LIGHTWEIGHT ACCESS CONTROL FRAMEWORK FOR IOT DEVICES

by

PITSILADI ADAMANTIA

B.A. Technological Institute of Crete, 2013

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

DEPARTMENT OF APPLIED INFORMATICS
AND MULTIMEDIA

SCHOOL OF APPLIED TECHNOLOGY

TECHNOLOGICAL EDUCATIONAL INSTITUTE OF CRETE

2017

Approved by:
Dr. Papadakis Nikolaos

Abstract

Interconnected computing systems, in various forms, are expected to permeate our lives, realizing the vision of the Internet of Things (IoT) and allowing us to enjoy novel, enhanced services that promise to improve our everyday lives. Key technologies in this new landscape are the various protocols which aim to allow the communication of applications on a global scale. The Extensible Messaging and Presence Protocol (XMPP) is a communication protocol for message-oriented middleware based on XML, enabling the near-real-time exchange of structured yet extensible data between any two or more network entities.

This thesis will focus on implementing the XMPP-based entities needed to create an access control framework for IoT devices and their services, via security policies residing on resource-rich infrastructure nodes. The solution adopted for the policy-based management will be based on eXtensible Access Control Markup Language (XACML). The developed entities will be deployed and evaluated on a variety of resource-constrained embedded platforms, based on a preliminary evaluation that will have to take place, considering the various XMPP libraries and the supported platforms.

Table of contents

List of Figures

List of tables

Acknowledgments

For this master thesis, I would like to thank my professor Dr. Papadakis Nikolaos for the opportunity who gave to me to deal with this field of Informatics and providing advice and continuous support. Also, I would like to thank my supervisor Dr. Fysarakis Konstantinos for the patience and guidance he showed to me every time I was "lost". Special thanks to Othonas Soultatos and George Alexiou for their continuously help. Last but not least, I would like to extend my deepest thanks to my family and friends.

Pitsiladi Adamantia

Dedication

To my brother, Andreas.

1 Introduction

1.1 Background

The idea behind the Internet of Things is to connect all the electronic devices to each other and the Internet. By saying electronic devices, it means almost all the electronic devices. From mobile phones and computers to coffee makers, cars, streetlights, building lifts, lamps, many of wearable gadgets and everything else that someone can imagine. The Internet as it is currently known is the backbone of the Internet of Things, but it is not necessary for the devices to have direct access to it. For example, a fitness band collects a plethora of data about the health and fitness of the user, transmits them to user's smartphone or tablet via Bluetooth, and then passes online to the cloud which use to record them. That means that we are talking about data collection from any electronic device or tiny sensor around us. Most of the humans have at least one device permanently connected to the Internet (smartphone or computer), and a broadband connection has become much more affordable than in the past. At the same time, technology has led to lower production costs, with more and more devices having built-in Wi-Fi or Bluetooth, as well as advanced sensors. Of course, it should be noted that with the advent of the IPv6 protocol, billions of connected devices will be allowed to have their own IP address to be able to communicate. According to Cisco **[**1**]**, by 2020 there will be 50 billion connected devices. From the above, there is a positive impact on our everyday life. One example is in the field of health where smart blood glucose measurement devices can send measurements directly to the patient's doctor to have a better picture of their health status with detailed daily measurements. There are many examples of the use of IoT devices, and all of them have the main characteristic. That characteristic is that all the IoT devices keep a huge number of personal data in their databases. This raises security issue.

1.2 Motivation

Because all the devices are connected to the internet, there must be security mechanisms that will not allow malicious and hackers to have access to them, and simultaneously protect the exchanged data and the personal data of each user. That is almost a solved problem for modern PCs and mobile devices. Unfortunately, this is not happening to the devices that comprise the IoT. Mostly, those devices have a very constrained set of resources such as small amounts of RAM and power, and capabilities (low processing power) that could not withstand security implementations used on other more powerful devices such as smartphones and tablets. Also, the large scale of integration of devices into the IoT creates the challenge to provide security. The rapid increase in IoT devices is enough to increase the need for security services, as many of

them are prone to attacks. In researches that have taken place, researchers have found gaps in spots of safety that have spread to garage doors **[**2**]** from cars **[**3**]** and skateboards **[**4**]**. In any case, it is necessary for every IoT system to have a security mechanism. The purpose of security is to separate the "good" from "bad" guys. Access control is one of the most important elements of security. It is security technique that can be used to regulate who or what can view or use resources in a computing environment. So, having the ability to control who has access to the IoT environment is very important.

1.3 Aim

Because of the need to protect the services of each IoT device and because of the constrained set of resources that the IoT devices have, in this work will focus on the implementation of a lightweight framework using security policies. The security mechanism will be based on the access control and to be more specific on the eXtensible Access Control Markup Language (XACML). The communication of the devices and the entities of the XACML will use the XMPP protocol, which is a protocol for instant messaging and one of the main protocols for IoT environments.

2 Related work

The concept behind the Internet of Things is that all the devices must communicate with each other. That communication consists the exchange of users or industry pieces of information. That means that there is a point that must pay attention: who can have access to the resources of the IoT devices. The following subsections present the existing work on the access control on the IoT.

2.1 IoT Access Control

IoT applications are mostly related to someone's personal life or industry. Security and privacy issues need to be addressed in all such environments. The middleware should have built in mechanisms to deal with such matters, along with user authentication, and the implementation of access control. In the IoT are including the following three types of Access Control. The Role-based access control (RBAC), the Credential-based access control (CBAC) and the Trust-based access control (TBAC). In the case of CBAC, it divides into two types. The first one is the Attribute-Based access control (ABAC), where on this type when a user has some unique attributes, it is possible to access a particular resource or piece of data. The second type is Capability-Based access control (Cap-BAC). On this type, the capability is a communicable, rights markup, which falls in with a value that individually specifies specific access rights to objects owned by subjects. To improve the security of the system, some of the access control methods include encryption and key management mechanisms and to improve this. The above three methods are combined each other.

Assume a simple example, that a doctor in a case of an emergency (i.e. traffic accident) is urgently needed to have access to the patient's/user's location. In **[**5**]** writers propose an identity-based system that handles personal information, like a private place, in emergencies. This system includes one subscription, user authentication, policy, and a subsystem for customer terminal status control. This system will authenticate the user through the policy subsystem. That will ensure that individual users who are authorized will be able to access a user's location. In **[**6**]**, writers, write down the existing authentication and access control and design an appropriate control access protocol for the IoT. The authors, for the access control policy, use an RBAC authorization. This method based on requester's specific roles and application which related to the network of IoT. However, because there is a large number of users that participate in the IoT network, RBAC cannot operate properly and cannot assign permissions in advance, and the method places a high toll on perception nodes in the entire communication process. The authors notice that the reliability evaluation in the actual situation does not apply, with the result that writers make changes to the protocol to the detriment of safety and function-based aspects **[**6**]**

and analyze the performance of an improved protocol. In **[**7**],** authors make changes to the protocol on **[**6**]**, in the piece of security mechanism and analyze the performance of the improved protocol. The new protocol includes a registration phase, login and authentication phase. The new protocol includes new capabilities for password recovery and modification capabilities to help users manage passwords. When it is time for the registration phase, each user has to register with the main registry. As a result, the basic parameters must be negotiated and computed between the user and the gateway node in the connection and verification phases. This completes a mutual authentication between nodes in two phases of login and authentication.

RBAC and ABAC, are the most traditional access control models. An important issue of the RBAC and the ABAC is that they cannot be directly applied to IoT, that's because of limitations of flexibility **[**8**]**. Existing research papers mainly combine ABAC with other methods to obtain suitable access control methods for IoT. The IoT environment is highly dynamic, and it has a huge number of users. That makes the RBAC model not capable because it cannot allow permissions in advance with usual access control methods. Also, the authorization process with ABAC is a little complex. The problem is that cannot apply to the dynamic and real-time environment of IoT. The number of rules immediately increases by users, attributes and growth. Despite all the difficulties mentioned, there are still some advantages that can be utilized in RBAC and ABAC. ABAC can solve the problem of dynamic propagation problems of users, while RBAC can effectively solve the problem of competencies with time and location changes. By combining the above in **[**9**]**, the authors propose a hybrid access control model based on role and attribute (ARBHAC). This model is capable of solving the large-scale dynamic problem of users in IoT. This model pre-assigns roles for nodes/users based on the property expression of nodes/users, and then assigns the roles distribution and corresponding permissions to the appropriate nodes/users. The model performs a property rule policy language and a solution to the conflict with the verbosity policy. Also, it makes easy the complexity of traditional ABAC in rights allocation and policy management. Nevertheless, it is not enough to deal with the policy conflict and verbosity processing. That's because the proposed model still needs the administrator to manage roles and permissions licenses and needs further research to improve its usability.

Another access control mechanism is Ciphertext-policy attribute-based encryption (CP-ABE). In this access control, user’s private key is associated with a set of attributes and a ciphertext specifies an access policy over a defined universe of attributes within the system. The main problem with this access control mechanism is the lack of a valid key. While researchers have found solutions, there is a problem in transmitting and consuming complexity. The other solutions require a strong confidence to decrypt the data. In **[**10**]**, the authors propose an activity-based access control mechanism. This mechanism is a generalized version of the environment. The purpose of this access control is to make decisions based on users' perceptions and the state

of the system. This is accomplished as follows. Using a finite state machine and a CP-ABE, which is an asymmetric encryption mechanism, a real-time access control policy can be adapted to changes in user and system status. This machine can also be used to simulate the status of the users. Each object has a unique connection to a finite state machine, which is manufactured from the design phase of the system. The finite state machine also includes all the probable states of life cycle of the system. It specifies the object state changes based on predefined time and manner. Each state matches to a set of access rights. The program also takes into account the real-time features of the revocation mechanism. Another type of mechanism for access control is called attribute-based CP-ABE. Authors have proposed **[**11**]** a revocation mechanism. This is based on a batch method and can reduce processing complexity and overall consumption and does not need to add additional nodes to the system. In **[**12**]**, makes a reference in a new relative access control method. The method is centered on Access Proximity Access Control (PBAC). This is a highly advanced access control approach that can implement flexible, proximate, dynamic, contextual access. PBAC, ABAC, and MDS (Security Driven by Model) are all about expressing and implementing security and privacy requirements. PBAC, ABAC, and MDS (Security-Driven by Model) are used for privacy. The PBAC uses proximity policy with applicants and resources. An ABAC extension is necessary to support fine-grained, flexible, environmentally sensitive proximity-based access control policies. To make ABAC more usable and PBAC policy safer, it is recommended to use an MDS mechanism to create MDS and MDS certification automation strategies. The paper describes the approach management policy using MDS and the extensive ABAC system with a detailed example of intelligent system transfers. So ABAC will be more flexible, and the PBAC policy will be safer. The paper also describes the policy management approach by using MDS and extended ABAC with a detailed example involving intelligent transport systems.

Authors in **[**13**]** suggest an access control model (Cap-BAC) based on existing abilities. This model is based on the principle of the minimum privilege to manage access to information control services and procedures. For information control procedures to be performed, the user must present his /her authorization certificate to the service provider. The certificate is issued by the owner of a resource to the desired users. This ensures that these users can request services/resources. It indicates the availability of a security mechanism and correlation among access authorization in addition to the need for processes that are understandable and accessible. The paper recognizes the default principle of least benefit, recommends revocation of benefit and control of validity for a guarantee of competency, providing adequate security and flexibility for the practical application of the program. The trouble with this solution is that it requires the possibility of publishing all the main certificates and the choice when a certification body submits a request. The solution to the problem is to provide authorized policy ability. This can

be used to set a specific service and then generate appropriate access rights for recognized and authorized users. Another work at this mechanism is shown in **[**14**]**. This work provides a basic architecture of security. In this architecture, the secure support and verifiable interactions can easily be deployed on a mobile platform. Also, in this work referred to mobile e-health applications in medication control. The SSL protocol is proposed for the M-Health Security Protocol (MHSP), which is an application layer protocol for authentication between the network and the application. The usage of that protocol is to provide a secure channel using the public key certificate for authentication review. Thru the use of personal electronic health records, this architecture can set up and manage prescription services in a mobile environment. Depend on RFID technology and a simple and intuitive interface, the architecture can be used for mobile e-health applications (M-Health).

The environment of IoT for security purpose needs an access control mechanism which is flexible, lightweight and adaptive to deal with the reliable communication between devices. Authors have proposed **[**15**]** an access control system for IoT with the above characteristics (TAC-IoT). This mechanism is a lightweight authorization, designed specifically for networking model based on trust. It provides a reliable security mechanism from beginning to end in IoT. It has been successfully implemented in a practical experimental stage, by evaluating by using constrained and unconstrained networking equipment To stand out with the universal nature and to ensure credible and reliable communication between the IoT devices it is necessary a flexible, lightweight and adaptive access control mechanism. The aim is to provide a reliable security mechanism from beginning to end. In **[**16**],** it is reported that the known access control model does not thrive in distributed dynamic environments. The reason of that is the identity of the two devices which cannot be known before. The trust relationship between the devices directly affects their behavior. That is, when the devices trust each other they are more willing to share information with each other.

3 Technical Background

3.1 Access Control

One of the most important requirements of any information management system is the protection of data against non-permitted levels disclosure and unauthorized changes while ensuring their availability for legitimate users. Enforcement of protection, therefore, requires' that every access to a system controlled and that only authorized access can be performed. This process defines access control **[**17**]**,

3.1.1 XACML

XACML (eXtensible Access Control Markup Language) is an XML-based language for setting policies and for controlling access to some resources. Since February 2003, it has been a standard for OASIS (Organization for the Advancement of Structured Information Standards). One of the key features of XACML is the fact that properties are used to express policies. That is ABAC (Attribute-based Access Control), access-based access control model. ABAC uses structural module attributes within a structured stream that defines access control rules and describes access requests. Attributes are sets of labels or attributes that can be used to describe all entities to be examined for approval purposes. Policies created with the use of XACML are in the form of XML. The request for access to resources, as well as the corresponding answer, are also determined by using XML. XACML provides a language for policies that allow users to define the access control requirements for the resources of their applications. The language includes data type, functions and combinational logic which allow simple and complex rules to be defined. It also includes a simple decision-making approach. When one has created a policy that protects a resource, and there is a request for access to it, then the functions compare the characteristics of the application with the features that are contained in the policy rules. So if the application is in line with the rules, there is a positive, if not a negative, decision to access **[**18**]**. Assume a simple scenario. A user wants to have access to a resource of a device which is in this example is a thermostat. The request that the PEP forwards to the PDP entity is the following. The subject is " user@pits/Smack". The "user" inform us who wants to have access, the "pits" inform about the server and the "Smack" the resource(here is Smack because is the code is written in Java, in any other case it can be e.g. "mobile"). The resource is the name of the device, which is "Thermostat."

```
<Request>
<Subject SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="http://www.w3.org/2001/XMLSchema#string"><AttributeValue>user@pits/Smac
```

```
k</AttributeValue>
</Attribute>
</Subject>
<Resource>
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#string"><AttributeValue>Thermostat</At
tributeValue></Attribute>
</Resource>
<Action>
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string"><AttributeValue>GetTemperature
</AttributeValue></Attribute>
</Action>
</Request>
```

Table 1 An example of request in XACML

The rule is "permit" in case that the request includes the subject " user@pits/Smack" and is from the "Thermostat".

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy
   xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os
      access_control-xacml-2.0-policy-schema-os.xsd"
   PolicyId="test850"
   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-
overrides">
  <Target>
   <Actions>
      <Action>
       <ActionMatch
          MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">GetTemperature</AttributeValue>
         <ActionAttributeDesignator
            AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
       </ActionMatch>
      </Action>
    </Actions>
        <Subjects>
      <Subject>
       <SubjectMatch
          MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">user@pits/Smack</AttributeValue
>
         <SubjectAttributeDesignator
```

```
            AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch
          MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
         <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">Thermostat</AttributeValue>
<ResourceAttributeDesignator
            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ResourceMatch>
      </Resource>
    </Resources>
  </Target>
 <Rule
    RuleId="urn:oasis:names:tc:xacml:1.0:conformance-test:IIA1:rule"
    Effect="Permit">
<Target>
          <Subjects>
      <Subject>
        <SubjectMatch
          MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
         <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">user@pits/Smack</AttributeValue
>
         <SubjectAttributeDesignator
            AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch
          MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
         <AttributeValue

DataType="http://www.w3.org/2001/XMLSchema#string">Thermostat</AttributeValue>
         <ResourceAttributeDesignator
            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ResourceMatch>
      </Resource>
```

```
    </Resources>
  </Target>
 </Rule>
 <Rule
    RuleId="urn:oasis:names:tc:xacml:1.0:conformance-test:IIA1:rule"
    Effect="Permit">
 </Rule>
 </Policy>
```

Table 2 An example of Policy in XACML

The following lines shows the response that the PDP entity forwards to the PEP.

```
<Response>
<Result ResourceID="Thermostat">
<Decision>Permit</Decision>
<Status>
<StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
</Status>
</Result>
</Response></Result>
</Response>
```

Table 3 An example of Response in XACML

3.1.1.1 Architecture

An XACML architecture commonly consists of the following components:

- Policy Enforcement Point (PEP): It performs access control, by making decision requests and enforcing authorization decisions **[**18**].**
- Policy Administration Point (PAP): Creates and manages policies or policy sets [18]
- Policy Decision Point (PDP): It evaluates requests against applicable policies and renders an authorization decision **[**18**]** .
- Policy Information Point (PIP): It acts as a source of attribute values. **[**18**]**
- Context Handler: It orchestrates the communications among the stakeholders, converts, if necessary, messages between their native forms and the XACML canonical form, and collects all necessary information for the PDP.
- Environment: Provides additional information independent of a particular subject, resource or action. Refers to features related to the environment and can affect a PDPs decision, e.g. accident scene or hospital emergencies.

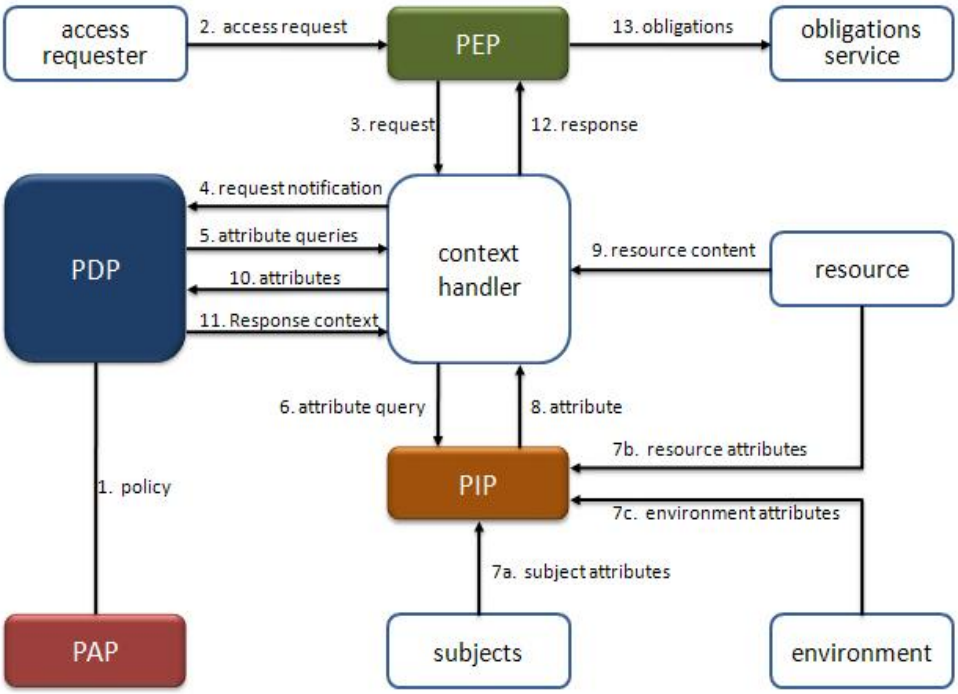

Figure 1 XACML Architecture

The model works in the following order:

1. PAP has Policies and PolicySets stored, which are available in PDP. These Policies and PolicySets represent the entire policy for a particular target
2. The PEP receives the access requests and promotes them in native format.
3. The context handler constructs a XACML request and finishes the PDP.
4. The PDP requests any information regarding the properties of subjects, resources action
5. The context handler then asks the attributes from PIP.
6. PIP acquires the requested attributes.
7. PIP returns the requested properties to the context handler.
8. The context handler also includes the resource
9. The context handler sends the requested properties to the PDP.
10. The PDP appreciates politics.
11. The PDP returns the answer to the context handler
12. The context handler converts the response to the physical language of the PEP and at the same time returns the answer. If the result is to allow access, then the PEP accepts access to that resource - "Permit", otherwise it denies access - "Deny".

3.1.1.2 XACML Structure

The XACML language is as expressive as the natural language. For example, the expression "a user wants to apply an operation to an information resource under a given

condition" includes four grammar units, a subject, an action, a resource, and the environment in which the request is made.

- The subject asks for access to an information resource.
- The act that the subject wants to accomplish.
- The resource, recognized as the information or object to which the act is affected.
- The environment, which is identified as the context in which the access request is made. The most common interface properties are related to the current time and location where the access request is made.

3.2 IoT Protocols

There are so many IoT protocols, that is like an IoT protocol's war. Various of them aim to address some issues of scaling and inherent limitations regarding resources, like CPU, memory, power, etc. In the above lines, some of them are going to be described and combined. Because of this work is focused on XMPP protocol, in the below lines this work will be emphasized at that protocol.

3.2.1 XMPP

Extensible Message and Presence Protocol (XMPP - formerly called Jabber) [19] is a set of open protocols for instant communication, based on XML and created by IETF. It is comprised of a set of standardizations for the XML flow protocols (Streaming Protocols) for instant messaging. More specifically, as determined in RFC 3920 [20] the transmission of data to XMPP is based on an XML flow protocol that allows for exchange XML messages between two points of the network.

3.2.1.1 Architecture

Thus, the Internet infrastructure in instant messaging and presence is the existence of hundreds of thousands of servers running software such as Openfire **[**21**]** and ejabberd. Each user has an account and a unique ID on the network called a JID. This identity is the user's address on the XMPP network. Based on this, it is possible to identify the user from the server. By installing a server software, the domain name to which the users who have accounts and therefore access to the server is defined. This name is part of the JID address. The standard form of a JID user@domain and is called bare JID. Additionally, it can be used the address in the user@domain resource (full JID) format. The RFC-6122 defines for XMPP the format of the URI as xmpp: user@ domain/resource for each XMPP entity. The resource is defined as the user's specific resource, that is, the specific way which the user is connected to the server.

It is an identifier, that is used to define the route of messages as a user can connect with the same account from different locations using different devices or different client/server. Finally, the existence of the resource implies the possibility of updating the presence of the user for each different connection. The identifier can simply be in the form of a string. XML stream refers to the basic technical structure of XMPP.

When a user initiates a session with a server, a TCP connection opens and then negotiates the creation of an XML broadcast channel with it. A corresponding channel in the opposite direction opens the server. Once the channel has been negotiated, it can start the exchange between XML messages (XML stanza, RFC 3920 **[**22**]):** <message/>, <presence/> and <iq/> , which are going to be described in a following section.

Here is a simplified example **[**23**]** of the XML stream that illustrates the dialogue between a user and another user as well as the on-board communication with the server.

