Master of science in Electrical Engineering with emphasis on
Telecommunication Systems
October- 2017

Performance analysis of Proxy based encrypted communication in IoT environments

Security and Privacy ~ Distributed systems security

Budda Shiva Tarun

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering with emphasis on Telecommunication Systems. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author(s):
Budda Shiva Tarun
E-mail:
bstarun123@gmail.com
shbu16@student.bth.se

University advisor:
Dr. Dragos Ilie
Assistant Professor of Telecommunication Systems
Department of Computer Science and Engineering

University Examiner:
Dr. Adrian Popesco
Professor of Telecommunication Systems

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

ABSTRACT

Context: The Internet of Things is an emerging technology which has made our life easier. IoT refers to network of the physical devices that are reachable over Internet. IoT environment use Low power wireless Private Area Network (6LOWPAN) which enables IoT devices to interoperate in a standard way. Security among these devices is an important concern that needs to be addressed. There is no standard way to protect end-to-end communication among IoT devices. The major drawback of the IoT devices is that they are resource constrained and are unable to run the fully featured IP stack. To address this, IETF defined Constrained Application Protocol(COAP) that includes HTTP functionalities. Datagram Transport Layer Security (DTLS) can protect the COAP traffic. There are other solutions based on the IPsec and Host Identity protocol(HIP). This thesis focusses on the performance evaluation of spliced TLS/DTLS and IPsec/DTLS protocols in an IoT environment.

Objectives: The objectives of this research are:

- To implement proxy-based security protocols (for example, TLS, DTLS and IPsec) in an emulated IoT environment.
- To extract performance metrics such as CPU Utilization, memory utilization, elapsed time, network overhead for spliced TLS/DTLS, spliced IPsec/DTLS, spliced TLS/plain, Spliced IPsec/plain.
- Compare spliced TLS/DTLS, spliced IPsec/DTLS based on protocol features and statistical analysis of the data.

Methods: A virtual network is set up in the Linux kernel which consists of the server, client and the border router. 6LoWPAN devices are emulated in the Linux kernel in separate namespace where COAP servers are deployed. A HTTP-COAP proxy is deployed on the border router which is also emulated in the separate namespace. The TLS client in the global namespace invokes the server methods (GET, POST, PUT) for an operation to be performed. Secure communication is established among these devices using the TLS/DTLS, IPsec/DTLS protocols. Tests are performed by implementing TLS/DTLS, IPsec/DTLS various performance metrics like CPU utilization, memory utilization, network overhead, elapsed time are calculated for the various tests that are considered. Statistical analysis is done the performance metrics that are considered.

Results: Based on the tests performed and the performance metrics extracted, statistical analysis is done. CPU Utilization, Memory Utilization, Network Overhead and elapsed time for spliced TLS/DTLS, spliced IPSec/DTLS, spliced TLS/Plain, Spliced IPSec/DTLS are calculated respectively and the corresponding graphs are plotted to give the better picture visually. Limitations of implementing the spliced protocols in the constrained devices are provided.

Conclusions: From this research based on the limitations and implementation of protocols TLS/DTLS protocols has better performance in terms of CPU Utilization, memory utilization, network overhead when compared to the spliced IPsec/DTLS

Keywords: DTLS, IoT, IPsec, Performance, Security.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor Dr. Dragos Ilie for his support and encouragement during the entire course work. It was a great experience to work under his supervision. His immense knowledge and constant supervision makes him a great teacher.

I would like to thank Dr. Adrian Popescu for conducing and managing this course efficiently.

I would also like thank my partner Kuna Vignesh for the constant support and encouragement during the course work. He stood on behalf of me in the hard times which I faced during this thesis work.

I am very grateful to my parents and my brother for the constant support and encouragement. They thought me the value of the hard-work, Discipline in my life. Without their encouragement and motivation, I may not be here today.

I would like to extend my deep sense of gratitude to all my friends and the faculty of the BTH.

Lastly, I would like to thank the almighty for blessing with good health during my stay in the Sweden.

.

Contents

ABSTRACT ..... **I**

ACKNOWLEDGEMENTS ..... **II**

LIST OF FIGURES ..... **V**

1 INTRODUCTION **1**

1.1 INTERNET OF THINGS 1
1.2 SECURITY IN IOT 3
1.3 PRIVACY 4
1.4 IOT SECURITY FRAME WORK 4
1.5 IOT NETWORK MODEL 5
1.6 MOTIVATION 5
1.6.1 Problem Statement *5*
1.6.2 Aims and Objectives *6*
1.6.3 Research Questions *6*
1.7 OUTLINE OF DOCUMENT 7
1.8 SPLIT OF WORK 7

2 STATE OF ART **9**

3 BACKGROUND **18**

3.1 LOW POWER WIRELESS PRIVATE AREA NETWORK (6LOWPAN) 18
3.1.1 Headers *18*
3.1.2 Addressing Format *19*
3.1.3 Routing *20*
3.2 CONSTRAINED APPLICATION PROTOCOL (COAP) 20
3.2.1 Message ID *20*
3.2.2 Request and Response *21*
3.2.3 Piggy backed Response *21*
3.2.4 Separate Response *21*
3.2.5 GET *21*
3.2.6 POST *22*
3.2.7 PUT *22*
3.2.8 DELETE *22*
3.2.9 Message Format *22*
3.2.10 Option Values *23*
3.2.11 Message Translation *24*

4 SECURITY PROTOCOLS **25**

4.1 INTERNET PROTOCOL SECURITY (IPSEC) 25
4.1.1 Introduction *25*
4.1.2 Functionality *25*
4.1.3 Authentication Header *26*
4.1.4 Encapsulating Service Pay Load *26*
4.1.5 Services *27*
4.1.6 UDP Encapsulation of ESP Packets *27*
4.1.7 Modes of Operation *28*
4.1.8 Public Key Cryptography *28*
4.1.9 StrongSwan *29*
4.2 DATAGRAM TRANSPORT LAYER SECURITY (DTLS) 29
4.2.1 Providing Reliability For Handshake *30*
4.2.2 Packet Loss *30*
4.2.3 Reordering *30*
4.2.4 Message Size *31*
4.2.5 Replay Detection *31*
4.2.6 DTLS Handshake Protocol *31*
4.2.7 Hand-shake Message Fragmentation and Reassembly *33*

4.2.8 Security Analysis ... *33*
4.3 TRANSPORT LAYER SECURITY (TLS) ... 33
4.3.1 TLS Record Protocol ... *33*
4.3.2 TLS Handshake Protocol ... *34*
4.3.3 Advantages of TLS ... *34*
4.4 HTTP-COAP PROXY ... 34
4.4.1 Congestion Control ... *36*
4.4.2 Types of Proxies ... *36*
4.4.3 Cross Protocol URI Mapping ... *37*
4.4.4 Message Layer ... *37*
4.4.5 Operations ... *38*
4.4.6 Caching ... *38*
4.4.7 IPv4 and IPv6 ... *39*

5 METHODOLOGY ... 40

5.1 OMNET++ ... 40
5.2 CONTIKI ... 40
5.3 COOJA ... 41
5.4 RISK STATE ... 41
5.5 ALTERNATIVE CHOSEN ... 41
5.6 NETWORK NAMESPACES ... 41
5.7 LINUX WPAN PROJECT ... 42
5.8 WPAN-TOOLS ... 42
5.8.1 PAN ID ... *42*
5.9 IOT LOGICAL NETWORK ARCHITECTURE: ... 42
5.10 IMPLEMENTATION ... 43
5.10.1 Test Bed ... *43*
5.10.2 IPSec VPN ... *45*
5.10.3 DTLS ... *45*
5.11 TESTS PERFORMED ... 46

6 RESULTS AND ANALYSIS ... 47

6.1 PERFORMANCE METRICS ... 48
6.1.1 CPU Utilization ... *48*
6.1.2 Memory Utilization ... *48*
6.1.3 Network Overhead ... *48*
6.1.4 Elapsed Time ... *49*
6.2 TEST1 ... 49
6.2.1 Scenario 1 ... *49*
6.2.2 Scenario 2 ... *51*
6.3 TEST2 ... 53
6.3.1 Scenario 1 ... *53*
6.3.2 Scenario 2 ... *55*
6.4 TEST3 ... 57
6.4.1 Scenario 1 ... *57*
6.4.2 Scenario 2 ... *59*
6.5 TEST4 ... 61
6.6 TEST5 ... 63
6.7 DTLS LIMITATIONS ... 65
6.8 IPSEC LIMITATIONS ... 66

7 CONCLUSIONS AND FUTURE WORK ... 68

7.1 CONCLUSIONS ... 68
7.2 FUTURE WORK ... 68

REFERENCES ... 69

8 APPENDIX ... 72

8.1 IPSEC IMPLEMENTATION ... 72
8.2 TEST6 ... 72

LIST OF FIGURES

Figure 1 Device to Gateway Communication....2
Figure 2 Back-end Data-Sharing Communication....3
Figure 3 Security Framework....4
Figure 4 IoT Network Model....5
Figure 5 6LoWPAN addressing header....18
Figure 6 6LoWPAN Addressing format....19
Figure 7 COAP Message Format....23
Figure 8 IPv6 packet encapsulated using AH in transport mode....26
Figure 9 IPv6 packet encapsulated using AH in tunnel mode....26
Figure 10 IPv6 encapsulated ESP header in transport mode....26
Figure 11 IPv6 encapsulated ESP header in tunnel mode....27
Figure 12 UDP- Encapsulated Transport mode header....27
Figure 13 UDP- Encapsulated tunnel mode header....27
Figure 14 DTLS Architecture and DTLS Handshake....31
Figure 15 HTTP-COAP Proxy Deployment Scenario....35
Figure 16 IoT Logical Network Architecture....43
Figure 17 Scenario1....47
Figure 18 Scenario2....47
Figure 19 CPU Utilization at server and proxy for HTTP-GET....49
Figure 20 Memory Utilization at server and Proxy for HTTP-GET in scenario1....50
Figure 21 Network Overhead at the server and proxy for HTTP-GET in scenario1....50
Figure 22 Elapsed Time at server and proxy for the HTTP-GET in scenario1....51
Figure 23 CPU Utilization at the server and proxy for HTTP-GET in scenario2....51
Figure 24 Memory Utilization at server and proxy for HTTP-GET in scenario2....52
Figure 25 Network overhead at server and proxy for HTTP-GET in scenario2....52
Figure 26 Elpased time at the server and proxy for HTTP-GET in scenario2....53
Figure 27 CPU Utilization at server and the proxy for HTTP-POST in scenario1....53
Figure 28 Memory Utilization at the server and proxy for HTTP-POST....54
Figure 29 Network overhead at the server and proxy for HTTP-POST....54
Figure 30 CPU Utilization at the server and proxy for HTTP-POST secnario2....55
Figure 31 Memory Utilization at the server and the proxy for HTTP-POST for scenario2....56
Figure 32 Network overhead at server and proxy for HTTP-POST....56
Figure 33 Elapsed time at server and proxy for HTTP-POST....57
Figure 34 CPU utilization at the serer and the proxy for HTTP-PUT....57
Figure 35 Memory utilization at the server and proxy for HTTP-PUT....58
Figure 36 Network overhead at server and proxy for HTTP-PUT....58
Figure 37 CPU utilization at server and proxy for HTTP-PUT in scenario2....59
Figure 38 Memory utilization at server and proxy for HTTP-PUT in scenario2....60
Figure 39 Network Overhead at server and proxy for HTTP-PUT in scenario2....60
Figure 40: Memory utilization at proxy for test-4....61
Figure 41: CPU Utilization at proxy for test-4....62
Figure 42: Elapsed Time at proxy for Test-4....62
Figure 43: Network Overhead at proxy for Test-4....62
Figure 44: Memory Utilization at proxy for Test-5....63
Figure 45: CPU Utilization at proxy for Test-5....63
Figure 46:Elapsed Time at proxy for Test-5....63
Figure 47: Network Overhead at proxy for test-5....64
Figure 48: CPU Utilization at proxy for Test-6....73
Figure 49: Memory Utilization at proxy for test-6....73
Figure 50: Elapsed Time at proxy for Test-6....73
Figure 51: Network overhead ar proxy for Test-6....74

LIST OF ABBREVIATIONS

AES- Advance Encryption Standard
AH- Authentication Header
CPU- Central Processing Unit
CoAP- Constrained Application Protocol
DTLS- Datagram Transport Layer Security
ESP-Encapsulating Security Payload
HTTP-Hyper Text Transfer Protocol
HIP- Host Identity Protocol
IETF- Internet Engineering Task Force
IoT- Internet of Things
IKEv2: Internet Key Exchange Protocol Version 2
IPSec- Internet Protocol Security
IP- Internetworking Protocol
IPv6- Internetworking Protocol version 6
M2M- Machine to Machine
TLS- Transport Layer Security
LLN- Low Power Lossy networks
6LoWPAN- Low Power Wireless Personal Area Network
REST- Representational State Transfer
RAM- Random Access Memory
ROM-Read Only Memory
UDP- User Datagram Protocol
WPAN-Wireless Personal Area Network

1 INTRODUCTION

1.1 Internet of Things

The concept of Internet of things was invented by Peter T. Lewis in September 1985 in a speech delivered at a U.S. Federal Communications session. It is an emerging trend of technical, social and economic significance. IoT is the inter-networking of smart devices, buildings and vehicles embedded with electronics, sensors, actuators and network connectivity that enable these objects to generate, collect and exchange data. Things in IoT sense include hardware, software, data and service and vary from devices for heart monitoring, biochip transponders on farm animals, electric clams in coastal waters, automobiles with built in sensors, devices for rescue operation, and DNA analysis devices.

Projections for the impact of IoT on the internet and economy are impressive, and researchers are anticipating as many as 100 billion connected IoT devices and a global economic impact of $11 trillion by 2025. Although, equally rapid development is happening in the services that are used to monitor and control the IoT devices. We can say, these are universally built on Internet technology and are most commonly implemented using web services. It is the combination of Internet connected embedded devices and Web based services that makes the IoT a very powerful paradigm. The potential of Internet of things relies on policies that respect privacy choices, privacy rights which poses a need to ensure privacy expectations for user trust and confidence in the Internet and the connected devices. Mobile phones as anyone can say have become almost universally enabled IP embedded devices making up the largest segment of devices part of IoT. IoT may force a shift of the popular notion of what it means to be "on the internet" where passive interaction with the internet will dominate and the outcome being a "hyperconnected world". In passive interaction, the devices are not always active and could be brought to life using some other connected device. It's not an easy thing because we must ensure third party applications have easy access to data generated.

IoT devices operate in 4 modes:

- **Device-to-Device communication**: The communication is done directly and mostly used in home automation systems.

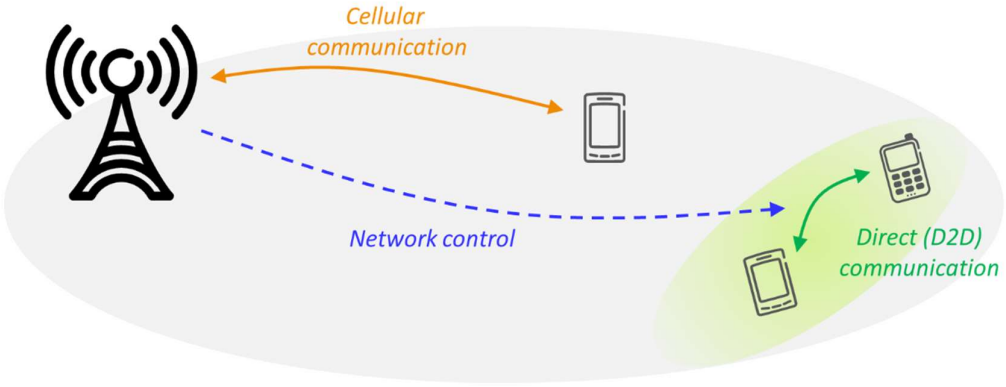


Figure 1 Device to Device communication

- **Device-to-Cloud Communication**: The device is connected to a router or Ethernet and that ultimately connects to the cloud service provider.

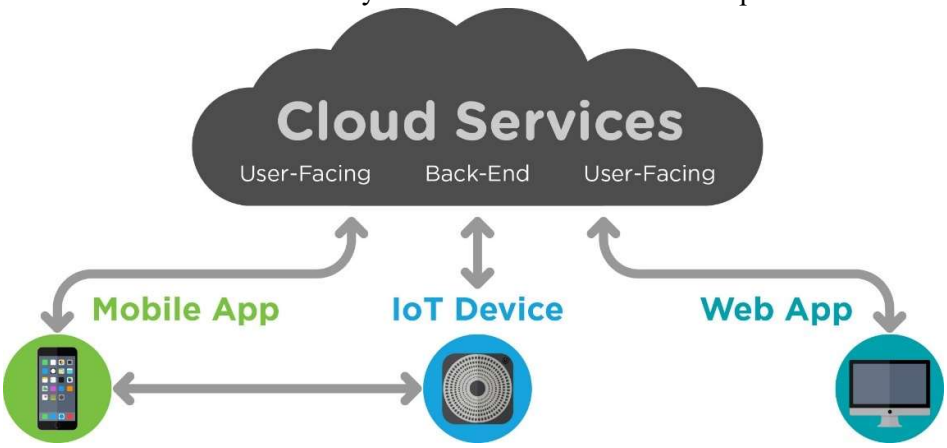


Figure 2 Device to cloud communication

- **Device-to-Gateway Communication**: The application level gateway is being used where gateway acts as an intermediary between the device and the cloud service providing all the functionalities.

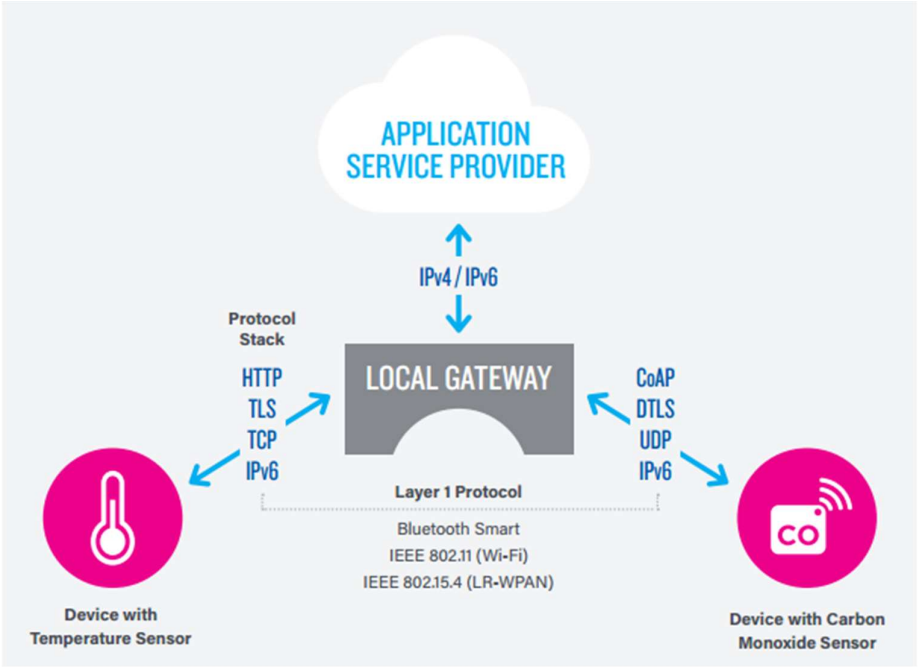


Figure 1 Device to Gateway Communication

- **Back-end Data-Sharing Communication**: Data is exported and analyzed from cloud services in combination with data from other sources.

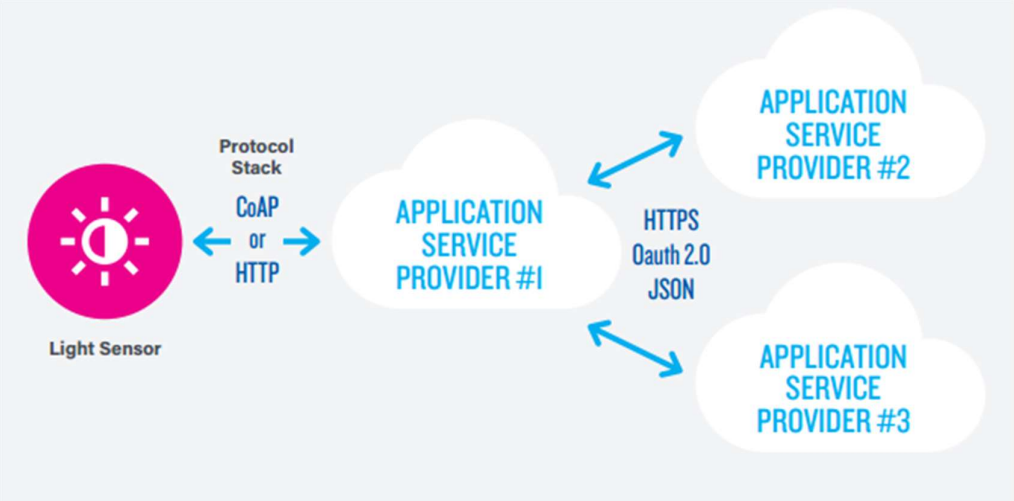


Figure 2 Back-end Data-Sharing Communication

Effective IoT communication models act as catalysts for technical innovation and improve the opportunity for commercial growth. The key aspects to be addressed are security, privacy, interoperability and standards, legal, regulatory and rights and emerging economies and development. REST is the suitable architecture that allow things to communicate over Hypertext Transfer Protocol and easily adoptable for IoT applications to provide communication from a device to a central server.

1.2 Security in IoT

The IoT devices should have open access to all data consumers and controllers. Privacy and data integrity must be ensured while retaining the isolation of the information among the several consumers. Security in IoT devices is complex and can be deployed on a platform with potentially limited resources. Security architecture of the IoT environments should address the major issues.

- Device identity and authentication to several networks securely
- Maintain availability of the data or the service.

The threats in IoT environments are like that of traditional IT environments but the impact is different. Many security considerations in IoT protocols rely on encryption. The authentication and authorization protocols are complex and require high computational resources which a IoT device lacks. These protocols also require the user intervention in terms of configuration and provisioning thus the initial configuration must be protected from tampering and other forms of compromise throughout the entire lifecycle of the device. The protocol should also be delay tolerant. The other element in the security of the IoT include strong identities and strengthening of network centric methods like DNS with DNS sec and DHCP to prevent attacks. The communication and data transport channels should be secured to provide bi-directional communication

1.3 Privacy

Privacy is an important concern in securing the IoT networks. IoT networks generate the traceable signature of the device and behavior of the end users. These issues are more relevant in healthcare applications. Hence it is essential to verify the device ownership and owner's identity else it may lead to pitfalls. A mechanism called shadowing is being proposed which provides the digital shadows of that enable the user objects to act on their behalf, storing a virtual identity that contains the information about their attributes. Identity management in the IoT may offer new opportunities to increase security by combining diverse authentication methods for humans and machines.

1.4 IoT Security Frame Work

To address the security challenges related to the highly diverse IoT environments a security framework is proposed by IETF
A secure IoT framework mainly consists of the four basic elements

1. Authentication
2. Authorization
3. Network Enforced Policy
4. Secure Analytics: Visibility and control
5.

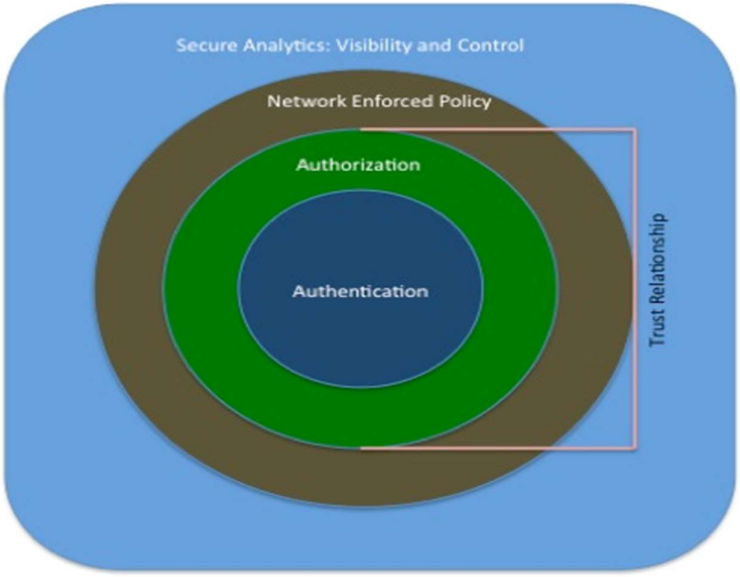


Figure 3 Security Framework

Authentication: The authentication layer used to provide the identity information of IoT entity. When connected the IoT device needs to access the IoT infrastructure, trust relation needs to be initiated based on the identity of the device. The way to store and present the information differ from device to device.
Authorization: The second layer of this framework controls the access of the device throughout the network fabric. It builds up a core authentication layer by leveraging the identity information of an entity. With authentication and authorization, a trust relationship is established between the IoT devices to exchange the information.

Network Enforced Policy: This layer includes the elements that route and transport the end -point traffic securely over the IoT infrastructure.
Secure analytics: Visibility and Control: This layer defines the services by which all the elements are should provide the telemetry for gaining visibility and gaining the control over the entire IoT ecosystem. Further it includes all the elements that provides that aggregate and correlate the information to provide reconnaissance and threat detection.

