Technical University of Crete

School of Electrical & Computer Engineering

Electronics & Computer Architecture Division

A Secure Network-Layer Bridge for Wireless Sensor Networks

by

Emmanouil Palavras

A Thesis submitted in partial fulfilment of the requirements for the Diploma in Electrical & Computer Engineering

June 2017

Thesis Committee

Associate Professor Ioannis Papaefstathiou	*Thesis Supervisor*
Associate Professor Aggelos Bletsas	
Dr. Konstantinos Fysarakis (FORTH)	

This page is intentionally left blank.

Abstract

As the Internet of Things (IoT) is becoming more and more popular, Wireless Sensor Networks (WSNs) market share increases, due to their efficiency. WSNs typically consist of embedded nodes with inherent limitations in processing power, energy, memory and communications bandwidth. It is an emerging technology that demonstrates true potential through many applications. Although they can be easily integrated into existing systems and products, hostile environments mandate the deployment of secure schemes to protect sensitive data being transmitted. Standardized security mechanisms, though, were not designed with resource restrictions in mind, rendering their applicability in such environments ineffective, if not impossible.

6LoWPAN (IPv6 over Low-power Wireless Personal Area Networks) activities have paved the way for using IPv6 protocols in WSNs by introducing appropriate header compression formats. These allow the efficient exchange of IPv6 packets over IEEE 802.15.4 based networks. However, when a message protected with encryption methods that utilize header compression needs to cross the boundaries of a WSN and establish a secure channel with a remote party, thus offering end-to-end security, a gateway should be used that will facilitate message relay from one network to the other without removing protection.

This thesis explores appropriate solutions and describes the implementation of a secure gateway that can be deployed by a WSN to ensure end-to-end security between a sensor node and a remote party. This requires the development of a border router to guarantee seamless communication between WSNs and infrastructure nodes that allows secure remote access to WSN resources to authorized parties. Moreover, the secure gateway enables the communication between nodes that implement different protocols or are outside the WSN (e.g. Ethernet).

Keywords: Internet of Things, IoT, Wireless Sensor Network, WSN, 6LoWPAN, security, secure communication

Acknowledgments

As this ship is approaching the dock, it is my obligation to thank everyone who supported me throughout these years.

First of all, I would like to thank my family for providing me with support and encouragement, making it possible for me to study for and get this diploma.

Additionally, I would like to express my gratitude to my thesis supervisor, Associate Professor Ioannis Papaefstathiou, for giving me the opportunity to explore the world of embedded systems and therefore their limitations.

Moreover, I have to thank Dr. Konstantinos Fysarakis for his guidance and his support that lead me overcome all the hindrances that turned up. His help and suggestions acted like a lighthouse in the storm.

Last but not least, I feel obliged to thank my friends, the old ones that somehow managed to tolerate me for that long and the new ones I made during my studies. They are always there when I need them and for that I cannot express my gratitude enough. For privacy reasons (think security), I will only mention two who were directly related to this thesis. Thank you, Sotiris, for your help and cooperation regarding some common obstacles we encountered during our thesis and for your MQTT source code that you provided me. Thank you, Stelios, for the proofreading of this thesis and the perceptive comments you made.

Thank you, myself, for taking a life vest, although it was not enough for heavy sea…

To my dear grandmother, Eleni,
for her unparalleled support.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

6LBR	6loWPAN/RPL Border Router
6LoWPAN	IPv6 over Low-power Wireless Personal Area Networks
ACL	Access Control List
ADC	Analog-to-Digital Converter
AES	Advanced Encryption Standard
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CCM	Counter with CBC-MAC (Cipher Block Chaining Message Authentication Code)
CCTV	Closed-Circuit TeleVision
CERN	The European Organization for Nuclear Research
CETIC	Centre of Excellence in Information and Communication Technologies (Belgium)
CoAP	Constraint Application Protocol
CPU	Central Processing Unit
CSMA/CA	Carrier-Sense Multiple Access with Collision Avoidance
DDoS	Distributed Denial of Service (Attack)
DES	Data Encryption Standard
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DODAG	Destination Oriented Directed Acyclic Graph
DSP	Digital Signal Processor
DTLS	Datagram Transport Layer Security
DVI	Digital Visual Interface
DVR	Digital Video Recorder
ECB	Electronic CodeBook
EXI	Efficient eXtensible Interchange
GPIO	General Purpose Input/Output
GPU	Graphics Processing Unit
HDMI	High-Definition Multimedia Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IC	Integrated Circuit
IM	Instant Messaging
IoT	Internet of Things
IP	Internet Protocol (Address)
IV	Initialization Vector
JSON	JavaScript Object Notation
JTAG	Joint Test Action Group
LAN	Local-Area Network
LCD	Liquid-Crystal Display

LED	Light-Emitting Diode
LLN	Low-power and Lossy Network
LLSEC	Link-Layer SECurity
LR-WPAN	Low-Rate Wireless Personal Area Network
LWT	Last Will and Testament
MAC	Medium Access Control
MCU	MicroController Unit
MQTT	Message Queuing Telemetry Transport
NAS	Network-Attached Storage
NIST	National Institute of Standards and Technology (United States of America)
OS	Operating System
PAN	Personal-Area Network
PHY	The PHYsical layer of OSI model
PKCS#7	Public Key Cryptography Standard #7
QoS	Quality of Service
RAM	Random-Access Memory
REST	Representational State Transfer
RF	Radio Frequency
ROM	Read-Only Memory
RPC	Remote Procedure Call
RPL	IPv6 Routing Protocol for Low-power and Lossy networks
RTT	Round-Trip Time
SASL	Simple Authentication and Security Layer
SDRAM	Synchronous Dynamic RAM
SLAAC	Stateless Address AutoConfiguration
SOAP	Simple Object Access Protocol
SoC	System on a Chip
SSL	Secure Sockets Layer (TLS predecessor)
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB OTG	Universal Serial Bus, On-The-Go
VoIP	Voice over IP
WPAN	Wireless Personal-Area Network
WSAN	Wireless Sensor and Actuator Network
WSN	Wireless Sensor Network
XML	eXtensible Markup Language
XMPP	eXtensible Messaging and Presence Protocol

1. Introduction

"Engineering without security is just art"[1]

The proliferation of the Internet of Things has introduced new challenges regarding the security aspect of these devices. Of course, there are a few reasons for the significant increase in their popularity. They are easy to use or to integrate to existing systems, they offer convenience in day to day tasks and they come in with useful features. The Internet of Things includes sensors and actuators, among other things, that can be controlled and monitored from anywhere. Contemporary buildings and houses are now equipped with this state of the art technology making them 'smart' and energy efficient.

In general, any device/appliance will be able to connect to the Internet in the near future [1], but the security implications are tremendous. The vast majority of manufacturers are unconcerned by the security of their products and are only interested in what is cost efficient and profitable. The problem is exacerbated when the Internet connection capability is added to the product after its final design, leading to severe security risks, because it was not designed with security in mind.

Another key problem is the diversity of existing protocols. Although each protocol is ideal for different applications, it is often difficult to ensure machine to machine communication, when needed, especially in a secure manner. Proprietary protocols are also a big concern, because no one can guarantee that they are secure, unless third-party security auditing comes into place, but that is usually not the case.

One may wonder "But, what can go wrong?". Sadly enough, there are a lot of examples (read incidents) out there. The most obvious repercussions are the control of a system by a malicious adversary and the invasion of the owners' privacy. Imagine, for instance, a home security system that safeguards a house and is connected to the internet ready to take commands. It may also communicate with its manufacturer regarding these commands or possible firmware updates. If this communication is not secure, literally anyone can break into this house by simply disarming the system. They can also make sure the house is empty, by querying the status of the system. This example is anything but imaginary. Real home security systems examined by experts were found using weak security practices or none. [2] [3]

What about privacy? If a 'smart' thermostat sends its configuration out in the clear, it is trivial for anyone monitoring the configuration to deduce whether the building is empty. Needless to say that any Internet-connected device with a camera would be able to compromise its owner's privacy entirely. One of the latent implications of unsecure 'things' was demonstrated recently when a botnet of IoT devices attacked the blog of a journalist [4] and a DNS provider [5] using the Distributed Denial of Service (DDoS) attack achieved unprecedented loads.

When dealing with WSNs where low power consumption is being pursued, adding security mechanisms can be a daunting task. To achieve energy efficiency, these devices are constrained in resources and thus incapable of executing common resource-intensive security algorithms. IPv6 is also a contributing factor that cannot be overlooked. Although it allows any WSN node or any device in general to have a public IP address, it may lead to detrimental effects. Nobody can deny that the Network Address Translation mechanism offers some protection by

[1] For the meaning of the quote, see *5.3 Lessons Learned*.

hiding the network topology behind a public IP. With IPv6, a network administrator is left with proper network segmentation and a firewall to protect the devices.

1.1 Purpose

This thesis aims to the implementation of a secure gateway that ensures end-to-end security between a sensor node and a remote party or between sensor nodes. As each node may use different application protocols, the secure gateway is able to convert the transmitted data between protocols according to what protocols the recipient node understands.

1.2 Limitations

This thesis acts as a proof of concept of a secure gateway. Although it covers a variety of widely-used protocols, the supported protocols are not the only ones available. As far as the security algorithms used are concerned, the constrained resources of the Zolertia Z1 WSN nodes had a great impact on the amount of such algorithms tested. As an example, the Datagram Transport Layer Security (DTLS) could not fit inside the memory of this platform.

1.3 Method

The first step towards the development of this thesis was the familiarization with the Zolertia Z1 nodes, the Contiki OS they run, the 6LBR and the corresponding tools available. At the same time, a literature study was made on the available protocols supported by this platform and what security options exist. Additional protocols have been chosen that run on different platforms in order to demonstrate the idea of machine-to-machine communication regardless of the application protocol or the platform. Some algorithms have been excluded on account of the incapability of the platform's resources to handle them.

2. Technical Background

It is of paramount importance to describe any technology, algorithm or protocol mentioned herein, so that anyone will be able to understand the main concepts of this thesis. As this thesis is security related, it would be a notable omission not to state two of the most significant rules of security, except for the Kerckhoffs' principle [6], regarding cryptographic algorithms:

a) *You should not invent your own crypto.*

The main reason behind this is dubbed as "Schneier's Law" [7] which states that anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that they themselves cannot break. The rationale is that the creation of such algorithm requires strong cryptographic background and experience. The security of these algorithms has to be proved mathematically and cryptographers have to review the algorithm and its proof. All the well-known algorithms have withstood rigorous auditing throughout the years and their limitations, as well as their weaknesses are known and documented.

b) *You should not implement any crypto library.*

Despite the proven security of a crypto algorithm, an inexperienced individual may introduce vulnerabilities to the system if the implementation itself is not secure. A common example of this is the timing attack, a side channel attack in which the adversary attempts to compromise a system by analysing the time taken to execute such an algorithm.

These rules are not necessarily limited to cryptographic algorithms. They are still valid for any security algorithm or procedure.

2.1. The Internet of Things (IoT)

"The S in IoT stands for Security"

The Internet of Things includes any kind of device that is or can be connected to a network -not necessarily the Internet. Sensors, actuators, embedded devices are the obvious ones, but as these can be integrated into anything, IoT also includes buildings, appliances, vehicles. Some examples will make things more clear. Heating, Ventilation and Air Conditioning (HVAC) systems are installed throughout a building and its components such as thermostats are interconnected. The whole system can be controlled and be monitored from a remote place as well. Modern 'smart' televisions can now be connected to the Internet, offering much more features and capabilities. Digital Video Recorders (DVRs), that enable us to record a TV program, a Closed-Circuit Television (CCTV) feed or any feed for that matter, to a storage device, have now network capabilities extending the options both for record sources (Internet or network based) and storage destinations such as Network-Attached Storages (NAS) or a cloud provider. Interconnected vehicles are also en route. Apart from the internal entertainment system (videos, TV, Internet surfing, radio etc.), vehicle-to-vehicle communication will make possible the development of a more resilient accident prevention system, as well as alternative route suggestions to avoid traffic jams. According to studies, by 2020, a quarter billion connected vehicles will have automated driving capabilities [8].

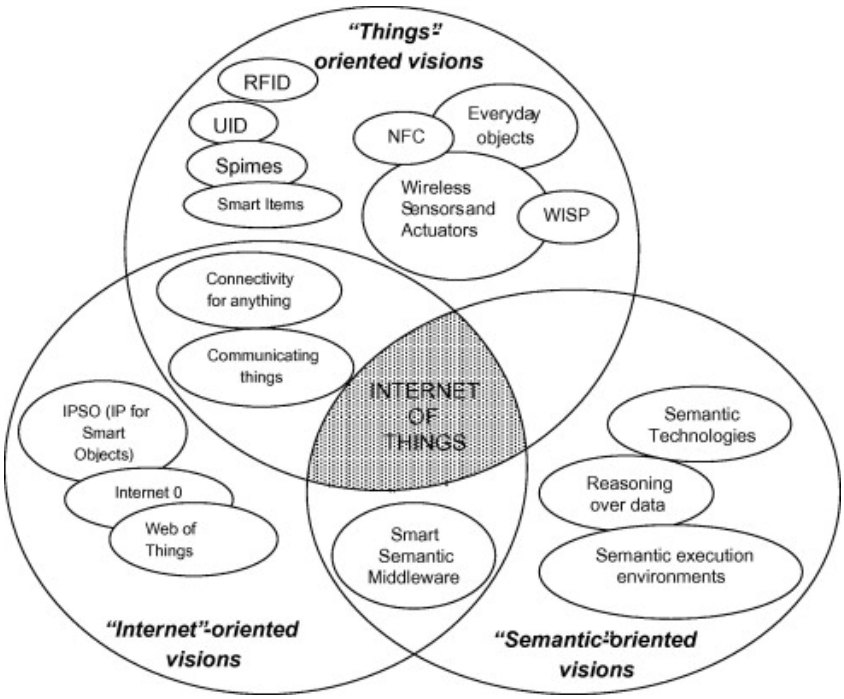


Figure 2.1: "Internet of Things" paradigm as a result of the convergence of different visions. [9]

The definition and the actual applications of the IoT are hardly bounded by the examples above. Depending on one's perspective, different and enthralling ideas may be conceived. The IoT paradigm shall be the result of the convergence of three main visions, as depicted in Figure 2.1. The 'Things'-oriented visions, whose focus is the objects and their integration into a common framework, the 'Internet'-oriented visions, which refers to the networking capability of the said objects, and the 'semantic'-oriented visions that are introduced in order to handle the representation and storing of the exchanged information between the objects [9]. With IoT, the possibilities are endless.

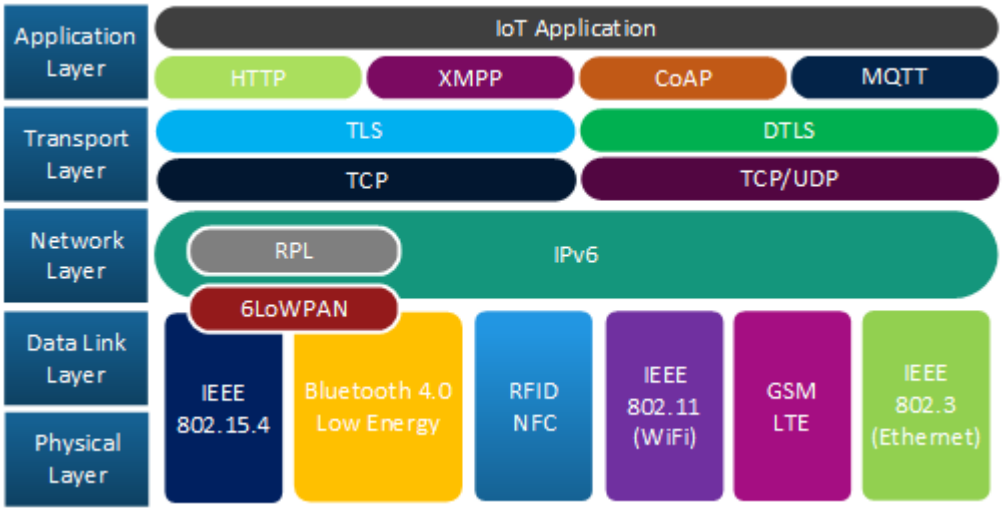


Figure 2.2: An IoT Protocol Stack

The benefits of the IoT are also noteworthy. To name a few, significant cost savings, additional revenue streams, productivity improvement. [10] Besides the aforementioned examples, IoT can be used to improve living standards. By developing Smart Cities [11], optimization of public services such as transport, parking and lighting can be achieved.

Furthermore, the numerous data gathered can be used to increase transparency and support the coffers of a country [9]. A variety of wearable health devices may provide vital information to doctors enabling them to act on-time accordingly. Especially the elderly would benefit from not only such devices but also from monitoring implants such as pacemakers.

Although the IoT Protocol Stack depicted in Figure 2.2 mainly contains the protocols that were utilized in this thesis, there is a plethora of protocols that are currently used in IoT. Many of these protocols are not limited to the IoT world but they are also used in our everyday life. This makes the integration of IoT devices even more smooth and easy, without the need of special equipment in many cases. Researchers try to create new protocols, that are especially designed for resource-constrained devices, with the integration of existing infrastructure in mind, so that these new protocols can easily be accepted and embraced. An example of this is the 6LoWPAN protocol which is based on the IPv6 (see below).

It is beyond any refutation that IoT -like anything- can be exploited for nefarious causes. Unfortunately, the lack of legislation that forces manufacturers to make their products compliant with strict security standards, when combined with unmotivated manufacturers to properly secure their products, can only lead to deleterious results. Apart from the DDoS attacks mentioned in the Introduction, which were mainly utilized by the Mirai botnet [12] consisting of unsecure CCTVs, DVRs and others with hardwired or common/unchanged passwords, there is also a self-propagating smart light bulb worm that spreads rapidly and could also lead to DDoS attacks or simply destroy them [13]. The problems do not stop there. New vulnerabilities are going to be exposed, thus more exploits, more attacks. Another fact that makes IoT so unwanted by anyone who comprehends the security risks they bring with them [14], is that many of them cannot be updated in any way, rendering them a high-value target to a malicious party. In the wake of all these risks and the continuous proliferation of IoT, a great deal of security considerations have emerged [15].

2.2. Wireless Sensor Networks (WSNs)

A common aspect of IoT are Wireless Sensor Networks. These are networks consisting of a fair or large amount of relatively small wireless devices equipped with various sensors. An individual may refer to them as Wireless Sensor and Actuator Networks (WSANs), because there are many applications where sensing is not enough and a need to act upon these devices as a result of a measurement exists. Therefore, some devices are able to control a system whether it is a microprocessor or an electric valve. The fact that they are wireless makes them more appealing compared to installing cables, thus they are easier to deploy and flexible, especially when wiring installation is impossible or has prohibitive costs.

These devices are low-cost, low-power and in addition to their batteries they are usually also equipped with an ambient energy harvesting device, such as solar panels or devices that transform kinetic energy -usually vibrations- to electrical energy, diminishing the necessity for battery replacements. Furthermore, a WSN may be heterogeneous, meaning that the devices that constitute a WSN may have different functions, have been manufactured by different vendors, etc. Should they implement the same protocols and follow the same standards, they will have no problem communicating with each other.

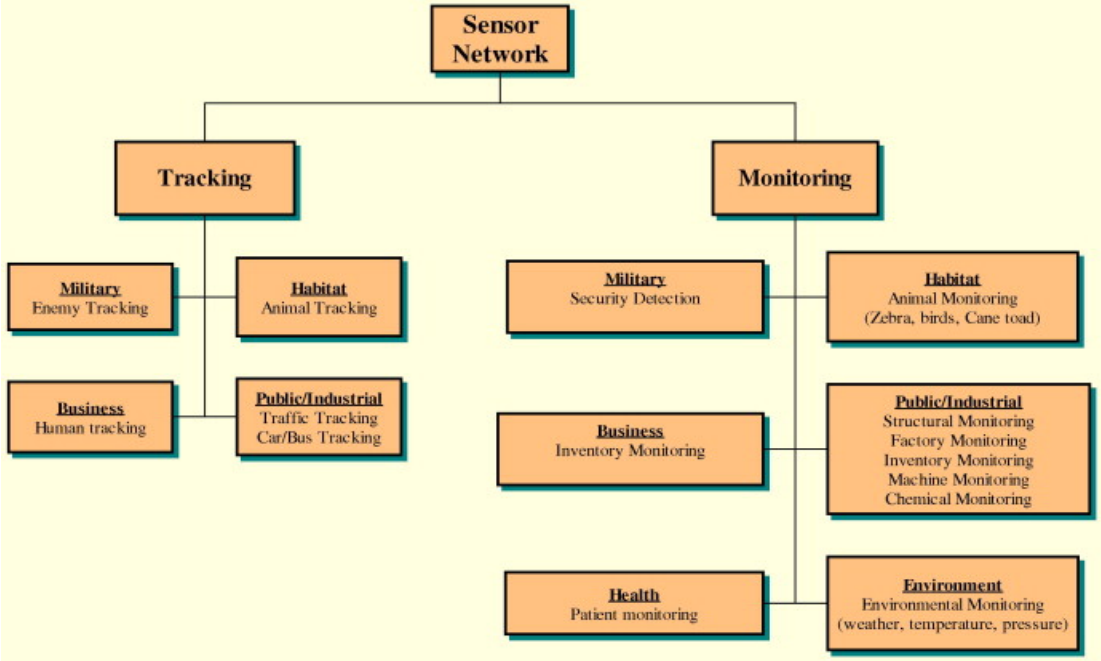


Figure 2.3: Overview of sensor applications [16]

WSNs can be used in a variety of applications as shown in the Figure above. They are frequently found in the smart grid, which is the power grid with a sense of smart devices that enable an efficient way to manage every complex subsystem. Online monitoring of transmission lines and substations may lead to reduced black-out periods. Apart from smart water networks, which can also take advantage of WSNs, any building without a proper cable infrastructure can be turned into a smart one with minimum cost [16].

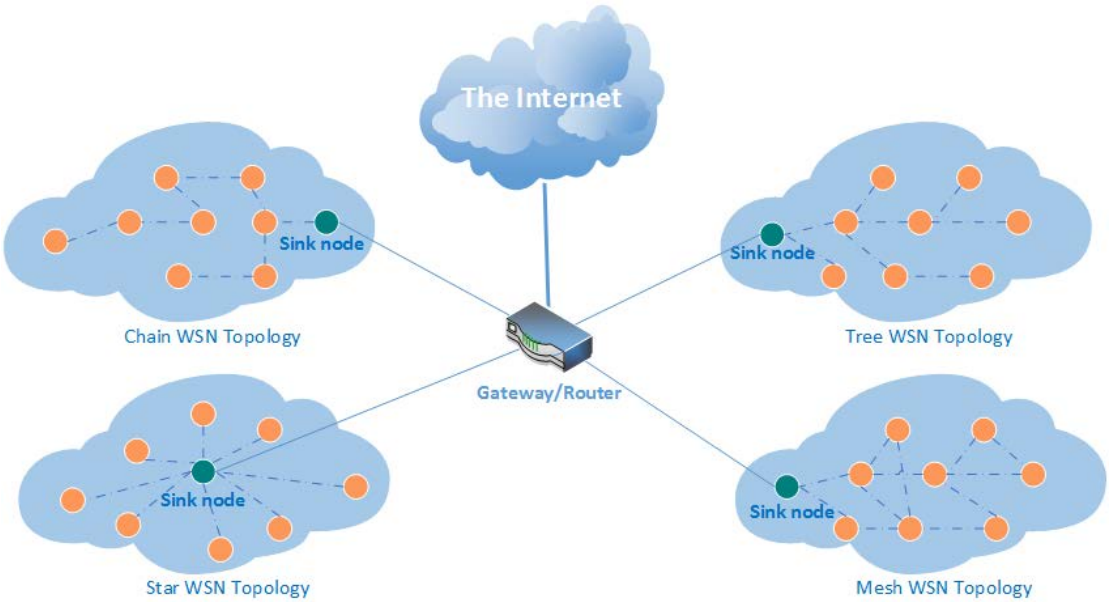


Figure 2.4: WSN topologies

Depicted in the Figure above, WSNs can be deployed in various topologies depending on the needs and constraints of every case. Although the mesh topology enables nodes to establish multiple paths, thus creating redundancy and high availability -especially when combined with more than one sink nodes-, it requires the corresponding protocols to be implemented and run

in those devices, that may not be feasible. Another key fact is the scalability some of these topologies offer. Primarily, the range of these devices can be limited in view of lower power consumption, but there is no need for every node to have the sink node within its range, since a neighbour node can relay its messages. As a result, a WSN may cover large areas, if each node has at least one other node within its range.

2.3. Communication Protocols

The following subsections describe the protocols utilized in this thesis. These protocols belong to one of the four first layers of the Open Systems Interconnection (OSI) model: Physical layer, Data link layer, Network layer and Transport layer as depicted in Figure 2.2.

2.3.1 IEEE 802.15.4

When it comes to low-power and resource-constrained devices deployed in a WSN, the need for a suitable communications standard emerges. The IEEE 802.15.4 is one of the most established standards for that purpose and defines the physical layer (PHY) and medium access control (MAC) sublayer specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs) [17]. The first version of this standard was released in 2003 by the IEEE 802.15 working group.

There are several protocols that are based on IEEE 802.15.4, such as ZigBee, 6LoWPAN (see below), SNAP and Thread, meaning that they develop the layers not defined by this standard. For low consumption to be attained, the transfer rate is bounded to 250 Kibps. The achievable range is 10m to 20m. Depending on one's needs these can be adjusted in favour of more power efficiency, especially in situations where neighbour nodes are even closer or lower data rates are viable. Despite the low transfer rate, this standard can also be used in real-time applications and also features collision avoidance through CSMA/CA on the physical layer. As far as security is concerned, the CCM* protocol is included in this standard, offering authentication and encryption.

2.3.2 Internet Protocol version 6 (IPv6)

The Internet is based on the relaying of data packets, called datagrams, across network boundaries. The communications protocol that enables this routing is the Internet Protocol (IP) [18]. There are currently two versions: 4 and 6. The former was the only version deployed and used until the need of more addresses emerged. The exponential growth of network-capable devices led to the exhaustion of the nearly 4.3 billion addresses available (2^{32} to be exact). IPv6 offers $7.9 \cdot 10^{28}$ *more* addresses (2^{128}) which is an enormous number, implying that each device may be assigned with one or more public IPv6 addresses.

While the representation of an IPv4 address is 4 bytes with each byte being separated by dots ($x.x.x.x$), the IPv6 address consists of eight groups of four hexadecimal digits with the groups being separated by colons ($hhhh{:}\ hhhh{:}\ hhhh{:}\ hhhh{:}\ hhhh{:}\ hhhh{:}\ hhhh{:}\ hhhh$). One of the main features of IPv6 is the Stateless Address AutoConfiguration (SLAAC) which enables each host to configure itself automatically. Moreover, the design of IPv6 results to less processing in routers, emphasizing the end-to-end principle of network design [19].

2.3.3 6LoWPAN

A common misconception was that the Internet Protocol (IP) is too heavy weight for Personal Area Networks (PANs) where devices with limited resources are used. As a new protocol in IP's position would require complex gateways and tools for configuration,

management and debugging, 6LoWPAN was created. 6LoWPAN stands for IPv6 over Low-power Wireless Personal Area Networks [20]. IPv6 uses stacked headers, which implies that only the necessary information is included in headers depending on the actual needs. For instance, if the packets are short enough, the header field for fragmentation is not needed. By exploiting stacked headers and the compression of the IPv6 header itself, a minimum of 4 bytes is achieved. The following facts contribute to the compression:

- the low-order 64 bits of an IPv6 address (the link local address can be the device's MAC address)
- the 802.15.4 frame carries these MAC addresses
- Several fields in the IPv6 header are static.

Table 2.1: Stack Comparison

	ZigBee	Zensys	6LoWPAN
Code Size with mesh	32K to 64K+	32K	22K
Code Size w/o mesh	Not Possible	Not Possible	12K
RAM requirements	8K	<2K	4K
Header Overhead	8 to 16 bytes	Proprietary	2 to 11 bytes
Network Size	~65K	232	2^{64}
RF Radio support	802.15.4	Proprietary	802.15.4 ++
Transport Layer	None	Proprietary	UDP/TCP
Mesh Network Support	ZigBee	Zensys	Many
Internet Connectivity	ZigBee Gateway	Zensys Gateway	Bridge/Router

6LoWPAN creators made the Table 2.1 above, in order to demonstrate its benefits over other commonly used stacks. For more information about a 6LoWPAN header, one can read [20] or any other reliable source for that matter.

2.3.4 RPL

It has become evident that resource-constrained devices require special handling. Among the protocols and technologies created to serve these devices is the RPL, an IPv6 Routing Protocol for Low-power and lossy networks. Routing is the mechanism that defines the right path a message should take to reach its destination. Depending on the routing protocol, the notion of the right path usually means that it is an existent path with the minimum possible cost. The cost is based on various metrics such as the capacity of the links through which the packet is going to travel, how congested they are and how many different nodes (hops) are between the sender and the recipient for this route. The requirements that existing routing protocols failed to comply with are the limited memory required for storing the routing state, the proper handling of link failures, the imposed constraints of the control traffic and the actual properties of links and nodes that determine the cost of each routing path. [21]

Low-power and Lossy Networks (LLNs) usually comprise up to thousands of resource-constrained nodes, typically supporting only low data rates over lossy links. Traffic patterns such as multipoint-to-point or vice versa are common in LLNs. [22] RPL specifies how a Destination Oriented Directed Acyclic Graph (DODAG) can be built using a set of metrics and an objective function that computes the best path according to these metrics. Both link and node properties

are considered in the computation of the best path. As there may be traffic with different sets of requirements, multiple graphs can be active simultaneously.