```
C: <?xml version='1.0'?>
    <stream:stream
      to='example.com'
      xmlns='jabber:client'
      xmlns:stream='http://etherx.jabber.org/streams'
     version='1.0'>
S: <?xml version='1.0'?>
<stream:stream
   from='example.com'
   id='someid'
   xmlns='jabber:client'
   xmlns:stream='http://etherx.jabber.org/streams'
   version='1.0'>
... encryption, authentication, and resource binding ...
C: <message from='usre@pits'
      to='thermostat@pits'
   xml:lang='en'>
C: <body>GetTemperature </body>
C: </message>
S: <message from='usre@pits"'
     to='thermostat@pits'
   xml:lang='en'>
S: <body>deny</body>
S: </message>
C: </stream:stream>
S: </stream:stream>
```

Table 4 Example of XML Stream

It is important to emphasize that it is also stated in RFC-6120 **[**24**]** that the basic data structure of the protocol is the XML message, which is the means of transporting information from point to point to the network. Steps describing a user-to-server connection process and termination according to RFC-6120 are the following:

• Identify the IP address and the port to be logged on by the user. The s2c communication port is 5222.
• Start a TCP connection.
• Create an XMPP transmission channel through the TCP connection
• Preferential TLS security negotiation for channel encryption.
• Certification using SASL
• Assign a resource to that broadcast channel.
• Exchange an unlimited number of messages
• Exchange unrestricted XML stanzas with other XMPP entities on the network.
• Closing the XML broadcast channel.
• Terminate the TCP connection.

The XMPP architecture still allows inter-domain/inter-server interconnection in an analogous way to the one described above and the gateway to which it communicates 5269. It is, therefore, possible for users who have an account on a particular server be available for communication to users of a differential server configured to communicate with the original and to route the messages accordingly.

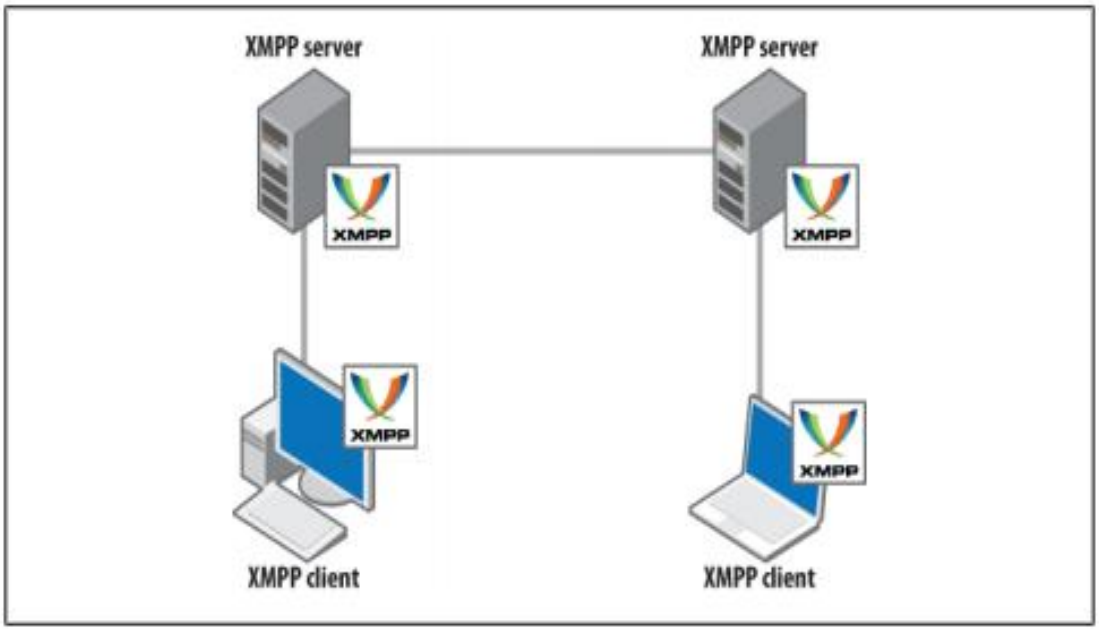

Figure 2 Diagram of architecture server-client

Source: XMPP: The Definitive Guide

As mentioned, a basic property that defines the XMPP protocol is the capability to be extended. By this, it means, that it is possible to use and create extensions, ie templates describing the XML format of files that can be included in the three message formats (message, presence, iq) in order to provide additional communication functionality. Since XMPP is purely XML technology, extensions are implemented using XML namespaces. This specifies the type and

structure of the payload that may contain any stanza to obey a particular extension pattern and to be understandable as a message to the recipient. Expansion matching is done both in the name of the element and in the namespace.

In this case, the extension is the <html/> element with the namespace xmlns = "http://jabber.org/protocol/xhtml-im". This is the XHTML-IM extension XEP-0071 which is used to set the appearance of a <message /> stanza body. So far, dozens of extensions have been developed by the XMPP development community, and the XMPP Standards Foundation publishes a large majority of such extensions. However, users can quickly set up their extensions for their personal use to add extra usability to their applications.

3.2.1.2 XML stanzas

XMPP uses a communication unit called Stanza. Stanza has the role of the package or message that it encounters in other communication protocols and their importance depends on the following factors:

1. The element may vary between the values message, presence, and iq. Each stanza of a different type is routed differently from the server and used other than users.
2. The type element that depends on the kind of stanza and determines how the recipient will handle it.
3. The payload, which is either managed by the user or is used by some automated process depending on its character.

3.2.1.2.1 Message

Message type stanza is the basic way of transmitting information (Push) and separated according to the value of the type element to:

1. Normal, which is very similar to email.
2. Chat for a session between two nodes of the XMPP network
3. Group chat which are messages that are exchanged in context a network where each message is addressed to those who are involved in the team.
4. Headline used to send notifications
5. Error message related to a message sent to previous communication between two nodes for which message whichever node encounters the problem will send a message to let him know of his type. In addition to the type element, the stanza also contains the elements to and from, which describes the addresses of the sender and the recipient in the form of an identifier (JabberID).

```
<message from="user@pits/Smack"
   to="thermostat@pits/Smack"
   type="chat">
<body>GetTemperature</body>
```

```
</message>
```

Table 5 Example of message <message/> type "chat"

3.2.1.2.2 Presence

One of the features that distinguish the communication systems real time is the ability to update its nodes network for the availability of a node if it is connected and available for communication. This feature is described with the field presence of stanza and is an important catalyst for its use Internet in communication and collaboration between users and there is a greater chance of interacting if they know the availability of each user participating in their contact list. The presence stanza is a specific implementation of it Publish / Subscribe model where someone subscribes receives updates regarding the availability of another user with which user wants to interact.

```
<presence from="thermostat@pits/Smack"
   to="user@pits/Smack"
   <status>Available</status>
</presence>
```

Table 6 Example of message <presence/>

3.2.1.2.3 Iq (info/query)

Those messages provide a structure for interactions between users and the server or an XMPP entity in the form of requests/responses. Unlike the <message /> stanzas, iq can only have a particular load that has a standard structure and precisely identifies the action to be performed by the recipient. Reply to the sender. For IQ stanza the type field gets the values:

1. Get where the requesting node asks for information.
2. Set with which a node provides information to a request.
3. Result the recipient of a request sends the results of one request to get or recognizes a set request.
4. Error with which the recipient node or some other intermediary node, e.g. an XMPP server, updates the node that it does the set or get request that there was no possibility of processing. **[**25**]**

```
<iq from="thermostat@pits/Smack"
id=" pk1456z "
to="user@pits/Smack"
type="get">
</iq>
```

Table 7 Example of message <iq/>

3.2.1.3 Connection security

The XMPP supports communication security in the SASL and TLS encryption protocol. When the XML stream begin between the partitions and after sending the streams

<stream:stream...> the side that connects the receiver to the stream, sends the stream features. Stream features are the requirements or preferences for encryption and connection authentication. An indicative such message is as follows:

```
<stream:features>
        <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'>
        <required/>
        </starttls>
</stream:features>
```

Table 8 Indicative part of XML stream - stream feature, for or not requiring cryptography

This message indicates that the connection receiver requires TLS encryption. The link initiator must then send a message that marks the beginning of the TLS negotiation as follows: <Starttls xmlns = 'urn: ietf: params: xml: ns: xmpp-tls' />

3.2.1.4 Previous work

The XMPP protocol has been used in many fields of IoT networks. One of this is in the field of the health care. In hospitals it is a huge need the inter-department communication. The problem with this communication is a simple request for information between departments can suffer considerable latency and be highly variable in terms of quality and delivery. In **[**26**][**27**]**,authors proposes an architecture which could decrease the turnaround in information exchange between departments. They want that architecture to offer controlled data sharing within a hospital environment and ultimately improve the quality of care for the patient. Also, in the field of health the work **[**28**]** advocates for applying IoT architecture in providing smarter, connected and personalized healthcare solutions in smart homes. Another field is the smart home. Authors have proposed **[**29**]** that instant messaging is suitable for smart home systems by showing that the requirements can be met by its features. In **[**30**]**, the authors introduced a service infrastructure for the Internet of Things which seamlessly integrates real world objects, backend-systems and mobile devices. The platform relies on XMPP as communication protocol to support efficient and highly scalable communication between all building blocks. In **[**31**]**, authors deal with a seamless integration, discovery, and employment of smart objects into the Internet infrastructure under Human-to-Machine (H2M) communication aspects.

3.2.2 COAP

Constrained Application Protocol (CoAP) is a client-server information transfer over the Internet, similar to HTTP, but is intended for devices with limited resources. Most devices used in a Wireless Sensor Network (WSN-Wireless Sensor Network) have limited RAM capacity and limited data processing capabilities. These devices are typically connected to low bandwidth networks such as 6LoWPAN, which have a high rate of packet loss. Using CoAP, we try to apply the basic HTTP data for application data transfer using restricted networks.

CoAP is a protocol that has elements similar to HTTP, but allows us to achieve low overhead and multicast. However, because HTTP is based on the TCP protocol, which uses point to point communication, making it complicated for use in Internet of Things applications. For this reason, CoAP uses the UDP protocol that is lightweight with respect to TCP and allows multicast that meets the need for group communication. In addition, CoAP is based on the REST architecture that enables it to use GET, PUT, POST, and DELETE methods, the use of which helps to overcome the unreliability of the UDP protocol. **[**32**]**.

The use of CoAP is one of the most widespread solution in IoT applications and the protection of communication between devices and protection from integrity, confidentiality and authentication is taken seriously. CoAP uses the DTLS protocol to protect transmitted data by implementing TCP techniques. In contrast to other Network Layer protection protocols, DTLS works at Applcation Layer to provide end-to-end communication between nodes **[**33**]**. To succeed in protecting data exchanged over the network, it uses 2-way authentication through the use of handshake. There are three kinds of handshake between client and server before you begin to transfer any application information. Initially a handshake takes place on either side. It is not necessary to confirm it identity on the other side. Then the server's confirmation handshake proves identity to the user. Finally, the user must verify their identity on the server to complete the authentication process between server and client **[**34**]**. There is significant research interest in CoAP, with numerous efforts to leverage its extremely lightweight interactions in domains such as smart homes **[**35**]**, mobile IoT deployments **[**36**]**, cloud services **[**37**]**, healthcare **[**38**]**, smart cities **[**39**]** and industrial WSNs **[**40**]**.

3.2.3 DPWS

Service-Oriented Architectures (SOA) are often used to improve the flexibility and reuse of components in complex distributed applications. This is accomplished by modeling functional departments as independent services. The Devices Profile for Web Services (DPWS) can be used to implement a Service-Oriented Architecture (SOA) architecture that fits into the application-centric device, thus enabling the application of SOA. Architecture in the field of network devices. DPWS (Device Profile for Web Services) was developed to allow the capabilities for Secure Web Service (WS) on Resource-Constraint Devices. DPWS is a basic technology for device communication that can be easily compiled and expanded with other specifications and technologies. The DPWS has an architecture that resembles Web Service Architecture (WSA) to better match device scenarios. The main difference is Multicast Service Discovery with WS-Discovery that does not requires a central service register, such as UDDI (Universal Description, Discovery, Integration). However, the use of the service in device services is similar to using the service in the WSA architecture, according to which DPWS

devices can be directly integrated into WSA-based systems [41]. The use and benefits of DPWS have been studied extensively in the context of various applications areas, which, other than the ones already mentioned, include automotive and railway systems [42], industrial automation [43], eHealth [44], smart cities [45] and smart homes [46].

3.2.4 MQTT

Message Queuing Telemetry Transport (MQTT) is the dominant protocol in the context of messaging between nodes of an IoT network. It is lightweight, based on a publicly available code and is easy to implement. These features make it ideal for use on networks involving limited capacity devices such as Machine to Machine (M2M) and Internet of Things (IoT). The MQTT implements the Publish / Subscribe model for messaging where a network node, called the publisher, can send messages to another node (or more) called a subscriber by using an intermediary server called a broker. The user of the broker as an intermediary node achieves the filtering of the messages and their delivery to all interested nodes. The separation between the publisher and the subscriber can be differentiated into a three-dimensional framework.

- The publisher does not know the existence of the subscriber and vice versa. This dimension is called space decoupling
- It is not necessary for the publisher and the subscriber to be connected at the same time. Time decoupling dimension
- Services running on a node need not be interrupted either during the process of sending or receiving a message. This dimension is called synchronization decoupling

MQTT supports many different types of messages with the most usable from the end-users is "connect","publish", "subscribe" and "unsubscribe" messages. The other types of messages are used by internal communication mechanisms of the protocol and by message streams [47]. MQTT uses topics to simplify the process of sending messages. Issues are used to make it easier for the broker to know which subscriber is interested in the messages. Topics are organized in a hierarchical manner, like the folder structure in a file system; e.g. “home/kitchen/oven/temperature” could be a topic where a device can subscribe to get updates on the oven’s temperature. When a client publishes a message, the Broker then relays this message to all clients subscribed to the message’s topic. Thus, all interactions are asynchronous and clients only communicate directly with the Broker. As with the previous protocols, researchers have already studied MQTT in a variety of domains, including eHealth applications [48] [49], WSNs and smart grid [50], smart homes [51] and also mobile IoT contexts [52] among others.

3.2.5 Comparison

This thesis will focus on XMPP protocol as an application layer protocol, for the sake of completeness, a comparison between the different application layer protocols shall be made at this point. However, the comparison, will be articulated only between the protocols that use TCP or UDP as their transport protocol, because the other ones using HTTP as transport protocol are not suitable for lightweight implementations on constrained devices due to the overhead introduced by HTTP. As a result, the HTTP itself, and the SOAP protocol will not be included in the comparison.

The first protocol that will be overviewed is DPWS. It is very similar to UPnP (Universal Plug and Play), but is mainly used on large scale enterprise networks rather than home networks in which UPnP is preferred. DPWS is also natively built in many versions of Windows. DPWS relies mainly on UDP but also uses TCP for its transport and is a defined set of minimal implementation constrains to enable secure Web Service messaging, discovery, description, and synchronous and asynchronous communication on resource-constrained devices. Because of its Web Services nature, DPWS describes two types of services: hosting services and hosted services. A hosting service is associated with a single device, while a device may accommodate many hosted services. Other services available from DPWS are: Discovery services, Metadata exchange services and Publish/Subscribe event services. DPWS was used in the EU Research Project SOCRADES **[**53**]** that focused on implementing, testing and piloting prototypes of DPWS-enabled devices for use in the industrial automation domain and is currently under research for use in "smart" cities, "smart" homes and other applications.

Constrained Application Protocol (CoAP) is an application layer protocol designed for use in constrained devices in order to allow them to communicate interactively over the Internet. CoAP is designed with simplified integration of WSN nodes into the web, as its target, while also retaining some basic features such as multicast support, very low overhead, and simplicity. A mapping of CoAP to HTTP is also defined enabling access to CoAP resources via HTTP, as simple as loading a website on a browser. It follows the request/response model, where a client interacts with the server using a subset of the HTTP methods that are: GET, PUT, POST and DELETE. CoAP is specified on RFC 7525 **[**54**]**, while other extensions are currently under the progress of the standardization process. CoAP is probably the most popular application layer protocol used in IoT with numerous examples in domains such as smart homes, mobile IoT deployments, cloud services, healthcare, smart cities, and industrial WSNs.

MQTT is a TCP/IP messaging protocol with a default port in 1883. Typically focuses on providing asynchronous data transfers between distributed devices. Its focus is on reliable messaging, including message buffers and Quality of Service (QoS) facilities, controlled by centralized entities. The architecture that follows is the publish / subscribe (pub/sub) model that

comes as an alternative to the traditional client-server model where two devices communicate directly. Here, the customer who sends a message (pub) does not know the existence of the recipient (sub). There is a third piece of the puzzle, the broker, who is known both to the publisher and the subscriber. The job of the broker is to receive all incoming messages and to forward them appropriately so that specific messages reach specific recipients.

	DPWS	**CoAP**	**MQTT**	**XMPP**
Version	1.1	RFC 7252	3.1.1	RFC 7622
Standard Status	Version 1.1, OASIS (2009)	IETF	Version 3.1.1, ISO/IEC 20922 (2016), OASIS (2014)	IETF
Type	Service Oriented protocol	Resource Oriented protocol	Message Oriented protocol	Message Oriented protocol
Transport Layer	TCP &UDP	UDP	TCP	TCP
M2M Communication	Yes	Yes	Yes	Yes
Synchronus Communication	Yes (Service Invocation)	Yes (Request/response, via HTTP methods)	No	real time
Asynchronus Communication	Yes	Yes	Yes	Yes
Communication	Publish/Subscribe to Service - WS-Eventing	Observe Resource - RFC 7641	Publish/Subscribe to Topic	Publish/Subscribe
Discovery	WS-Discovery	Yes (RFC 5785 / RFC 6990)	No	Yes (XEP-0030)
QoS	Not integrated	Elementary support	Yes (3 modes)	Not integrated
Security	Payload encryption, WS-Security, TLS, IPSec, 802.15.4	Payload encryption, DTLS, IPSec, 802.15.4	Payload encryption, DTLS, IPSec, 802.15.4	SASL, TLS, Non-native end-to-end encryption

Table 9 IoT protocols comparison

According to the abovementioned and the aggregated/comparative table, several reasons have led us to choose this protocol. Initially, the MQTT is both lightweight and has a considerably small message size, although messages do not include the sender's information. CoAP is one of the most lightweight protocols, but it does not support publish/subscribe technology that the other protocols have. The DPWS, in turn, provides exactly what XMPP is. The significant difference lies in the fact that the messages it sends to DPWS are significantly larger, which is something we do not want. Since the birth of XMPP was intended to support IM systems for which real-time communication and exchange of information is required, it is a fact that this protocol ensures the immediate and uninterrupted exchange of small-sized messages. Because of its architecture, it is easily expandable, so it is preferred in IoT applications, having the ability to transmit information to multiple nodes of a network in real time. For all the above reasons, the XMPP is chosen.

4 Proposed framework

4.1 Architecture

The proposed framework follows the architecture of the XACML schema. The architecture composed of the entities of PEP, PDP and PIP. The existence of PEP is responsible for the communication between the devices and the PDP. The PEP is also responsible for receiving the request from the device and to translate it into to the format that the PDP can understand. The PDP can read only requests which are written in the XML language. After, it gets the request, to make the decision, forwards the request and asks PIP for the policies - if any. The PIP keeps all the policies. The policies follow the same XML format language. The process for searching the available policies begins. After it finds the policy with the same attributes, sends the policy to the PDP. The PDP is the Policy Decision Point. As for its name, the PDP starts to evaluate the request with the policy. The decision depends on the "rule" that the policy has. Assume that the decision is "permit". Informs the PEP for the decision and the PEP informs the device. Figure 3, shows that architecture.

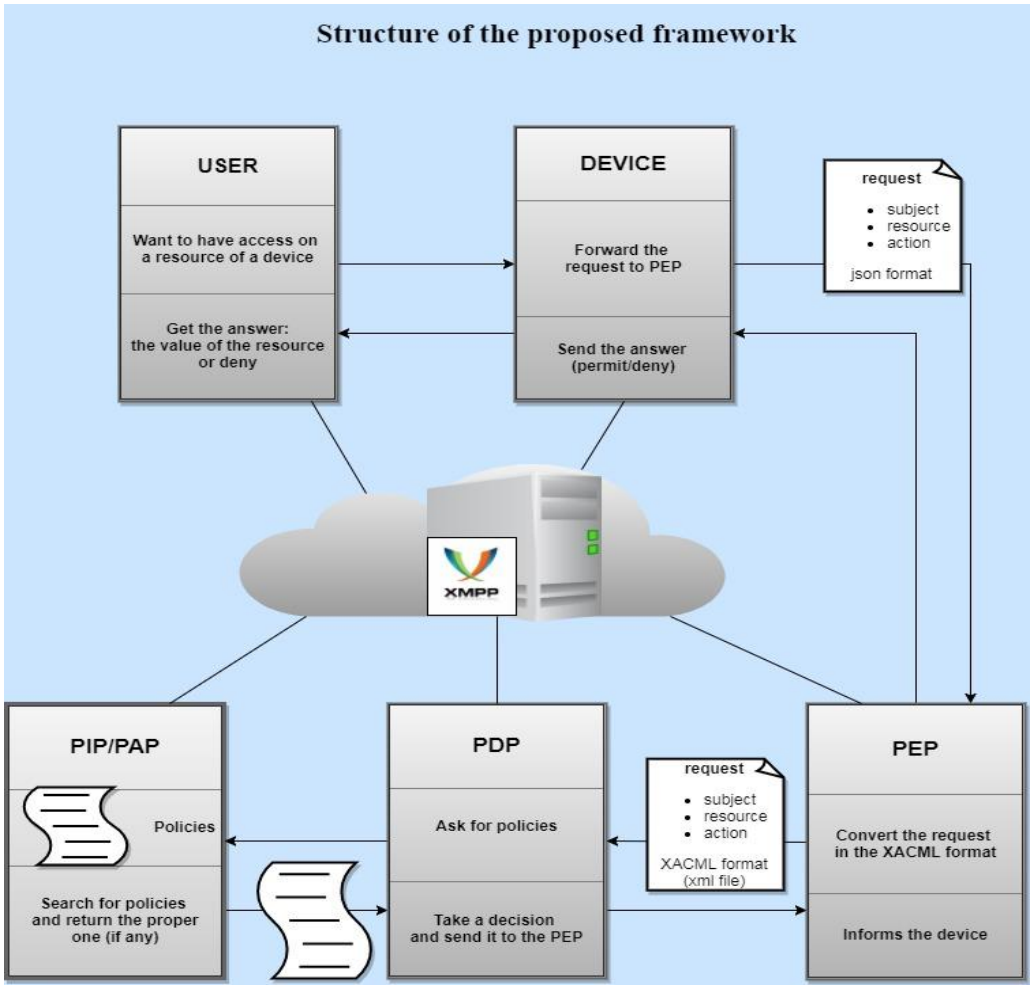

Figure 3 Structure of proposed framework

4.2 Security

Generally, to start communicating with the server, each user must authenticate its flow through traditional authentication methods or through it SASL **[**55**]**. SASL is a framework that enables implementation (server or application) to support different methods authentication according to its requirements. According to the protocol the proposed procedure is SASL. The authentication mechanisms offered by the context are the following:

- PLAIN. The simplest authentication method based on a passphrase. The password is not encrypted, so its use requires it encryption of the communication channel.
- DIGEST-MD5. This method is safer than plain as password is encrypted before it is transmitted to the channel. Nevertheless the method it presents many compatibility problems.
- SCRAM. The suggestion to replace DIGEST-MD5 is SCRAM (Salted Challenge Response Authentication Mechanism). The method offers a high level of security, especially if combined with channel encryption TLS.
- EXTERNAL. This method allows the entity to use its digital certificate to install the TLS and by using the same certificate to authenticate the service. This method is mainly used for communications between servers and for a small number of users who have a digital certificate.
- ANONYMOUS. This method allows users to authenticate themselves without having to register before. When authenticating the user via SASL, the server sends the user one list of available authentication mechanisms in order of preference. The user selects the authentication mechanism who desires and sends them his credentials. Depending on the authentication mechanism chosen follows a series of requests/replies until the user is verified. The details of the authentication process are described in Core RFC.

5 Implementation

At this point, all the details about the tools utilized and the implementation process will be presented and discussed. For this master thesis designed four java projects. These are a user, a device and the components of the XACML (PEP, PDP and PIP). In the following lines each module will be described.

5.1 Communication

According to the XMPP protocol architecture, it is necessary for the communication to set up a xmpp server.

5.1.1 XMPP servers

There are so many servers for this protocol. XMPP began as Jabber and at the begging, it had only one server, the jabberd. Because of the popularity of the protocol many more servers appeared. The most well-known and popular by now are the jabberd server which started in C but now appears to be a mix of C and C++ code, jabberd 2.x which is a rewrite of jabberd. Also, another one is ejabberd which is an XMPP server written in Erlang and Tigase which is a Java server. Last but not least is Openfire - a Java server Openfire Server.

5.1.2 Openfire server

Openfire server **[21]** is an XMPP server, written in Java language and written by JIVE Software. It is easy in installation and management. It offers credibility and uncomplicated operation and is compatible with the implementation of this dissertation. Is responsible for managing user accounts, identifying their credentials (JID and password), logging their activity, and finalize the stanzas.

Below are some security mechanism of the Opnefire server.

5.1.2.1 SSL Certificates

Openfire can generate self-signed DSA and RSA certificates through its Administrator Web Console. These can be used for server-to-server as well as for client-server communication.

5.1.2.2 SSL Guide

Openfire's SSL **[**56**]** support is built using the standard Java security SSL implementation (javax.net.ssl.SSLServerSocket). To make the connection secure, there are two types of a certificate on an SSL server. The first category is named "keystore" and contains the keys and certificates of the server. These security credentials are used to authenticate to clients

and other servers that the server is somewhat operating on behalf of a particular domain. The server only needs to act as one domain; the user only needs one key entry and certificate in the keystore. The second type of certificates is called the “truststore.” The role of "truestore" is to prove that another server (or a client) is legitimately operating on behalf of a particular user. In most of the cases, the truststore is empty. This has the consequence that the server does not attempt to validate the client connections by using SSL. Although, the XMPP authentication method authenticates users in-band. However, for server-to-server communication, it needs an SSL authentication. The role of certificates is to guarantee that a particular part is what they claim to be. Certificates are trusted based on who signed the certificate. A very usual certificate that usually used for internal use on trusted networks is called “self-signed” certificate. The communication channel encrypted. Either the client of r server must verify their self-signed certificate through some other channel. For higher security deployments, users it is better to get the certificate signed by a certificate authority (CA).

5.1.3 Smack library

Smack library **[**57**]** is an Open Source XMPP (Jabber) client library for instant messaging and presence. A pure Java library, it can be embedded into the applications to create anything from a full XMPP client to simple XMPP integrations such as sending notification messages and presence-enabling devices. Some basics classes are described below.

The main class for communication that our projects will use is the XMPPConnection. As the name refers, it represents an XMPP connection to a remote server. A more sophisticated configuration of the connection attributes is possible, and that can be achieved by using the ConnectionConfiguration class. Multiple parameters can be configured, with the most significant being the server host and port.

