Wireless Solutions and Authentication Mechanisms for Contiki Based Internet of Things Networks

Samet KALYONCU

Master’s Thesis, 15 credits

Halmstad 2013-09-30
Wireless Solutions and Authentication Mechanisms for Contiki Based Internet of Things Networks

Master's Thesis in Computer Network Engineering
September 2013

Author: Samet Kalyoncu
Supervisor: Urban Bilstrup
Examiner: Tony Larsson

School of Information Science, Computer and Electrical Engineering
Halmstad University
PO Box 823, SE-301 18 HALMSTAD, Sweden
Preface
I would like to express my deep gratitude to the ones who helped me and supported me during this thesis. First and foremost, I would like to thank to Professor Tony Ingemar Larsson for giving me the opportunity to work on this thesis and providing the necessary equipment. Furthermore, I want to give my special thanks to my supervisor Urban Bilstrup for his valuable guidance and advices. To Suleyman Savas, PhD student and a good friend, for keeping me on the track when I got lost in c programming. To Mariano Alvira, from Redwire LLC, for patiently answering all my questions regarding the evaluation kit. To all the anonymous developers and friends, from the contiki developers mail group, for their tips and tricks.

Last but not least, I would like to give my most sincere thanks to my family and my fiancée in Turkey for all the sacrifices they made for me and for their endless support.
Abstract

Internet of Things, is a new expression described as the future of the internet, promises a new world surrounded by tiny smart objects interacting with the environment, communicating with each other, and controlled over internet. Investigating which low power wireless solution and authentication mechanism fits best for IoT networks, and applying these technologies on simulator and real hardware is the main task of this project.

Bluetooth Low Energy, ANT, 6LoWPAN and ZigBee are investigated low power wireless technologies which might be used to create an IoT network. Yet, BLTE and ANT have narrower application areas compared to the others, therefore ZigBee and 6LoWPAN technologies are investigated in depth and compared as the 2 promising solutions for implementation and integration of Internet of things concept. SPINS, TinySec, TinyECC, SenSec, MiniSec, ContikiSec and AES CCM are the main security frameworks especially designed for wireless sensor networks providing confidentiality, authentication and integrity. These frameworks were described and compared to find out most suitable authentication mechanism for IoT networks. Contiki OS is used as the operating system of nodes during the implementation of network both on simulator Cooja and real hardware.

ZigBee and 6LoWPAN were compared considering interoperability, packet overhead, security and availability. As a result 6LoWPAN came forward due to providing high interoperability and slightly less packet overhead features. ZigBee devices require extra hardware to operate with different technologies. Among the discussed security frameworks, ContikiSec and AES CCM were highlighted because of flexibility, providing different levels of security. Resource limited characteristic and diversity of IoT applications make flexibility a very useful feature while implementing a security framework. Experiments committed to implement a working IoT network were not hundred percent successful. 6LoWPAN was successfully implemented but implementation of the security framework was failed due to compatibility issues between the sensor and the router node. Firmware of the sensor node is not designed to provide any kind of security, therefore security features of the router node is also disabled.
# Table of Contents

Preface .................................................................................................................... iii

Abstract .................................................................................................................. v

List of Figures ......................................................................................................... ix

List of Tables .......................................................................................................... ix

1. INTRODUCTION ........................................................................................................ 1

   1.1 Motivation ............................................................................................................ 1

   1.2 Aim and Research Question ............................................................................. 1

   1.3 Thesis Structure ................................................................................................. 2

2. INTERNET OF THINGS VISION .................................................................................. 3

   2.1 What is Internet of Things? .............................................................................. 3

   2.2 Technology ........................................................................................................ 4

   2.3 Architecture ...................................................................................................... 4

   2.4 Applications ...................................................................................................... 5

   2.5 Security Challenges ......................................................................................... 7

3. AUTHENTICATION in IoT NETWORKS .................................................................. 10

   3.1 Definition .......................................................................................................... 10

   3.2 Functional Principle ........................................................................................ 10

   3.3 Authentication Mechanism Examples .......................................................... 11

4. CONTIKI OS .............................................................................................................. 14

   4.1 System Architecture [32, 33] ........................................................................ 14

   4.2 Contiki Features ............................................................................................... 16

      4.2.1 Memory Allocation and Management ....................................................... 16

      4.2.2 Full IP Networking ................................................................................. 16

      4.2.3 Power Awareness .................................................................................... 16

      4.2.4 6lowpan, RPL, CoAP ............................................................................ 16

      4.2.5 Cooja Network Simulator .................................................................... 19

      4.2.6 Sleepy Routers ....................................................................................... 19

      4.2.7 Protothreads ............................................................................................. 19

      4.2.8 Coffee Flash File System ................................................................. 19

      4.2.9 Contiki Shell ............................................................................................ 20

      4.2.10 Rime Stack ............................................................................................ 20
5. WIRELESS PROTOCOLS FOR LOW POWER DEVICES ................................................................. 21
  5.1 Bluetooth Low Energy ........................................................................................................ 21
  5.2 ANT ................................................................................................................................... 22
  5.3 ZigBee ................................................................................................................................ 22
  5.4 6LowPAN .......................................................................................................................... 25

6. METHODOLOGY ..................................................................................................................... 28
  6.1 Hardware Selection ............................................................................................................ 28
  6.2 Software Simulation ........................................................................................................... 33
  6.3 Hardware Implementation ................................................................................................. 35
    6.3.1 Configuring Red-io ................................................................................................... 35
    6.3.2 Configuring TH12 ................................................................................................... 38
    6.3.3 Debugging ................................................................................................................ 39

7. DISCUSSION & CONCLUSION ............................................................................................ 41

8. REFERENCES ......................................................................................................................... 44
List of Figures

FIGURE 1 A NEW DIMENSION (SOURCE: INTERNATIONAL TELECOMMUNICATION UNION [45]) .................................................. 3
FIGURE 2 IOT ARCHITECTURE (SOURCE: DATANG TELECOM TECHNOLOGY & INDUSTRY GROUP [46]) ........................................ 5
FIGURE 3 CATEGORIZATION OF INTERNET OF THINGS ........................................................................................................ 8
FIGURE 4 CONTIKISEC PACKET FORMATS (SOURCE: [30]) .................................................................................................................. 12
FIGURE 5 PARTITIONING INTO CORE AND LOADED PROGRAMS (SOURCE: [32]) ................................................................. 14
FIGURE 6 CONTIKI OS ARCHITECTURE (SOURCE: [33]) ..................................................................................................................... 15
FIGURE 7 RPL NETWORK TOPOLOGY ........................................................................................................................................ 17
FIGURE 8 COAP LAYERS (SOURCE: [36]) ..................................................................................................................................... 18
FIGURE 9 COOJA SCREENSHOT ............................................................................................................................................... 19
FIGURE 10 ZIGBEE NETWORK TOPOLOGY (SOURCE: [47]) ....................................................................................................... 23
FIGURE 11 ZIGBEE STACK ARCHITECTURE (SOURCE: [38]) ..................................................................................................... 23
FIGURE 12 ZIGBEE FRAME FORMATS ....................................................................................................................................... 24
FIGURE 13 6LOWPAN PROTOCOL STACK MODEL .......................................................................................................................... 25
FIGURE 14 6LOWPAN FRAME FORMATS (SOURCE: [36]) .................................................................................................................... 26
FIGURE 15 FRAGMENTATION EXAMPLE (SOURCE: [36]) .............................................................................................................. 26
FIGURE 16 THINGSQUARE COMPONENTS (SOURCE: [43]) ......................................................................................................... 29
FIGURE 17 THINGSQUARE NODE ACCESSING INTERNET (SOURCE: [43]) ............................................................................... 30
FIGURE 18 TH12 TEMPERATURE AND HUMIDITY SENSOR (SOURCE: [44]) ............................................................................... 31
FIGURE 19 RED-io ROUTER (SOURCE: [44]) ............................................................................................................................... 32
FIGURE 20 PROG12 MODULE (SOURCE: [44]) ............................................................................................................................. 32
FIGURE 21 CREATE NEW SIMULATION ...................................................................................................................................... 33
FIGURE 22 MY SIMULATION ............................................................................................................................................................ 34
FIGURE 23 CREATE MOTE TYPE .................................................................................................................................................. 34
FIGURE 24 SIMULATION OF A IPv6 MESH NETWORK .................................................................................................................. 35

List of Tables

TABLE 1 FIVE COMPONENTS OF AUTHENTICATION SYSTEM (SOURCE: [24]) .................................................................................. 11
TABLE 2 BTLE TECHNICAL DETAILS .............................................................................................................................................. 21
TABLE 3 ANT TECHNICAL SPECIFICATIONS ................................................................................................................................. 22
TABLE 4 COMPARISON FOR BLE, ANT, ZIGBEE AND 6LOWPAN ........................................................................................................ 27
TABLE 5 THINGSQUARE EVALUATION KITS .................................................................................................................................. 28
TABLE 6 TECHNICAL SPECIFICATIONS ....................................................................................................................................... 29
TABLE 7 TH12 TECHNICAL SPECS ................................................................................................................................................... 31
TABLE 8 RED-io LOG SCREEN ........................................................................................................................................................... 39
TABLE 9 TH12 LOG SCREEN ............................................................................................................................................................ 40
1. INTRODUCTION

1.1 Motivation
A new concept associated with the “Future Internet” is called “Internet of Things (IoT)" describes a vision where real objects become part of the internet: where each object is uniquely identified, and accessible to the network [1]. The internet of things; is a technological revolution that represents the future of computing and communications, depends on dynamic technical innovation in a number of important fields such as wireless sensors and nanotechnology [2]. Its application areas include a wide variety of industries: electric power, transportation, industrial control, retail, public services management, health, oil and etc. [3]. Vehicle anti-theft, radiation monitoring, security monitoring, automatic vending, city sound monitoring, machine maintenance and public transportation management are instance functions which can be achieved with IoT.