1.5 IoT Network Model

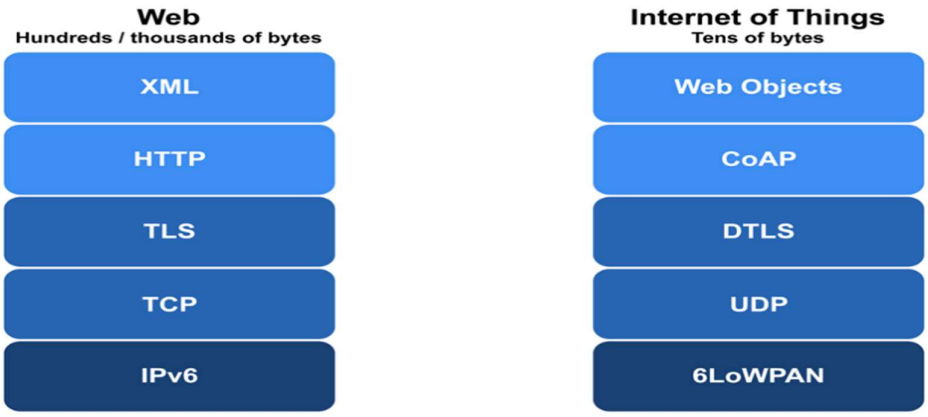


Figure 4 IoT Network Model

Most IoT devices use IEEE.802.15.4 standard which specifies operation of low rate wireless personal area networks. It operates on physical and media access control layers of 6LoWPAN network.
TCP has more overhead and consumes more energy which is not desirable in IoT environment as IoT devices are resource constrained and generate real time data. So, the much lightweight UDP is used.
Embedded IoT devices are often incapable of running full IP stack, and therefore IETF designed a lightweight protocol called Constrained application protocol (CoAP) which include HTTP functionalities required for IoT devices.
DTLS and IPsec protocols provide security for CoAP communication.

1.6 Motivation

1.6.1 Problem Statement

The communication in modern IoT environments is based on the IEEE 802.15.4 standard. The IETF has defined a scaled-down version of the IPv6 protocol known as Low Power Wireless Private Area Networks (6LoPWAN). The purpose of 6LoWPAN is to enable 802.15.4 devices to interoperate in a standardized way with devices using other communication standards and with conventional IPv6 nodes.

Although 802.15.4 devices use the AES-128 block cipher to encrypt hop-by-hop communication within the IoT network, there is no standardized approach to protect end-to-end communication in particular in the case when one communication end is located in the 802.15.4 network and the other is placed in the Internet. This case is

quite common in scenarios where data from sensor networks is collected into remote clouds for advanced data mining.

In addition to pure data collection, cloud entities may engage in machine-to-machine (M2M) communication. It is therefore quite useful if RESTful web services frameworks from conventional systems can be reused for IoT. The major road block in doing this is that many IoT devices are resource-constrained and thus unable to run a fully-featured IP stack and much less support a regular HTTP implementation. To address this problem, IETF has defined the Constrained Application Protocol (CoAP) that includes HTTP functionalities adequate for IoT environments (e.g., CoAP data is transported over UDP).

Just like Transport Layer Security (TLS) can provide confidentiality for HTTP, Datagram Transport Layer Security (DTLS) can protect CoAP traffic. However, IETF does not mandate the use of DTLS and therefore other solutions based on IPsec and Host Identity Protocol (HIP) have been proposed as well. The focus for this thesis is to compare the performance of proxy-based communication with (D)TLS and IPsec in an IoT environment

1.6.2 Aims and Objectives

The aim of this work is to investigate the performance of proxy-based encrypted communication with an IoT environment. The proxy, typically running in a 6loWPAN border router, splices the connection from a host on the Internet and the router with the connection between the router and the destination IoT device. Each of the two spliced connections can use a different type of security protocol, for example TLS on the first connection and DTLS on the second. The objectives of this project are:

- To implement proxy-based security protocols (for example, TLS, DTLS and IPsec) in an emulated IoT environment.
- To extract performance metrics such as elapsed time, CPU and memory utilization for TLS, DTLS and IPsec.
- Compare TLS and IPsec based on protocol features and statistical analysis of the data.

1.6.3 Research Questions

1. What is the performance overhead in implementing Spliced TLS/DTLS compared to spliced TLS/plain-text communication with the IoT environment and what are its limitations?
2. What is the performance overhead in implementing spliced IPsec/DTLS communication and when compared to spliced IPSec/plain-text communication with the IoT environment?
3. Is there a significant difference in performance overhead between TLS/DTLS and IPsec/DTLS?
4. Based on performance overhead and implementation challenges which protocol combination is best suited to provide confidentiality?

1.7 Outline of Document

- Chapter 1 i.e., this chapter consists of the basic overview of the thesis along with the background knowledge necessary including IoT, IPsec, DTLS, HIP. It also includes the motivation for the thesis, the problem statement, research questions, Aims and Objectives.
- Chapter 2 deals with the state of Art which tells us about the existing works done related to IoT as a part of the Literature study.
- Chapter 3 deals with the methodology adopted in our thesis to build the test bed to perform the tests necessary for the thesis.
- Chapter 4 shows all the results recorded during the experiments performed and statistical analysis of the obtained results.
- Chapter 5 gives the conclusion i.e. answers to the research questions obtained during the thesis and possible future research that can be done to extend the existing thesis work.
- Chapter 6 provides the references which were helpful during the thesis.
- Chapter 7 ends the document with the Appendix section.

1.8 Split of Work

Section	Topic	Contributor
1.Introduction	1.1 Internet of Things	Vignesh Kuna
	1.2 Security in IoT	Shiva Tarun
	1.3 Privacy	Shiva Tarun
	1.4 IoT Security Framework	Vignesh Kuna
	1.5 IoT Network Model	Shiva Tarun
	1.6 Motivation	Vignesh Kuna Shiva Tarun
	1.7 Outline of the document	Vignesh Kuna
2. State of Art	Survey of Related Works	Vignesh Kuna Shiva Tarun
3. Background	3.1 Low Power Wireless Private Area Network	Shiva Tarun
	3.2 Constrained Application Protocol	Vignesh Kuna
4. Security Protocols	4.1 Internet Security Protocol	Shiva Tarun Vignesh Kuna
	4.2 Datagram Transport Layer Security	Shiva Tarun Vignesh Kuna
	4.3 Transport Layer Security 4.4 HTTP- COAP Proxy	Shiva Tarun

5. Methodology	5.1 Network Namespaces	Shiva Tarun
	5.2 Linux-wpan Project	Shiva Tarun
	5.3 Wpan tools	Vignesh Kuna
	5.4 Test bed	Vignesh Kuna Shiva Tarun
	5.5 Implementation of Protocols	Vignesh Kuna Shiva Tarun
	5.6 Tests Performed	Shiva Tarun
	5.7 OMNeT++	Shiva Tarun
	5.8 Contiki OS	Vignesh Kuna
	5.9 Cooja Simulator	Vignesh Kuna
	5.10 Risk State	Shiva Tarun Vignesh Kuna
	5.11 Alternative Chosen	Shiva Tarun Vignesh Kuna
6. Results and Analysis	6.1 Performance Metrics	Shiva Tarun
	6.2 Tests Performed	Shiva Tarun
7.Conclusions and Future Work	7.1 Conclusions	Shiva Tarun
	7.2 Future Work	Shiva Tarun

2 STATE OF ART

Securing IoT: A standardization perspective

In paper[1], the state of art of about security in IoT is given. A detailed description about the standard security protocols to be used in IoT is provided. IP router makes interoperability possible. IETF has standardized 6LoWPAN. Different security protocols that can be adopted in DTLS have been described. IPsec and HIP were also discussed. The method of DTLS handshakes occurred and the advantages were told. Existing cipher suites are being discussed such as symmetric ciphers which uses AES-CCM. The RAM and ROM being consumed increased for DTLS. Community is working on a single security protocol suite which is based on DTLS to provide the security for IoT devices. Research is being done on reducing the complexity of using DTLS and optional features which can be avoided. The work provided us the brief overview of DTLS and feasibility of its usage in providing security in IoT environments.

IPv6 Security Tools: A systematic Review:

The authors have given an overview on the tools available for testing IPv6 security through penetration testing which is a good way of ensuring security for the system in[2] Internet Protocol version 6 (IPv6) is a new routing protocol which has been deploying dramatically over the past years. This is introduced by IETF to overcome some of IPv4 limitations. People believe that IPv6 is secure more than IPv4, which this idea is not correct It is important to know that it should be assessed regularly. The best way to defense is to look at security as a hacker by penetration testing. This paper covers and reviews some of the fundamental penetration testing and monitoring tools. Moreover, we provided a collection of tools that is not available in any literature review. They concluded with some of the tools best suitable such as nmap, THC-IPv6, SI6 and wireshark. The work has provided the brief overview of the various tools that are available for testing and monitoring purpose which we initially thought of considering them for the performance evaluation.

Securing Diameter: Comparing TLS, DTLS, IPSec

The authors provided a comparative study on TLS, DTLS and IPsec on securing diameter in [3]. They have considered transmission header, connection establishment and processing overhead. They have concluded saying TLS has least number of Round Trip Times(RTTs) but has more processing overhead. DTLS has the least processing overhead and manageable number of RTTs.

This research has provided us the various performance aspects that needs to be taken in consideration in the comparing the various security protocol. The work compared the three protocols in terms of number of RTTs and connection establishment time and processing delays. As these parameters are already evaluated, hence we have decided to evaluate the security protocols in terms of other performance metrics (CPU utilization, Memory utilization, network overhead, elapsed time) in various different scenarios (i.e, in end-to-end and proxy) in an emulated IoT environment**.**

Security Analysis of IoT protocols: A focus on COAP

The authors have done a research on security of CoAP over DTLS providing some solutions in [4]. They provided an overview of existing techniques of security for physical, MAC, and network layers. CoAP has support for M2M requirements in constrained environments, UDP binding with support for unicast and multicast requests, asynchronous message exchanges, low message overhead, parsing complexity and supports URI, has simple proxy and caching capabilities. There are different implementations of CoAP such as Californium, Erbium, jCoAP, LibCoAP exist. Different models of CoAP security are NoSec, PresharedKey, RawPublickey and certificates. Key management is also an issue to be looked after in CoAP.

The work provided us the information regarding the various implementations of CoAP protocol and their advantages and disadvantages. The work also provided the a feasibility study of implementing the security protocols on the existing CoAP implementations.

Security of IP based IoT: HIP and DTLS

In [5], in-order to ensure the security of the devices in the network the interaction of the devices must be regulated during entire life cycle. When a device joins an IoT network, the IoT network has to authorize its joining, such that it is provisioned and configured with the corresponding operational parameters, thus providing network access. During operation, devices will interact with each other requiring pairwise session keys, and for that, key management is needed
To address the above issue the authors proposed two security architectures by adapting the existing IP protocol and while relying on the single protocol. The first is based on the HIP protocol which uses pre-shared keys while the second is based on the DTLS protocol.

This particular research has detailed description of implementing the DTLS based security solution in IoT environment.

Cross- platform protocol development based on OMNeT++

In[6], the Authors have motivated the need for simulation and proposed a simulation mechanism for implementing IoT using OMNeT++. Hardware centric development method is inappropriate for the fact of reduced debugging and testing capabilities. Cost and effort setup the testbeds would be rather high compared to simulation environments. Apart from using TinyOS and Contiki OS (IoT operating systems) the authors have proposed a new cross-platform OS named CometOS under GPLv3. They haven't provided information regarding the network protocols developed. Here, the user protocols are implemented and executed without porting effort. The architecture is given consists of classes, module, Input Gate, Output Gate, Message and Object similar to OMNeT++. A MAC abstraction layer has been defined to restrict platform dependent code as much as possible and airframes to support serialization and deserialization of data. It is evaluated in OMNeT++. HelioMesh framework was deployed which provided with most of the findings of this paper. The simulation time and real time were synchronous.

This research has provided the usage of simulation methodology for the cross-platform protocol development and performance evaluation. The work has motivated us to choose the simulation methodology for our research at the earlier stage of the work.

A 6LoWPAN model for OMNeTT++

In[7], Contiki is embedded into OMNeT++ by compiling Contiki as a library with its interfaces mapped into OMNeT++.This enables the use of Contiki implementations like 6LoWPAN directly in OMNeT++.Although Cooja is a handy tool with various strong points it is just a simulation environment with a scope limited to the wireless communication between sensor nodes. Cooja simulators can only model a sensor network which is often a cornerstone in complex IoT networks. If different sensor networks separated with the border gateways are combined, then OMNeT++ is the only solution. Both Contiki and OMNeT++ are linked using statistically compiled and linked bridge between OMNeT++ and Contiki. Contiki can also be extended into the new platform. Contiki interface calls are redirected and mapped into OMNeT++ in this platform. Whenever IPv6 packets arrive through the connected gate from the upper layer they are converted into Contiki format and written into a packet buffer. Transport layer protocols are converted into the Contiki data format using the process. Transport layer packets are then encapsulated into 6LoWPAN packets. In addition, Contiki and OMNeT++ clock needs to be mapped. This work has provided the basic guideline and instructions in linking the OMNeTT++ and Cooja simulator in the Contiki operating system.

Security Protocols and privacy issues into 6LoWPAN

[8]The paper gives a view of the different ways to achieve security in the IoT. Compressed IPSec and DTLS protocols were taken into consideration and end-to-end use cases were proposed to assess how to implement these protocols in a real system. It tells that DTLS is quite heavy for constrained nodes but is scalable and is compatible with a RESTful environment. For IPSec, key establishment and cipher suite negotiation pose a threat while compressed IPSec has features to ensure source authentication and data confidentiality. This research has stated the various security aspects provided by the IPSec and implementation challenges in the 6LoWPAN networks.

LoWPAN multi-layered security protocol based on IEEE 802.15.4 security features

[9]Glissa and Medeb proposed a new security protocol “Combined 6LoWPSec” that operates alternately at the MAC and adaptation layers and provides end-to-end and hop-by-hop security and tolerating attacks caused by IPv6 hosts or the local ones in the network. By studying about the various attacks on 6LoWPAN networks, an algorithm AES-CCM* has been proposed that provides data integrity, confidentiality, authentication, availability and malicious intrusion detection. The written algorithm was implemented and evaluated using Cooja network simulator in a Contiki OS and parameters such as memory overhead, energy consumption and end-to-end delay were analyzed.

The work has also provided the information regarding the various possible security threats in the 6LoWPAN networks. The work has given idea of analyzing security protocols in which we have considered in this work in terms of providing security against these security threats.

Security as a COAP resource: An Optimized DTLS implementation for the IoT

In this research[10], authors focused on optimizing the implementation of DTLS protocol for COAP by using Elliptical curve cryptography(ECC) and minimizing ROM occupancy. The authors implemented the proposed solution on a magnode mote and analyzed the performance. It is a 8bit ultra low power 16MHz microcontroller having 16kb ram and 128kb rom which is enough to store a tinyOS stack. They developed assembly code routines allowing efficient use of registers to reduce memory operations. An ECC library was developed where ECC was implemented as a MNT curve. Tests were performed to calculate computational overhead, energy consumption and ROM occupancy and tradeoffs were established.

This work has evaluated the DTLS protocol in terms of energy consumption in the real hardware device. As this work has evaluated the DTLS in terms of the energy consumption. We haven't chosen the energy consumption as one of our performance metric.

Lightweight DTLS Implementation in COAP-based IoT

[11]Vishwas and Kewal highlighted on the need for providing a framework for implementing DTLS in IoT and a real-world scenario is illustrated also providing information on the ongoing standardization activities in the security domain. The implementation of TinyDTLS and the interconnection between the main DTLS module was shown. Different interfaces are defined for the forward and reverse traffic flows where the pseudo code that is being invoked for the read and write operations. An application example with a web browser as the client, web layer as the gateway and a temperature sensor as the IoT device was performed implementing end-to-end DTLS and performance analysis was done.

An Enhanced DTLS protocol for Internet of things applications

The research in [12] is focused on optimizing the performance of DTLS protocol in the constrained IoT environment. In-order to prevent DoS attacks, a cookie exchange technique has been introduced. If cookies are used in the authentication phase, lots of energy will be consumed and will be delayed, so, a proxy has been setup to save energy and prevent latency. Comparisons were done with regards to standard DTLS implementation and the proposed enhanced DTLS protocol. The proposed DTLS version shows good performance in comparison with original DTLS in-terms of computation time, packet overhead, with significant reduction in RAM usage and allowing the devices to save energy.

A DTLS based end-to-end security architecture for the IoT with 2-way authentication

In [13], DTLS authentication based on RSA is adopted. For scenarios, such as devices dealing with sensitive data, an access control server is introduced into the architecture between the gateway and internet. A tamper proof chip is enabled on the sensor node and all the cryptographic operations carried inside them. A microcontroller of 48kb RAM and an OpenSSL server were used to implement the proposed architecture. Latency, energy consumption and memory were the parameters taken into consideration by the authors. They concluded that the solution provides message integrity, confidentiality and authenticity with affordable energy, end-to-end latency and are aiming to implement without a TPM where DTLS using PSK must be adopted. The work has us provided the information regarding the implementation of DTLS protocol with exchange of x.509 certificates for the authentication purposes. The same way of DTLS implementation was followed in this research for the performance evaluation.

Securing Communication in 6LoWPAN using IPSec

In [14] the authors provide an end-to-end secure communication between IP enabled IoT devices and the internet. They claim that the research was the first compressed lightweight design, implementation and execution of 6LoWPAN extension for IPSec. A specification of IPsec for 6LoWPAN for AH and ESP were given. They implemented IPSec for 6LoWPAN networks and proved that it is feasible to secure 6LoWPAN networks using IPSec. Performance evaluation in terms of code size, packet overhead and communication was done. Next header compression (NHC) consist of NHC octet where 3 bits are used to encode IPv6 extension header ID(EID). The NHC for AH after optimal compression is 16 bytes and is 24 bytes for ESP compressed NHC. The impact of IPsec in terms of memory footprint, packet size, energy consumption under the different configurations are measured and analyzed. Based on the research, they conclude that IPsec is possible to implement and feasible to use Compressed IPsec for securing the communication. They were planning to investigate if an automatic key exchange protocol (IKE) for 6LoWPAN would be feasible. This work has provided the IPSec implementation and specification for the 6LoWPAN
Networks.

6LoWPAN compressed DTLS for COAP

In the research described in [15], the authors provide 6LoWPAN compression mechanisms for the DTLS protocols. 6LoWPAN-generic header compression(GHC) had been proposed as a plugin for 6LoWPAN which can be used to compress UDP. UDP-GHC is defined to compress DTLS by providing GHC for DTLS messages. 6LoWPAN-GHC for compressing the record and handshake headers were proposed where an encoding for both the record and handshake header are defined. 6LoWPAN-GHC for compression of the ClientHello message was also proposed where the handshake message is sent twice- with cookie and without cookie. In the client message, only random field is sent and all other fields are elided. Similarly, 6LoWPAN-GHC header for compression of ServerHello message is also proposed. By using the proposal, the authors saved about 60% at the record layer which is

common in all the DTLS messages. They plan to implement and evaluate the performance of the compressed DTLS. This work has given an idea about the feasibilty of compression of DTLS protocol less performance overhead.

Certificate-free Collaborative Key Agreement based on IKEv2 for IoT

In this paper[16], the research was done and an algorithm was proposed like IKEv2 where the receiver agrees upon the secret key, based on the collaborative values from the constrained and the unconstrained nodes. The proposed CKES algorithm is a collaborative key agreement where there are separate channels for IPsec and Ike for providing security. Like IPsec uses IKEv2 for automatic key.

A survey on techniques for securing the 6LoWPAN:

Paper [17] discusses about the possible threats in the 6lowpan layer and the mechanisms which is used till now to prevent from DOS attacks. IPsec can be used with 6LoWPAN but the drawback is that its encryption resource intensive and it increases overhead. It does not provide any protection against eavesdropping and network side attack like DOS attack. This paper mainly discusses about the different types of attacks and states about the methods to defend them like fragmentation & re-assembly buffer and bot neck attack along with different requirements of 6LoWPAN. The author discusses about MT6D techniques which provides protection against internal and external attacks. Hence use of that techniques is highly recommended.
The work has provided the information which is useful in evaluating the security protocols in terms confidentiality, privacy and data integrity specifically in resource constrained environment.

HTTP-COAP cross protocol Proxy: An implementation View Point:

Paper [18]discusses about the mapping of COAP to HTTP and issues related to its deployment and security. The author discusses about the cross-protocol proxy. It internally combines a client and a server for each of the two protocols, together with a couple of functions joining the two communication flows of the same full-duplex peering. the cross proxy enables bridging constrained networks and embedded platforms to the internet by means of a general purpose, widely available and supported architecture, so removing the need for application-specific sinks

When the http cross URI is offered at the proxy, the authority and the path components of the URI may in general be different from their original CoAP URI. In order to ease the URI mapping process and to facilitate the information retrieval, it is strongly recommended that a consistent URI mapping technique is used.

When a pre-defined threshold of outstanding requests is reached, the cross proxy may decide to cut any further incoming HTTP request by rejecting it as early as it is received using a 503 Service Unavailable error response, which signal the temporary unavailability of the target resource. An HTTP client receiving such a response will retry later to get the resource, maybe hinted by a Retry-After indication associated to the previous rejection response.

signal the temporary unavailability of the target resource. An HTTP client receiving such a response will retry later to get the resource, maybe hinted by a Retry-After indication associated to the previous rejection response.

Another option to reduce traffic towards popular resources involves establishing an Observe relationship with those resources. In fact, by using the Observe mechanism, a fresh representation of the resource is always sent to the proxy as soon as that representation is available at the hosting server, thus saving the request message soliciting the revalidation of the cached representation. The use of Observe is particularly efficient in the presence of a duty-cycled CoAP server, which can efficiently emit the notifications during its wake-up periods.
The cross proxy should continuously monitor the traffic pattern to dynamically pick the resources with which is convenient to establish an Observe relationship. Since a request/ response exchange involves a total cost of two messages, the cross proxy should mantain Observe relationships only towards those resources having an average time between two client requests (TR) lower than twice the resource representation freshness lifetime (FR). This work has provided the details about the implementation of the HTTP-CoAP proxy in the IoT environments.

End-to-End transport security in IP based IOT:

Paper [19] mainly discusses about mapping of TLS and DTLS to enable E2E secure communication with focus on the security issues. Is discusses about the IoT security architecture which consists of a 6LoWPAN border router (6LBR) and a group of nodes running COAP. The 6LBR interconnects the Low power lossy network (LLN) to the internet thus allowing access to the COAP/6LoWPAN devices any where from the internet. In this architecture, it is assumed that the back-end is a CoAP/HTTP client, while the node in the LLN is implemented as a CoAP server that provides resources and services to be accessed and managed remotely. It is further assumed that the nodes in the LLN are battery operated, whereas the 6LBR is equipped with a power supply. This architecture can be seen as a typical deployment for buildings to facilitate building automation and control.
A6LoWPAN border router provide any authentication but still could be useful in protecting the LoWPAN devices to some extent but does not apply to all security threats. The work has described about the security attacks related to resource exhaustion in low power lossy network networks. The work has showed us a different dimension of security threats that needs to be addressed in proper manner.

Security analysis of COAP in IoT:
This research [20] mainly focuses on the implementation of DTLS and IPsec protocols in securing the COAP. The research concludes the fact that that both protocols failed to meet the security requirements. This paper also discusses the issues in deploying these protocols in the IoT environments. The research was useful in addressing the various issues in deploying these protocols in IoT environments.

Performance Evaluation of HTTP-COAP proxy for WSN:
The paper [21] discusses about the design and implementation of the reverse proxy which can be accessed by client supporting the HTTP.A reverse proxy is paced at the edge of the 6LoWPAN network. Caching mechanism at proxy is discussed. With freshness caching mechanism cache inside the proxy, prior

existing CoAP response can be used to satisfy incoming requests, as long as the data has not expired. The paper mainly focuses on how proxy handles HTTP GET request to access the sensor network. This work discusses about important characteristics of the proxy. Firstly, as the number of concurrent clients goes up, the latency increases in the linear manner. Secondly, caching mechanism at the proxy effectively decrease the latency in the constrained network. The work evaluated the performance of HTTP-CoAP caching mechanism for HTTP-GET requests.

Connecting the Web of things lessons learned from implementing HTTP-COAP proxy:

The research paper [22] discusses about mapping of the HTTP protocol to the COAP protocol. It discusses about the types of proxies and their advantages and disadvantages. The author has briefly discussed about the various issues in mapping the COAP protocol with HTTP protocol. The research was useful in studying the feasibility of implementing the HTTP-CoAP Cross protocol proxy and its implementation in the IoT environments.

Study of impact of adding security in a 6LoWPAN based network

This paper [23] discusses about the end to end link layer security in the 6LoWPAN stack. It confirms that the 6LoWPAN behaves well in terms of memory and latency. The impact of providing link layer security in 6LoWPAN is acceptable and its impact even decreases when number of devices increases or number of message decreases. By limiting the number messages send by an application the impact of security can be halved. The work has provided the information regarding the impact of link layer security in 6LoWPAN on the resources available on the devices.

Analytical study of security aspects in 6LoWPAN networks

Paper [24] discusses about the various threats and attacks at the various layers in 6LoWPAN. At the physical layer there is possibility of the jamming attack and tampering attack. At the link layer there is a possibility of the exhaustion attack, interrogation attack. At the adaptation layer there is a possibility of disruption of fragmentation and reassembly operations. The network layer of the 6LoWPAN is most prone to security attacks like sink hole attack, hello flood attack, black hole attack, sybil attack, wormhole attack, spoofing attack, smurf attack. At transport layer, the attacker tries to exhaust the energy of the victim node via multiple connection requests by Flooding attack, or force him to react with synchronization messages imitating error messages by the De-synchronization attack. At the application layer there is a possibility of overwhelming attack aimed at destroying the routing by generating huge traffic to the Edge Router, and Path-based Dos attack aimed depleting resources by injection of false messages. This work also discusses about security requirements to applications.