```
ConnectionConfiguration config = new ConnectionConfiguration(server, 5222);
XMPPConnection connection = new XMPPConnection(config);
connection.connect();
connection.login("thermostat", password);
```

Table 10: XMPP Configuration code

From a valid XMPPConnection, it can recover a ChatManager object. ChatManager keeps track of references to all current chats and allows to create Chat instances, i.e. series of messages sent between two users/devices. Chat object is able to send messages to other chat participants. It can use plain strings or construct XMPP message packets, represented by the Message class. Message class provides direct access to all of the message attributes, such as the message type. For processing incoming messages, it must to implement the MessageListener interface and attach it to chat.

```
ChatManager chatmanager = connection.getChatManager();
MyMessage messg = new MyMessage(chatmanager);

connection.getChatManager().addChatListener(new ChatManagerListener() {
        public void chatCreated(final Chat chat, final boolean createdLocally) {
           chat.addMessageListener(new MessageListener() {
              public void processMessage(Chat chat, Message message) {
                 System.out.println("\n Received message: "
                       + (message != null ? message.getBody() : "NULL"));
              } }); } });
```

Table 11: ChatManager code

For each connection of projects is created once ChatManager. That means that for the communication between the user and the device it created once for sending and received messages and it keeps open until the user decides to close the connection.