To create a real life scenario, one can imagine a smart home; a house full of smart objects: smart TV, smart phone, lights, heating system and etc. All these objects are connected to each other creating an IoT network and connected to the internet with a special gateway device. Owner is able to control these devices with his smart phone or laptop from his job or anywhere he can connect to the internet. For a more specific situation: the owner of the house checks the temperature sensor values and decides to turn the heating system of the house on just before he left his job so that he can have a "warm" welcome when he reach the house. Since variety of smart objects are on the market, working with Bluetooth, ZigBee, ANT or 6LoWPAN, the most important component of this scenario is to build the IoT network inside the house; while the main challenge is to avoid the unauthorized people to get control of your house.

1.2 Aim and Research Question
The aim of this thesis is to build up a working IoT network with authentication feature enabled: different sensor devices operating and communicating with each other as well as the internet. The main questions need to be answered are:

- Which wireless solution suits best to create an IoT network considering interoperability, packet overhead, security and availability?
- Which authentication mechanism fits most to the needs of an IoT network considering resource limited behavior?

One who reads this report can also find answers to the following questions:

- What is Internet of Things?
- What are the main features of Contiki OS, Cooja and Thingsquare?
1.3 Thesis Structure
This project report consists of 7 main chapters in addition to this introduction. Chapter 2 explains the Internet of Things concept, describing its technology, architecture and application areas. In Chapter 3 the definition of Authentication for IoT networks is given as well as the functional principle and examples. Chapter 4 provides an overview to the Contiki Operating System, clarifying its system architecture and main features. Chapter 5 is a general introduction for low-power wireless protocols which highlights ZigBee and 6LoWPAN. Chapter 6 is the Methodology chapter where all the methods used in this project is listed and the process to implement the working IoT network is described. Chapter 7 discusses the advantages and disadvantages of low power wireless protocols and authentication mechanisms by making comparisons. And finally it concludes the topic, evaluates the hardware and concepts that are used, and gives suggestions for further improvements and future work.
2. INTERNET OF THINGS VISION

2.1 What is Internet of Things?
The term IoT is first mentioned in “The Road Ahead” by Bill Gates in 1995, and later in 1999 the Auto-ID Labs of Massachusetts Institute of Technology introduced the concept of IoT [5, 6]. There is no standard definition for IoT; but considering functionality: “Things have virtual identities operating in smart spaces using intelligent interfaces to connect and communicate with each other”, considering the seamless integration: “Interconnected objects having an active role in what might be called Future Internet” [7].

International Telecom Union (ITU) published an annual report of IoT in 2005 and extended the concept posing the foreground: “Any Time, Any Place and Any Thing Connection” [6]. Figure 1 shows this new dimension of connectivity.

With this new dimension IoT vision promises a world, where anything can connect to the internet from anywhere and at anytime. One might be curious what real changes will this new concept can make in people's daily lives. The word "smart" is getting much and much popular nowadays, especially in electronics: smart TV, smart phone, smart board and etc. IoT will increase the number of smart devices around changing daily life habits for most of us. A real life scenario previously mentioned in the introduction section and can be extended for a clearer view of the changes IoT will trigger: the mentioned smart home has a smart coffee machine in the kitchen connected to the internet and can be...
controlled with an application in owner's mobile phone. Before he goes to sleep, he sets the time from his bed when he wants coffee to be ready next morning just like setting up an alarm clock to wake him up. This small example can seem not much of a change or some might say it is unnecessary. But when the integration of IoT concept is finished, each person will find a necessary change in it.

2.2 Technology

To achieve a successful integration, IoT must be supported from some innovational technologies.

Radio frequency identification (RFID) is one of the key enablers of IoT. Connecting and communicating with small objects require a uniquely identified system and real time tracking, which RFID can provide by getting information about the location and status of objects.

Sensor technology is the second major actor in IoT, playing the role to bridge the gap between the physical world and information world, collecting data from the environment, generating information, raising awareness of the context [7].

The IoT can be divided into three processes [8]:

- **a) Signing and Identification:** RFID, a non-connecting Auto-ID technology, uses radio frequency to identify the target object and gather relevant information (such as identity, status and location). RFID consists of three components: tag, reader and antenna. Every single tag has an electronic code, labeling the target object for identification. Reader; can be in 2 forms: handset or fixed, is charged to read the tag’s information. And antenna sends the radio frequency signal between the tag and the reader [9].

- **b) Information Transmission:** Information transmission is divided into two parts: wired transmission and wireless transmission. The transmission of identification data between tag and reader, and the transmission between the reader and internet backbone network are held wirelessly. Current communication network including internet is the backbone of IoT, the transmission between the information processing center and the external communication devices is typically wired transmission.

- **c) Backstage Intelligent Processing:** A center responsible for pooling, converting and analyzing the collected data, adapting the information according to the specific needs of users.

2.3 Architecture

IoT has 3-layer architecture in most of the references scanned during this project, consists of perception layer (sensing layer), network layer and application layer, as shown in figure 2 [2, 8].
A. The Perception layer: is the sensing component of IoT, includes 2-D bar code labels and readers, RFID tags and reader-writers, camera, GPS, sensors, terminals and sensor network. Main task is to identify, measure, control and gather information from the real world.

B. The Network layer: is the brain of IoT with a function of transmitting and processing data, includes convergence network of communication, network management center and intelligent processing center.

C. The Application layer: is a combination of IoT’s social division and industry demand, provides user interface and intelligent application services according to different needs, enabling intelligent control to the items and building an intelligent perceptional world.

2.4 Applications

IoT has a great potential for social, environmental and economic impact; precise information about the status, location and identity of things lead us to smarter decisions and smarter actions. IoT concepts have been used in a variety of domains: logistics, transport, asset tracking, smart environments, energy, defense, agriculture and etc. [1]

Different researchers made different classifications for IoT applications, two classification examples are given below for better understanding.
According to [Fleisch, 2010], IoT is relevant in every step in every value chain [10]. He introduced seven main value drivers, the first four based on machine to machine communication and the rest based on integration of users:

- **Simplified manual proximity trigger**: Things can exchange their identity information when they moved into each other’s sensing range. Once the identity is known, a specific action or transaction can be triggered.
- **Automatic proximity trigger**: An action is triggered automatically when things reach one another’s sensing range. Identities are pre-known.
- **Automatic sensor triggering**: A smart thing can collect data via any type of sensor (temperature, humidity, acceleration, orientation, vibration, etc.). It senses its condition and environment, communicates the information, enables prompt decision making.
- **Automatic product security**: A thing can provide security based on the interaction between the thing and its representation. For instance, a QR-code containing specific URL pointing specific information.
- **Simple and direct user feedback**: Things can have simple mechanisms to provide feedback for humans, often in the form of audio (beep) or visual (flashing light) signals.
- **Extensive user feedback**: Things (linked to a service via gateway device such as a smart phone) can provide rich services to a human. Detailed product information is a good example of extensive user feedback.
- **Mind changing feedback**: The combination of real world and cyberspace might cause a new level of change in people’s daily behavior. A possible example can be changing the driving behavior while sensors in the vehicles communicate with each other or a base station.

[Chui, Löffler and Roberts, 2010] provide another classification of IoT applications [11]:

- **Information and Analysis**
- **Automation and Control**

In Information and analysis category, decision making services are improved by getting relevant updates from the nodes in the environment, allowing a more accurate analysis of the status to be done. This category applies to tracking (e.g. products in logistics value chain), situational awareness (e.g. smart buildings, environmental condition sensors) and sensor-driven decision analytics (e.g. user shopping patterns).

Automation and control category works over processed and analyzed data. Process optimization in a chemical facility would be a good example; sensors measure the composition of a chemical compound, send it to central service, according to analyzed data actuators will be adjusted to fine tune the composition.

Some global companies, large organizations and even governments have seen the potential of IoT; the list of ongoing large scale activities of IoT is given below:
• IBM’s ‘Smarter Planet’ initiative aims to add intelligence to systems and processes that interface with the world [12].
• Microsoft’s ‘Eye on Earth’ creates a platform where water and air quality of large number of European countries can be viewed [13].
• HP’s ‘Central Nervous System for the Earth’ aims to populate the planet with billions of small sensors [14].
• Cluster of European Research Projects on the Internet of Things (CERP-IoT) compromises of many research activities focused on ‘internet connected and inter-connected world of objects’ [15].
• In South Africa, ‘the Internet of Things Engineering Group’ at the CSIR Meraka Institute is focused on creating a framework allowing for channel agnostic communication between things and applications in IoT [16].

2.5 Security Challenges
In IT terminology ‘security’ term refers to the provision of confidentiality, authentication, integrity, authorization, non-repudiation, and availability. But in IoT context the emphasis is not only on these required security services, but also how these services are realized and executed in the overall system [17].