RPL offers three security modes. In the "unsecured" mode, the RPL control messages do not utilize any security mechanisms, although there may be other security primitives such as link-layer security. The "pre-installed" mode requires each node to have pre-installed keys before joining the network and the "authenticated" mode enables nodes with pre-installed keys to join as leaf nodes and forwarding nodes obtain keys from an authentication authority. There are different levels of security for each RPL message that provide integrity, replay protection and confidentiality. Delay protection is also an option.

2.3.5 Transmission Control Protocol (TCP)

TCP is one of the most commonly used transport layer protocols. Packets transmitted using this protocol are delivered in-order and reliably, so that protocols on the application layer do not have to deal with lost, duplicated or out-of-order packets. TCP employs flow and congestion control. The former is achieved by limiting the transfer rates in a way not to overwhelm the receiver's buffers whereas the latter ensures that the links between the sender and the receiver will not get congested leading to network performance degradation.

2.3.6 User Datagram Protocol (UDP)

UDP is another commonly used transport layer protocol. This protocol is far less complex than TCP as it aims at speed rather than reliability, meaning that there is no guarantee that each packet (datagram) will arrive at its destination. Deduplication and ordering have to be handled by the application layer. While a TCP connection requires a handshake prior to the connection establishment, UDP is connectionless. It is best described as a best-effort protocol. All these features make UDP ideal for real-time applications, such as Voice over IP (VoIP), where latency is a primary concern compared to reliability.

2.4. Application-Level Protocols

2.4.1 HyperText Transfer Protocol (HTTP)

In 1989, Tim Berners-Lee, the inventor of the World Wide Web, began developing the HTTP with his team at CERN. HTTP is one of the most used application protocols as it is an integral part of the World Wide Web. When someone visits a webpage via a web browser, HTTP is used to make the communication between the client (e.g. browser) and the server (a webserver) possible, although it is not the only protocol available. A webpage is a Hypertext, a structured text with links (hyperlinks) to other webpages, that is delivered as a response to the browser's request. Each request-response transaction constitutes an HTTP session, although persistent connections may be used to allow more than one transaction without paying the penalty of establishing a new connection. The said penalty is due to the fact that HTTP uses TCP.

HTTP is also a stateless protocol, meaning that there is no need for the webserver to retain information between sessions. So, if it weren't for the cookies, there would be no way to distinguish whether a user is authenticated or not. Visiting a webpage requires the corresponding Uniform Resource Locator (URL) of the webpage, because HTTP resources are identified by those. That is to say, a client asks a particular webserver for a resource which can be a document, a webpage etc. HTTP defines keywords known as verbs or methods which instruct the webserver what to do with the said resource. For instance, *GET* is used when the client wants to retrieve the resource, *PUT* is used when the client updates or modifies the resource and so on.

2.4.2 REpresentational State Transfer (REST)

As the Web was evolving, a need for Web services started to emerge. It became evident that the Web is not only webservers serving webpages and browsers getting those pages, but also other electronic devices that would be able to communicate with each other. This functionality is offered by a web service, which facilitates the machine-to-machine communication, a fundamental functionality in the abysmal IoT world. RESTful Web services enables access and modification of Web resources using stateless operations. Created by Roy Fielding in 2000 and defined in his doctoral dissertation, RESTful Web services almost always go hand in hand with HTTP. More specifically, through the HTTP verbs and Uniform Resource Identifiers (URIs) a device makes an HTTP request for a Web resource and receives a response in a predefined format like HTML, JSON or XML.

With that in mind, the World Wide Web can be viewed as a REST-based architecture. In fact, there are many Web frameworks that promote the use of RESTful Web services to deliver webpages since smartphone applications would need RESTful Web services anyway. The main advantage over other mechanisms like Remote Procedure Calls (RPCs) and Web services like Simple Object Access Protocol (SOAP) is the simplicity and the fact that it is fully-featured although it is lightweight.

2.4.3 Websocket

Designers started making webpages more and more user friendly and more feature rich. Towards that step, JavaScript, the programming language that browsers can run natively, without any need for plugins, paved the way, enabling the development of dynamic webpages where, for instance, the content can change without reloading the whole page. That would not be possible without AJAX (Asynchronous JavaScript and XML) that allows requests to be sent to a server using JavaScript. As the number of users increased, polling for changes became expensive due to the growth in the number of requests.

A bi-direction message exchange would be the perfect solution. The Websocket protocol [23] provides full-duplex communication channels allowing webservers and web browsers to push messages to each other. So, instead of having a browser asking the webserver periodically if the currently logged in user has any new email/notification etc., the webserver sends a message to the web browser when there is a new email/notification. Unfortunately, not all web browsers support Websocket and this is the reason why APIs (Application Programming Interfaces) such as Socket.IO exist. In case Websocket is not supported, the API will use any of the available fallback methods like Flash and AJAX long-polling.

2.4.4 Constrained Application Protocol (CoAP)

Resource-constrained devices have their own Web transfer protocol of course. CoAP is based on the REST model because it is lightweight and very common, eliminating the need for a developer to spend time studying a new protocol. Its requirements are limited to 10 KiB of RAM and 100 KiB of code space. [24] CoAP messages can be either confirmable or non-confirmable. There are also ACK and RESET messages. The protocol uses UDP on its transport layer, and Datagram Transport Layer Security (DTLS) to make it secure. DTLS [25] is designed for UDP, based on TCP's TLS protocol and provides equivalent security guarantees.

Although it follows the request-response model of REST, meaning that HTTP methods such as GET and POST can be used, an OBSERVE method is also available as an extension, enabling the clients to keep up-to-date data. The protocol behind OBSERVE [26] is based on the observer design pattern and follows a best-effort approach for sending new data to clients,

providing eventual consistency between the states of server and clients. Another interesting feature implemented is the CoRE Link Format [27] which defines a specific link format (`/.well-known/core`) a client can use to get server's available resources without knowing them a priori.

2.4.5 Message Queuing Telemetry Transport (MQTT)

MQTT is a publish-subscribe messaging transport protocol. It is lightweight, suitable for resource-constrained devices and limited network bandwidth. It is an ISO standard [28] and runs on top of the TCP/IP protocol. The publish-subscribe messaging pattern allows data sources (i.e. any device) to publish messages without the a-priori knowledge of who the recipients are. Data sinks (i.e. any device) are able to subscribe to topics, which are message classes, in order to receive any published message on that topic. This pattern abolishes the need for polling. A message broker is the coordinator that receives all the messages for each topic and delivers the right messages to the right recipients according to their subscriptions.

Quality of Service (QoS) is another feature of the MQTT and it specifies what are the guarantees regarding the delivering of a message. As networks may be unreliable or the application may not be able to tolerate lost messages, one can choose the right QoS level, according to their needs. The QoS levels are three:

- QoS 0: at most once
  It is often called "fire and forget" because it guarantees a best effort delivery. There is no acknowledgment and the message is sent one time. The delivery guaranty is provided by the underlying TCP protocol.
- QoS 1: at least once
  When the broker gets a message, it sends an acknowledgment (PUBACK) to the publisher and the publisher will resend the message in case of timeouts until a PUBACK is received. Of course, there are circumstances where a broker will get the same message multiple times, e.g when the PUBAC is lost. The application has to handle these possible duplicates.
- QoS 2: exactly once
  This level guarantees that each message will be received exactly once. It is the slowest among the other levels because of the multiple message exchange it requires between the broker and the client.

Network problems can also lead to disconnections. MQTT is equipped with persistent sessions which relieves the client from subscribing to topics every time it gets reconnected due to a disconnection. When combined with QoS levels greater than zero, a client may receive messages that would get delivered when it was offline. The Last Will and Testament (LWT) feature comes to enhance further the MQTT reliability. This feature enables each client to have the broker inform all the subscribers of a particular topic that the client was disconnected ungracefully.

As far as security is concerned, MQTT allows the authentication of each client through username-password or unique identifiers, such as MAC address or serial numbers, that can be registered at connection time. All of these are sent in plaintext, requiring that the channel is encrypted. As MQTT uses TCP/IP, TLS can be used for that purpose. Regarding resource-limited devices, where TLS is not a viable option, payload encryption can be used instead. Authorization is also implemented through Access Control Lists (ACL) stating on which topics and what access (publish/subscribe) each client has.

2.4.6 Extensible Messaging and Presence Protocol (XMPP)

Also known as Jabber, the eXtensible Messaging and Presence Protocol (XMPP) is a communications protocol based on eXtensible Markup Language (XML). It was developed for Instant Messaging (IM), presence information and contact list maintenance. Currently, XMPP is an open standard, allowing the implementation in any license. Some of the most used and proprietary messaging platforms, such as Google's Gtalk, Facebook Chat and Skype, are either based on XMPP or provide a limited support in a way to enable message exchange between different services. The address used in XMPP is called jid (Jabber id) and its form is identical to the email's form, referring to a username at a specific XMPP/Jabber server.

The fact that it is based on XML, is one of its strengths as it is extensible and flexible. According to the needs of each application, custom functionality can be added as extensions, making it ideal for gaming, monitoring, Voice over Internet Protocol (VoIP), identity services and IoT. In particular, IoT takes advantage of features like publish/subscribe and authentication, as well as extensions that are custom tailored to handle sensor data or provide services like discovery and provisioning. Unfortunately, the use of XML comes at a cost; it imposes an overhead compared to binary solutions, since it is text based, although there is an extension called Efficient eXtensible Interchange (EXI) Format -which is not limited to XML- enabling an efficient serialization into binary. Another drawback is the binary data transfer, which is very common in IM. There are two approaches: in-band binary data must be encoded using base64 or binary data have to be transmitted out-of-band for better performance.

XMPP servers can be equipped with TLS certificates to ensure that all the incoming and outgoing traffic is encrypted and that a client can verify the server's identity. Through Simple Authentication and Security Layer (SASL), the user authentication can be centrally managed, a useful feature for large organizations and corporations. Moreover, one can install their own server, making sure that their messages are not accessible to other parties.

2.5. Security Protocols

2.5.1 Transport Layer Security (TLS)

This protocol is used on a wide variety of applications that communicate through TCP and provides privacy and data integrity between them. Signing into a portal (e.g. e-banking), a lot of sensitive data have to be transferred. Not only the username and the password but also personal data such as banking accounts and their balances in the case of e-banking, emails in the case of webmail, etc. TLS makes sure that the connection between the client (e.g. browser) and the server is private because the transmitted data are encrypted and no one can figure out the contents of the transmitted packets along the way. Integrity is also ensured (it usually goes hand in hand with encryption) so that no one can tamper the contents of the packets.

Using certificates the identity of the communicating parties can be authenticated. A browser verifies that the URL matches the received certificate and the certificate is still valid, making sure that the remote server is the right one. Certificate Authorities, that issue these certificates, also offer Extended Validation certificates, certifying that the entity named on the certificate is indeed the one the certificate belongs to, making much more difficult to issue bogus certificates. Clients can also use certificates to prove their identity to a server.

TLS supersedes SSL and is more secure. Although known attacks and vulnerabilities exist to SSL and early versions of TLS, it is usually a burden to use the latest version because outdated software may not be able to connect. Forward secrecy is also a feature that TLS provides. This

feature ensures that the disclosure of encryption keys cannot compromise prior communications recorded in the past.

2.5.2 The Advanced Encryption Standard (AES)

In 1997, the National Institute of Standards and Technology (NIST) of the United States of America announced a request for candidate algorithm nominations for the Advanced Encryption Standard, in their effort to replace the existing Data Encryption Standard (DES) [29]. All submissions had to meet the following minimum acceptability criteria:

- Symmetric (secret-key) algorithm
- Block cipher
- Support key sizes of 128, 192 and 256 bits, and a block size of 128 bits.

Numerous additional requirements were imposed, of course [30].

After a three-year worldwide research and a lot of submissions, the Rijndael cipher was selected as the Advanced Encryption Standard. Developed by two Belgian cryptographers, Vincent Rijmen and Joan Daemen (note their last names and the name of the algorithm), the Rijndael is a family of ciphers with different key and block sizes, however NIST decided that only the three members that satisfy the third minimum acceptability criterion (see above) would be included in the standard [31].

AES is fast in both software and hardware because it is based on the substitution-permutation network design principle [32]. The substitution-permutation network uses substitution and permutation boxes (S-boxes, P-boxes) to produce the ciphertext block. The S-boxes define the mapping between the input bits and the output ones whereas the P-boxes shuffle the bits across S-boxes. Although S-boxes may depend on the key, some implementations have them both precomputed because both have to satisfy Shannon's confusion and diffusion properties [33].

Mode of Operation

As a block cipher, AES can only encrypt/decrypt one block -a fixed-length group of bits- at a time. It is obvious that the 128-bit (16 bytes) block size is not enough for every case. This is where modes of operation come into play. A mode of operation describes the way a block cipher can be repeatedly applied securely to inputs larger than the block size. Several modes exist, some of them are vulnerable (ECB), some others provide authenticated encryption[2] and some can be parallelized. The choice depends on the needs of each application and the environment that the data are exposed to.

The Cipher Block Chaining (CBC) mode, which is used in this thesis, was invented by William F. Ehrsam, Carl H. W. Meyer, John L. Smith and Walter L. Tuchman in 1976. Its operation is quite simple: before the encryption, the plaintext of the current block is *XORed* with the resulted ciphertext of the previous block. For the first block, an Initialization Vector (IV) is used, which is a block of random bits. Although the encryption is not parallelizable, the decryption is.

[2] Authenticated encryption ensure that the receiver can verify both that the data have not been tampered with (integrity) and the source of the message (authenticity).

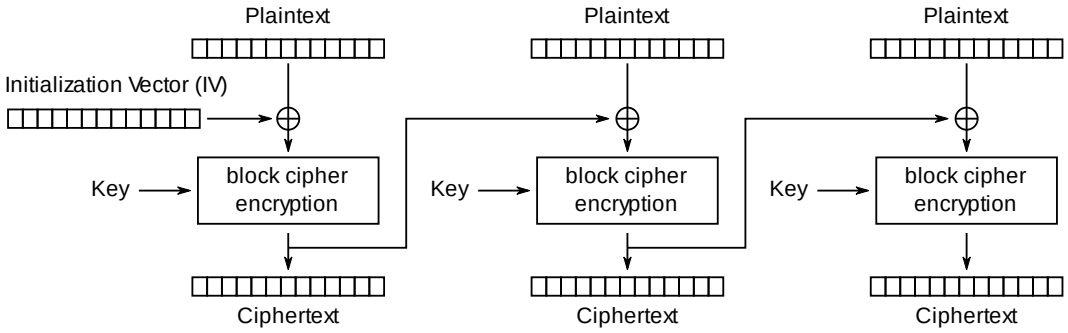


Figure 2.5: CBC encryption diagram[3]

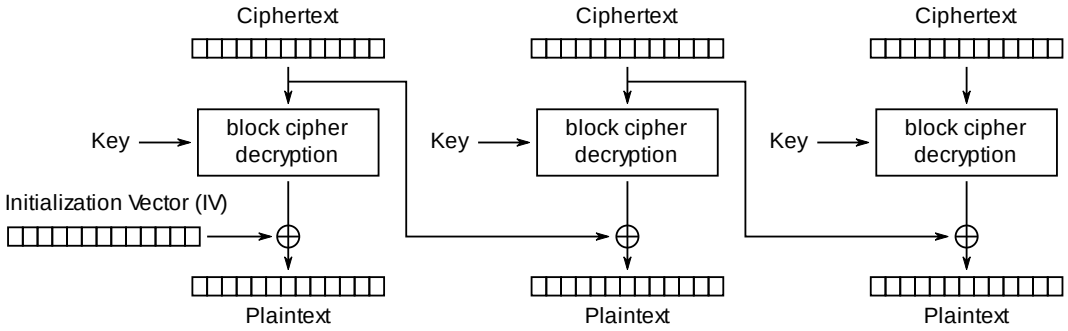


Figure 2.6: CBC decryption diagram[4]

A common implementation mistake is keeping the IV the same. The IV has to be different each time and it is not required to be encrypted (it can be sent in plaintext), or else the same plaintext input will produce the same ciphertext result, enabling an attacker to distinguish between two ciphertexts or worse, to perform a chosen plaintext attack. In such an attack, the adversary can ask for the ciphertext of any plaintext, in a way to reveal the encryption key. Padding oracle attacks can also compromise the ciphertext if the oracle (i.e. the device performing the encryption/decryption) leaks information about whether the padding of a particular ciphertext is correct or not.

Padding

When the message length is not an exact multiple of the block size, padding is used to complete the remaining bytes to reach the block size. There are a variety of padding algorithms, depending on one's needs but both sides should implement the same one. In this thesis, the PKCS#7 [34] has been chosen. The value of each added byte is the number of bytes that are added. For instance, if the block size is 16 bytes and the last block is 10 bytes, the 6 remaining bytes will be filled with the value 6. This works for block sizes that are at least 2 bytes and no

[3] By WhiteTimberwolf (SVG version) - PNG version, Public Domain, https://commons.wikimedia.org/w/index.php?curid=26434096

[4] By WhiteTimberwolf (SVG version) - PNG version, Public Domain, https://commons.wikimedia.org/w/index.php?curid=26434095

more than 255 bytes. In case the message length is an exact multiple of the block size, an additional full-padded block is added so the deciphering algorithm can determine with certainty that there is no chance the last byte of the last block is part of the message and not a padding.

3. Implementation

In this chapter, the software and hardware used will be presented, as well as the setup architecture and what are the implemented features of the IoT Bridge.

3.1. Software

3.1.1 The Contiki Operating System

The Operating System (OS) is an integral part of a device. Especially when the said device has limited resources, the operating system must be designed as light-weight as possible. Contiki is one of the available OSs for such devices focusing on low-power IoT. Despite the fact it needs only about 10 KiB of RAM and 30 KiB of ROM, Contiki provides a TCP/IP stack for both IPv4 and IPv6, a Rime stack, which is a lightweight layered communication stack for low-power wireless sensor networks [35], and multitasking.

Contiki is open source, written in C language and supports a variety of different microprocessors and RF transceivers. One of its distinctive features is its ability to dynamically download program code at run-time, thus enabling updates and bug patching in an operational network, eliminating the need to collect or visit each deployed device. This also means that security updates can also be delivered in time, making harder for an adversary to exploit the device. Contiki's kernel is event-driven, providing concurrency without the need for locking mechanisms or per-thread stacks, which would consume memory. Furthermore, preemptive multithreading is also available as an application library to programs, which can link in case they require it. Preemption is to ensure that incoming events, like sensor input or incoming communication packets, will be handled even though an application may have long running computations. [36]

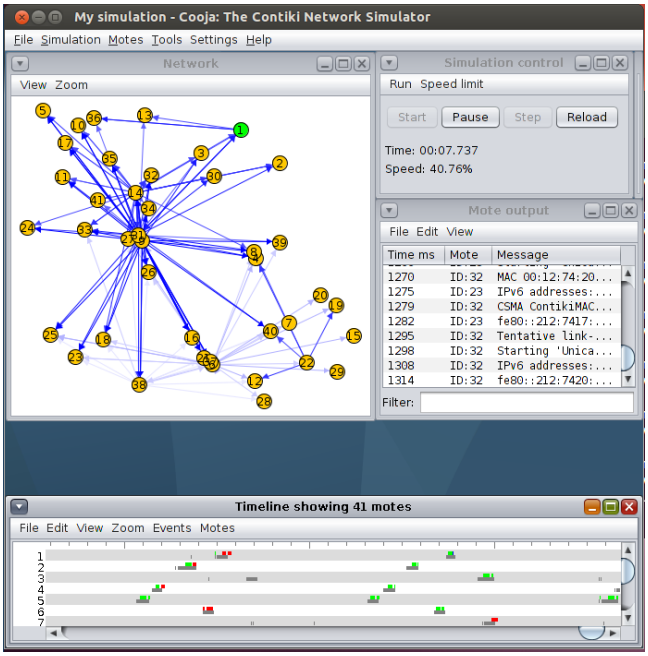


Figure 3.1: Contiki's Cooja simulator[5]

[5] CC-BY-SA-3.0, https://commons.wikimedia.org/wiki/File:Contiki-ipv6-rpl-cooja-simulation.png

Although event-driven programming can reduce the memory overhead, writing, maintaining and debugging such code can be challenging. Protothreads [37] is an abstraction that allows the event-driven programming through a thread-like style. The memory overhead imposed is only two bytes per protothread and the execution time overhead is on the order of a few processor cycles. It would be a notable omission if there was no reference of the Contiki's network simulator called Cooja. Cooja enables the simulation of networks that consist of Contiki nodes. These nodes may be abstract, as well as emulated, meaning that the entire hardware of the node is emulated. For instance, a simulation may include a number of Zolertia Z1 motes, a number of Skymotes etc. Using Cooja, one can test their software with hundreds of motes without the need to buy them, making debugging simpler. Of course, the simulation of the network and the emulation of the hardware may be far from reality, but nonetheless is Cooja offers a place to start.

3.1.2 6LBR

The interconnection of WSNs which is based on IEEE 802.15.4 with an existing IPv6 network (i.e. a LAN) based on Ethernet, requires a border router that acts as an intermediary. 6LBR is a deployment-ready 6LoWPAN/RPL Border Router solution [38]. It is being developed by a research institute in Belgium called CETIC (Centre of Excellence in Information and Communication Technologies) and is responsible for handling traffic between the IPv6 and IEEE 802.15.4 interfaces. As it is based on Contiki OS, it can be deployed on low-cost, open source embedded hardware platforms, like the Raspberry Pi.

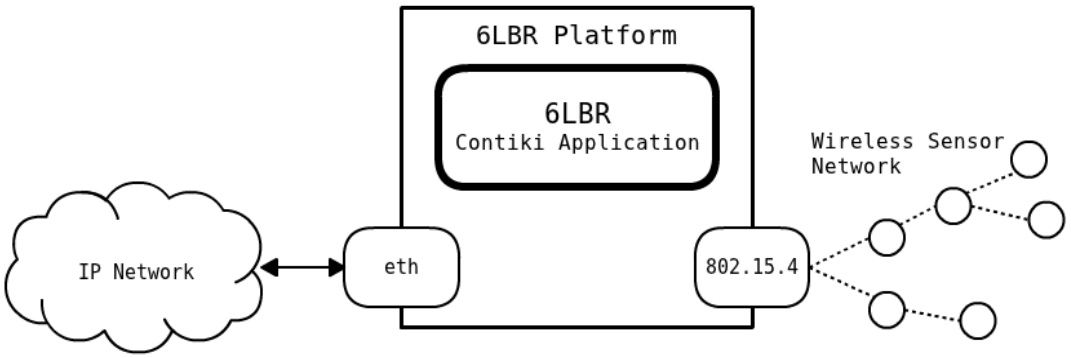


Figure 3.2: 6LBR Platform diagram

6LBR supports a variety of network architectures (modes), allowing the user to integrate it in the current network topology depending on their needs. For instance, one may choose to isolate the WSN nodes to their local network or to assign them a valid public-accessible IPv6 address. Through its webserver, the user can take advantage of the various configuration options, as well as monitor it (e.g. traffic statistics) [39]. It comes with examples that demonstrate its capabilities and its usage. As it is under active development, one may find bugs not only in the development versions but also in the stable ones.

3.2. Hardware

3.2.1 Zolertia Z1

The Z1 mote[6] is, actually, the first commercially available platform by Zolertia [40]. It is a general-purpose low-power development platform for WSNs and supports some of the most employed open source operating systems like TinyOS and Contiki. The network stacks supported include 6LoWPAN, Texas Instruments' SimpliciTI and Z-Stack.

Figure 3.3: Zolertia Z1 board

It is based upon the second generation MSP430 ultra low-power microcontroller and the CC2420 transceiver which is IEEE 802.15.4 compliant and ZigBee ready. Equipped with an antenna, a 52-pin expansion connector and a micro-USB connector for power and debugging, it is ideal for rapid prototyping. [41]

Table 3.1: Zolertia Z1 Specifications

MCU	**MSP430F2617**
CPU	16 bit, 16 MHz RISC
RAM	8 KiB
Flash Memory	92 KiB
Transceiver	**CC2420**
Radio Frequency	2.4 GHz
Effective Data Rate	250 Kibps

Besides the two digital built-in sensors (a programmable accelerometer and a temperature sensor), Z1 can support up to 4 external sensors. The board can be powered in many ways: Battery Pack (2xAA/AAA), Coin Cell (up to 3.6V), USB. [42]

[6] Mote: A tiny computer for remote sensing.

3.2.2 Raspberry Pi

The Raspberry Pi is a single-board computer that can be used as a development platform. Its size is that of a credit card and it is available in many versions, with different specifications each as new ones are coming to production. Apart from its own operating system, Raspbian -which is based on Debian-, it supports various Linux distributions, RISC OS, as well as Windows 10 IoT Core.

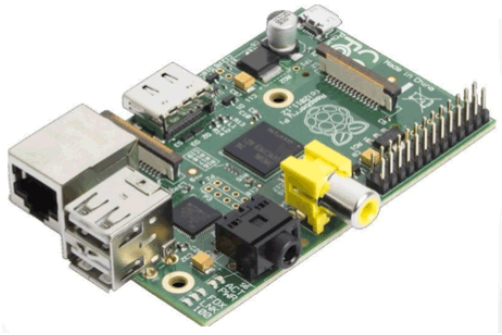


Figure 3.4: Raspberry Pi 1 Model B

Although different versions have different features, all of them have a memory card slot, which acts as a storage device for OS and user data, and a number of General Purpose Input/Output (GPIO) headers. Some of them are equipped with USB ports, an Ethernet port, video and audio outputs. There are also several accessories that can be connected to it such as cameras -via a specific CSI connector-, touch displays, or even expansion boards. Raspberry Pi can be powered by micro-USB or GPIO header.

Table 3.2: Raspberry Pi 1 Model B Specifications

SoC	**Broadcom BCM2835**
Architecture	ARMv6 (32 bit)
CPU	700 MHz single core
SDRAM	256 MiB (GPU shared)
GPU	VideoCore IV 250 MHz

One can find a plethora of projects available on the World Wide Web, which are based on this platform. It was primarily developed to promote computer science in schools [43] but its use in home automation is rather significant. [44] [45]

3.2.3 BeagleBoard-xM

A higher-end development platform is BeagleBoard-xM, with an open hardware design. It is a modified version of BeagleBoard and started shipping on August 2010. Although it is about twice the size of a Raspberry Pi, it is equipped with more Input/Output interfaces. It supports Ångström Linux, Android, XBMC and Ubuntu, but there are other OSs that can be run as well such as RISC OS and FreeBSD.

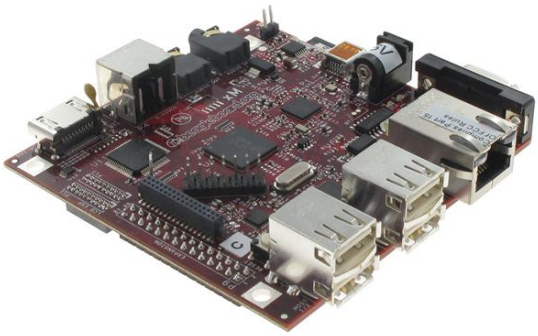


Figure 3.5: BeagleBoard-xM

Among the Input/Output interfaces are DVI-D (with an HDMI connector because of its smaller size), S-Video, USB OTG, 4 USB ports, Ethernet, RS-232, JTAG connector, stereo in/out jacks. There are also a camera port and an expansion port. It can be powered through the dedicated power socket or through the USB OTG connector. Two buttons can be found on the board. One is a reset button and the other's function can be defined by the user.

Table 3.3: BeagleBoard-xM Rev C 1.0 Specifications

SoC	**TI Cortex A8 DM3730**
Architecture	ARMv7 (32 bit)
CPU	1 GHz single core
SDRAM	512 MiB (GPU shared) 200 MHz
GPU	PowerVR SGX530x 200 MHz
DSP	TMS320C64x+ 800MHz

3.2.4 BeagleBone

The BeagleBone was released about a year after the BeagleBoard-xM with a slightly smaller size than Raspberry Pi's. It is the previous version of BeagleBone Black and supports Ångström Linux, Ubuntu and Android. It is equipped with Ethernet, a USB port type A and a USB for power and debugging, eliminating the need for JTAG emulator. A power connector is also available.

Figure 3.6: BeagleBone

A number of expansion boards called "Capes" are available. These can be stacked onto the BeagleBone board (up to four) adding new interfaces like an LCD touchscreen, DVI-D, Breadboard and RS-232.