The IEEE 802.15.4 defines eight types of security located in the data link layer, which are all based on the AES (Advanced Encryption Standard), these types are: encryption only (AES-CTR), only authentication (AES-CBC-MAC) or encryption and authentication (AES-CCM). Each category supports authentication that comes in three variants depending on the size of the MAC.

IPv6 designed for non-low-resource devices, and security protocol IPsec works well on these devices, but it is very greedy to 6LoWPAN devices. IPsec defines two security

modes: AH and ESP. AH provides integrity and authentication, and ESP provides integrity and confidentiality. The fact that IPsec also requires another header (AH or ESP) in each packet, it makes its use difficult in 6lowpan environments. IPsec requires two peers communicate to share a secret key which is usually implemented dynamically with IKE (Internet Key Exchange) protocol. The work also proposes the key management system and intrusion detection system for 6LoWPAN networks.

This is useful for our work as it as given a detailed description of the various security requirements in the 6LoWPAN networks and existing security architectures their advantages and dis-advantages.

3 BACKGROUND

3.1 Low Power Wireless Private Area Network (6LoWPAN)

It is the acronym that combines the latest version of the internet protocol and Low-power wireless personal are networks. 6LoWPAN allows the smallest of the devices which possess limited processing ability to transmit information wirelessly using an internet protocol. Low power wireless embedded radio technology usually has limited bandwidth (20-250kbits/sec) and frame size of the order 40k bytes. IEEE 802.15.4 defines four types of frames: beacon frames, MAC command frames, acknowledgement frames, and data frames. IPv6 packets MUST be carried on data frames IEEE 802.15.4 defines several addressing modes: it allows the use of either IEEE 64-bit extended addresses or (after an association event) 16-bit addresses unique within the PAN. Multicast is not supported natively in IEEE 802.15.4. Hence, IPv6 level multicast packets must be carried as link-layer broadcast frames in IEEE 802.15.4 networks. The MTU size for IPv6 packets over IEEE 802.15.4 is 1280 octets. However, a full IPv6 packet does not fit in an IEEE 802.15.4 frame. Starting from a maximum physical layer packet size of 127 octets (aMaxPHYPacketSize) and a maximum frame overhead of 25 (aMaxFrameOverhead), the resultant maximum frame size at the media access control layer is 102 octets. Link-layer security imposes further overhead, which in the maximum case (21 octets of overhead in the AES-CCM-128 case, versus 9 and 13 for AES-CCM-32 and AES-CCM-64, respectively) leaves only 81 bytes available. This is obviously far below the minimum IPv6 packet size of 1280 bytes, a fragmentation and reassembly layer should be present at the layer under IP layer. 6LoWPAN defines this as the adaptation layer. Since the IPv6 header is 40 bytes long it is leaving only 41 bytes for any upper layer protocols like UDP. UDP uses 8 bytes in header leaving only 33 bytes for the data to be transmitted at the application layer. So, an adaptation layer must be provided to compensate the IPv6 requirements of a minimum MTU.

3.1.1 Headers

All LoWPAN encapsulated datagrams transported over IEEE 802.15.4 are preceded by an encapsulation header stack. In a LoWPAN header, the analogous header sequence is mesh addressing, hop- by-hop options, fragmentation, and finally, payload. If more than a LoWPAN header is used in a single packet, then it must appear in the order:
Mesh addressing header, Broadcast and Fragmentation header.

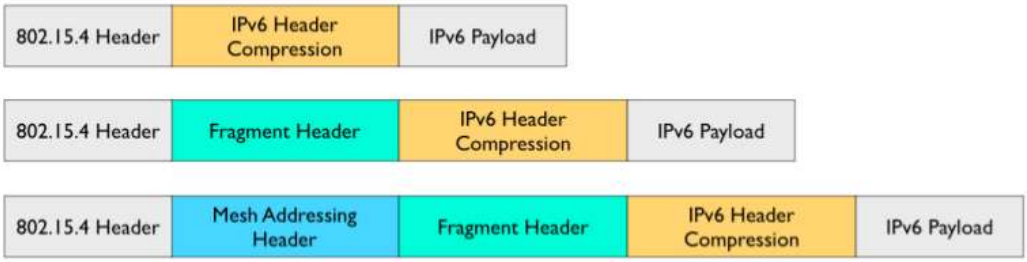


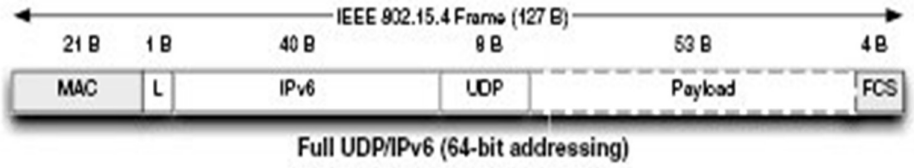
Figure 5 6LoWPAN addressing header

LoWPAN headers have the dispatch value with the header fields that follow and have ordered constraints relative to all headers. A dispatch value is treated as an unstructured namespace. Few symbols are sufficient to represent the current LoWPAN functionality where each symbol has its own importance. If the entire payload datagram fits in a single 802.15.4 frame, fragmentation header is not necessary. If it doesn't fit to one, then its broken into fragments.

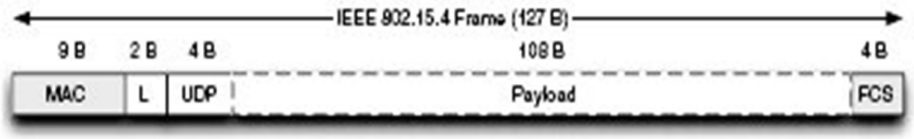
All link fragments of a datagram must be multiples of 8 bytes. Datagram size is the 11bit field that encodes the size of the entire IP packet and it shall be of the same size for all link layer formats. The value of the datagram tag is the same for all link fragments of a payload datagram. It is 16 fields long and the value wraps from 65535 to 0. Datagram offset is present only in the second and subsequent link fragments and will specify the octet.

3.1.2 Addressing Format

All 802.15.4 devices have a EUI-64 address, 16 bits short addressed are also possible. Pseudo 48-bit addresses are formed as: Left 32 bits are formed by joining 16 zero bits to the 16bit pan id. 16bit short address is added to the 32bit address to form the 48bit address. This forms the interface identifier.



Full UDP/IPv6 (64-bit addressing)



Minimal UDP/6LoWPAN (16-bit addressing)

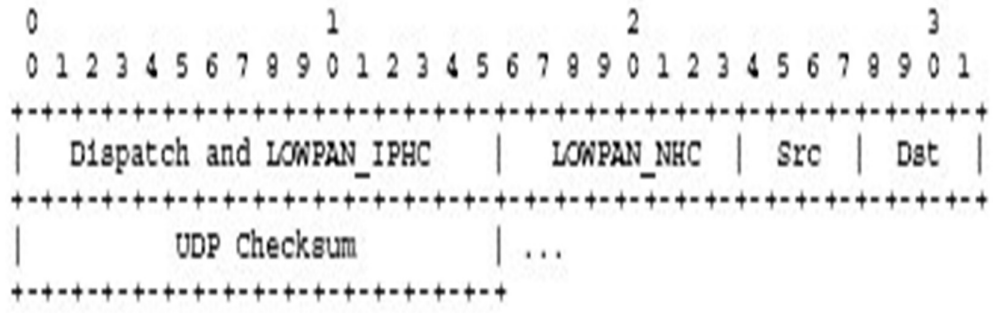


Figure 6 6LoWPAN Addressing format

An ipv6 link local address of an 802.15.4 interface is formed by adding FE80::/64 as the prefix. Layer 2 and layer 3 compression can be integrated because of the very limited packet size. IEEE 802.15.4 devices are generally deployed in multi hop networks. Hardware transmission of data cannot be done unless it is more than an octet, hence padding must be used. By virtue of joining the same 6LoWPAN network, the devices in the network share some things. HC1 header is assumed to be common in

the 6LoWPAN network. The source and destination address can be derived from the layer 2 source and destination address, the packet length can be derived from the datagram size field or even the layer 2 frame length form 802.15.4 frame PDU, next being whether the packet is UDP, TCP or ICMP. The only field that is carried in full is the hop limit. Depending on the states they share, different combinations of encoding the lowpan_HC1 header exist.

5th and 6th bits of LoWPAN_HC1 allows compressing the next header field i.e. TCP/UDP /ICMP. In the case of an UDP header, source port, destination port and length shall be altered. Checksum field is not changed and is carried in full. 8bit octet is compressed to 4 octets. If a UDP header is compressed, then based on the degree of compression, the non-compressed and partially compressed fields only are sent instead of the entire header.

3.1.3 Routing

If two devices cannot reach each other directly, the sender being the originator and the receiver being the destination, then the originator device may use intermediate devices as forwarders for the packet to reach its destination. For this, we need to include the link-layer addresses of both the originator and the destination. For LoWPAN broadcasting, an additional mesh routing functionality is encoded using a routing header following the mesh header. The broadcast header consists of LOWPAN_BC0 dispatch followed by sequence number. The sequence number is used to detect duplicate packets. The sequence number is incremented by the originator whenever it sends a new mesh broadcast or multicast packet. Additional mesh routing capabilities such as the mesh routing protocol, source routing can be defined by additional routing headers.

3.2 Constrained Application Protocol (COAP)

CoAP is one to one protocol for transferring info b/w clients and servers. It is state based and not purely event based. They send and receive UDP packets. It has built in support for content negotiation and discovery allowing devices to find ways of exchanging data. It is an efficient restful protocol, which is ideal for constrained devices and networks. It is designed for IoT and is compatible with proxy infrastructures. It mainly uses two security protocols to secure the network traffic namely DTLS and IPsec. CoAP defines four types of messages: Confirmable, Non-confirmable, Acknowledgement, Reset.

The basic exchanges of the four types of messages are somewhat orthogonal to the request/response interactions; requests can be carried in Confirmable and Non- confirmable messages, and responses can be carried in these as well as piggybacked in Acknowledgement messages.

3.2.1 Message ID

Each message contains a Message ID used to detect duplicates and for optional reliability. Confirmable message is retransmitted using a default timeout and

exponential back-off between retransmissions, until the recipient sends an Acknowledgement message (ACK) with the same Message ID and when a recipient is not at all able to process a Confirmable message, it replies with a Reset message (RST) instead of an Acknowledgement (ACK). A message that does not require reliable transmission can be sent as a Non-confirmable message (NON). These are not acknowledged, but still have a Message ID for duplicate detection. When a recipient is not able to process a Non-confirmable message, it may reply with a Reset message (RST).

3.2.2 Request and Response

CoAP request and response semantics are carried in CoAP messages, which include either a Method Code or Response Code, respectively. A Token is used to match responses to requests independently from the underlying messages. Optional request and response information, such as the URI and payload media type are carried as CoAP options.

3.2.3 Piggy backed Response

In a piggybacked response, a request is carried in a Confirmable (CON) or Non-confirmable (NON) message, and, if immediately available, the response to a request carried in a Confirmable message is carried in the resulting Acknowledgement (ACK) message. There is no need for separately acknowledging a piggybacked response, as the client will retransmit the request if the Acknowledgement message carrying the piggybacked response is lost

3.2.4 Separate Response

In a separate response, when the server is not able to respond immediately to a request carried in a Confirmable message, it simply responds with an Empty Acknowledgement message so that the client can stop retransmitting the request. When the response is ready, the server sends it in a new Confirmable message which then needs to be acknowledged by the client. If a request is sent in a Non-confirmable message, then the response is sent using a new Non-confirmable message.

CoAP makes use of GET, PUT, POST, and DELETE methods in a similar manner to HTTP. New methods do not necessarily have to use requests and responses in pairs. Even for existing methods, a single request may yield multiple responses.

3.2.5 GET

The GET method retrieves a representation for the information that currently corresponds to the resource identified by the request URI. If the request includes an Accept Option, that indicates the preferred content-format of a response. If the request includes an ETag Option, the GET method requests that ETag be validated and that the representation be transferred only if validation failed. Upon success, a

2.05(Content) or 2.03 (Valid) Response Code SHOULD be present in the response. The GET method is safe and idempotent.

3.2.6 POST

It requests that the representation enclosed in the request be processed. The actual function performed by the POST method is determined by the origin server and dependent on the target resource. It usually results in a new resource being created or the target resource being updated. If a resource has been created on the server, the response returned by the server SHOULD have a 2.01(Created) Response Code and SHOULD include the URI of the new resource in a sequence of one or more Location-Path and/or Location-Query Options. If the POST succeeds but does not result in a new resource being created on the server, the response SHOULD have a 2.04 (Changed) Response Code. If the POST succeeds and results in the target resource being deleted, the response SHOULD have a 2.02 (Deleted) Response Code. POST is neither safe nor idempotent.

3.2.7 PUT

It requests that the resource identified by the request URI be updated or created with the enclosed representation. The representation format is specified by the media type and content coding given in the Content-Format Option, if provided. If a resource exists at the request URI, the enclosed representation SHOULD be considered a modified version of that resource, and a 2.04 (Changed) Response Code SHOULD be returned. If no resource exists, then the server MAY create a new resource with that URI, resulting in a 2.01 (Created) Response Code. If the resource could not be created or modified, then an appropriate error Response Code SHOULD be sent. PUT is not safe but is idempotent.

3.2.8 DELETE

The DELETE method requests that the resource identified by the request URI be deleted. Delete response Code SHOULD be used on success or in case the resource did not exist before the request. DELETE is not safe but is idempotent. The protocol supports the caching of responses in order to efficiently fulfill requests. Simple caching is enabled using freshness and validity information carried with CoAP responses. Proxying is useful in constrained networks to limit network traffic, to improve performance, to access resources of sleeping devices, and for security reasons.

3.2.9 Message Format

CoAP is based on the exchange of compact messages. It could also be used over other transports such as SMS, TCP, or SCTP. The message format starts with a fixed-size 4-byte header, variable-length token value between 0 and 8 bytes long, sequence of zero or more CoAP. Options in Type-Length-Value (TLV) format, and an optional payload that takes up the rest of the datagram. (include message format picture).

Table 3 Message Format

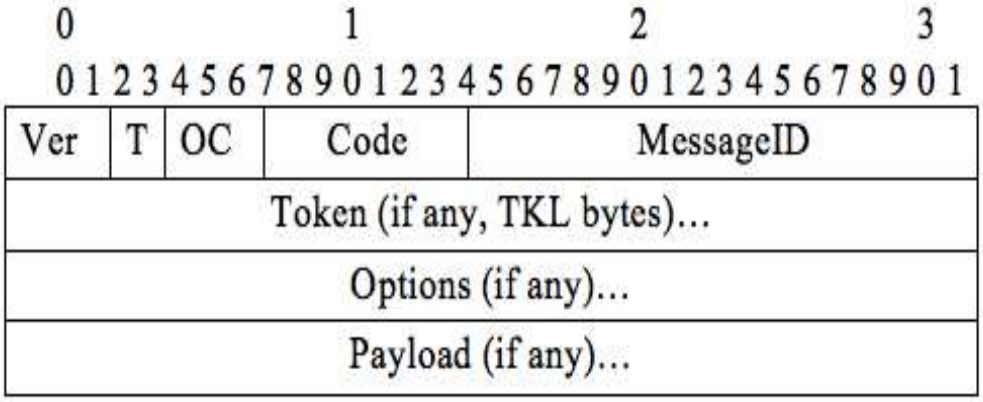


Figure 7 COAP Message Format

The fields in the header are defined as follows:
Version: 2-bit unsigned integer. Indicates the CoAP version number. Messages with unknown version numbers MUST be silently ignored.
Type (T): 2-bit unsigned integer. Indicates if this message is of type Confirmable (0), Non-confirmable (1), Acknowledgement (2), or Reset (3).
Token Length (TKL): 4-bit unsigned integer. Indicates the length of the variable-length.
Token field: (0-8 bytes) Lengths 9-15 are reserved, MUST NOT be sent, and MUST be processed as a message format error.
Code: 8-bit unsigned integer, split into 3-bit class i.e. most significant bits and 5-bit detail i.e. least significant bits. The class can indicate a request (0), a success response, a client error response, or a server error response. As a special case, Code 0.00 indicates an Empty message. In case of a request, the Code field indicates the Request Method; in case of a response, a Response Code.
Message ID: 16-bit unsigned integer in network byte order. Used to detect message duplication and to match messages of type Acknowledgement/Reset to messages of type Confirmable/Non-confirmable.

3.2.10 Option Values

CoAP defines several options that can be included in a message. Each option instance in a message specifies the Option Number of the defined CoAP option, the length of the Option Value, and the Option Value itself. Option Delta: 4-bit unsigned integer. A value between 0 and 12 indicates the Option Delta. Three values are reserved for special constructs. Option Length: 4-bit unsigned integer. A value between 0 and 12 indicates the length of the Option Value, in bytes. Three values are reserved for special constructs. Different option value formats are empty, opaque, unit and string.
Empty: A zero-length sequence of bytes.
Opaque: An opaque sequence of bytes.
Unit: A non-negative integer that is represented in network byte order using the number of bytes given by the Option Length field.
String: A Unicode string that is encoded using UTF-8 in Net-Unicode form.

3.2.11 Message Translation

A CoAP endpoint is the source or destination of a CoAP message. With no security, the endpoint is solely identified by an IP address and a UDP port number. With other security modes, the endpoint is identified as defined by the security mode. A Confirmable message always carries either a request or response, unless it is used only to elicit a Reset message, in which case it is Empty. The sender retransmits the confirmable message at exponentially increasing intervals, until it receives an acknowledgement or runs out of attempts. As a more lightweight alternative, a message can be transmitted less reliably by marking the message as Non-confirmable. Non-confirmable message MUST NOT be acknowledged by the recipient. A recipient MUST reject the message if it lacks context to process the message properly or has a message format error. To enable the receiver to act only once on the message, Non-confirmable messages specify a Message ID as well. Message correlation is An Acknowledgement or Reset message is related to a Confirmable message or Non-confirmable message by means of a Message ID along with additional address information of the corresponding endpoint.
It defines 4 security modes to achieve security:

1. No Sec
2. Pre-shared key
3. Raw-Public-Key
4. Certificate

Most important concerns of the security lie in authentication, authorization, encryption, digital signing and infrastructure security. Protocols must use the core security mechanisms and use IT standards. Some standard protocols that can be used are TLS, IPSEC/VPN, SSH, DTLS (UDP ONLY), HTTPS, SFTP, SNMP V3, secure boot loader and automatic fallback.

4 SECURITY PROTOCOLS

4.1 Internet Protocol Security (IPsec)

4.1.1 Introduction

In 1994 the Internet Architecture Board (IAB) issues a report titles “Security in the Internet Architecture”. The report identified the key area for security mechanisms. To provide security, the IAB included authentication and encryption as necessary features for the next generation IP which has been issued as IPv6. Fortunately, these security capabilities were designed to be usable both with the current IPv4 and IPv6. IPsec provides the capability to secure communication across LAN, WAN and across the internet. Internet protocol security is a suite of protocols that provides security to the internet communication at the IP layer. The principal feature of IPsec is that it enables it to support these varied applications is that it can encrypt and authenticate all the traffic at IP level. Thus, all the distributed applications that implement IPsec will be secured.

4.1.2 Functionality

IPsec provides services at the IP layer by enabling a system to select required security protocols, determine the algorithms to use for the services and to put in place any cryptographic keys required to provide the requested services. It is most commonly used to provide a virtual private network (VPN) between two locations i.e. gateway to gateway, between a remote user and a company network i.e. host to gateway, between two users i.e. end to end/ host to host security. Internet key exchange (IKE) is the protocol which is responsible for key negotiation and management providing dynamically negotiated and updated keying material for IPsec. It can be used for IPv4 and IPv6. Three versions of the IPsec implementations exist with latest being IPsec-v3 incorporating the lessons learned from the previous versions. The main additions are in IPsec-v2, a security association(SA) is uniquely identified by the combination of Security Parameters Index(SPI), protocol(ESP/AH) and destination address, whereas in IPsec-v3, unicast SA is identified uniquely by SPI/ protocol optionally. For multicast SA, it is a combination of SPI, destination address, optionally even the source address. More flexible Security policy database(SPD) selectors which includes ranges of values and ICMP message types. Decorrelated Security association database(SAD) i.e. order-independent replaced the former ordered SAD. Extended sequence numbers were added. For ESP, combined mode algorithms were added, recommending changes to packet format and processing. Null authentication which was mandatory in ESP-v2 is made optional. Two protocols are used to provide the security: an authentication protocol designated by the header of the protocol, Authentication Header (AH); and a combined encryption /authentication designated by the format of the protocol, Encapsulating Security Payload (ESP).

4.1.3 Authentication Header

It provides only authentication. Services offered are data integration, data origin authentication and replay protection service which can be made optional. HMAC-SHA/ HMAC-MD5 algorithms are used for securing the data integrity. Shared secret key provides the data origin authentication to create message digest. Replay protection is done by allotting a sequence number along with AH header.

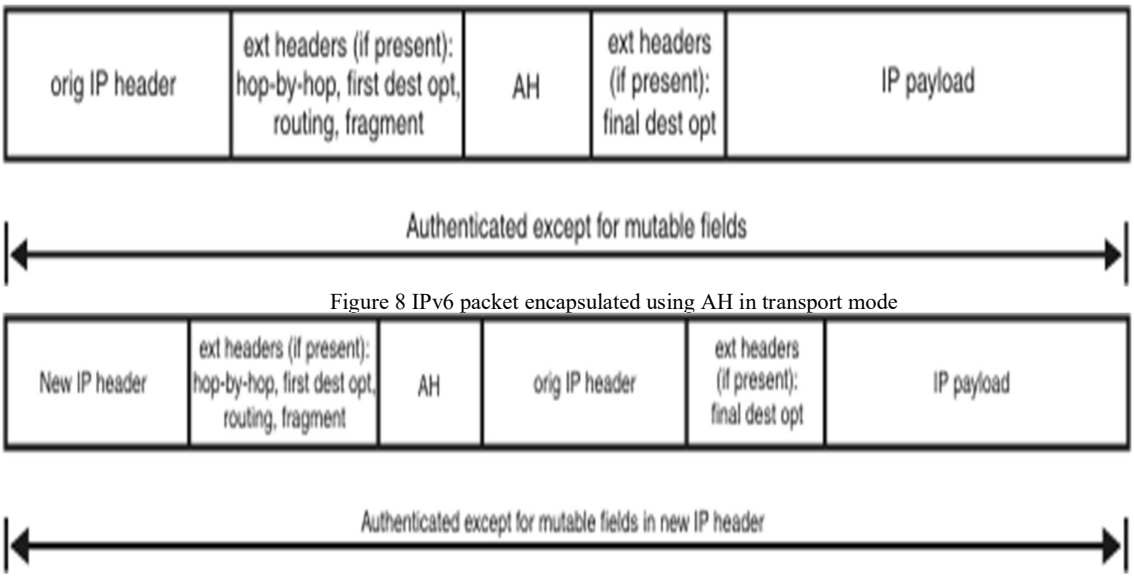


Figure 8 IPv6 packet encapsulated using AH in transport mode

Figure 9 IPv6 packet encapsulated using AH in tunnel mode

4.1.4 Encapsulating Service Pay Load

It provides data confidentiality in addition to the features provided by AH. It could be used for confidentiality alone, authentication alone, or both of them together. ESP uses the same algorithms as AH, but it only authenticates the IP datagram portion of the IP packet.

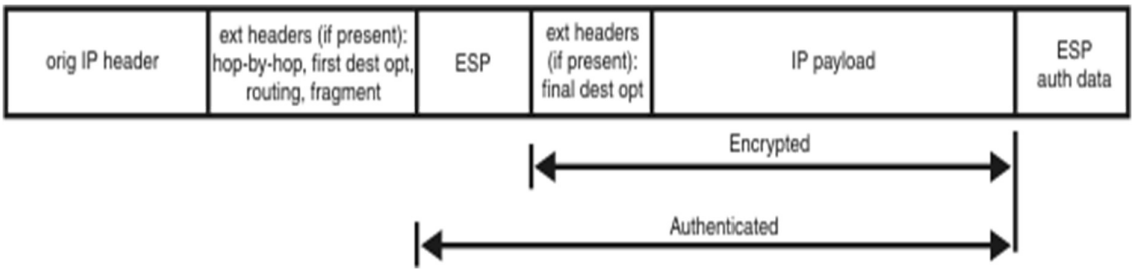


Figure 10 IPv6 encapsulated ESP header in transport mode

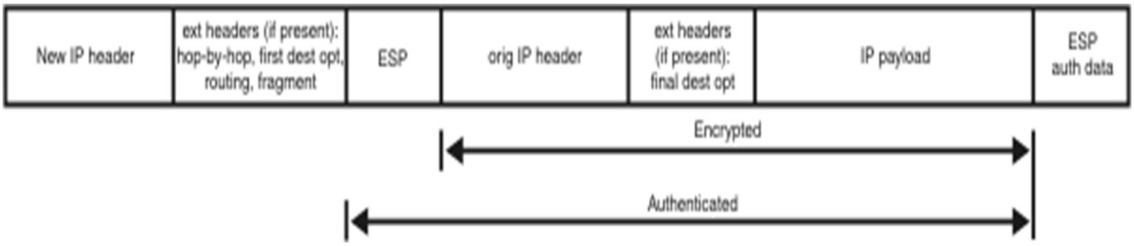


Figure 11 IPv6 encapsulated ESP header in tunnel mode

4.1.5 Services

IPsec provides the following security services:

- Access control
- Connectionless integrity
- Data origin authentication
- Rejection of payload packets
- Confidentiality via encryption
- Limited traffic flow confidentiality

4.1.6 UDP Encapsulation of ESP Packets

While building an ESP packet, it can be further encapsulated by placing a UDP header in front of the ESP header and this process is known as UDP encapsulation. This allows IPsec traffic to successfully traverse a NAT device.
In the transport mode, an UDP header is inserted between the IP header and the ESP header of a normal transport mode ESP packet.