It is difficult to see the resource limited behavior of IoT devices in the real life scenario mentioned earlier, the owner can replace the battery of any sensor or device at home when it went off. But for larger scale IoT networks such as fire warning system for a forest with thousands of tiny temperature sensors, it is a great challenge to replace a battery, and for that reason the weakest point of IoT architecture for an intruder is certainly the power management framework. Previously, attackers had to limit the number of targets that they could simultaneously force offline, or find an exploit that would cause a specific service to crash. But when the device runs off battery, attackers don’t need to do anything particularly technical, and all services of the device forced offline [18].

A variety of security classifications have been made for IoT, however, a single categorization including the whole IoT concept is represented in detail. [Mayer, 2009] categorized IoT into eight titles as shown in Figure 3 [19].
**Communication:** There are different communication protocols to enable secure information exchange between the devices and provide integrity, authenticity, and confidentiality (e.g. TLS, IPSec). But problem arises during the application of these protocols. Especially, in tiny devices like sensors which have limited processing power, it is very difficult to apply these protocols. Botnets and DDoS are the specific attacks that target the availability of communication.

**Sensors:** The integrity and authenticity of sensor data is a popular topic for current researchers which can be handled in the form of watermarking [20]. Since an attacker can put its own sensor physically near and sense the same area, confidentiality of sensor data relies on communication confidentiality. Privacy is the main problem in Sensors, usually caused by people that are unaware of being sensed. (Langheinrich 2001) gives 3 guidelines to handle this problem: “(1) users must be aware that they are being sensed (‘notice’), (2) users must be able to choose whether they are being sensed and be able to opt-out (‘choice and consent’), and (3) users must be able to remain anonymous (‘anonymity and pseudonymity’)” [21].

**Actuators:** It is assured that an attacker cannot control an actuator, therefore security necessity is low. Privacy and availability of an actuator is critical depending on the kind of actuator and the scenario.

**Storage:** There are entrenched security mechanisms in storage; however, their employment can be very complicated. Privacy, on the other hand, is another major consideration. To ensure that stored data doesn’t contain sensitive private information, anonymization and pseudonymization mechanisms must be established. Final consideration about storage security is mentioned by [Schneier 2008]: “Often, too much information is stored that is not necessary for the actual system” [22].

**Devices:** In devices context, integrity refers that device doesn’t contain any malware which is also called “admissibility” by [Schneier 2006] [23]. Trusted Platform Computing (TPM) aiming to protect the integrity of devices, building TPM modules for devices, yet fully TPM-capable operating systems are missing.
**Processing:** The keynote of processing security is the design and implementation of processing algorithms. Assuring sensitive information available in processing data is not being sent to non-trusted devices or storage, and defining a standard set of privacy preserving mechanisms are major goals of processing security.

**Localization and Tracking:** Integrity, authenticity and confidentiality of localization and tracking is based on communication security. In privacy and availability context; ensuring that there is no way an attacker can reveal the identity of the person or the object in localization data, and providing robust reference signals for localization which cannot be manipulated by an attacker are the security requirements for a secure localization and tracking mechanism.

**Identification:** The security sensitivity of identification is similar to localization and tracking with a small exception, higher prominence on integrity. Due to the used technology (e.g. RFID or biometry for identification and, GSM or GPS for localization and tracking), it is easier for an attacker to manipulate the identification process.
3. AUTHENTICATION in IoT NETWORKS

Authentication is one of the most important parts of a network security scheme, achieving the goal to ensure that the identity of a user or service is correct. A variety of authentication mechanisms and protocols exist, and yet all serve the same purpose [24].

In the following a generic definition for authentication is given as well as the functional principle and finally common authentication mechanisms for IoT networks is listed.

3.1 Definition

Authentication is the mechanism to prove if someone or something is, indeed, who or what it claims to be. [Bishop, 2003] defined as: “Authentication is the binding of identity to a subject.” [25]

The IoT networks in most cases require an authenticated user in order to log onto the network or access the services. The main approaches to user authentication fall into three main types [Menkus, 1988]:

- Knowledge-based: e.g. password, PIN (Personal Identification Number) – referred to as “what the user has”. It is based on private information supplied by the user.
- Possession-based: e.g. memory card and smart card tokens – referred to as “what the user is”. It is based on private objects that the user possesses.
- Biometric-based: e.g. fingerprint, iris scan, signature dynamics – referred to as “what the user knows”. It is based on anatomical, physiological, or behavioral characteristics. [26]

Each of these approaches carries some advantages and disadvantages, thus it is a hard decision to make which approach fits your network’s needs, considering security, ease of use and ease of administration. Combining more than one type of authentication in one network is also a common way, in order to be considered as strong authentication; at least two types of authentication must be combined in one network scheme.

3.2 Functional Principle

The formal description provided by [Bishop, 2003] defines 5 components of authentication system as listed in Table 1 [24]. There are 2 information sets; A and C, to hold the authentication and complementary information. 3 function sets: S,F,L; S function is the selection function, responsible for generation and alteration of information sets. F is the complementary function, obtains the complementary information corresponds to authentication information. L function is the one outputs true or false according to the comparison between complementary information and authentication information.
Component | Designator | Description
--- | --- | ---
**Authentication Information** | A | Set of specific information with which entities prove their identities

**Complementary Information** | C | Set of complementary information that is used by the system to validate authentication information.

**Complementary Functions** | F | Set of complementary functions $f:A \rightarrow C$
Generate appropriate $c \in C$ given $a \in A$

**Authentication Functions** | L | Set of authentication functions to verify identity $l \in l:A \times C \rightarrow \{\text{true, false}\}$

**Selection Functions** | S | Set of selection functions that enable an entity to generate/alter $A$ and $C$
$s \in S$
$s:\ ? \rightarrow A$

Table 1 Five components of authentication system (Source: [24])

### 3.3 Authentication Mechanism Examples

Authentication is generally the first process taking place when a node is connecting to a new network, performed with an authentication server which uses a network access protocol. There are different authentication protocols and security mechanisms (providing authentication) exist in IoT networks. To get a better understanding for why and when authentication is needed in IoT networks, the real life scenario example can be extended as: the owner of the smart home is having some guests and one of these guests wants to show his holiday pictures to the others. In order to do this, he is willing to connect his smart phone to the smart TV. For a secure connection and to avoid any intruder to connect the IoT network at home authentication is needed to take place.

In [30] a list of common authentication protocols and security mechanisms is given as well as their well-known issues:

- **SPINS**[41], is the first security framework designed for WSNs, optimized to support resource limited environments. It provides data confidentiality and authentication, consists of two secure building blocks: µTesla and SNEP. However SNEP was never fully specified or implemented [27].

- **TinySec**, the first fully implemented link layer security suite, presented in 2004, written in nesC programming language and included in the official TinyOS \(^1\) release [27]. It provides confidentiality, integrity and message authentication; uses Skipjack\(^2\) as default block cipher and

---

\(^1\) TinyOS is an open source, BSD-licensed operating system designed for low-power wireless devices, such as those used in sensor networks, ubiquitous computing, personal area networks, smart buildings, and smart meters.

\(^2\) Skipjack is an algorithm for encryption developed by the U.S. National Security Agency (NSA) and was declassified in 1998. It operates on data blocks of 64 bits with an 80-bit key.
Cipher Block Chaining Message Authentication Code (CBC-MAC) for authentication. TinySec is no longer admitted as a secure mechanism for today’s security threats.

- **TinyECC** [42], is a public key cryptography software package designed for TinyOS in 2008, provides special optimization switches for the programmers to control the resource consumption.
- **SenSec**, introduced in 2005 with improved security features, uses Skipjack-X (more secure than Skipjack) and is not vulnerable to brute force attacks [28].
- **MiniSec**, provides confidentiality, authentication and replay protection; has two operating modes, one for single-source and one for multi-source broadcast communications. It run under TinyOS using Offset Codebook (OCB) shared key encryption mechanism [29].
- **ContikiSec**, is a ContikiOS based security mechanism. Since wireless sensor networks support many different applications, ContikiSec is designed for flexibility. It has three different security levels: Confidentiality-only (ContikiSec-Enc), Authentication-only (ContikiSec-Auth), and authentication with encryption (ContikiSec-AE). Figure 4 shows the packet formats in ContikiSec for different security levels.

```
<table>
<thead>
<tr>
<th>PBL 6</th>
<th>S 2</th>
<th>H 2</th>
<th>PAYLOAD 0-128</th>
<th>C 2</th>
<th>T 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CONTIKI PACKET

<table>
<thead>
<tr>
<th>PBL 6</th>
<th>S 2</th>
<th>H 2</th>
<th>IV 2</th>
<th>PAYLOAD 0-128</th>
<th>C 2</th>
<th>T 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CONTIKISEC-Enc PACKET

<table>
<thead>
<tr>
<th>PBL 6</th>
<th>S 2</th>
<th>H 2</th>
<th>PAYLOAD 0-128</th>
<th>MAC 4</th>
<th>T 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CONTIKISEC-Auth PACKET

<table>
<thead>
<tr>
<th>PBL 6</th>
<th>S 2</th>
<th>H 2</th>
<th>IV 2</th>
<th>PAYLOAD 0-128</th>
<th>MAC 4</th>
<th>T 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CONTIKISEC-AE PACKET
```

Figure 4 ContikiSec Packet Formats (Source: [30])

“ContikiSec has been designed to balance low energy consumption and security while conforming to a small memory footprint” [30].

- **AES CCM**: CCM is a new operation mode for Advanced Encryption Standard (AES) which combines the existing Counter (CTR) and CBC-MAC modes in order to provide encryption and authentication at the same time. The process is divided into two: CBC-MAC is responsible for authentication, generates a 128-bit value called Message Integrity Code (MIC) and appends it to the plaintext MPDU; while CTR is providing encryption and decryption for MPDU data and MIC.
The receiving node compares the MIC value received and the one it calculates after decryption, matching values ensure authentication and integrity.

