CHIST-ERA

User empowerment for SEcurity and privacy in Internet of Things

Frameworks Deployment

Deliverable number: D3.3

Version 1.0

Funded by the Future and Emerging Technologies (FET) CHIST-ERA programme of the European Union.

Project Acronym: USEIT
Project Full Title: User empowerment for SEcurity and privacy in Internet of Things
Call: 2015
Grant Number: 20CH21_167531
Project URL: http://useit.eu.org

Editor:	Antonio Skarmeta, Universidad de Murcia (UMU) – Spain
Deliverable nature:	Report
Dissemination level:	Public
Delivery Date:	July 2018
Authors:	Salvador Pérez, University of Murcia Jorge Gallego, University of Murcia Ramón Sánchez University of Murcia Pedro González-Gil, University of Murcia Antonio Skarmeta, University of Murcia Nouha Oualha, CEA Alexis Olivereau, CEA

Abstract

This document describes the deployment of the C-ITS and Smart Objects frameworks on IoT-enabled environments. On the one hand, the C-ITS framework is deployed on different scenarios with V2V communications, in particular, a city, a highway and a national road. In this sense, a series of simulations focused on the Packet Delivery Ratio (PDR) and vehicle density have been performed. On the other hand, the Smart Objects framework is integrated on a smart building scenario with the aim of protecting large amounts of sensible data in a efficient and flexible way. Furthermore, an extension of this framework is also proposed in order to provide an intrusion detection and response mechanism in this smart building use case.

Contents

List of Figures

Executive Summary

USEIT wants to use the experience on previous results from projects, such as ABC4Trust, FutureID, Sociotal, Smartie, ITTSv6, and others, and focus on the identification of the security and privacy components needed to extend and support the objectives of USEIT vision.

List of Acronyms

SCIM	System for Cross-domain Identity Management
NGSI	Next Generation Service Interfaces

1 Introduction

The purpose of the present deliverable is to carry out the deployment of the proposed C-ITS and Smart Objects frameworks on IoT-enabled scenarios. Towards this end, the deliverable will firstly focus on describing the integration of both frameworks, which were introduced in D3.2, over real use cases (see D1.3). Subsequently, it will delve into the main interactions performed between the instantiated components and the required FI-WARE enablers (they were pointed in D3.1), identifying the corresponding messages and processes in each case to carry out their functionality.

1.1 Related deliverables

This deliverable considers other work provided in the following deliverables:

- D1.3 describes different real use cases in which the proposed frameworks may be deployed, thus leveraging their advantages.
- D3.1 offers a description about the main FI-WARE enablers used in USEIT to enable security and privacy.
- D3.2 define the C-ITS and Smart objects frameworks, identifying the main entities and phases defined to carry out their functionality.

1.2 Deliverable outline

The document consists of two technical chapters:

- Chapter 2 examines the C-ITS framework deployment through a series of simulations in terms of Packet Delivery (PDR) Ratio and vehicle density.
- Chapter 3 focus on the integration of the Smart object framework on a real IoT use case, by considering both the components of such framework and the required FI-WARE enablers. Then, it details the main interactions between such components to fulfill the use case functionality. This chapter presents also an extension of the propose smart building use case that includes an intrusion detection and response system.

2 C-ITS framework deployment

In this chapter, the deployment of the C-ITS framework is examined. Towards this end, we use Omnet++ [1] together with the Veins open source vehicular network simulation framework [2]. Three different scenarios are defined, specifically, a city, a highway and a national road, in order to study the Packet Delivery Ratio (PDR) evolution in correlation with the car density on the roads. Other key factors that need to be taken into account are the speed of the vehicles and the impact of buildings that interfere in Line of Sight (LOS) communications. City simulations will be the ones that will evaluate this latter case, whereas highway and road simulations will be oriented to the study of the speed impact.

2.1 Simulation parameters

The simulations are based on a principal node that follows a defined route and interacts with the rest of the vehicles. The main node sends Request broadcast messages every 20 secs (tunable), when the others cars receive this packet, they answer with a Reply message. The message type used is the Wave Short Message (WSM) [3], its format consists of a variable-length header followed by a variable length payload, as shown in Figure 2.1. At the moment, these messages only contain an ID and the message type in the payload field.

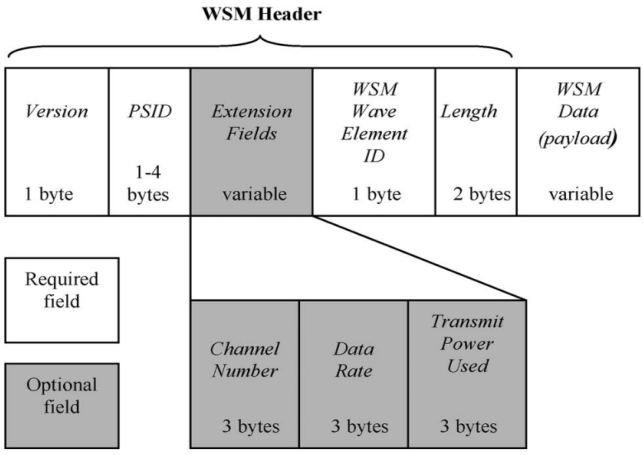

Figure 2.1: Wave Short Messages format

The protocol used is 802.11p, and its parameters inside the simulation are:

- TxPower: 20 mW.
- Bitrate: 6 Mbps.
- Sensitivity: -89 dBm.
- Thermal noise: -110 dBm.
- Antenna type: monopole located on the roof.

It is important to notice that the simulation time changes because the main node may leave the scenario earlier, or later, depending on the traffic. Therefore, the number of request sent may also change.

2.2 City

The simulation takes place in the city of Erlangen (Germany), particularly, in an area that measures approx. 2500m x 2500m. It is based on the route shown in Figure 2.2, followed by a car. The route does not change, in order to control this main node. The other cars move around the scenario randomly in each run of the simulation.

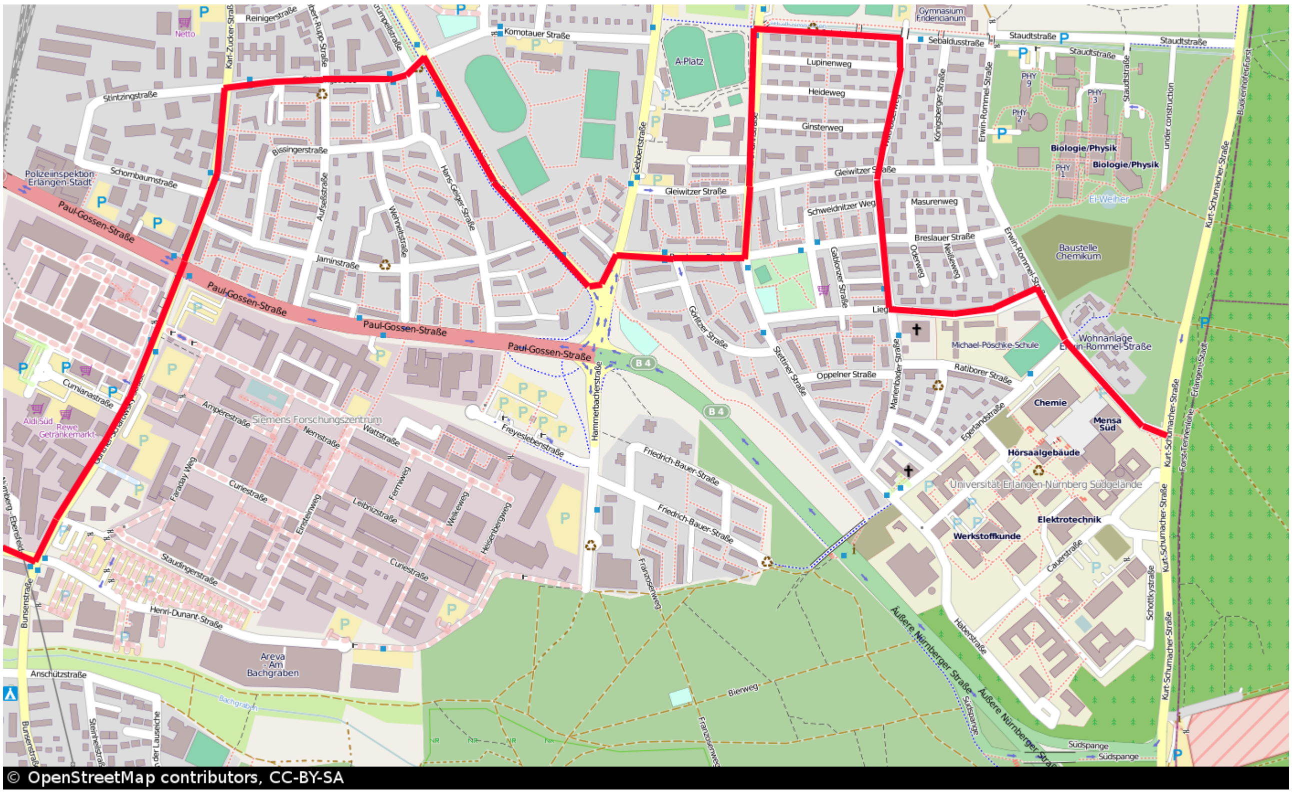

Figure 2.2: Route followed by the main car

Six different scenarios have been defined, varying cars density (n^{o} of nodes):

- 100
- 250
- 500
- 650
- 800
- 1000

For each scenario, 20 runs with different car routes have been simulated. In the following, the average results are presented. Figure 2.3 shows the average PDR and corresponding Confidence Intervals (95%) for each scenario.

In Table 2.1 the results of the different simulations are shown. The collected data is the duration (s) of the route, the number of requests sent, the number of requests that have received at least one ack, the PDR (%), and the confidence interval (95%) of the PDR.

In Figures 2.4 to 2.9, the histograms show the frequency of the number of acks received for a Request message. It collects information from the 20 runs in each scenario.

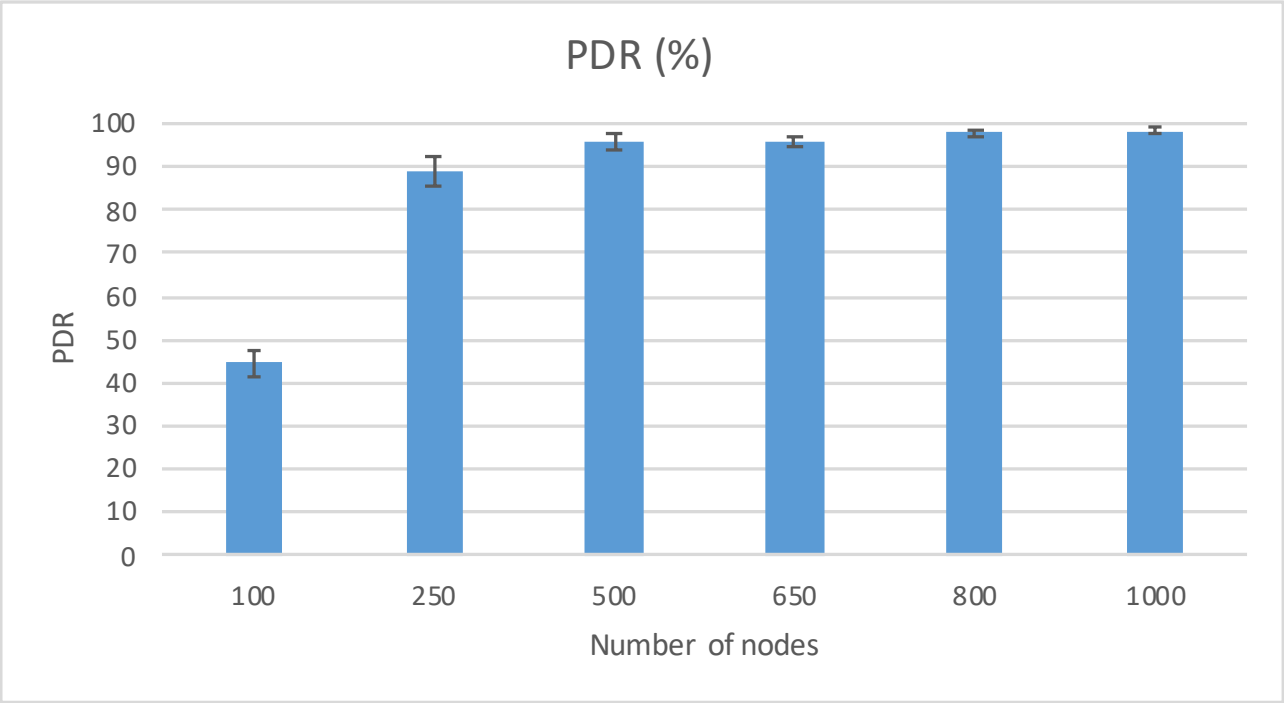

Figure 2.3: Average PDR for each scenario in the city simulation

Node number	Time (s)	Req sent	Req confirmed	PDR (%)	Conf int
100	426.6	21	9.4	44.8	3.06
250	467.9	22.95	20.4	88.9	3.34
500	1019.3	50.5	48.45	95.9	1.86
650	1148.65	56.85	54.6	96	1.01
800	1576.7	78.2	76.55	97.9	0.69
1000	1953.3	97.25	95.55	98.3	0.84