```
public void sendMessage(String message, String buddyJID2) throws XMPPException     {
      System.out.println(String.format("Sending message '%1$s' to user %2$s", message,
buddyJID2));
      if (chat == null) {
         chat = chatManager.createChat(buddyJID2, messageListener);
         buddyJID = buddyJID2;
         chat.sendMessage(message);
      } else {
         if (buddyJID2.equalsIgnoreCase(buddyJID)) {
            chat.sendMessage(message);
         } else {
            chat = chatManager.createChat(buddyJID2, messageListener);
            buddyJID = buddyJID2;
            chat.sendMessage(message);
         }      }    }
```

Table 12 Created ChatManager code

5.2 User

User's basic operation is to send a request to the device and asks for information of a resource. For this, it has designed a simple user interface program which allows each user to discover available devices of servers. The interface of the program is shown in the figure above.

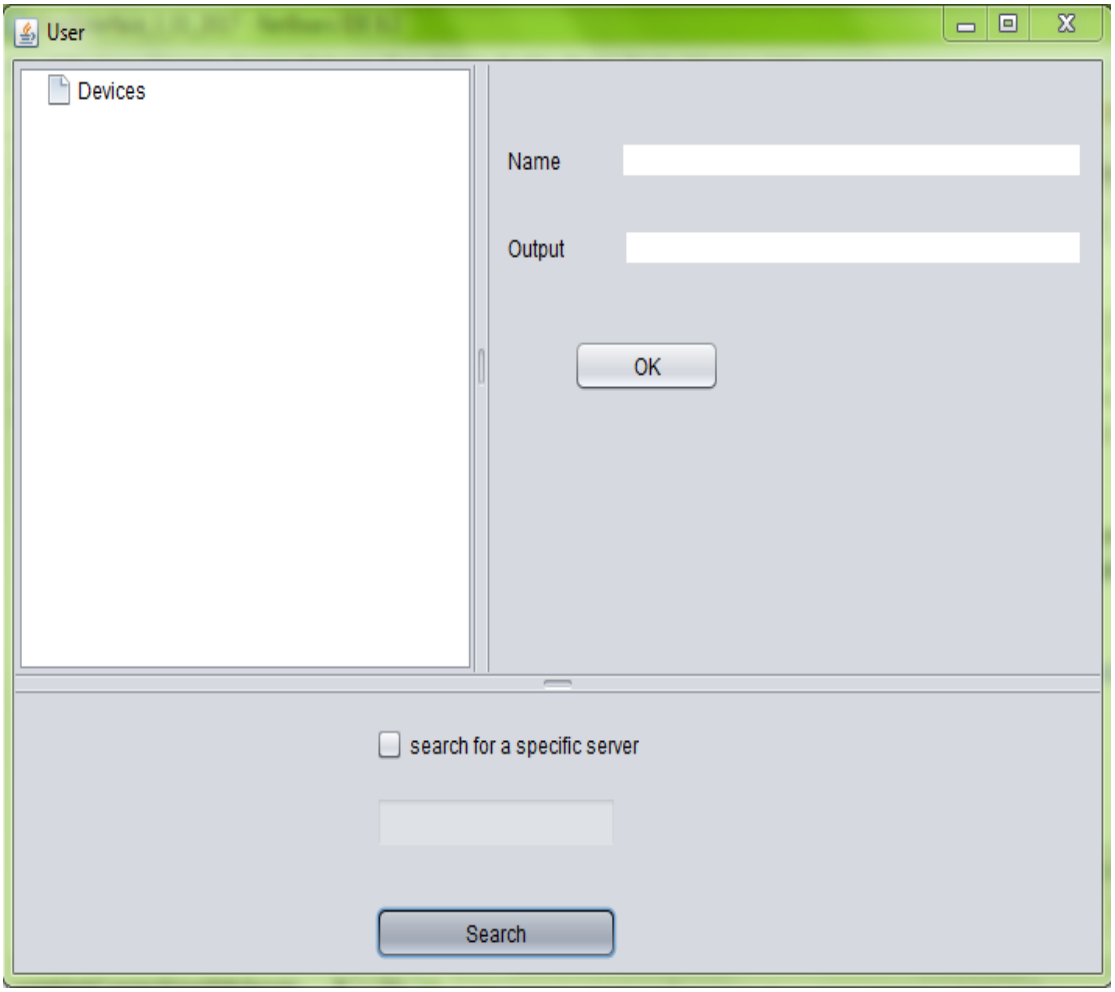

Figure 4 User interface

The user has two choices. The first one is just to push the button "search " and automatically a request sends to the central server and that server shows the online servers that are connected to him and shows their devices. The other choice is to select a specific server that wants and find out the available devices of the servers and their available resources. The two options can combine each other, and the user can have the two options together.

Figure 5 shows the servers that are connected to the main server. For this example, it had chosen two public xmpp servers (jid.ooo and jabber.network)

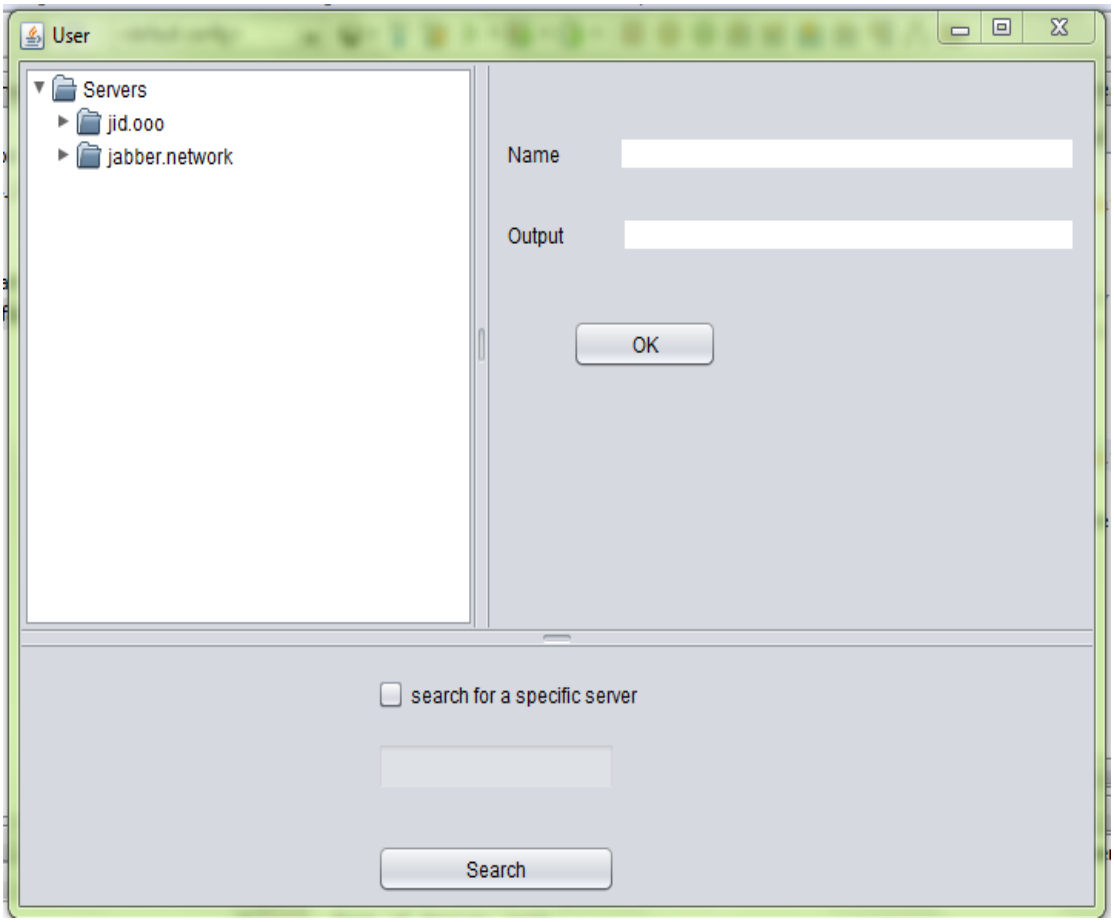

Figure 5 An overview

After the searching of a specific server, which is an openfire server with the name "pits".

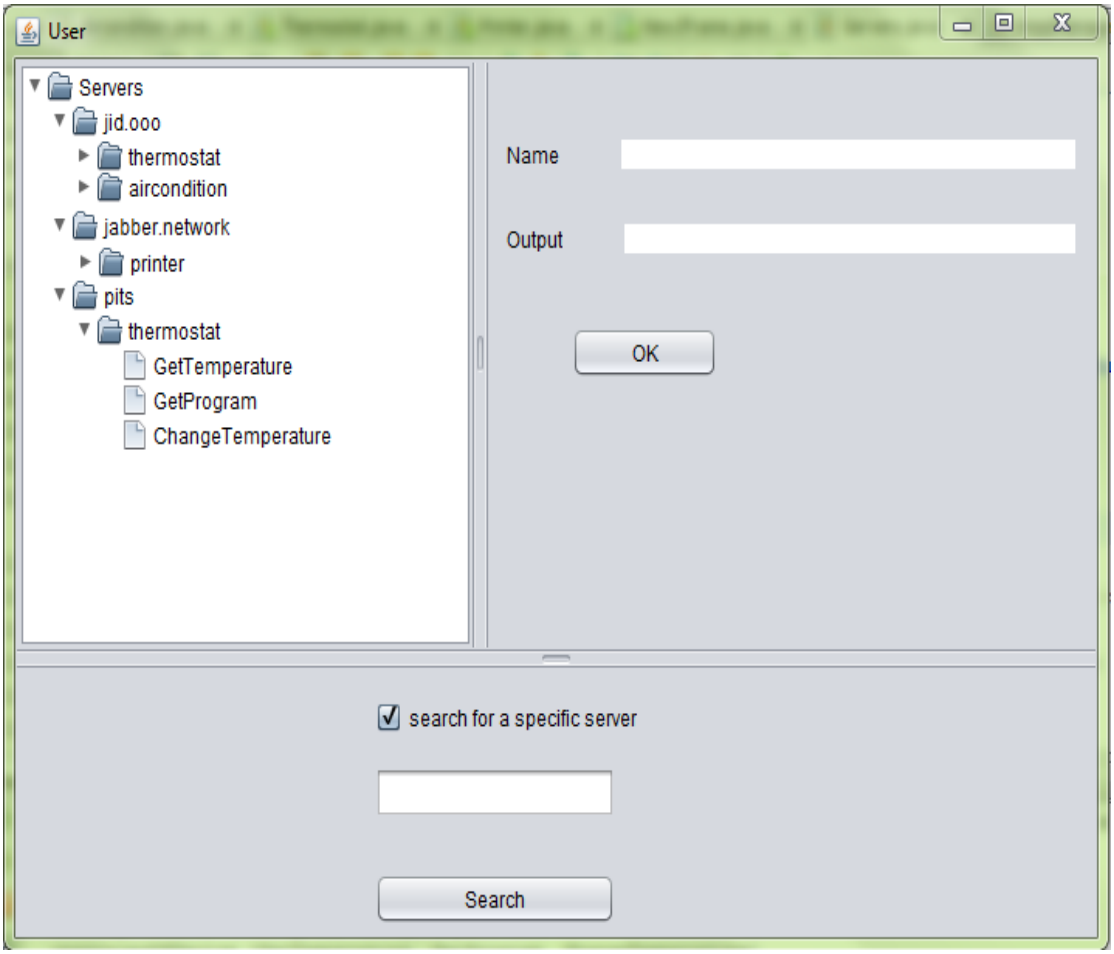

Figure 6 All servers and devices

In this figure, the user after all the choices can choose which device and resource want to access. In the paradigm below, want the server named "pits" and want to get the information about the temperature of the device thermostat. As seen in the figure the user cannot have the rights to know the temperature because the answer is denied.

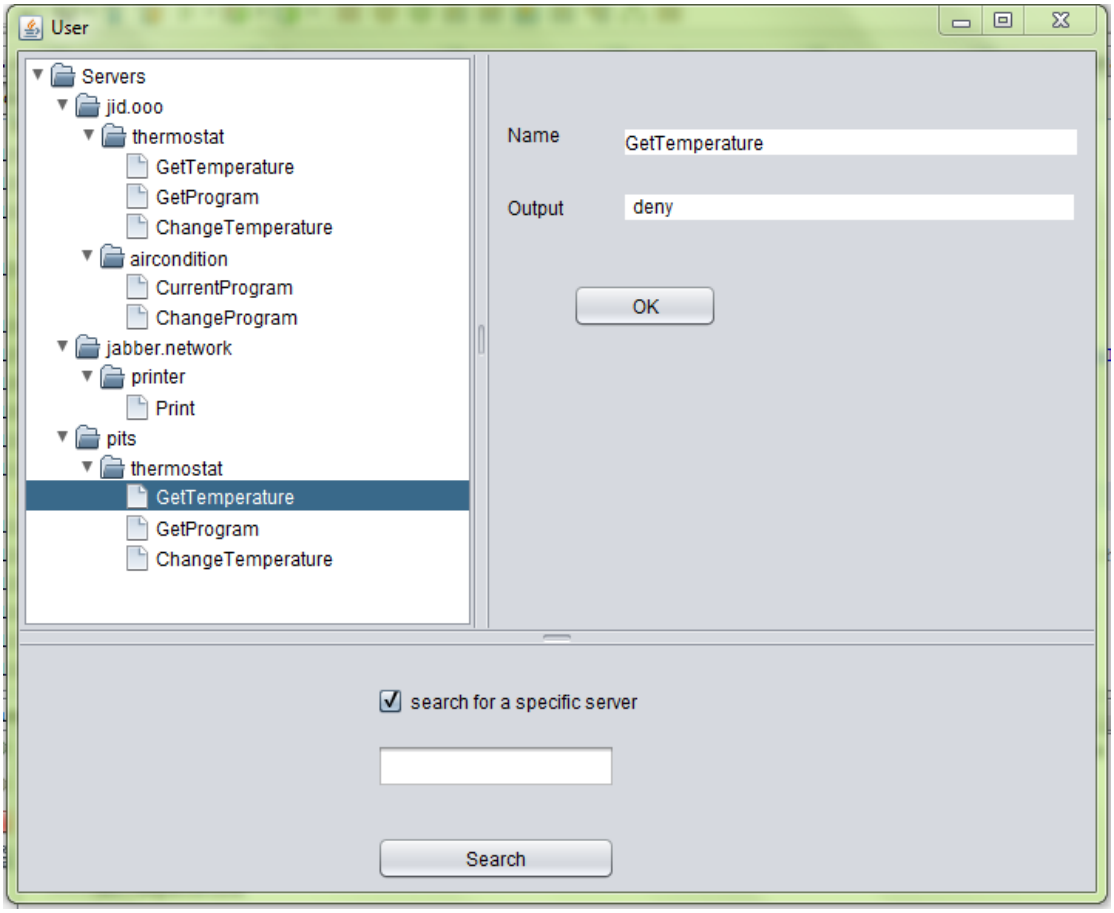

Figure 7 Result

5.3 Device

For the framework, it designed a pretty simple device. It consists a class named Informations. This class includes all the resources that a user can have access, like, if the device is a thermostat one of them can be the current temperature. The other class is named MyMessage and consists the communications functions of send and receive the messages. When the user starts to search for the devices and the information that can access with the device sends its pieces of information. After the user chooses i.e. the temperature, the device receives a message which contains the temperature and the name of the user. After that, a json object is formed which consist the Subject(name of the user), the Action(temperature) and the Resource(name of the device, i.e. Thermostat) and forwarded to PEP.

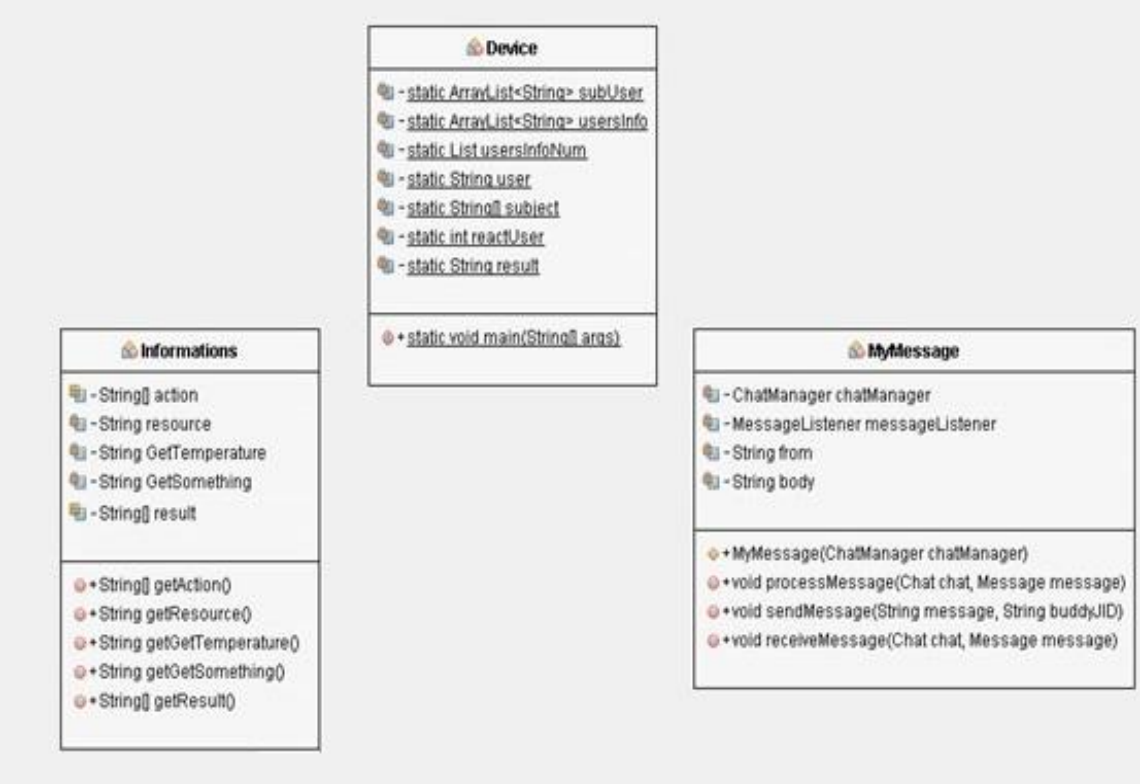

Figure 8 Device

5.4 PEP and PDP

In the same project had designed the component of PEP and the component of PDP. When PEP receives the message which consists the request in a json object format. As mentioned in a previous chapter, the PDP understands the XML language. For this, is called the function requestToPDP(Subject, Action, Request). In the same time while policyFinderShell is asked to find a policy for a certain request it makes a string from the request and passes it at the getRemotePolicy of pipCient. The getRemotePolicy returns an array of two strings which are derived by the reverse procedure that is used in xmppPipOperation with a simple variation if a policy is found it is returned to position 1 of the array else both positions are null of the other combinations (non-applicable, cannot determine, other error). Then the policyFinderShell uses that info to make a policy appropriate for the PDP and returns to the device the answer.

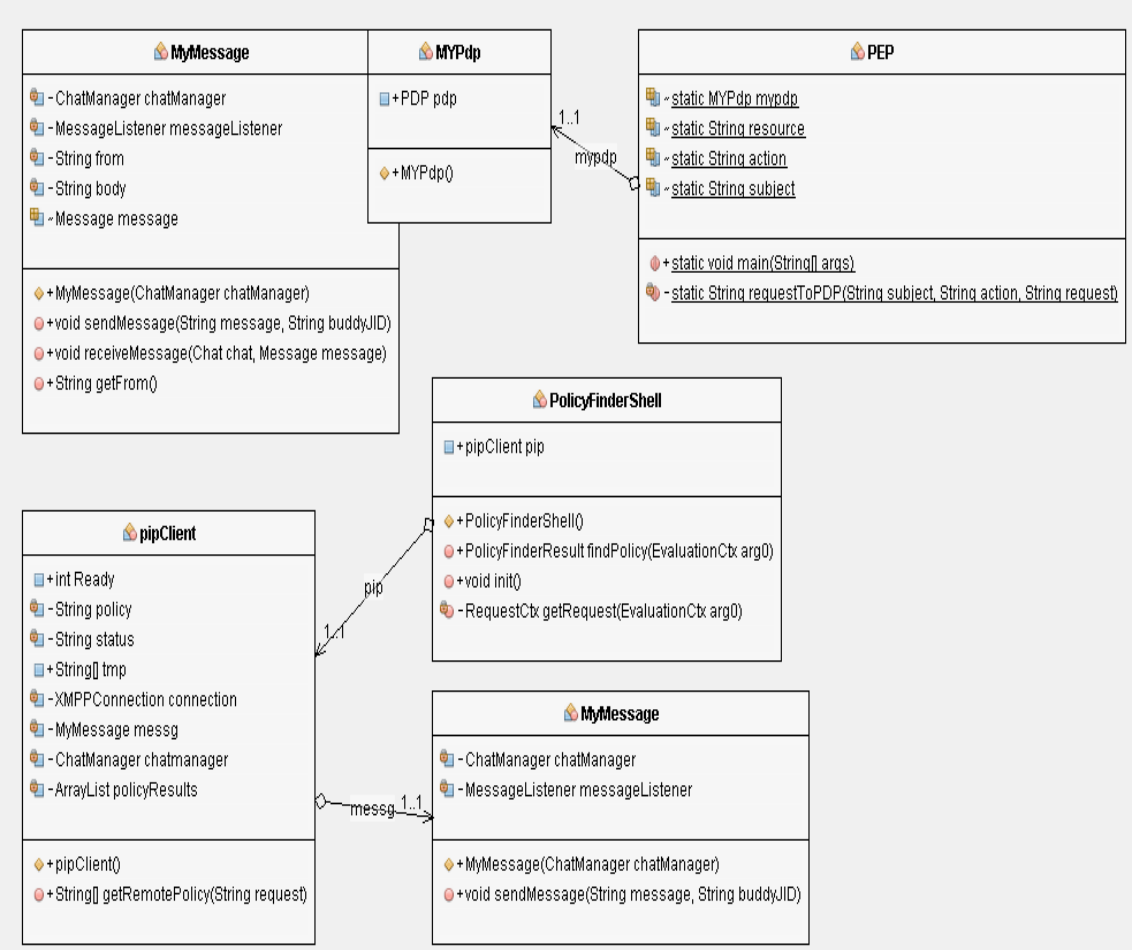

Figure 9 Class diagram of PEP/PDP

5.5 PIP

As mentioned in a previous chapter the role of the component PIP in the XACML schema is to match the request that the PDP sends with the policies that keep. The basic class in PIP project is the xmppPIPOperation. When the policyFinder is instantiated the FilePolicyModule from Sun's implementation is used to handle each policy file. The policies are found at a folder named policyDir. When its needed to find a policy in order PDP can evaluate a request the policyFinder.findPolicy method is called.

The basic operation of xmppPIPOperation is to find in the folder PolicyDir if there is a policy that matches the request, for that PIPResponce evaluate(String request) is responsible. It returns two values, the policyResponse and the policyStatus, which send to pipClinet to PDP.

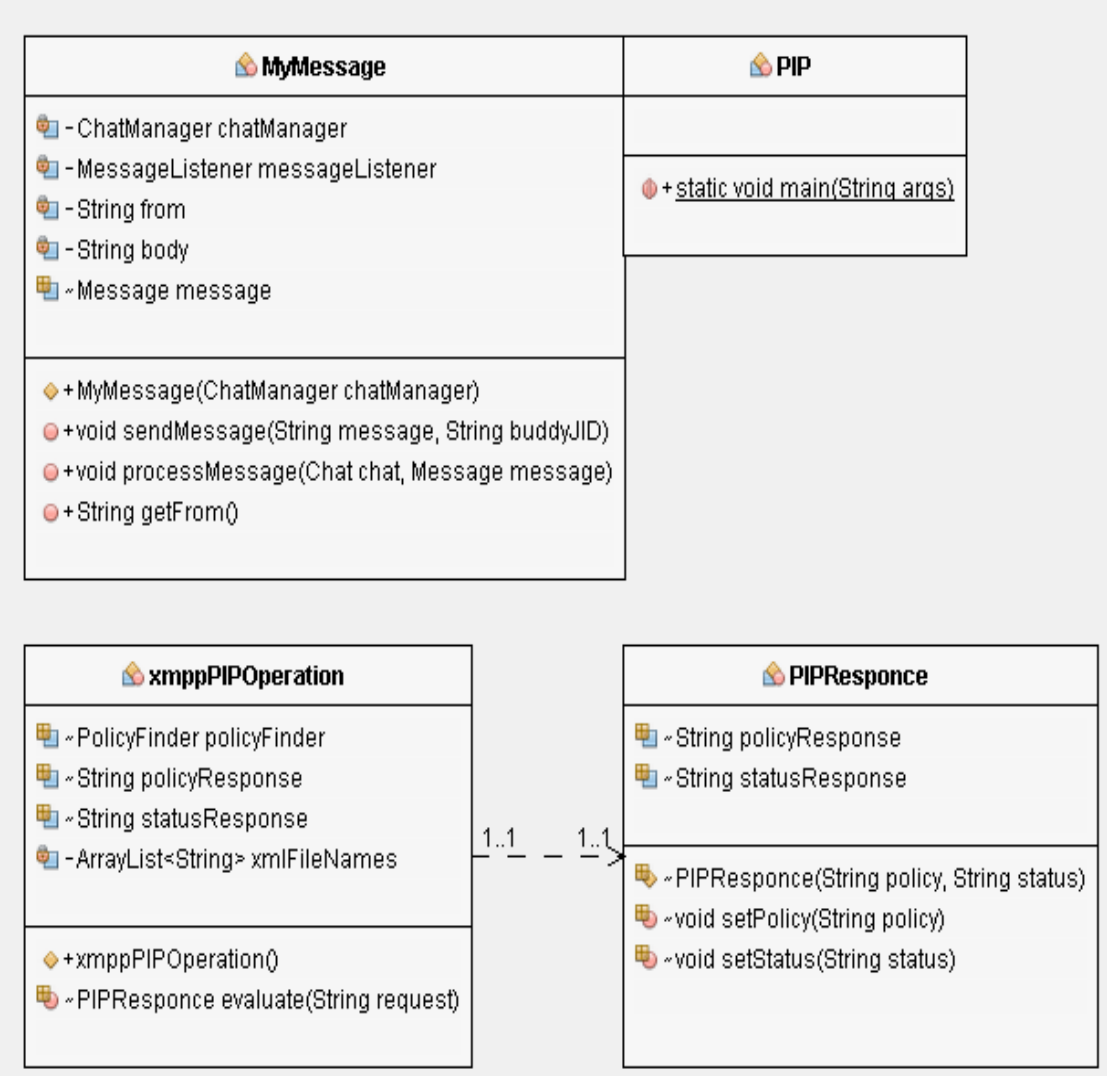

Figure 10 Class diagram of PIP

6 Evaluation

This chapter will present how the proposed framework is implemented and tested. Figure 14 is shown where every component is placed.

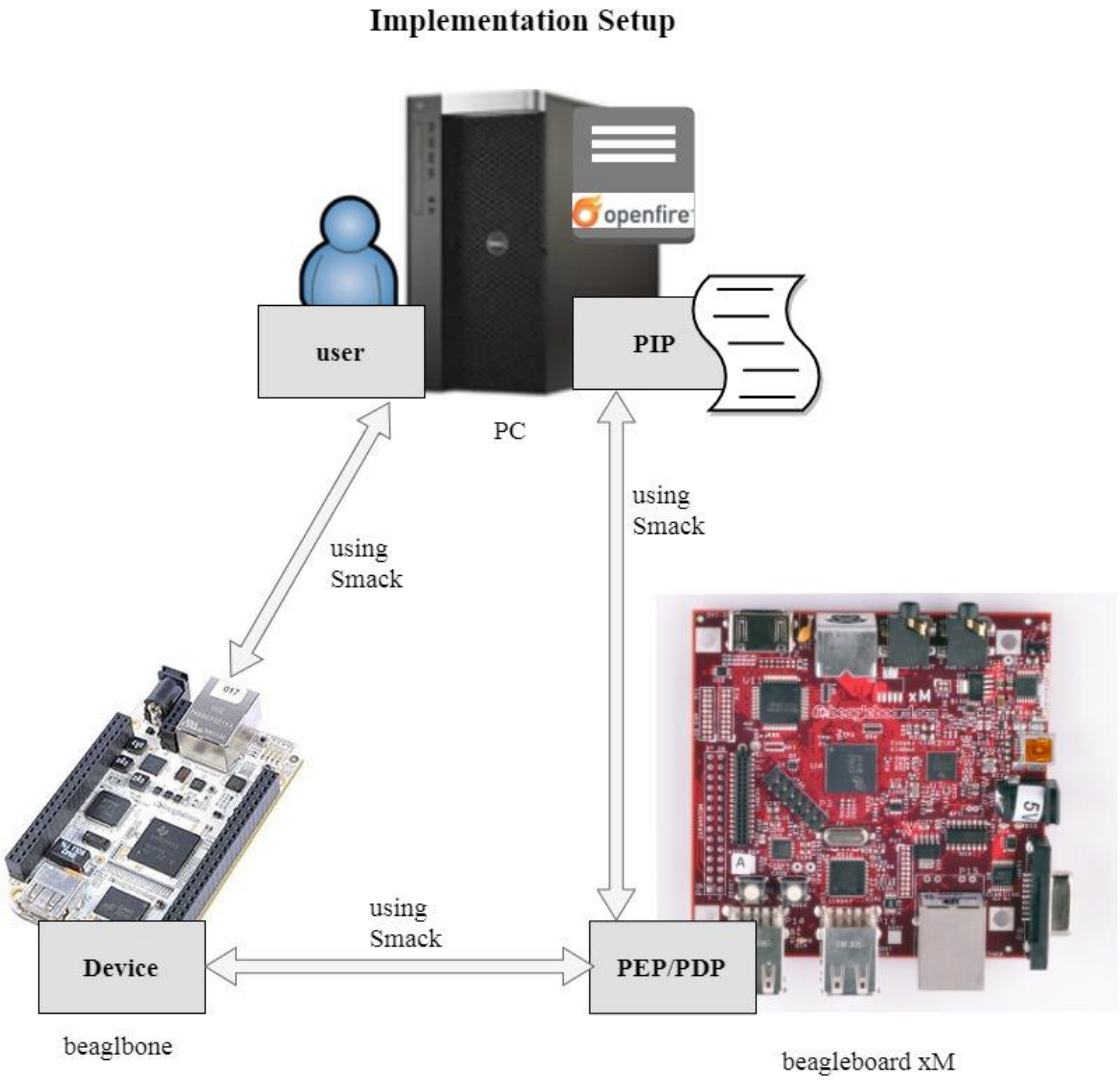

Figure 11 Implementation setup

The devices that used for this implementation test are a beagleboard xM, a beaglebone and a PC. In more details, the beagleboard xM is with 1GHz ARM and 512 MB LPDDR RAM running Ubuntu Linux. The bleagbone features are a ARM Cortex-A8 running at 720MHz and 256 MB also running Linux-based operating system. The PC processor is an Intel Xeon E5-2630V2 with 6 cores and running at 2.6Ghz. The RAM is 32GB and is DDR3 SDRAM - ECC clocked at 1866MHz At beagleboard xM is running the source code of the PDP and PEP and the device is running on the beaglbone. The user, the PIP and the xmpp server running on the PC.

6.1 Scenarios