To set up a strong security framework, one must know the characteristics and needs of his network very well. Setting up a security framework for a smart home and for a fire warning system in a forest are two very different things. In smart home scenario power consumption is not a very big issue same as administrative management. The owner can replace a finished battery easily or change a non-working sensor easily, so he doesn't have to care how much extra resources security framework will use, he can just have the maximum security. But on the other hand, the administrator of a fire warning system for a huge forest must balance the power consumption and the level of security. He must choose a framework to get the maximum efficiency from the nodes while providing enough security to avoid false alarms from the intruders.

Among the listed authentication mechanisms SPINS and TinySec are a little bit out date for today's standards. According to the specific needs and characteristics of the network the rest of the mechanisms in the list can be implemented, although not each of them is very secure. Flexibility feature of ContikiSec makes it the most preferable. Three different levels of security can cover needs of most of the IoT networks.
4. CONTIKI OS

Contiki is a lightweight open source operating system (OS) written in C; connecting tiny, low-cost, battery-operated and low-power objects to the internet. Contiki is a highly portable operating system, build around an event-driven kernel. Its roots reach back to 2001, when Adam Dunkels\(^3\) wanted to connect unexpected things to internet and developed uIP (micro IP)\(^4\) which spreads very quickly across the embedded world. Developments followed one another and in 2003, first Contiki was ready to be introduced.

Another open source OS designed for low power wireless devices is TinyOS, developed at the University of California. TinyOS shows many similarities, advantages and disadvantages compared to Contiki. Since TinyOS is out of the scope of this project, only Contiki’s architecture and features are introduced below. For those interested in more details about TinyOS, please refer to [48], and a detailed comparison of TinyOS and Contiki OS is made in [49].

4.1 System Architecture [32, 33]
The Contiki OS has a modular architecture, following the event driven model at the kernel while providing optional threading facilities to individual processes. A running Contiki system can be divided into 4 main parts: the kernel, the program loader, libraries and processes where a process may be either a service or an application program. A process is defined by an event handler function and an optional poll handler function. During compile time system is partitioned into two: the core and loaded programs as shown in Figure 5.

---

\(^3\) Adam Dunkels, Ph.D., is a Swedish entrepreneur, programmer and founder of Thingsquare

\(^4\) uIP was an open source TCP/IP stack capable of being used with tiny 8- and 16-bit microcontrollers
The core is compiled into a single binary image and stored in the devices, generally not modified after deployment. Programs are loaded by the program loader which obtains program binaries either by using communication stack or by using directly attached storage.

The Contiki kernel is a lightweight event scheduler sending out the events to running processes and calling the processes’ polling handlers. The execution of a process can only be triggered by dispatched events or a polling mechanism. The kernel does not preempt an event handler once it has been scheduled. Therefore event handlers must run to completion or use internal mechanisms to achieve preemption.

There are two kinds of events supported by the Contiki kernel: synchronous events and asynchronous events. Synchronous events are sent out immediately to target processes and scheduled while asynchronous events are enqueued and dispatched later.

The polling mechanism in Contiki kernel consists of high priority events that are scheduled in-between each asynchronous event. It is used by processes that operate near hardware to get status updates. When a poll is scheduled, it causes all the processes which implement the poll handler to be called according to their priority.

All operating system facilities (sensor data handling, communication, device drivers, etc.) are provided in the form of services. Each service has its own interface and implementation, applications using a particular service must only know the service interface not the implementation of it. Figure 6 shows the block diagram of the Contiki OS architecture.
4.2 Contiki Features

The Contiki OS have plenty of useful features, listed below with brief explanations.

4.2.1 Memory Allocation and Management

Contiki is designed for tiny systems that can work only a few kilobytes of memory available. A standard Contiki configuration requires 2 kilobytes of RAM and 40 kilobytes of ROM; therefore it is highly memory efficient and provides a set of memory allocation mechanisms (memory block allocation-`memb`, a managed memory allocator-`mmem` and standard C memory allocator-`malloc`).

Contiki supports dynamic memory management and dynamic linking of the programs, using managed memory allocator (`mmem`) with a primary task of keeping the allocated memory free from fragmentation by clustering the memory when blocks are free.

4.2.2 Full IP Networking

Contiki provides a full IP network stack; each application can use both IPv4 and IPv6 stack. It contains an implementation of µIP, a TCP/IP stack for 8 bit micro-controllers written in C, with minimum set of features to support TCP, UDP, ICMP and IP protocols.

4.2.3 Power Awareness

Contiki is designed for extremely low-power systems which may need to run for years on a pair of AA batteries. It doesn’t provide any power saving functions; putting the devices into sleep mode or other kind of power saving action must be handled by the applications. However, Contiki provides a system power consumption estimation mechanism to see where the power was spent.

4.2.4 6lowpan, RPL, CoAP

Contiki supports the recently standardized IETF protocols for low power IPv6 networking: 6LoWPAN, RPL IPv6 multi-hop routing protocol, and the CoAP restful application-layer protocol.

4.2.4.1 6LoWPAN

Resource limited characteristic of nodes in wireless sensor networks is the biggest obstacle between IP stack and low-rate wireless personal area networks. The IEEE standard 802.15.4 specifies physical and media access control layers for low-cost, low-speed, low-power wireless personal area networks known as 6LoWPAN. Detailed information concerning 6LoWPAN is given in section 5.4.

4.2.4.2 RPL

Routing Protocol for Low Power and Lossy Networks (RPL) is a routing protocol designed for IPv6 networks which have high packet loss rates by the IETF routing over low power and lossy network (ROLL) group. Due to memory limitations on the nodes in low-power lossy networks (LLNs), RPL is designed as a proactive distance vector protocol. It starts to build a tree like topology as soon as the network is initialized. Each node has a parent node acting as a gateway; when a node receives a packet with an unknown route, it simply forwards it to his parent. Figure 7 shows the RPL network topology.
RPL routing operation held in 2 ways: Routing Upward and Routing Downward. Each node keeps a Destination Oriented Directed Acyclic Graph (DODAG) for the topology information, containing the paths from the leaves to the root. The DODAG is built according to 4 rules [37]:

- Path metrics
- The Objective Function (OF)
- Node policies
- Loop avoidance rules

![Figure 7 RPL Network Topology](image)

For routing upward the information in DODAG is enough, each node sends the packet to its preferred parent until it reaches the root (border router). But for routing downward, RPL uses Destination Advertisement Object (DAO) messages to support the downwards traffic. RPL routing process is explained in more detail in Section 6.2.

### 4.2.4.3 CoAP

The Constrained Application Protocol (CoAP) is an application layer web protocol designed by the IETF to provide a REST⁵ (Representational State Transfer) interface with a lower cost of bandwidth and implementation complexity. Basic features of CoAP are listed by [50] below:

---

⁵ Representational State Transfer (REST) is a simple stateless architecture that generally runs over HTTP. REST is often used in mobile applications, social networking Web sites, mash up tools and automated business processes.
• Constrained machine-to-machine web protocol
• Representational State Transfer (REST) architecture
• Simple proxy and caching capabilities
• Asynchronous transaction support
• Low header overhead and parsing complexity
• URI and content-type support
• UDP binding (may use IPsec or DTLS)
• Reliable unicast and best-effort multicast support
• Built-in resource discovery

CoAP have 2 layers in the protocol stack as shown in Figure 8: CoAP Methods and CoAP Transactions.

Transactions layer is responsible for reliable UDP messaging, uses 4 different message types:

• CON indicates that a confirmation message is expected from the receiving end.
• NON indicates that no confirmation is needed.
• ACK is the acknowledgement message sent as a reply to CON.
• RST is the other reply type for the CON message, meaning some context is missing and a reset is needed.

CoAP supports the basic methods of GET, POST, PUT, DELETE, which are very similar to the HTTP methods. UDP is the default transport layer protocol for CoAP, it optionally supports DTLS for providing high level security.

![Figure 8 CoAP Layers (Source: [36])](image-url)
4.2.5 Cooja Network Simulator
Cooja is the network simulator provided by Contiki OS. Different mote types can be emulated at the hardware level, allowing the users to inspect the precise behavior of their network. Figure 9 shows a screenshot from cooja network simulator. User instructions and detailed information of Cooja will be discussed in Chapter 6.3.

4.2.6 Sleepy Routers
In a Contiki based network nodes may need to relay messages from others for reaching the destination. These relay nodes, called as ‘routers’, can be battery operated and fall into sleep mode between each relay message by the ContikiMAC radio duty cycling mechanism.

4.2.7 Protothreads
Contiki supports preemptive multithreading by using protothreads; stackless and lightweight, fully written in C to save memory and provide a nice control flow in the code. Protothreads is a mixture of the event-driven and multi-threaded programming mechanism with a very small memory overhead (two bytes per protothread).

4.2.8 Coffee Flash File System
Contiki provides file system support for devices with external flash memory chip in the form of the Coffee file system [34]. Coffee provides a programming interface for building efficient storage abstractions by using a small and constant RAM footprint per file. Coffee requires 5 kb ROM for the code and 0.5kb RAM at run-time in a standard Contiki configuration.
4.2.9 Contiki Shell
Contiki provides an optional command-line shell with a set of useful commands for development and debugging of Contiki systems.