Table 3.4: BeagleBone Rev A6 Specifications

SoC	**TI Cortex A8 AM3358/9**
Architecture	ARMv7 (32 bit)
CPU	720 MHz single core
SDRAM	256 MiB (GPU shared) 200 MHz
GPU	PowerVR SGX530x 200 MHz

3.3. The IoT Bridge

The IoT Bridge runs an active server for each protocol listening for incoming connections. Besides the MQTT broker that accepts connections based on the MQTT protocol, the currently supported protocols are the following:

- HTTP through a RESTful API
- CoAP
- XMPP

Starting from the HTTP, an HTTP client can connect to an HTTP resource natively or use a reverse proxy if there is a need to drive the traffic through the IoT Bridge. To access a CoAP resource, an HTTP client uses the specific resource for CoAP passing the id of the device to connect to and the path. There is a direct matching between HTTP and CoAP methods, so in order to get a value from a CoAP resource, the HTTP client should use the GET method, as the same method would be used to get the value using the CoAP. There is a specific resource that allows to issue a *DISCOVER* from a HTTP client, because there is no corresponding method in the HTTP. Using websocket, one can take advantage of the CoAP's *OBSERVE* method, although there are some reliability problems due to the limited multithreading capability of the BeagleBoard-xM. An HTTP client can also talk to the MQTT broker and thus communicate with MQTT devices indirectly. To publish a message, there is a specific resource for the MQTT broker that an HTTP client can send the message and the topic it goes to, using the *PUT* method. Websocket will be used in the case of a subscription. Like the MQTT broker, XMPP resources can also be reached the same way, via *PUT* and Websocket, by hitting the corresponding HTTP URL.

The CoAP server exposes the available resources like the HTTP server, because their RESTful logic is very close. There is a specific resource available on the CoAP server for each protocol. One for CoAP, one for HTTP, one for MQTT and one for XMPP. The first two are much alike, as they share the same logic. The third one allows a CoAP client to communicate with the MQTT broker using the *PUT* and OBSERVE CoAP methods. The same is true in the case of XMPP. A CoAP client can of course connect to a CoAP resource (server) natively, or use the IoT Bridge.

As far as the XMPP is concerned, a client that implements this protocol can send a special crafted message to the jid that the IoT Bridge uses to exchange messages. The said message contains the device id, the method (*GET*, *PUT* etc) and the payload. The protocol of the target device is derived from the device id, providing a uniform way to access each device, including the MQTT broker. Another XMPP resource can be accessed directly using its jid.

The MQTT protocol requires a broker to work. Thus, all MQTT clients communicate with the broker. This is the reason an MQTT client is absent from the following table. The IoT Bridge facilitates the message exchange between the broker and the other implemented protocols.

Table 3.5: IoT Bridge Communication Methods Between Protocols

		Servers			
		CoAP	**HTTP (RESTful API)**	**MQTT (broker)**	**XMPP**
Clients	**CoAP**	Natively or special CoAP resource	HTTP resource	*PUT* (publish), *OBSERVE* (subscribe)	*PUT* (publish), *OBSERVE* (subscribe)
	HTTP	*DISCOVER, GET, PUT, POST, DELETE, OBSERVE*	Natively or Reverse Proxy	*PUT* (publish), Websockets (subscribe)	*PUT*, Websockets
	XMPP	*DISCOVER, GET, PUT, POST, DELETE, OBSERVE*		*PUBLISH, SUBSCRIBE*	Natively

The following table contains the main software used during the implementation of this thesis. In the case of 6LBR, more versions (newer and older) were used and tested throughout the development, but due to instabilities or other problems, the version on which the evaluation tests were made is the one mentioned in the table. The Contiki OS is included in the 6LBR and its version at that time was 3.0.

Table 3.6: Versions of Software and Libraries

6LBR	1.4.0 (c09dec4) 24 June 2016
Mosquitto MQTT Broker	1.4.9
Ejabberd XMPP Server	17.01
MSP430-gcc compiler	4.7.0 0120322 (mspgcc dev 20120911)
SleekXMPP python library	(3898b01) 18 August 2015
Paho MQTT client for python	1.2 (377dad6) 3 June 2016
CoAPthon python library	4.0 (1749b26) 23 May 2016

3.4. Setup Architecture

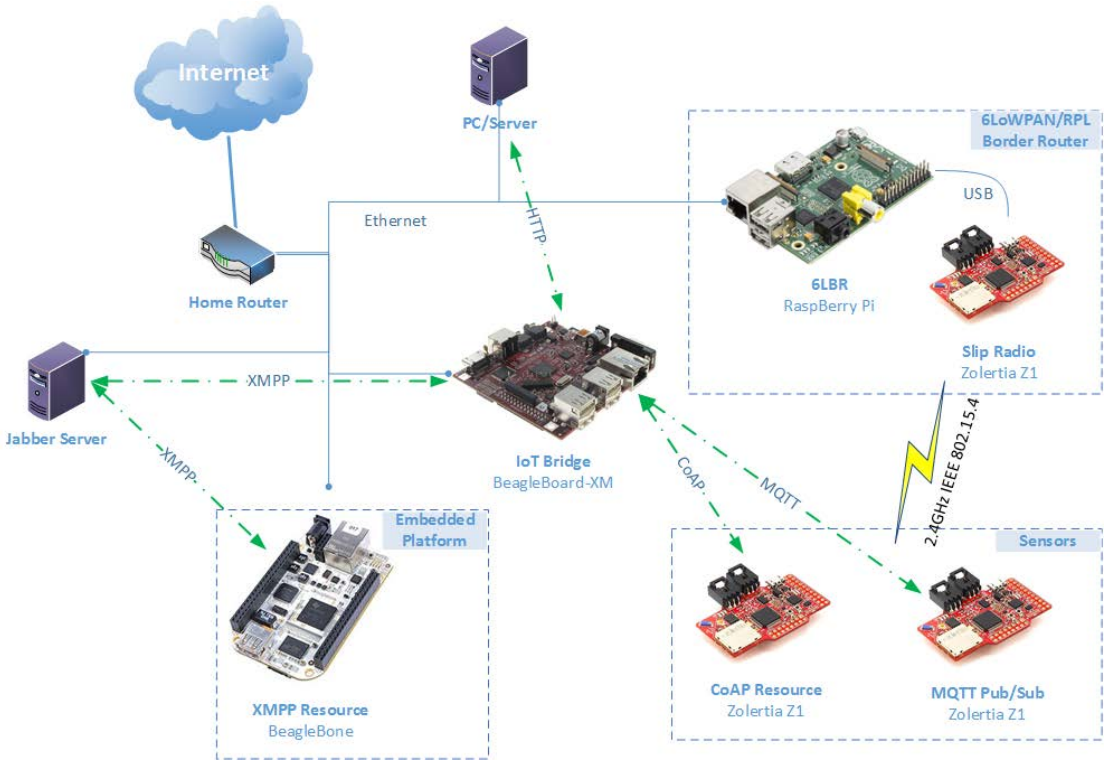


Figure 3.7: Setup Architecture

As can be seen in the Figure above, the setup is composed by different devices. Like in the IoT world, where the devices are heterogeneous and the protocols are plenty, this setup is not far from reality. Although the major parts are in the same network, there is nothing that restricts any of the devices to be put outside of it, far away from others. The IoT Bridge could have been in the datacentre of a company, serving the very same functions to company's customers.

The heart of the setup is the IoT Bridge. It enables the exchange of information between devices that use different protocols. Any device can connect to the IoT Bridge and request a resource from another device. The IoT Bridge will then translate the specific request to the recipient's protocol and make that request on behalf of the sender. When a response is received, it is being translated and sent back to the sender who requested the resource in the first place. To give a specific example, let's assume that a PC/Server needs the temperature measured by a CoAP device. Firstly, the PC/Server makes an HTTP request to the IoT Bridge through its RESTful API. The parameters of this request are the address of the CoAP device, what is the path of the resource (e.g. sensors/temperature) and the action (e.g. *GET*). The IoT Bridge looks up the address of the CoAP device to determine what protocol the recipient supports. Based on the parameters, the IoT Bridge sends the request to the CoAP device and gets its temperature. Then, it responds to the PC/Server with the actual temperature it got from the CoAP device.

Furthermore, the IoT Bridge can encrypt the payload using a pre-defined key that can be different between devices. This provides encryption in cases where either the protocols used or the devices themselves do not support any other type of security mechanisms and also offers separation between devices to ensure that a malicious device cannot capture what is being transmitted by other devices.

On the same device (BeagleBoard-xM), the MQTT Broker named Mosquitto is running. That is not the case for the Jabber-XMPP server, which is installed in a PC on the same network.

An XMPP server outside the local network could be used but that would lead to increased round-trip-times (RTT) during the evaluation phase and thus making a comparison impossible. The XMPP could have been installed on the BeagleBone-xM, but the PC was more appealing because of the limited resources of the device on which both the IoT Bridge and the MQTT Broker run. A Jabber/XMPP Server requires a Relational DataBase Management System (RDBMS) to work, which imposes more Input/Output load to the storage and the usage of RAM. Taking also into consideration the fact that the MQTT Broker needs storage too, in order to hold all the information required for its function (e.g. topics, messages, subscribers, etc.), the installation of a Jabber/XMPP Server would certainly cripple the performance.

There are two Zolertia Z1 motes that act as sensors, one that runs a CoAP server which means it can receive requests for its resources (e.g. reading a sensor's value, turning on/off a LED) and one that runs MQTT code with the ability to publish (e.g. a sensor's value) and subscribe (e.g. to a topic which handles its LEDs). These motes communicate wirelessly via IEEE 802.15.4 with the sink mote called Slip Radio. The latter is also a Zolertia Z1 and is a vital part of the 6LBR. All three Zolertia Z1 motes run Contiki OS.

The 6LBR is the boarder router for the motes' network and it is consisted of two devices: a Raspberry Pi on which the software runs and a Zolertia Z1 (Slip Radio) which enables 6LBR to communicate via IEEE 802.15.4. The 6LBR is configured to serve IPv6 addresses to Z1 sensors that belong to Home Router's subnet, making them accessible on the local network.

An embedded platform cannot be missing from an IoT world. A BeagleBone acts as an XMPP Resource. The communication between the IoT Bridge and the XMPP Resource is handled by the local Jabber server as mentioned earlier.

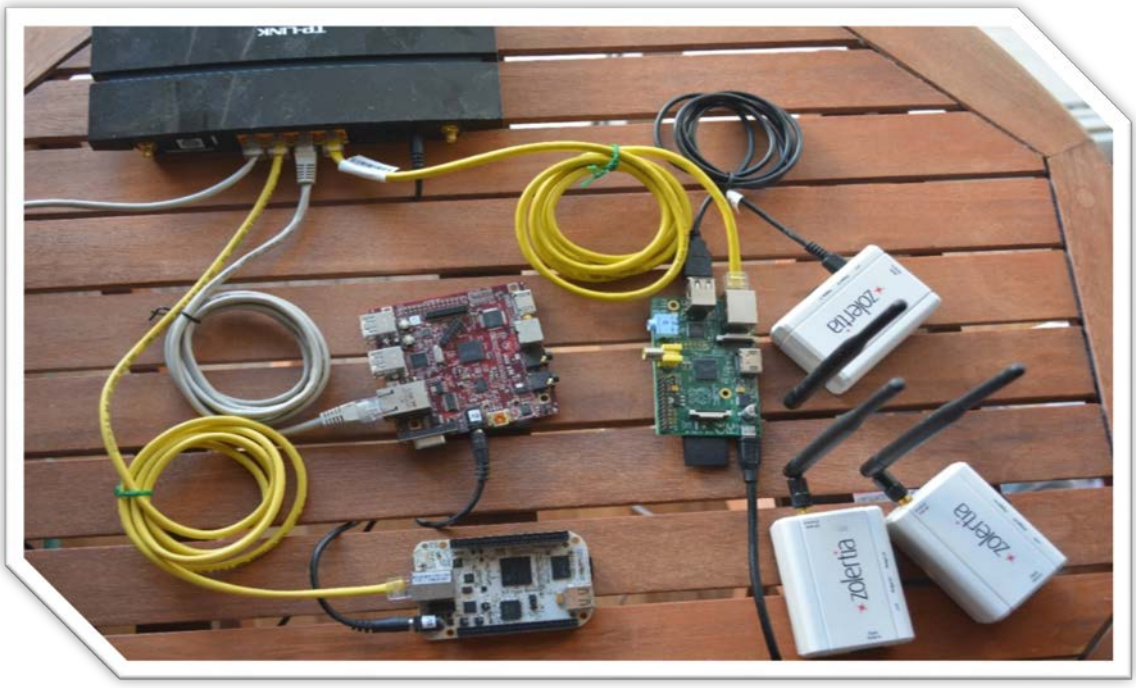


Figure 3.8: A Real Image of the Setup Architecture

4. Evaluation

This chapter is devoted to the performance evaluation of the IoT Bridge and the various security options available. More specifically, what load is imposed on the BeagleBoard-xM on which the IoT Bridge runs, but also on the devices that offer a specific resource. Detailed charts for each device can be found in *Annex A*.

Table 4.1: Cross-Protocol Bridging – Interaction Scenarios

Scenario 1	HTTP – CoAP
Scenario 2	HTTP – MQTT
Scenario 3	HTTP – XMPP
Scenario 4	MQTT – CoAP
Scenario 5	XMPP – MQTT
Scenario 6	XMPP – CoAP

The IoT Bridge supports a variety of protocols, as discussed in the previous section, but the number of possible combinations between protocols and their implemented functions/methods is quite large. Therefore, the performance evaluation was conducted with six different representative scenarios. In each scenario twenty consecutive requests/responses were sent, depending on the nature of each scenario. The interval between the requests was one second in order to examine how the whole setup respond to such a heavy load. In the first three scenarios, three different payload sizes were tested, whereas in the last three scenarios the payload size was fixed but an additional interval of ten seconds was tested. The payload sizes were chosen with the limitations of all devices in mind. A restriction in the maximum allowed of 64 bytes in the IP buffer of the RPL Border Router, without modifying its code, is imposed, meaning that the AES-CBC encrypted message can be at most 16 bytes long, because the Initialization Vector (16 bytes) is also sent together with the ciphertext and these 32 bytes are converted to hexadecimal values in order to be sent and handled, doubling the size to 64 bytes. The payload sizes were 1, 8 and 16 bytes and a padding according to PKCS #7.

Starting from the IoT Bridge, the measurements included CPU utilization, RAM utilization in mebibytes, bandwidth used and power consumption. With the assistance of the *psutil* python library, the measurements regarding CPU and RAM were limited to the actual process of the IoT Bridge. That is not the case for the bandwidth that was based on the interface statistics. The power consumption was measured by utilizing the internal Analog-to-Digital Converter (ADC) paying the price with quantized results due to the low resolution of the ADC. The Round-Trip Time (RTT) was also measured and will be discussed on each scenario.

Unfortunately, the power consumption could not be measured using the ADC on the Beaglebone on which the XMPP resource was run. The alternative was to measure the voltage with a multimeter but there was a high risk of short-circuiting and destroying the board. The same holds true in the case of BeagleBoard-xM. Moreover, such rapid measurements (1 second) would require a digital multimeter that can log to a file. CPU and RAM utilization as well as bandwidth are measured the same way as on BeagleBoard-xM.

As far as the measurements on the Zolertia Z1 platform are concerned, besides the power consumption on which the following chapter is devoted to, there was also measurement of the Round-Trip Time in the case of MQTT publisher when the mote initiated the whole message exchange. For that purpose, the QoS 1 was used because the RTT between the mote and the

broker could be measured by measuring the time elapsed between the publish event and the moment the PUBACK is received.

Threats to Validity

Although the whole procedure of measurements was handled with care, the experimental results were anyway prone to error, due to both internal and external validity threats. As far as the internal validity threats are concerned, the software libraries that were chosen for the implementation of the IoT Bridge, as well as, the conduction of measurements, contribute to the results up to a point. Selecting different software libraries, different programming languages (where possible) and changing the implementation to a more efficient one, will certainly lead to a more efficient result. Furthermore, the protocols, software and libraries utilized were used with default configuration, meaning that there was no attempt to select an optimal configuration, as this require experience.

The external validity threats include the equipment, hardware. The devices themselves are also limiting the efficiency of the whole procedure. Whilst resource-constrained devices have their own purpose (to be energy efficient), the IoT Bridge could also run on a typical computer. Even though the measurements were contacted in a Local-Area Network (LAN), the network equipment, such as router, switches, etc. also impose a threat to validity. Having other devices using the network simultaneously may have a negative impact on the results, but in a real use case, networks will probably be under load as well.

4.1. Evaluation Methodology

4.1.1 Zolertia Z1 Power Specifications

Designing a low-power resource-constrained device has a major goal: to reduce the power consumption as much as possible. The challenge for a developer is to measure the power consumption when their program runs on the said devices. Fortunately, Contiki OS is equipped with a software tool called Powertrace that estimates the power consumption and to what degree each system activity (e.g. packet transmissions) has contributed to. Per experiments conducted by its creators, Powertrace is accurate to 94% of the device's energy consumption. [46] Powertrace utilizes high-accuracy timers (rtimers) and another software tool called Energest to calculate an average power consumption over the CPU ticks. Energest counts the number of rtimer ticks spent in each power state, such as radio transmission/receive, high/low-power CPU, using macros that handle overflows on signed integers (wraparound). The measurements are printed to the serial output of the device either periodically or at specific points on the program. Developers are free to choose which is suitable for their needs or use both.

Unfortunately, the output is rather cryptic as it mainly consists of numbers separated by space. This makes it also easier to handle with a script that consumes the data to process them. A look to the source code can be revealing as to what each number represents, but the actual meaning of each and how can be used in the calculation of the power consumption can be difficult to figure out.

```
279  P  2.0  0  14606 55658 4816 924  0   299 14606 55658 4816 924   0  299  (radio 8.16% / 8.16% tx 6.85% / 6.85% listen 1.31% / 1.31%)
```

Figure 4.1: Powertrace output example

Figure 4.1 shows a typical Powertrace output. The parenthesis at the end shows the utilization of each state of the radio throughout the sampling interval, and it is not used in the

calculation of the power consumption. Zooming in to the elements before the parenthesis, the Figure 4.2 matches each element with its identity as per the source code.

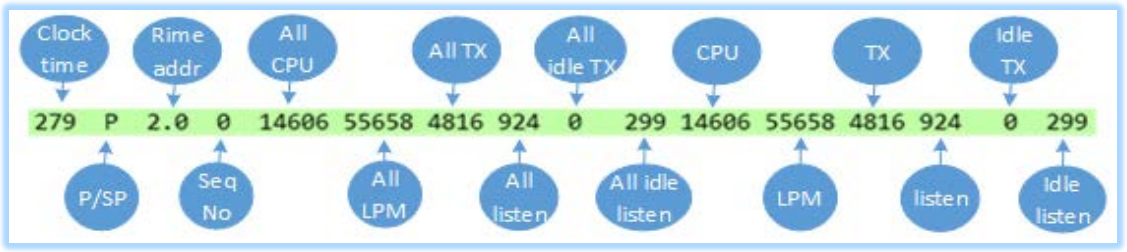


Figure 4.2: Powertrace output identities

The first number is the current system time in clock ticks as returned by the function *clock_time()*. When the device is powered on, the timer starts counting ticks in a way to hold the system's uptime. The number of ticks per second is platform dependent and the constant CLOCK_SECOND defines that number. In the case of Zolertia Z1, each second is 128 ticks. The second element is a tag (either P or SP) which indicates the types of the values Powertrace prints, because it allows both inspection of node-level energy behaviour (P) and of network-level protocol power profiles (SP). [46] The third element is the first two parts (separated by a dot) of the node's rime address, followed by a sequence number that increments on each Powertrace output. All the following numbers are rtimer ticks consumed by each node's function. The elements starting with "All" are aggregated values measured from the time the device was powered on. The elements without "All" is the difference between the aggregated values among two consecutive outputs. As the example implies, this is the first output (the sequence number is zero), so the aggregated values are equal to the non-aggregated ones (e.g. All CPU and CPU).

Rtimer is a library that uses its own clock module in order to achieve higher clock resolution and can be used for scheduling and execution of real-time tasks. The constant RTIMER_ARCH_SECOND defines the number of ticks per second which are 32768 in the case of Zolertia Z1. A cautious reader may have already spotted a latent problem. Although the numbers Powertrace outputs are in ticks, the clock time is measured by a different clock. The calculation of the energy of each part can be done using the following formula:

$$Energy\ [mW] = \frac{EnergestValue\ [ticks] \cdot current\ [mA] \cdot voltage\ [V]}{RTIMER\_ARCH\_SECOND\ [ticks/s]\ \cdot TimeDuration\ [s]} \quad (1)$$

For the sake of completeness, the formula below can be used to calculate the radio duty cycle, although it will not be used in the following measurements because of the inability to use the proper radio duty cycling mechanism in order to reduce the time of activity of the RF transceiver and thus its power consumption.

$$Radio\ Duty\ Cycle\ [\%] = \frac{TX + Listen}{CPU + LMP} \quad (2)$$

All values correspond to the print out of the Powertrace.

The formula for the calculation of the energy requires a value for the current and the voltage. By consulting the datasheet of Zolertia Z1 [41], one can find the table below with the Integrated Circuits (IC) and their current consumption.

Table 4.2: Approximate Current Consumption of Z1 circuits

IC	Operating Range	Current Consumption	Notes
MSP430f2617	1.8 V to 3.6 V	0.1 μA	OFF Mode
		0.5 μA	Standby Mode
		0.5 mA	Active Mode @ 1 MHz
		< 10 mA	Active Mode @ 16 MHz
CC2420	2.1 V to 3.6 V	< 1 μA	OFF Mode
		20 μA	Power Down
		426 μA	IDLE Mode
		18.8 mA	RX Mode
		17.4 mA	TX Mode @ 0dBm
ADXL345	1.8 V to 3.6 V	0.1 μA	Standby
		40 μA to 145 μA	Active Mode
M25P16	2.7 V to 3.6 V	1 μA	Deep Power Down
		4 mA to 15 mA	Active Mode
TMP102	1.4 V to 3.6 V	1 μA	Shutdown Mode
		15 μA	Active Mode

The first IC is the microcontroller, the second one is the RF transceiver, the third one is the onboard accelerometer, the forth one is the flash memory and the last one is the onboard temperature sensor. According to the datasheet, the nominal voltage of the Z1 is 3 V and that will be the value to be used in the calculation of the formula. As far as the current is concerned, Contiki is running at 8 MHz which is not mentioned on the table above. The datasheet of the microcontroller [47] states that the current consumption in Active Mode at 1 MHz is 515 μA when the program executes in the flash memory and 460 μA when the program executes in RAM. A normal operating free-air temperature is considered and a supply voltage of 3 V. The following figure from the datasheet reveals the linear function of the current and the frequency:

$$I\,[\mu A] = I[\mu A] \cdot f(DCO)[MHz]$$

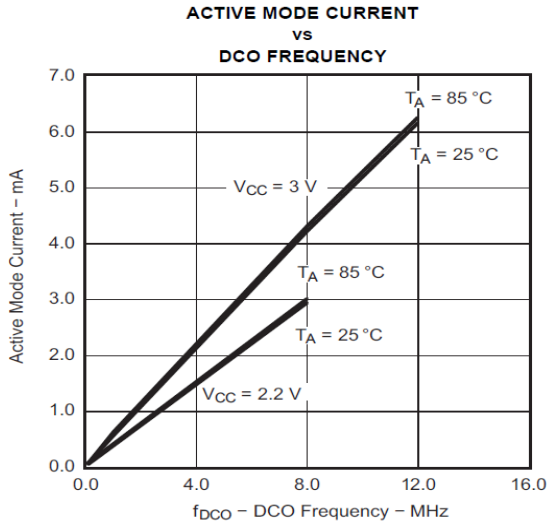


Figure 4.3: Active Mode Current vs DCO Frequency on MSP430

As the program executes in the flash memory, the current consumption in our case is

$$515\mu A \cdot 8MHz = 4.12mA$$

Because the maximum current consumption at 8 MHz is 4.48 mA, the average of the two values will be used: 4.3mA. So, the energy calculation formula will become:

$$Energy\ [mW] = \frac{EnergestValue\ [ticks] \cdot 4.3\ mA \cdot 3\ V}{32768\ \cdot \frac{t_1 - t_0}{128}}$$

where $t_1 - t_0$ is the difference of clock ticks between two consecutive printouts.

4.1.2 BeagleBoard-xM Power Consumption using ADC

In order to measure the power consumption of this board, one have to initialise the ADC first using the following commands.

```
i2cset -y -f 1 0x4a 0x00 0x01
i2cset -y -f 1 0x48 0xbb 0x08
i2cset -y -f 1 0x4a 0x06 0x28
i2cset -y -f 1 0x4a 0x07 0x00
i2cset -y -f 1 0x4a 0x08 0x28
i2cset -y -f 1 0x4a 0x09 0x00
```

The first two commands turn the MADC on and connect the ADC pins. The last four commands set which ADCs to read and average. These commands need to be executed each time the board is powered on. To make a measurement at a specific timepoint, the following commands have to be executed.

```
i2cset -y -f 1 0x4a 0x12 0x20
i2cget -y -f 1 0x4a 0x3d w
i2cget -y -f 1 0x4a 0x41 w
```

The measurement is conducted with the first command, whereas the two last return the voltage values in hexadecimal. The power consumption of the board is the absolute difference between these two values scaled by a factor a, as explained below.

$$Energy\ [W] = a \cdot |v_1 - v_2|$$

$$a = \frac{5V \cdot 2.17 \cdot 2.5V}{64 \cdot 1024 \cdot 0.1\Omega}$$

The two measured values are the voltage at the ends of a resistor, so that the power consumption can be calculated by the drop of the current on that resistor. The resistor is 0.1Ω and the input voltage is about 5V. The resolution of the ADC is 10-bits (1024 on the denominator) and its dynamic range is 2.5V. According to the instructions manual, the voltage on that resistor is the 46% of the actual voltage, so a multiplication by a factor of 2.17 is needed.

4.2. Scenario 1: HTTP - CoAP

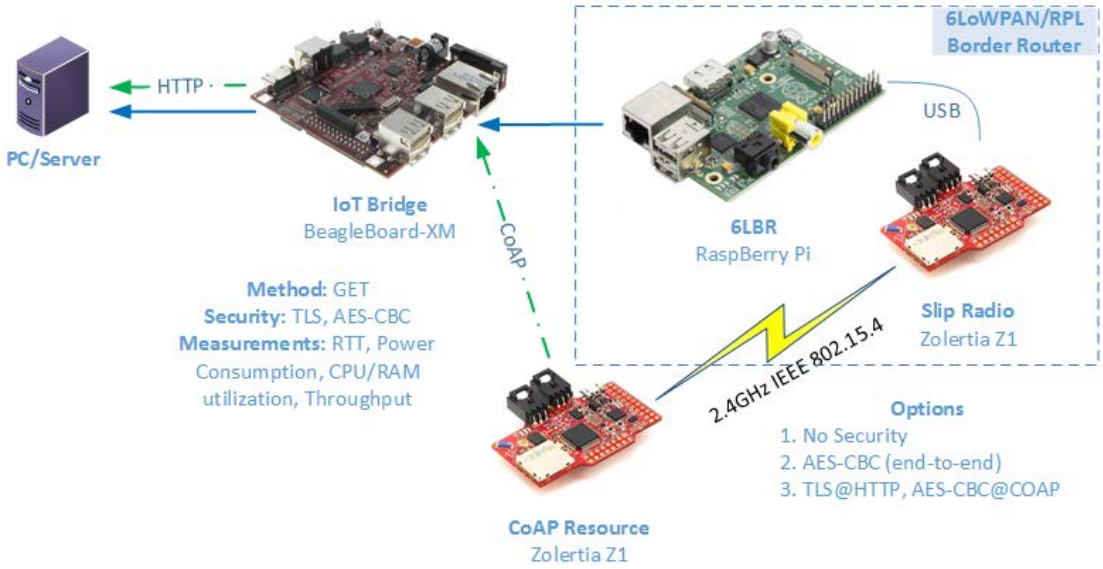


Figure 4.4: Scenario 1: HTTP - CoAP

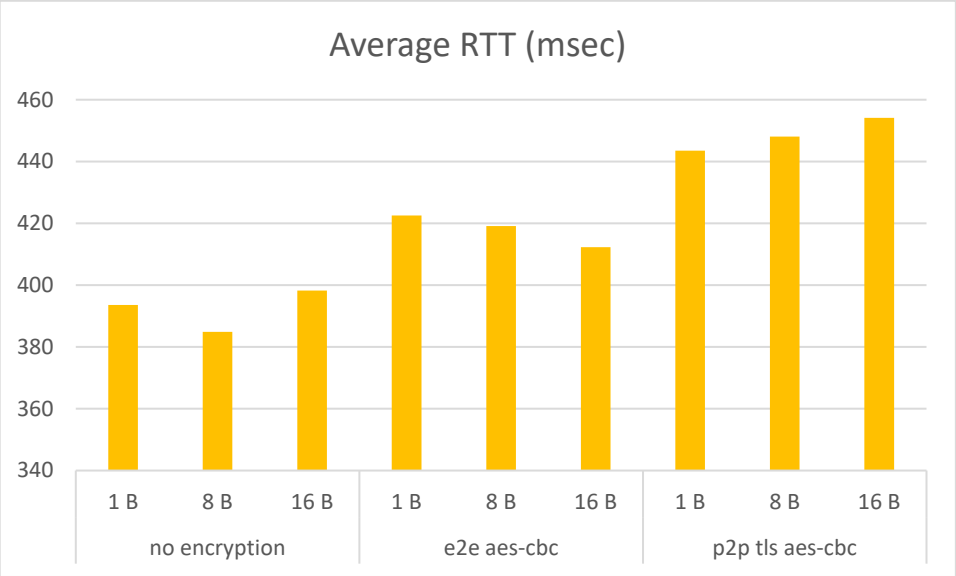
One of the most common setups where sensors are used is monitoring and information gathering through a PC or a server. Sensors record values which are being sent to a server for further analysis. This scenario, as well as the following two, demonstrates how the IoT Bridge assists in the interconnection between sensors and servers by implementing and translating between different protocols. In this case, there is a PC/Server which utilizes HTTP and a sensor mote which utilizes CoAP. Without the IoT Bridge, both devices would have to implement the same protocol (either HTTP or CoAP) in order to communicate.