To determine whether the proposed framework of this work yields, it will be evaluated in the same three scenarios with and without the XACML schema.The first three scenarios are without the proposed framework and the other three with it. The concept of the all the scenarios is a communication between a user and a device. Assume that the user finds out a device which is a thermostat and wants to know the current temperature.

6.1.1 Scenario 1

The user sends 100 sequential requests to the device asking for the temperature. The device just response with the information of the temperature.

6.1.2 Scenario 2

The user sends 1000 sequential requests to the device asking for the temperature. The device just response with the information of the temperature.

6.1.3 Scenario 3

The user sends 30 requests to the device. In contrast with the other two scenarios, every time that user gets the response waits for 30 seconds and then sends again the new request.

6.1.4 Scenario 4

The user sends 100 sequential requests to the device asking for the temperature. This scenario follows the proposed framework. The devices are connected to the same switch.

6.1.5 Scenario 5

The same scenario with scenario 4. The difference is that the connection is more realistic as the devices are not connected to the same switch.

6.1.6 Scenario 6

In this scenario, the user sends 30 requests. The difference with the other two scenarios is that between the requests is an interval time about 30 seconds. This scenario also follows the proposed framework.

6.1.7 Summary

Scenarios	Description	Access Control
1st	100 sequential requests to the device	No
2nd	1000 sequential requests to the device	No
3rd	30 requests to the device . Each request waits for 30 sec	No
4th	100 sequential requests. All the devices are connected to the same switch	Yes
5th	100 sequential requests.	Yes
6th	30 requests to the device with interval time 30 sec	Yes

Table 13 Summary

6.2 Results

For the above scenarios, we are going to evaluate how the CPU and the memory usage react in each scenario. Also, is very important to evaluate how fast or not is the proposed protocol. In the following figures for the delay, usually the first two values are much bigger than the others and the reason is for the handshake. For the sake of readability of the charts, we will not put the value of the first two requests.

According to the bellow figures, in scenarios 1 and 2 it seems that the CPU and the memory are not so stable during the communication. That is because of the garbage collection that java does for minimizing the usage of CPU. In scenario 3 the CPU load and the react of memory are also not stable. In the middle of the communication, it seems that the device maybe loss the connection with the server and start it again.

In scenario 4, the proposed framework is tested in a ideal network that all the devices are connected to the same switch in contrast to scenario 5 which is more realistic because the devices are connected in a more realistic network with other hops. We see that the delay on the user side is a bit bigger in scenario 5 than in scenario 4. Both the CPU and the memory on the device and stable and use a small load of CPU (only around 12%) and that is because they are waiting some milliseconds to get and forward the message. Both, the CPU load is very high and that's because on that device the PEP converts the request in an XACML format and the PDP has to get the decision. The PIP uses only 1,5% of the CPU because of the features of the PC. We see variation but on the PC are also running the xmpp server and the client. In scenario 6, the delay is quite stable. The memory has the same usage of bytes for all the communication In CPU its seems that at some points the session may be closed and starts again. The CPU in PDP and

PIP is very low because of the 30 seconds delay in every request. The memory in scenarios 4, 5 and 6 is stable because of the caching of memory.

6.2.1 Scenario 1

User
Average Time (ms)
22,17

Table 14 Scenario 1: Average time delay on user

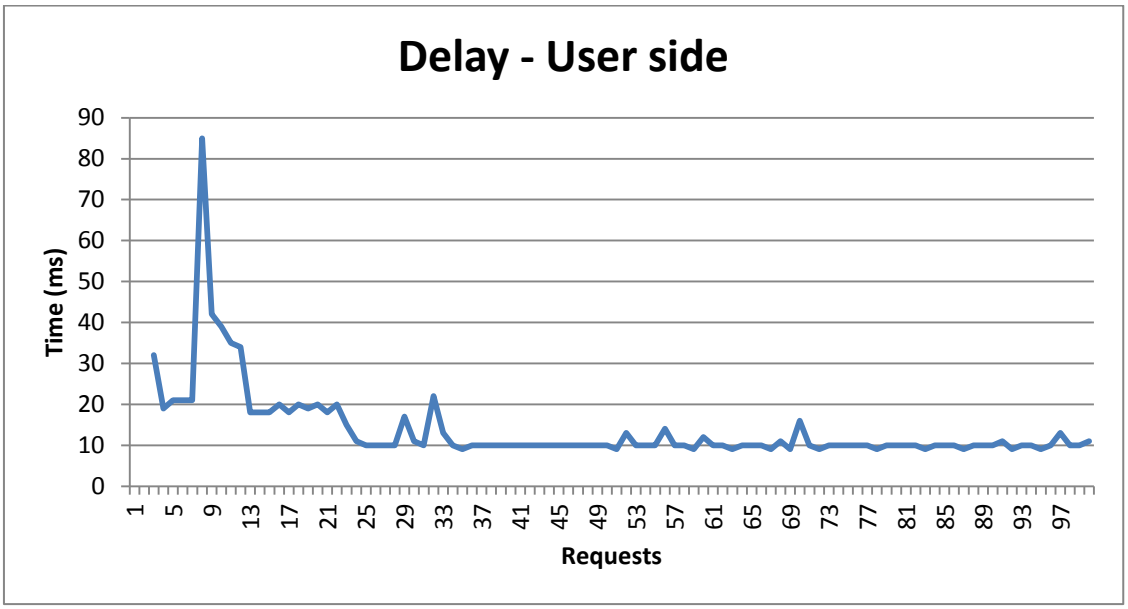

Figure 12 Scenario 1: Delay on user

The value of the first request is 215ms and 646ms on the second one. We notice that after the first 13 requests and during all the following; time starts stabilizing respectively.

Device			
CPU(%)	**Memory (bytes)**	**Rx(bytes)**	**Tx (bytes)**
96,7	41466429,44	483,36	686,72

Table 15 Scenario 1: Average CPU, Memory, Rx/Tx on device

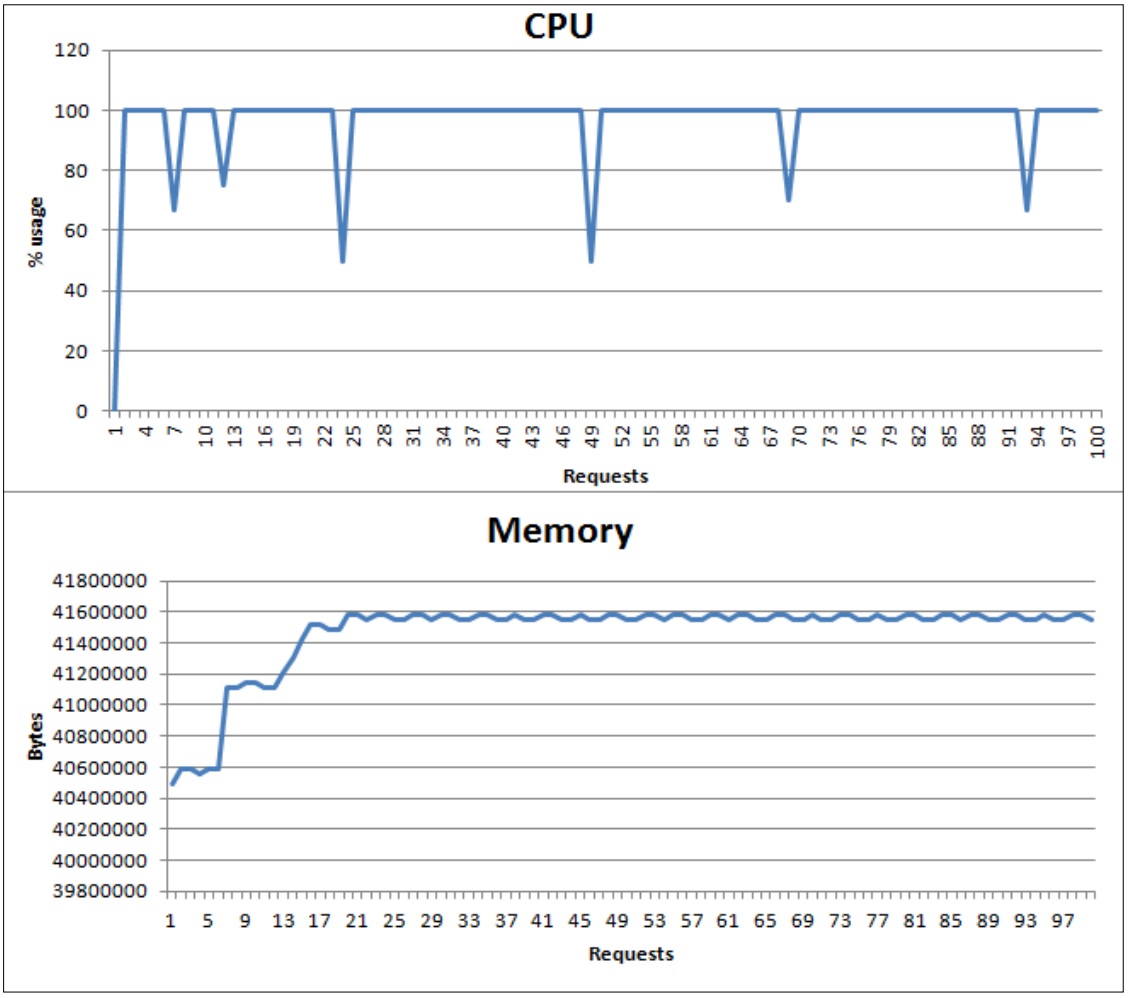

Figure 13 Scenario 1: CPU and Memory on device

The average usage of CPU on device is 96,7. In Figure 13, it seems that the CPU after some requests has some abrupt variation. This happens because of the garbage collection that java does. The memory after 20 requests of the same message uses a stable number of bytes and it actually makes sense because of memory's caching.

6.2.2 Scenario 2

User
Average Time (ms)
8,59

Table 16 Scenario 2: Average time delay on user

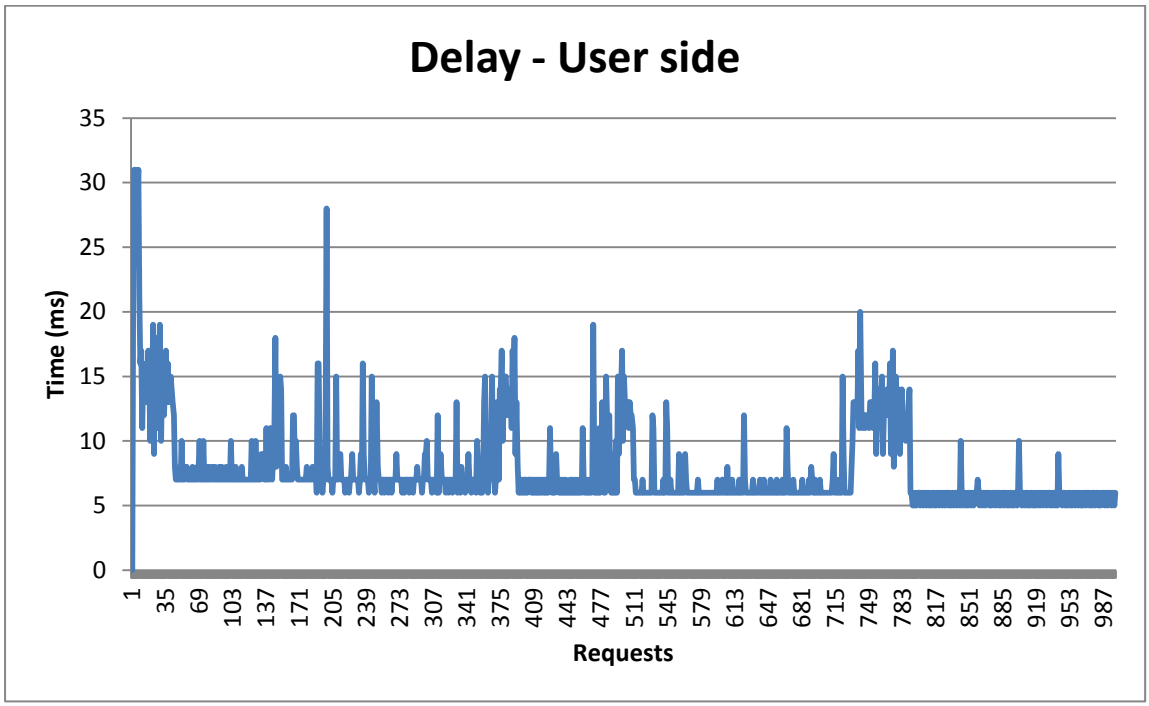

Figure 14 Scenario 2: Delay on user side

Figure 14 shows the time for each request, from the point where the user asks for an answer until the answer arrives . There is an instability for every request with lots of diversity .

Device			
CPU(%)	**Memory (bytes)**	**Rx(bytes)**	**Tx (bytes)**
98,6	40197726,21	212,9	203,8

Table 17 Scenario 2: Average CPU, Memory, Rx/Tx on device

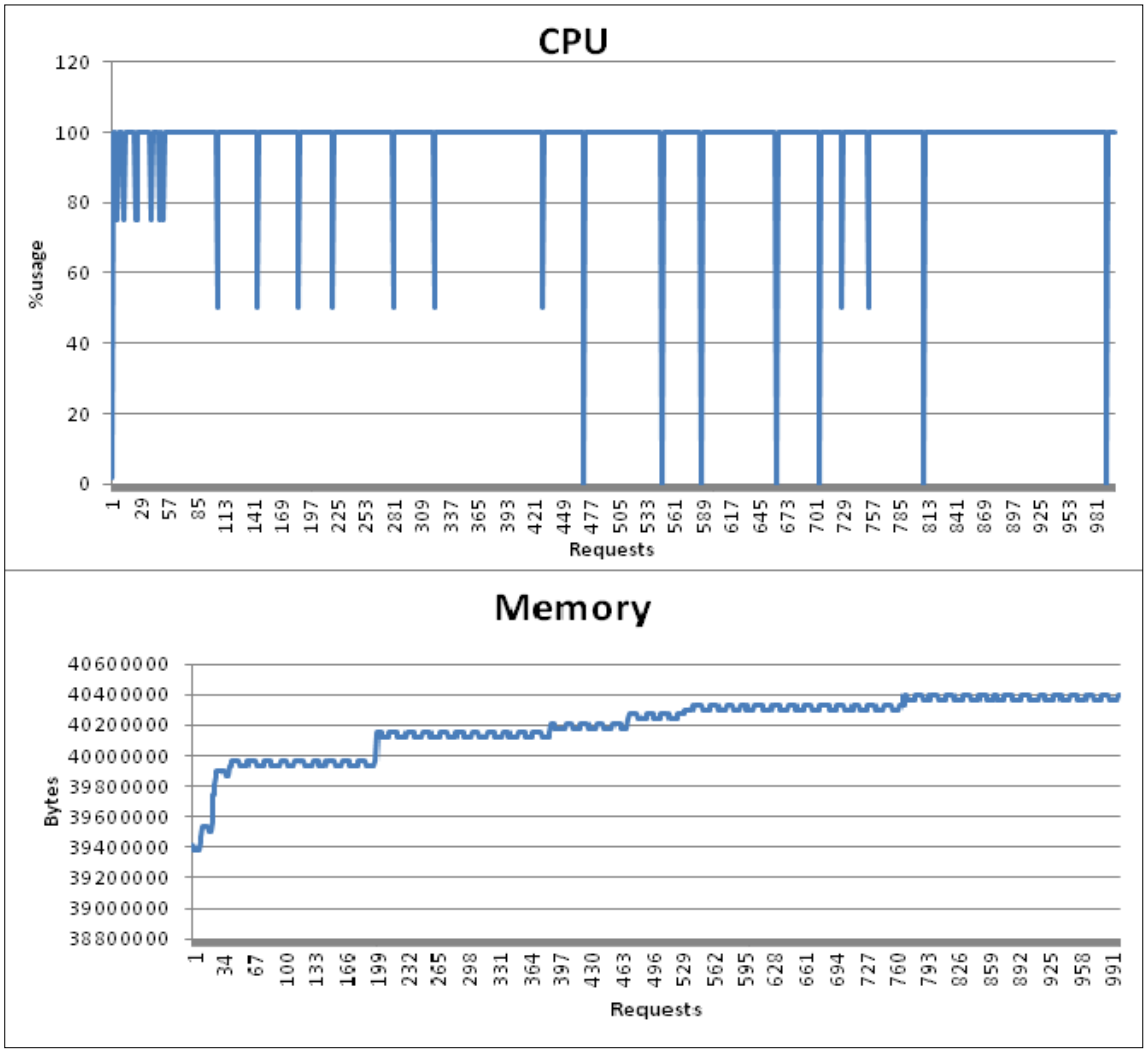

Figure 15 Scenario 2: CPU and Memory on device

In scenario 2, the CPU has many variations during the communication and this occures because of the garbage collection that tries to minimize the usage of CPU. The usage is very high, 98,6% and that was expected because of the massive number of requests. Concerning the memory; the number of bytes keeps increasing progressively.

6.2.3 Scenario 3

User
Average Time (ms)
14,6

Table 18 Scenario 3: Average time delay on user

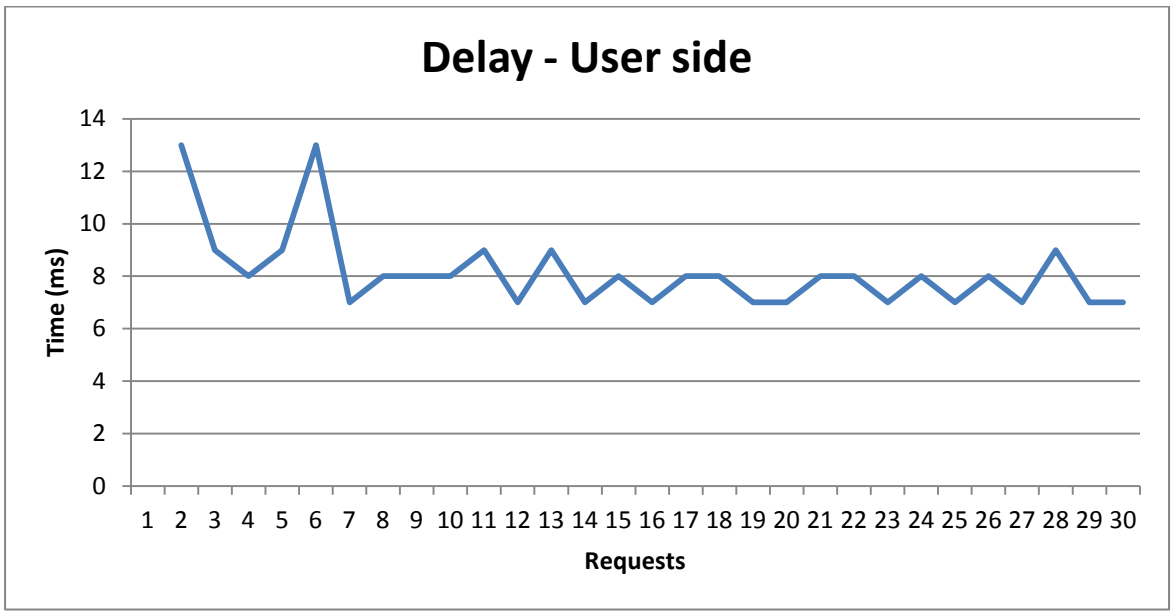

Figure 16 Scenario 3: Delay on user side

The value of the first request is 203ms. After the firsts requests that need more time, every request needs about 8ms.

Device			
CPU(%)	**Memory (bytes)**	**Rx (bytes)**	**Tx (bytes)**
2,16	43752380	39918,53	53047

Table 19 Scenario 3: Average CPU , Memory, Rx/Tx on device