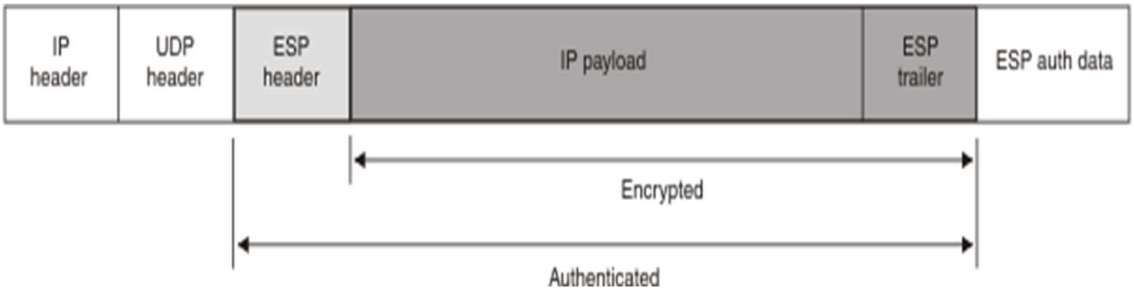


Figure 12 UDP- Encapsulated Transport mode header

In the tunnel mode, an UDP header is inserted between the new IP header and the ESP header of a normal tunnel mode ESP packet.

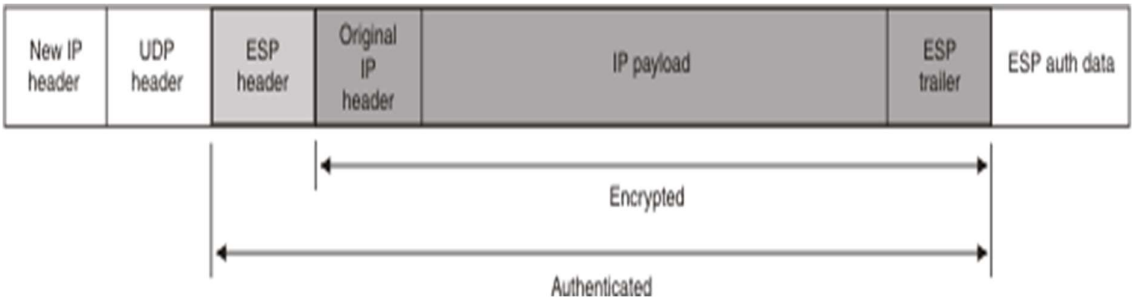


Figure 13 UDP- Encapsulated tunnel mode header

4.1.7 Modes of Operation

IPsec is operated in two modes namely Transport and Tunnel modes. Both AH and ESP support the two modes of use.

4.1.7.1 Transport Mode

Transport mode provides protection primarily for the upper layer protocols. Typically, transport mode is used for the end-to-end communication between two hosts. ESP in transport mode encrypts and optionally authenticates the IP payload but not the IP header. AH in transport mode authenticates the IP payload selected portions of the IP header. It retains the original IP header. So, in transport mode, the IP header reflects the original source and destination of the packet. It is routed and transported in the same manner as original packet.

4.1.7.2 Tunnel Mode

Tunnel mode provides protection to the entire IP packet. To achieve this, after the AH or ESP fields are added to the IP packet, the entire packet plus security fields is treated as the payload of the new outer IP packet with a new outer IP header. The entire original, inner, packet travels through a tunnel from one point of an IP network to another, no routers along the way can examine the inner IP header. ESP in tunnel mode encrypts and optionally authenticates the entire inner IP packet, including the inner IP header. AH in tunnel mode authenticates the entire inner IP packet and selected portions of outer IP header. A datagram encapsulated in tunnel mode is routed, tunneled through the security gateways with a possibility that the secured IPsec packet does not flow through the same network as the original datagram. Tunnel mode is required if one of the IKE peers is a security gateway that is applying IPsec on behalf of another host or hosts.

4.1.8 Public Key Cryptography

A Public key infrastructure(PKI) provides the means to establish trust by binding public keys to identities. Digital certificates are essential components of PKI. It contains a public key and a distinguished name(DN) which is the identifier of the entity owning the key carried by the certificate. Here, the issuer signs his certificate and makes it available for secure communication. Recipients must control the certificate authenticity prior to using the public key which can be achieved by obtaining public key of the certificate issuer and use it to verify the certificate signature. Public key of the issuer is made available through a certificate. Certificate Authority(CA) enjoys the trust of many users. CA on the highest level is the root CA which signs the certificate by itself. Certificate verification is a must which ensures the trust of all the certificates in the chain, failure to do so is considered a serious security threat. Public CAs should have the trust among the public because these have financial interests in securing the data while private CAs generate certificates within the organization. The standard format for digital certificates for PKI is X.509. Latest version being X.509v3 which addressed various issues in the prior versions. We have used the latest version i.e. X.509v3 standards for generating the certificates to secure the communication within the IoT network.

4.1.9 StrongSwan

StrongSwan is a keying daemon which used the Internet Key exchange protocols (IKEv1 and IKEv2) to establish security associations. Besides authentication and key material IKE also provides means to exchange configuration information to negotiate IPsec SAs which are called child_SAs. The association is divided to 2 parts:

1. Actual IPsec SAs describing the algorithms and keys used to encrypt and authenticate the traffic.
2. Policies that define which network traffic shall use such an SA.

StrongSwan only installs the negotiated IPsec SAs and SPs into kernel using an API, but the actual IPsec traffic is handled by the IPsec stack of the operating system kernel. For authentication, several methods can be used such as Pre-shared key, Extensible Authentication Protocol, eXtended Authentication and Public Key Authentication which uses RSA/ECDSA X.509 certificates to verify the authenticity of the peer. We use Public Key Authentication in our thesis to implement the IPsec VPN.
The main motivation behind the selection of StrongSwan over other IPSec implementation software's like OpenSwan etc., is its wide adaptability to different Linux distributions, implementation of both IKEv1 and IKEv2 key exchange protocols, extendibility to many plugins and enhanced documentation reports. StrongSwan is a complete IPSec solution providing encryption and authentication to servers and clients. Various StrongSwan parameters are:

- **IP address**: The left and right IP addresses are assigned with the VPN1's private IP
  and VPN2 peer's public IP respectively.
- **IDs**: Leftid and Rightid are denoted to specify the identity of the left and right endpoint.
- **Subnets**: The left and right subnets are the private subnets for secure data access.
- **IKE and ESP protocols**: The IKE and ESP options are used to denote various combinations of encryption algorithm and hashing algorithms.
- **Keying tries**: This option can be set to any positive integer, indicating the number of attempts to be made to negotiate a connection.
- **Ike-lifetime and lifetime**: These parameters indicate the period after which the keying channel of connection and instance of a connection.
- **Authby**: This option denotes the key distribution mechanism used to authenticate
  the peer-to-peer gateways.
  Key exchange: The protocol used for key exchanges.

4.2 Datagram Transport Layer Security (DTLS)

Network traffic is secured by many techniques. Transport Layer Security(TLS) is the most widely used protocol for securing web traffic and email protocols. It operates in a transparent connection oriented channel and runs over reliable transport channel such as Transmission Control Protocol (TCP). Application protocols using User Datagram Protocol (UDP) have increased over the past few years. IoT devices use the CoAP

protocol for communication which runs over UDP and so cannot use TLS. Thus, there is a need for datagram compatible variant of TLS. IETF has proposed DTLS in April, 2006 to maximize the amount of code reuse and minimize new security invention.

The unreliability of the UDP pose two challenges in implementing TLS at two different levels:

1. TLS does not allow independent decryption of the individual records as the integrity check depends on the sequence number. If the record N is not received, then the integrity check on the on-record N+1 will fail.
2. The TLS handshake layer assumes that handshake messages are delivered reliably and breaks if those messages are lost.

DTLS is a communication protocol used for giving security to datagram based applications and preventing eavesdropping, tampering, message forgery. It is designed to secure the communication among applications such as online gaming, live streaming, Internet telephony and in the communication between resource constrained devices which use UDP as the transport layer protocol. DTLS uses IPsec ESP mechanism and explicitly adds the sequence numbers. It is designed to run in application space and doesn't need any kernel modifications. It doesn't compensate for lost or re-ordered data traffic.

4.2.1 Providing Reliability For Handshake

In TLS, messages must be transmitted and received in a defined order. If there is mismatch in the order, then it will produce an error. This feature is incompatible with the packet reordering and message loss of the UDP. The TLS handshake messages create a problem of the IP fragmentation as these messages are larger than size of the datagram. DTLS addresses these two issues.

4.2.2 Packet Loss

To address the issue of the packet loss DTLS uses retransmission timer. During the initial phase of the DTLS hand shake the client sends the client hello message to the server and expects a hello verify request. If the client does not receive the hello verify request after the specified amount of time then the timer expires and the client knows either client hello or hello verify request has been lost. The client retransmits the message and when the server receives the retransmission it knows to re transmit. The server also has the retransmission timer and it retransmits when the timer expires. The timeout and retransmission do not apply for the hello verify request. The hello verify request is designed to be small enough that it will not itself be fragmented, thus avoiding the concerns about multiple hello verify requests.

4.2.3 Reordering

The handshake message is assigned a specific sequence number within that handshake in DTLS. The peer which receives this handshake message determines whether is the next message that it expects or not. If the received message is same as the expected message, then it processes it. If not, then it queues the message for the future handling once all the previous have been received.

4.2.4 Message Size

The size of the DTLS and TLS hand shake messages are quite large when compared to the size of the UDP datagrams. To address this issue each DTLS handshake message is fragmented into several DTLS records each is intended to fit in a single IP datagram. Each handshake message contains both the fragment offset and fragment length. Thus, the recipient in possession of all the bytes of a handshake message can reassemble the original unfragmented message.

4.2.5 Replay Detection

There is optional support for the record reply detection. DTLS uses the techniques like the IPsec AH/ESP by maintaining a bit map window of the received records. Records that are too old to fit in the window and the records that have previously been received are silently discarded.

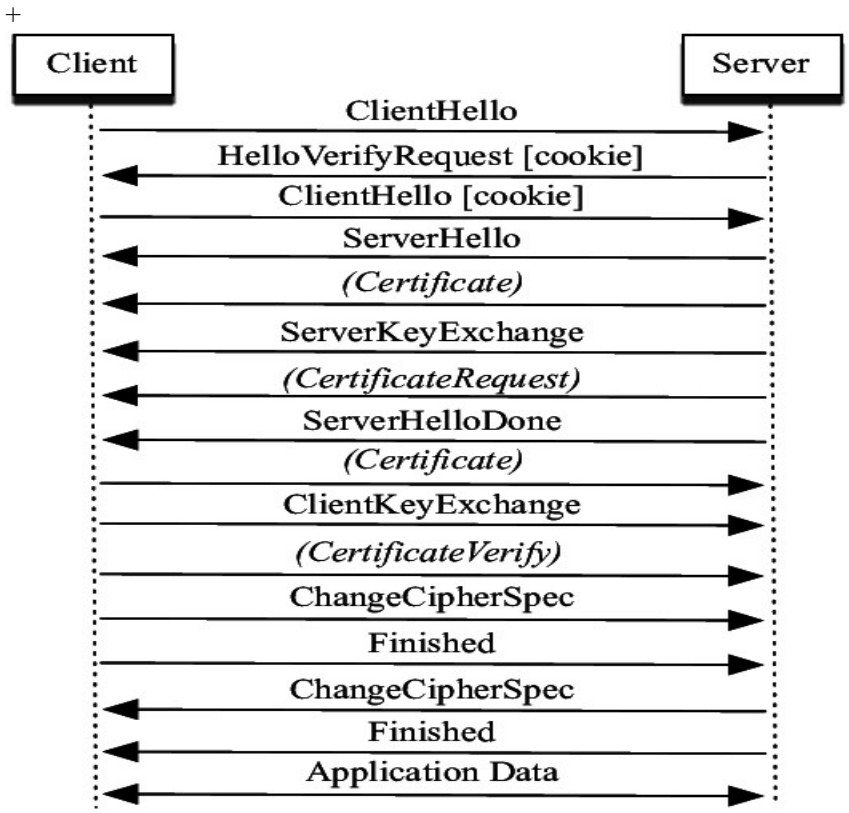


Figure 14 DTLS Architecture and DTLS Handshake

4.2.6 DTLS Handshake Protocol

DTLS uses all the features of the TLS with three major changes.

1. To prevent the Denial of service, attack a stateless cookie exchange has been added in DTLS

2. To handle message loss, reordering and DTLS message fragmentation slight modifications are made to the handshake header.
3. To handle message loss retransmission timers are included.

Apart from the above stated exceptions DTLS message formats, flows and logic are same as the TLS.

4.2.6.1 Counter Measures to DOS

There are two types of DOS attacks which are of major concern in DTLS

1. An attacker sends multiple series of handshake initiation requests which server to perform the expensive cryptographic operations.
2. An attacker can use the server as an amplifier by sending connection initiation messages with the forged source of the victim.

To protect the system for these two types of the attack DTLS uses the stateless cookies techniques. When the client sends the ClientHello message to the server the server responds with the Hello verify Request message which contains the stateless cookie. The server then verifies the cookie. This mechanism makes DOS attack with the spoofed IP address difficult but does not provide any protection against DOS form the valid IP address. The cookie generation techniques used in DTLS is like that of Photuris and IKEv2.

4.2.6.2 Hand-shake Message Format

DTLS uses the modified version of the TLS 1.2 handshake header to handle message loss reordering and message fragmentation.

```
struct {
    HandshakeType msg_type;
    uint24 length;
    uint16 message_seq;
    uint24 fragment_offset;
    uint24 fragment_length;
    select (HandshakeType) {
      case hello_request: HelloRequest;
      case client_hello:  ClientHello;
      case hello_verify_request: HelloVerifyRequest;// New type
      case server_hello:  ServerHello;
      case certificate: Certificate;
      case server_key_exchange: ServerKeyExchange;
      case certificate_request: CertificateRequest;
      case server_hello_done: ServerHelloDone;
      case certificate_verify:  CertificateVerify;
      case client_key_exchange: ClientKeyExchange;
      case finished: Finished;
    } body;
  } Handshake;
```

4.2.7 Hand-shake Message Fragmentation and Reassembly

The DTLS handshake message must fit into the single transport layer datagram but the handshake messages are larger than the size of the datagram. To address this issue DTLS provides a mechanism for fragmenting the handshake message over several records that can be retransmitted separately to avoid IP fragmentation. The handshake message is divided into a series of N contiguous data ranges which are not larger than the size of the fragment should contain the entire handshake message. The sender then creates N handshake messages with same message sequence value as the original handshake message. Each new message is labeled with the fragment offset and fragment length. The length field is same as the original message. When the receiver receives the handshake fragment, it will buffer until it has the entire handshake message. DTLS receiver must be able to handle the overlapping fragment ranges. This allows sender to re transmit the handshake messages with smaller fragment sizes.

4.2.8 Security Analysis

Modern security protocols and available proof techniques make it a tough task to prove the security of a protocol without at least making some assumptions about the attack model. DTLS is implemented using the OpenSSL library and much of the code is reused form TLS server making it well tested and stable.

4.3 Transport Layer Security (TLS)

TLS protocol is designed to provide the data privacy and integrity between the two communicating entities. This protocol is application independent. The protocol is composed of two layers namely the TLS record protocol and the TLS handshake protocol. The TLS record protocol ensure the connection between the communication parties to private and reliable. The TLS record protocol encapsulates the various higher-level protocols. One among such encapsulated protocol is TLS handshake protocol. The TLS handshake protocol is responsible for the exchange of the cryptographic material between the client and server before the transmission the actual data. TLS handshake protocol provides the security which has three properties. The main advantages of TLS are it provides cryptographic security, extensibility, relative efficiency.

1. Authentication of the peers is done by using asymmetric or public key cryptography. The authentication is required for at least one of the communicating entities.
2. The negotiation of the shared key in a secure manner to make the secret unavailable to the eavesdroppers.
3. Reliable negotiation such that no attacker interferes in the communication between the two parties.

4.3.1 TLS Record Protocol

The TLS record Protocol is a layered protocol. This protocol takes the messages to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts and transmits the result. Received data is decrypted,

verified, reassembled, and then delivered to higher-level clients. A TLS connection state is the operating environment of the TLS record protocol. It specifies a compression algorithm, an encryption algorithm and a MAC algorithm.

4.3.2 TLS Handshake Protocol

TLS handshake protocol produces the session rate parameters, which operates on top of the TLS record layer. To initiate the communication the server and client agree on the protocol version, select cryptographic algorithms, optionally authenticate each other and use public-key encryption techniques to generate shared secrets.

The TLS Handshake protocol provides connection security that has three basic properties:

- The peer's identity can be authenticated using asymmetric or public key, cryptography. This authentication can be made optional, but is generally required for at least one of the peers.
- The negotiation of a shared secret is secure: the negotiated secret is unavailable to eavesdroppers, and for any authenticated connection the secret cannot be obtained, even by an attacker who can place himself in the middle of the connection.
- The negotiation is reliable: no attacker can modify the negotiation communication without being detected by the parties to the communication.

One advantage of TLS is that it is application protocol independent Higher-level protocols can layer on top of the TLS protocol transparently.

4.3.3 Advantages of TLS

- Strong authentication, message privacy and integrity
- Interoperability
- Algorithm flexibility
- Ease of deployment
- Ease of use

4.4 HTTP-COAP Proxy

The ability to interwork with HTTP is crucial in allowing smart constrained objects to integrate into the Web, thus pursuing the Internet of Things paradigm. In fact, the CoAP protocol has been designed to let the communication of CoAP endpoints span beyond their native networks, and integrate easily with other protocols sharing the same HTTP-like structure. The shared design and semantics principles – which also allow to naturally lay RESTful applications on top of both protocols regardless which of the two protocols is actually carrying the payloads – suggest that, in principle, a lightweight translation process (in fact a simple syntactic mapping) should be sufficient to let the application payloads flow seamlessly from one network to the other and vice versa.

A translating intermediary, which understands the semantics and syntax of both protocols, is the ideal candidate for achieving unobtrusive and transparent mapping

between CoAP and HTTP endpoints. This kind of intermediary is called cross protocol proxy, or cross proxy for short. It internally combines a client and a server for each of the two protocols, together with a couple of translation function, that takes care of joining the two communication flows of the same full-duplex peering. The proxy has an important role in the architecture of IoT, since the proxy allows the direct web service based communication between internet and the end-points and constrained devices.

The most commonly envisaged Web of Things use case is the one where an HTTP client, that is most likely a Browser, is interested in querying a CoAP resource hosted within a constrained network. To address this scenario a HTTP-CoAP proxy needs to be deployed as a reverse proxy at the edge of the constrained network, as shown in Figure 15. The said HTTP-CoAP proxy is in charge to retrieve resources on behalf of the HTTP client from one or more servers, within the constrained network, using the CoAP protocol. The proxy then returns these resources to the HTTP client as though they originated from itself, shielding completely both the constrained network topology and the actual protocols used with in it.

.

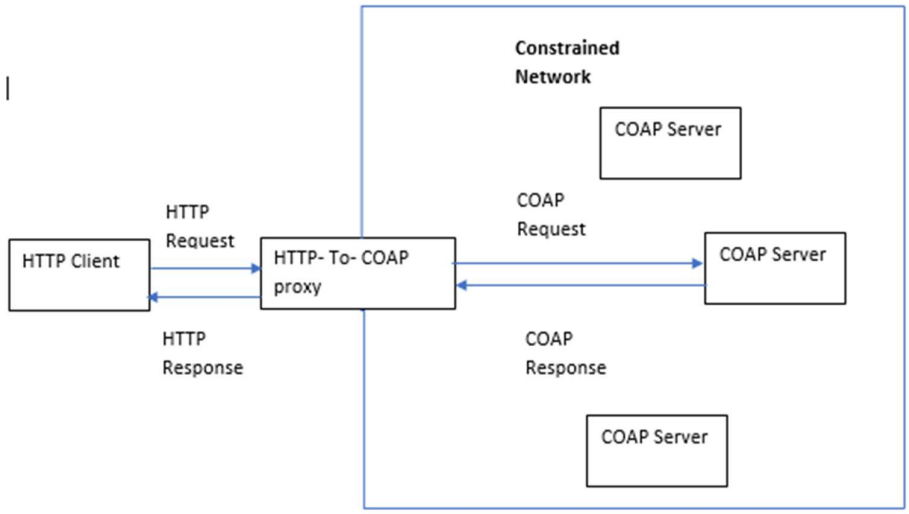


Figure 15 HTTP-COAP Proxy Deployment Scenario

Deploying the cross proxy at the edge of the constrained network present several advantages:

- **Configuration Overhead**: CoAP clients require either static proxy configuration or proxy discovery support. This overhead is simplified if the proxy is placed on the same network domain of the client.
- **CoAP to HTTP**: It also make possible for a CoAP client to easly interrogate an HTTP resource deployed on the unconstrained Internet.
- **TCP/UDP**: Translation between CoAP and HTTP implies also UDP to TCP mapping. As UDP performance over the unconstrained Internet may not be

adequate and also multicast is often not supported, the TCP/UDP conversion needs to be performed as closest as possible to the constrained network.

- **Caching and Flow Control**: All the CoAP traffic towards the unconstrained Internet has to pass through the same proxy making both caching and flow control function extremely efficient.

The proxy translates on the application layer and can therefore can be located anywhere in the internet. This in contrast to Wireless Sensor Networks (WSN) gateways. It is not feasible to design a fully transparent proxy. At least one partner must have the knowledge of the existence of the proxy.

Furthermore, as known from common HTTP proxies, the proxy can cache resources and, consequently, reduce network traffic inside the constrained network. This is also useful for CoAP to CoAP communication inside the constrained network, especially, when many clients request resources from a single server.

A CoAP or CoAPS scheme of a request URI indicates that a mapping should be performed. However, HTTP does not require an absolute Request-Uri, and many clients (like Firefox that was used for testing) omit the protocol scheme and the URI host. If no scheme is giving, the proxy cannot determine if mapping to COAP should be performed or if the client wishes the proxy to behave as a non-mapping HTTP proxy

The CoAP core specification defines 14 different header options. The Uri-Host, Uri-Path and Uri-Query options are mapped to the HTTP request URI and vice versa. The Proxy-Uri option is handled by the proxy itself. All the remaining COAP header options are translated directly to the HTTP header options and vice versa.

4.4.1 Congestion Control

For its gateway role between the unconstrained Internet network and the low-capacity IoT network, the cross proxy has an important role in keeping low the congestion level in the constrained environment. In fact, the total number of outstanding requests towards the network of CoAP servers should be strictly limited. When a pre-defined threshold of outstanding requests is reached, the cross proxy may decide to cut any further incoming HTTP request by rejecting it as early as it is received using a 503 Service Unavailable error response, which signal the temporary unavailability of the target resource. An HTTP client receiving such a response will retry later to get the resource, maybe hinted by a Retry-After indication associated to the previous rejection response.

4.4.2 Types of Proxies

There are two types of proxies. They are:

4.4.2.1 Forward Proxy

In this type of proxy, the client sends request to the server and the proxy then forwards the request to the server. For server, the proxy behaves as if it would be a client. The client needs to be configured to make use of the proxy

4.4.2.2 Reverse Proxy

In this type the client does not about the existence of the proxy and it expects a normal serer at the location of the proxy. The proxy then forwards the request to the hidden server. Therefore, the server must be associated to the proxy. Consequently, the client does not need to know about proxy and no configuration is required on the client side.

Mostly forward proxies are used in the Internet and the term proxy refers to a forward proxy. The proxy used in this thesis is also a forward proxy, which means that the client needs to be configured, not the server.

4.4.2.3 Protocol aware access

In this case the client knows about the cross-protocol translation and specifies if the translation is to be performed

4.4.2.4 Protocol-agnostic access

The client does not know about the cross-protocol translation. The mapping is silently done by the proxy

4.4.3 Cross Protocol URI Mapping

Both CoAP and HTTP define their own native Uniform Resource Identifier (URI) scheme. An HTTP user agent may not support the CoAP scheme natively; for this reason, it is worth extending the aforesaid translation function with a URI mapping facility. In this way, a CoAP resource could be served using an HTTP URI at a cross proxy, making it accessible from browsers even if they are completely unaware of the CoAP protocol domain.

When the HTTP-COAP cross URI is offered at the proxy, the authority and the path components of the URI may in general be different from their original CoAP URI. In order to ease the URI mapping process and to facilitate the information retrieval, it is strongly recommended that a consistent URI mapping technique is used. The following are a couple of simple and static URI mapping techniques that can be conveniently used:

- **Homogeneous Cross URI**: A cross URI is said homogeneous, when the same resource is identified univocally by the authority and path components of the URIs, irrespective of the scheme.
- **Embedded Cross URI**: A cross URI is said embedded when the native resource URI is completely contained in its translated URI.

4.4.4 Message Layer

CoAP provides either reliable or unreliable messaging. In case of an incoming CoAP request (CoAP to HTTP mapping), the CoAP client determines if reliable or unreliable messaging is used. In case of a HTTP request (HTTP

to CoAP mapping), the proxy acts as client and therefore predetermines if reliable or unreliable messaging is used. By default, the proxy uses reliable messaging, however, the proxy can be easily configured to use unreliable messaging.

4.4.5 Operations

HTTP defines more options than CoAP. PUT, GET, POST and DELETE are known by both protocols and can therefore be translated directly. The HTTP HEAD operation allows a HTTP client to only request the HTTP header. When translated to CoAP, the CoAP GET operation is used. After receiving the response, the payload of the response is removed during translation. All other operations (TRACE, OPTIONS, CONNECT, PATCH) cannot be mapped and a 501 Error ("Not Implemented") is returned by the proxy.