4.2.10 Rime Stack
When bandwidth is critical or the IPv6 networking stack is overkill, Contiki provides another wireless networking stack called Rime, supporting simple operations such as sending a message to all neighbors or more complex operations like network flooding and address-free multi-hop semi-reliable data collection.
5. WIRELESS PROTOCOLS FOR LOW POWER DEVICES

Since the terms 'internet of things, smart objects, smart homes' become so popular, the interest on low cost wireless protocols has increased. Recent wireless network protocols used for low power smart objects can be listed as: Bluetooth Low Energy (BTLE), ANT, ZigBee and 6LowPAN. Contiki OS supporting 6LowPAN and partially Zigbee puts a highlight on these two for the following paragraphs.

5.1 Bluetooth Low Energy

The newest version of Bluetooth wireless technology, also known as Bluetooth Low Energy (BTLE), is announced in July 2010, designed to satisfy the needs of new modern wireless applications such as ultra-low power consumption, fast connection times, reliability and security. Table 2 shows the technical details of BTLE.

<table>
<thead>
<tr>
<th>Data Transfer</th>
<th>Supports very short data packets between 8 octets up to 27 octets with a transfer speed of 1Mbps.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency Hopping</td>
<td>Uses the adaptive frequency hopping to minimize interference from other technologies in the 2.4 GHz ISM Band.</td>
</tr>
<tr>
<td>Host Control &amp; Power Consumption</td>
<td>Intelligent controller allowing the host to sleep for longer periods of time and be woken up only when the host needs to perform some action, saves great current since the host is assumed to consume more power than the controller. Comparing to classical Bluetooth; if classic Bluetooth gets 1 as reference value for power consumption, than for BTLE this value is between 0,01-0,05.</td>
</tr>
<tr>
<td>Latency</td>
<td>Supports connection setup and data transfer as low as 3ms. Applications form a connection and then transfer authenticated data in few milliseconds for a short communication burst before quickly tearing down the connection.</td>
</tr>
<tr>
<td>Range</td>
<td>Over 100 meters.</td>
</tr>
<tr>
<td>Robustness</td>
<td>Uses a strong 24 bit CRC on all packets ensuring the maximum robustness against interference.</td>
</tr>
<tr>
<td>Security</td>
<td>Full AES-128 encryption using CCM to provide strong encryption and authentication of data packets.</td>
</tr>
<tr>
<td>Topology</td>
<td>Uses a 32 bit access address on every packet for each slave, allowing billions of devices to be connected. The technology is optimized for one-to-one connections while allowing one-to-many connections using a star topology.</td>
</tr>
<tr>
<td>Primary Use</td>
<td>Mobile phones, gaming, PCs, sport &amp; fitness, medical, automotive, industrial, automation, home electronics etc.</td>
</tr>
<tr>
<td>Profiles</td>
<td>Proximity profile, battery status, weight scale, heart rate monitor, humidity etc.</td>
</tr>
</tbody>
</table>

Table 2 BTLE Technical Details
5.2 ANT
ANT, a proven protocol for ultra low power wireless applications, allowing the network nodes to operate for years on a coin cell battery. Beyond the power efficiency feature, it includes reliable data communications, flexible and adaptive network operation and cross-talk immunity. Its primary application is in the sports and fitness fields to implement personal-area networks for performance and health monitoring. Some of its technical specifications are represented in Table 3 below.

<table>
<thead>
<tr>
<th>Date Transfer</th>
<th>Operates in the 2.4 GHz ISM band, transfers packets with 8-byte of payload and data rate of 1 Mbps.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak Power Consumption</td>
<td>~ 17 mA</td>
</tr>
<tr>
<td>Range</td>
<td>30 meters</td>
</tr>
<tr>
<td>Robustness</td>
<td>Designed to operate over eight channels. However, the sensor-node chipsets often operate on a single channel. Employs Time Division Multiplexing (TDM) system to increase reliability, and bursting to use the available spectrum aggressively by blocking other ANT devices in the vicinity.</td>
</tr>
<tr>
<td>Security</td>
<td>64bit key</td>
</tr>
<tr>
<td>Latency</td>
<td>~ 'zero'. Although ANT has 'zero' latency, it requires the receiving device to listen continuously which is not preferable for low power devices.</td>
</tr>
<tr>
<td>Topology</td>
<td>P2P, star, tree, mesh</td>
</tr>
</tbody>
</table>

Table 3 ANT Technical Specifications

5.3 ZigBee
ZigBee; a network layer protocol built on top of IEEE standard 802.15.4 MAC, designed to address the unique needs of low-cost, low-power wireless sensor and control networks by ZigBee Alliance, a consortium of more than 70 companies who have joined together to create and promote the new standard, widely used in markets as building automation, personal health care, industrial control and lighting, commercial control and etc.

Some of the characteristics of ZigBee technology is listed below [38]:

- General operation in the 2.4GHz frequency band according to IEEE 802.15.4, regional operation in the 915Mhz (Americas) and 868Mhz (Europe)
- Frequency agile solution operating over 16 channels in the 2.4GHz frequency
- Incorporates power saving mechanisms for all device classes, plus support for battery-less devices
- Discovery and pairing mechanism with full application confirmation
- Multiple star topology and inter-personal area network (PAN) communication, various transmission options including broadcast
- Security key generation mechanism, AES-128 support
Figure 10 shows ZigBee stack architecture, while Figure 11 displays a ZigBee network topology example.

Figure 10 ZigBee Network Topology (Source: [47])

Figure 11 ZigBee Stack Architecture (Source: [38])
The IEEE 802.15.4 MAC defines four frame structures for ZigBee with a maximum size of 127 bytes:

- Beacon frame used by the coordinator to transmit beacons.
- Data frame used for all transfers of data.
- Acknowledgement frame used for confirming successful transmission.
- MAC command frame used for handling all MAC peer entity control transfers.

Figure 12 shows 4 different frame formats used in Zigbee.
5.4 6LowPAN

6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks) is a new protocol, designed to overcome the resource limitation obstacle of nodes, enabling all the capabilities of IPv6 over IEEE 802.15.4 standard. 6LoWPAN basically introduces an adaptation layer between the IP stack’s link and network layers to enable efficient transmission of IPv6 datagram over 802.15.4 links, dramatically reducing IP overhead [39]. The adaptation layer is an IETF proposed standard and provides header compression to reduce transmission overhead, fragmentation to support the IPv6 minimum MTU requirement, and support for layer-two forwarding IPv6 datagram over multiple radio hops [40]. Figure 13 shows the 6LoWPAN protocol stack model.

![6LoWPAN Protocol Stack Model](image)

**Figure 13 6LoWPAN Protocol Stack Model**

**Header Compression:** IPv6 header fields are compressed by assuming usage of common values. Any fields which can be derived from link-level information or simple assumptions of shared context are elided from a packet.

The frame size of 802.15.4 is 127 octets. In an uncompressed scenario; IPv6 header is 40 octets, UDP header is 8 octets, and MAC header can be 25 octets or up to 46 octets according to the security features. As a result, in a non secure uncompressed frame 54 octets is left for payload data, while 33 octets is left with 128 bit AES security is on. Figure 14 shows compressed and uncompressed frame formats.
Fragmentation: IPv6 packets which can not fit into a single frame are fragmented. Figure 15 shows a fragmentation example. The first fragment includes a FRAG header with datagram size (11 bits) and datagram tag (16 bits). The following fragments add the offset byte (8 bits) to the FRAG header.

Layer-two Forwarding: To support layer-two forwarding of IPv6 datagrams, the adaptation layer can carry link-level addresses for the ends of an IP hop.
Table 4 below shows a brief comparison of BLE, ANT, ZigBee and 6LoWPAN according to interoperability, power consumption, packet overhead and security.

<table>
<thead>
<tr>
<th></th>
<th>BLE</th>
<th>ANT</th>
<th>ZigBee</th>
<th>6LoWPAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interoperability</td>
<td>Needs extra devices to connect IP domain</td>
<td>Needs extra devices to connect IP domain</td>
<td>Needs extra devices to connect IP domain</td>
<td>Part of Internet protocol itself, doesn’t require any extra device to connect to internet</td>
</tr>
<tr>
<td>Power Consumption</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Packet Overhead</td>
<td>Less</td>
<td>Less</td>
<td>Less</td>
<td>Very Less</td>
</tr>
<tr>
<td>Security</td>
<td>128bit</td>
<td>64bit</td>
<td>128bit</td>
<td>128bit</td>
</tr>
</tbody>
</table>

Table 4 Comparison for BLE, ANT, ZigBee and 6LoWPAN
6. METHODOLOGY

As described in Introduction chapter, the main goal of this project is to build up a working IoT network with authentication feature on. To accomplish such a task one should be familiar with the terms: "Internet of Things, Contiki, IPv6, 6LoWPAN, ZigBee, Authentication in Sensor Networks..." Since Internet of Things is a new concept, finding valid and reliable information was a challenge for the researcher; used electronic databases with keyword-searching methods to locate traditional and online sources on the topic. IEEE Xplore was the primary database used to gather information, while Halmstad University's online library catalogue was another helpful source to locate books, student papers and articles about related topics. At the end of collecting all papers and articles, the researcher categorized them under 4 keywords: Internet of Things, Contiki, Low-Power Wireless Technologies and Authentication. After the categorization process the researcher did a quick review to each paper under each category to determine a relevance and usability index (A number between 1-10, 1: useless irrelevant, 10: very useful), discarding articles with relevance index less than 5. And finally, the papers passed the quick review process were studied deeply to gain upper knowledge on each topic.