Table 2.1: Results of the city simulation for each scenario

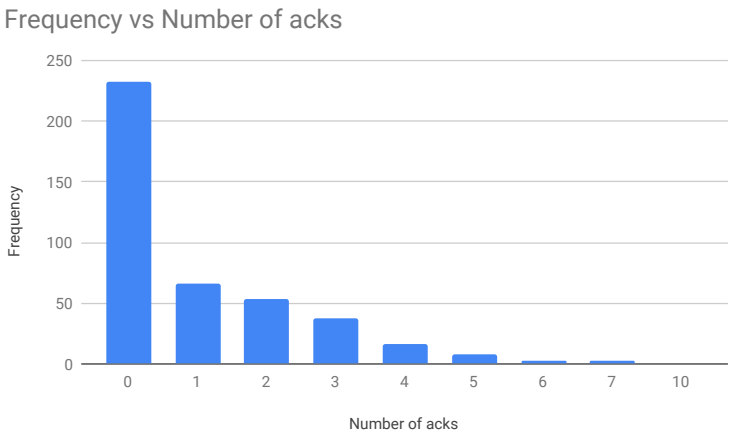

Figure 2.4: 100 nodes histogram

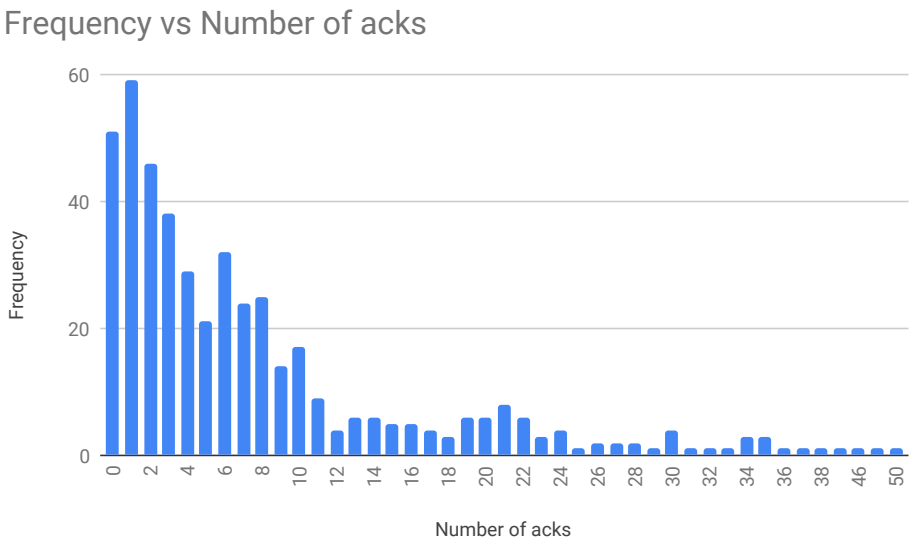

Figure 2.5: 250 nodes histogram

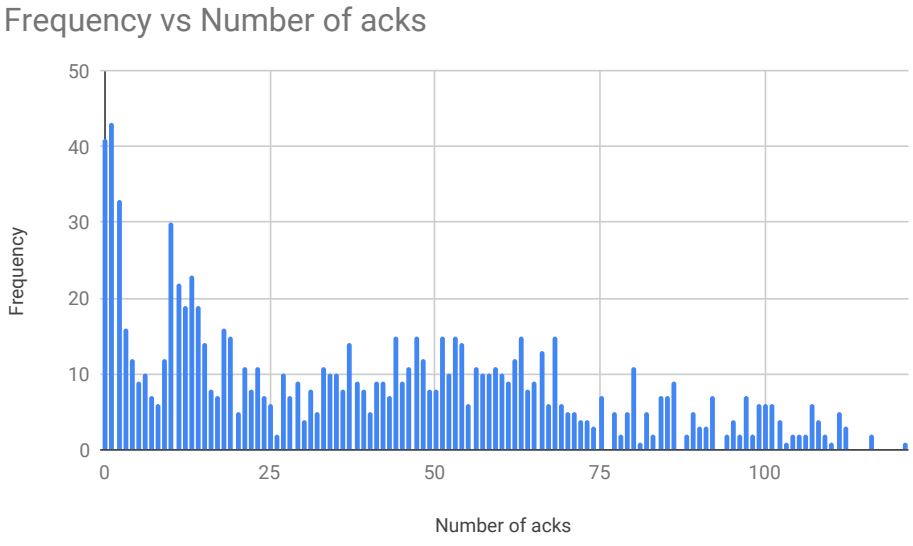

Figure 2.6: 500 nodes histogram

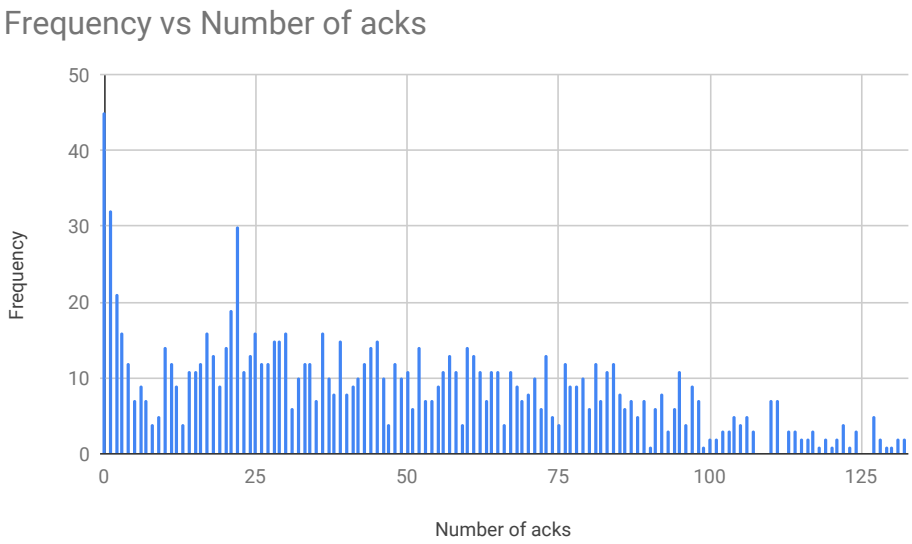

Figure 2.7: 650 nodes histogram

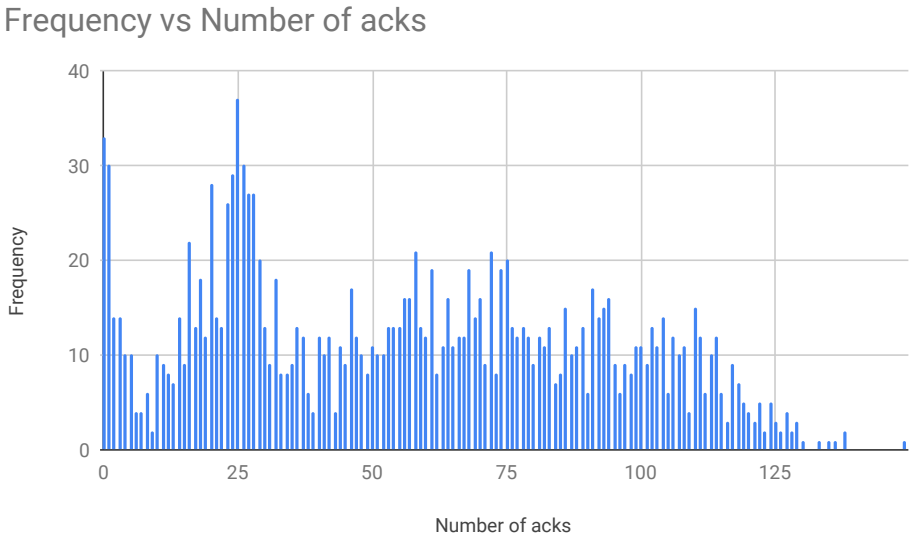

Figure 2.8: 800 nodes histogram

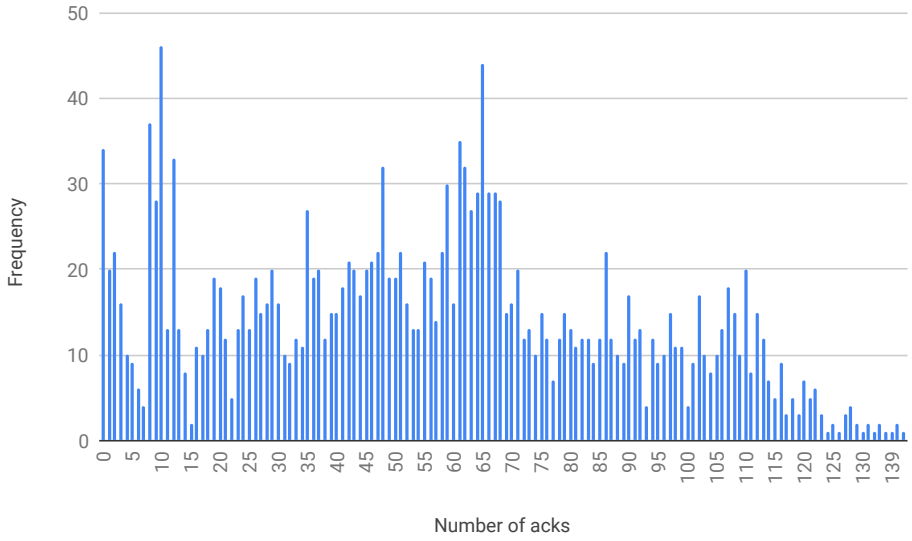

Figure 2.9: 1000 nodes histogram

2.3 Highway

For this simulation, a highway section has been created with the SUMO-GUI. It is 20 km long and has 3 intersections, separated by 6 km. The highway has two lanes each way. In Figure 2.10 can be seen a detail of the intersections. The maximum permissible speed is 120 km/h or 74.5 mph. The simulation is based on a main node (a car) that travels from the right side to the left side of the section. The other cars move around the scenario randomly in each run of the simulation.

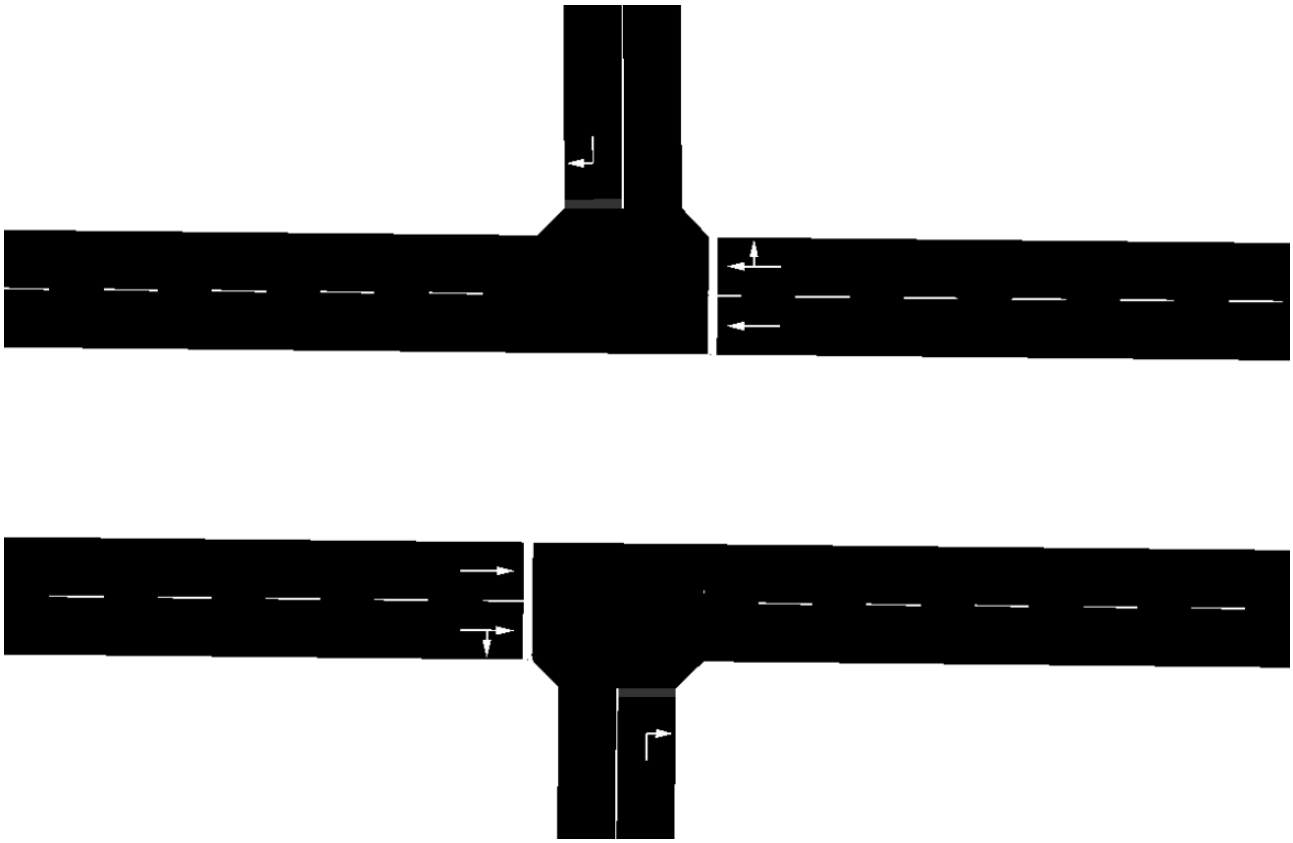

Figure 2.10: Detail of the intersections

Six different scenarios have been defined, varying cars density (n^o of nodes):

