation using CORSIM. In *2000 Winter Simulation Conference Proceedings (Cat. No.00CH37165)*, volume 2, pages 1143–1147 vol.2.

[33] Debajyoti Pal. A comparative analysis of modern day network simulators. In David C. Wyld, Jan Zizka, and Dhinaharan Nagamalai, editors, *Advances in Computer Science, Engineering & Applications*, volume 167, pages 489–498. Springer Berlin Heidelberg.

[34] Byungkyu Park and Hongtu Qi. Microscopic simulation model calibration and validation for freeway work zone network - a case study of VISSIM. In *2006 IEEE Intelligent Transportation Systems Conference*, pages 1471–1476.

[35] Young-Hoon Park, Dong-Hyun Je, Min-Ho Park, and Seung-Woo Seo. Efficient rekeying framework for secure multicast with diverse-subscription-period mobile users. 13(4):783–796.

[36] Adrian Perrig, Ran Canetti, Dawn Song, and J. D. Tygar. Efficient and secure source authentication for multicast. In *In Network and Distributed System Security Symposium, NDSS '01*, pages 35–46.

[37] Adrian Perrig, Ran Canetti, J. D. Tygar, and Dawn Song. *The TESLA Broadcast Authentication Protocol*.

[38] Adrian Perrig, Dawn Song, and J. D. Tygar. ELK, a new protocol for efficient large-group key distribution. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, SP '01, pages 247–. IEEE Computer Society.

[39] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. In *Wireless Networks*, pages 189–199.

[40] M. Piórkowski, M. Raya, A. Lezama Lugo, P. Papadimitratos, M. Grossglauser, and J.-P. Hubaux. TraNS: Realistic joint traffic and network simulator for VANETs. 12(1):31–33.

[41] F. Qu, Z. Wu, F. Y. Wang, and W. Cho. A security and privacy review of VANETs. 16(6):2985–2996.

[42] Sandro Rafaeli and David Hutchison. A survey of key management for secure group communication. 35(3):309–329.

[43] N. Renugadevi, G. Swaminathan, and A. S. Kumar. Key management schemes for secure group communication in wireless networks - a survey. In *2014 International Conference on Contemporary Computing and Informatics (IC3I)*, pages 446–450.

[44] P. Sakarindr and N. Ansari. Survey of security services on group communications. 4(4):258–272.

[45] Alan T. Sherman and David A. McGrew. Key establishment in large dynamic groups using one-way function trees. 29(5):444–458.

[46] Neha Singh, Saurabh Singh, Naveen Kumar, and Rakesh Kumar. Key management techniques for securing MANET. In *Proceedings of the ACM Symposium on Women in Research 2016*, WIR '16, pages 77–80. ACM.

[47] Christoph Sommer, Zheng Yao, Reinhard German, and Falko Dressler. On the need for bidirectional coupling of road traffic microsimulation and network simulation. In *Proceedings of the 1st ACM SIGMOBILE Workshop on Mobility Models*, MobilityModels '08, pages 41–48. ACM.

[48] M. Steiner, G. Tsudik, M. Waidner, Michael Steiner, Gene Tsudik, and Michael Waidner. *CLIQUES: A New Approach to Group Key Agreement.*

[49] Ahren Studer, Fan Bai, Bhargav Bellur, and Adrian Perrig. *Flexible, Extensible, and Efficient VANET Authentication.*

[50] András Varga and Rudolf Hornig. An overview of the OMNeT++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pages 60:1–60:10. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[51] A. A. Wagan and L. T. Jung. Security framework for low latency vanet applications. In *2014 International Conference on Computer and Information Sciences (ICCOINS)*, pages 1–6.

[52] D. Wallner, E. Harder, and R. Agee. Key management for multicast: Issues and architectures.

[53] S.Y. Wang, C.L. Chou, and C.C. Lin. The design and implementation of the NCTUns network simulation engine. 15(1):57–81.

[54] Chung K Wong, Mohamed G. Gouda, and Simon S. Lam. Secure group communications using key graphs.

[55] Manisha Yadav, Karan Singh, and Ajay Shekhar Pandey. Key management in efficient and secure group communication. pages 196–203. IEEE.

[56] Yuh-Shyan Chen Yun-Wei Lin. Routing protocols in vehicular ad hoc networks: A survey and future perspectives. 26(3):913–932.

[57] Wen Tao Zhu. Optimizing the tree structure in secure multicast key management. 9(5):477–479.

[58] André Zúquete. *Segurança em Redes Informáticas*. FCA - Editora de Informática, Lda., 4a edição aumentada edition.