4.4.6 Caching

HTTP-COAP proxy maintains a cache of the received responses. Having a cache allows to avoid sending duplicate idempotent requests into the constrained network, if a fresh representation of the target resource is already stored at the proxy. Another option to reduce traffic towards popular resources involves establishing an Observe relationship with those resources. In fact, by using the Observe mechanism, a fresh representation of the resource is always sent to the proxy as soon as that representation is available at the hosting server, thus saving the request message soliciting the revalidation of the cached representation. The use of Observe is particularly efficient in the presence of a duty-cycled CoAP server, which can efficiently emit the notifications during its wake-up periods.

During the implementation of the proxy, it turned out, that only the Max-Age option of CoAP does not allow to determine exactly how long a resource is valid due to the transmission time

Because CoAP does not provide a Date header option like HTTP, it is not possible to determine exactly when the resource was created. Based on the assumption, that the creation time is at some point between the time when the request was sent and the respond is received, three strategies are conceivable of how to determine how long a resource is valid. All three cases are described as follows:

- The creation time is assumed to be at the time when the request is sent by the client. This leads to a False Positive (resource is withdrawn although it is still valid).
- The creation time is assumed to be at the time when the response is received. This leads to a false negative (resource is assumed as valid but already expired).
- A synchronous transmission is assumed (the request transmission time equals the response transmission time) and the creation time is approximated by the half of the roundtrip time. This can reduce the error but the error classification cannot be determined.

The proxy implements the third case by default, however, the proxy can be easily changed to use any other strategy. CoAP does not define a Date option like HTTP with

respect to constrained-devices that often does not have a global clock. However, a Date option could be useful in case of CoAP devices that have a global clock.

4.4.7 IPv4 and IPv6

Because TCP (transport layer of HTTP) and UDP (transport layer of CoAP) can be used with both IPv4 and IPv6, the proxy can also be used for IPv4-IPv6 protocol translation. Currently IPv4 is the prevailing protocol version in the Internet, whereas constrained-devices with 6LoWPAN require IPv6 based communication. The intermediate proxy solves this issue.

5 METHODOLOGY

This section describes about the methodologies adopted to achieve the objectives of the thesis.

- Literature review about the related works to gain sufficient knowledge about end to end DTLS and IPsec based secure communication in IoT environments.
- Setup a 6LoWPAN (Low Power Wireless Private Area Network) Network in namespaces consisting of several motes. Each mote acts as an IoT device.
- Build COAP servers on the 6LoWPAN motes and COAP clients to request operations to be done on the server.
- Secure the communication through end-to-end DTLS protocol in IoT environments.
- Secure the communication through end-to-end IPsec protocol in IoT environments.
- Conduct tests to extract the performance metrics elapsed time, CPU utilization, network overhead and memory utilization.
- Statistical analysis of the results obtained.

5.1 OMNeT++

OMNeT++ is an extensible, modular, component-based C++ simulation library and framework for network simulators. It is used to simulate WSN and MANET protocols, queuing networks, multiprocessors and other distributed hardware environments, validating hardware architecture. It offers an Eclipse-based IDE which is a graphical runtime environment. It performs real time simulation, network emulation, database integration and several other functions. Modules can be combined with each other via gates. It supports parallel distributed simulation where the existing algorithm can be updated or new owns can be plugged in. INET framework can be considered as the standard protocol library of OMNeT++.

5.2 Contiki

Contiki is an open source operating system for IoT. Contiki provides powerful low-power Internet communication. Contiki provides a full IP network stack, with standard IP protocols such as UDP, TCP, and HTTP, in addition to the new low-power standards like 6lowpan, RPL, and CoAP. Contiki supports the recently standardized IETF protocols for low-power IPv6 networking, including the 6lowpan adaptation layer, the RPL IPv6 multi-hop routing protocol, and the CoAP RESTful application-layer protocol.it Contiki is designed to operate in extremely low-power systems. To assist the development of low-power systems, Contiki provides mechanisms for estimating the system power consumption and for understanding where the power was spent.

5.3 Cooja

Contiki devices often make up large wireless networks. Cooja, the Contiki network simulator, makes this tremendously easier by providing a simulation environment that allows developers to both see their applications run in large-scale networks or in extreme detail on fully emulated hardware devices. A simulated Contiki Mote in COOJA is an actual compiled and executing Contiki system. The system is controlled and analyzed by COOJA.

5.4 Risk State

As stated, we have installed Contiki OS inside Ubuntu 16.04 machine and got the cooja simulator running the 6LoWPAN network and the IoT motes. The next job in hand was to run OMNeT++ inside the same virtual machine running the Contiki OS was done. We required INET and INET-MANET frameworks to link Contiki OS to the OMNeT++ simulator. After installing them, we succeeded in connecting Contiki OS to OMNeT++ simulator. During the process of linking OMNeT++ to Contiki OS, we have noticed that DTLS had no support under OMNeT++ and IPsec had very little support in Contiki OS. Addressing these shortcomings was deemed a very high risk for the thesis work. We understood, we wouldn't be successful in implementing end-to-end DTLS and IPsec for the communication of the IoT devices.

5.5 Alternative Chosen

Therefore, we have chosen an alternative which is to setup an emulated testbed by using operating system virtualization where there was very good support for DTLS and IPsec which are the protocols under study for the thesis. This approach would be even more realistic than simulation i.e. the previous method. We will setup an IoT site i.e. a virtual network consisting of servers, clients and a border/edge router through which the communication will take place. We have emulated the test bed on the single Linux kernel using the network namespaces.

5.6 Network Namespaces

Network namespaces are the mechanisms for resource and process isolation. In Linux, a network namespace is an instance of the TCP/IP stack implemented by the kernel. The Namespaces are isolated from each other and from the from the global namespace. Within a namespace one can have a different and separate instance of the network interfaces and routing tables that are independent of each other. These namespaces allow the process to communicate over the network without interfering the other process. A namespace within the kernel can be connected to other name spaces and to the external world using the virtual Ethernet (veth) interfaces. Virtual Ethernet interfaces always comes in pairs. They act like a pipe between two namespaces, anything that goes to from end will come out from the other end. Physical interfaces cannot be assigned to the new namespaces. They can only be assigned to global namespace.

5.7 Linux WPAN project

This project is intended to provide support for IEEE 802.15.4 and 6LoWPAN support in mainline Linux operating system. It provides IEEE 802.15.4 layer with Soft MAC layer for various transceivers. IT also provides support for the 6LoWPAN fragmentation and reassembly along with header compression with IPv6 header compression (IPHC) and Next Header compression (NHC) for UDP. It also provides Link layer security. This project provides a virtual driver which is mostly used for testing. This driver is useful in testing the different network stacks in the virtual environment. We have used version 0.7 with network namespace support for this thesis.

IEEE 802.15.4 stack has a capability to support multiple radio drivers, as well as multiple logical interfaces on one radio. Also, the stack includes so-called fakelb radio driver, which provides several interconnected virtual radio devices, thus making possible to implement several "virtual" radio nodes on one host. We have used this feature of the stack without real hardware. The latest version of the Linux kernel by default has nl802154 interface which is standard net link interface for the WPAN devices. To access the standard net link interface, we require WPAN tools.

5.8 WPAN-Tools

WPAN tools are the user space tools to configure and test the IEEE 802.15.4 devices. The package contains two CLI tools for working with IEEE 80.15.4 devices via net link interface. The two CLI tools that WPAN tools contain are:

- **IWPAN**: To configure a WPAN device to create a 6LoWPAN network in the kernel. IWPAN splits the configurable device into physical and a WPAN device. The settings under the physical device are the hardware specific settings of the radio such as transmit power and channel settings. The settings under the WPAN device contain interface specific settings such as MAC address and PAN id
- **WPAN Ping**: For the network testing of 6LoWPAN nodes.

5.8.1 PAN ID

Every WPAN has a 64-bit number which is used as the network identifier. This number is called as the PAN ID. We need to set the PAN ID manually to the WPAN while creating the network. A device can join the WPAN network if it has PAN ID of the network.

5.9 IoT Logical Network Architecture:

The IOT network architecture consists of the following elements namely IoT devices, border router, edge router. The IoT devices are typically constrained devices which grouped together to form the 6LoWPAN network. Generally, there might be multiple 6LoWPAN networks depending the on the functionality and the scenario in which they

are implemented. (For example all the temperature sensors in a particular house can be grouped into one network and all the light sensors into the another network.). The 6LoWPAN networks here are referred as the IoT sites. These multiple IoT sites are connected to border router. A border router acts as the interface between the 6LoWPAN network and the local network. The border router and the IoT sites are connected to the internet through a device called as Edge router. An edge router is a device which is located at the boundary of the network that is used to connect to the external networks and also to the internet. The Client host is located somewhere in the internet and tries to access the resources on the COAP server which resides on the IOT device. The following figure shows the logical architecture of the IoT network.

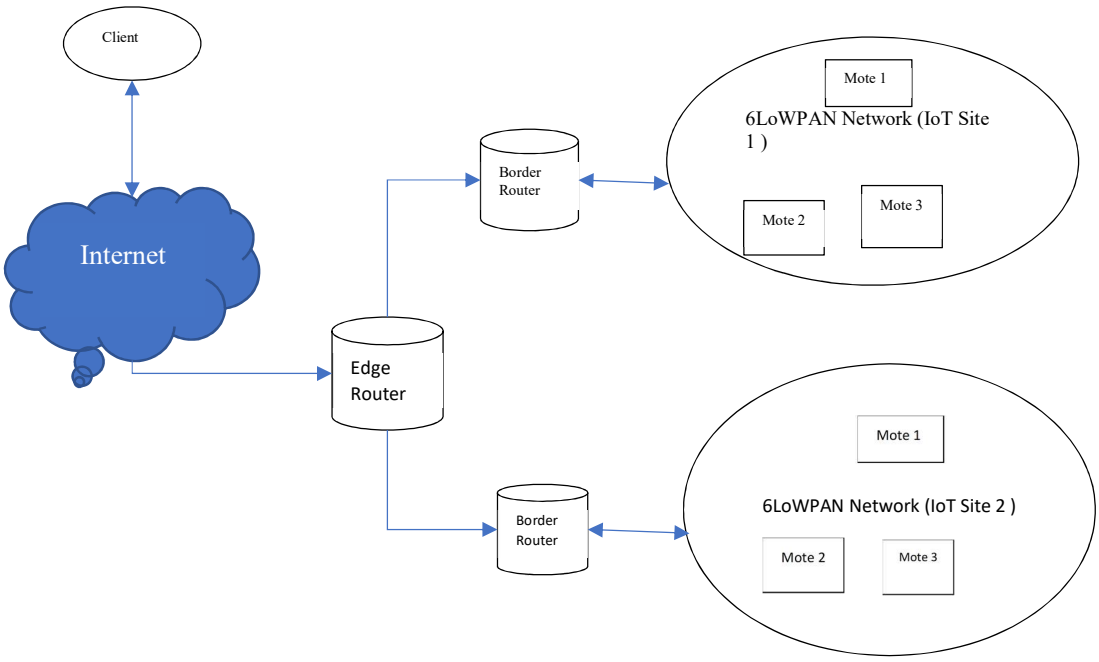


Figure 16 IoT Logical Network Architecture

5.10 Implementation

5.10.1 Test Bed

The logical network architecture is emulated on the test bed. The figure shows exactly how the test bed is created. Network namespaces are used in building the 6LoWPAN network. The setup was done in an Ubuntu 17.04 machine inside a virtual box. The elements that are discussed in the logical network architecture are emulated using the namespaces, thus having the multiple instances of the TCP/IP stack running on the single host.

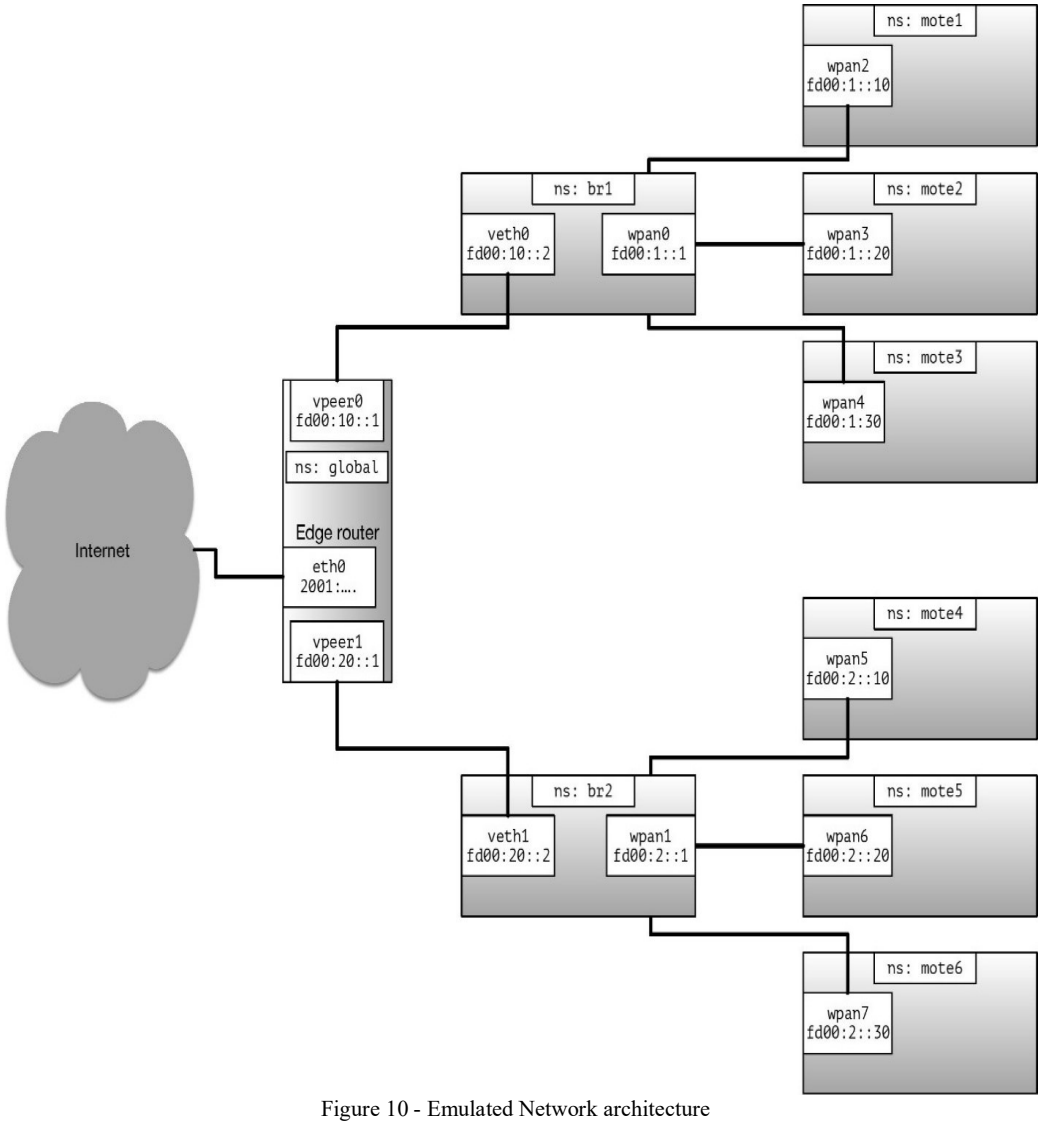


Figure 10 - Emulated Network architecture

Namespaces are used to create border routers and motes because, if they we were created in the global namespace, the Linux kernel would be able to apply IPv6 optimizations which may lead to behavior non-compliant with 6LoWPAN and even would be able to hide certain traffic which is not preferable.

The fakelb Linux module is used to create the IoT motes. They use a pre-defined set of radio settings which we modified as per our requirement. The communication is enabled by setting up the same Personal Area Network (PAN) id for all the IoT devices in an IoT site which enables communication between the motes. Border routers have two network interfaces, LoWPAN/WPAN interface to communicate with other IoT devices within the site and virtual Ethernet interface to communicate with the outside world. we use veth-vpeer pair to connect the border router and edge router, forwarding is enabled between the interfaces. Forwarding must be enabled between the edge and the border routers as well to communicate via the internet. Edge router is in the global namespace has three interfaces: two veth devices, each connecting to a border router, and regular Ethernet interface connecting to the Internet.

IoT devices use the light weight CoAP for communication which should be protected by using proxy based end-to-end DTLS and IPsec protocols. CoAP servers are deployed on the IoT motes and CoAP clients are setup in the global namespace. The communication is secured using end-to-end proxy based IPsec and (D)TLS protocols. Standard tests are performed to extract the desired performance metrics., elpased time, CPU utilization, network overhead and memory utilization are measured by doing suitable tests.

Performing the tests yield results on which statistical analysis is done using probability principles. We will establish tradeoffs among the protocols used in terms of throughput, response time, memory and CPU utilization which are the performance metrics considered and features of the respective protocols also will be taken into consideration.

5.10.2 IPSec VPN

As mentioned in the section 5.9 the client in our logical network architecture is located in the remote location and tries to access the resources on the IoT through the internet. To secure this type of scenario a remote access VPN has been set up. This type of configuration is also called as the Road warrior configuration.

In remote access VPN configuration, all client hosts will have the VPN client software. Whenever a host wants to send data to the protected network, the VPN client software encapsulates and encrypts the data before sending it over the public infrastructure to the target VPN gateway. When the gateway receives packet, it acts in the same way as the site-to-site peer decrypting and detaching of the IP header before forwarding it to its private subnet.

The IPsec VPN uses two types of key management methods. They are: Pre-shared key, Digital signatures. Here in this work we preferred to choose Digital certificates as it provides the flexibility of revoking the certificates of the clients rather than rather than re configuring every client if security of the client is compromised.

As we are using the public key cryptography with digital certificates a certificate authority (CA) is required. A certification authority is an entity that has trust of many users. If the organization is very large there will be many CAs which are arranged hierarchically. At the lower level, there will be the end users. The end users ask their intermediate CA to sign their certificate. The intermediate CA's certificate is signed by a CA from a higher level till the CA on the higher level is reached.

5.10.3 DTLS

Test COAP servers with support for the DTLS are deployed on the 6LoWPAN test nodes which are present in the separate namespaces. HTTP-COAP proxy server with TLS/DTLS support is deployed on the border router which is present in the namespace BR1. A test HTTP client program is deployed in the global namespace. The test COAP servers, HTTP-COAP proxy, Test COAP client use GNU TLS communication library to implement TLS and DTLS protocol.

The GNU TLS library is chosen because it is simple to use and integrated with the rest of the linux base libraries. This library consists a back-end which is designed to provide secure communication by keeping TLS and Public key infrastructure complexity outside the applications.
x.509 certificates which are generated above are used for the DTLS protocol also.

5.11 Tests Performed

To evaluate the performance of these protocols the following tests are made.

- **Test1: To send HTTP-GET request**
  The GET method request a representation of the specified resource. Here we send an HTTP -GET request to retrieve data from the COAP server. The HTPP- COAP mapping is done by the proxy server located at border router.
- **Test2: To send HTTP-POST request**
  The POST method is requests the COAP server to accept the data enclosed in the body of the message, most likely for storing.
- **Test3: To send HTTP-PUT request**
  The HTTP-PUT request method creates a new resource or replaces a representation of the target resource with the request payload. The PUT request is idempotent ie, calling it once or several times successively has the same effect.
- **Test4: Send a request with an unsupported method**:
  In this test a request is sent to COAP server with an unsupported method, that is the request sent by the client to server does not follow the rules of the server. The HTTP-COAP proxy does not map and send this request to the COAP server, but it simply responses the client with an error code 400, Bad request.
- **Test5: Send a request with an unsupported Accept header value:**
  In this test the client send an a request with an unsupported accept header value to the proxy server. The proxy response with a status code 502 bad gateway to the client.

6 RESULTS AND ANALYSIS

The main focus of the work is to implement and analyze the spliced connections between the server, client and the proxy. The above, mentioned tests are performed in the two scenarios.

Scenario 1: In scenario 1 the spliced TLS/DTLS communication is implemented and is compared with spliced TLS/Plain text communication.

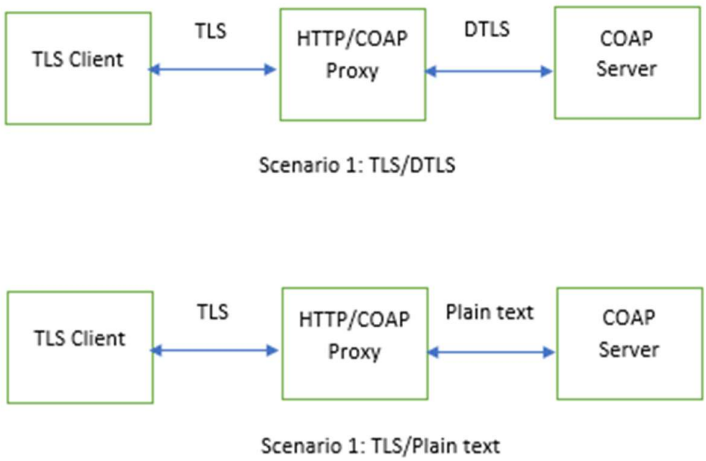


Figure 17 Scenario1

Scenario 2: In scenario 2 a remote access VPN is setup between the client and border router, from the border router to the IoT device DTLS communication is enabled. Thus, this is a spliced IPsec/DTLS communication is established and is compared with spliced IPSec/plain text communication.

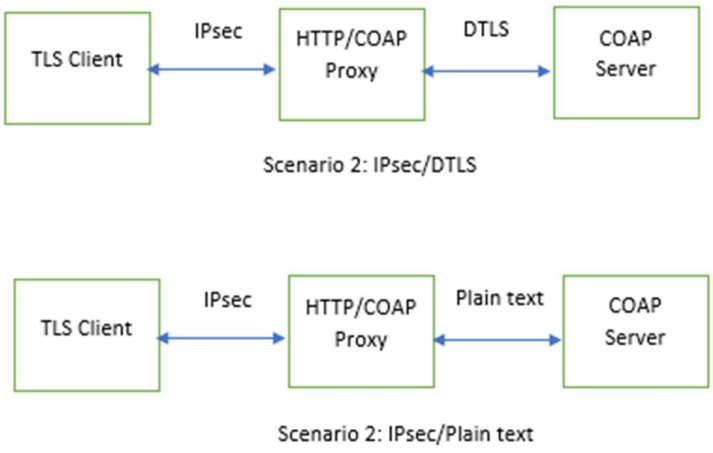


Figure 18 Scenario2

The following section explains the motivation for the considering the various performance metrics.

6.1 Performance Metrics

The main aim of the research is to evaluate the performance of the security protocols in the constrained devices with limited resources. Hence, we considered the impact of adding security on the resources that are available on constrained devices. The key metrics that are considered in this are research are:

6.1.1 CPU Utilization

As the constrained devices have the limited computational capability. CPU utilization has important implications on other system performance characteristics, namely power consumption, each additional percentage of CPU utilization consumes more input from the outlet. Hence it is important to study the impact of adding security on the CPU utilization in constrained devices. CPU Utilization is calculated using the following formula:

$$\%CPUutl = \left(\frac{TPkernel + TCkernel + TPuser + TCuser}{Total\ Elapsed\ time}\right) * 100$$

The CPU Utilization is calculated using PS command line utility.

6.1.2 Memory Utilization

As the constrained devices also have limited memory, it is important to analyze and estimate the amount of memory required for each process. Hence impact of adding security on the memory is also a crucial thing which needs to be considered. A high memory usage for the security may lead to the poor performance of the devices in other functions. Here in this research we considered the amount of the memory in RAM that has been allocated to the server and proxy during the implementation of the protocols that are considered in this research. In Linux, we have contents of the memory usage associated to a process are present in the file, /proc/pid/statm. From this file, we have taken the value of resident set size (RSS) which is the amount of memory in the RAM that has been allocated to process. We have also considered the total memory required for the execution the process.

6.1.3 Network Overhead

It is the amount of the additional data that has been transmitted over the network when compared to plain text communication. The constrained devices are designed to operate in the limited bandwidth. Hence it important to consider the amount of additional traffic that has been added due to the addition of security to the data. Wireshark is used to calculate the network overhead.

6.1.4 Elapsed Time

It is the amount of time the process has been running in CPU. Here in this research, elapsed time is the amount of time the server process has taken to serve the multiple requests from the client

The following sections provides the detailed analysis of the results that are obtained by implementing the two scenarios. The results for the two scenarios (for the various protocol combinations) are shown in the single graph for the better understanding and analysis.

6.2 Test1

To send HTTP-GET request: The graphs show below are for CPU utilization, memory utilization, network overhead, elapsed time when 100 HTTP-GET request are sent to the COAP server in each test. The test is performed in the two different scenarios. 30 iterations are made for each test. Average, standard deviation and 95% CI are calculated. (See section Appendix for values)

6.2.1 Scenario 1

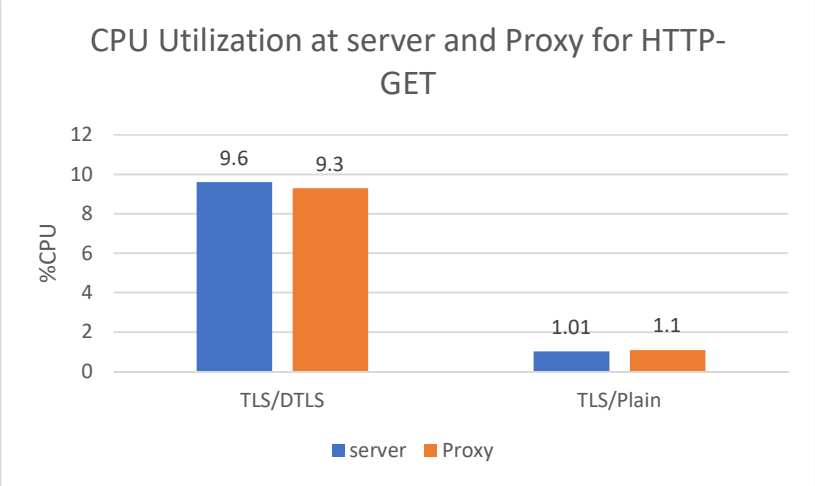


Figure 19 CPU Utilization at server and proxy for HTTP-GET

CPU Utilization: From the above graph, it can be observed that the CPU Utilization for the spliced TLS/DTLS is much higher when compared to the plain text communication. It is also observed that the CPU utilization at server is slightly higher when compared to CPU utilization at the proxy but in the spliced TLS/plain text communication the CPU utilization at the proxy is slightly greater than of the server.