- 100
- 250
- 350
- 450
- 500
- 1000

For each scenario, 20 runs with different car routes have been simulated. In the following, the average results are presented. Figure 2.11 shows the average PDR and corresponding Confidence Intervals (95%) for each scenario.

In Table 2.2 the results of the different simulations are shown. The collected data is the duration (s) of the route, the number of requests sent, the number of requests that have received at least one ack, the PDR (%), and the confidence interval (95%) of the PDR.

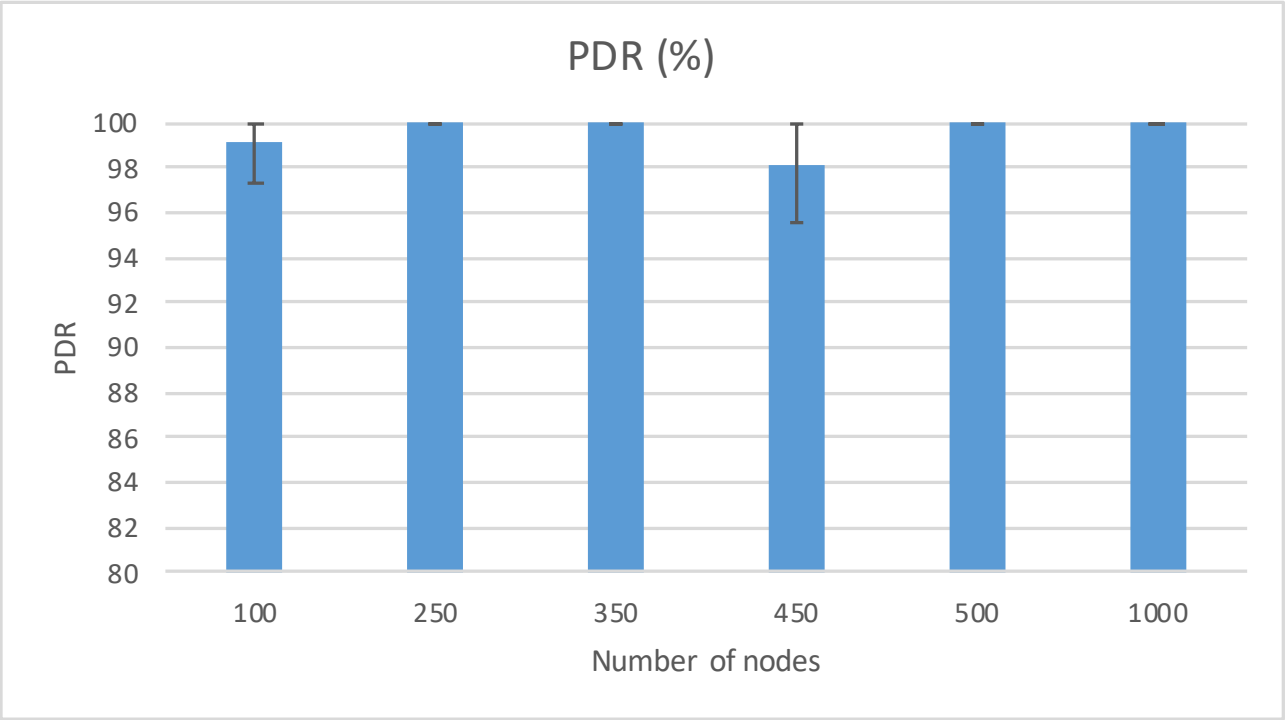

Figure 2.11: Average PDR for each scenario in the highway simulation

Node number	Time (s)	Req sent	Req confirmed	PDR (%)	Conf int
100	597	29	28.75	99.1	1.69
250	597	29	29	100	0
350	597.05	29	29	100	0
450	597.3	29	28.45	98.1	2.57
500	596.95	29	29	100	0
1000	597	29	29	100	0

Table 2.2: Results of the highway simulation for each scenario

In Figures 2.12 to 2.17, the histograms show the frequency of the number of acks received for a Request message. It collects information from the 20 runs in each scenario.