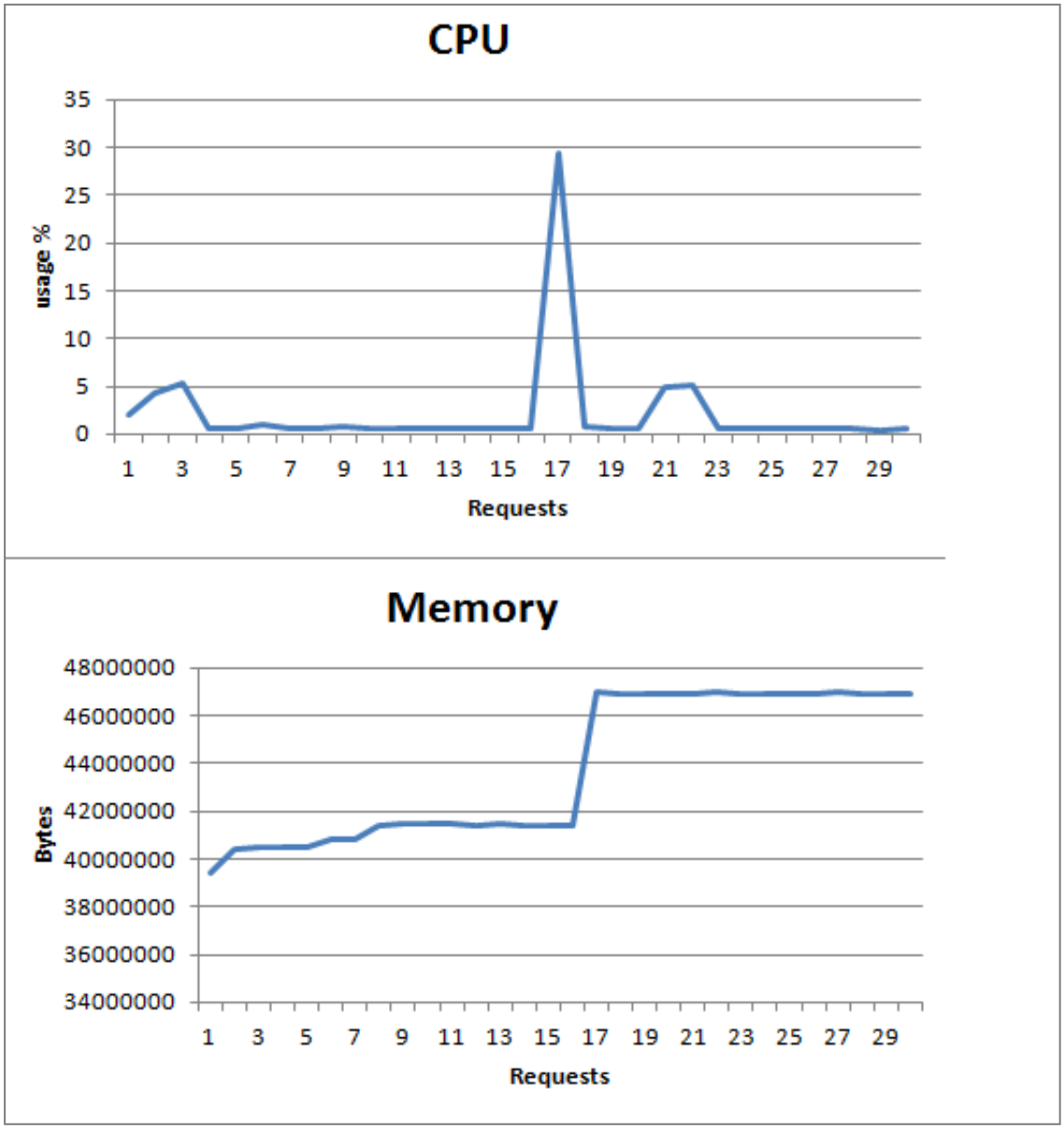

Figure 17 Scenario 3: CPU and Memory on device

The average usage of the CPU is very low, 2,16%. This occurs because of the delay in each request . However it seems that during the communication connection stops and reconnects and this is how CPU's usage is very high on the 17th request. The memory is stable during the communication, except the 17th request that maximizes and uses much more bytes.

6.2.4 Scenario 4

User
Average Time (ms)
327,6

Table 20 Scenario 4: Average time delay on user

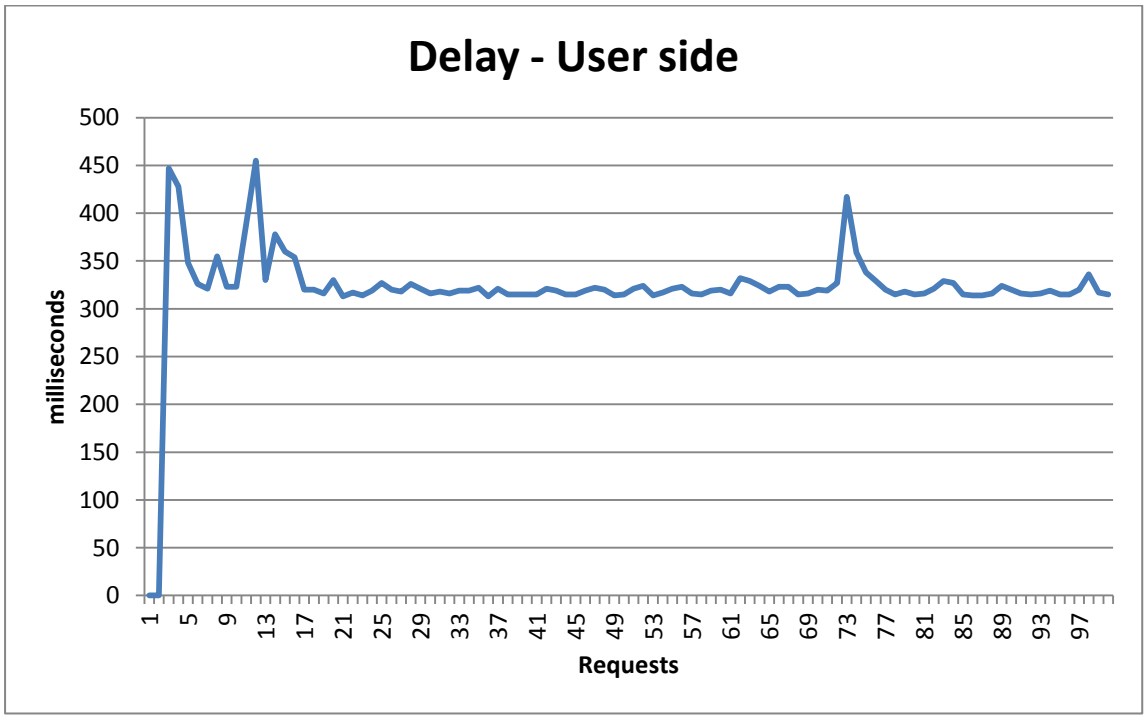

Figure 18 Scenario 4: Delay on user side

The value of the first request is 2535 ms and 1071ms on the second one. For reasons of readability of the charts, we will not add those requests. In that scenario the time that is needed for each request is much bigger and the reason for that is the proposed framework. Although after the first requests the time is stable around the 320ms.

.

Device			
CPU(%)	**Memory (bytes)**	**Rx (bytes)**	**Tx (bytes)**
12,65	166699827,2	817,77	795,64

Table 21 Scenario 4: Average CPU, Memory, Rx/Tx on device

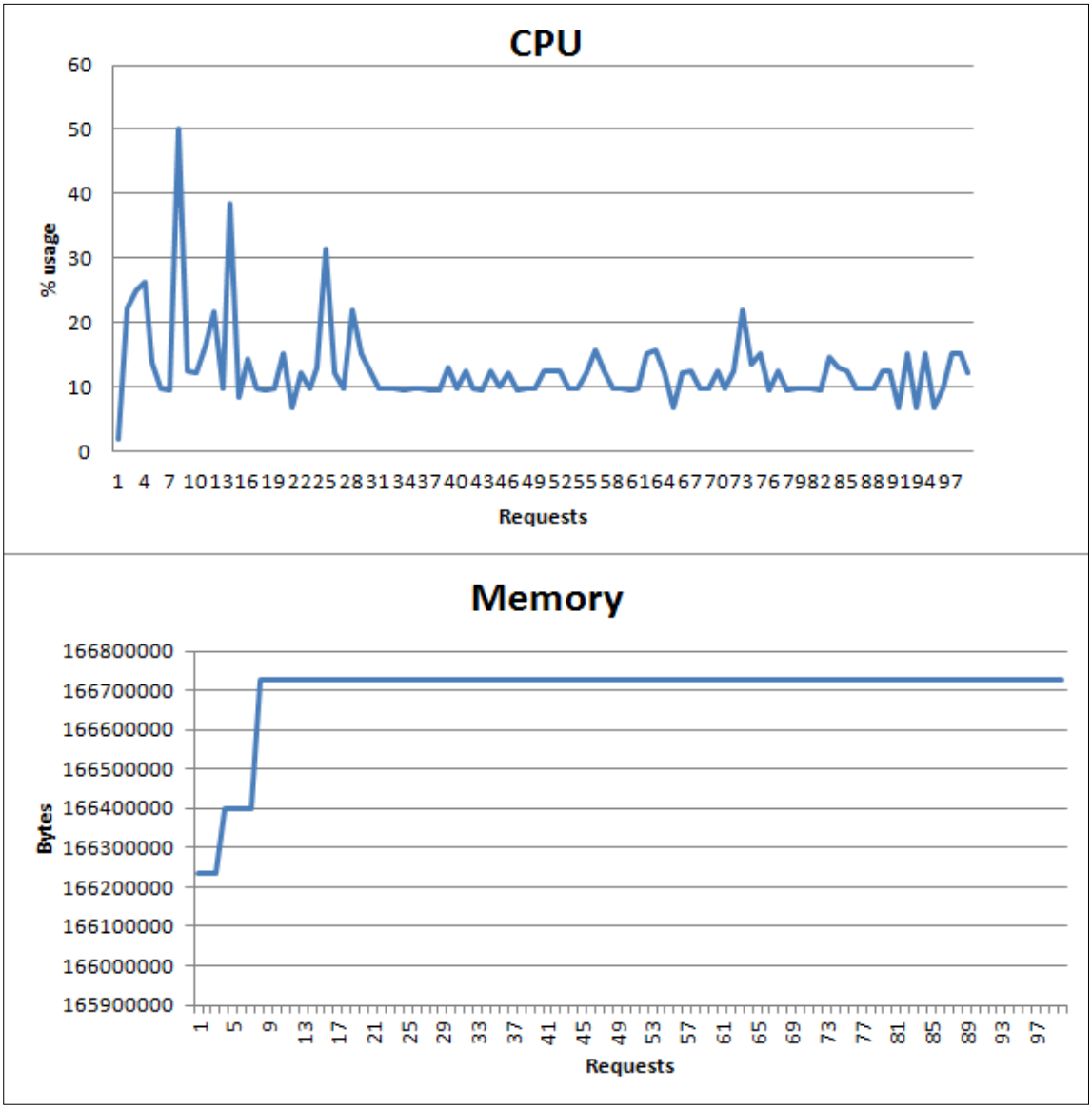

Figure 19 Scenario 4: CPU and Memory on device

The usage of CPU on device is low but not stable in comparison with memories' stability.

PDP			
CPU(%)	**Memory (bytes)**	**Rx (bytes)**	**Tx (bytes)**
86,63	210194432	3303,02	1449,24

Table 22 Scenario 4: Average CPU, Memory, Rx/Tx on PDP

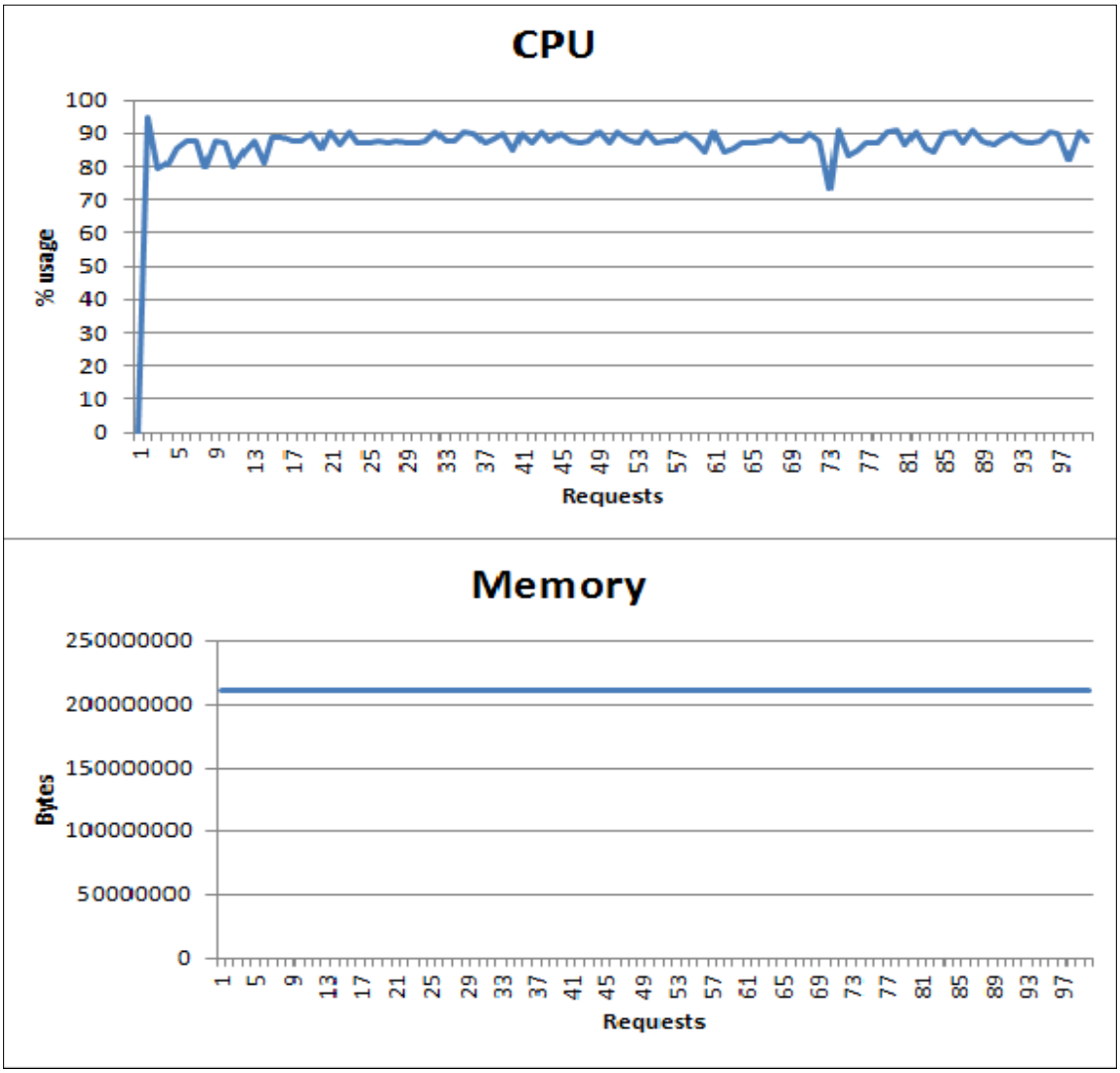

Figure 20 Scenario 4: CPU and Memory on PDP

As it was expected, the CPU load on PDP is high and the bytes that needed are stable.

PIP			
CPU(%)	**Memory (bytes)**	**Rx (bytes)**	**Tx (bytes)**
1,3	12986150912	2952,04	5104,36

Table 23 Scenario 4: Average CPU, Memory, Rx/Tx on PIP

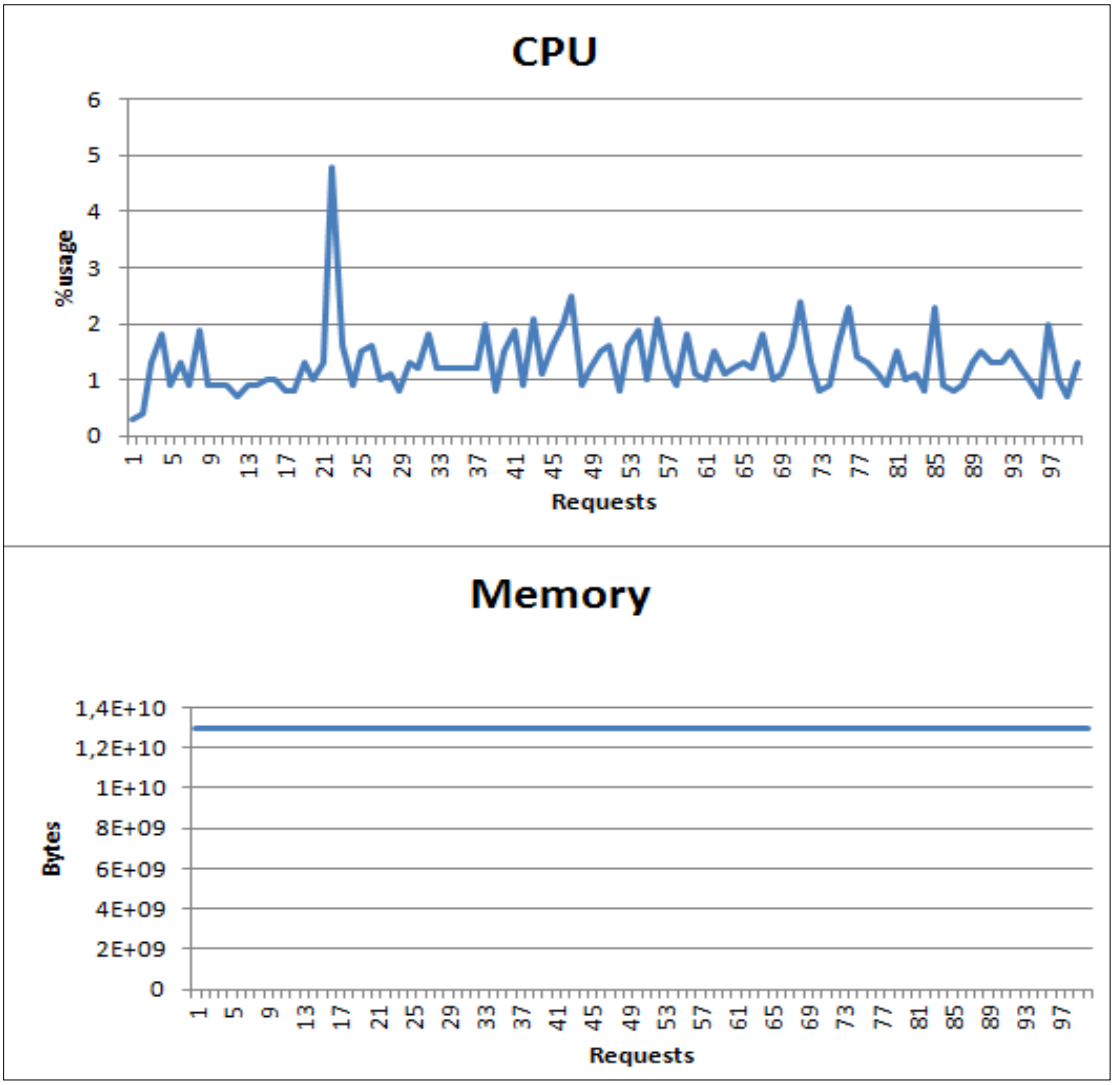

Figure 21 Scenario 4: CPU and Memory on PIP

The usage of CPU is extremely low because of the machine that runs this entity. The bytes of memory are remain same for all the requests.

6.2.5 Scenario 5

User
Average Time (ms)
391,2

Table 24 Scenario 4: Average time delay on user

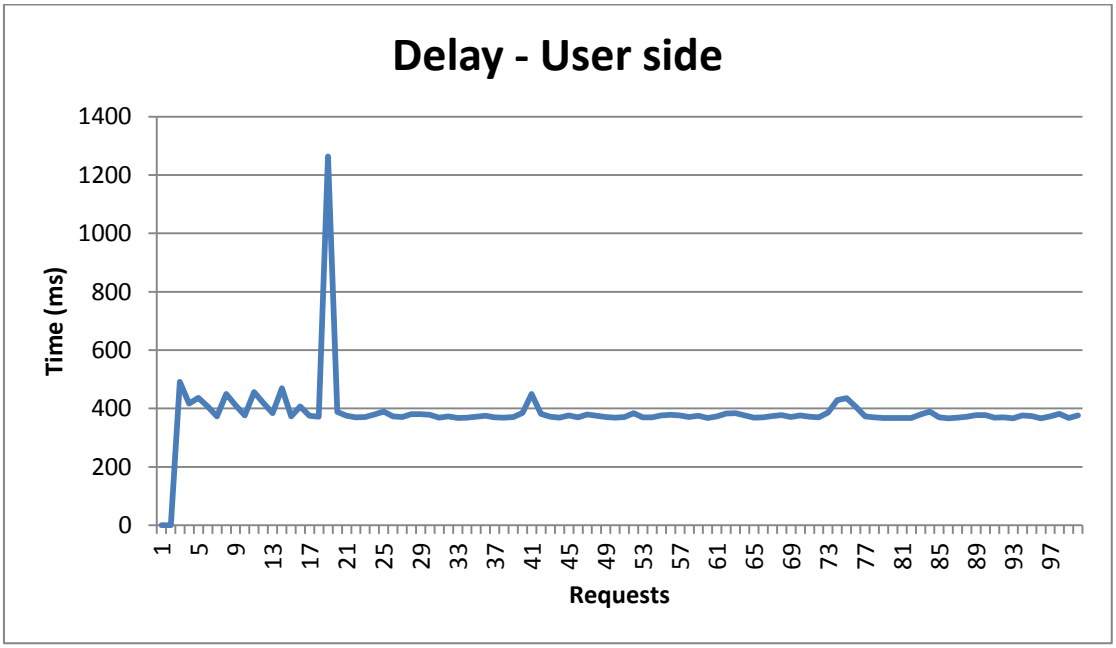

Figure 22 Scenario 5: Delay on user side

Because of a more realistic scenario (than scenario 4) the time that is needed is quite higher, but still stable.

Device			
CPU(%)	**Memory (bytes)**	**Rx (bytes)**	**Tx (bytes)**
13,235	166759629	858,98	797,5

Table 25 Scenario 4 : Average CPU, Memory, Rx/Tx on device

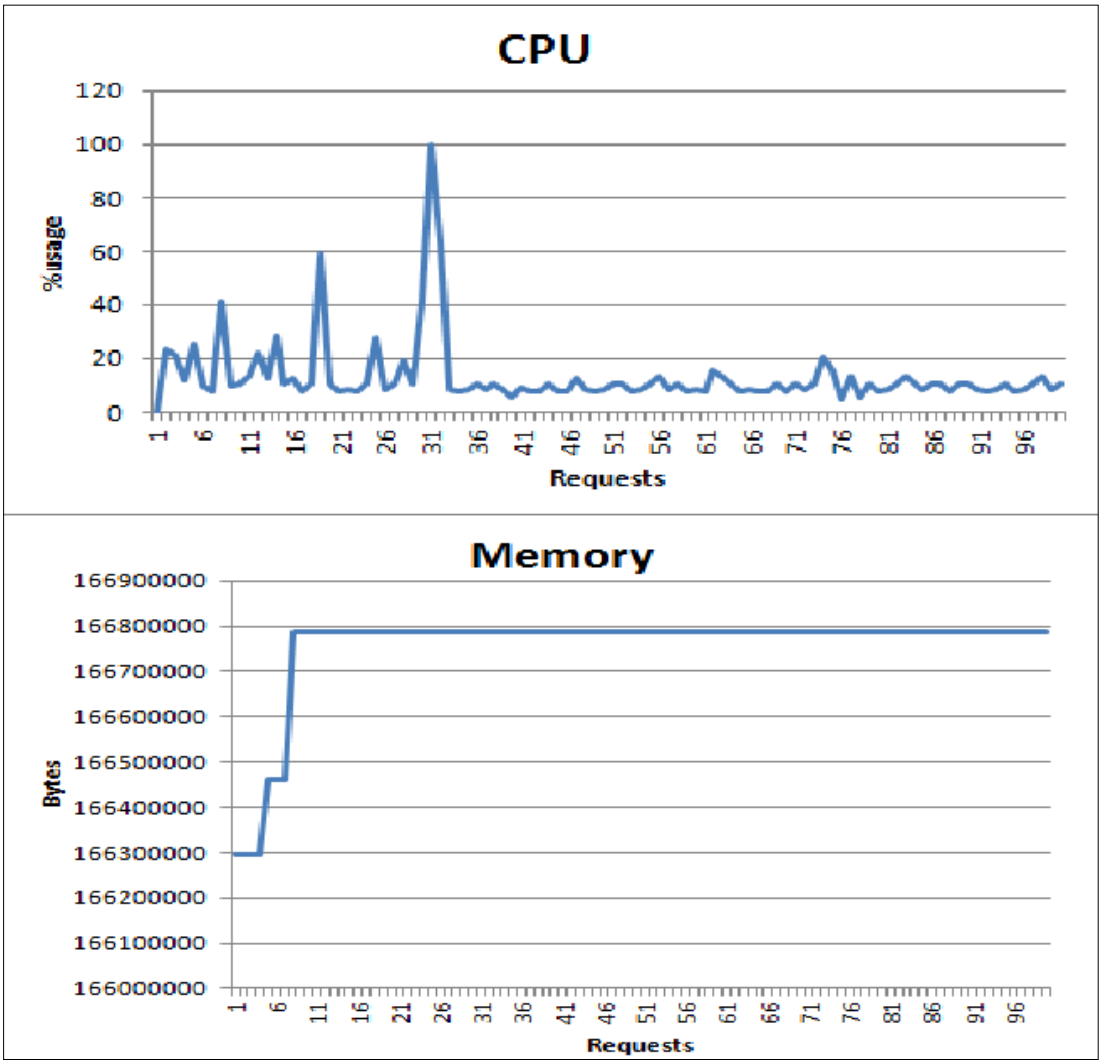

Figure 23 Scenario 5: CPU, Memory on device

CPU's behavior is not stable enough but still it's pretty close to the previous scenario 4. Bytes allocated to the memory remain the same for all requests.

PDP			
CPU(%)	**Memory (bytes)**	**Rx (bytes)**	**Tx (bytes)**
85,6	210198528	6229,02	11251,22

Table 26 Scenario 4: Average CPU, Memory, Rx/Tx on PDP

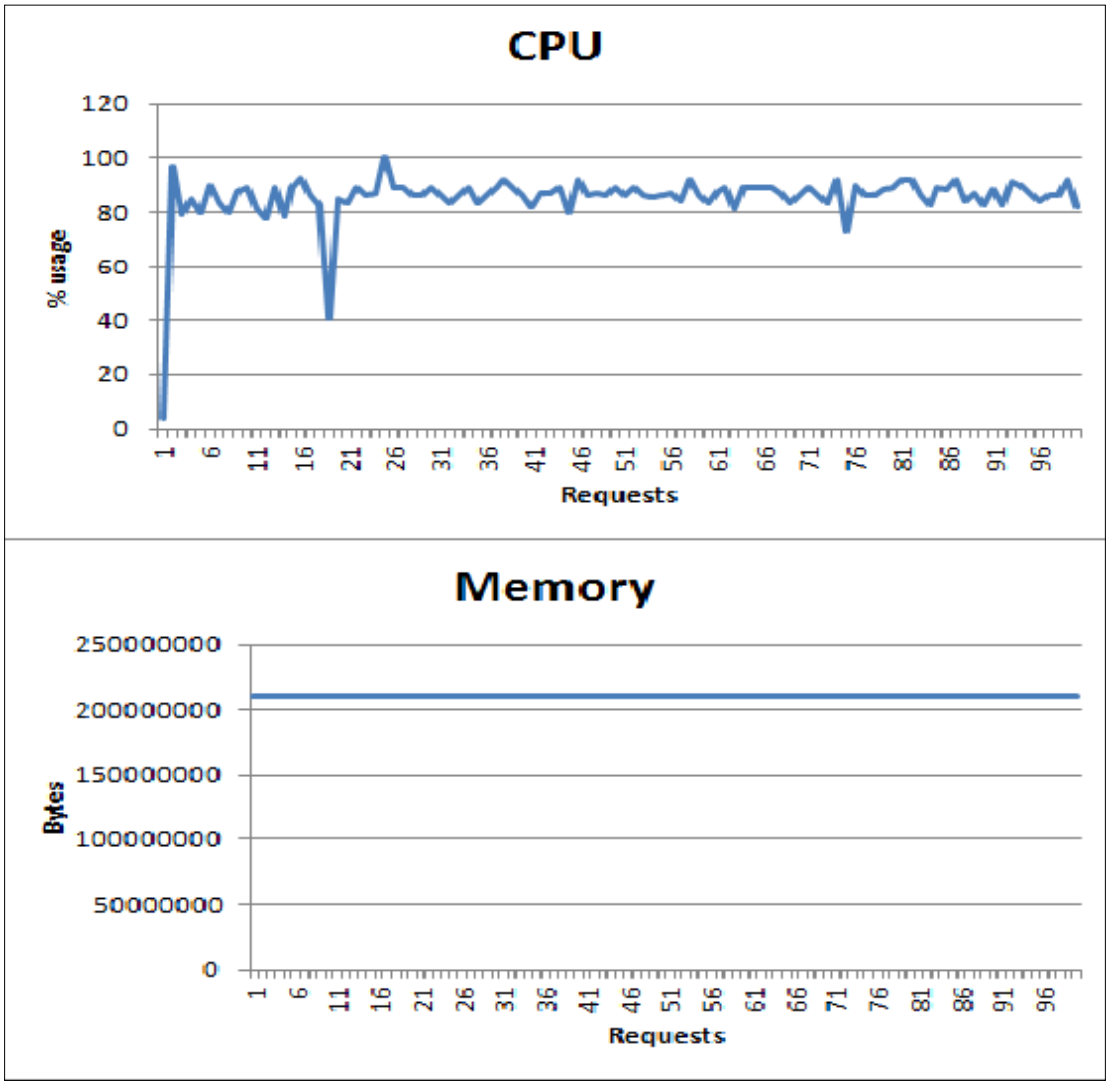

Figure 24 Scenario 5: CPU, Memory on PDP

The CPU load on PDP is high and the bytes that needed are stable.

PIP			
CPU(%)	**Memory (bytes)**	**Rx (bytes)**	**Tx (bytes)**
1,1	12986150912	13297,74	10405,56

Table 27 Scenario 5: Average CPU, Memory, Rx/Tx on PIP

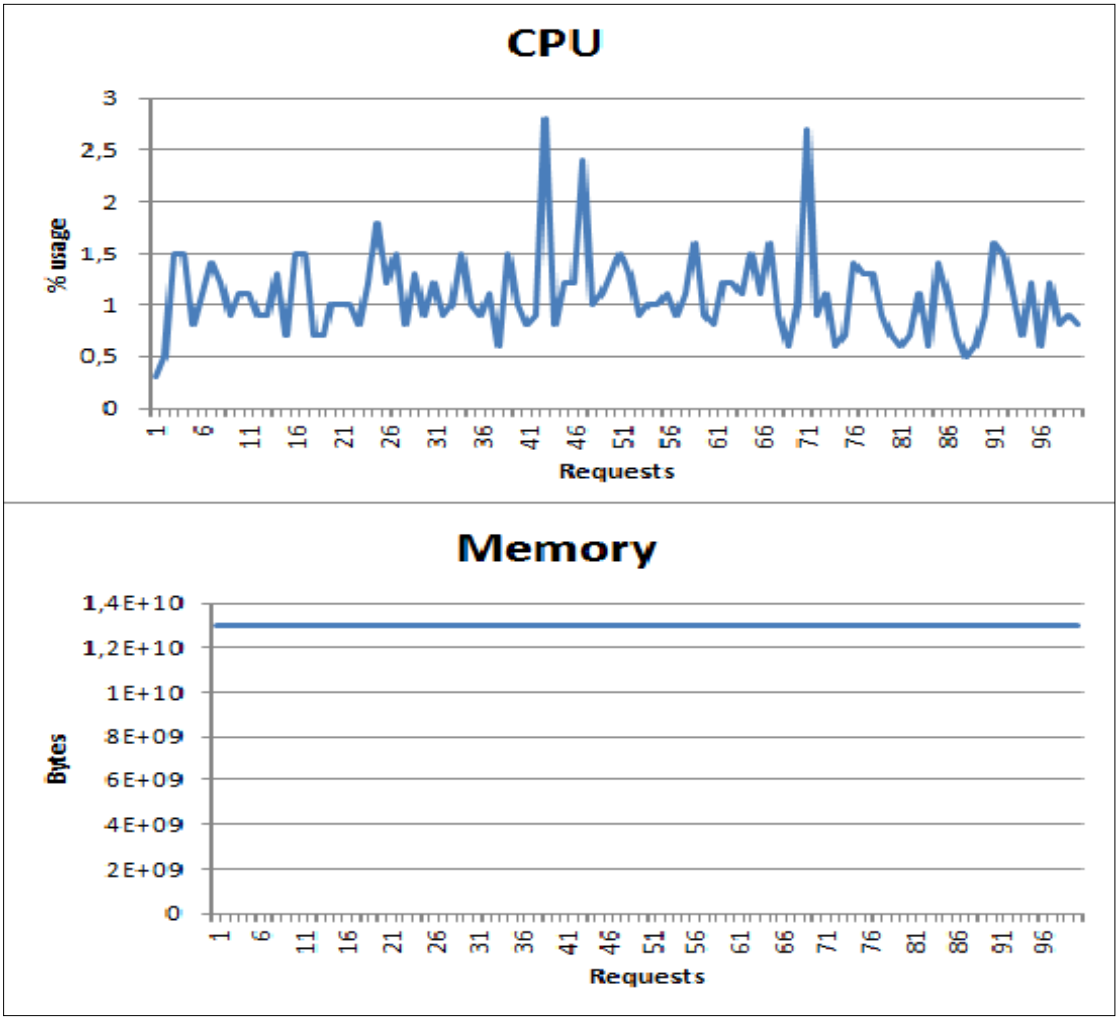

Figure 25 Scenario 5: CPU and Memory on PIP

CPU's load is very low and the reason for that depends on the characteristics of the machine. However memory remains unchanged during the procedure

6.2.6 Scenario 6

User
Average Time (ms)
467,3

Table 28 Scenario 6: Average time delay on user

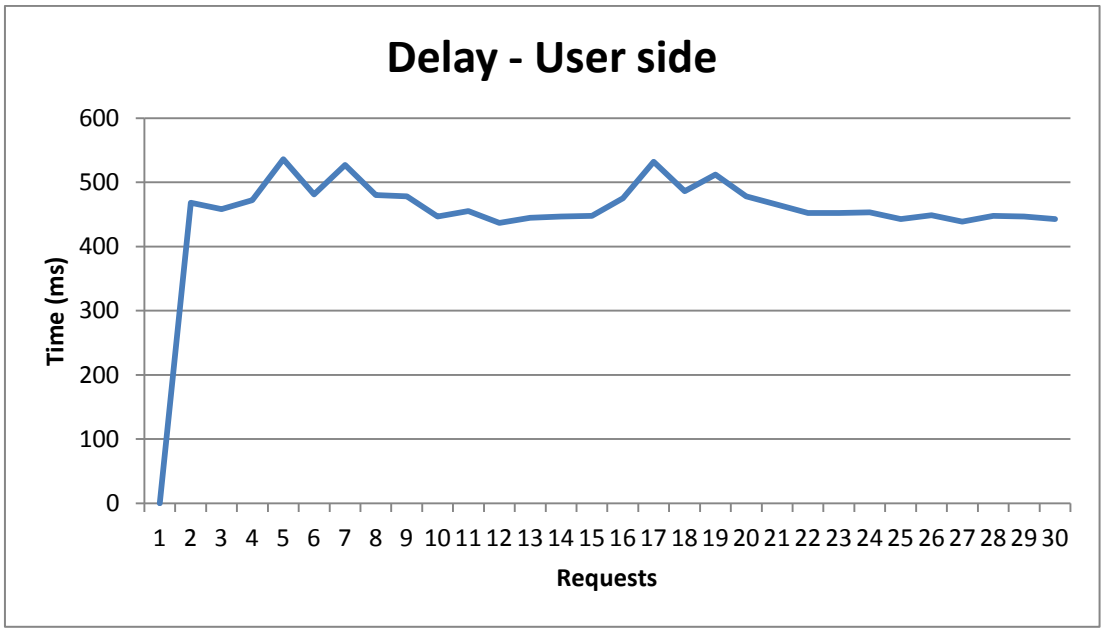

Figure 26 Scenario 6: Delay on user side

The value of the first request is 2449ms. Time remains stable during the communication

Device			
CPU(%)	**Memory (bytes)**	**Rx (bytes)**	**Tx (bytes)**
0,7	167847116,8	12318,7	1299,7

Table 29 Scenario 6: Average CPU, Memory, Rx/Tx on device

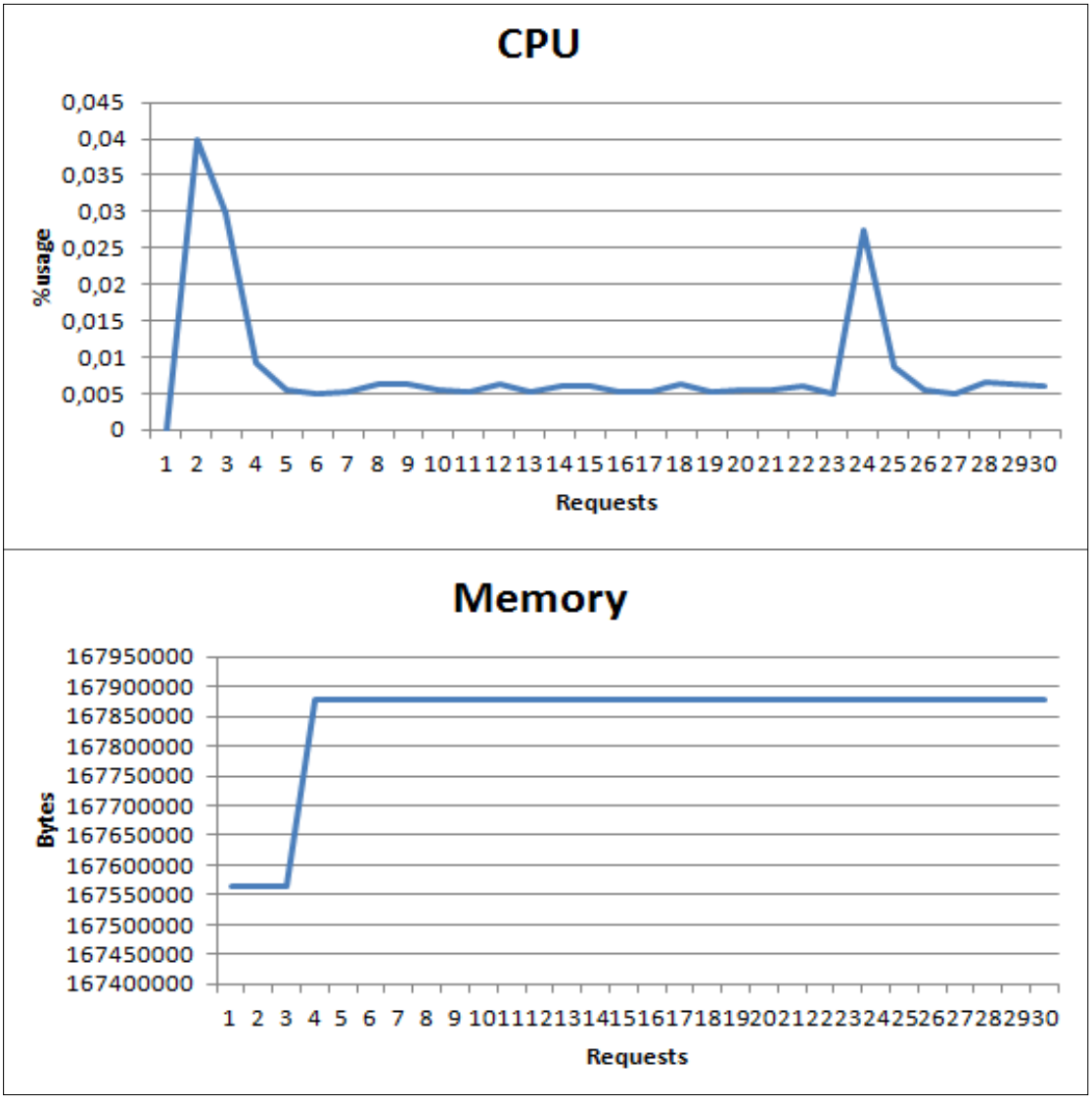

Figure 27 Scenario 6: Average CPU, Memory on device

The act of CPU on device is very low because of the conditions of the scenario. The insert of delay in each request give us a low usage and almost in every request the same usage. After the firsts requests, memory uses the same number of bytes because of caching.

PDP			
CPU(%)	**Memory (bytes)**	**Rx (bytes)**	**Tx (bytes)**
1,7	210333696	9790,1	27934

Table 30 Scenario 6: Average CPU, Memory, Rx/Tx on PDP

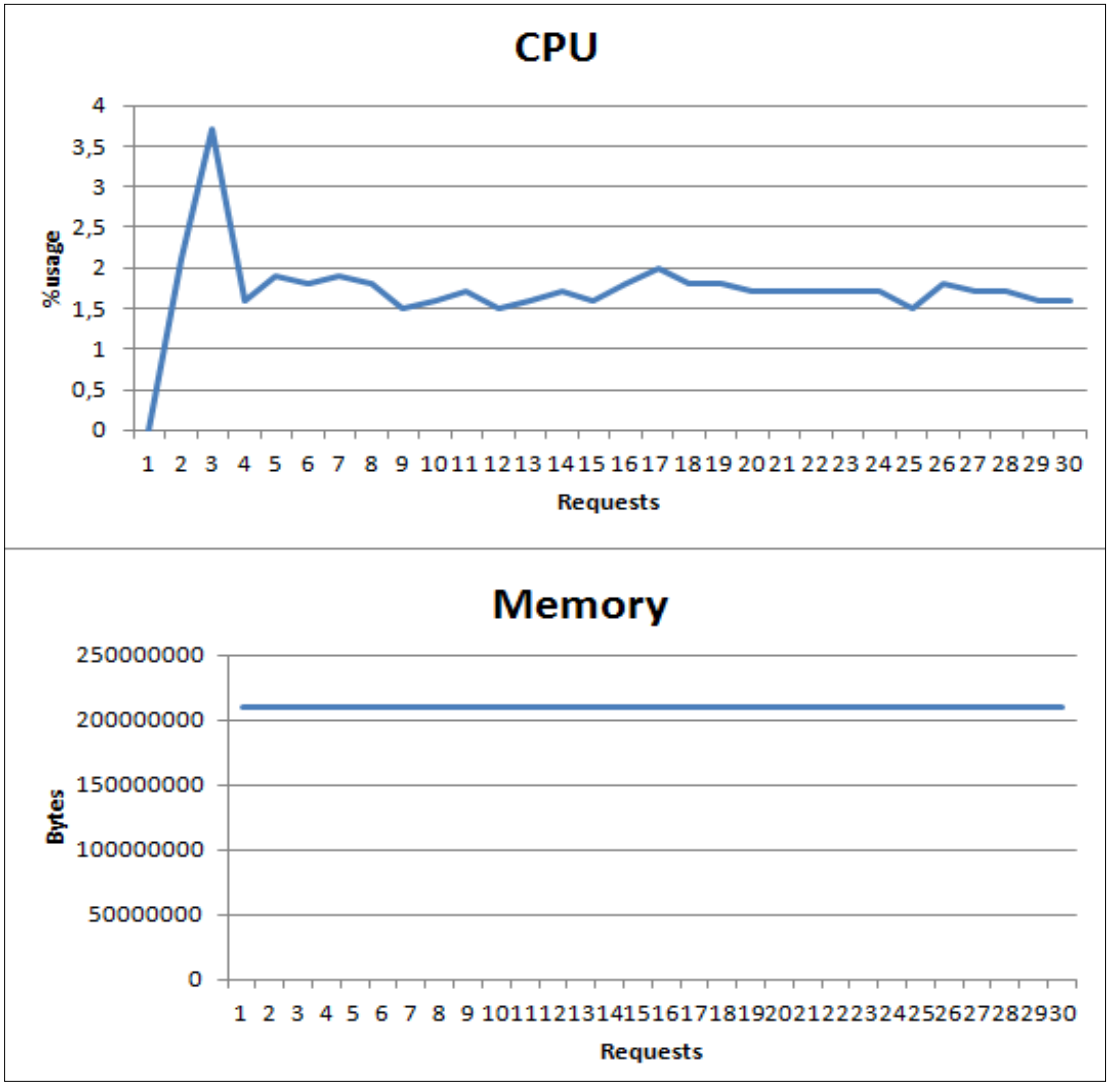

Figure 28 Scenario 6: Average CPU, Memory on PDP

Low CPU usage, but bigger than the usage of CPU on device. The bytes are the same for each request.

PIP			
CPU(%)	Memory (bytes)	Rx (bytes)	Tx (bytes)
1,7	12988178159	2270089	210571

Table 31 Scenario 6: Average CPU, Memory, Rx/Tx on PIP

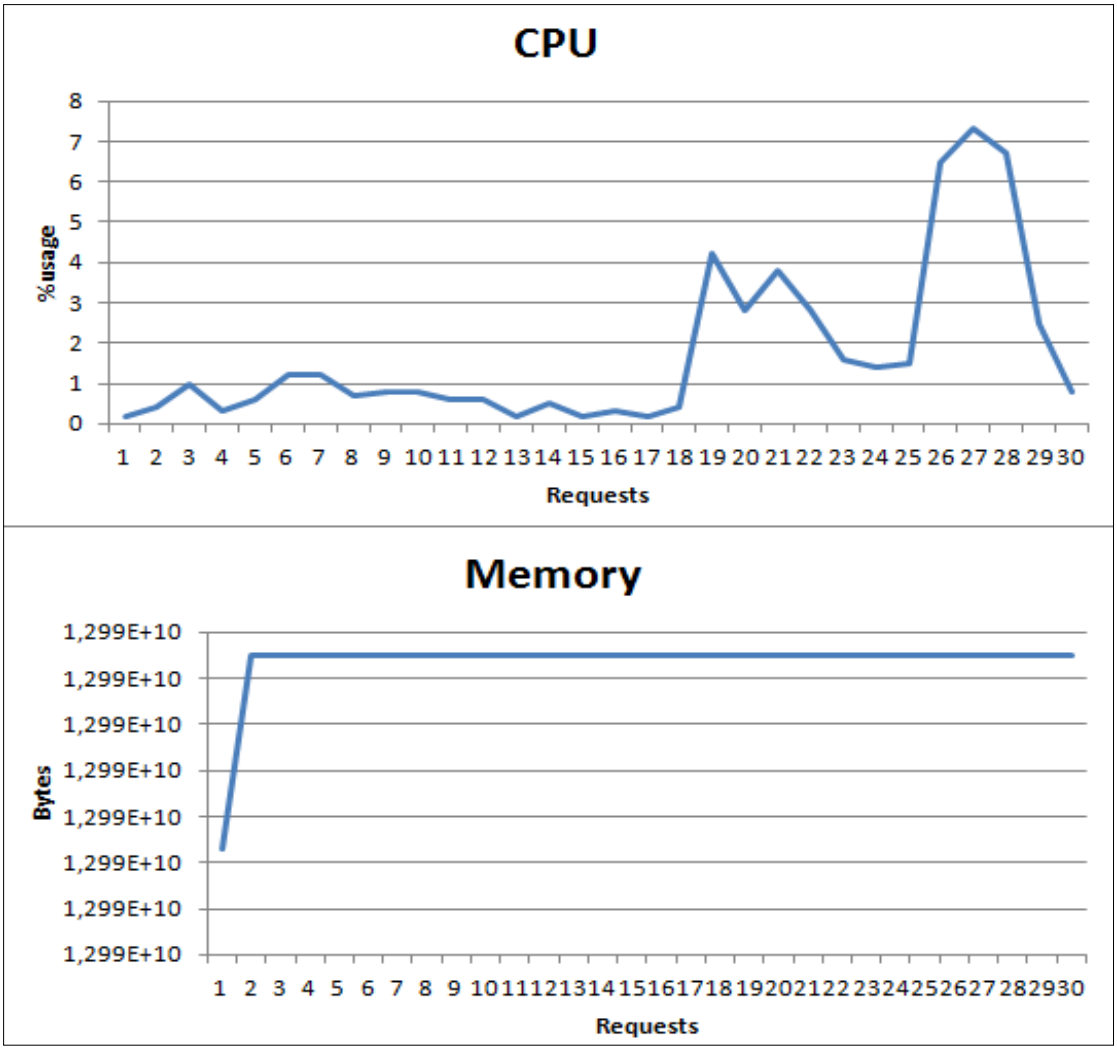

Figure 29 Scenario 6: Average CPU, Memory on PIP

Low CPU usage but with lots of variations due and a stable memory.

6.3 Comparison

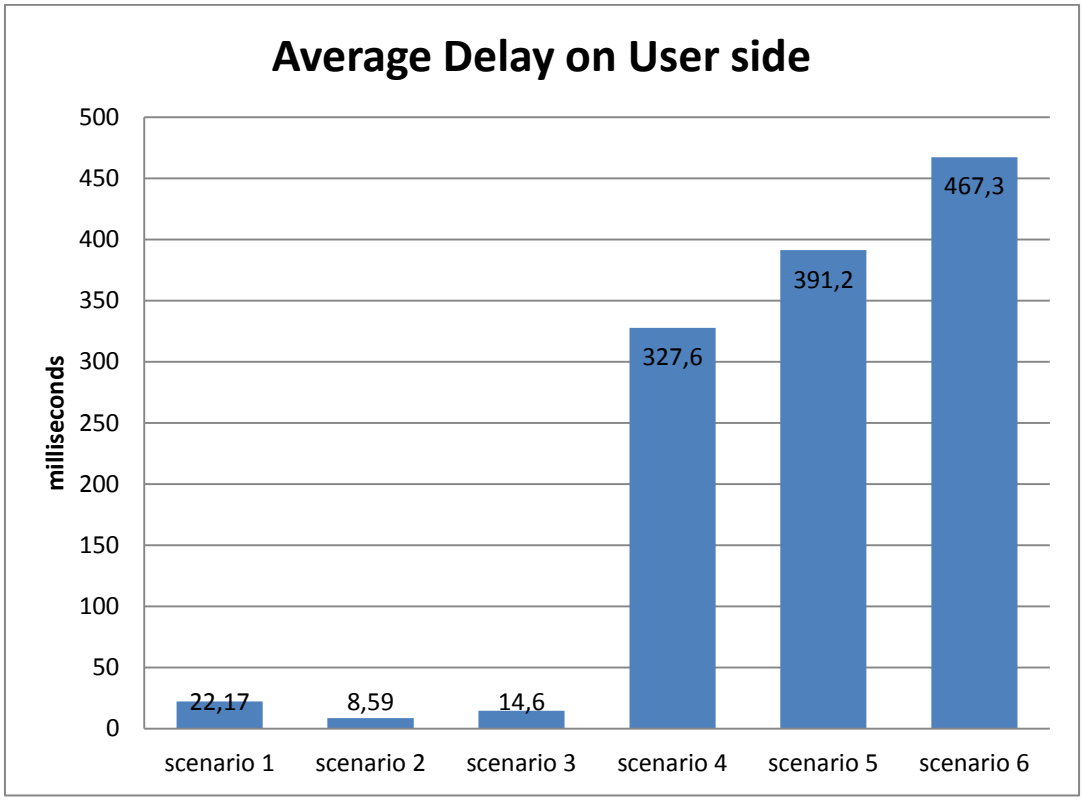

Figure 30 Average Delay

Figure 30 shows the average delay on users side in each scenario. The average delay on the user is higher in scenarios 4,5,6 which have the proposed framework. The lower average time is in scenario 2 which have the 1000 requests. It is about the caching of the memory. In the scenario 4 and 5, despite of the same scenario there is a difference on the value of the delay which is because of the network that each scenario evaluated. The scenario 6 requires more time due to the delay of 30sec between requests.

CPU(%) on Device					
Scenario 1	**Scenario 2**	**Scenario 3**	**Scenario 4**	**Scenario 5**	**Scenario 6**
0,96	98,6	2,16	12,6	13,2	0,7

Table 32 CPU on device

The biggest CPU load is in scenario 2 with the 1000 requests. In scenario 4 and 5 is almost the same and much lower. In scenario 6 is the lower one -nearly 1%- and that is because of the 30 seconds of delay in each request. The above correspondence was predictable. The difference between scenario 2 and the others is about of the plethora of the requests.

CPU on PDP (Mbytes)		
Scenario 4	**Scenario 5**	**Scenario 6**
86,6	85,6	1,7