Plot

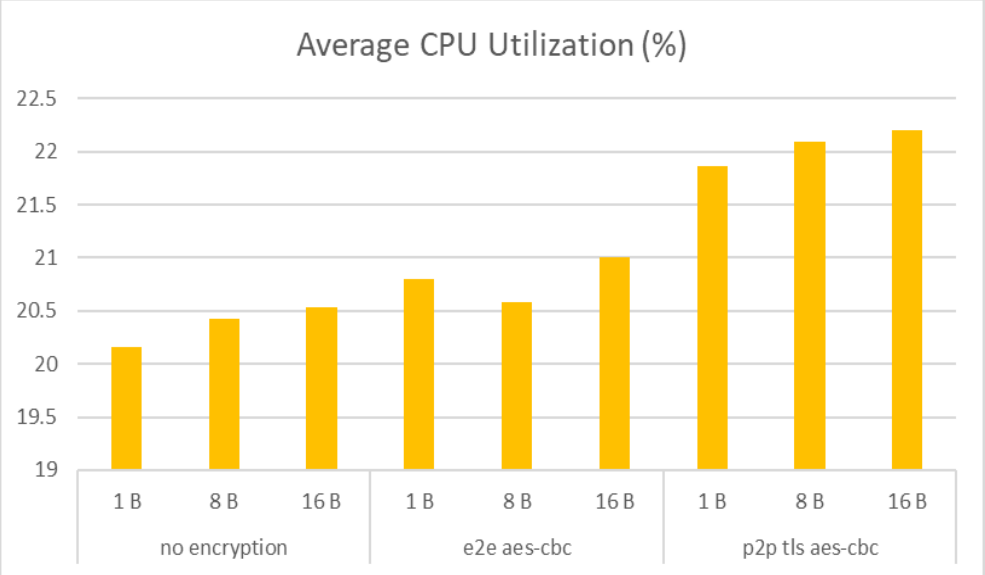
A computer sends an HTTP *GET* request to the IoT Bridge with parameters the id of the CoAP device and the path of the wanted resource. The IoT Bridge, on its turn, sends a CoAP *GET* request on the corresponding CoAP device, the device responds and the IoT Bridge returns the response to the computer.

Evaluation

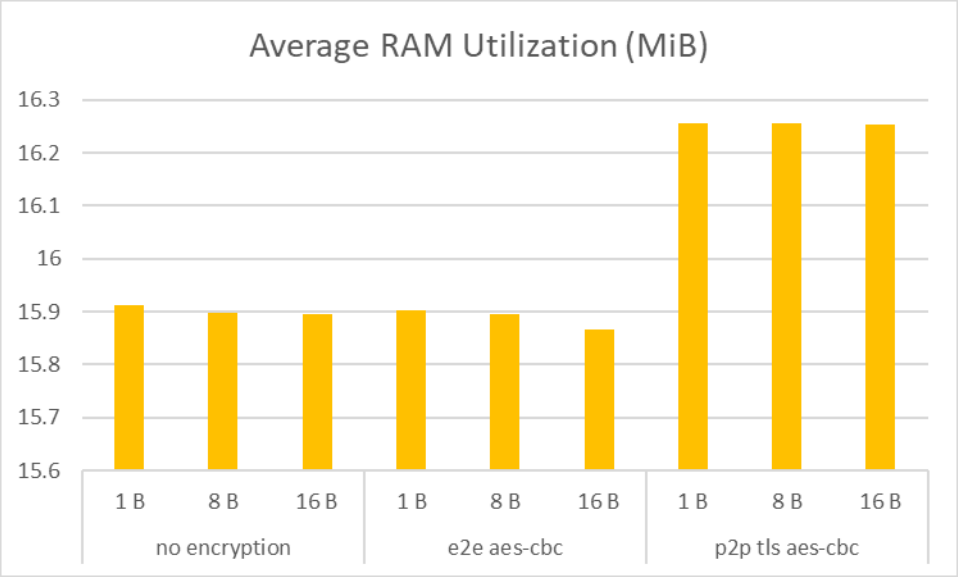
With an interval of 1 second between consecutive requests, the evaluation includes different modes of encryption and different payload sizes as described previously.



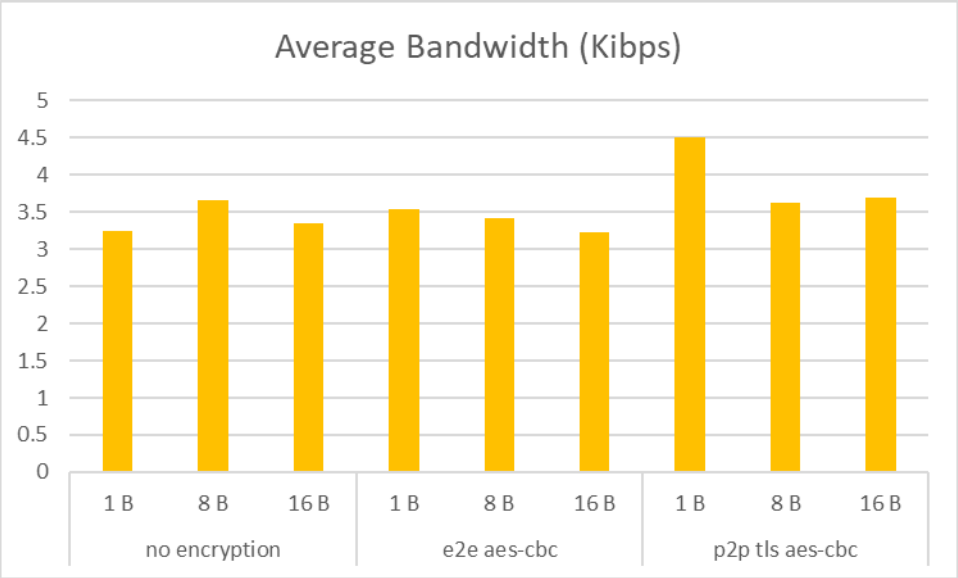
The Figure above depicts the average RTT of 20 consecutive requests. The RTT is the time elapsed from the moment the client sends a request to the IoT Bridge until the former receives the response from the IoT Bridge. It is evident that the more complex the encryption scheme is, the more time it requires. As far as the difference in RTT between different payload sizes is concerned, one can attribute it to the instability of the network, although the differences are not significant. Padding required to fill the block prior the encryption may also be a contributing factor.



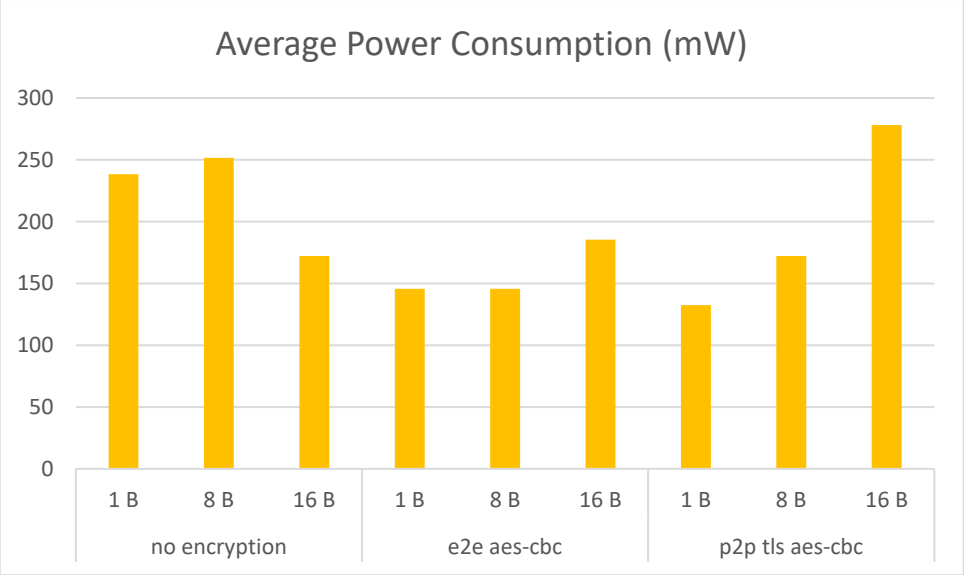
The CPU utilization is greater when the IoT Bridge is involved in the encryption scheme. That is, in the case of point-to-point encryption where the TLS is used in the communication using the RESTful API (HTTP).



As expected, like the CPU utilization, RAM utilization follows the same pattern regarding the TLS encryption. To the IoT Bridge, there is almost no difference between no encryption and end-to-end encryption, because it does not encrypt/decrypt the payload, but rather forwards it. In the case of point-to-point, TLS utilizes memory in order to encrypt/decrypt the traffic.



As shown in the Figure above, the variation of bandwidth is quite small, between different encryption modes and payload sizes. It may also be attributed to other processes running concurrently using the network.



The ADC's inaccuracy is quite obvious.

4.3. Scenario 2: HTTP - MQTT

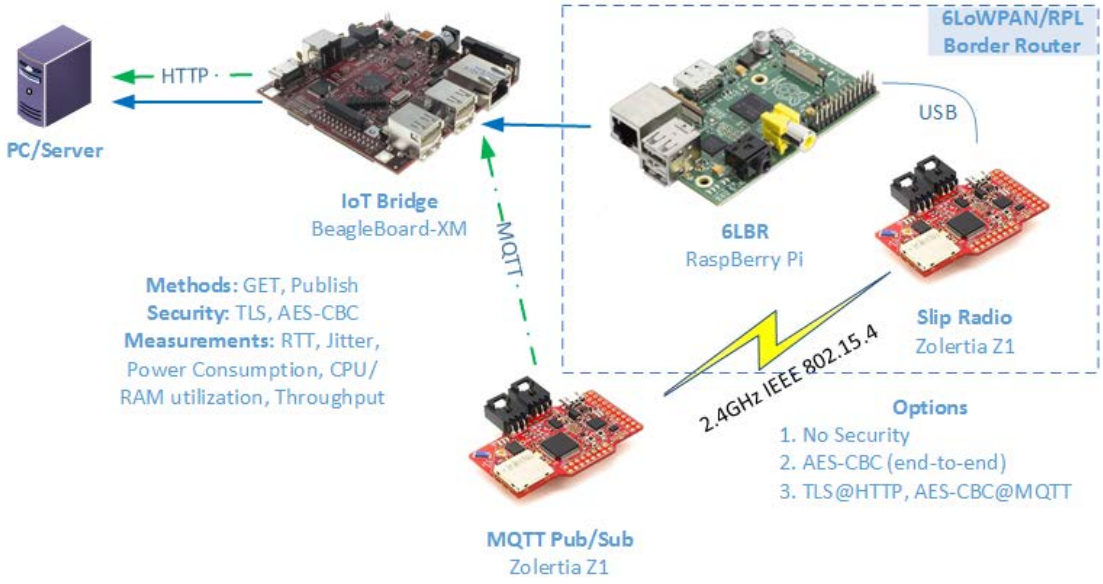


Figure 4.5: Scenario 2: HTTP - MQTT

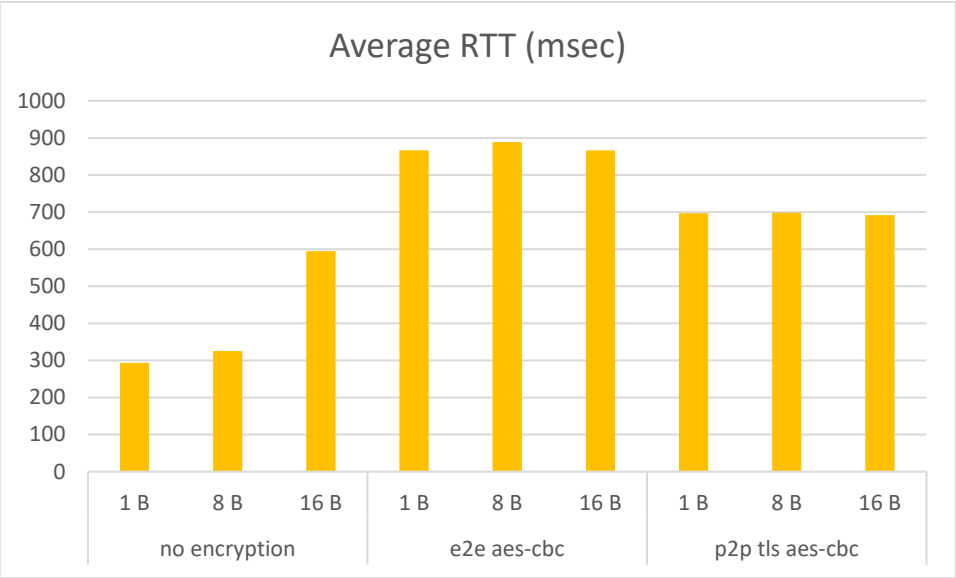
Polling can be ineffective and expensive. There is no point in capturing values when one may be only interested in getting notified when the measured values change. This cuts down the network traffic and the requests that hit the PC/Server. MQTT is an ideal protocol for this job. In this scenario, one can visit a website that displays the measured values each time a new value is published by the MQTT mote.

Plot

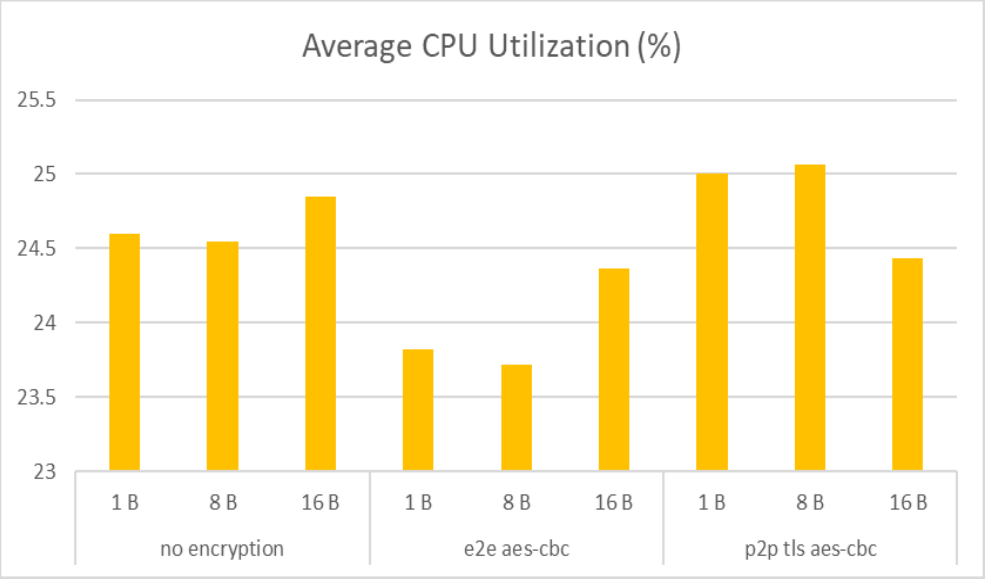
A computer requests a webpage from the IoT Bridge containing JavaScript code that opens a Websocket between the browser running on the computer and the IoT Bridge. The request sent for the Websocket to be established, includes the topic on which it wishes to subscribe and

the subscription is forwarded by the IoT Bridge to the MQTT broker. The MQTT device publishes a message periodically to a specific and the MQTT broker is responsible for forwarding this message to subscribers of that topic, meaning that the IoT Bridge receives each message and sends it to the computer through the open Websocket connection.

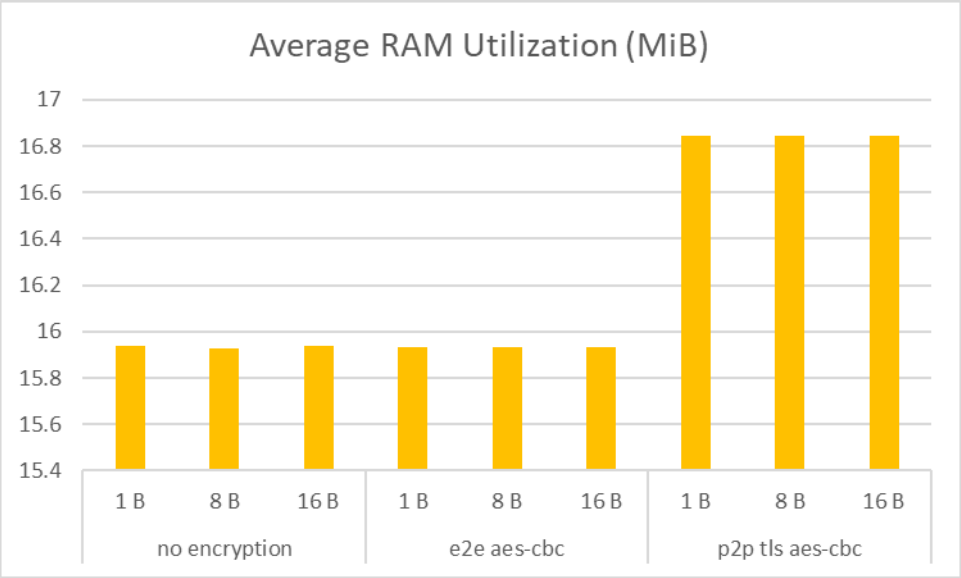
Evaluation



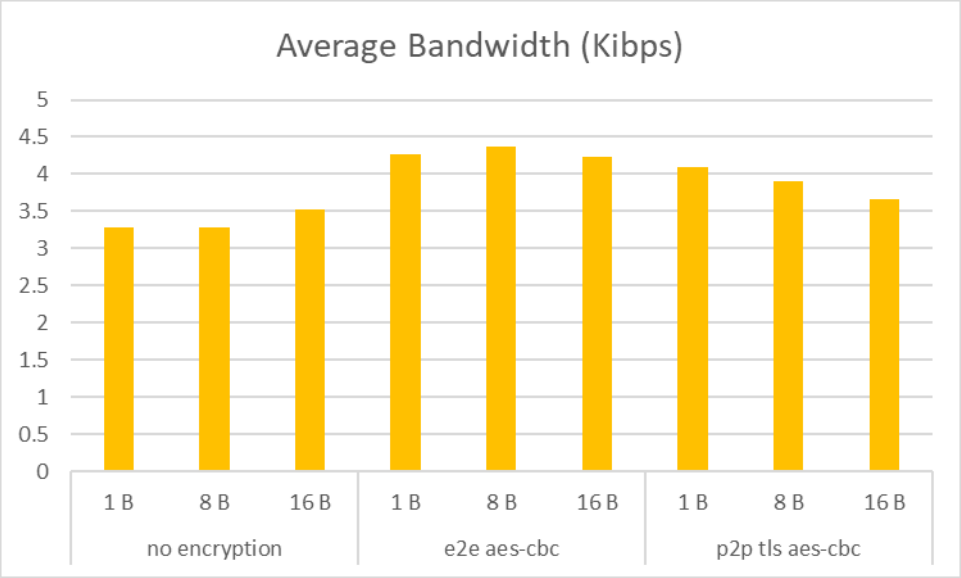
The average RTT is increased when the Z1 mote is required to encrypt the payload prior to publishing it. Moreover, the asynchronous nature of this scenario makes it hard to measure the RTT with precision, because there is no common clock among the devices.



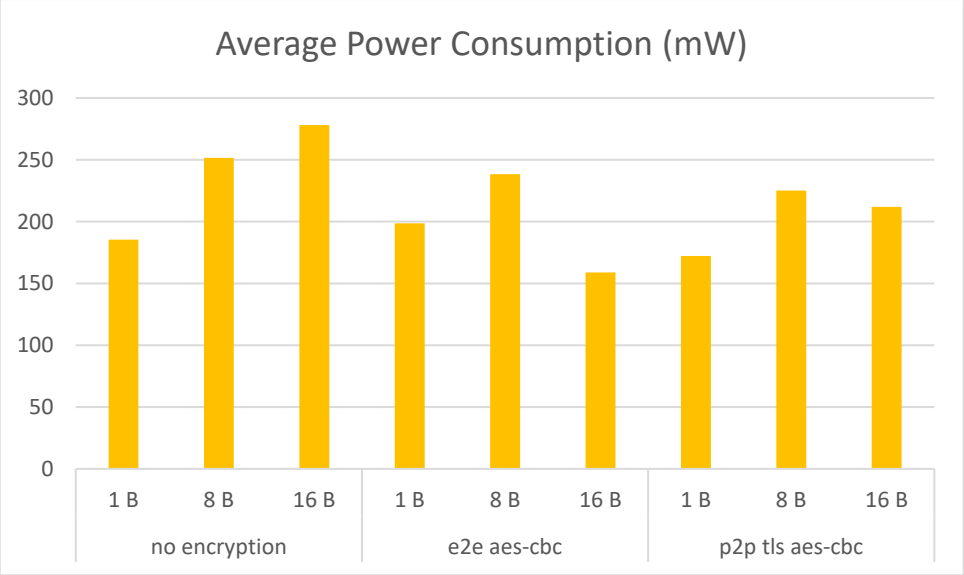
The utilization of the CPU is quite the same. The variation can be attributed to the time each measurement was taken and whether different services, like the Broker, were active at that particular time.



It is evident from the Figure above, that p2p utilizes more RAM due to the fact that this is the only mode in which the IoT Bridge encrypts/decrypts the messages.



There is no significant variation between the bandwidth values. The fact that the differences are visible can be attributed to the difficulty of synchronizing measurements in an asynchronous scenario, meaning that the bandwidth used by broker may have been included too.



4.4. Scenario 3: HTTP - XMPP

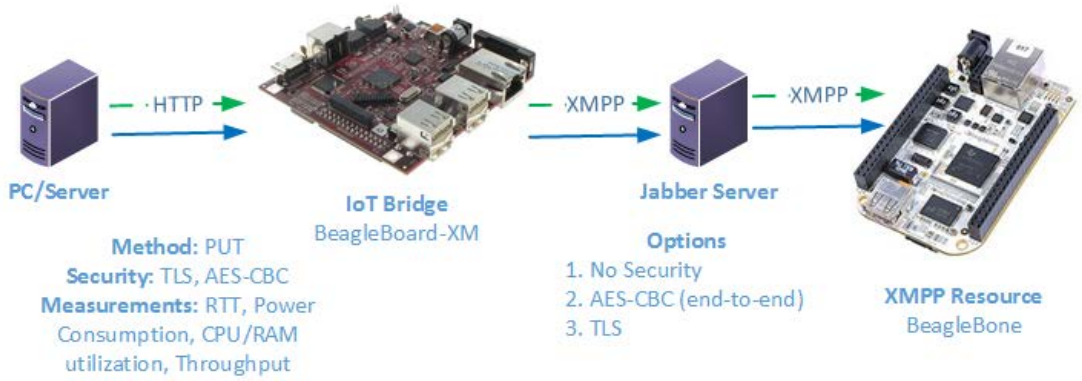


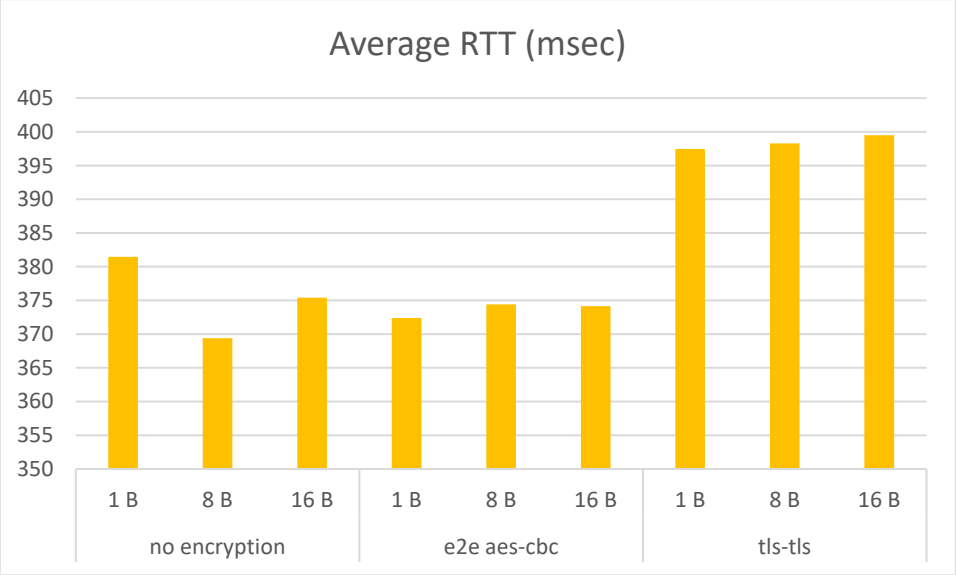
Figure 4.6: Scenario 3: HTTP - XMPP

XMPP is another asynchronous messaging protocol. Although, sensors may implement this protocol, there is another way this protocol can be exploited. Nowadays, a plethora of team collaboration tools and services has emerged, which allows team members to communicate effectively. Their features can be expanded using integrations with numerous services, one of which may be an alerting system that sends messages to team members when something happens. In this scenario, a PC/Server that captures and analyses data, sends a message to an XMPP resource, a resource that could be an integration to the said collaboration tools.

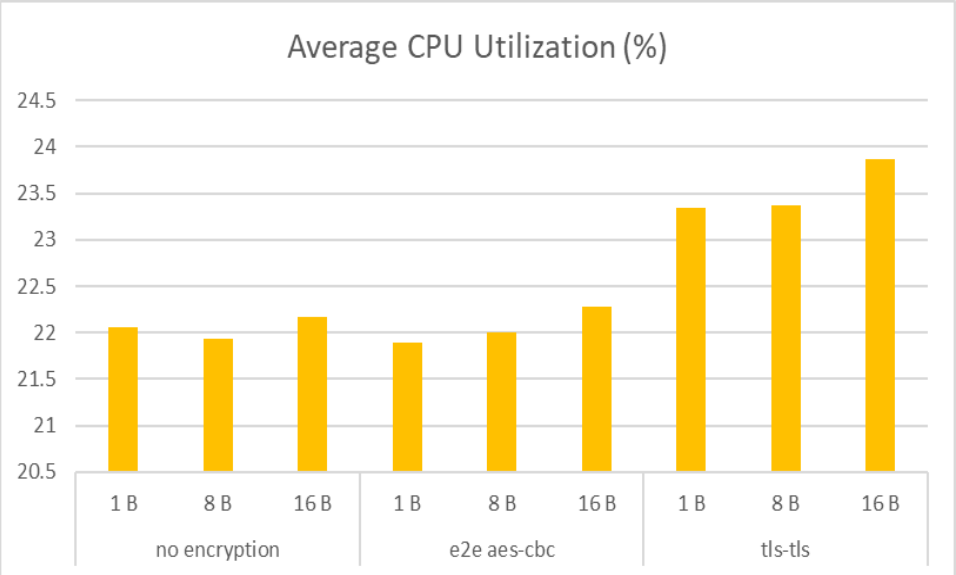
Plot

A computer issues an HTTP *PUT* request to the IoT Bridge with parameters the jid of the receiver and the payload. The IoT Bridge forwards the request to the XMPP server which, in turn, sends it to the XMPP device. The XMPP device responds back and the response takes the same path back to the computer.