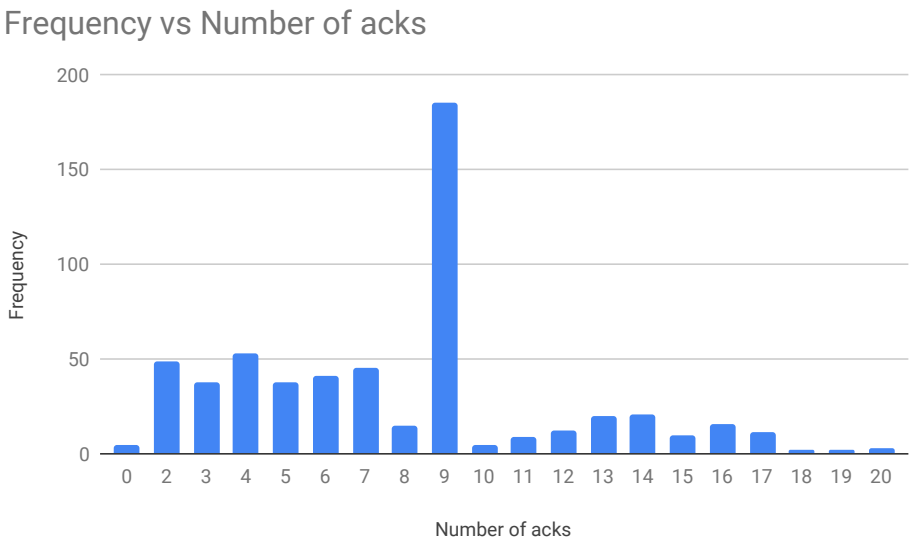

Figure 2.12: 100 nodes histogram

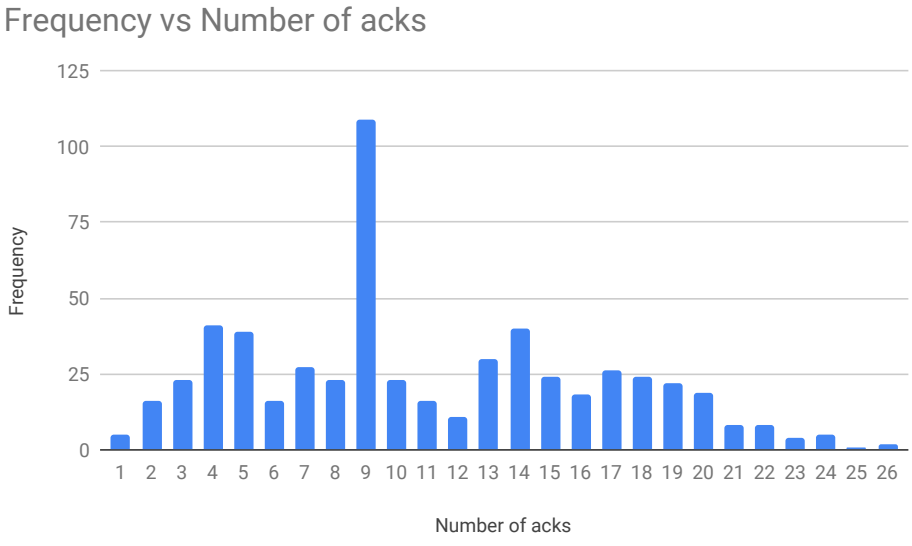

Figure 2.13: 250 nodes histogram

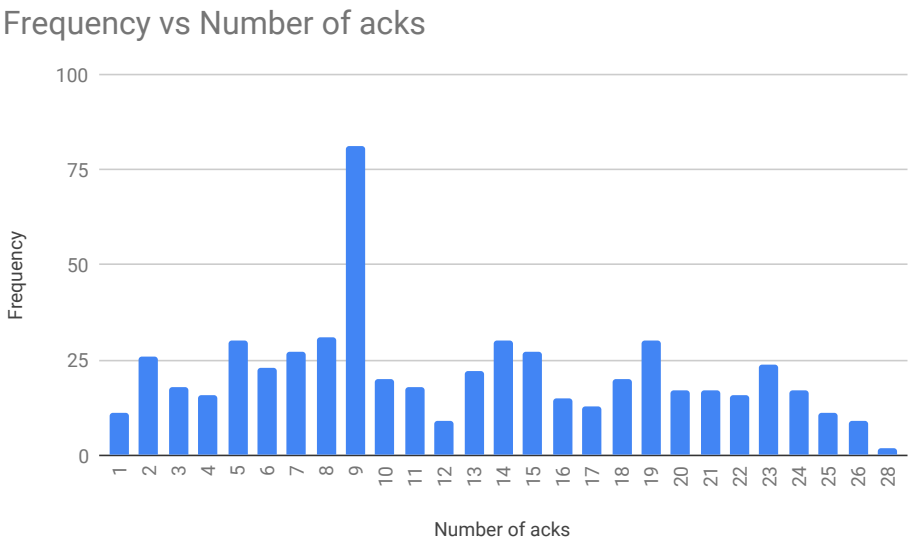

Figure 2.14: 350 nodes histogram

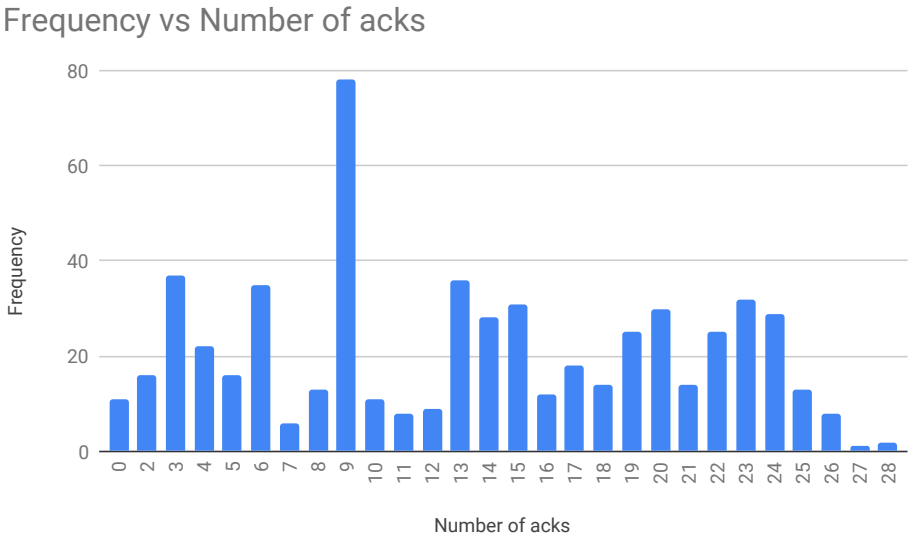

Figure 2.15: 450 nodes histogram

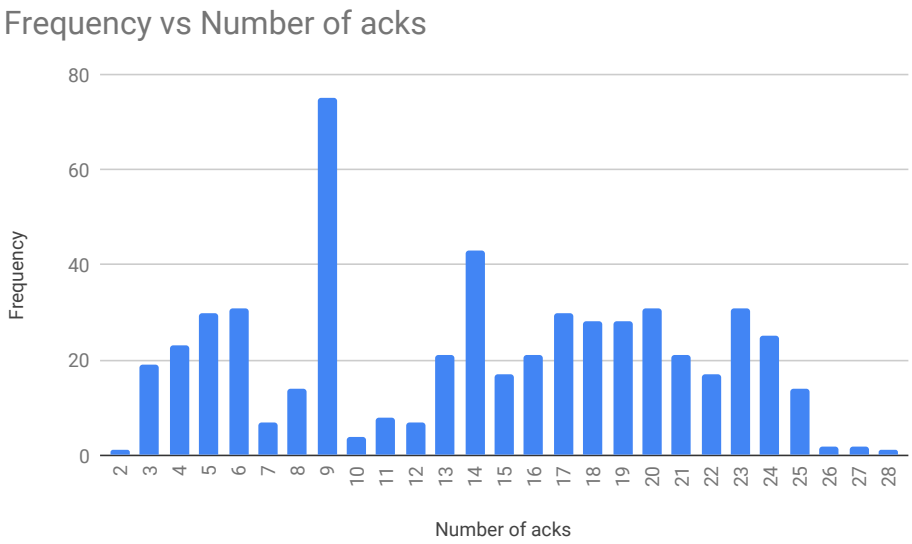

Figure 2.16: 500 nodes histogram

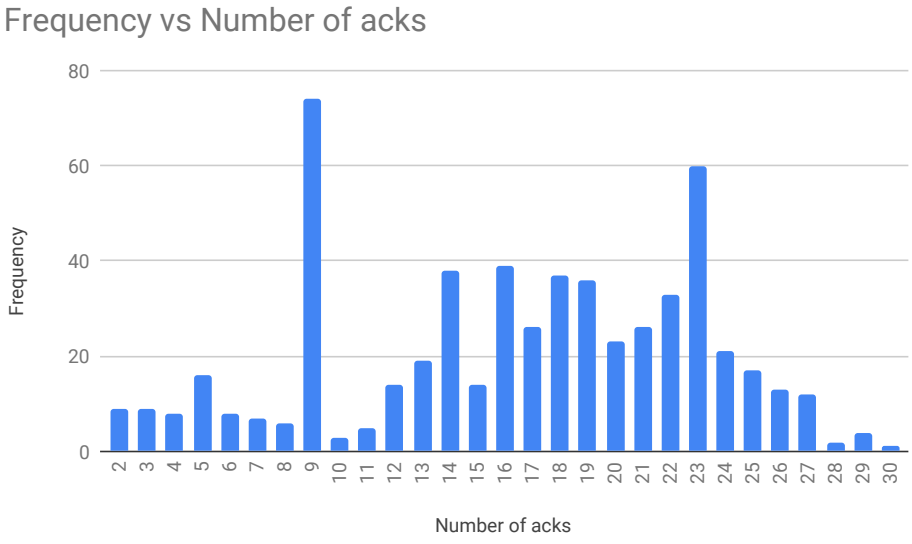

Figure 2.17: 1000 nodes histogram

2.4 National

For this simulation, a national route section has been created with the SUMO-GUI. It is 20 km long and has 7 intersections, separated by 3 km. The national has two lanes each way. In Figure 2.18 can be seen a detail of the intersections. The maximum permissible speed is 100 km/h or 62 mph. The simulation is based on a main node (a car) that travels from the right side to the left side of the section. The other cars move around the scenario randomly in each run of the simulation.

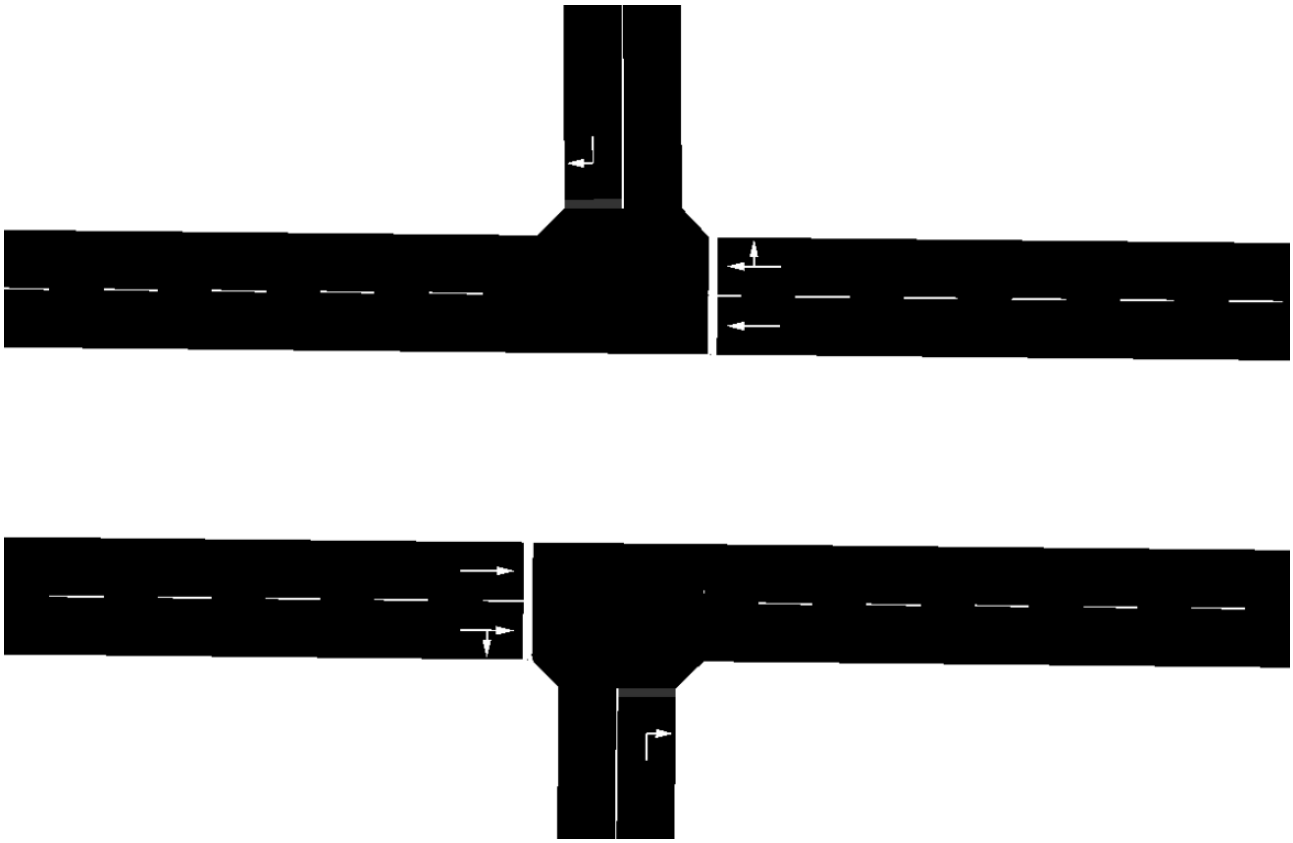

Figure 2.18: Detail of the intersections

Six different scenarios have been defined, varying cars density (n^o of nodes):

- 100
- 250
- 350
- 450
- 500
- 1000

For each scenario, 20 runs with different car routes have been simulated. In the following, the average results are presented. Figure 2.19 shows the average PDR and corresponding Confidence Intervals (95%) for each scenario.

In Table 2.3 the results of the different simulations are shown. The collected data is the duration (s) of the route, the number of requests sent, the number of requests that have received at least one ack, the PDR (%), and the confidence interval (95%) of the PDR.

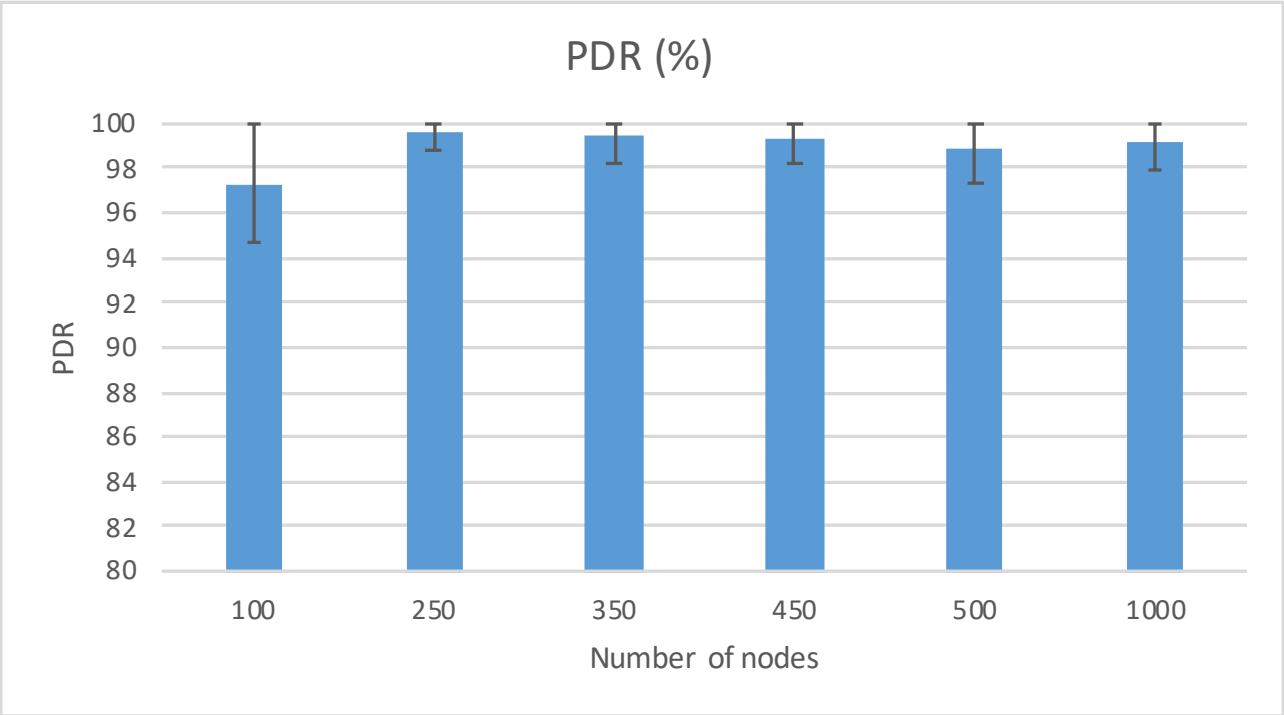

Figure 2.19: Average PDR for each scenario in the national simulation

Node number	Time (s)	Req sent	Req confirmed	PDR (%)	Conf int
100	713.1	35	34.05	97.3	2.55
250	713.25	35	34.85	99.6	0.84
350	713.15	35	34.8	99.4	1.12
450	713.05	35	34.75	99.3	1.14
500	713.15	35	34.6	98.9	1.54
1000	713.35	35	34.7	99.1	1.16

Table 2.3: Results of the national simulation for each scenario

In Figures 2.20 to 2.25, the histograms show the frequency of the number of acks received for a Request message. It collects information from the 20 runs in each scenario.