6.1 Hardware Selection

The researcher came across, the term "Thingsquare" during the hardware selection phase of this project, a Swedish originatated company providing complete solutions for IoT networks. The availability and less complex characteristic of Thingsquare devices motivated the researcher to focus on Thingsquare evaluation kits, as shown in Table 5.

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Name</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texas Instruments</td>
<td>CC2538dk</td>
<td>CC2538</td>
</tr>
<tr>
<td>Texas Instruments</td>
<td>TRXEB/CC1120emdk</td>
<td>MSP430/CC1120</td>
</tr>
<tr>
<td>ST Microelectronics</td>
<td>STM32W-SK</td>
<td>STM32W</td>
</tr>
<tr>
<td>Redwire</td>
<td>TH12/Red-io</td>
<td>MC13224v</td>
</tr>
</tbody>
</table>

Redwire's evaluation kit was purchased after a discussion with academic supervisors; lower cost, availability and pre-loaded firmware features of these devices were the keystones of selection phase.

6.1.1 Thingsquare [43] & Evaluation Kit [44]

Thingsquare Technology

Thingsquare, founded in 2012 by a group of IoT pioneers, is an end-to-end software platform which connects devices such as wireless sensors, light bulbs or thermostats to internet, allowing them to be controled by smartphones or laptops.

The technology has three major components: the mesh, the router and the cloud; as shown in Figure 16.

The Mesh: Thingsquare enabled low-power devices automatically discover each other and form a mesh network using IPv6. A range of wireless chips from several manufacturers are available to support thingsquare technology.
The Router: One of the nodes in the mesh network has internet connection, runs a special contiki based router software which is exceptionally light-weight and allows the other nodes in the network to connect internet. The router runs on tiny microcontrollers and system-on-chips requiring only 5 kilobytes of ROM and 1 kilobyte of RAM.

The Cloud: The other side of the router representing internet is known as the cloud. Thingsquare devices connect to internet without any manual configuration. Thingsquare cloud services include an online software development environment for thingsquare developers.

![Figure 16 Thingsquare Components (Source: [43])](image)

The list of the layers and corresponding protocols for each layer in thingsquare architecture is given in Table 6 below.

<table>
<thead>
<tr>
<th>LAYER</th>
<th>PROTOCOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radio</td>
<td>IEEE 802.15.4</td>
</tr>
<tr>
<td>Link</td>
<td>Ethernet</td>
</tr>
<tr>
<td>Encryption</td>
<td>AES128</td>
</tr>
<tr>
<td>MAC</td>
<td>CSMA</td>
</tr>
<tr>
<td>Network</td>
<td>IPv6/6LoWPAN</td>
</tr>
<tr>
<td>Mesh</td>
<td>RPL</td>
</tr>
<tr>
<td>Transport</td>
<td>UDP/TCP</td>
</tr>
<tr>
<td>Application</td>
<td>HTTP/WebSocket/Custom</td>
</tr>
<tr>
<td>API</td>
<td>REST</td>
</tr>
</tbody>
</table>

Table 6 Technical Specifications
**Thingsquare Routing Process**

When a thingsquare device boots up, it starts to send RPL Discovery (DIS) messages announcing that it would like to connect to a thingsquare network. Unless the encryption is manually turned off, the messages are encrypted and only the related devices can hear the messages.

The thingsquare router is the root of the RPL network, all nodes in the network have roots to the router and router’s children. The root forms the RPL network by sending RPL DIO messages to its neighbors, causing all other nodes who got the RPL DIO message to send out their own RPL DIO message containing the information of the routing metric which is the hop count to the root. The nodes with lower routing metric are considered as closer to the root and preferred as the next hop when sending out a packet. According to the number of the nodes in the network, it usually takes few minutes for all devices to exchange RPL DIOs and the network to be stabilized. When the network is converged, the nodes send DIOs seldomly in order to avoid overloading the network. Thingsquare nodes have multiple routes to reach the root, best route is chosen according to the lowest route metric, when the best route failed during the transmission process, the nodes switch to a better route for retransmission.

The router node of a thingsquare network knows all the routes to all the nodes while other nodes only know the routes for the nodes below them in the network tree. In order to route downwards in the network tree from root to nodes, the devices exchange RPL DAO messages.

**Thingsquare Router**

Thingsquare router is the gateway for thingsquare nodes to reach internet directly, responsible for the translation of the low power IPv6 packets from within the thingsquare mesh network to IPv4 which than can be routed onto the internet.

When the mesh network is formed, the router sends DNS information to all its childs. If a node wants to reach the internet server, it sends a DNS request to the DNS server provided by the router which is later routed to the internet by the router. The DNS server responds with an IPv4 address which the DNS name in the DNS request refers; the router gets the response, translates the IPv4 address into IPv6 and sends it to the related child. Now the node can communicate with internet via IPv6 address. The router is basically the bridge between IPv6 and IPv4. Figure 17 shows an example of a thingsquare node communicating the internet.

![Figure 17: Thingsquare Node Accessing Internet](Source: [43])
**Evaluation Kit**

The evaluation kit used in this project, is manufactured by Redwire LLC, consists of three TH12 low power humidity and temperature sensors, one IO embedded router(Red-io) and a programming module called PROG12.

**TH12** is an internet connected low-power wireless temperature and humidity sensor using 2xAA batteries and has 3 to 5 years of run life with the standard configuration parameters. Table 7 shows the technical specifications of Figure 18.

<table>
<thead>
<tr>
<th>TH12</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Networking</td>
<td>RPL/6LoWPAN</td>
<td>COAP REST interface</td>
<td></td>
</tr>
<tr>
<td>Pre-loaded Firmware</td>
<td>Turn on and Go</td>
<td>Reports temperature, humidity and battery voltage every 120 seconds</td>
<td>Configurable via COAP</td>
</tr>
<tr>
<td>Sensor</td>
<td>Temperature</td>
<td>Accuracy: +/- 0.5C</td>
<td>Resolution: 0.1C</td>
</tr>
<tr>
<td></td>
<td>Humidity</td>
<td>Accuracy: +/- 5% RH (+/- 2%RH typ.)</td>
<td>Resolution: 0.1% RH</td>
</tr>
<tr>
<td></td>
<td>Battery Voltage</td>
<td>+/- 5 mV</td>
<td></td>
</tr>
<tr>
<td>Indicators</td>
<td>Red led flashes when batteries are installed</td>
<td>Green led flashes when lowpan connection is acquired.</td>
<td></td>
</tr>
<tr>
<td>Power Source</td>
<td>2xAA Batteries</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dimensions</td>
<td>79mm x 54mm x 33mm</td>
<td>117gr including batteries</td>
<td></td>
</tr>
</tbody>
</table>

**Table 7 TH12 Technical Specs**

![Figure 18 TH12 Temperature and Humidity Sensor (Source: [44])](image)
**Red-io** is the low-power router running preloaded Thingsquare firmware, supports NAT64 and DNS64. It is powered by USB cable, can be programmed and debugged via USB port without any extra hardware. Hardware detail of Figure 19 is listed below:

- Redwire M12 mc13224v module
- ENC28J60 SPI-to-ethernet controller
- FT2232 dual channel USB-to-serial and USB-to-JTAG converter controller
- Link and Activity LEDs
- 2 general purpose LEDs (red and green)

![Figure 19 Red-io Router (Source: [44])](image)

**PROG12** is a USB to Tag-Connect programmer interface for TH12 devices. It, is shown in Figure 20, can be used to provide power or serial UART access for TH12s, can reset or erase the firmware running in TH12s.

![Figure 20 PROG12 Module (Source: [44])](image)
6.2 Software Simulation
Cooja, is a highly useful tool for developing Contiki based IoT networks, allowing developers to test their code before they run it on target hardware, supports variety of hardware including Thingsquare devices. Compared to working on real hardware, using this simulator with huge number of nodes is very useful and cost-effective. The researcher followed the steps described below to create a basic Thingsquare network.

Step1. Opening Cooja and creating a new simulation
In Instant Contiki 2.6.1, there is a desktop icon for Cooja, clicking the icon opens up a new terminal window and the application compiles itself. After a while Cooja network simulator opens up with a blue empty window. Clicking the **New Simulation** menu under the **File** tab, will bring up the Create New Simulation window, as shown in Figure 21. The researcher did not make any changes in advanced settings part and clicked the create button. Cooja brings up the new simulation. The **Network** window, at the top left of the screen, shows all the motes in the simulated network. The **Timeline** window, at the bottom of the screen, shows all communication events in the simulation over time. The **Mote output** window, on the right side of the screen, shows all serial port printouts from all the motes. The **Notes** window on the top right and the **Simulation control** window to start, pause, and reload the simulation. Figure 22 shows the simulation window.
Step 2. Adding motes to the simulation

In order to be able to add a mote to the network, the researcher must first create a mote type by following: Motes -> Add Motes -> Create New Mote Type from the top menu. When the cursor is on Create New Mote Type, Cooja lists the supported mote types for simulation. To simulate a Thingsquare device, the generic mote called Cooja Mote is chosen from the mote types. A new window called Create Mote Type: Compile Contiki opens, shown in Figure 23.