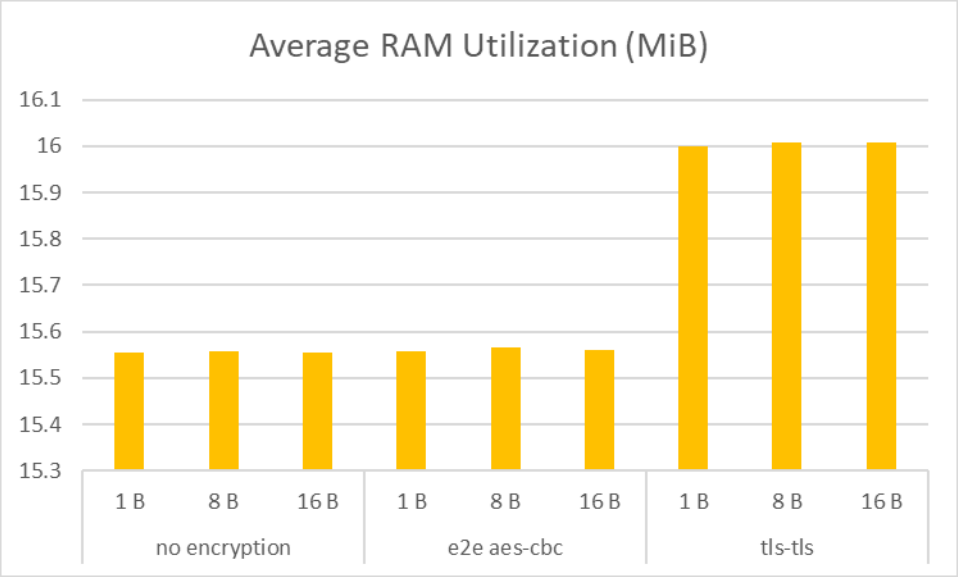
Evaluation



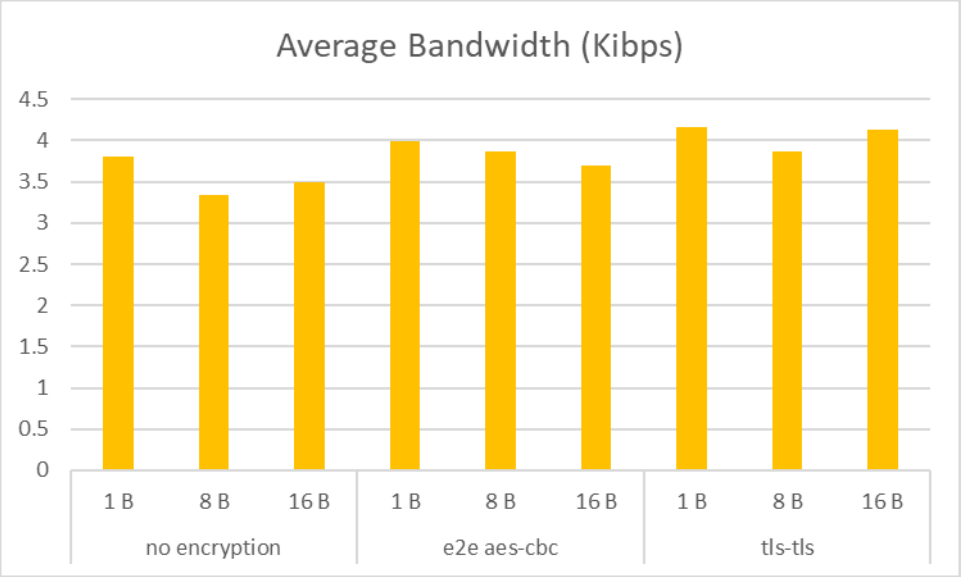
As can be seen clearly from the Figure above, the TLS requires more time because all devices (client, XMPP Server, IoT Bridge, XMPP resource) have to encrypt/decrypt the payload, whereas in end-to-end encryption only the client and the XMPP resource are involved in this procedure.



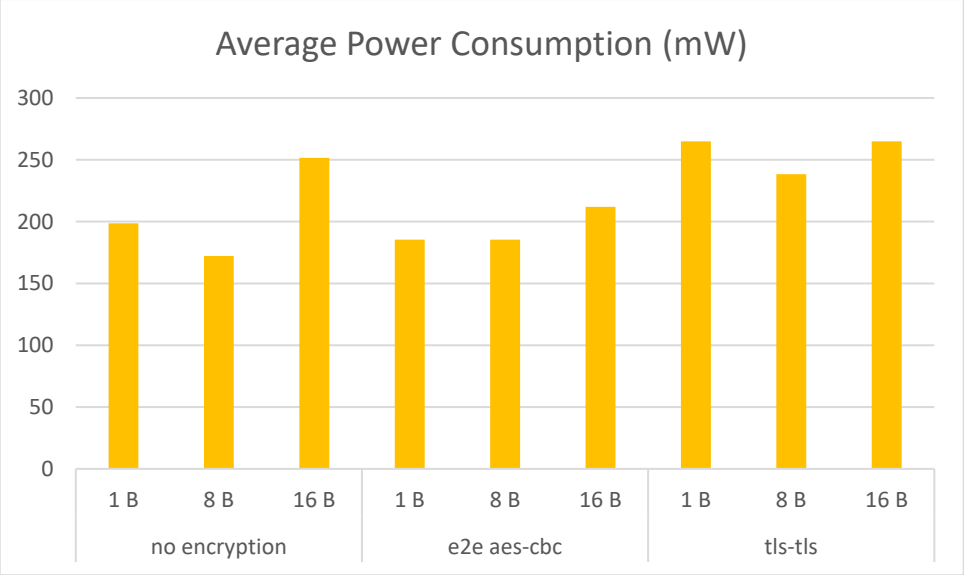
It comes as no surprise, that the CPU utilization is a bit higher when TLS is used, because in the first two modes, the IoT Bridge does not encrypt/decrypt anything. In the first two modes, the utilization is quite the same.



Like in the previous scenarios, TLS consumes more RAM, because in that case the IoT Bridge joins the encryption/decryption game.



There is no significant difference among different encryption modes and payload sizes.



4.5. Scenario 4: MQTT - CoAP

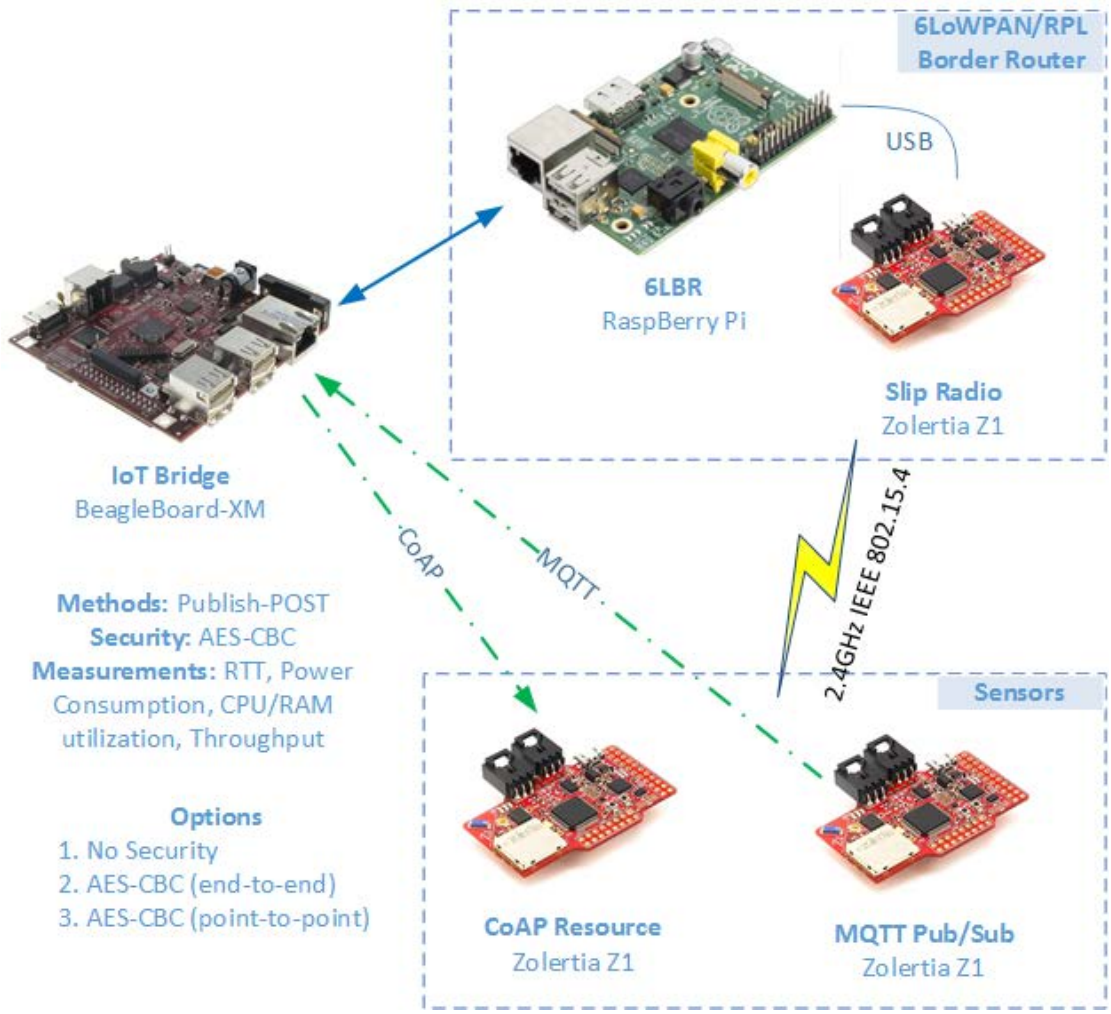


Figure 4.7: Scenario 4: MQTT - CoAP

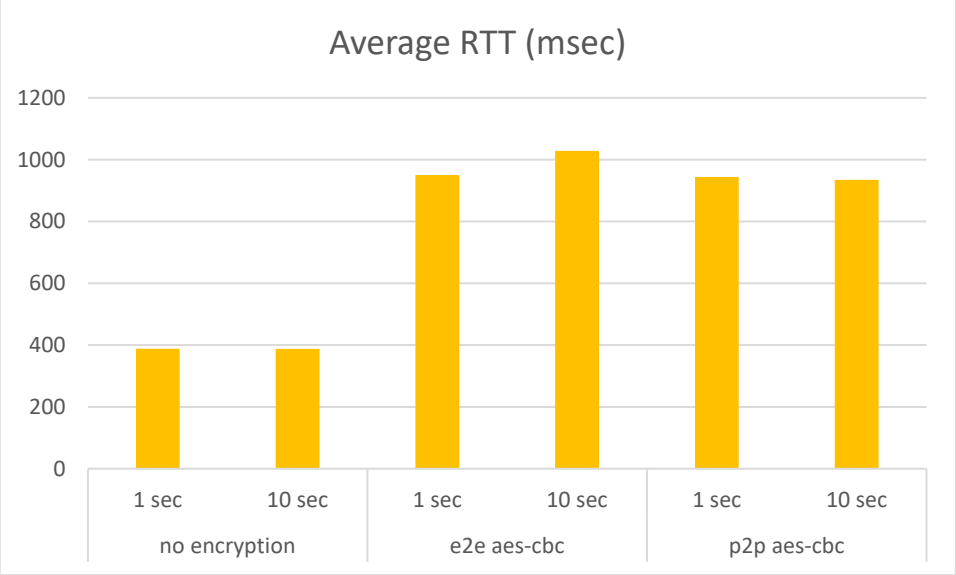
Sensors may also require communication to other sensors. Typical examples are networks that lack of central control and therefore actions are taken collectively, and communication between a sensor and an actuator (e.g. a water leak detector is triggered and instructs the actuator to close the main valve.) In this scenario, the IoT Bridge enables the communication between an MQTT mote and a CoAP mote.

Plot

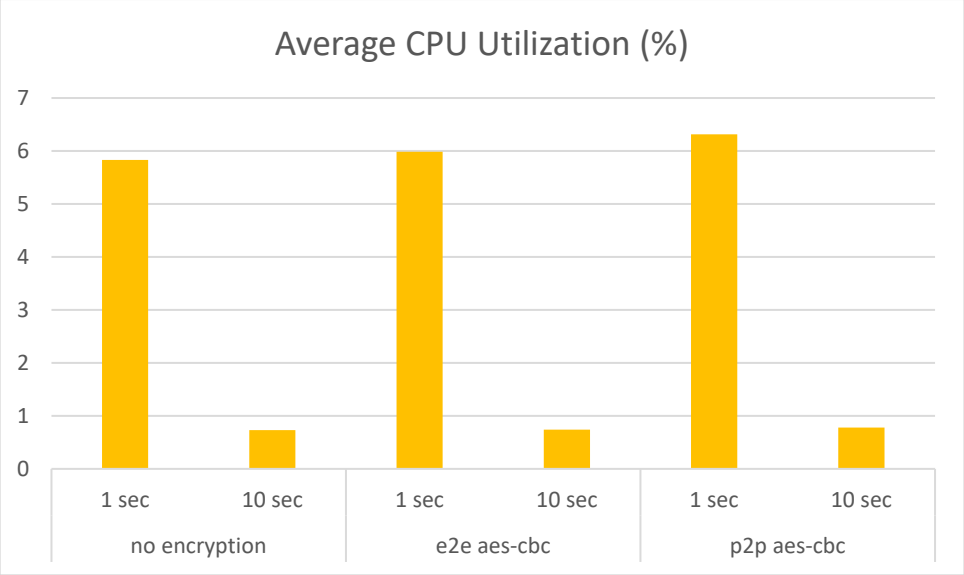
The MQTT device publishes a message periodically on a specific topic. The CoAP is subscribed to this topic via the IoT Bridge. As a subscriber on behalf of the CoAP device, the IoT Bridge receives the message from the MQTT broker and forwards it to the CoAP device by issuing a CoAP *POST* request.

Evaluation

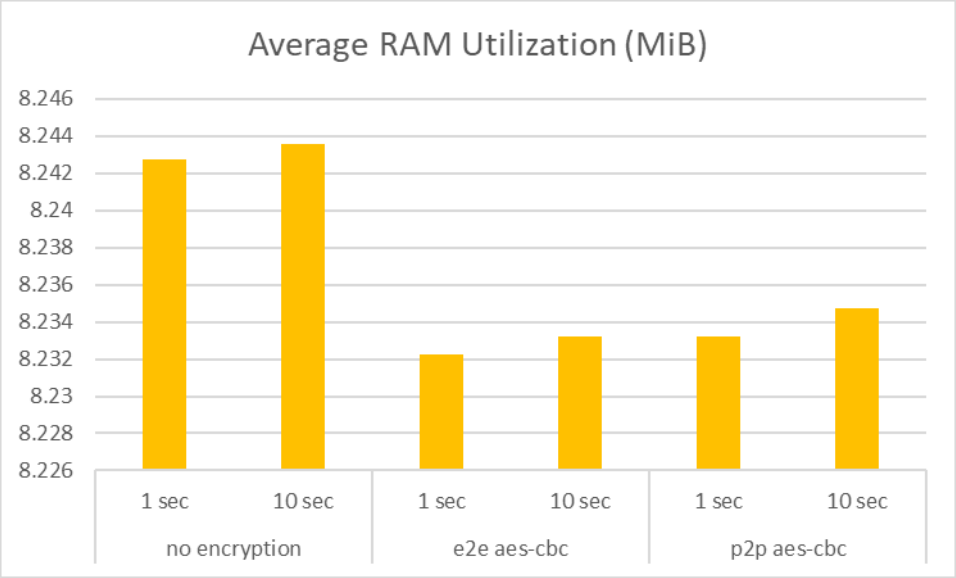
The last 3 scenarios share the same payload size and two intervals are tested (1 and 10 seconds).



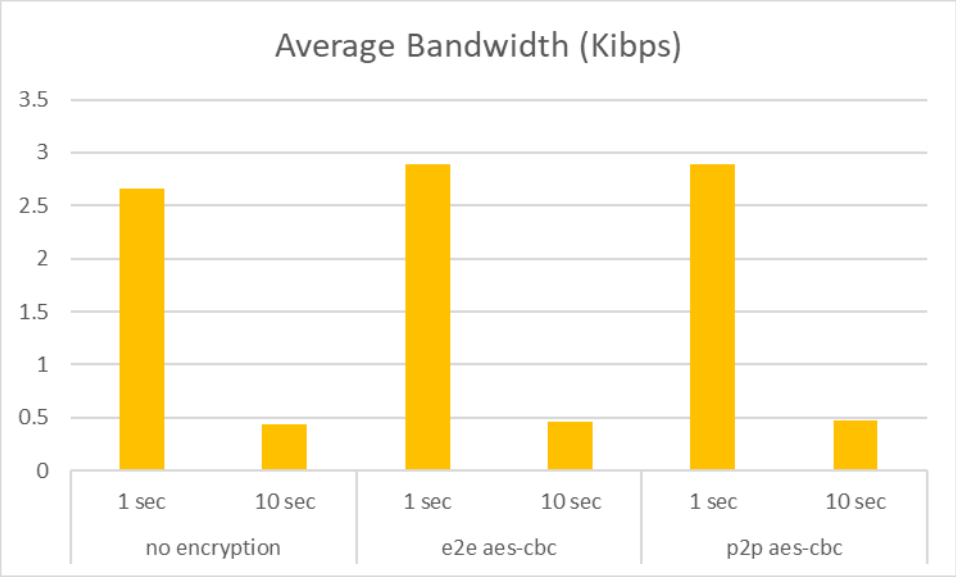
This scenario includes two Z1 motes communicating through the bridge, so the increased RTT is due to the fact that both motes have to encrypt/decrypt the payload. Moreover, the MQTT, as an asynchronous protocol, makes it hard to calculate an exact RTT value for the whole scenario.



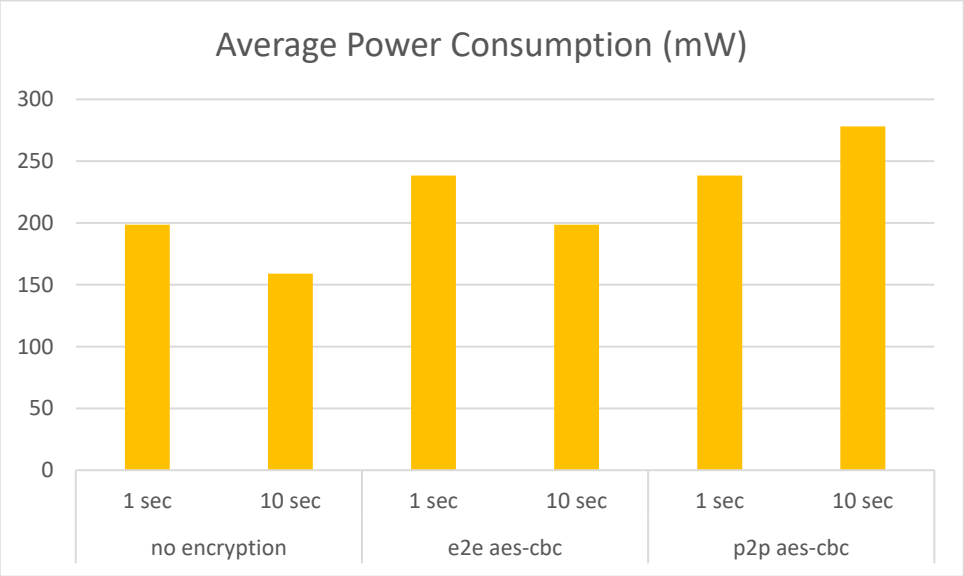
The CPU utilization of the IoT Bridge is almost imperceptible, especially in the case of the 10-second interval. The CPU utilization does not differ between encryption modes.



It may seem odd that no encryption uses more RAM than the other modes, but one should note the actual difference is about 0.01MiB which is 10.24KiB of RAM, that may have been used by any concurrent process or may not be freed in time.



The average bandwidth in 1second interval is about the same compared to other scenarios.



4.6. Scenario 5: XMPP - MQTT

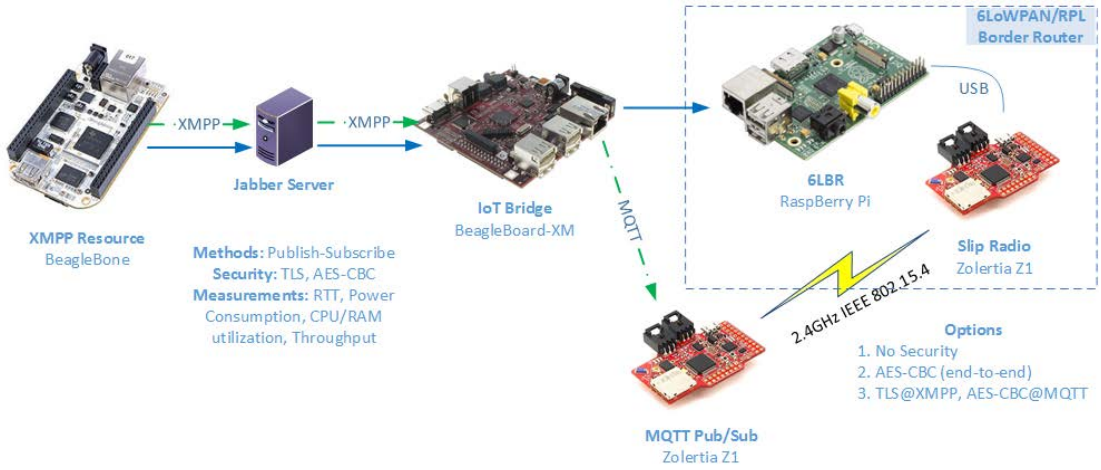


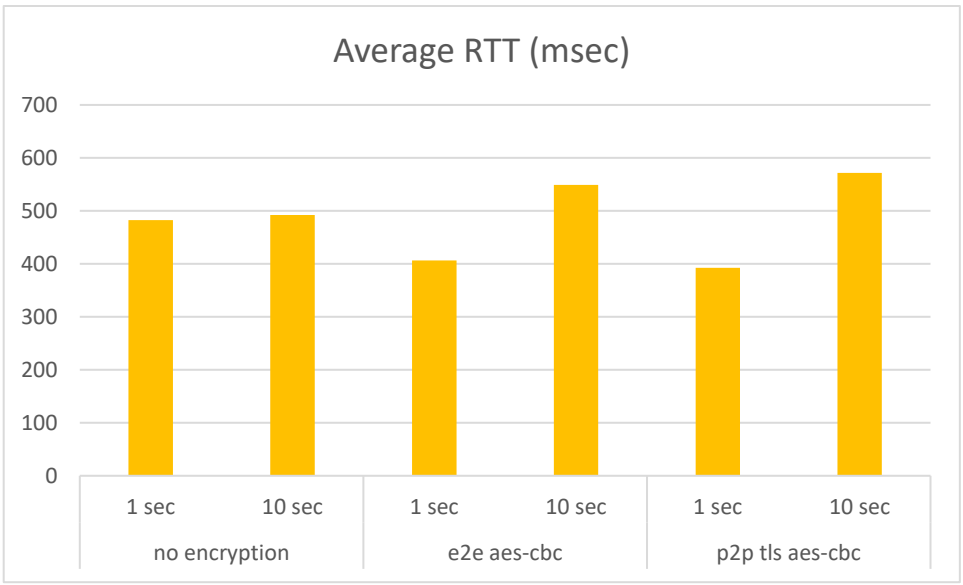
Figure 4.8: Scenario 5: XMPP - MQTT

Two asynchronous protocols such as XMPP and MQTT also require the IoT Bridge in order to exchange messages between each other. Team collaboration tools and their integrations can be part of this scenario too, where a special crafted message (e.g. a command) sent by a team member can be received by a sensor/actuator.

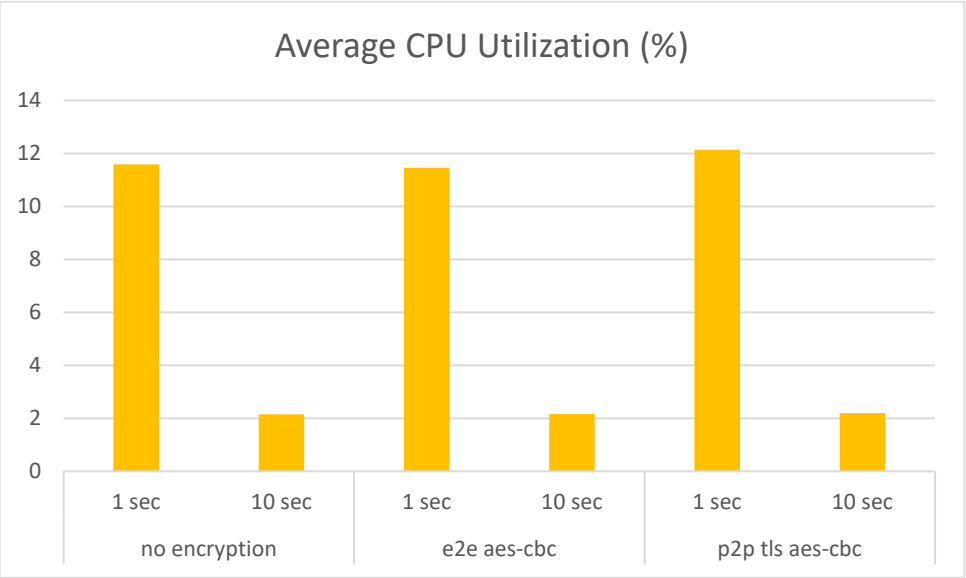
Plot

The XMPP device publishes a message to the MQTT broker through the IoT Bridge. More specifically, the XMPP device sends a message to the IoT Bridge using the latter's jid, containing the topic and the payload. The XMPP server relays the message to the IoT Bridge which, in turn, publishes it to the MQTT broker. The MQTT device receives the message because it is a subscriber on that topic.

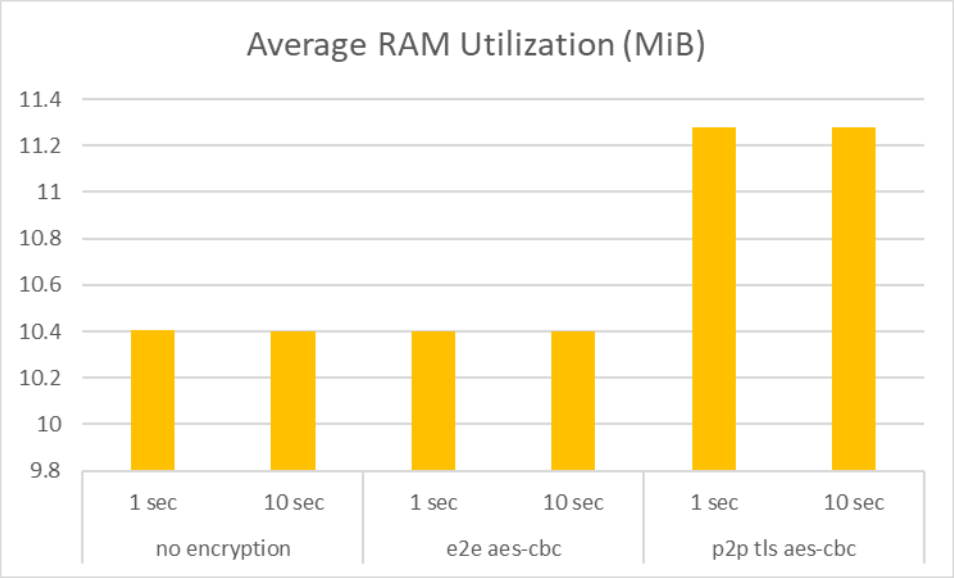
Evaluation



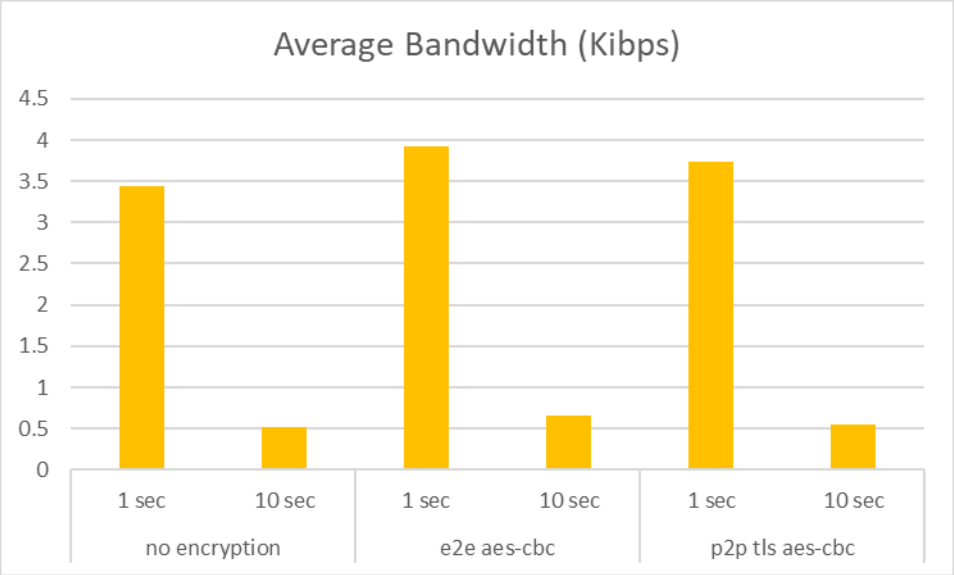
The time elapsed between the *PUBLISH* event sent the MQTT Broker and the moment the MQTT client received the message cannot be measured due to the lack of a common clock.