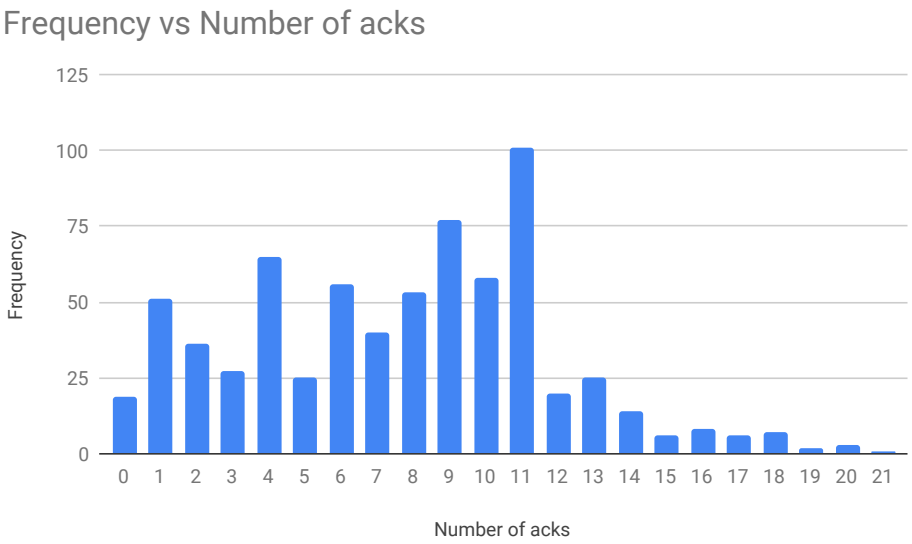

Figure 2.20: 100 nodes histogram

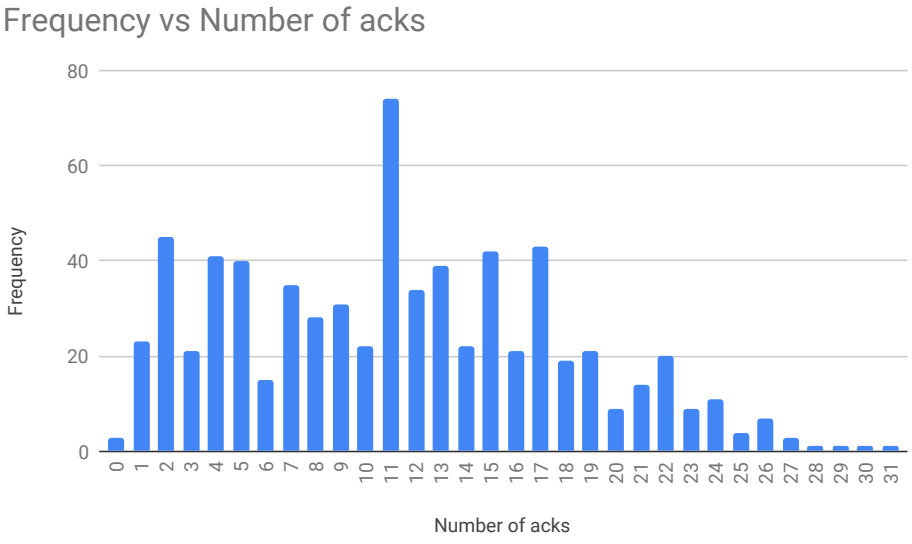

Figure 2.21: 250 nodes histogram

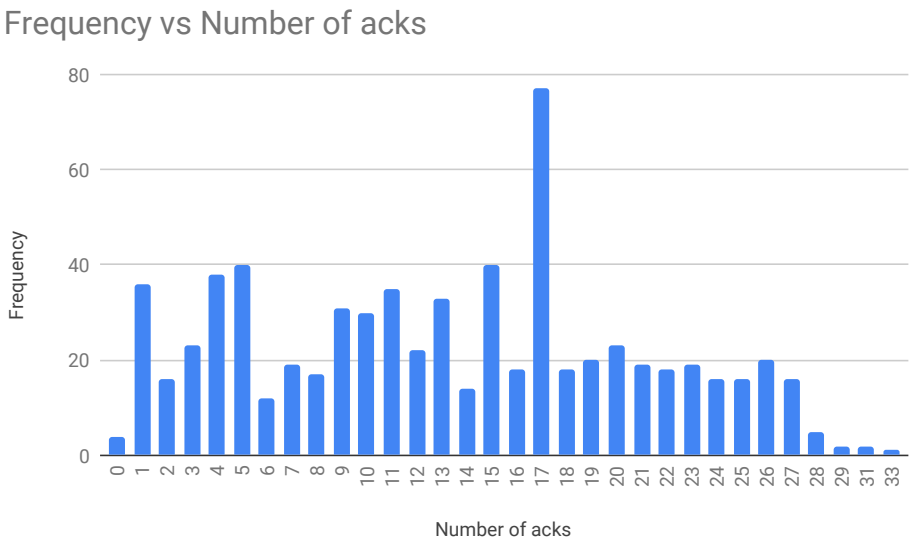

Figure 2.22: 350 nodes histogram

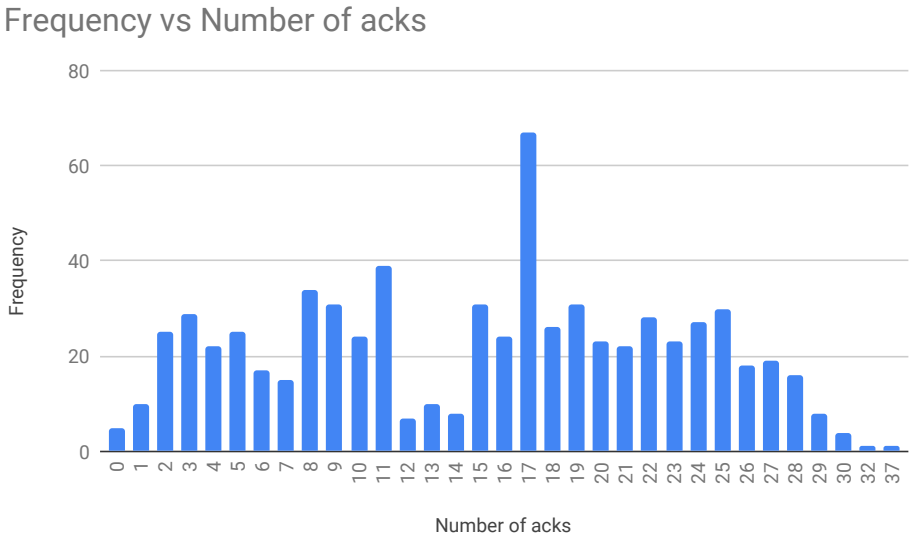

Figure 2.23: 450 nodes histogram

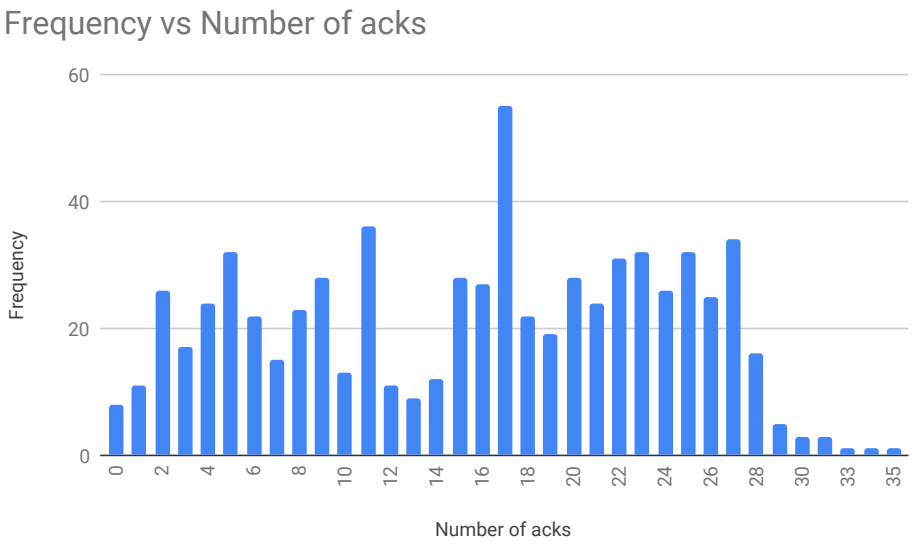

Figure 2.24: 500 nodes histogram

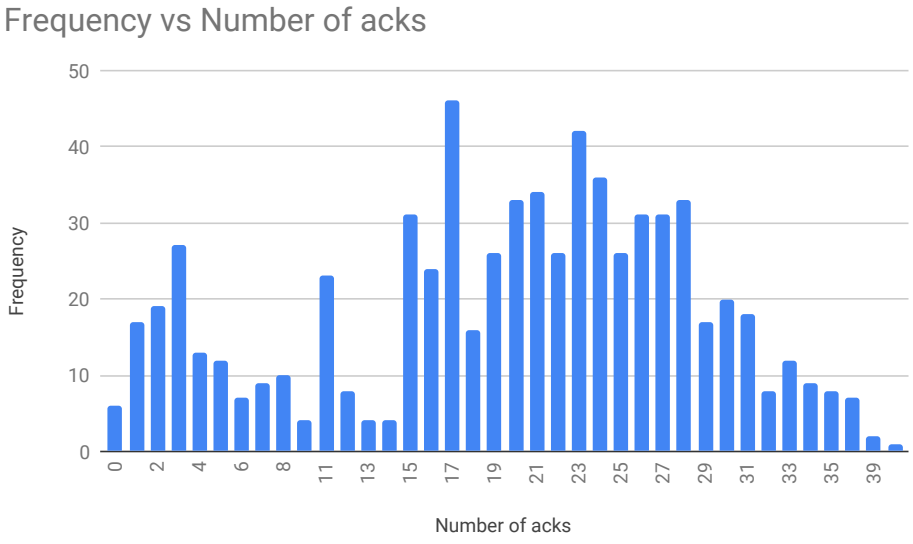

Figure 2.25: 1000 nodes histogram

2.5 Results

It can be seen that a 100% PDR is never achieved in the city simulation. The high number of buildings in the map blocks the communications even in the scenarios with high car density.

In the highway simulations, 100% PDR is easily achieved due to the fact that there is no obstacles between the different vehicles of the simulations. However, in the 100 and 450 nodes scenarios, and in the national road simulations, the 100% PDR is not achieved because the random route generation scheduled some zones in which there were no other cars in the surroundings of the main node. Furthermore, in the national roads there are more entries and exits than in the highways, thus the vehicles spend less time in the scenario and the principal vehicle encounters a fewer number of cars in its way.

In consequence, it can be seen that in order to achieve a 100% PDR in an urban scenario, the vehicle density has to be higher than in a highway or a national road. This is because the impact of the obstacles in the communications, buildings in this case, is higher than the effect of the speed.

3 Smart objects framework deployment

This chapter is intended to show the integration of the Smart objects framework on the *Smart Building* real use case, which has been already introduced, to protect and share large amounts of data. Towards this end, we deploy each framework component on different devices and FI-WARE enablers in order to provide certain functionality required by our solution. In this sense, it should be pointed out that the enablers employed in this case are the Orion Context Broker and the Keyrock IdM, as previously mentioned. Subsequently, we delve into the main phases of the Smart objects framework, that is, *Phase 1 - Symmetric key establishment*, *Phase 2 - Symmetric key encryption and storage*, *Phase 3 - Encrypted data event publication* and *Phase 4 - Encrypted data event retrieval*, and describe interactions performed between the instantiated components and FI-WARE enablers with the aim of offering a more comprehensive view about functionality of this framework.

3.1 Integration on the *Smart Building* use case

From the introduced *Smart Building* use case, we carry out the integration of the main entities identified in the Smart objects framework by deploying each of them on a different device or FI-WARE enabler, as show in Figure 3.1. Accordingly, when the *Gateway* device receives data from *Data Sources* (step 0.1), it contacts the *ABE Service* (*ABES*) to establish a symmetric key (step 1). Such key has an associated lifetime, so that the *Gateway* will use it to protect incoming data of the same type (e.g., related to users) as long as the key is not expired; in case of key's expiry, a new key will be established.

Then, the *Gateway* acts as *CP-ABE Delegator* entity and provides a CP-ABE access policy to the *ABES* (playing the *CP-ABE Assistant* role), in order to protect such key (step 2.1). For instance, when a user accesses the building and a RFID reading occurs , the *Gateway* could send the policy {*role="building_administrator" or role="smart_service"*}, so that only the building administrator and the smart service would be able to access certain sensitive user's data, such as its identifier. Subsequently, when the *ABES* receives the corresponding access policy, it encrypts the symmetric key by the CP-ABE scheme and stores it on the *Symmetric Key Database*, acting as the *KSS* entity (step 2.2). It should be pointed out that previous steps will be only performed when a new symmetric key needs to be established; otherwise, the *Gateway* encrypts incoming data by using the established symmetric key, and generates a new event including such encrypted information. In particular, when data come from a RFID reading, it obtains the RFID reader's location (step 0.2), as well as the user's identifier and mobility condition from the *Keyrock IdM* (step 0.3). To this end, the *Gateway* makes use of the System for Cross-domain Identity Management (SCIM) 2.0 [4] and Identity API v3 interfaces provided by this FI-WARE enabler. Then, the *Gateway* only encrypts the user's identifier, which is included with the RFID reader's location and the user's mobility in the event. This way, these unprotected data are used by the emergency service, while the user's identifier is kept protected, thus preserving users' privacy (this service does not know who user is).