Figure 22 My Simulation

Figure 23 Create Mote Type
In this window, the researcher changed the mote description to **Thingsquare Router** and set **Contiki process/ Firmware** to `/home/user/thingsquare-mist-1.1.0/examples/router-node/router-node.c` and pressed the **Compile** button. Clicking the **Create** button after compilation is over, opens the **Add motes** (**Thingsquare Router**) window, where one can specify the number and the position of the mote. For this project the researcher added 1 router node with random positioning, and followed the same steps to create another mote type called **Mesh Node**. He set the firmware for the mesh node as `/home/user/thingsquare-mist-1.1.0/examples/mesh-node/mesh-node.c` and add 6 mesh nodes to the network.

**Step3. Running the simulation**

One router and 6 regular mesh nodes are now present in the network window of the simulation, start button from the simulation control window runs the simulation, additionally from the same window one can change the speed limit of the simulation to %100 and observe the real-time behavior of the network. Figure 24 shows a printed screen from the simulation.

![Simulation of a IPv6 Mesh Network](image.png)

**6.3 Hardware Implementation**

Both Red-io and the TH12s came with pre-loaded firmware, however they are not really designed to work together, at least not by default. In order to connect them with each other and build an IoT network, modification of their firmware is required.

**6.3.1 Configuring Red-io**

**Step1. Downloading the Firmware**

The red-io is running the Thingsquare Mist firmware which is a Contiki based open source software and can be downloaded from Thingsquare’s official webpage. When the download is complete, the
researcher extracted it on the Home folder of Instant Contiki 2.6.1 virtual machine. To be able to work on mc1322x devices like red-io and th12, Sourcery G++ Lite 2008q3-66 for ARM EABI toolchain must be installed. Following commands were used one by one to get and install the toolchain:

```
user@instant-contiki:~$ apt-get install ia32-libs
user@instant-contiki:~$ sh arm-2008q3-66-arm-none-eabi.bin
user@instant-contiki:~$ PATH=$PATH:$HOME/CodeSourcery/Sourcery_G++_Lite/bin
user@instant-contiki:~$ tar xjvf arm-2008q3-66-arm-none-eabi-i686-pc-linux-gnu.tar.bz2
user@instant-contiki:~$ export PATH=$PATH:$HOME/arm-2008q3/bin
```

**Step2. Modifying the Firmware**

The downloaded firmware consists of 6 main folders: apps, contiki, dev, examples, platform and tools. Since this firmware is written for different hardware platforms, not each file in these folders were actively in use of the red-io. After analyzing these folders, the researcher pointed out the c and header files which are related with the red-io.

The required modification in red-io is to set up a manual IPv6 site-local address which later can be set in Th12 as the sink's IP address and all packets can be sent here. `/thingsquare-mist-1.1.0/platform/red-io/contiki-red-io-main.c` is the main c file running in red-io and `contiki_maca_set_mac_address()` is the function responsible for setting the mac address and IPv6 address. If the variable `M12_SERIAL` is not defined or it is equal to zero, than the function creates a mac address in the form of `EC473C4D12000000`, where Redwire's OUI=EC473C M12=4D12 and next six nibbles are randomly chosen M12 serial number as hex. The researcher defined the variable M12_SERIAL as:

```
#define M12_SERIAL 0xCA093C
```

And relatively the full mac address is set to `EC:47:3C:4D:12:CA:09:3C` while site local IPv6 address gets the value `fc00::ee47:3c4d:12ca:93c`.

**Step3. Erasing the Existing Firmware**

BBMC is a small tool in the libmc1322x library using bit banging functions to erase and reset Redwire devices. Red-io does not require any extra hardware for programming and debugging, so the researcher connected it to the virtual machine via usb cable, opened up a new terminal and run the following commands:
Step 4. Flashing the Modified Firmware to Red-io

*mc1322x-load.pl* is another tool in *libmc1322x* library used for loading the images to the devices, according to the used options of this tool, it can either load the image to RAM or flash it to the ROM. Following commands used to build the modified firmware and flash it to the device:

```bash
user@instant-contiki:~$ cd thingsquare-mist-1.1.0/examples/router-node/
user@instant-contiki:~/thingsquare-mist-1.1.0/examples/router-node$ make TARGET=red-io
...
user@instant-contiki:~/thingsquare-mist-1.1.0/examples/router-node$ cd libmc1322x/tools/mc1322x-load.pl -f libmc1322x/tests/flasher_m12.bin -s thingsquare-mist-1.1.0/examples/router-node/router-node_red-io.bin -t /dev/ttyUSB1 -c 'libmc1322x/tools/ftditools/bbmc -l redbee-econotag erase'
```

- Found 1 devices with vendor id 0x0403 product id 0x6010
- Opening device 0 interface 1 using layout redbee-econotag
- setting VREF2 erase
- toggle reset
- waiting for erase
- setting VREF2 normal
- toggle reset
- done.

CONNECT

Size: 11416 bytes
Sending libmc1322x/tests/flasher_m12.bin
secondary send...
.Detecting internal nvm
nvm_detect returned: 0x00 type is: 0x00000001
nvm_erase returned: 0x00
ready
Size: 66708 bytes
Sending thingsquare-mist-1.1.0/examples/router-node/router-node_red-io.bin
Prog-Size: 66704 bytes
done sending files.
write_len: 0x00010494
6.3.2 Configuring TH12

Step1. Downloading the Firmware
Th12 firmware is based on the Contiki OS and released under the same open source license. To get the Th12 firmware following commands were used:

```
user@instant-contiki:~$ git clone https://github.com/malvira/th12.git
user@instant-contiki:~$ cd th-12
user@instant-contiki:~/th-12$ git submodule init
user@instant-contiki:~/th-12$ git submodule update
```

Step2. Modifying the Firmware
The TH12 uses 6LoWPAN and RPL to auto-configure global IPv6 address and when it is connected, a RESTful UDP based protocol CoAP is used to interact with it. The issue between TH12 and red-io is that Th12 is running a CoAP client software and it requires a CoAP server in the other side to communicate which does not exist in red-io. And the red-io is using NAT64 to operate between IPv6 and IPv4. So the downstream communication to initiating connections from ipv4 to the 6lowpan nodes is not possible. The only practical way to make these devices to communicate is to clear the default sink name in TH12 and hardcode the site-local IPv6 address of the device to POST. Manually configured IPv6 address of red-io can be set as the device to POST or a special prefix "::ffff:0/8", well-known by the router, can be used to configure the TH12 to POST to a server outside the NAT. A testing COAP server is present on internet with IP address 129.132.15.80. The researcher uses the special prefix, turns the decimal values of IPv4 address to hexadecimal and creates the IPv6 site local address of this server: ::ffff:8184:f50.

coap-post-sleep.c is the main c file which defines the variable `sink_addr` in TH12, but no value was assigned:

```
uip_ip6addr_t sink_addr;
```

The researcher first defines a default sink address:

```
#define DEFAULT_SINK_ADDRESS(x) uip_ip6addr(x, 0, 0, 0, 0, 0xFFFF, 0x8184, 0x0F50)
```

And adds the code line below inside the function `th12_config_set_default` to assign its value to related variable:

```
DEFAULT_SINK_ADDRESS(&c->sink_addr);
```
Step 3. Erasing the Existing Firmware
In this step, the researcher used the exact same commands described in erasing the firmware of red-io. The only difference in the process is that the TH12 does not have direct USB connection, it is required to use the PROG12 USB to TAG-CONNECT converter for serial access.

Step 4. Flashing the Modified Firmware
It is similar to flashing to the red-io, the following commands were used to flash the new image file into the TH12:

```
user@instant-contiki:~/th-12$ cd th-12
user@instant-contiki:~/th-12$ make TARGET=th12-lowpower
...
user@instant-contiki:~/th-12$ cd
user@instant-contiki:~/th-12$ libmc1322x/tools/mc1322x-load.pl -f libmc1322x/tests/flasher_m12.bin -s th-12/coap-post-sleep_th12-lowpower.bin -t /dev/ttyUSB1 -c 'libmc1322x/tools/ftditools/bbmc -l redbee-econotag erase'
```

6.3.3 Debugging
The researcher used debugging to observe if the code changes made the desired effect on the devices. After connecting the devices to virtual machine, terminal command `screen` was used with logging option at baud rate 115200 to see the debug messages from the devices:

```
user@instant-contiki:~/th-12$ sudo screen -L /dev/ttyUSB1 115200
user@instant-contiki:~/th-12$ sudo screen -L /dev/ttyUSB2 115200
```

Table 8 below is the omitted log screen from router device, showing that the device is configured with the desired IPv6 address and successfully got an IPv4 address from the DHCP server.

```
mc1322x init
...
setting long mac 0xec473c4d_12ca093c
Rime configured with address EC:47:3C:4D:12:CA:09:3C
CSMA Drowsie, channel check rate 8 Hz, radio channel 26
Tentative link-local IPv6 address fe80:0000:0000:0000:ee47:3c4d:12ca:093c
IPv6 addresses: fc00::ee47:3c4d:12ca:93c
fe80::ee47:3c4d:12ca:93c
MAC addr ec:47:3c:ca:09:3c
Starting DHCPv4
Inited
Requested
Warning: AES encryption is disabled
Starting 'Router node'
MAC addr ec:47:3c:ca:09:3c
Starting DHCPv4
DHCP Configured with 192.168.1.69
```

Table 8 Red-io Log Screen
Table 9 below is the omitted log screen from one of the sensor nodes. As it can be seen from the highlighted messages; the sink name and path is empty, while the sink IP address is correctly configured with the corresponding IPv6 address of the COAP server. The node checks the sink’s information, since the name is null, it uses the IP address to post. "do_dht" message indicates that the node is measuring the sensed values and displays these values as temperature, humidity and battery voltage in the following message. When the sensor data is ready, "do post" indicates a CON coap POST is sent to the server and a "NON post" is used immediately to put the node into sleep mode to save power. The last highlighted line of the log states that the node was never able to get an acknowledgement from the server, woke up twice during this log, never failed to post and never had to reply.

mc1322x init
... 
setting long mac 0x00050c2a_8cf31810 
Rime configured with address 00:05:0C:2A:8C:F3:18:10 
...
Tentative link-local IPv6 address fe80:0000:0000:0000:0205:0c2a:8cf3:1810 
Starting 'Temp/Humid Sensor' 'DNS resolver' 
Sleeping Temp/Humid Sensor 
...
sink name: 
sink path: 
...
ip addr: ::ffff:8184:f50 
sink_ok 0 wakes 0 failed 0 retry 0 
...
sink check with CON 
sink name null, trying with ip ::ffff:8184:f50 
...
do_dht scheduled 
can't sleep now, sleep not ok 
...
Power ON wake timeout expired. Ok to sleep 
go to sleep 
...
do_dht buf: {"t":27.0C","h":25.4","vb":2396mV"}
do post 
NON post 
do_dht expired 
sink_ok 0 wakes 2 failed 0 retry 0 
...