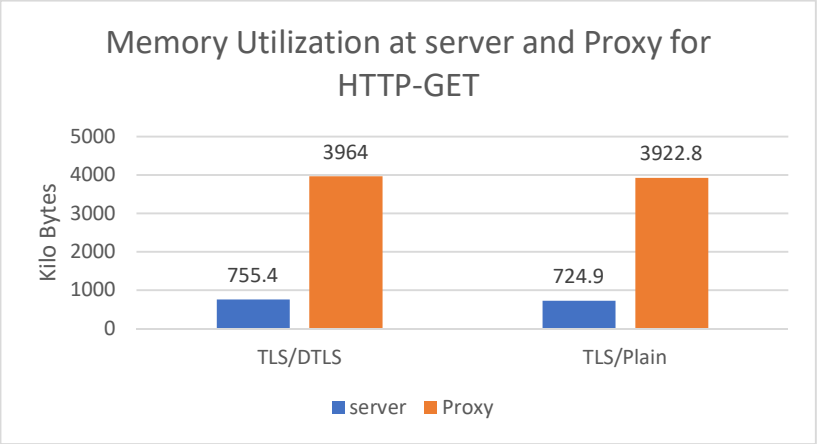


Figure 20 Memory Utilization at server and Proxy for HTTP-GET in scenario1

Memory Utilization: From the above graph, one can observe that the memory usage at the proxy is much higher when compared to the memory usage at the server in both spliced TLS/DTLS and spliced TLS/Plain communications. The memory usage at the COAP server is slightly greater in spliced TLS/DTLS when compared to the spliced TLS/Plain text communication. The memory usage at the proxy in spliced TLS/DTLS is slightly greater when compared to that of the spliced TLS/Plain. Thus, it can be concluded that the memory usage in spliced TLS/DTLS is slightly greater than that spliced TLS/Plain text when HTTP-GET method is used access the resource on the server. There is a very little impact on the memory usage in spliced TLS/DTLS at both the server and proxy when HTTP-GET method is used

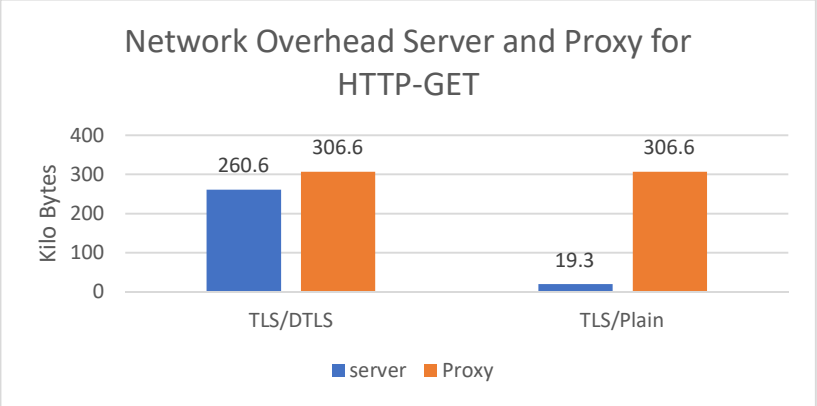


Figure 21 Network Overhead at the server and proxy for HTTP-GET in scenario1

Network Overhead: The network overhead is higher at both server and proxy when spliced TLS/DTLS is implemented due to the encryption of the data that is being transmitted.

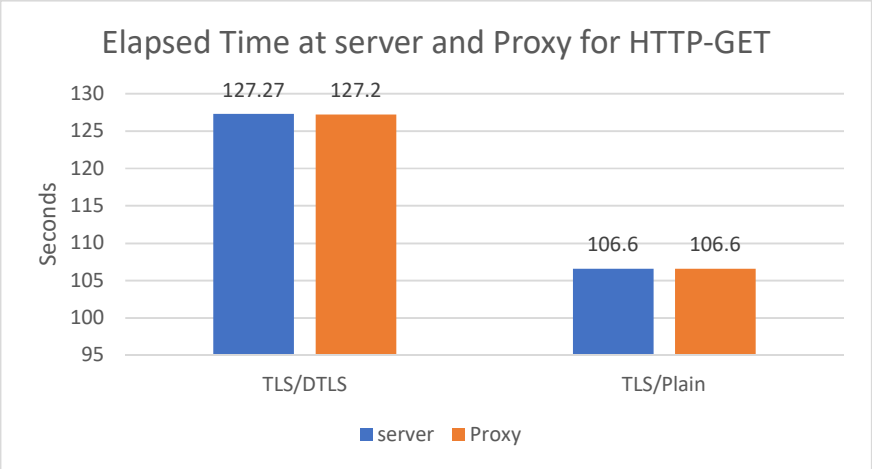


Figure 22 Elapsed Time at server and proxy for the HTTP-GET in scenario1

Elapsed Time: The elapsed time for the spliced TLS/DTLS and spliced TLS/Plain is almost same both at the server and proxy when HTTP-GET requests are sent. The Elapsed Time is higher in spliced TLS/DTLS communication at both the server and proxy when compared to that of spliced TLS/Plain

6.2.2 Scenario 2

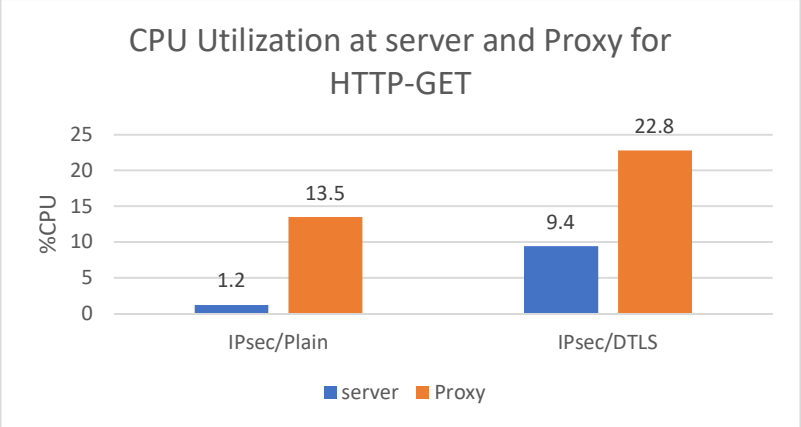


Figure 23 CPU Utilization at the server and proxy for HTTP-GET in scenario2

CPU Utilization: The CPU Utilization at the proxy is very high when spliced IPsec/DTLS is implemented. The CPU utilization at server is much greater in spliced IPsec/DTLS when compared to that of the spliced IPsec/Plain text communication. The CPU Utilization at both the server and the proxy is very when spliced IPSec/DTLS is implemented.

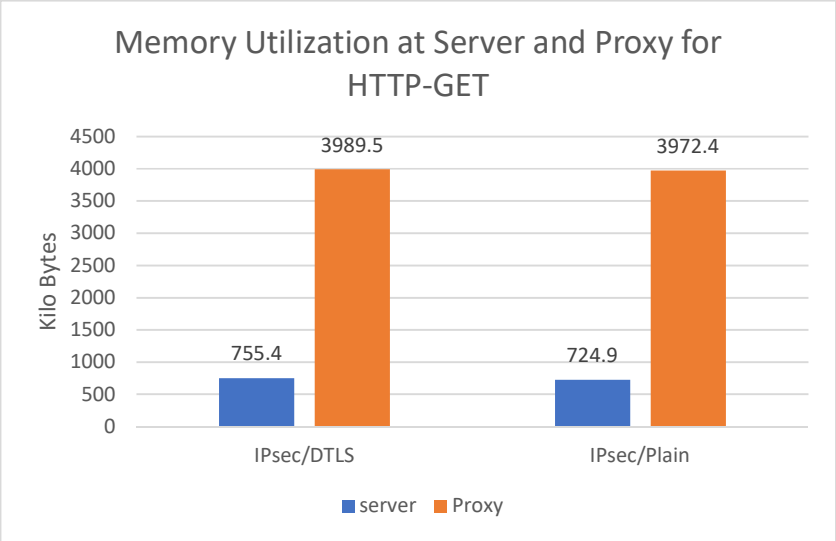


Figure 24 Memory Utilization at server and proxy for HTTP-GET in scenario2

Memory Utilization: The memory utilization at the proxy is very high when compared to that of the server in both the spliced connections. The proxy server uses high memory in spliced IPsec/DTLS when compared to that of the spliced IPsec/Plain. But in this scenario there is a very little impact on the memory usage at both server and proxy.

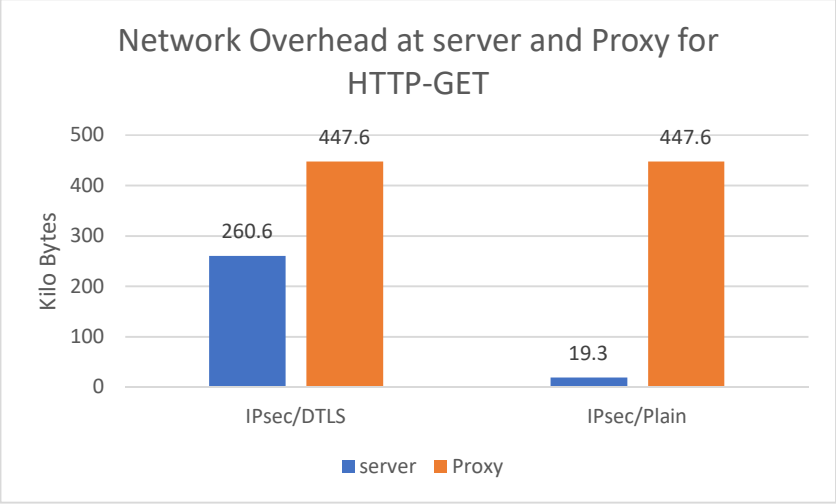


Figure 25 Network overhead at server and proxy for HTTP-GET in scenario2

Network Overhead: From the above figure, we can observe that the spliced IPsec/DTLS has more network overhead when compared to that of the spliced IPsec/Plain both at the server and proxy.

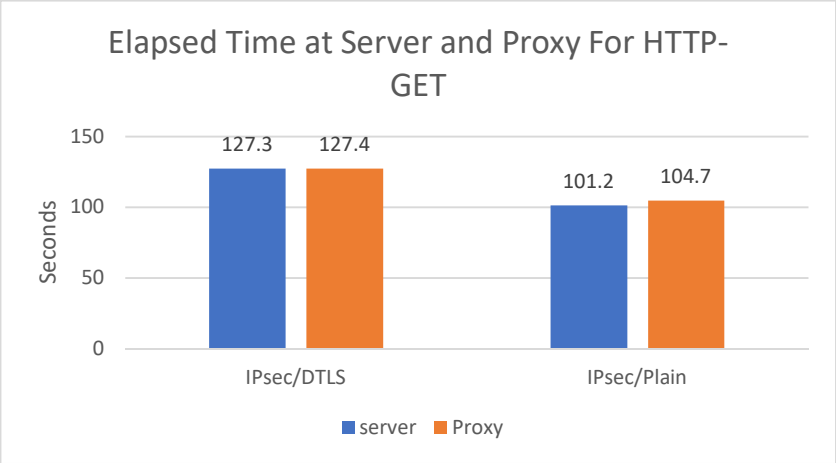


Figure 26 Elpased time at the server and proxy for HTTP-GET in scenario2

Elapsed Time: From the above figure, it can be observed elapsed time for the spliced IPsec/DTLS is higher when compared to that of the spliced IPsec/Plain both at server and the proxy.

6.3 Test2

To send HTTP-POST request: The graphs show below are for CPU utilization, memory utilization, network overhead, elapsed time when 100 HTTP-POST request are sent to the COAP server in each test. The test is performed in the two different scenarios. 30 iterations are made for each test. Average, standard deviation and 95% CI are calculated.

6.3.1 Scenario 1

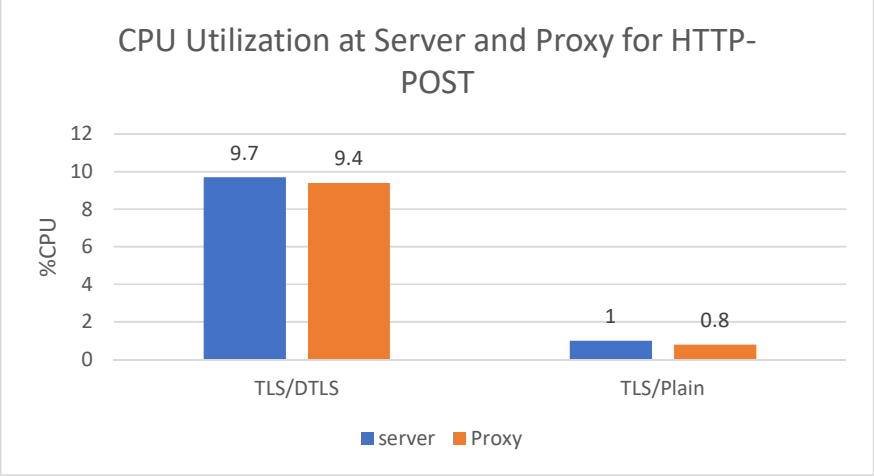


Figure 27 CPU Utilization at server and the proxy for HTTP-POST in scenario1

CPU Utilization: From the figure, it can be observed that the CPU utilization for the spliced TLS/DTLS is much higher when compared to that of the spliced TLS/Plain text communication. The CPU Utilization at the server is slightly higher when

compared to that of the proxy in spliced TLS/DTLS and also in the Spliced TLS/Plain communication.

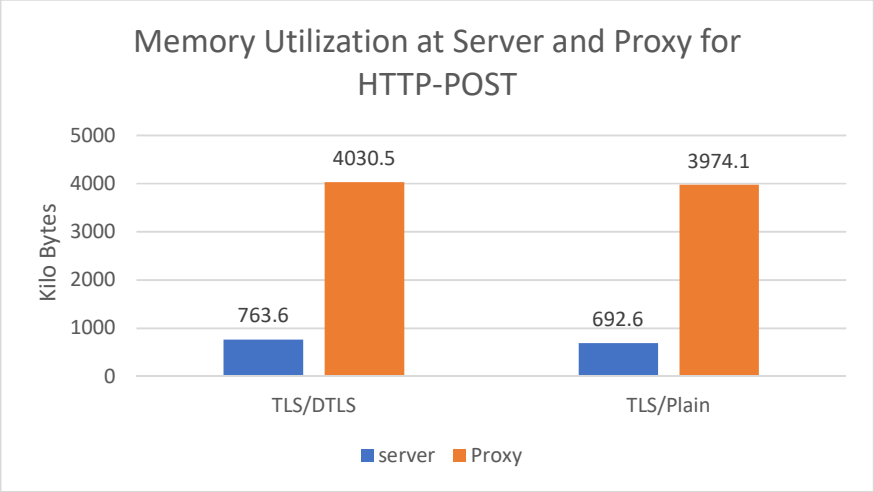


Figure 28 Memory Utilization at the server and proxy for HTTP-POST

Memory Utilization: From the above figure, we can observe that the memory utilization at the proxy is higher in the spliced TLS/DTLS when compared to that of spliced TLS/Plain communication. The memory utilization at server is slightly higher in spliced TLS/DTLS when HTTP-POST requests are sent to the server.

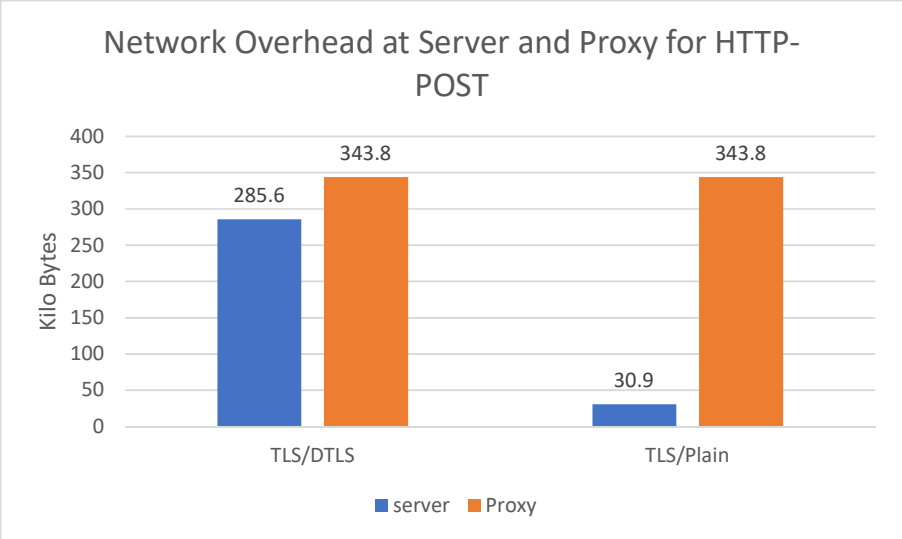


Figure 29 Network overhead at the server and proxy for HTTP-POST

Network overhead: From the above graph, one can observe that network overhead is higher at the proxy when compared to that server in both the spliced connections. But in spliced TLS/DTLS there is high network overhead at the server also. The network overhead remains constant at the proxy in the both spliced connections when HTTP-GET requests are sent to the server.

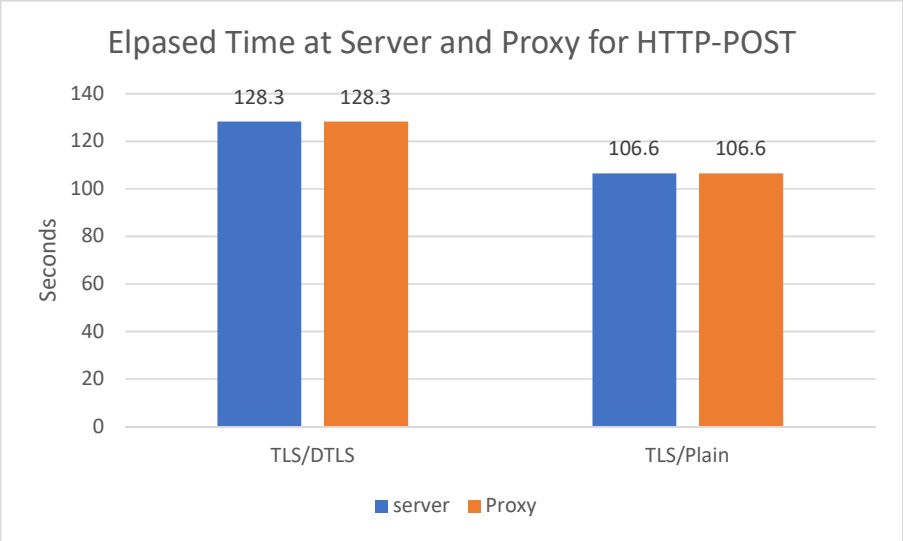


Figure 30 Elapsed Time at server and Proxy for HTTP-POST

.

Elapsed Time: The elapsed time is same when spliced TLS/DTLS and IPsec/DTLS is implemented. The TLS/plain has slightly higher elapsed time when compared to IPsec/Plain protocol combination.

6.3.2 Scenario 2

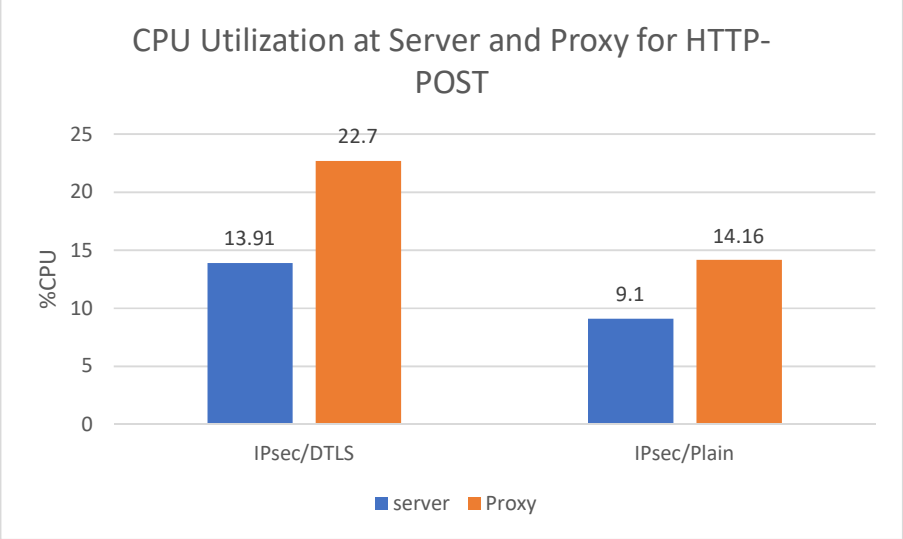


Figure 31 CPU Utilization at the server and proxy for HTTP-POST secnario2

CPU Utilization: From the above graph, one can observe that there is a high CPU utilization at the proxy when spliced IPsec/DTLS is implemented. There is a high CPU utilization at the server in spliced IPsec/DTLS when compared to that of IPsec/Plain.

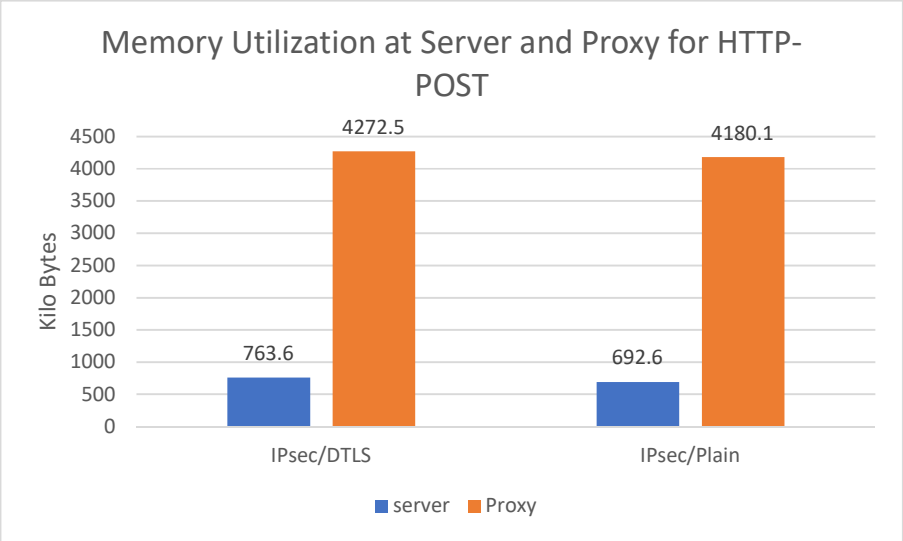


Figure 32 Memory Utilization at the server and the proxy for HTTP-POST for scenario2

Memory Utilization: From the above figure, it is evident that memory utilization at the proxy is much higher when compared to that of the server in both the spliced connections.

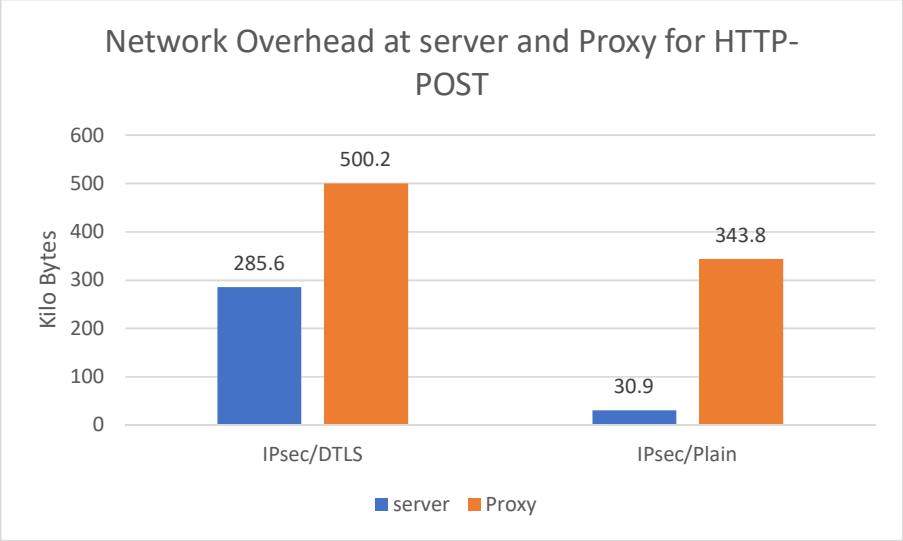


Figure 33 Network overhead at server and proxy for HTTP-POST

Network overhead: From the above graph, one can say that the network overhead in the spliced IPsec/DTLS is higher than that of spliced IPsec/Plain connection.