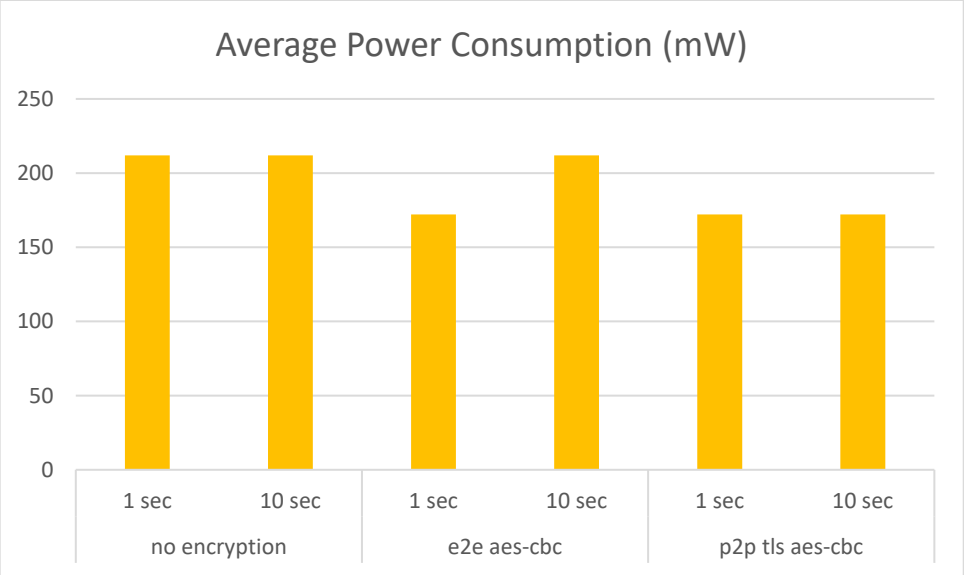
The CPU utilization is almost the same among different encryption modes and it is rather small regarding the 1 second interval.



The use of TLS and the fact that the IoT Bridge is required to decrypt and then re-encrypt the message increase RAM utilization.



With the interval of 1 second, the bandwidth utilization is almost the same as in previous scenarios. That is the case for the interval of 10 seconds too, regarding the last three scenarios in general.



4.7. Scenario 6: XMPP - CoAP

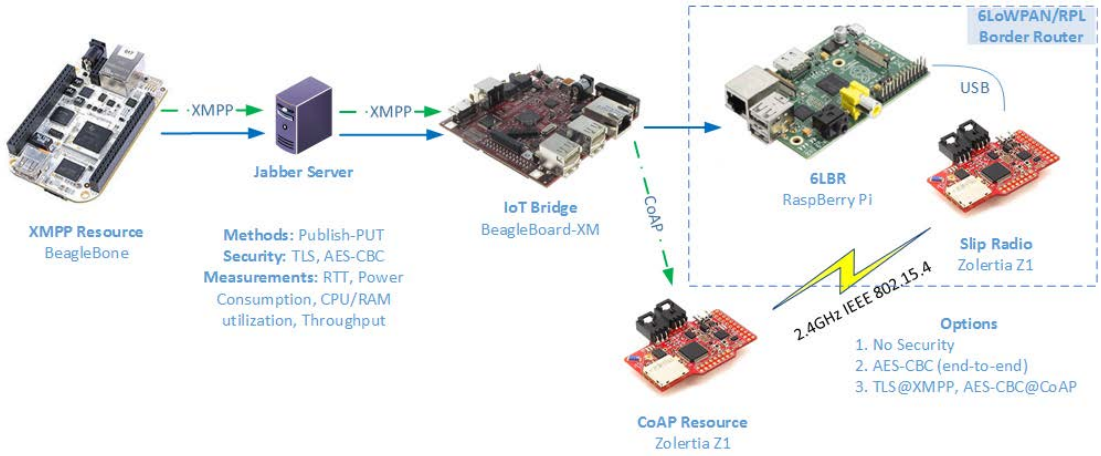


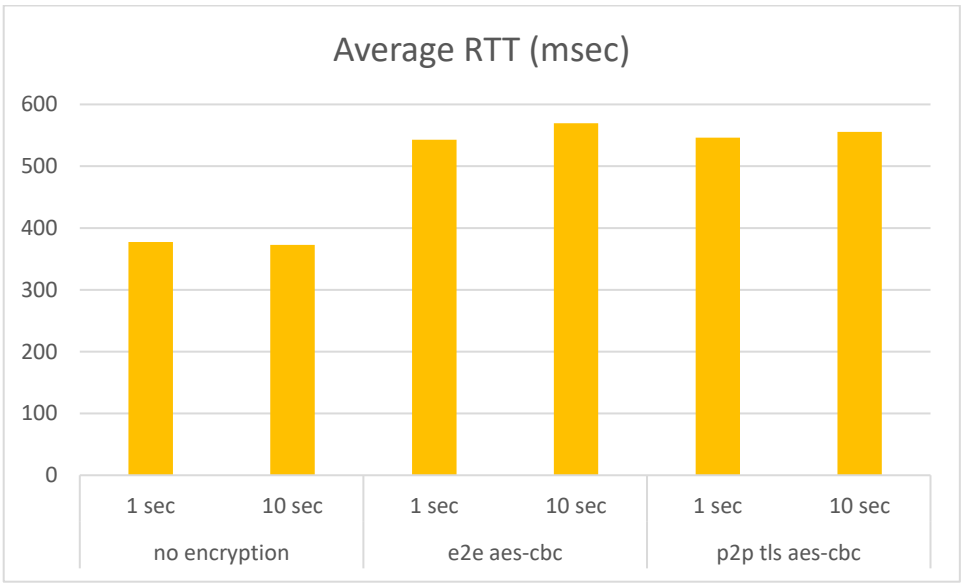
Figure 4.9: Scenario 6: XMPP - CoAP

There are a lot of messaging protocols but most of them are proprietary. The fact that XMPP is an open standard is an advantage and many services expose their proprietary protocols through an XMPP API.

Plot

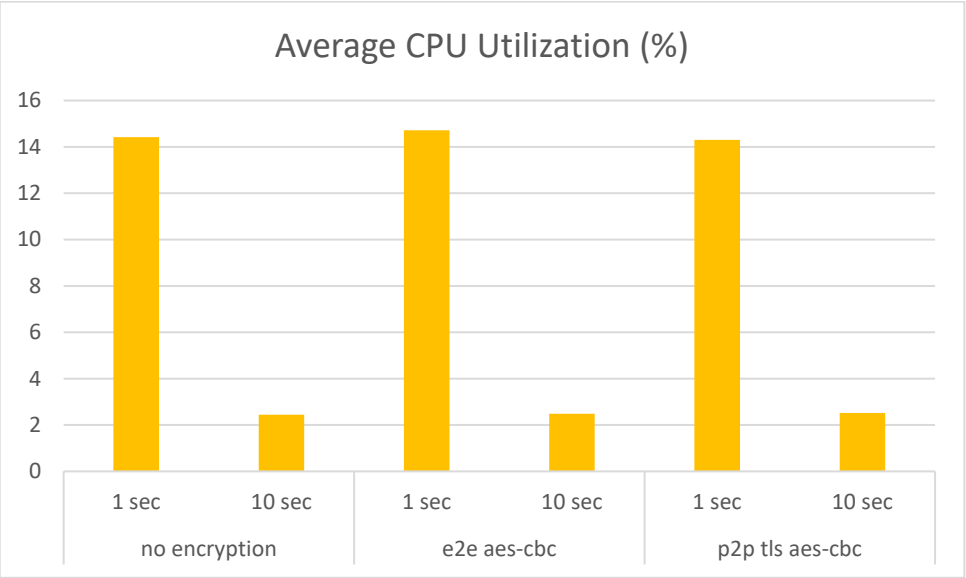
The XMPP device publishes a message to the IoT Bridge, containing the id of the CoAP device, the path of the resource and the action to be taken (*PUT*). The IoT Bridge upon receiving the message, it issues a CoAP *PUT* request to the CoAP device. The CoAP device responds back and the response is routed to the XMPP device through the IoT Bridge.

Evaluation

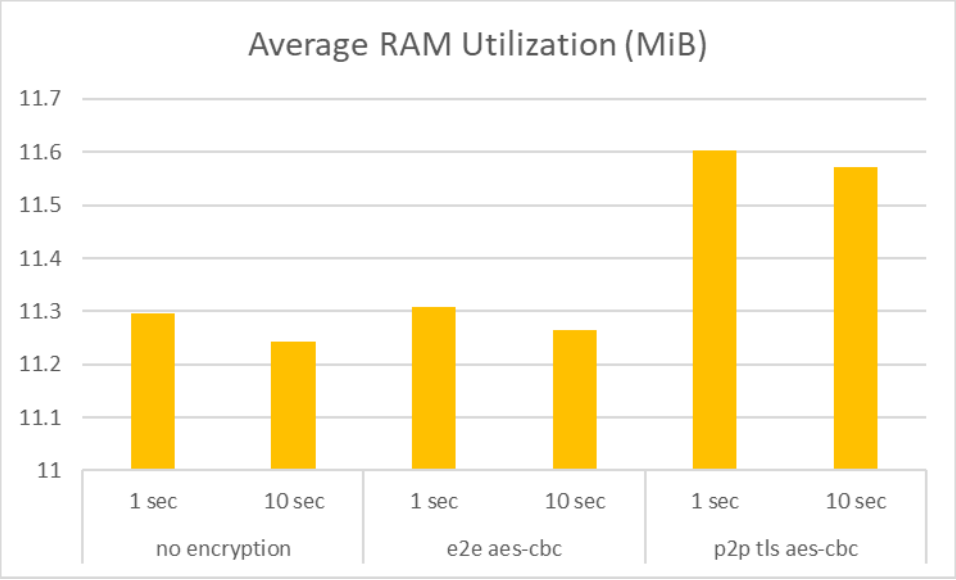


With no encryption, the CoAP client can respond much faster. When encryption is used, the RTT is almost steady, because the effect of the CoAP client (Z1) struggling to

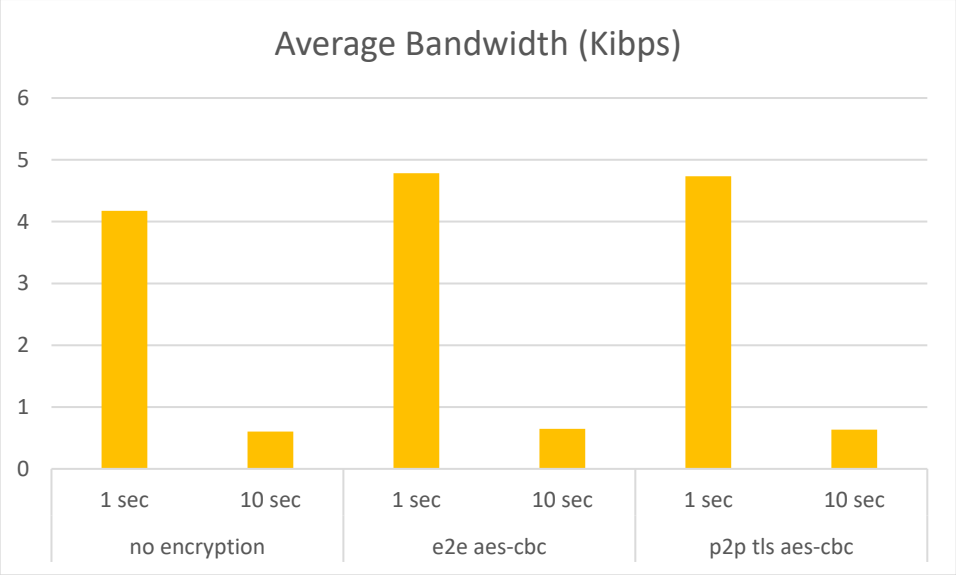
encrypt/decrypt is greater than the effect the TLS and the encryption/decryption have on the IoT Bridge.

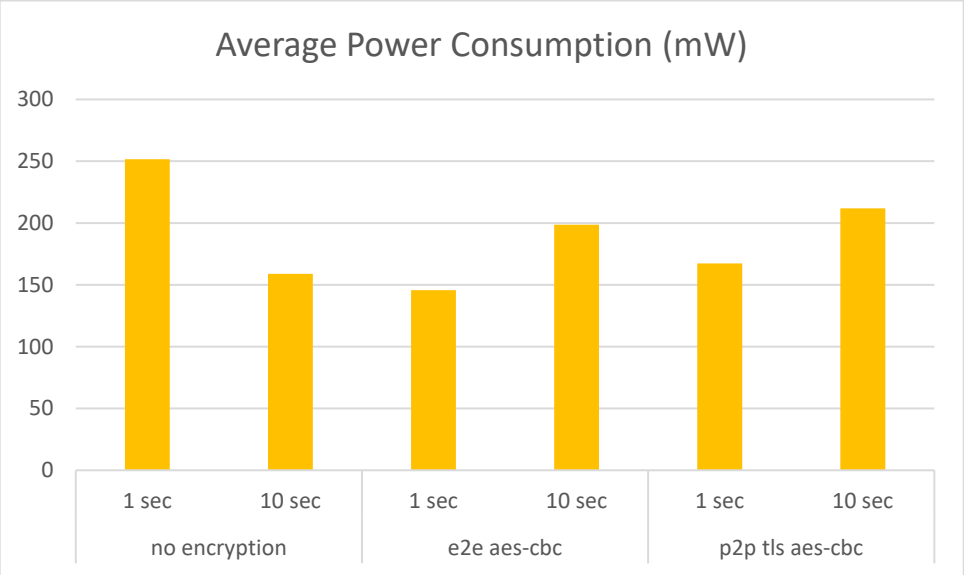


By comparing the first two modes with the last one (p2p), one can see that the encryption does not increase the utilization of CPU. In the last two scenarios, the TLS connection is established once. With that in mind, it is evident that in the last three scenarios the CPU utilization does not change when the IoT Bridge encrypts/decrypts (in peer-to-peer encryption mode). In the first three scenarios that was not the case, as there was an HTTP server with TLS utilized and each connection to the IoT Bridge required a TLS handshake.



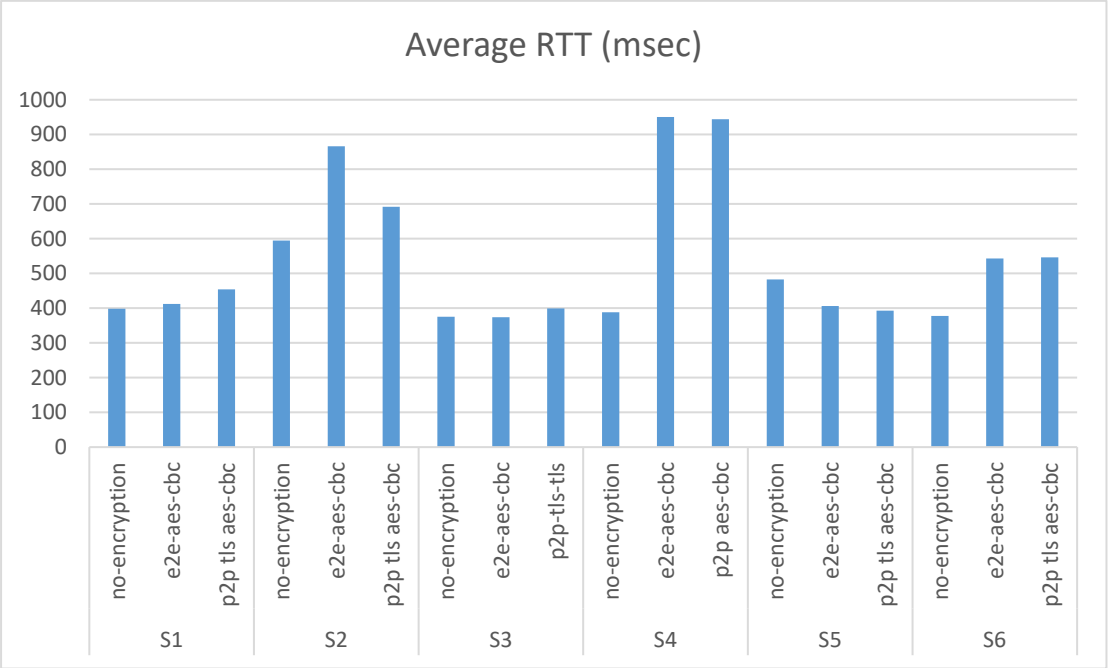
The CPU utilization may be not affected by the encryption/decryption, but RAM utilization surely is. It is an expected result in this scenario, as well as in the previous ones, because the encryption/decryption algorithms require space to hold their calculations during the execution of each step.


Average Bandwidth (Kibps)
6
5
4
3
2
1
0
1 sec
10 sec
1 sec
10 sec
1 sec
10 sec
no encryption
e2e aes-cbc
p2p tls aes-cbc


Average Power Consumption (mW)
300
250
200
150
100
50
0
1 sec
10 sec
1 sec
10 sec
1 sec
10 sec
no encryption
e2e aes-cbc
p2p tls aes-cbc

4.8. Comparison among scenarios

The demonstration of the results ends with a comparison between different scenarios on various measurements. With an interval of 1 second between requests, which is faster than a typical traffic load, and 16 bytes of payload, the following charts depict the average of 20 consecutive requests per security mode (no encryption, end-to-end AES-CBC encrypted payload and point-to-point encrypted payload) and scenario.



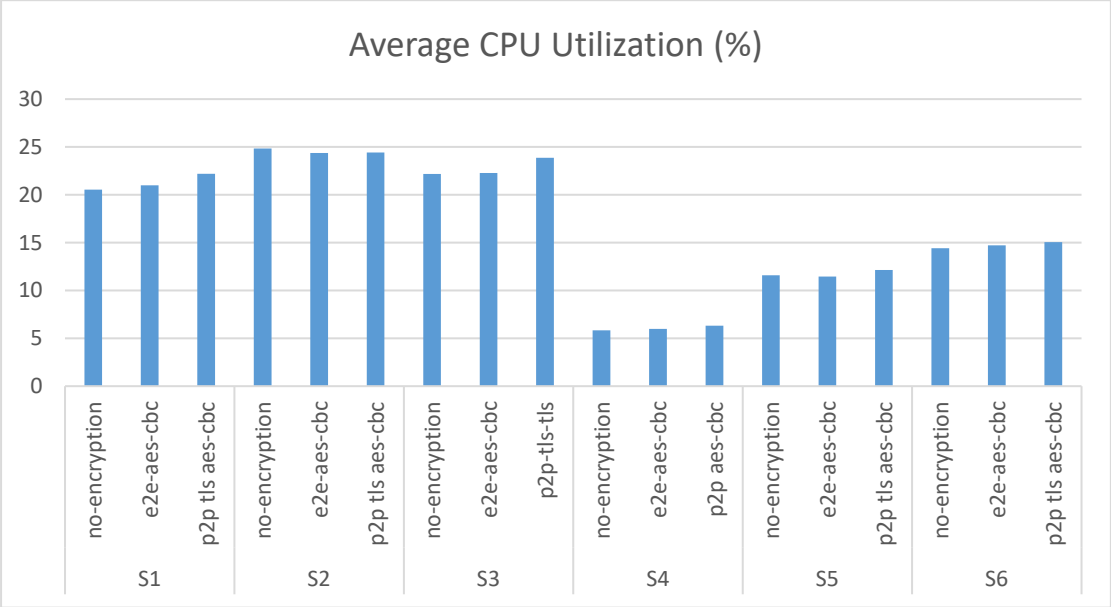
The Figure above displays the average Round-Trip-Time in milliseconds as seen from the aspect of the client. In particular, in Scenario 1, this is the time elapsed from the moment the PC sent the HTTP request till the moment it received the response from the IoT Bridge (which forwarded the response from Z1). The same holds true for Scenarios 3 and 6 too. In Scenarios 2, 4, 5, the RTT is the sum of individual RTTs between each connection (e.g. in S2: PC-IoT Bridge and Z1-Broker), due to the asynchronous nature of these scenarios.

In Scenario 1, the results are quite anticipated, as there is a slight increase in end-to-end encryption and a greater one in case of point-to-point encryption. In the first case only the sender and the receiver are required to encrypt and decrypt the messages whereas in the second case there is a TLS handshake during the connection establishment and the double encryption/decryption the IoT Bridge does before it forwards each message. Scenarios 3 and 5 are quite similar to Scenario 1.

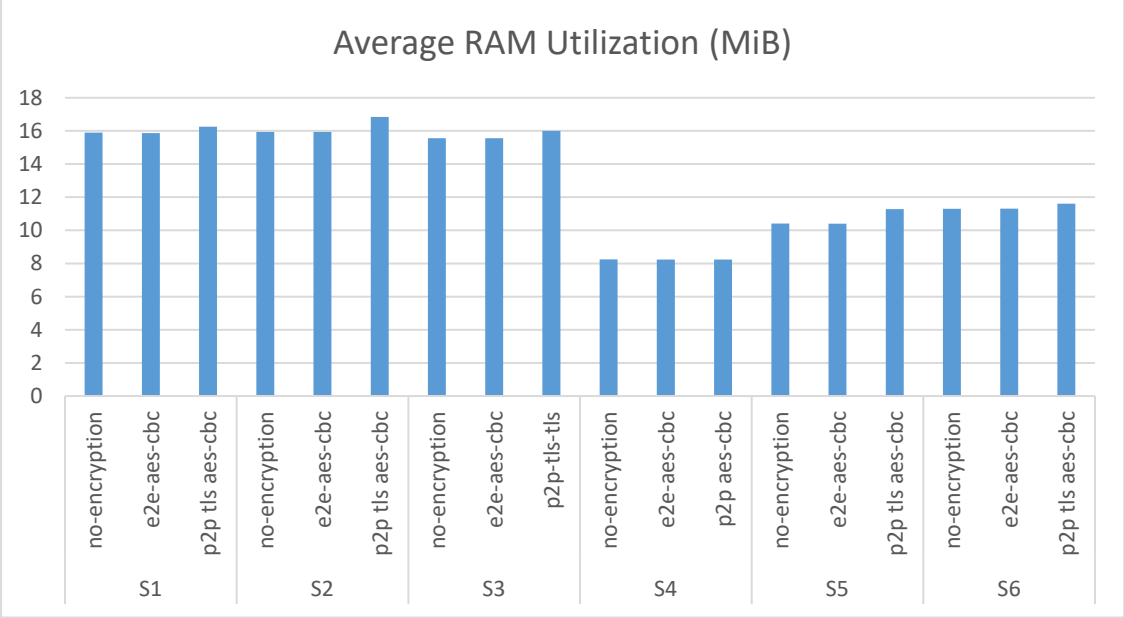
In Scenarios 2, 4 and 6, the impact of encryption on Z1 running MQTT is rather significant. Another factor that makes these scenarios different is the fact that MQTT is asynchronous and the actual RTT is difficult to be measured. Moreover, one should take into account the fact that the PUBACK, the broker sends upon message publishing, makes Z1 wait and the fact that a message has to be encrypted and sent every 1 second pushes the device to its limits.

Overall, the impact of the IoT Bridge in RTT is notable, but encryption's/decryption's contribution to the RTT, when handled by the IoT Bridge, is minor. The reason is that the impact of encryption on resource-constrained devices is far larger than the one of IoT Bridge, although the latter is not running on specialized/fast hardware and is not optimized. Considerable is also

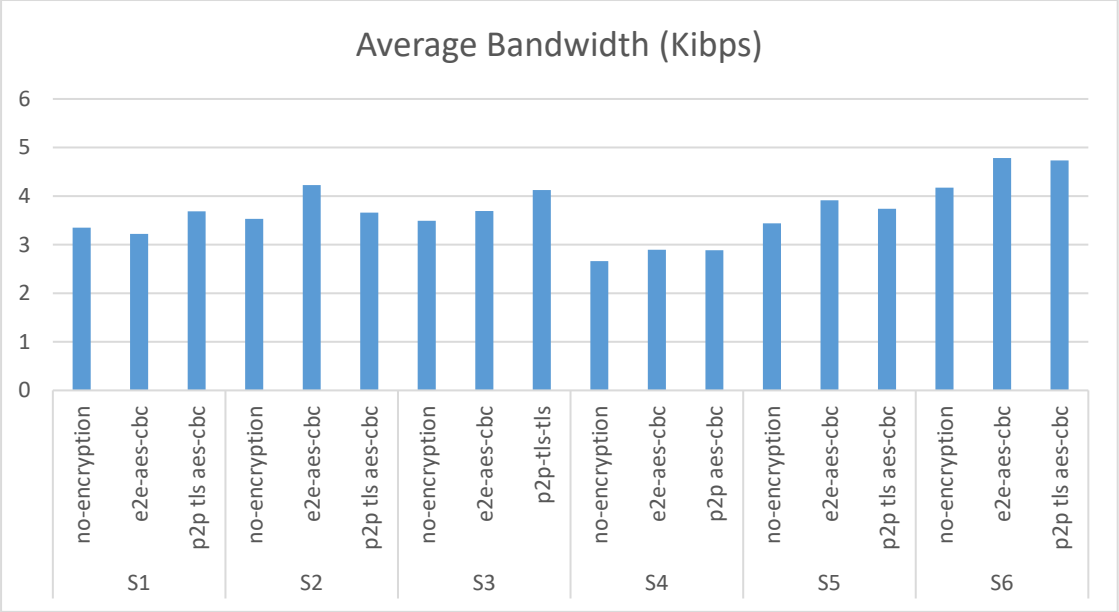
the fact that the network on which these scenarios were tested was not equipped with professional hardware and there were times the network performance was a bit sluggish, impacting the results.



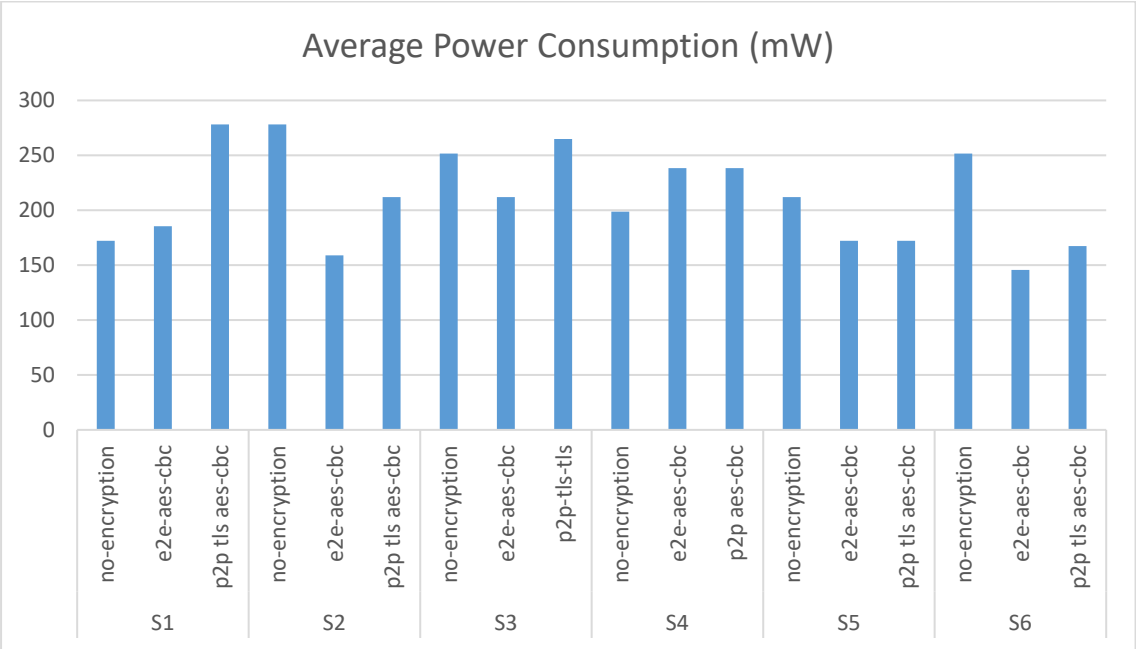
The Figure above demonstrates the average CPU utilization of the IoT Bridge. The impact of the RESTful API (HTTP), which is utilized in the first 3 scenarios, cannot be overlooked. Besides that, the IoT Bridge seems to handle the high traffic volume with ease. Scenario 4 is the most lightweight because the CoAP server it utilizes and the connection to the MQTT Broker are not as resource-heavy as the HTTP server and the XMPP connection.



The average RAM utilization in mebibytes is depicted in the above Figure. The RESTful API has likewise impacted the RAM utilization. In the last two scenarios, the IoT Bridge holds an open connection with the XMPP Server and that is the main reason, RAM is more utilized than in Scenario 4.



The average bandwidth, in kibibits per second, regarding the incoming and outgoing traffic is shown in the Figure above. Considering that the interval is 1 second, the bandwidth used is not that high, owing to the fact that the payload is rather small (16 bytes).



Another measurement was the power consumption. As shown in the Figure above, the power consumption in milliwatts was limited. It should be noted that the measurements include everything running such as IoT Bridge, MQTT Broker, OS, and various services. Having in mind the utilization of CPU, RAM, bandwidth, the low power consumption is much anticipated.

5. Conclusions

Sooner or later, good or bad, everything comes to an end. This time has arrived and a retrospection is necessary.

5.1. Recapitulation

This thesis resulted in developing an IoT Bridge that allows a secure message exchange between devices that may use different protocols. More specifically, the currently supported protocols are: HTTP, CoAP, MQTT, XMPP. The IoT Bridge offers payload encryption in point-to-point communication, which is very useful when the communicating devices are not trusted and a global encryption key cannot be shared, thus obstructing devices from intercepting messages from other devices.

The plethora of heterogeneous protocols are posing a challenge to the evolution and the adaption of the IoT. From the very beginning, when pioneers created what is now taken as granted, such as computers, networks, wireless connections, advanced software etc., companies were creating their own protocols and programming languages, an understandable choice because there were no standards in these sectors back then. Nowadays, the development is so rapid, that these standardised protocols are not ideal for some products. For instance, a low-power, resource-constrained device requires protocols that are efficient and able to run on such a platform. Under these circumstances, the creation of new protocols and new standards is favoured, exacerbating the landscape of the heterogeneous protocols.