Once the event has been generated, it is published on the *Orion Context Broker* (as *ESS* (step 3)) by using the OMA Next Generation Service Interfaces (NGSI) [5]. Then, this enabler forwards the event to those *Services* (as *Applications*) previously subscribed on its type (step 4.1). Subsequently, *Services* request the CP-ABE encrypted symmetric key from the *Symmetric Key Database* (step 4.2) and try to decrypt it with their CP-ABE private keys. If the decryption process is successful, the corresponding service will be able to retrieve data by using the decrypted symmetric key. Particularly, while RFID reader's location and user's mobility will be accessible for all subscribed services, only the building administrator and the smart service will be able to access the user's identifier.

So far, we have shown the integration of the Smart objects framework on the *Smart Building* use case to properly

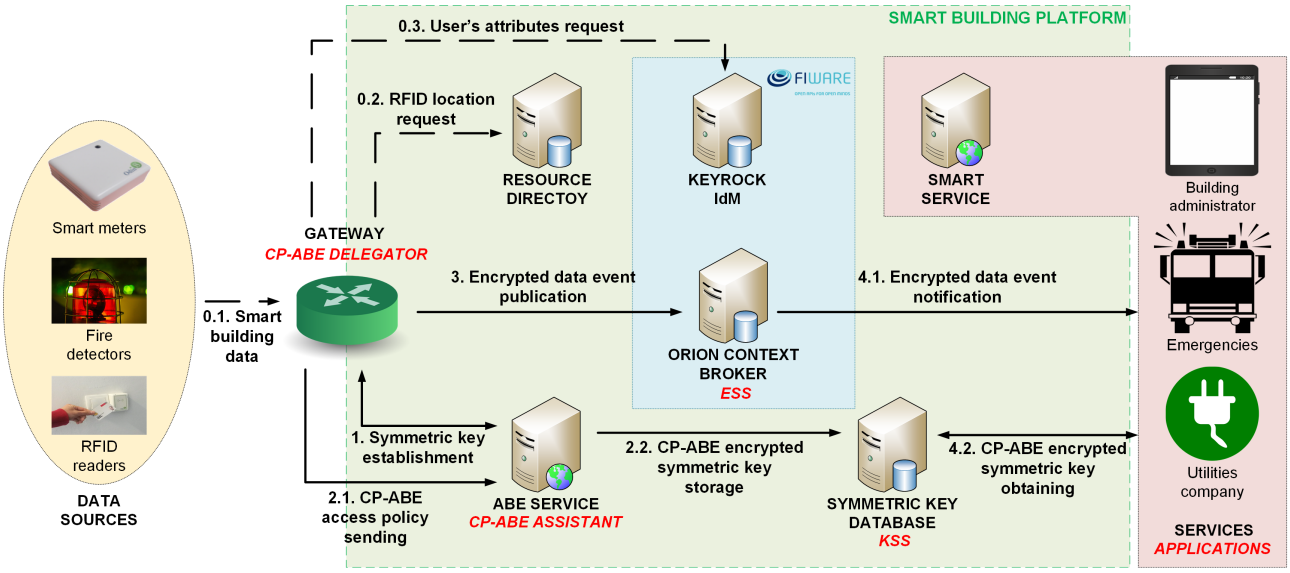

Figure 3.1: Integration of the Smart objects framework on the *Smart Building* use case

protect data coming from different data sources in an efficient and flexible way. Next section provides a detailed description of the main messages and processes required by our solution to fulfill with their functionality.

3.2 Interactions

In this section, we delve into the interactions performed by the involved devices and FI-WARE enablers in the use case previously described, particularly, when a RFID reading occurs. As already mentioned, note that we assume that the *Gateway* and the corresponding *Service* have obtained the CP-ABE public parameters, in order to perform CP-ABE cryptographic operations. Similarly, the *Service* has gotten its corresponding CP-ABE private key through communication with an *Attribute Authority* entity, as described in [6, 7]. In addition, we have also considered that the *Service* is subscribed to the *Orion Context Broker* to be notified about any event referring to users' data. Taking account these premises, Figure 3.2 shows a set of previous steps that are required before to launch our solution. Specifically, when the *Gateway* receives information about a RFID reading (*step 0.1.a.*), it requests and gets the location of the corresponding reader ($reader\_location$) from the *Resource Directory* by using the reader identifier (*steps 0.2.a.* and *0.2.b.*). Similarly, the *Gateway* also requests and obtains both the user's id ($user\_id$) and the user's mobility condition ($user\_mobility$) from the *Keyrock* by using the RFID card identifier (steps *0.3.a.* and *0.3.b.*). It should be pointed out that payload including in the response message (0.3.b.) follows a specific format, similar to the one shown in Listing 3.1. Finally, the *Gateway* confirms RFID reading data reception to the corresponding reader (*step 0.1.b.*).

Listing 3.1: *Keyrock* response payload for a user attribute request

```
{
    "user": {
        "user_id": "2d6f5391-6130-48d8-a9d0-01f20699a7e",
        "user_name": "User1",
        "user_mobility": "reduced-mobility"
    }
}
```

- *user_id* unequivocally identifies the corresponding user.
- *user_name* is the user's name.
- *user_mobility* indicates the user's mobility condition, that is, if it is a user with reduced mobility or not.

Once the previous steps are performed, our proposal starts, as shown in Figure 3.3.

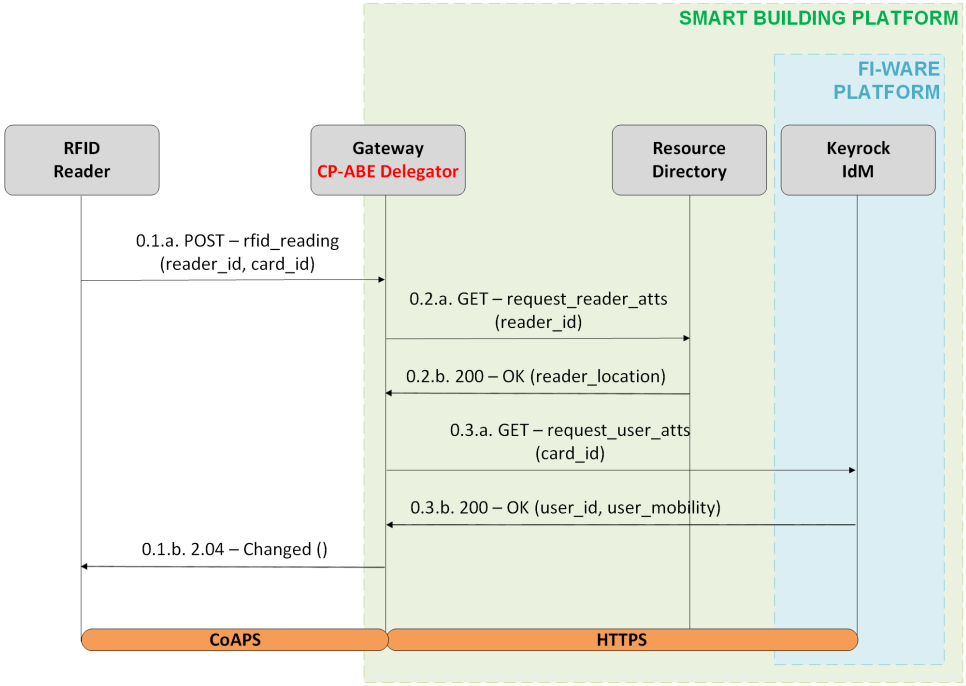

Figure 3.2: Previous interactions to the launch of our proposal on the *Smart Building* use case

During the first phase, the *Gateway* and the *ABES* establish a symmetric key (SYMK). For this purpose, the Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) algorithm [8] is used. By using the ephemeral version of DH, the establishment of a new symmetric key will require a new key pair, thereby increasing the untraceability of the encrypted data flow. Thus, the *Gateway* firstly generates an ephemeral elliptic curve key pair by using a specific elliptic curve (e.g., NIST P-256). Then, it includes the public key and the selected curve into a GW_{EPK} structure, which is sent to the *ABES* (step 1.1). Listing 3.2 shows a GW_{EPK} example following the format specified by JWA [9]. Such information will be used by the *ABES* to set the parameters to be used for the ECDHE algorithm.

Listing 3.2: Example of ephemeral public key information

```
{
    "alg": "ECDH-ES",
    "enc": "A128GCM",
    "apu": "QWxpY2U",
    "apv": "Qm9i",
    "epk":
    {
        "kty": "EC",
        "crv": "P-256",
        "x": "gI0GAILBdu7T53akrFmMyGcsF3n5dO7MmwNBHKW5SV0",
        "y": "SLW_xSffzlPWrHEVI30DHM_4egVwt3NQqeUD7nMFpps"
    }
}
```

- *alg* indicates the algorithm to generate SYMK (ECDHE).
- *enc* specifies the algorithm that will be used to encrypt data (AES GCM with a 128-bit key).
- *apu* and *apv* contain the *Gateway* and *ABES* identifiers, encoded as a base64url string.
- *epk* is the ephemeral public key represented as a JWK [10].
  - *kty* identifies the key type (EC).
  - *crv* that specifies the elliptic curve (P-256)
  - *x* and *y* parameters contain the EC point coordinates encoded in base64url.

When the *ABES* receives this message, it generates its own ephemeral elliptic curve key pair according to

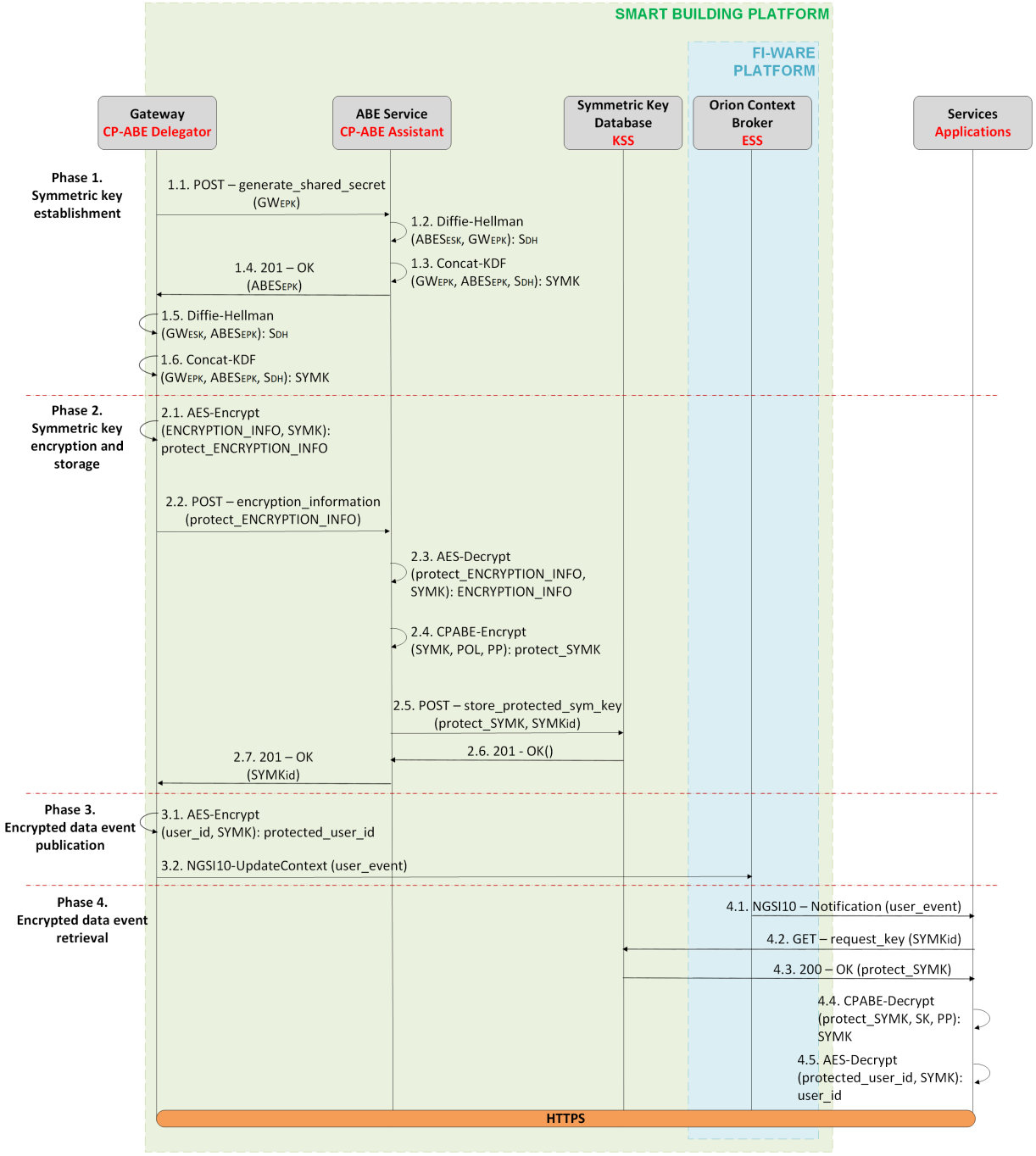

Figure 3.3: Smart objects framework interactions for the *Smart Building* use case

the value of *crv*. Then, it runs the ECDHE algorithm to calculate a shared secret (S_{DH}) with the *Gateway* (step 1.2) by using its ephemeral private key ($ABES_{ESK}$) and the GW_{EPK}. To enhance the strength of the shared symmetric key generation process, we have adopted the Concatenation Key Derivation Function (Concat-KDF) [11] to derive the SYMK from the S_{DH}. This function uses the GW_{EPK}, the *ABES* ephemeral public key ($ABES_{EPK}$) and the S_{DH} to generate SYMK (step 1.3). Then, the *ABES* sends the $ABES_{EPK}$ to the *Gateway*, following the example of Listing 3.2 (step 1.4). Upon receiving this message, the *Gateway* completes the ECDHE algorithm execution to obtain the S_{DH} (step 1.5) by using the $ABES_{EPK}$ and its ephemeral private key (GW_{ESK}). In addition, it executes the Concat-KDF function to derive the SYMK that will be shared by both entities (step 1.6).