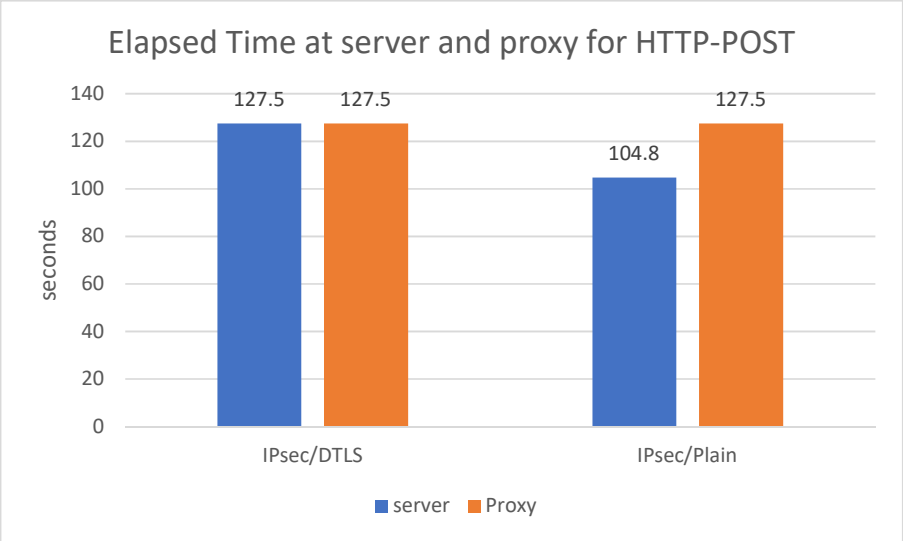


Figure 34 Elapsed time at server and proxy for HTTP-POST

Elapsed Time: The elapsed time at proxy and the server remains constant in spliced IPsec/DTLS. In spliced IPsec/plain connection the elapsed time the proxy is slightly higher when compared to the server.

6.4 Test3

To send HTTP-PUT request: The graphs show below are for CPU utilization, memory utilization, network overhead, elapsed time when 100 HTTP-PUT request are sent to the COAP server in each test. 30 iterations are made for each test. The test is performed in the two different scenarios. Average, standard deviation and 95% CI are calculated

6.4.1 Scenario 1

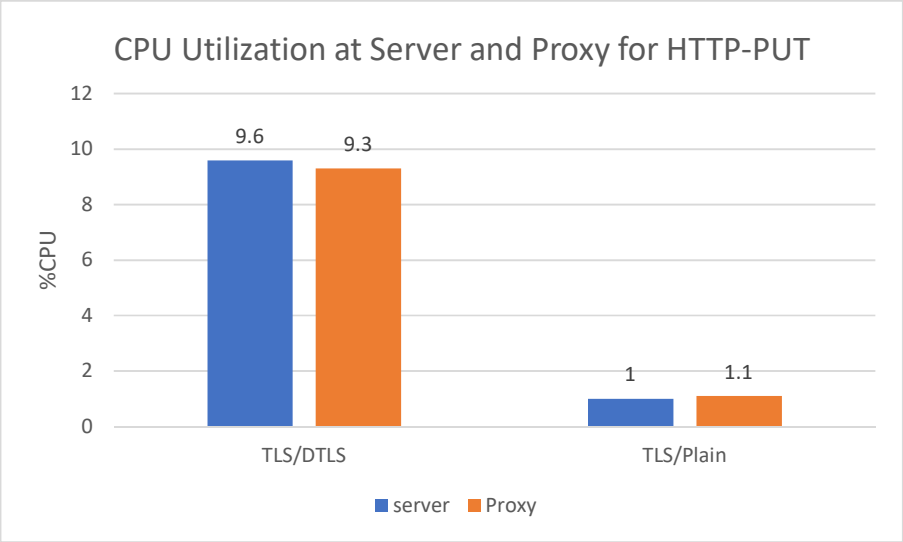


Figure 35 CPU utilization at the serer and the proxy for HTTP-PUT

CPU Utilization: From the above figure, it is evident that the CPU utilization for the spliced TLS/DTLS much higher when compared to that spliced TLS/Plain communication. The CPU Utilization at the server is slightly higher than that of the proxy when HTTP-PUT requests are sent where as in the spliced TLS/plain the CPU utilization at proxy is slightly higher when compared to that of the server.

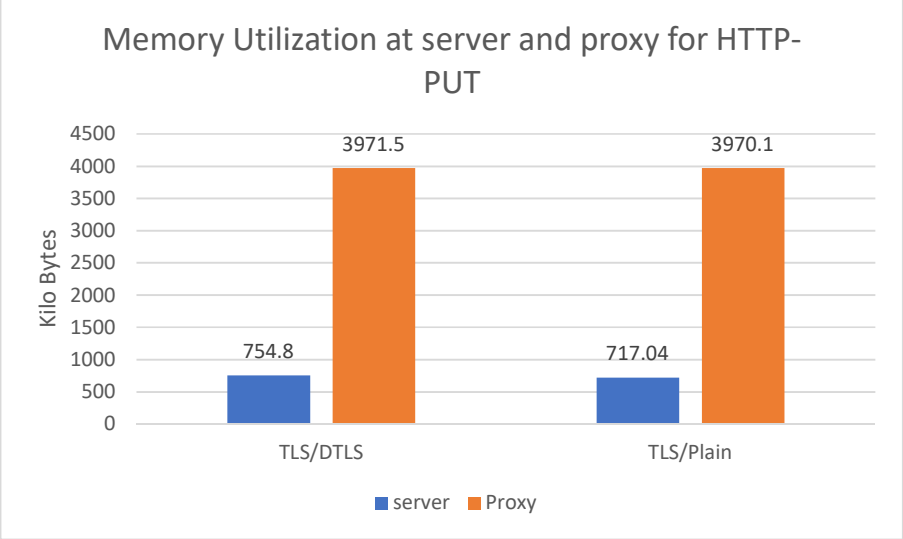


Figure 36 Memory utilization at the server and proxy for HTTP-PUT

Memory Utilization: The memory utilization at the proxy is higher when compared to that of the server in the both spliced connections when HTTP-PUT request are sent to the server.

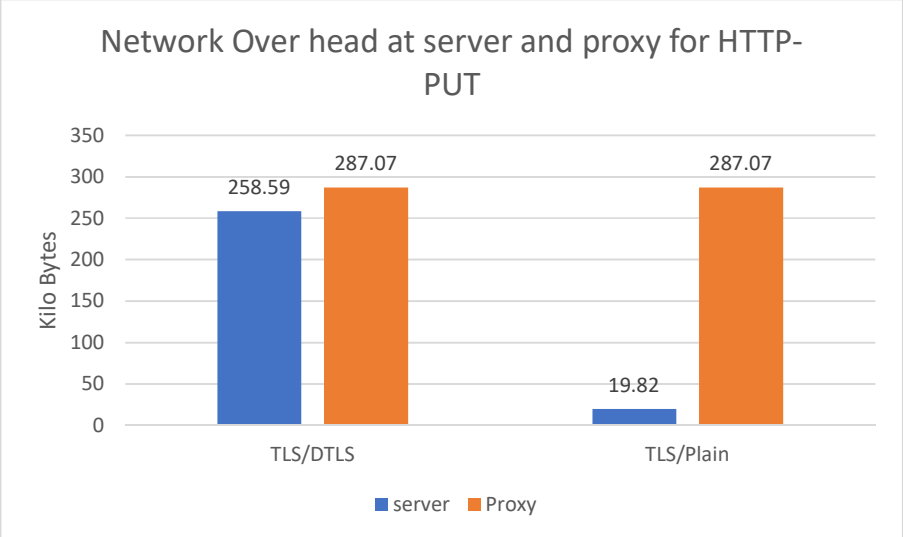


Figure 37 Network overhead at server and proxy for HTTP-PUT

Network Overhead: From the above graph it evident that the network overhead for the spliced TLS/DTLS is much higher when compared to that of the spliced TLS/Plain text communication**.**

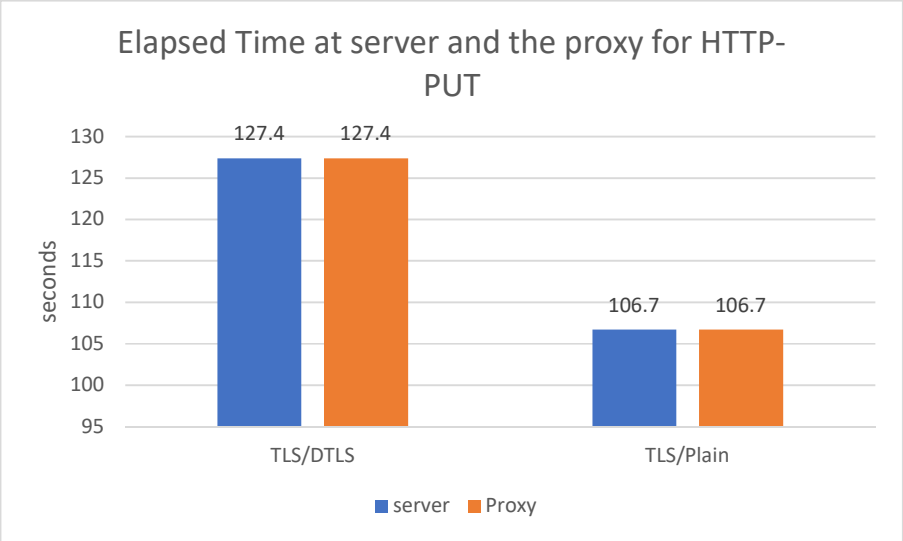


Figure 38 Elapsed Time at server and proxy for HTTP-PUT

Elapsed Time: The elapsed time is same server and the proxy in spliced TLS/DTLS and spliced TLS/Plain communications. The elapsed time in the spliced TLS/DTLS is much higher when compared to that of the spliced TLS/Plain communication.

6.4.2 Scenario 2

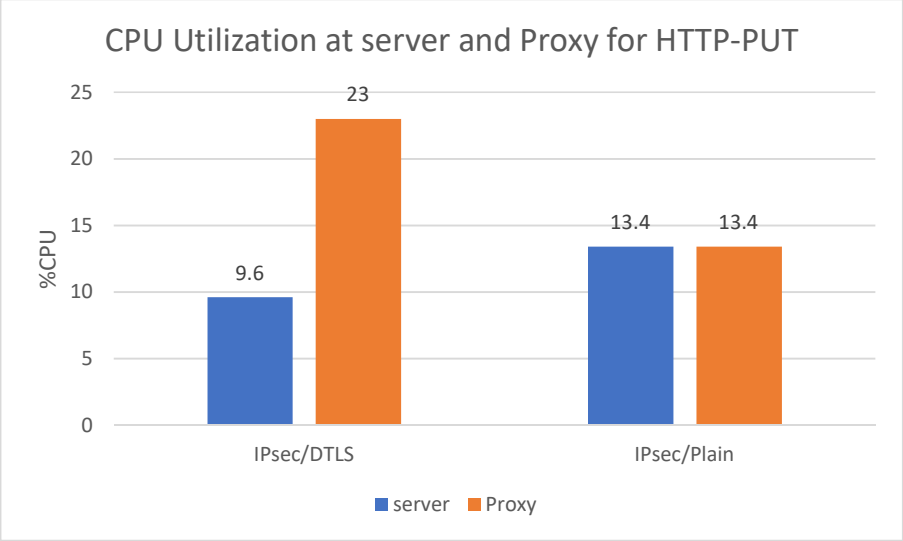


Figure 39 CPU utilization at server and proxy for HTTP-PUT in scenario2

CPU Utilization: From the above figure, one can observe that the CPU utilization at the proxy is much higher when compared to that of the CoAP server in spliced IPsec/DTLS. In spliced IPsec/Plain the CPU utilization almost remains constant.

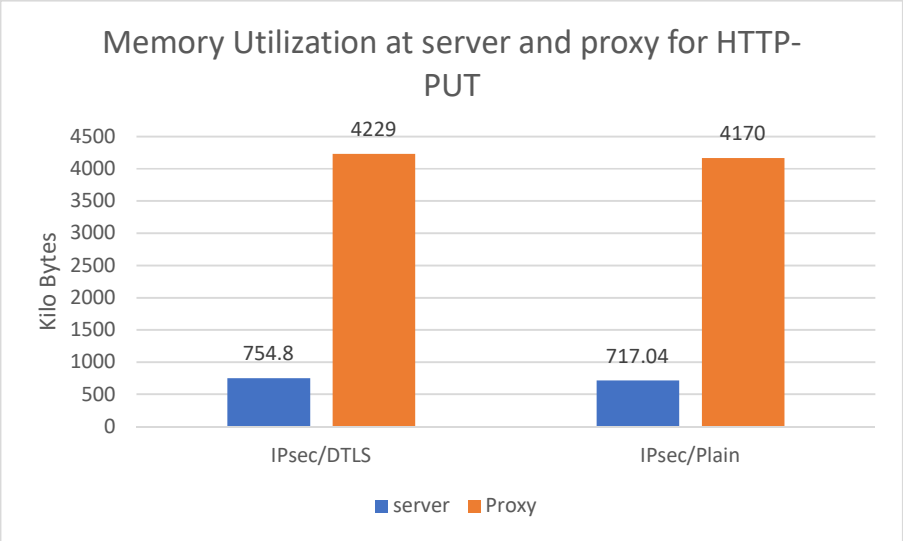


Figure 40 Memory utilization at server and proxy for HTTP-PUT in scenario2

Memory Utilization: The memory utilization at the proxy is higher when compared to that of the server in both spliced connections. There significantly high amount of the memory utilization at the proxy in spliced IPsec/DTLS connection when compared to that of the spliced IPsec/Plain communication.

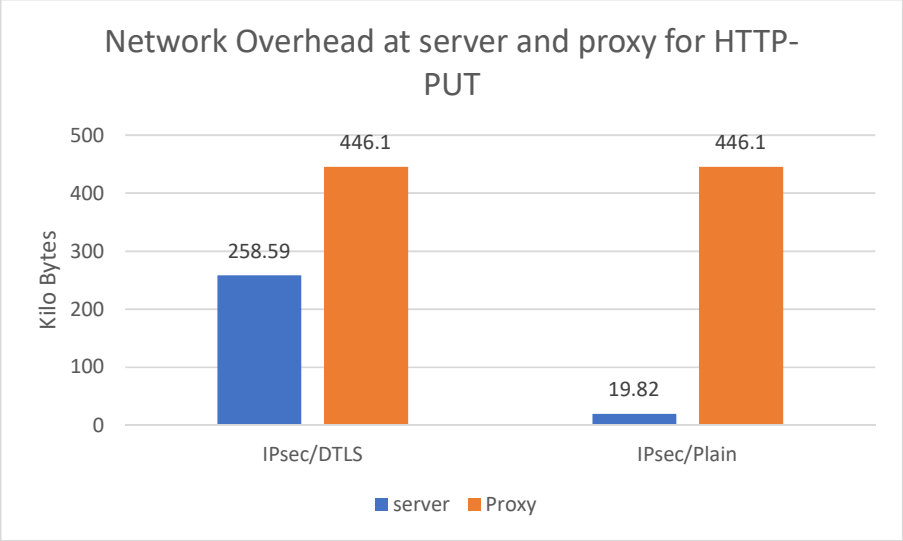


Figure 41 Network Overhead at server and proxy for HTTP-PUT in scenario2

Network Overhead: From the above figure, we can observe that there is a high network overhead at the proxy when compared to that at the server in both the spliced communications. There is a high network overhead at the server in spliced IPsec/DTLS when compared to that of the spliced IPsec/Plain communication.

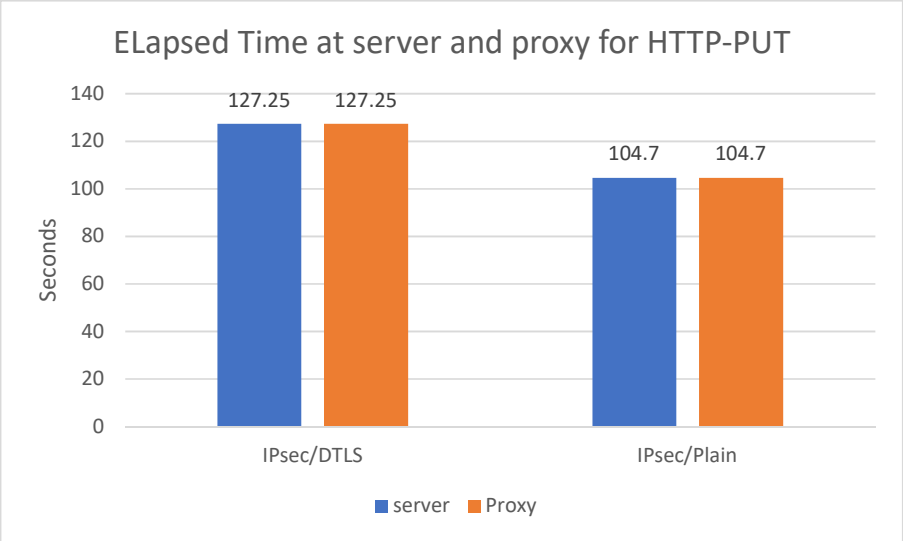


Figure 42: Elapsed Time for HTTP-PUT at server and Proxy

Elapsed Time: The elapsed time is same at the both server and the proxy in the two spliced connection scenarios. The elapsed time is high in spliced IPsec/DTLS when compared to that of the spliced IPsec/Plain text communication.

6.5 Test4

To send a request with an unsupported Accept header value: The graphs show below are for CPU utilization, memory utilization, network overhead, elapsed time when request with an unsupported Accept header value request are sent to the COAP server in each test. 30 iterations are made for each test. Average, standard deviation and 95% CI are calculated.

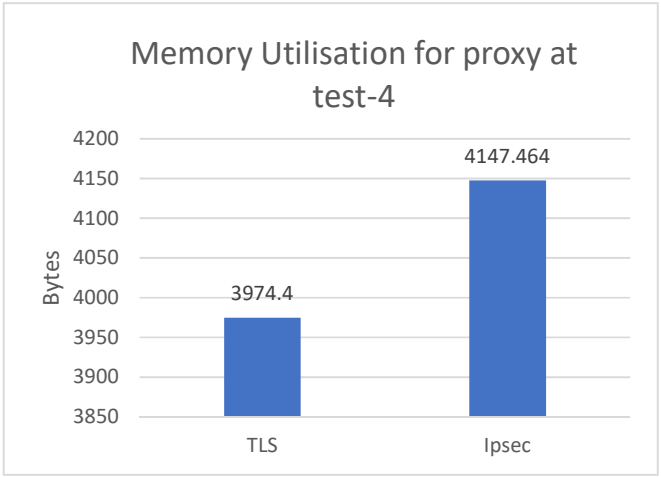


Figure 43: Memory utilization at proxy for test-4

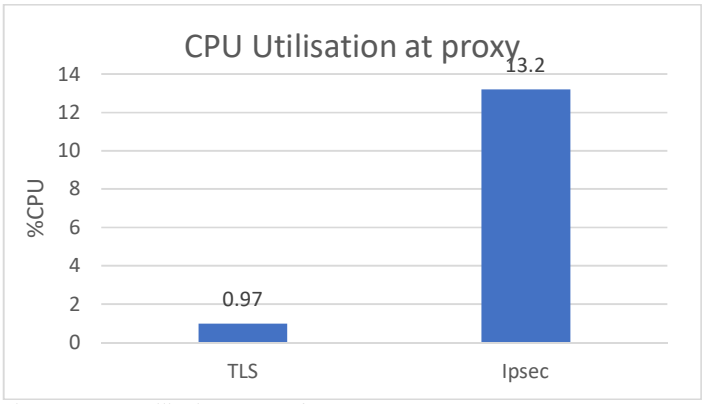


Figure 44: CPU Utilization at proxy for test-4

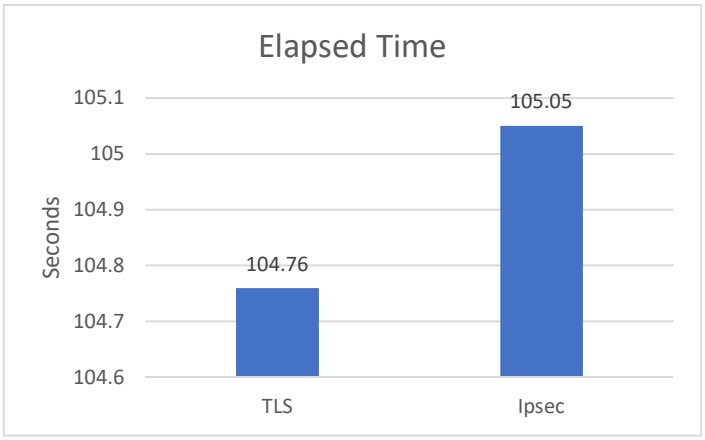


Figure 45: Elapsed Time at proxy for Test-4

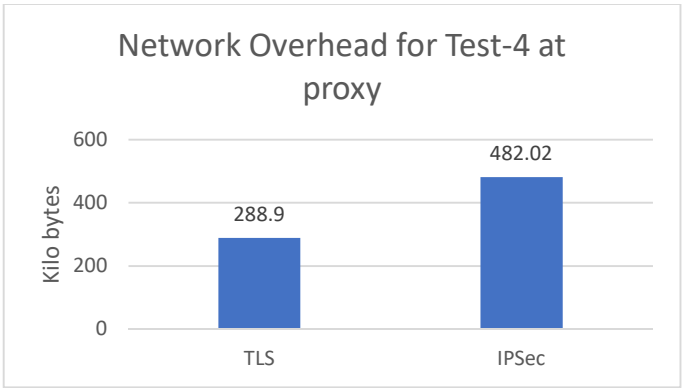


Figure 46: Network Overhead at proxy for Test-4

IPSec protocol has higher CPU Utilization, memory Utilization and network overhead when compared to the TLS protocol at proxy when a request with unsupported accept header value is sent to the Proxy by the client.

6.6 Test5

Send a request with an unsupported scheme in the request-URI
The graphs show below are for CPU utilization, memory utilization, network overhead, elapsed time when request with an unsupported scheme in the request-URI are sent to the COAP server in each test. 30 iterations are made for each test. Average, standard deviation and 95% CI are calculated.

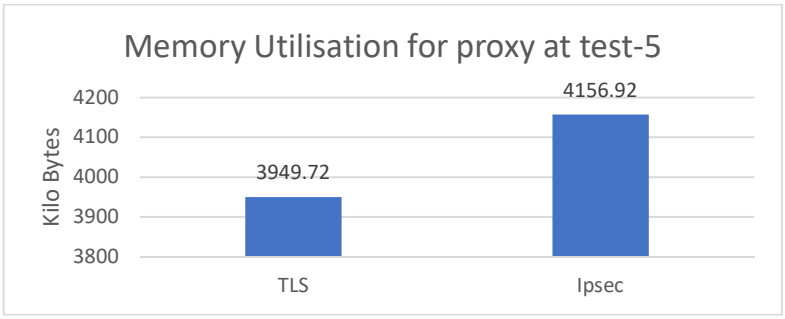


Figure 47: Memory Utilization at proxy for Test-5

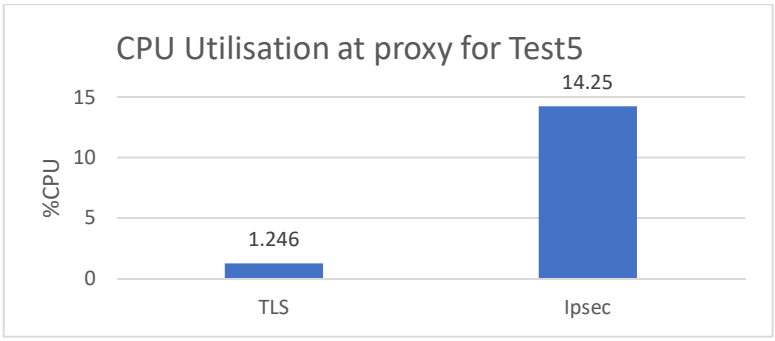


Figure 48: CPU Utilization at proxy for Test-5

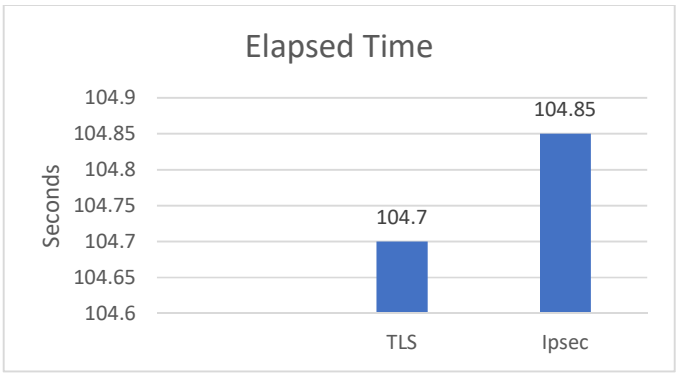


Figure 49:Elapsed Time at proxy for Test-5

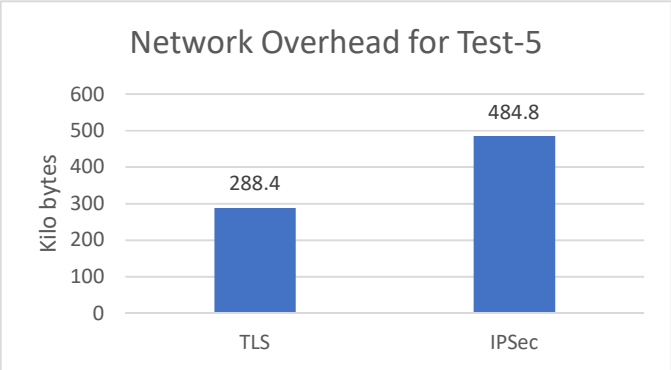


Figure 50: Network Overhead at proxy for test-5

IPSec protocol has higher CPU Utilization, memory Utilization and network overhead when compared to the TLS protocol at proxy when a request with unsupported accept header value is sent to the Proxy by the client.