The more protocols are utilised among the devices, the more exigent the adoption of interoperability is going to be. Interoperability is an integral part of the IoT world. Without it, the desirable interconnection between heterogeneous devices cannot be achieved. Imagine, for example, some manufacturers ignoring the IEEE 802.11 (WiFi) standard. It would be certainly burdensome to create an infrastructure for these devices, because wireless Access Points would have to either support the different protocols each manufacturer uses (instead of WiFi) or a variety of Access Points would have to be purchased in order to cover the whole gamut. It goes without saying, that this situation is not scalable and unjustifiably expensive. The road towards interoperability means that new IoT applications should be extensible, so that new functions and features can be integrated without compromising the existing ones, building on top of the already implemented communication technologies.

Protocol heterogeneity is also a threat to security. The plethora of protocols, especially the proprietary ones, makes security burdensome as professionals cannot focus on each and every protocol to audit and propose security fixes and features. Interoperability and security go hand in hand, because not only the standardised protocols can be made secure, but also frameworks and solutions in general can be built to assist proactive defence and threat identification.

5.2. Hindrances

"…slowly crashlooping…"

Due to the very nature of the embedded and resource-constrained devices, a plethora of challenging problems and difficulties emerged. This is hardly an exhaustive list of the hindrances encountered throughout the development.

The first and most persistent problem was the inability of 6LBR to work properly inside the Virtual Machine. Although the network interface of the said VM was in Bridge Mode, in order

to obtain a valid IPv6 address from the router's Dynamic Host Configuration Protocol (DHCP) server, 6LBR had difficulties in making the WSN nodes accessible to the Local-Area Network (LAN). After a lot of effort and experimentation with various parameters and modes in each involved software and hardware (router's configuration, host OS, virtualization software, guest OS, 6LBR), there was no reliable outcome. The trial-and-error phase included running 6LBR on actual hardware which, in this occasion, was a Raspberry Pi. Fortunately, 6LBR was fully functional in that case, putting all these efforts to an end. It was the only problem that limited resources were not to blame for. It is possible that any part of the whole stack of the involved software could have interfered with the proper function of 6LBR. Concerning the writer's view, the virtualization software may have been responsible.

Another problem was the buggy version 4.7.0 of the msp430-gcc compiler that somehow bounded the utilizable ROM of Zolertia Z1 to 56 KiB out of 92 KiB. In cases, such as the use of cryptographic algorithms, which are space-demanding, this was a significant obstacle. A lot of attempts were made to update to a newer version, but to no avail.

A consequence of the aforementioned problem may have been the inability to properly run the tinyDTLS which would lead to point-to-point encryption of the CoAP protocol. Several constants such as the maximum number of neighbours, the maximum number of observers and the length of the packet, have to be significantly reduced, in a way to reduce the requirements in ROM and make it fit in. Unfortunately, this was not enough, as the tinyDTLS was not working properly. Due to unknown reasons, the handshake was never completed successfully.

Attempting to secure the WSN, LLSEC was the way to go, but it ended up being a dead end. After some effort, the compilation was successful but no connection was established between the mote and 6LBR. Newer versions were tried, along with some examples included in the repository, but no configuration led to a successful connection.

Some questions were sent to the mailing lists regarding any minor or major problem encountered. Although not all questions received an answer, their answers were enlightening. Another fruitless attempt was the one to install 6LBR to BeagleBoard-xM, so that all bridge-related functions are housed on the same device. Despite the fact that the compilation was successful and the webpage of 6LBR was functioning properly, the very moment the 6LBR was starting any SSH session was terminated and, after a while, motes started losing their connection to 6LBR. Four different OS images were tested, in an attempt to fix the problem, but the result was common to all of them: an orange screen upon boot and a black screen with a blinking cursor next.

5.3. Lessons Learned

The obvious ones would be security-related facts and experience with resource-constrained devices. Of course, these were the case, among other things, but the most important lesson learned, regarding the writer's view, is the following. A lot of people, regardless if they are specialists or not, are accusing manufacturers and developers for insecure products. In fact, there were such accusations in the Introduction of this thesis.

One of the main reasons for this, is the lack of experience. Many developers create programs that run on computers or smartphones, which both have large amounts of resources when compared to resource-constrained devices. The moment they start writing programs for these devices, they run into issues, such as reducing the size of their code and extensive optimization, that they may have never dealt with. Moreover, they may have to work towards

their own solutions for already solved problems, because the existing ones were not developed with these devices in mind.

There may be a plethora of reasons why a product is not secure, from incompetent designers/developers to limited budget. No one can deny that a major contributing factor is that security is not the primary concern and it is considered way after the preliminary design of the product, leading not only to difficulties in implementing and deploying security practices but also to vulnerabilities.

Security is hard, really hard. It is multilevel, multifaced and unbound, but none of these are an excuse to ignore it entirely or postpone its realization. To put it another way, "engineering without security is just art". With no intent to undermine the importance of art, anything an engineer designs have to withstand any reasonable (or not) amount of hardship, depending on the application. This is clearly illustrated by a building. It takes a team of engineers to design it and build it, but if they do not safeguard that the structure will withstand the proper static (i.e. appliances, furniture, machinery) and dynamic (i.e. people) load, it is not a building that people can use, rather than a visual art that people can observe.

5.4. Future Work

A variety of aspects of the IoT Bridge can be improved. First of all, more security mechanisms can be added, such as authentication, encryption schemes etc. If more than one encryption schemes are available, a negotiation can be used between the client and the IoT Bridge, like the one done during the TLS handshake. There is also a plethora of protocols either specialized for IoT devices or not, that can be implemented and supported by the IoT Bridge. An important note here is that the more complexity increases by adding new features and mechanisms, the more difficult it becomes to assess the IoT Bridge as a whole in matters of security.

The implementation of the IoT Bridge itself could also be optimized in order to become faster regarding RTT, by choosing the ideal programming languages, software libraries and what the target hardware would be (a platform like BeagleBone-xM or something more powerful). During the last year, big companies that offer cloud services have also introduced IoT-related solutions, either similar in concept to the IoT Bridge or software libraries that can be used to integrate their cloud services (such as Big Data analysis, Machine Learning, storage, security etc.) to new or current solutions. In the case of the IoT Bridge, these libraries can be utilized. The integration with voice assistants would also be interesting. For instance, the IoT Bridge can expose desired features to such assistants ensuring both the interoperability between the assistants and the various IoT devices, and their security, in a way to block any not authorized/untrusted command.

Emerging standards are numerous in the world of IoT. NarrowBand IoT is a Low Power Wide Area Network radio technology standard that aims to the interconnection of IoT through the cellular network. LWM2M (LightWeight Machine-to-Machine) is a standard that includes several protocols like CoAP, DTLS, SenML (Sensor Markup Language) etc. The purpose of LWM2M is the management of applications running on M2M devices and their remote control.

Bibliography

[1] ABI Research, "More Than 30 Billion Devices Will Wirelessly Connect to the Internet of Everything in 2020," 09 May 2013. [Online]. Available: https://www.abiresearch.com/press/more-than-30-billion-devices-will-wirelessly-conne/. [Accessed 11 November 2016].

[2] Rapid7, "R7-2015-23: Comcast XFINITY Home Security System Insecure Fail Open," Rapid7, 5 January 2016. [Online]. Available: https://community.rapid7.com/community/infosec/blog/2016/01/05/r7-2015-23-comcast-xfinity-home-security-system-insecure-fail-open. [Accessed 18 October 2016].

[3] L. Lamb, "Home insecurity: No alarms, false alarms and SIGINT," 2014. [Online]. Available: goo.gl/YY4nDp.

[4] B. Krebs, "KrebsOnSecurity Hit With Record DDoS," 21 September 2016. [Online]. Available: https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/. [Accessed 28 September 2016].

[5] S. Hilton, "Dyn Analysis Summary Of Friday October 21 Attack," 26 October 2016. [Online]. Available: http://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/. [Accessed 5 November 2016].

[6] H. C. A. v. Tilborg and S. Jajodia, "Kerchhoff's principle," in *Encyclopedia of Cryptography and Security*, Springer US, 2011, p. 675.

[7] B. Schneier, "Schneier's Law," April 2011. [Online]. Available: https://www.schneier.com/blog/archives/2011/04/schneiers_law.html. [Accessed November 2016].

[8] Gartner, "Gartner Says By 2020, a Quarter Billion Connected Vehicles Will Enable New In-Vehicle Services and Automated Driving Capabilities," 26 January 2015. [Online]. Available: http://www.gartner.com/newsroom/id/2970017. [Accessed 11 November 2016].

[9] L. Atzori, A. Iera and G. Morabito, "The Internet of Things: A survey," *Computer Networks,* vol. 54, no. 15, pp. 2787-2805, 28 October 2010.

[10] A. S. Reddy, "Reaping the Benefits of the Internet of Things," May 2014. [Online]. Available: https://www.cognizant.com/InsightsWhitepapers/Reaping-the-Benefits-of-the-Internet-of-Things.pdf. [Accessed November 2016].

[11] A. Zanella, N. Bui, A. Castellani, L. Vangelista and M. Zorzi, "Internet of Things for Smart Cities," *IEEE Internet of Things Journal,* vol. 1, no. 1, pp. 22-32, 14 February 2014.

[12] Akamai, "How the Mirai botnet is fueling today's largest and most crippling DDoS attacks," November 2016. [Online]. Available: https://www.akamai.com/cn/zh/multimedia/documents/white-paper/akamai-mirai-botnet-and-attacks-against-dns-servers-white-paper.pdf. [Accessed 6 November 2016].

[13] E. Ronen, C. O'Flynn, A. Shamir and A.-O. Weingarten, "IoT Goes Nuclear: Creating a ZigBee Chain Reaction," [Online]. Available: http://iotworm.eyalro.net/iotworm.pdf. [Accessed 19 November 2016].

[14] B. Schneier, "Security and the Internet of Things," 1 February 2017. [Online]. Available: https://www.schneier.com/blog/archives/2017/02/security_and_th.html. [Accessed 1 February 2017].

[15] J. Singh, T. Pasquier, J. Bacon, H. Ko and D. Eyers, "Twenty Security Considerations for Cloud-Supported Internet of Things," *IEEE Internet of Things Journal,* vol. 3, no. 3, pp. 269-284, 23 July 2015.

[16] J. Yick, B. Mukherjee and D. Ghosal, "Wireless sensor network survey," *Computer Networks,* vol. 52, no. 12, pp. 2292-2330, 22 August 2008.

[17] IEEE Standard for Low-Rate Wireless Networks, "IEEE Std 802.15.4-2015," *a,* 2015.

[18] IETF, "RFC791: Internet Protocol," September 1981. [Online]. Available: https://tools.ietf.org/html/rfc791.

[19] J. H. Saltzer, D. P. Reed and D. D. Clark, "End-To-End Arguments in System Design," *ACM Transactions on Computer Systems,* vol. 2, no. 4, pp. 277-288, November 1984.

[20] G. Mulligan, "The 6LoWPAN architecture," *EmNets '07 Proceedings of the 4th workshop on Embedded networked sensors,* pp. 78-82, 25 June 2007.

[21] J. Vasseur, N. Agarwal, J. Hui, Z. Shelby, P. Bertrand and C. Chauvenet, "RPL: The IP routing protocol designed for low power and lossy networks," April 2011. [Online]. Available: http://www.ipso-alliance.org/wp-content/media/rpl.pdf. [Accessed January 2017].

[22] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur and R. Alexander, "RFC6550: RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," March 2012. [Online]. Available: https://tools.ietf.org/html/rfc6550.

[23] I. Fette and A. Melnikov, "RFC6455: The WebSocket Protocol," December 2011. [Online]. Available: https://tools.ietf.org/html/rfc6455.

[24] C. Bormann, M. Ersue and A. Keranen, "RFC7228: Terminology for Constrained-Node Networks," May 2014. [Online]. Available: https://tools.ietf.org/html/rfc7228.

[25] E. Rescorla and N. Modadugu, "RFC4347: Datagram Transport Layer Security," 2006 April. [Online]. Available: https://tools.ietf.org/html/rfc4347.

[26] K. Hartke, "RFC7641: Observing Resources in the Constrained Application Protocol (CoAP)," September 2015. [Online]. Available: https://tools.ietf.org/html/rfc7641.

[27] Z. Shelby, "RFC6690: Constrained RESTful Environments (CoRE) Link Format," August 2012. [Online]. Available: https://tools.ietf.org/html/rfc6690.

[28] ISO, "ISO/IEC 20922:2016, Information technology -- Message Queuing Telemetry Transport (MQTT) v3.1.1," June 2016. [Online]. Available: https://www.iso.org/standard/69466.html.

[29] NIST, "Status of the Advanced Encryption Standard (AES) Development Effort," 1999.

[30] "Announcing Request for Candidate Algorithm Nominations for the Advanced Encryption Standard (AES)," in *Federal Register*, vol. 62, 1997, pp. 48051-48058.

[31] NIST, "Announcing the Advanced Encryption Standard (AES)," *Federal Information Processing Standards,* 26 November 2001.

[32] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, T. Kohno and M. Stay, "The Twofish Team's Final Comments on AES Selection," 15 May 2000. [Online]. Available: https://www.schneier.com/academic/paperfiles/paper-twofish-final.pdf.

[33] S. Claude, "A Mathematical Theory of Cryptography," 1 September 1945. [Online]. Available: https://www.iacr.org/museum/shannon/shannon45.pdf.

[34] R. Housley, "RFC5652: Cryptographic Message Syntax (CMS), 6.3 Content-encryption Process," September 2009. [Online]. Available: https://tools.ietf.org/html/rfc5652#section-6.3.

[35] A. Dunkels, "Rime-a lightweight layered communication stack for sensor networks," *Proceedings of the European Conference on Wireless Sensor Networks (EWSN),* p. Poster/Demo session, January 2007.

[36] A. Dunkels, B. Gronvall and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," *Local Computer Networks, 2004. 29th Annual IEEE International Conference on,* 16-18 November 2004.

[37] A. Dunkels, O. Schmidt, T. Voigt and M. Ali, "Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems," *SenSys '06 ,* pp. 29-42, 31 October 2006.

[38] CETIC, "6lbr," [Online]. Available: https://calculator.s3.amazonaws.com/index.html. [Accessed 13 November 2016].

[39] CETIC, "6LBR in a Nutshell," 1 March 2016. [Online]. Available: https://github.com/cetic/6lbr/wiki/6LBR-in-a-Nutshell. [Accessed 13 November 2016].

[40] Zolertia, "Zolertia Z1," Zolertia, [Online]. Available: http://zolertia.io/z1.

[41] Zolertia, "Z1 Datasheet," 11 March 2010. [Online]. Available: http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf. [Accessed 18 November 2016].

[42] Zolertia, "Z1," 26 April 2013. [Online]. Available: http://zolertia.sourceforge.net/wiki/index.php/Z1. [Accessed 18 November 2016].

[43] Raspberry Pi, "Teachers' Guide to Raspberry Pi," [Online]. Available: https://www.raspberrypi.org/learning/teachers-guide/. [Accessed November 2016].

[44] S. Jain, A. Vaibhav and L. Goyal, "Raspberry Pi based interactive home automation system through E-mail," *Optimization, Reliabilty, and Information Technology (ICROIT), 2014 International Conference on ,* pp. 277-280, 6 February 2014.

[45] V. Vujović and M. Maksimović, "Raspberry Pi as a Sensor Web node for home automation," *Computers & Electrical Engineering 44,* pp. 153-71, 31 May 2015.

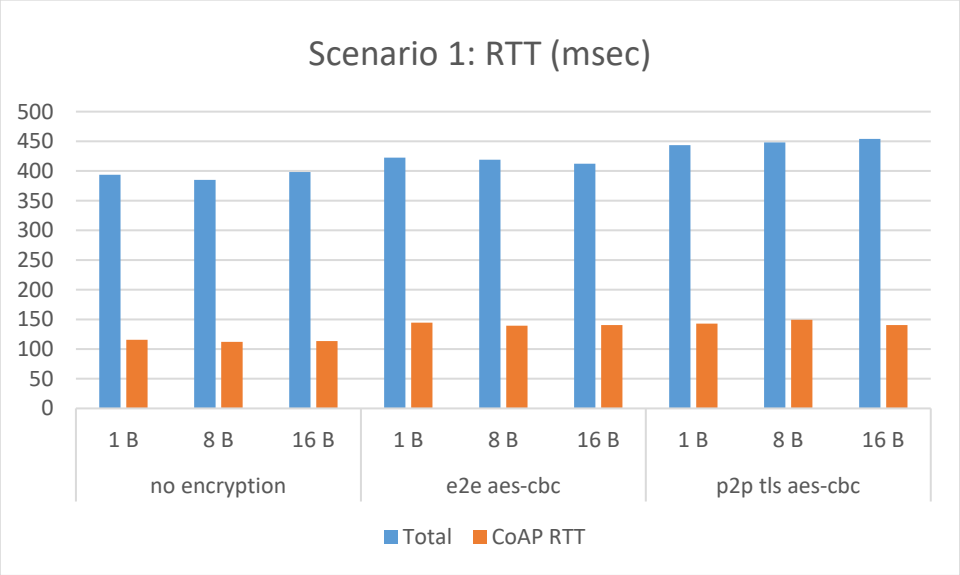
[46] A. Dunkels, J. Eriksson, N. Finne and N. Tsiftes, "Powertrace: Network-level Power Profiling for Low-power Wireless Networks," March 2011. [Online]. Available: https://core.ac.uk/download/pdf/11435067.pdf?repositoryId=362.

[47] Texas Instruments, "MSP430F261x and MSP430F241x Mixed Signal Microcontroller Datasheet," November 2012. [Online]. Available: http://www.ti.com/lit/ds/symlink/msp430f2417.pdf.

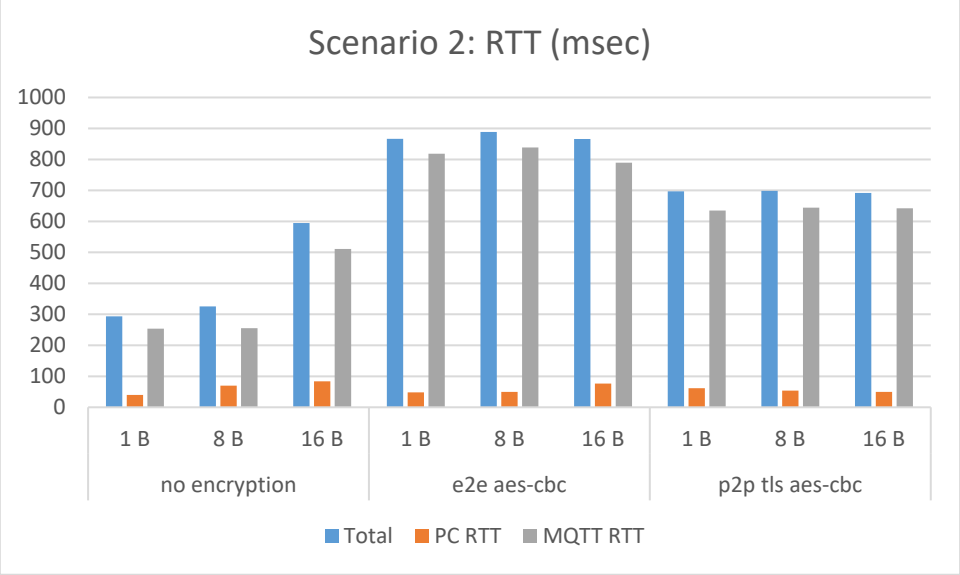
Annex A

This Annex contains graphs based on measurements taken on Z1 devices (running CoAP or MQTT) and the BeagleBone running an XMPP resource. In the first section, there are charts that demonstrate the RTT of each device, so that is clear which part of the architecture contributed the most on RTT per request.

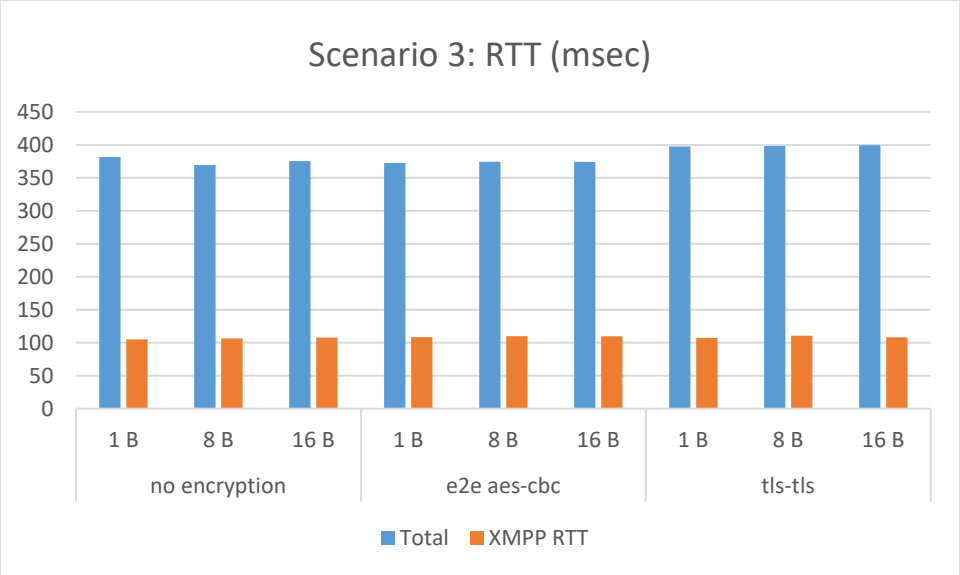
RTT broken into pieces



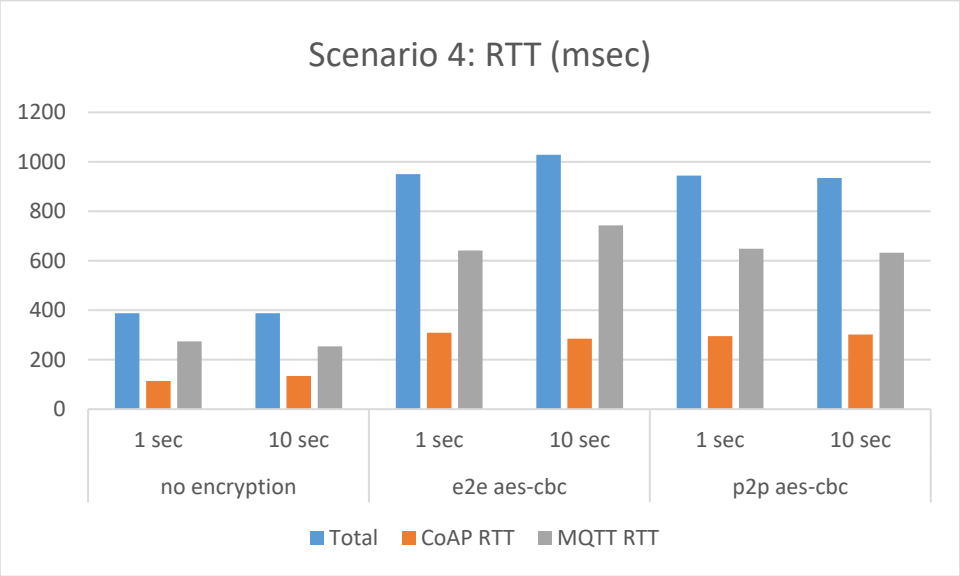
The total RTT is the time taken from the moment the PC/Server sent the request to the IoT Bridge till it got a response. The CoAP RTT is the time taken from the moment the IoT Bridge sent the request to the CoAP device (on behalf of the PC/Server) till it got a response. It goes without saying that the IoT Bridge contribution to the total RTT is significant.



The asynchronous nature of the technologies involved in this scenario (MQTT, Websockets) makes it hard to calculate the RTT precisely. The total RTT is calculated as the sum of the partial RTTs. PC RTT refers to the connection between the PC running the open Websocket and the IoT Bridge running the Websocket server. MQTT RTT refers to the connection between the MQTT Z1 device and the MQTT Broker hosted together with the IoT Bridge on BeagleBoard-xM. So, the contribution of the IoT Bridge cannot be calculated.

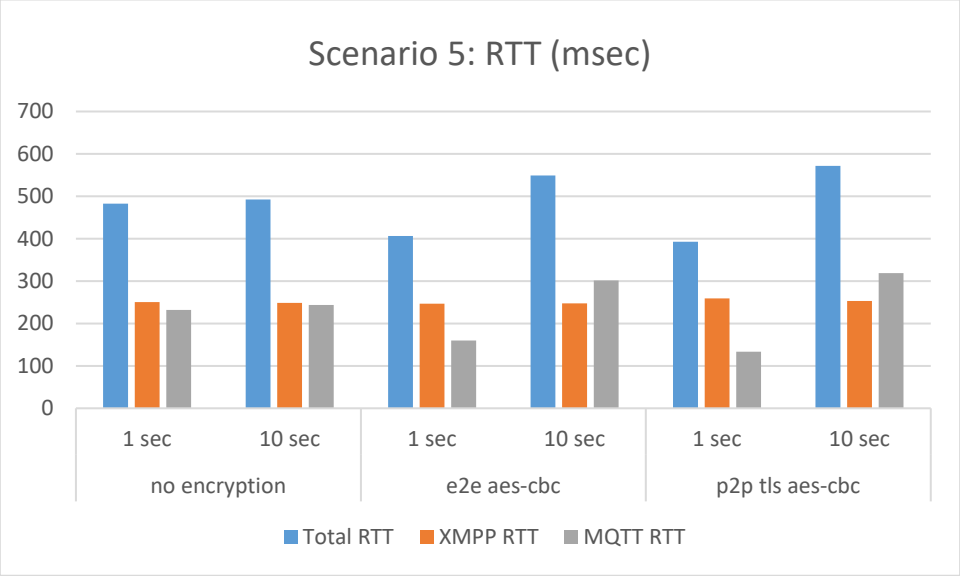


Total is the RTT as seen by the PC/Server and XMPP RTT is the RTT for the connection between the IoT Bridge and the XMPP device (including the XMPP server). It is clear that the IoT Bridge takes a notable portion of time to handle the request and the response.

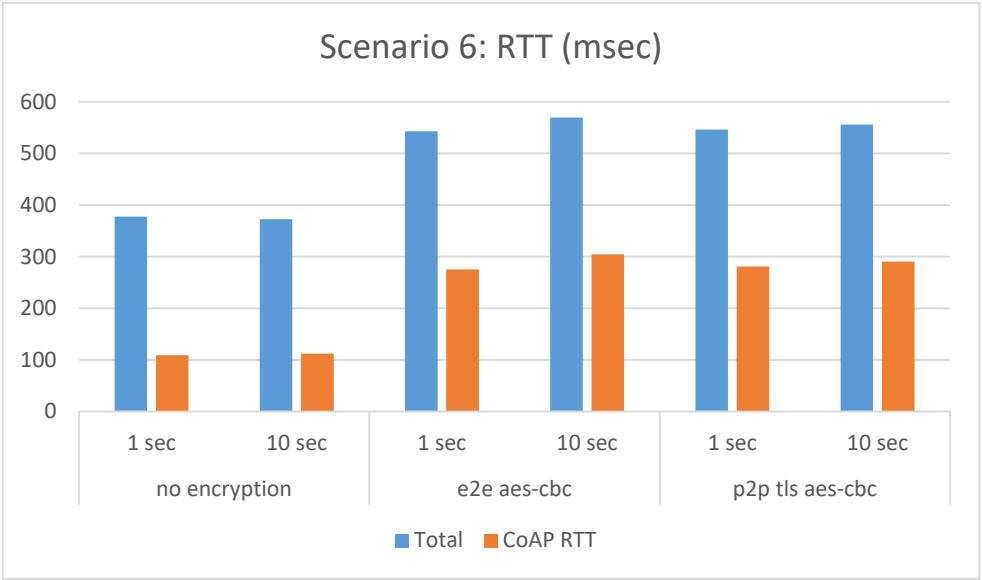


MQTT RTT is the time taken from the moment of *PUBLISH* till the *PUBACK* response, so it refers to the connection between the MQTT Z1 device and the MQTT Broker. CoAP RTT

refers to the connection between the IoT Bridge and the CoAP Z1 device. Total is the sum of the two.