The second phase focuses on protecting the computed SYMK by using a CP-ABE policy (POL). Towards this end, the *Gateway* includes POL into a ENCRYPTION_INFO structure. Listing 3.3 shows an example of this structure.

Listing 3.3: Example of information related to symmetric key encryption

```
{
    "timestamp": "2018-12-03T16:18:02Z",
    "device_id": "http://SmartBuilding/Gateway01",
    "policy":
    {
        "specs": "role=building_administrator OR role=smart_service",
        "metadata": [
        {
            "name": "CreationDate",
            "value": "2018-12-24T12:34:32Z",
            "type": "http://sensorml.com/ont/swe/property/DateTimeStamp"
        }
        ]
    }
}
```

- *timestamp* indicates when the message was generated according to ISO 8601 [12] format. By following this format, the interoperability between the *Gateway* and the *ABES* is facilitated.
- *device_id* identifies the *Gateway*.
- *policy* provides details about POL to be used to encrypt the SYMK.
    - *specs* represents the POL as a tree data structure, where leaf nodes correspond to the different attributes and intermediate nodes are the AND/OR logical operators.
    - *metadata* are a set of attributes providing additional information about the POL.

The ENCRYPTION_INFO is encrypted by using AES with the SYMK (step 2.1) and sent to the *ABES* (step 2.2). Upon receiving this message, the *ABES* decrypts the ENCRYPTION_INFO (step 2.3). Then, it executes the CP-ABE encryption operation with the provided POL to protect the SYMK (step 2.4). Furthermore, the *ABES* generates a unique key identifier associated with the protected SYMK ($SYMK_{id}$) that will be used by *Services* to get such SYMK at phase 4. Next, the *ABES* stores the protected SYMK and the $SYMK_{id}$ on the *Symmetric Key Database* (steps 2.5 and 2.6). Then, the *ABES* sends the $SYMK_{id}$ to the *Gateway* (step 2.7). This identifier is used by the latter at phase 3 to identify the SYMK that is employed to encrypt data of events. In addition, the *Gateway* establishes a limited lifetime for the SYMK ($SYMK_{lifetime}$). This way, when such $SYMK_{lifetime}$ expires, phase 1 should be performed again. Therefore, in case SYMK is obtained by an attacker, it will only be able to recover the data encrypted with such specific key. Furthermore, note that the $SYMK_{lifetime}$ is based on the number of published events in order to delimit the amount of data that could be accessed in an unauthorized way, regardless of the *Gateway* publication rate.

In the phase 3, the *Gateway* uses the SYMK to encrypt the $user\_id$ (step 3.1). Then, it creates a new event ($user\_event$) including the protected user identifier ($protected\_user\_id$) along with the RFID reader's location ($reader\_location$) and the user's mobility condition ($user\_mobility$). In addition, other parameters are included in such event, specifically, the $SYMK_{id}$, the *Gateway* identifier and a set of metadata. Note that we have defined an event as a structure that follows the format specified in Listing 3.4.

Listing 3.4: Event example with encrypted data related to the user identifier

```
{
    "device_id": "http://SmartBuilding/Gateway01",
    "symmetric_key_id": "541594b1-2f8d-431a-a5a4-666393e4adc4",
    "encrypted_data": {
        "vale": "Ewhbw9e2cpyGaa5XDdOUoA==",
        "metadata": [
        {
            "name": "Description",
            "value": "User identifier",
            "type": "urn:org-user:identifier"
        }
        ]
    }
    "unprotected_data": [
```

```
    {
            "name": "User's location",
            "value": "room1",
            "type": "urn:org-user:location"
    },
    {
            "name": "User's mobility condition",
            "value": "reduced-mobility",
            "type": "urn:org-user:mobility"
    }
    ]
}
```

- *device_id* is a URI that identifies the *Gateway*.
- *symmetric_key_id* unequivocally identifies the SYMK. This identifier is used by *Services* to retrieve such key from the *Symmetric Key Database*.
- *encrypted_data* contains the AES encrypted data as a base64url string. Additionally, it also includes a set of *metadata* that provide additional information about the encrypted data, such as its creation date or the description.
- *unprotected_data* contains user's data that do not require to be protected. In this case, the user's location and user's mobility condition.

Then, when the event is created, the *Gateway* includes it within an NGSI-10 UpdateContext message (Listing 3.5), which is published by this device on the *Orion Context Broker* (step 3.2).

Listing 3.5: Update message example including a user event

```
{
    "contextElements": [
    {
        "type": "urn:org-smartbuilding:user",
        "isPattern": "false",
        "id": "2d6f5391-6130-48d8-a9d0-01f20699a7e",
        "attributes": [
        {
            "name": "User event",
            "value": user_event,
            "type": "urn:org-smartbuilding:event",
            "metadatas": [
            {
                ...
            }
            ]
        }
        ]
    }
    ],
    "updateAction": "UPDATE"
}
```

- *contextElements* is a list of context elements, where each element is identified by the *id* parameter. Note that a context element contains a list of attributes associated with it. In this case, the context element is the specific user, while an attribute is the $user\_event$.
- *updateAction* specifies action to be performed over the context element, particularly, updating the user's event.

The last phase (phase 4) begins when the *Service* receives the $user\_event$ from the *Orion Context Broker* through a NGSI-10 Notification message, as shown in Listing 3.6 (step 4.1). Note that the format of this message is similar to the one employed in the NGSI-10 UpdateContex message.

Listing 3.6: Notification message example including a user event

```
{
    "subscriptionId": "2451c09ed714fb3b37d7d5a8",
    "originator": "http://SmartBuilding/OrionContextBroker",
    "contextResponses" : [
    {
        "contextElement":
        {
            "attributes": [
```

```
            {
                "name": "User event",
                "value": user_event,
                "type": "urn:org-smartbuilding:event",
                "metadatas": [
                {
                    ...
                }
                ]
            }
            ],
            "type": "urn:org-smartbuilding:user",
            "isPattern": "false",
            "id": "-",
        }
    }
    ]
}
```

Then, the *Service* performs a request to the *Symmetric Key Databases* with the $SYMK_{id}$ included in the $user\_event$ just received to get the corresponding protected SYMK (steps 4.2 and 4.3). At this point, the *Service* tries to decrypt such SYMK using its SK previously obtained. If its SK satisfies the POL that was used to encrypt the SYMK, this *Service* will be able to decrypt it (step 4.4) and, therefore, it will be able to retrieve the user's identifier ($user\_id$) by employing the AES algorithm (step 4.5).

3.3 Extended *smart building* use case

The basic smart building use case consists of sensors deployed in a building. A building administrator is responsible of setting up the access policies associated with sensor measurements and managing attributes and keys for data users. Upon request from users, sensors send their measurements CP-ABE-encrypted with the associated access policies to requestors through a gateway. In the extended smart building use case, an intrusion detection and response system is introduced to the basic use case (refer to figure 3.4). The added system is composed of a set of intrusion detection system (IDS) probes and an IDS manager responsible of managing IDS probes and providing intrusion responses by collaborating with the building manager and leveraging on CP-ABE encryption and re-encryption.

In the following, section 3.3.1 describes the interactions between the different actors involved in the extended smart building use case, and section 3.3.2 details the intrusion detection and response system proposed for the use case.

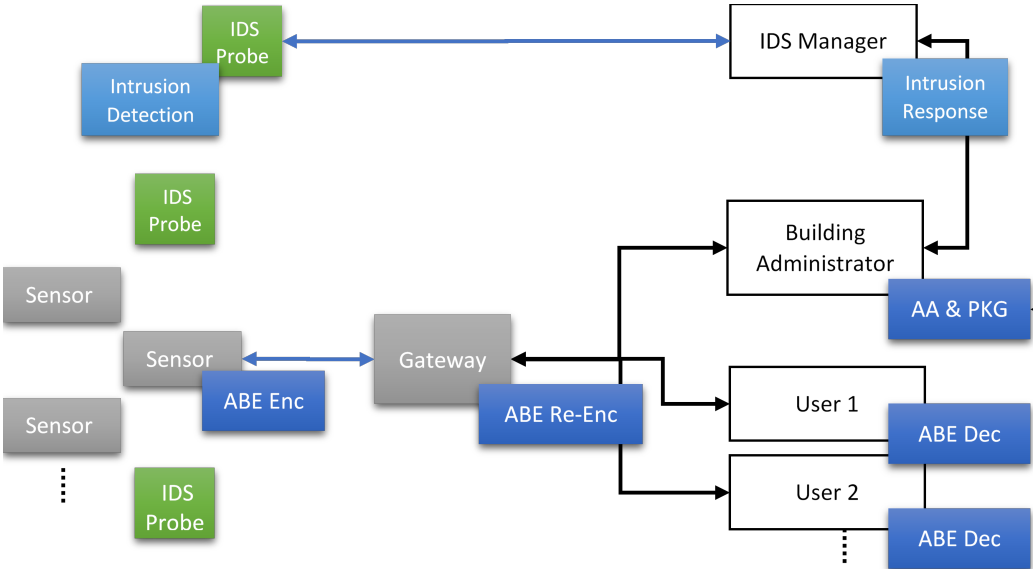

Figure 3.4: Extended smart building use case with IDS

3.3.1 Interactions

The communications of sensors deployed in the building are based on the CoAP protocol protected using the security profile of IEEE 802.15.4. On the other hand, the communications from the gateway to the building administrator and users are based on the client/server HTTP protocol. At setup, the building administrator sends ABE public parameters along with the current access policies associated with the sensor data to sensors deployed in the network (message 0 in figure 3.5). All this information is sent by the gateway using the PUT method of CoAP. Upon request from users using the GET method of CoAP (message 1.1 in figure 3.5), the sensors send measurements encrypted using CP-ABE through the gateway (message 1.2 in figure 3.5). To decrypt the received ciphertext, the users first request decryption keys from the building administrator if keys are not already provided beforehand (messages 1.3 and 1.4 in figure 3.5), and then decrypt the ciphertext.

The deployed IDS probes in the network inspect packets transmitted in the IoT network. If a probe detects an attack, it sends a message to the IDS manager (message 2.1 in figure 3.5). This latter is responsible of managing attack alerts and responses. It sends the adequate response to the detected attack in a request sent to the building administrator (message 2.2 in figure 3.5). The building administrator sends a request to the gateway to insert new IDS attributes to certain packets by performing CP-ABE re-encryption (message 2.3 in figure 3.5).