Table 33 CPU on PDP (scenarios 1, 2 and 3 are not mentioned since there is no PDP)

On the device that is running the PDP, the usage of CPU in scenarios 4 and 5 is almost the same and is about 85% both of them. It is quite a good usage pro rata of the constrained set of resources of the board. Scenario 6 has almost 2% of usage; the difference is big and is because of the big delay in that scenario.

CPU on PIP (Mbytes)		
Scenario 4	**Scenario 5**	**Scenario 6**
1,2	1,1	1,7

Table 34 CPU on PIP (scenarios 1, 2 and 3 are not mentioned since there is no PIP)

The Table 34 shows the usage of CPU on the device that running the PIP. Scenarios 4 and 5 is almost the same. The difference is in scenario 6. That differentiation is due to the delay that puts on that scenario.

Memory on Device (Mbytes)					
Scenario 1	**Scenario 2**	**Scenario 3**	**Scenario 4**	**Scenario 5**	**Scenario 6**
41,46	40,19	43,75	166,69	166,75	167,84

Table 35 Memory on device

The memory on device for the first three scenarios is almost the same and for the other three. We can conclude that the bytes that the memory uses are the same for each scenario.

Memory on PDP (Mbytes)		
Scenario 4	**Scenario 5**	**Scenario 6**
210.19	210.19	210.33

Table 36 Memory on PDP (scenarios 1, 2 and 3 are not mentioned since there is no PDP)

The memory acts in the same way in scenarios 4 and 5, as there are the same scenario. The difference is in scenario 6. The reason is the delay in each request and the memory don't caching because of that delay.

Memory on PIP (Mbytes)		
Scenario 4	**Scenario 5**	**Scenario 6**
12986.15	12986.15	12988.17

Table 37 Memory on PIP (scenarios 1, 2 and 3 are not mentioned since there is no PIP)

As in Table 36 is the same reaction and in Table 37. Scenario 6 is higher than the other two scenarios for same reasons.

7 Conclusion

The purpose of this work was to establish a secure communication in an IoT environment. Because of the constrained set of resources that the IoT devices have, we implemented a lightweight framework using security policies. The security mechanism based on the access control and to specify, on the eXtensible Access Control Markup Language (XACML). The communication of all the entities achieved by using the protocol for instant messaging the XMPP, by using the Smack library. The protocol with the proposed framework was tested in six scenarios. The first three scenarios were without the proposed framework and the other three with it. The concept of all the scenarios was a communication between a user and a device. The result's correspondence was the expected one. As expected, scenarios with the proposed framework, the response time was bigger than the scenarios without the framework. Nevertheless, the results demonstrated that the XMPP protocol is a protocol suitable for IoT environments, and, moreover, from the measurements presented it appears that the communication is quite fast and stable without extreme fluctuations. In conjunction with XACML policies, they provide a reliable, secure and consistent communication to those systems. It is expected in devices with limited resources that the CPU usage is high.

In this work, we have seen how the XMPP protocol reacts in a specific security system for IoT systems. A step further of this work would be a comparison of the proposed framework with the protocols mentioned above and to conclude which is an ideal protocol for those scenarios. Also, because of the everyday usage of mobile phones, the framework could be extended to an application for smartphones which can detect online devices in an IoT system. After that, the user can choose a device, see the features and can make changes, if has the proper certificates for that. Finally, in each transferred message, security could be applied via the encryption of the payload of the XMPP packets, e.g. using symmetric end-to-end encryption, such as the AES algorithm **[**56**]**.

8 References

[1] D. Evans, "The Internet of Things: How the Next Evolution of the Internet Is Changing Everything," April, 2011.

[2] A. Greenberg. (2015) This hacker's tiny device unlocks cars and opens garages. [Online]. http://www.wired.com/2015/08/hackers-tinydevice-unlocks-cars-opens-garages/

[3] A., Luo, B. Humayed, "Cyber-physical security for smart cars: taxonomy of vulnerabilities, threats, and attacks.," *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems.*, pp. 252-253, 2015.

[4] K. Zetter. (2015) Hackers can seize control of electric skateboards and toss riders. [Online]. https://www.wired.com/2015/08/hackers-can-seize-control-of-electric-skateboards-and-toss-riders-boosted-revo/

[5] Zhang J., Wen Q. Hu C., "An identity-based personal location system with protected privacy in IOT," Broadband Network and Multimedia Technology (IC-BNMT), 2011 4th IEEE International Conference on.

[6] Yang Xiao, C. L. Philip Chen Jing Liu, "Authentication and access control in the internet of things," 2012 32nd International Conference on Distributed Computing Systems Workshops.

[7] Hoon-Jae Lee, Sang-Gon Lee 2 Bruce Ndibanje, "Security Analysis and Improvements of Authentication and Access Control in the Internet of Things," Sensors 2014, 14(8), 14786-14805; doi:10.3390/s140814786.

[8] Salvatore Piccione, Denise Rotondi Sergio Gusmeroli, "IoT Access Control Issues: a Capability Based Approach ," Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing 2012.

[9] Yin Lihua Sun Kaiwen, *Attribute-role-based hybrid access control in the internet of things. In: Web Technologies and Applications*., 2014, pp. 333–343.

[10] Challal Yacine Touati Lyes, "Activity-based access control for IoT," Proceedings of the 1st InternationalWorkshop on Experiences 2015.

[11] Challal Yacine Touati Lyes, "Batch-based cp-abe with attribute revocation mechanism for the internet of things," 2015.

[12] Schreiner R. Lang U., "Proximity-Based Access Control (PBAC)," in *ISSE 2015*.: Springer, 2015, pp. 157–170.

[13] Internet of Things at Work: Architecture Approach to Interoperability. www.iot@work.eu.