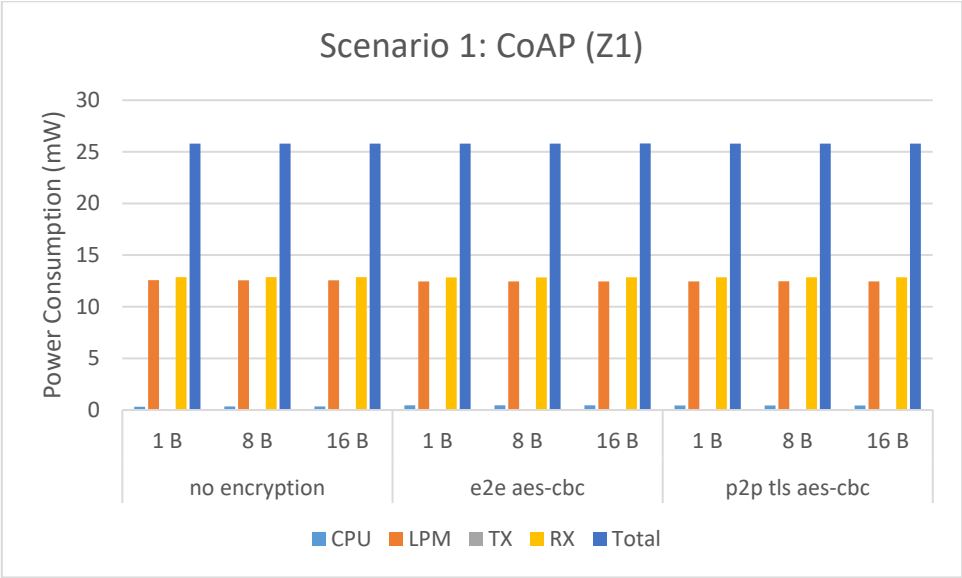


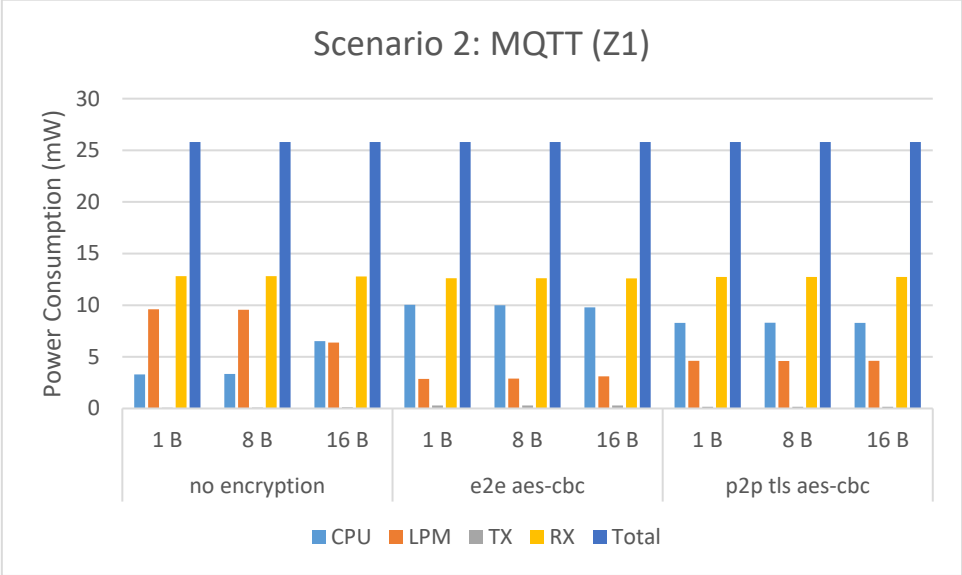
This scenario employs asynchronous protocols too. XMPP RTT is the time taken between the request from the XMPP device until it receives an acknowledgment that its request has been delivered and it is about to be processed. Thus, it refers to the following connections: XMPP Resource – XMPP Server – IoT Bridge, but it does not include the processing of the IoT Bridge. MQTT RTT is calculated the same way as in the previous scenarios. Total RTT is the sum of the two.

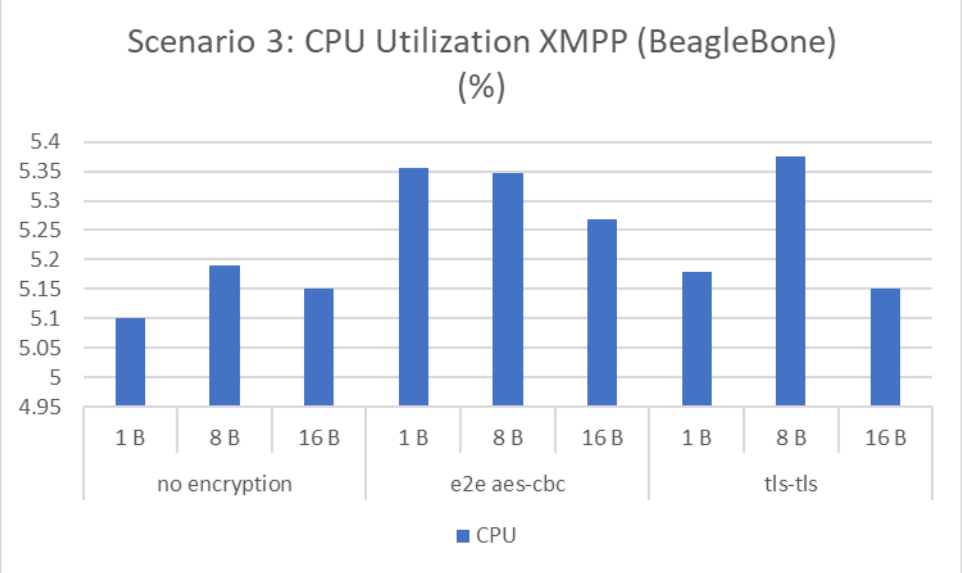


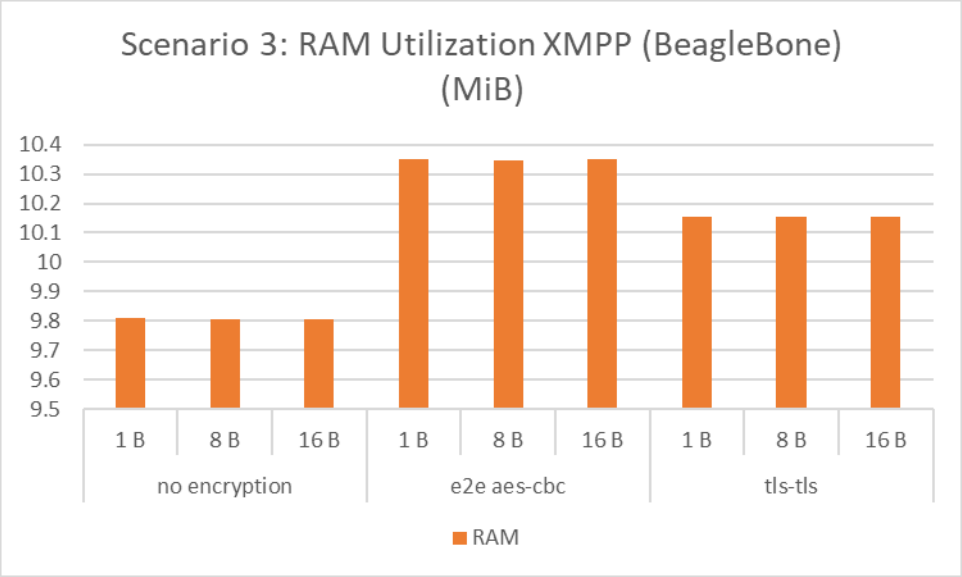
CoAP RTT refers to the connection between the IoT Bridge and the CoAP device, whereas the Total RTT is the total time, including the CoAP RTT, the processing time of the IoT Bridge and the connections: XMPP Resource – XMPP Server – IoT Bridge.

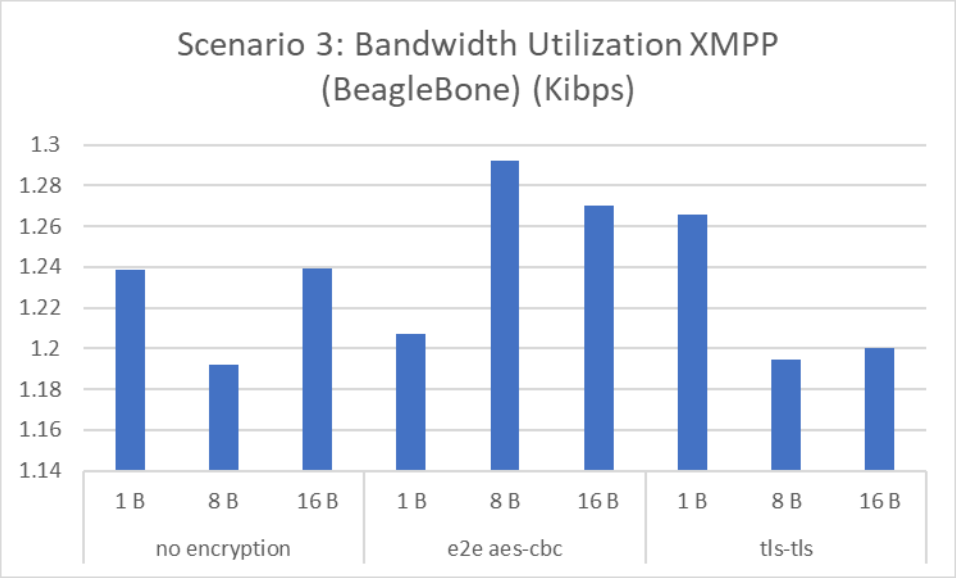
Measurements per device

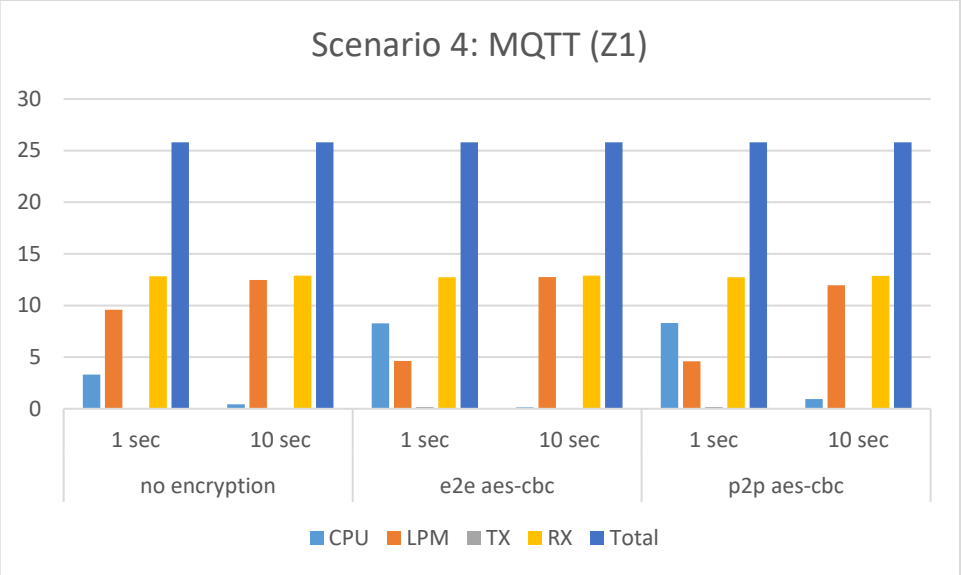


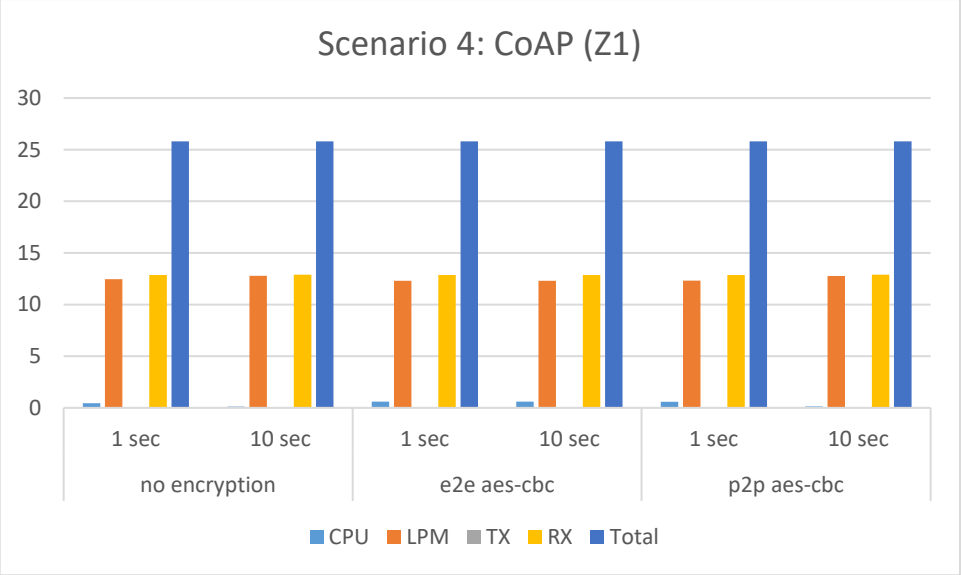


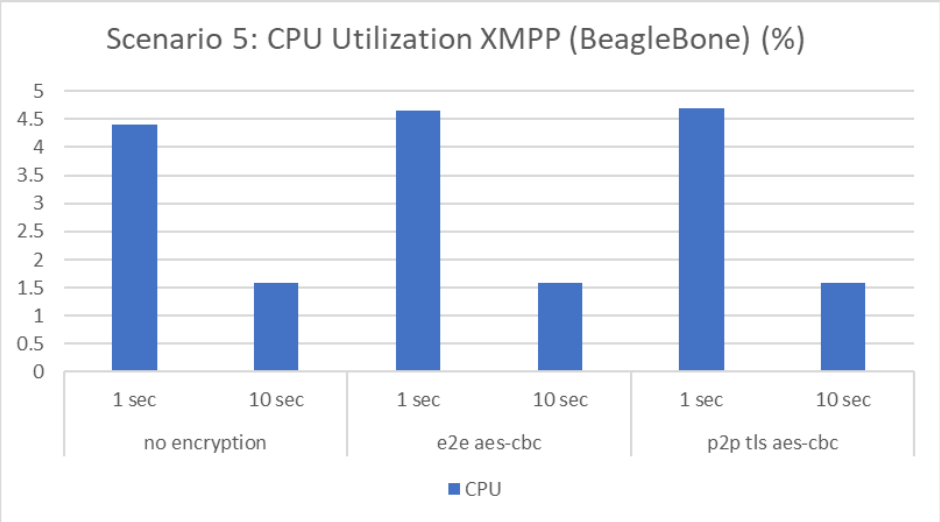


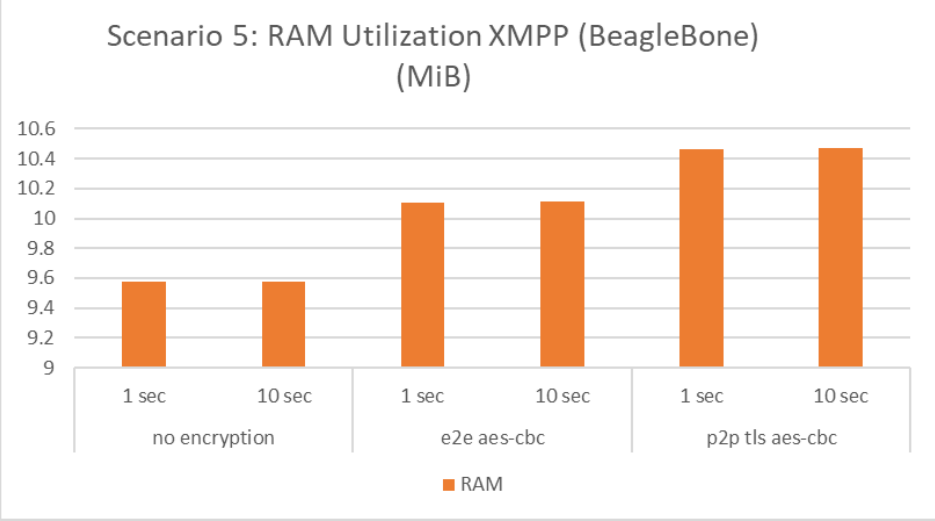

Scenario 3: RAM Utilization XMPP (BeagleBone) (MiB)
10.4
10.3
10.2
10.1
10
9.9
9.8
9.7
9.6
9.5
1 B
8 B
16 B
1 B
8 B
16 B
1 B
8 B
16 B
no encryption
e2e aes-cbc
tls-tls
RAM

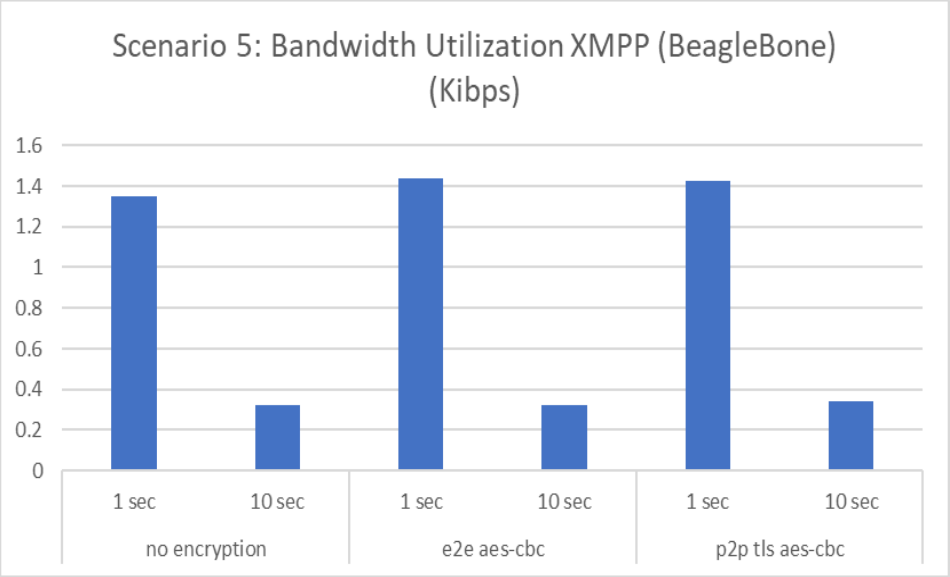

Scenario 3: Bandwidth Utilization XMPP (BeagleBone) (Kibps)
1.3
1.28
1.26
1.24
1.22
1.2
1.18
1.16
1.14
1 B
8 B
16 B
1 B
8 B
16 B
1 B
8 B
16 B
no encryption
e2e aes-cbc
tls-tls

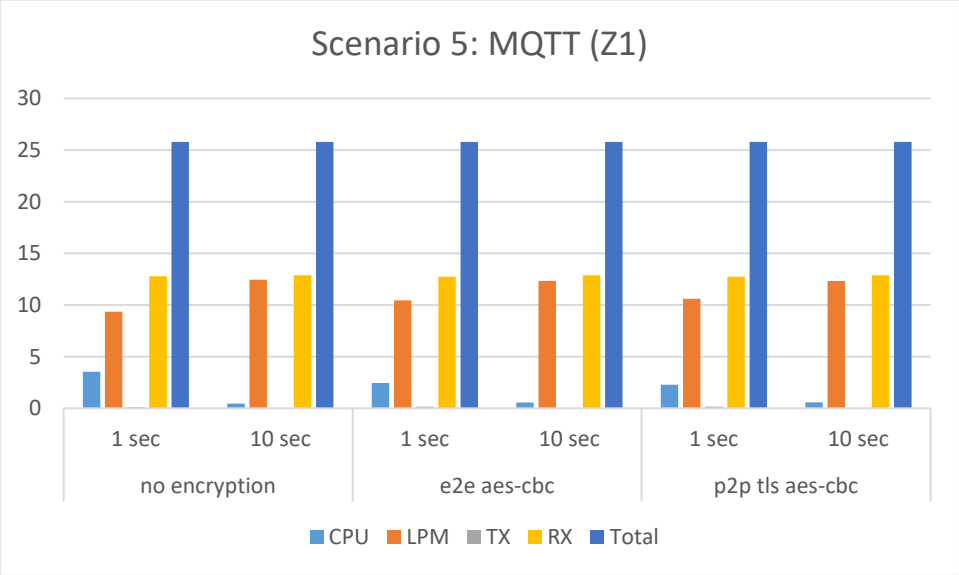

Scenario 4: MQTT (Z1)
30
25
20
15
10
5
0
1 sec
10 sec
1 sec
10 sec
1 sec
10 sec
no encryption
e2e aes-cbc
p2p aes-cbc
CPU
LPM
TX
RX
Total

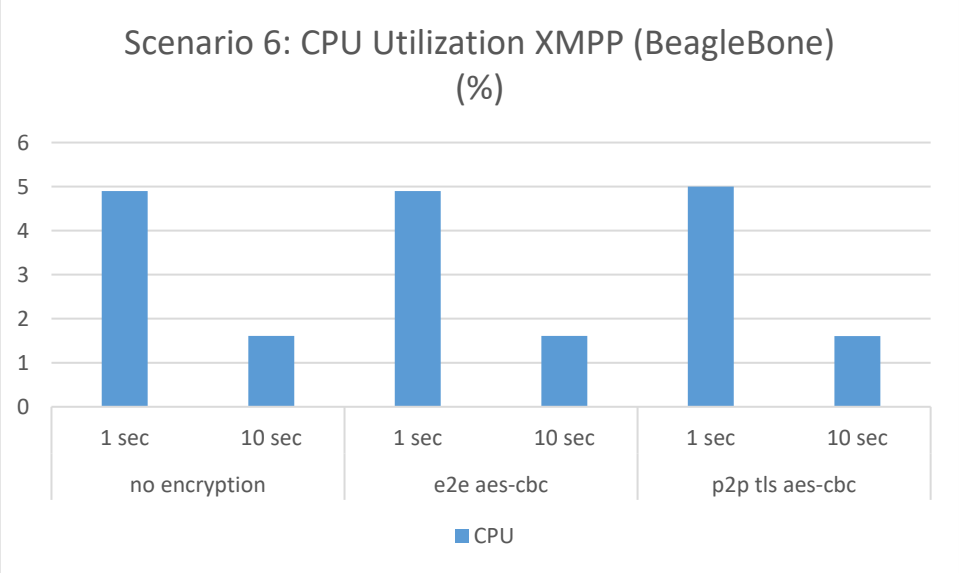

Scenario 4: CoAP (Z1)
30
25
20
15
10
5
0
1 sec
10 sec
1 sec
10 sec
1 sec
10 sec
no encryption
e2e aes-cbc
p2p aes-cbc
CPU
LPM
TX
RX
Total

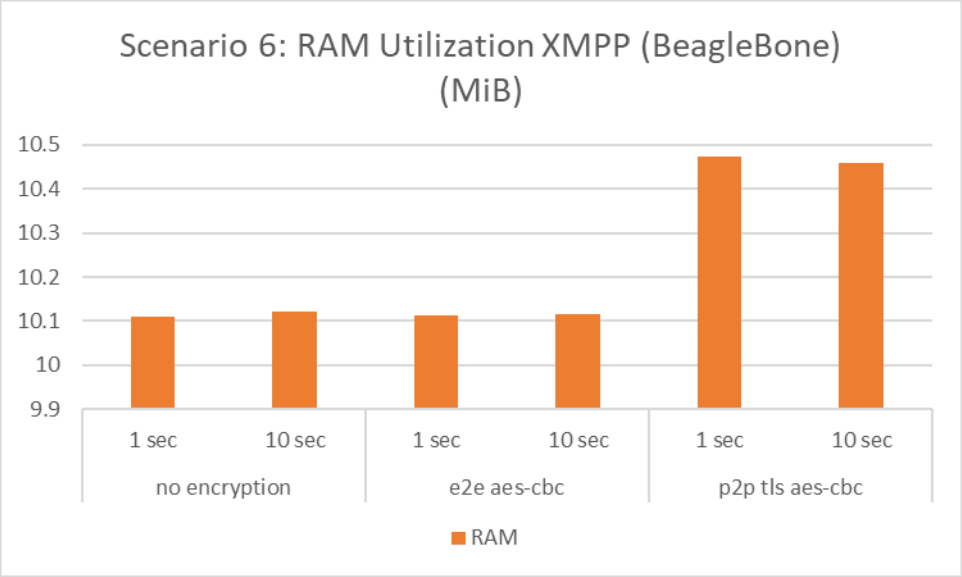

Scenario 5: CPU Utilization XMPP (BeagleBone) (%)
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
1 sec
10 sec
1 sec
10 sec
1 sec
10 sec
no encryption
e2e aes-cbc
p2p tls aes-cbc
CPU

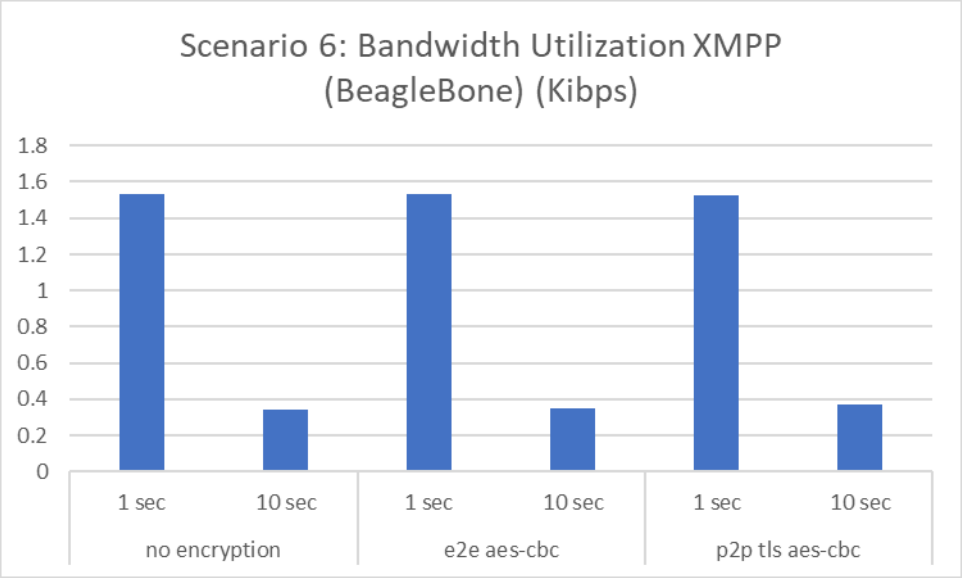

Scenario 5: RAM Utilization XMPP (BeagleBone) (MiB)
10.6
10.4
10.2
10
9.8
9.6
9.4
9.2
9
1 sec
10 sec
1 sec
10 sec
1 sec
10 sec
no encryption
e2e aes-cbc
p2p tls aes-cbc
RAM

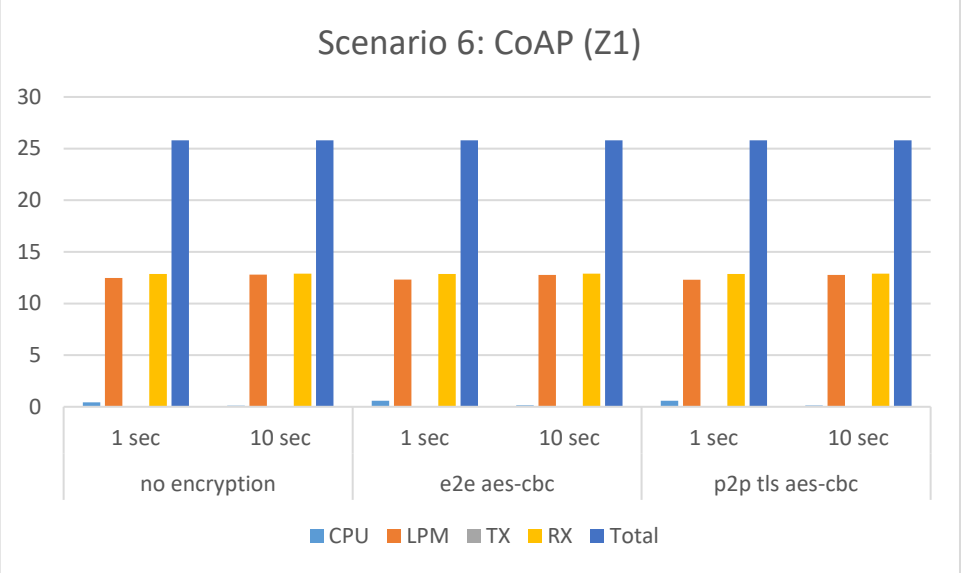

Scenario 5: Bandwidth Utilization XMPP (BeagleBone) (Kibps)
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
1 sec
10 sec
1 sec
10 sec
1 sec
10 sec
no encryption
e2e aes-cbc
p2p tls aes-cbc


Scenario 5: MQTT (Z1)
30
25
20
15
10
5
0
1 sec
10 sec
1 sec
10 sec
1 sec
10 sec
no encryption
e2e aes-cbc
p2p tls aes-cbc
CPU
LPM
TX
RX
Total


Scenario 6: CPU Utilization XMPP (BeagleBone) (%)
6
5
4
3
2
1
0
1 sec
10 sec
1 sec
10 sec
1 sec
10 sec
no encryption
e2e aes-cbc
p2p tls aes-cbc
CPU


Scenario 6: RAM Utilization XMPP (BeagleBone) (MiB)
10.5
10.4
10.3
10.2
10.1
10
9.9
1 sec
10 sec
1 sec
10 sec
1 sec
10 sec
no encryption
e2e aes-cbc
p2p tls aes-cbc
RAM


Scenario 6: Bandwidth Utilization XMPP (BeagleBone) (Kibps)
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
1 sec
10 sec
1 sec
10 sec
1 sec
10 sec
no encryption
e2e aes-cbc
p2p tls aes-cbc


Scenario 6: CoAP (Z1)
30
25
20
15
10
5
0
1 sec
10 sec
1 sec
10 sec
1 sec
10 sec
no encryption
e2e aes-cbc
p2p tls aes-cbc
CPU
LPM
TX
RX
Total