Table 9 TH12 Log Screen
7. DISCUSSION & CONCLUSION

The main purpose of this thesis was to compare the low power wireless protocols and authentication mechanisms designed for them, and building up a working IoT network with authentication as a practical work. In this chapter, the researcher shares his own comments and considerations, makes his own comparisons of low-power wireless technologies (ZigBee vs. 6LoWPAN) and authentication mechanisms, appraises the used evaluation kit and concludes the subject as well as suggesting future works.

Internet of Things vision was the first concept investigated, a very powerful expression that can turn our lives into latest science fiction movies when it is successfully implemented and integrated. Imagining a daily life surrounded by smart objects is not so difficult especially after seeing the popularity gained by smart phones in past couple of years. Internet of Things opened up a whole new window to our life, a totally new dimension; caused us to meet different new technologies like Contiki or Thingsquare, boosted up some old technologies which were already in the market such as ZigBee. But still it is an ongoing project, has so much to offer and way to go.

Contiki OS plays a critical role in IoT vision, open source behavior leads this operating system to become a standard for IoT. Among all useful features, full IP networking and, power and memory awareness are the most significant. During the short time of this thesis project 2 new versions of Contiki OS were released, 2.6.1 and 2.7; It is written in standard c programming language and supported by a world-wide developers community which grows rapidly and which is very serious about making Contiki the world-wide standard for IoT.

ZigBee is indisputably one of the most common low-resource wireless network technology and been in the market for more than 10 years; on the other hand, 6LoWPAN is a newer network protocol which is designed to bring up all the benefits of IPv6 to low resource wireless networks. Interoperability is the first factor for this comparison, and refers to the ability of a system or a device to work with other systems. For a ZigBee device to operate with non ZigBee systems requires a complex application layer gateway (ZigBee to IP translation), while a simple bridging device is enough for 6LoWPAN to work with Wi-Fi or Ethernet, hence 6LoWPAN offers better interoperability than ZigBee. Packet overhead is the second comparison factor; which is quite difficult considering the different packet types and communication scenarios. Nevertheless, 6LoWPAN does not require additional header information for IP routing and in the best case scenarios, 6LoWPAN creates one byte less overhead compared to Zigbee. Additionally, implementing the full network stack for ZigBee needs 90 kb space in ROM but only 30 kb for 6LoWPAN. From the security perspective, both ZigBee and 6LoWPAN provides 128bit AES encryption. Availability is the final comparison criteria; almost all major semiconductor manufacturers (Texas Instruments, Atmel and etc...) provide different development kits for both ZigBee and 6LoWPAN, one can easily say that they are equally available for more or less equal price. Being IP based standard gives a head start to 6LoWPAN in this race, simplicity and interoperability are the main pros. Yet, ZigBee is in this industry longer than 6LoWPAN and is adopted by many major companies. To say "6LoWPAN will replace ZigBee" would be a little bit extreme, but it will be the leading technology while ZigBee stays as a very important part of this industry.
Security, or more specifically authentication is one of the main obstacles standing in front of IoT vision, to authenticate a network with millions of nodes is not an easy task especially when the application areas and network requirements can vary a lot. However, the resource limited behavior is common in almost every node, helps us to generalize and create a security framework. Providing strong security which is divided into several levels or options to consider different needs of different applications and provide flexibility is the desired framework for IoT networks. Among the studied security mechanisms, ContikiSec is the one that fits this pattern, providing three different types of security options: encryption-only, authentication-only and encryption-authentication. For networks carrying very sensitive data and strict access rules, encryption-authentication mode should be implemented; but if power consumption is the most important concern than a less secure mode, encryption-only or authentication-only should be used. Thingsquare can be defined as the "practical" form of Internet of Things in some sense. And they choose to use AES CCM for all their wireless communications which is similar to ContikiSec, uses the same underlying block chiper AES and provides two options of security levels: secure and non-secure.

During the practical phase of this thesis, an Internet of Things network with 3 sensor nodes and a router was created with 6LoWPAN as the wireless communication medium. Besides, the router had an Ethernet connection to an IPv4 DHCP server to let the other nodes to connect to the internet. Yet, some compatibility issues between the router and sensor devices arose which were not discovered or considered while selecting the evaluation kit. These issues can be listed under two titles: TH12 originated issues and Red-io originated issues.

Since the transition of internet from IPv4 to IPv6 is not close to be over, IoT networks consist of two IP domains. IPv6, between the sensor nodes and the router; IPv4, between the router and the rest of the internet. The router Red-io must have at least one interface in each domain and works as a gateway between the two IP domains, uses NAT64 as network address translation protocol to translate the IP addresses between IPv4 and IPv6. The nodes inside the IPv6 domain have no idea what happens in the network after the router node. Same rule applies to the other side of the network. The design of NAT64 in red-io does not allow initial connections outside the IPv6 domain and therefore makes it difficult to receive the incoming UDP packets which were heavily used by the COAP-based firmware of TH12. This issue is comparatively solved by hard-coding the IPv6 address of the COAP server in the IPv4 domain using the special IPv6 prefix: "::ffff:0/8" to the TH12s.

The main issue originated from TH12 is the designed software which is based on COAP protocol. Th12s are primarily designed to work with a special 6LoWPAN router running a special Linux based software called BRamble. This router allows the sensors to connect directly to a web based COAP server: "lowpan.com" which provides different management tools. But since Contiki OS is one of the main investigation material in this thesis, a Linux based device is out of the scope of this project. Developers considered ease of management and power consumption during the design of TH12 firmware, but not security. Hence, AES CCM or any other security mechanism is not supported by the sensor nodes which leads the researcher to disable the security features of the router to keep the communication between the devices on.
Under all these circumstances, it is hardly understandable that "TH12/red-io" is listed as an evaluation kit in Thingsquare technology webpage. It requires a whole new firmware for the TH12s to fully support the Thingsquare Mist project. In fact, it is mentioned by Redwire LLC that a "non-COAP" based firmware may be available for TH12 sensors in the future to overcome the compatibility issues.

As future work, 3 different opinions are listed below;

- ZigBee Alliance introduced an IPv6-based full wireless mesh networking solution ZigBee IP in 2013, investigating this new technology and comparing with 6LoWPAN may be an interesting research study.
- By the end of 2013, Thingsquare introduced a new evaluation kit with full online software development platform support. Unlike the devices used in this study, they are designed to work together without any compatibility issues. Therefore, implementing an IoT network with security features on this evaluation kit can be a possible field for further studies.
- Californium is a COAP framework written in Java. Implementation of this framework as COAP server can lead us to investigate the sink side of the communication which was not possible during this study.
8. REFERENCES


URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6107386&isnumber=6107324


URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5579493&isnumber=5579103


URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5689806&isnumber=5688739


URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6017172&isnumber=6017158


URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6006307&isnumber=6006106


URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6009807&isnumber=6009617


URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5579543&isnumber=5579103


URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6201865&isnumber=6201368


URL: http://www.mckinseyquarterly.com/High_Tech/Hardware/The_Internet_of_Things_2538?gp=1


URL: http://www.eyeonearth.eu/en-us/Pages/Home.aspx

[14] HP, “Central Nervous System for Earth (CeNSE),” 2010
URL: http://www.hpl.hp.com/research/intelligent_infrastructure/

URL: http://www.internet-of-things-research.eu/

URL: http://ioteg.meraka.csir.co.za

URL: http://link.springer.com/article/10.1007/s11277-011-0385-5#

URL: http://www.zdnet.com/m2m-and-the-internet-of-things-how-secure-is-it_p3-7000008389/


URL: http://www.schneier.com/blog/archives/2006/08/updating_the_tr.html


[38] URL:http://www.zigbee.org/Specifications/ZigBee/Overview.aspx


[43] Thingsquare Homepage
URL: http://www.thingsquare.com/

[44] Redwire LLC Homepage
URL: http://redwirellc.com


[48] TinyOS Homepage
URL: http://www.tinyos.net/

URL: http://www.net.in.tum.de/fileadmin/TUM/NET/NET-2012-08-2/NET-2012-08-2_02.pdf

Samet KALYONCU
Bachelor Degree -- Computer Education and Instructional Technology -- Karadeniz Technical University, Turkey