[14] Joaquim Macedo, M. Joao Nicolau, Alexandre Santos Fabio Goncalves, "Security Architecture for Mobile E-Health," 2013.

[15] Jose Luis Hernandez Ramos , Antonio F. Skarmeta Gomez Jorge Bernal Bernabe, "TACIoT: multidimensional Trust-aware Access Control," 2015.

[16] Thakre Pravin A. , Prasad Neeli R. , Prasad Ramjee Mahalle Parikshit N., "A fuzzy approach to trust based access control in internet of things," 2013.

[17] Samarati Sabrina Capitani de Vimercati Vimercati Pierangela, *Access Control: Policies, Models, and Mechanisms.*, 2001.

[18] http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf.

[19] Openfire Server, http://www.igniterealtime.org/projects/openfire/.

[20] RFC 3920. https://tools.ietf.org/html/rfc3920.

[21] Kevin Smith, Remko Tronçon Peter Saint-Andre, *XMPP: The Definitive Guide*., 2009.

[22] RFC-6120. http://www.ietf.org/rfc/rfc6120.txt.

[23] Saint P. Andre Ed. (2004) Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. https://xmpp.org/rfcs/rfc3921.html.

[24] Christopher Foley and Eamonn de Leastar Leigh Griffin, "A HYBRID ARCHITECTURAL STYLE FOR COMPLEX HEALTHCARE SCENARIOS,".

[25] Johan Gustav Bellika Kristian Andreassen, "The Reliability of the XMPP Protocol Extensions as a File Transfer Mechanism in Dedicated Healthcare Networks,".

[26] Christian Bonnet, Amelie Gyrard, Rui Pedro Ferreira da Costa, Karima Boudaoud Soumya Kanti Datta, "Applying Internet of Things for Personalized Healthcare in Smart Homes,".

[27] Sangjoon Park, Hoon Ko,Hyun-Joo Moon, Jongchan Lee Jongmyung Choi, "Issues for Applying Instant Messaging to Smart Home,".

[28] Thomas Springer, Daniel Schuster, Alexander Schill, Ralf Ackermann, Michael Ameling Sven Bendel, "A Service Infrastructure for the Internet of Things based on XMPP,".

[29] Michael Kirsche Ronny Klauck, "Chatty Things - Making the Internet of Things Readily Usable for the Masses with XMPP,".

[30] Sensinode Hartke K. Shelby Z., "Constrained Application Protocol (CoAP)," Internet Engineering Task Force (IETF) 2012.

[31] Hong S., Ha M., Kim Y-J., Kim D., Jung W., "SSL-Based Lightweight Security of IP-Based Wireless Sensor Networks," Proceedings of the International Conference on Advanced Information Networking and Applications Workshop 2009.

[32] Schimitt C., Wen Hu, Bruning M. Kothmayr T., "A DTLS Based End-To-End Security Architecture for the Internet of Things with Two-Way Authentication," Technische Universitat- Munchen, Germany, 2012.

[33] Kai T. Hillmann, Stefanie Gerdes Olaf Bergmann, "A CoAP-gateway for smart homes," *Computing, Networking and Communications (ICNC)*, pp. 446-450, 2012.

[34] Jong-Tae Park Seung-Man Chun, "Mobile CoAP for IoT mobility management," *Consumer Communications and Networking Conference(CCNC)*, pp. 283-289, 2015.

[35] Carles Gomez, Ilker Demirkol, Matthias Kovatsch August Betzler, "Congestion Control for CoAP Cloud Services," 2014. [Online]. https://pdfs.semanticscholar.org/0390/ad92618d3bc96f456f1b347461731928df7a.pdf

[36] Michele Ruta, Eugenio Di Sciascio Hasan Ali Khattak, "CoAP-based Healthcare Sensors Network: a," 2014.

[37] Nicola Bui, Angelo Castellani, Lorenzo Vangelista, Michele Zorzi Andrea Zanella, "Internet of Things for Smart Cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22-32, Feb 2014.

[38] G. P. Hancke C. P. Kruger, "Implementing the Internet of Things Vision in vision in industrial wireless sensor networks," 2014.

[39] Elmar Zeeb, Steffen Pruter, Frank Golatowski, Dirk Timmermann, Regina Stoll Gruido Moritz, "Devices Profile for Web Services and the REST," July, 2010.

[40] V. Vaithayana, P. Raj, K. Gopalan, R. Amirtharaj V. Venkatesh, "A Smart Train Using the DPWS-based Sensor Integration," 2013.

[41] Antonio Mancina, Gaetano F. Anastasi, Giuseppe Lipari,Leonardo Mangeruca, Roberto Checcozzo, Fulvio Rusina Tommaso Cucinotta, "A Real-Time Service-Oriented Architecture for Industrial Automation," *IEEE Trans. Ind. Informatics*, vol. 5, no. 3, pp. 267–277, August 2009.

[42] S. PöhlsenS. SchlichtingM. SträhleF. FranzC. Werner, "A DPWS-Based Architecture for Medical Device Interoperability," *World Congress on Medical Physics and Biomedical Engineering*, pp. 82-85, 2009.

[43] Konstantinos Fysarakis,Ioannis Papaefstathiou,Charalampos Manifavas, Konstantinos Rantos,Othonas Sultatos , "Policy-based access control for DPWS-enabled ubiquitous devices," in *Proceedings of the 2014 IEEE Emerging Technology*., 2014, pp. 1-8.

[44] Pusik Park, Jongchan Choi, Dugki Min Rustam Rakhimov Igorevich, "iVision based Context-Aware Smart Home system," 2012.

[45] Obermaier D., "Getting Started with MQTT | A Protocol for the Internet of Things,".

[46] Gonçalo Louzada, Andreia Carreiro,António Damasceno Daniel Barata, "System of Acquisition, Transmission, Storage and Visualization of Pulse Oximeter and ECG Data Using Android and MQTT," 2013.

[47] Danilo F. S. Santos, Hyggo O. Almeida Yuri F. Gomes, "Integrating MQTT and ISO/IEEE 11073 for health information sharing in the Internet of Things," 2015.

[48] D. Piromalis, T. Iliopoulou, K. Agavanakis, M.Barbarosou, K. Prekas, K. Antonakoglou P. Papageorgas, "Wireless Sensor Networking Architecture of Polytropon: An Open Source Scalable Platform for the Smart Grid," 2014.

[49] Hoan-Suk Choi, Woo-Seop Rhee Seong-Min Kim, "IoT home gateway for auto-configuration and management of MQTT devices," 2015.

[50] Juan Carlos Cano, Carlos Calafate Jorge E. Luzuriaga, "Handling mobility in IoT applications using the MQTT protocol," 2015.

[51] EU Research Project SOCRADES 2006-2009. http://www.socrades.net.

[52] RFC 7252: The Constrained Application Protocol (CoAP),The Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc07252.

[53] (2012, June) RFC 4422 - Simple Authentication and Security Layer (SASL). http://www.ietf.org/rfc/rfc4422.txt.

[54] SSL Guide. http://download.igniterealtime.org/openfire/docs/latest/documentation/ssl-guide.html.

[55] Smack library. https://www.igniterealtime.org/projects/smack/.

[56] [Online]. https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf

[57] https://xmpp.org/.

[58] http://www.ietf.org/rfc/rfc3920.txt.