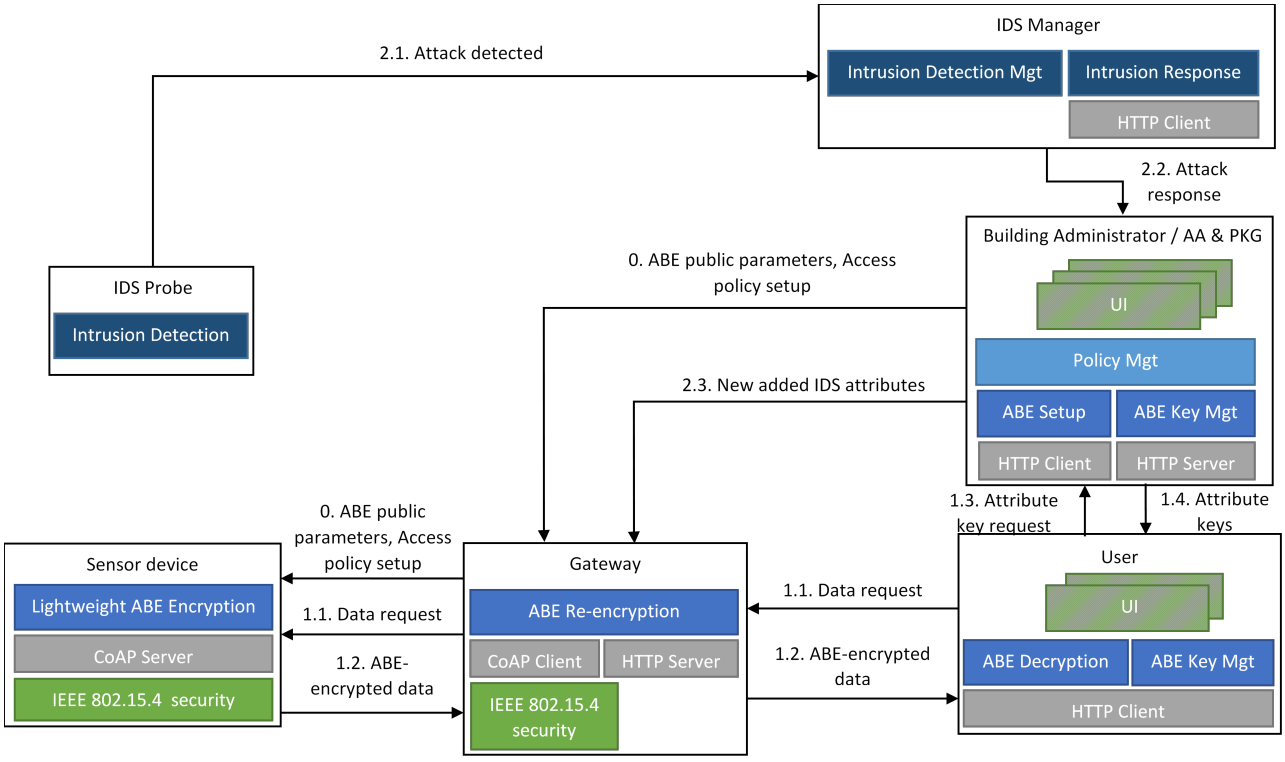

Figure 3.5: Actors and their interactions in the extended smart building use case

3.3.2 Detection and response to an attack

Encrypting data is essential to ensure confidentiality, but it is easier to detect malicious activity with access to all the sent data. While having access to meta-data can be enough to detect certain kinds of attacks (Denial of Service for example), it is better to have access to the payload of a packet to detect more subtle attacks. Additionally, it is better to label suspicious data to be processed in a safe and isolated environment instead of sending it to end users.

One advantage of using CP-ABE is that it allows using multiple attributes within an encryption policy. There-

fore, it is possible to remove or add an attribute that is specific to the IDS using the ciphertext-policy re-encryption algorithm.

3.3.2.1 Ciphertext policy re-encryption

The ciphertext-policy re-encryption applied to CP-ABE allows the network gateway to remove an attribute or insert a new one into a policy associated with a ciphertext without decrypting the ciphertext. As an illustrative example, we consider a ciphertext of data that is CP-ABE encrypted under an access policy: 'Temp OR Test', i.e., the ciphertext can be decrypted by users with attributes that satisfy either 'Temp' or 'Test'. Using policy re-encryption, the gateway reinforces the ciphertext by removing the 'Test' attribute, and then inserting the policy 'Policy'. Thus, the new generated ciphertext will be encrypted under the policy: 'Temp AND Policy', i.e., the ciphertext can be decrypted by users with attributes that satisfy both 'Temp' and 'Policy'. By repeating the insertion operation, the proposed solution allows to build a ciphertext under a much more strict and larger access policy.

- **Restriction operation:** For a CP-ABE ciphertext built under an access structure, the attributes that can be removed are defined such that there exists a subset composed of the remaining attributes and satisfying the access policy, because, otherwise the ciphertext cannot be decrypted.
- **Insertion operation:** We extend CP-ABE encryption with a new algorithm, the policy reinforcement algorithm. This algorithm aims to reinforce the ciphertext with a new policy. The policy reinforcement algorithm takes the input comprising the public key, the ciphertext that encloses an access policy, and a new access policy, and produces a new ciphertext under the new access policy as the output. The algorithm is described in detail in [**?**] with two instantiations of the proposed algorithm extending Bethencourt et al's scheme and Waters' scheme.

3.3.2.2 Data inspection

Replacing every policy by (policy OR "IS_IDS") at encryptor level will enable IDS to decipher the packets that are most suspicious and increase its accuracy. This way of operating is specific to CP-ABE and does not necessitate to give every possible private key (or the master key used to generate them) to the IDS. One drawback is that CP-ABE does not allow attribute revocation. However, in the unlikely case of an IDS key compromise, it would still be possible to change the configuration of the system and add a new attribute for the IDS.

With the ciphertext-policy re-encryption algorithm described in subsection 3.3.2.1, the added "IS-IDS" attribute can be also removed at the network gateway using the restriction operation i.e., outside the IoT network, the data is CP-ABE encrypted only under policy. After decrypting the message, the IDS would run two detection engines. The first one is a signature-based engine that detects malicious payload that try to exploit a vulnerability (such as a buffer overflow). In this case one would know for sure that an attack is ongoing and the system's response will take this information into account. The second detection engine is anomaly-based, and will only reflect how unusual the message is. In this instance, a softer response will be required in order to avoid disturbing the network.

3.3.2.3 Data tainting

The ciphertext-policy re-encryption algorithm proposed in subsection 3.3.2.1 allows new possibilities of response to ongoing attack. Anomaly-based detection is not perfect, therefore false positives are not to exclude. Hence, the response must be adapted to the degree of certainty of the detection: a packet part of a proven attack must be discarded, but discarding a suspicious and only probably harmful packet might alter the normal behavior of the network. A less extreme approach must be undertaken, and tainting suspicious packets allow for some flexibility while defending the system from attacks. The chosen response consists of tainting the packets

that are considered suspicious at the gateway level. Two kinds of suspicious packets are considered: the ones detected through anomaly-based detection and the ones detected through signature-based detection.

In order to taint irreversibly these packets, the data must be tainted directly. This can be done by reinforcing encryption of suspicious data by using a new attribute. This would limit the set of devices able to decrypt the tainted packet. Indeed, the least constrained devices of the network tainted data. In practice, one would change the policy of each suspicious packets to (policy AND "HANDLES_SUSPICIOUS"). Some of the recipients of this packet may no longer be able to decipher it, but the ones that are less vulnerable to attacks will still be able to handle them appropriately. This tainting method is called *soft tainting*.

Moreover, in the case of proven attacks detected by the signature-based detection engine, a similar technique can be used. Instead of dropping the malicious packets, re-encrypting the malicious messages with an attribute "HANDLES_DANGEROUS" could put the packet into quarantine. This would allow to inspect these malicious packets further in order to assess the objectives of the attack and analyze the attacker's behavior. Only the system administrator would get access to the private key related to this attribute, and no device in the network would be able to read the message. This tainting method is called *hard tainting*. This tainting method could also be used in order to respond to policies not enforced correctly. This could enable to prevent information from leaking to unauthorized users in the case of a policy less strict than the one enforced in the network.

IoT networks being wireless, this response cannot be enforced instantaneously. Indeed, by the time the IDS inspects a malicious packet and notifies the administrator, the packet would already have been processed and transmitted by the gateway. Therefore, the response would stop ongoing attacks rather than the first packet of the attacks. This also means that it is only possible to taint packets from a misbehaving user and not a specific malicious packet.

3.3.2.4 Detection and response algorithm

The detection and response algorithm has four different states:

- State A: the IDS analyses the policies used for encryption. At the same time signature and anomaly-based detection engines are running on the meta-data of the packets.
  - If the wrong policy is used or the anomaly-based detection engine triggers an alert, go to state B
  - If the signature detection engine triggers an alert, go to state C
- State B: the IDS keeps analyzing the policies used for encryption. The IDS decrypts the packets sent by the misbehaving device. It analyzes the content of the packets with both a signature and a anomaly-based detection engine.
  - If not possible to decrypt or an anomaly-based alert is triggered, go to state C
  - If signature-based alert, go to state D
  - After manual inspection of the device, go back to state A
- State C: The IDS keeps analyzing policies, decrypting the packets and asks the network gateway to taint the content of packets from the misbehaving device with "HANDLES_SUSPICIOUS" (corresponds to soft tainting)
  - If signature-based alert, go to state D
  - After manual inspection of the device, go back to state A
- State D: The IDS keeps decrypting the packets and asks the network gateway to taint the content of packets from the misbehaving device with "HANDLES_DANGEROUS" (corresponds to hard tainting)
  - After manual inspection of the device, go back to state A

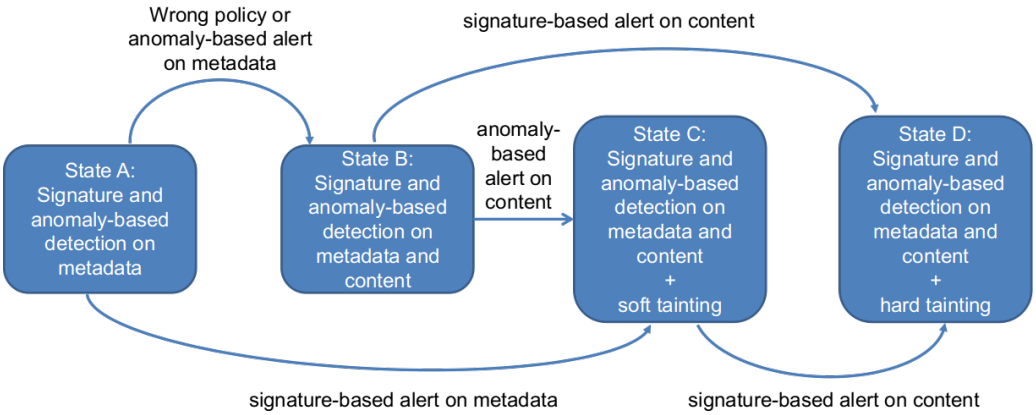

Figure 3.6: Detection and response algorithm

4 Conclusions

This deliverable is focused on describing the deployment of the C-ITS and Smart Objects frameworks proposed in D3.2 over real IoT scenarios, by considering different use cases previously described in D1.3. The objective at the end, it is to demonstrate the advantages of such frameworks in this type of scenarios.

In the case of C-ITS framework, we have evaluated the Packet Delivery Ratio evolution in relation to the car density on the roads. A series of simulations have been tested in three different scenarios: a city, a highway, and a national road. Their aim is to study the impact of vehicle speed and buildings in V2V communications.

Furthermore, we have also provided a comprehensive view of the Smart objects framework and the required interactions to protect large amounts of data in an efficient and flexible way in a real IoT scenario. Towards this end, we have integrated such framework on the Smart Building use case by the instantiation of the framework components on different devices and FI-WARE enablers, which were already pointed in D3.1. Then, we have described the main messages and processes performed by such devices and enablers to fulfill the corresponding functionality.

Bibliography

[1] "Omnet++ discrete event simulator website," https://omnetpp.org/, accessed: 2018-12-28.

[2] "Veins open source vehicular network simulation framework website," http://veins.car2x.org/, accessed: 2018-12-28.

[3] J. Kenney, "Dedicated short-range communications (dsrc) standards in the united states," *Proceedings of the IEEE*, vol. 99, pp. 1162 – 1182, 08 2011.

[4] P. Hunt, K. Grizzle, M. Ansari, E. Wahlstroem, and C. Mortimore, "System for cross-domain identity management: Protocol," Internet Requests for Comments, RFC Editor, RFC 7644, September 2015.

[5] O. M. Alliance, "Ngsi context management," *Open Mobile Alliance*, 2012.

[6] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Security and Privacy, 2007. SP'07. IEEE Symposium on*. IEEE, 2007, pp. 321–334.

[7] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2005, pp. 457–473.

[8] D. McGrew, K. Igoe, and M. Salter, "Fundamental elliptic curve cryptography algorithms," Tech. Rep., 2011.

[9] M. Jones, "Json web algorithms (jwa)," Tech. Rep., 2015.

[10] ——, "Json web key (jwk)," Tech. Rep., 2015.

[11] E. Barker, D. Johnson, and M. Smid, *Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography*. National Institute of Standards and Technology, 2006.

[12] G. Klyne and C. Newman, "Date and time on the internet: Timestamps," Tech. Rep., 2002.