6.7 Results from the previous research work

Paper [1]	Memory requirements in KB per DTLS handshake	
	ROM	RAM
State machine	8.15	1.9
Cryptography	3.3	1.5
Key management	1.0	0
DTLS record Layer	3.7	0.5
Total	16.18	3.9

[2]	IPSec	TLS	DTLS
Connection Establishment	6RTTs	3.5 RTTs	5 RTTs
Transmission Header size (Excluding IP)	105 Bytes	60 Bytes	60 Bytes

Processing Delay	IPSec	TLS	DTLS
AES encryption	750 µs	750µs	650µs
AES Decryption	500µs	450µs	400µs
SHA	1250 µs	800µs	650µs

TLS vs DTLS handshake comparison	TLS	DTLS
Message Type	1 bytes	1 bytes
Message Length	3 bytes	3 bytes
Message Sequence Number	Doesn't exist	2 bytes
Fragment Offset	Doesn't exist	3 bytes
Fragment Length	Doesn't exist	3 bytes

Total	4 bytes	12 bytes

TLS vs DTLS encryption overhead	TLS	DTLS
Record overhead	5 bytes	13
Encryption Algorithm	15 bytes	15
Integrity Algorithm	20 bytes	20
Total	40 bytes	48

[5]	Communication Overhead
802.15.4 Headers	168 Bytes
6LoWPAN Headers	480 Bytes
UDP Headers	96 Bytes
Application	487 Bytes

6.8 DTLS Limitations

The maximum frame-size for IEEE 802.15.4 standard is 127 bytes. After the header compressions, 75 bytes of data can be transmitted in a single frame. So, all DTLS packet must also be put into the available bytes. It prevents IP fragmentation but this leads to fragmentation at the handshake layer which in-turn adds extra overhead on the data transferred. Stateless cookie exchange adds an additional roundtrip to the handshake. When deployed on the constrained nodes I.e. IoT devices, due to the complexity of the protocol, significant impact on CPU utilization will be observed. Message buffer size on the server must be large enough to hold the multiple fragments for application level re-transmissions which requires encryption of every packet to be transmitted. Stateless cookie exchange adds an additional roundtrip to the handshake. DTLS does not support fragmentation of application data. If CoAP wants to avoid IP fragmentation, it must fit the data and all headers in a single IP packet. DTLS has a per-record overhead of 13 bytes for the record header. An implementation of CoAP with DTLS security will need to implement both the reliability mechanism for the DTLS handshake and the reliability mechanism of CoAP. This not only increases code size, but also prevents efficient retransmissions as each CoAP retransmission of the same data is a new transmission in DTLS. Long processing times can lead to spurious retransmissions.

DTLS protocol does not support multicast communications, which is an essential part of CoAP protocol and keys feature in the IoTs. DTLS handshake protocol could cause exhaustion attack of the resources of the battery powered devices even with stateless cookie mechanism. As a result, all nodes could lose their roles in the network and cause disruption to the entire communications. Third, even though DTLS could protect against replay attack by using bitmap window, nodes have to receive the packets first, process and sometimes even forward them. Without filtering proxy such as 6LoWPAN Border Router (6LBR), the possibility of this attack could render the network flooded. Managing such filtering on a 6LBR cannot by guaranteed on all scenarios. Moreover, the processing of replied packets is energy consuming. Handshake messages fragmentation stills an issue although a friendly solution was proposed without a validation. Besides, to verify the handshake messages, the hash function is needed to

be performed on all messages which mean larger buffer is needed for some nodes, and this is not applicable in each case. DTLS security features do not fit well for CoAP. For instance, the loss of a message in-flight requires the retransmission of all messages in-flight. On the other hand, if all messages in-flight are transmitted together in a single UDP packet, more resources are required for handling of large buffers. Further, If CoAP client needs Internet access that needs the CoAP/HTTP mapping process, and then DTLS handshake process remains a challenge. Particularly, it is not clear whether a partial mapping between TLS and DTLS can be performed. This issue could also be more complicated because a CoAP client would not be able to recognize which device has initiated the request. Finally, CoAP messages cost the network only 2 transactions (1 roundtrip); one message from the client (request) and the other from server (Response). If DTLS is used, 4 round trips are required; 3 round trips for DTLS (~ 40-50 Bytes) plus 1 round trip for CoAP before CoAP's actual contents are exchanged.

6.9 IPSec Limitations

We need to consider various issues such as computational power, memory size (RAM & ROM) and power management. Because the communication is wireless, network constraints such as low bitrates, variable delays and possibly high packet loss also needs to be considered. Although IPsec provides greater security, there are some issues in deploying the IPsec in the network. Connection establishment is slow, frequent connection drop-outs occur when implemented in the practical real network. There is reduction of the computing performance. Packet size is greatly increased which in-turn increases the network overhead. Policies need to be configured by applications. Some standard specifications even require that the application is aware of the underlying security mechanisms. This is not possible in IPsec. IPsec policies and security association parameters are tightly bound with IP addresses. Decisions cannot always be made based on port numbers, given that not all protocols use static port numbers.

- IPsec was originally not designed to deal with the constrained environments. There are certain issues that were not taken into consideration while designing the IPsec to make it suitable for the constrained environments.
- When transmitting small packets, the encryption process of IPsec generates a large overhead, thus diminishing the performance of the network.
- The mobility is an issue in IoTs when it comes to security Associations (SA). SA is uniquely identified by three parameters: Security Parameter Index (SPI), Destination IP address and security protocol identifier. If a node changes its IP address after the creation of the SA, then another SA needs to be created which will contribute to performance degradation.
- IPSec is embedded in the IP stack, thus any changes or modifications will require kernel level.
- Configuring/Managing/Troubleshooting IPSec and IKE are complex tasks giving huge number of constrained participating in the network. It might be case that misconfiguration security parameters of IPSec could lead to security holes and performance issues.
- The support to the multicast communication for the IPSec is difficult.
- Finally, according to CoAP's draft, it is possible to use IPSec (ESP) with layer-2 encryption hardware that supports the use of AES-CBC (128- bit keys).

However, if this approach is applicable, not all the devices can process the IPSec leading to operations complexities.

Besides as for mentioned concerns of both IPSec and DTLS protocols. We believe that the are some more common issues related to both the IPSec and DTLS protocols

- IPSec and DTLS require extra messages to negotiate the security parameters and setup security parameters and setup the security associations. This will not increase the overhead in the network but also drain out the resources in the constrained network.
- The proposed solution is based on the implementation of the spliced TLS, DTLS, IPSec which mean-+s the presence and support of these protocols is manadatory.
- IPSec and DTLS rely on other protocols such as the Internet Key Exchange (IKE) and the Extensible Authentication Protocol (EAP) for setting up the secure association; this implies that these extra protocols (IKE and EAP) need to be supported by all constrained devices vendors.
- IPSec and DTLS have originally been designed to secure connections between two static and remote devices, they strive to provide the most possible secure connection between the two ends, without considering the QoS, the network trustworthiness or any other limitations on the end devices. However, when considering providing security in the constrained environment, there is a need for more dynamic and sensible measures that consider the constrained nature of the end devices when negotiating the security parameters.

7 CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

RQ4: Is there a significant difference in performance overhead between TLS/DTLS and IPsec/DTLS?

From our detailed experimentation and analysis, we can conclude that there is a signification difference in the performance overhead between spliced TLS/DTLS and spliced IPSec/DTLS when compared to the spliced TLS/plain, Spliced IPSec/plain in the proxy based system for all the HTTP methods. Implementation of IPSec VPN between the border router and the client uses high amount of CPU at border router. There is a high memory usage at the server and the border router when compared to other protocol combinations in spliced IPSec/DTLS. The IPSec VPN imparts high overhead in the network when compared to the TLS/Plain, IPsec/Plain. The elapsed time for the IPSec/Plain, TLS/Plain is almost same.

RQ5: Based on performance overhead and limitations which protocol combination is best suited to provide confidentiality?

Based on the performance overhead and implementation limitations spliced TLS/DTLS uses less CPU, memory, Overhead, when compared to the IPSec/DTLS. The spliced IPSec/DTLS Provides better security when compared to the spliced TLS/DTLS but uses more resources which is not accepted in the constrained devices. So, TLS/DTLS combination provides better security in terms of confidentiality with minimal usage of the resources when compared to the Spliced IPsec/DTLS.

We also conclude that Through our observation and analysis, providing E2E security in IoT is very important aspect, mainly due to many possible usage scenarios, i.e, IPSec/DTLS, IPSec/Plain and TLS/Plain, TLS/DTLS, that have different constraints and requirements. Our analysis also reveals that having a secure E2E connection between two end hosts only provides a secure communication channel; the constrained devices can still be vulnerable to resource exhaustion, flooding, replay and amplification attacks.

7.2 Future Work

This research mainly focuses on the impact of adding security in the constrained networks with respect to CPU utilization, memory utilization, network overhead and elapsed time. It would be interesting to study the power consumption the constrained devices when security is enabled. The same work can be further extended to the other security protocols like HIP and implementing them on the real devices.

REFERENCES

[1] S. L. Keoh, S. S. Kumar, and H. Tschofenig, "Securing the Internet of Things: A Standardization Perspective," *IEEE Internet Things J.*, vol. 1, no. 3, pp. 265–275, Jun. 2014.

[2] R. Z. Khan and A. Shiranzaei, "IPv6 security tools #x2014; A systematic review," in *2016 International Conference on Computing, Communication and Automation (ICCCA)*, 2016, pp. 459–464.

[3] A. Hussein, I. H. Elhajj, A. Chehab, and A. Kayssi, "Securing Diameter: Comparing TLS, DTLS, and IPSec," in *2016 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET)*, 2016, pp. 1–8.

[4] R. A. Rahman and B. Shah, "Security analysis of IoT protocols: A focus in CoAP," in *2016 3rd MEC International Conference on Big Data and Smart City (ICBDSC)*, 2016, pp. 1–7.

[5] O. Garcia-Morchon, S. L. Keoh, S. Kumar, P. Moreno-Sanchez, F. Vidal-Meca, and J. H. Ziegeldorf, "Securing the IP-based Internet of Things with HIP and DTLS," in *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, New York, NY, USA, 2013, pp. 119–124.

[6] S. Unterschütz, A. Weigel, and V. Turau, "Cross-platform Protocol Development Based on OMNeT++," in *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, ICST, Brussels, Belgium, Belgium, 2012, pp. 278–282.

[7] "A 6LoWPAN model for OMNeT++." [Online]. Available: http://dl.acm.org/citation.cfm?id=2512781. [Accessed: 29-Aug-2017].

[8] C. Hennebert and J. D. Santos, "Security Protocols and Privacy Issues into 6LoWPAN Stack: A Synthesis," *IEEE Internet Things J.*, vol. 1, no. 5, pp. 384–398, Oct. 2014.

[9] G. Glissa and A. Meddeb, "6LoWPAN multi-layered security protocol based on IEEE 802.15.4 security features," in *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2017, pp. 264–269.

[10] A. Capossele, V. Cervo, G. D. Cicco, and C. Petrioli, "Security as a CoAP resource: An optimized DTLS implementation for the IoT," in *2015 IEEE International Conference on Communications (ICC)*, 2015, pp. 549–554.

[11] V. Lakkundi and K. Singh, "Lightweight DTLS implementation in CoAP-based Internet of Things," in *20th Annual International Conference on Advanced Computing and Communications (ADCOM)*, 2014, pp. 7–11.

[12] Y. Maleh, A. Ezzati, and M. Belaissaoui, "An enhanced DTLS protocol for Internet of Things applications," in *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, 2016, pp. 168–173.

[13] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, and G. Carle, "A DTLS based end-to-end security architecture for the Internet of Things with two-way authentication," in *37th Annual IEEE Conference on Local Computer Networks - Workshops*, 2012, pp. 956–963.

[14] S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig, "Securing communication in 6LoWPAN with compressed IPsec," in *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*, 2011, pp. 1–8.

[15] S. Raza, D. Trabalza, and T. Voigt, "6LoWPAN Compressed DTLS for CoAP," in *2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems*, 2012, pp. 287–289.

[16] M. Lavanya and V. Natarajan, "Certificate-free collaborative key agreement based on IKEv2 for IoT," in *2017 Third International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB)*, 2017, pp. 397–400.

[17] S. Vohra and R. Srivastava, “A Survey on Techniques for Securing 6LoWPAN,” in *2015 Fifth International Conference on Communication Systems and Network Technologies*, 2015, pp. 643–647.

[18] A. P. Castellani, T. Fossati, and S. Loreto, “HTTP-CoAP cross protocol proxy: an implementation viewpoint,” in *2012 IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2012)*, 2012, vol. Supplement, pp. 1–6.

[19] M. Brachmann, S. L. Keoh, O. G. Morchon, and S. S. Kumar, “End-to-End Transport Security in the IP-Based Internet of Things,” in *2012 21st International Conference on Computer Communications and Networks (ICCCN)*, 2012, pp. 1–5.

[20] T. A. Alghamdi, A. Lasebae, and M. Aiash, “Security analysis of the constrained application protocol in the Internet of Things,” in *Second International Conference on Future Generation Communication Technologies (FGCT 2013)*, 2013, pp. 163–168.

[21] A. B. Sulaeman, F. A. Ekadiyanto, and R. F. Sari, “Performance evaluation of HTTP-CoAP proxy for wireless sensor and actuator networks,” in *2016 IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob)*, 2016, pp. 68–73.

[22] C. Matthias, S. Kris, B. An, S. Ruben, M. Nele, and A. Kris, “Study on impact of adding security in a 6LoWPAN based network,” in *2015 IEEE Conference on Communications and Network Security (CNS)*, 2015, pp. 577–584.

[23] A. Rghioui, M. Bouhorma, and A. Benslimane, “Analytical study of security aspects in 6LoWPAN networks,” in *2013 5th International Conference on Information and Communication Technology for the Muslim World (ICT4M)*, 2013, pp. 1–5.

[24] E. Dijk, A. Rahman, T. Fossati, S. Loreto, and A. Castellani, “Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP).” [Online]. Available: https://tools.ietf.org/html/rfc8075. [Accessed: 13-Sep-2017].

[25] Z. Shelby, K. Hartke, and C. Bormann, “The Constrained Application Protocol (CoAP).” [Online]. Available: https://tools.ietf.org/html/rfc7252. [Accessed: 13-Sep-2017].

[26] T. Dierks <tim@dierks.org>, “The Transport Layer Security (TLS) Protocol Version 1.2.” [Online]. Available: https://tools.ietf.org/html/rfc5246. [Accessed: 13-Sep-2017].

[27] E. Rescorla and N. Modadugu, “Datagram Transport Layer Security Version 1.2.” [Online]. Available: https://tools.ietf.org/html/rfc6347. [Accessed: 13-Sep-2017].

[28] P. K. Kamma, C. R. Palla, U. R. Nelakuditi, and R. S. Yarrabothu, “Design and implementation of 6LoWPAN border router,” in *2016 Thirteenth International Conference on Wireless and Optical Communications Networks (WOCN)*, 2016, pp. 1–5.

[29] “Securing the Internet of Things: A Proposed Framework,” *Cisco*. [Online]. Available: https://www.cisco.com/c/en/us/about/security-center/secure-iot-proposed-framework.html. [Accessed: 13-Sep-2017].

[30] “strongSwan - Documentation.” [Online]. Available: https://strongswan.org/documentation.html. [Accessed: 13-Sep-2017].

[31] N. Kushalnagar, G. Montenegro, D. E. Culler, and J. W. Hui, “Transmission of IPv6 Packets over IEEE 802.15.4 Networks.” [Online]. Available: https://tools.ietf.org/html/rfc4944. [Accessed: 13-Sep-2017].

[32] P. Thubert and J. W. Hui, “Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks.” [Online]. Available: https://tools.ietf.org/html/rfc6282. [Accessed: 13-Sep-2017].

[33] “linux-wpan.” [Online]. Available: http://wpan.cakelab.org/. [Accessed: 13-Sep-2017].

[34] M. Badra and I. Hajjeh, “Enabling VPN and Secure Remote Access using TLS Protocol,” in *2006 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, 2006, pp. 308–314.

[35] S. Raza, S. Duquennoy, T. Voigt, and U. Roedig, “Demo abstract: Securing communication in 6LoWPAN with compressed IPsec,” in *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*, 2011, pp. 1–2.

8 APPENDIX

8.1 IPsec Implementation

As the border router in our case runs in the separate namespace called br1 and the global namespace acts as the edge router in the present test scenario. Therefore, multiple instances of the StrongSwan needs to be run in separate namespaces. Every file found in /etc/netns/<name> for a given netns is bind mounted over its corresponding counter-part in /etc/. This can be used to provide different config files for each instance, but may also be used to redirect the so called piddir, where the Charon and starter daemons create their PID files and UNIX sockets.

So, in order to run the multiple instances of the StrongSwan in the separate namespaces, the StrongSwan should be configured and installed with –piddir =/etc/ipsec.d/run. After the installation of StrongSwan the corresponding pidirs are created for the separate namespaces.

The top-level CA is referred as the root CA. The certificate of the root CA is self-signed. To ensure the security the complete chain of the certificates needs to be verified. A root CA and intermediate CA is created with their separate configuration files. Private keys are generated for the root CA, intermediate CA, edge router and the border router which are of length 2048 bytes. A self-signed certificate was generated for the root CA. To create a certificate for the intermediate CA as certificate signing request (CSR) was generated on behalf of the intermediate CA. The CSR contains the intermediate CA's public key and an identifier for a digital signature algorithm. The CSR is signed using the intermediate CA's private key using the specified. Intermediate CA certificate is generated using CSR. After the generation of the certificates a certificate chain file is generated which contain intermediate CA and the root CA certificate. This is done as the end user applications (In our case VPN clients) needs to verify the entire certificate chain.
Similarly, CSRs are generated for the edge router and border router. The certificates are generated using the CSRs.

- The intermediate CA certificate, root certificate and certificate chain file are kept in the /etc/ipsec.d/cacerts and also in /etc/wpan1/ipsec.d/cacerts folder.
- The certificate of border router is kept in the /etc/wpan1/ipsec.d/certs.
- The certificate of the edge router is kept in the /etc/ipsec.d/certs.
- The private key of the edge router should be kept in /etc/ipsec.d/private folder
- The private key of the border router should be kept in /etc/wpan1/private folder.

8.2 Test6

Send a request with an unsupported method

The graphs show below are for CPU utilization, memory utilization, network overhead, elapsed time when request with an unsupported method are sent to the COAP server in each test. 30 iterations are made for each test. Average, standard deviation and 95% CI are calculated.

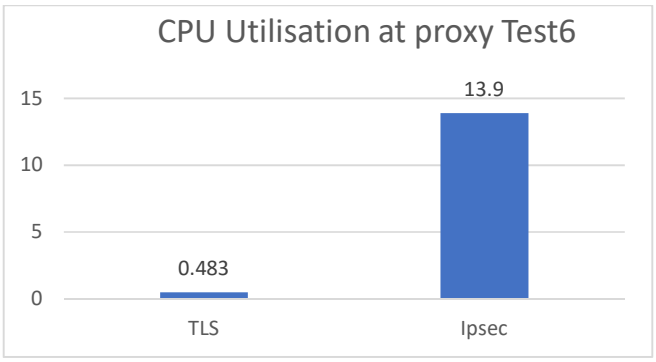


Figure 51: CPU Utilization at proxy for Test-6

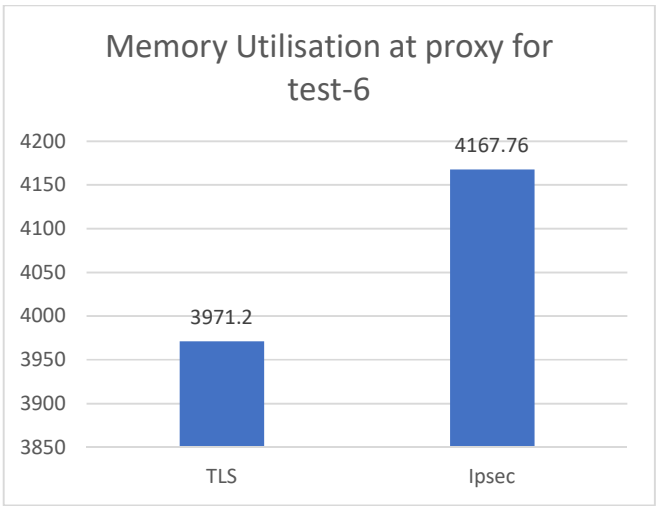


Figure 52: Memory Utilization at proxy for test-6

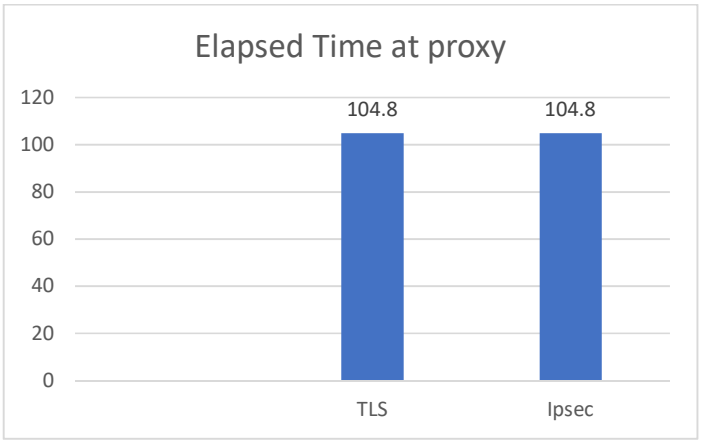


Figure 53: Elapsed Time at proxy for Test-6

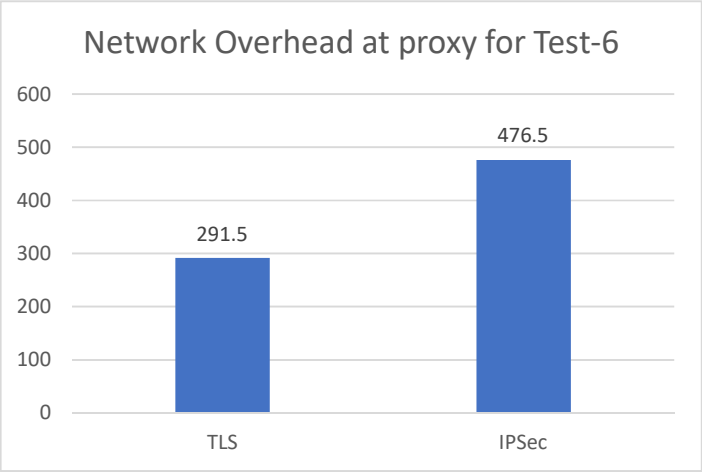


Figure 54: Network overhead ar proxy for Test-6

IPSec protocol has higher CPU Utilization, memory Utilization and network overhead when compared to the TLS protocol at proxy when a request with unsupported accept header value is sent to the Proxy by the client.

CPU Utilization server

		TLS/DTLS	TLS/PLAIN	IPSEC/DTLS	IPSEC/Plain
TEST-1	Average	9.4	1.01	9.44	13.26
	Standard Deviation	0.32	0.04	0.41	1.36
	95% CI	0.12	0.3	0.23	0.4
TEST-2	Average	9.73	1.03	9.50	13.9
	Standard Deviation	0.22	0.56	0.8	1.1
	95% CI	0.08	0.23	0.4	0.3
TEST-3	Average	9.62	1.01	9.62	13.48
	Standard Deviation	0.15	0.05	0.12	1.13
	95% CI	0.05	0.2	0.05	0.3

CPU Utilization at proxy

		TLS/DTLS	TLS/PLAIN	IPSEC/DTLS	IPSEC/Plain
TEST-1	Average	9.33	1.12	22.84	13.45
	Standard Deviation	0.31	0.5	0.27	0.7
	95% CI	0.12	0.18	0.14	0.15
TEST-2	Average	9.73	1.12	22.70	14.16
	Standard Deviation	0.26	0.8	0.7	1.0
	95% CI	0.09	0.4	0.3	0.7
	Average	9.33	1.11	23	13.48

TEST-3	Standard Deviation	0.26	0.3	0.12	0.6
	95% CI	0.97	0.7	0.05	0.4
TEST-4	Average	0.97	1.12	13.2	13.48
	Standard Deviation	0.09	0.5	0.1	1.25
	95% CI	0.03	0.06	0.01	0.78
TEST-5	Average	1.4	1.1	14.25	13.56
	Standard Deviation	1.4	0.5	0.001	1.14
	95% CI	1.54	0.23	0.2	0.3
TEST-6	Average	0.04	1.11	13.9	13.6
	Standard Deviation	0.12	0.5	0.001	0.4
	95% CI	0.04	0.02	0.1	0.8

Memory Utilization at server:

		DTLS	PLAIN
TEST-1	Average	755	725
	Standard Deviation	47.16	33.23
	95% CI	17.94	15.89
TEST-2	Average	764	693
	Standard Deviation	31.90	32.65
	95% CI	11.91	9.68
TEST-3	Average	755	717
	Standard Deviation	39.76	37.4
	95% CI	14.84	18.5

Memory utilization at proxy for test (1 to 3)

		TLS/DTLS	TLS/PLAIN	IPSEC/DTLS	IPSEC/Plain
TEST-1	Average	3964	3922	3989	3972
	Standard Deviation	91.16	76.23	50.50	49.24
	95% CI	34.67	40.13	6.99	25.26
TEST-2	Average	4030	4272	3974	4180
	Standard Deviation	95.30	85.56	146.33	152
	95% CI	47.50	45	18.19	20.36
TEST-3	Average	3972	4171	3970	4229
	Standard Deviation	124.95	98.3	108.26	95.3
	95% CI	46.65	15.23	11.42	13.48

Memory utilization at proxy for test (4 to 6)

		TLS	IPsec
TEST-4	Average(kb)	3949	4156
	Standard Deviation	145.70	170.19
	95% CI	54.4	18.19
TEST-5	Average(kb)	3974	4147
	Standard Deviation	124.84	170.19
	95% CI	46.61	18.19
TEST-6	Average(kb)	3971	4167
	Standard Deviation	103.86	140.0
	95% CI	38.76	18.2