Master of science in Electrical Engineering with emphasis on Telecommunication Systems

Thesis no: MSEE-2017: 42

Performance analysis of end-to-end DTLS and IPsec-based communication in IoT environments

Security and Privacy ~ Distributed systems security

Kuna Vignesh

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):
Kuna Vignesh
E-mail:
Kuna.vignesh@gmail.com
viku16@student.bth.se

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

University Examiner:
Dr. Adrian Popescu
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 end-to-end DTLS and IPsec protocols in an IoT environment.

Objectives: The aim of this work is to investigate the performance of end-to-end communication with an IoT environment over DTLS and IPsec, respectively. The objectives of this project are:

- To implement end-to-end DTLS and IPsec, respectively, in an emulated IoT environment.
- To extract performance metrics such as CPU and memory utilization, elapsed time and network overhead for DTLS and IPsec.
- Compare DTLS and IPsec based on protocol features and statistical analysis of the data.

Methods: A virtual network will be setup consisting of the server, client and the border router. 6LoWPAN devices are emulated in the Linux kernel in different network namespaces which will be the servers. Clients in our case run in the global namespace and invoke server methods remotely. Communication is done via the border router that is implemented in one of the namespace and will be secured using DTLS or IPsec protocols. Tests are performed implementing DTLS and IPsec over COAP to extract the performance metrics. Statistical analysis is done on performance metrics CPU and memory utilization, elapsed time and network overhead.

Results: Based on the tests performed and the performance metrics extracted, statistical Analysis is done. The metrics for DTLS enabled and IPsec enabled end-to-end communication and Plain text communication are calculated respectively and the corresponding graphs are plotted. Limitations of implementing DTLS and IPsec protocols are also described.

Conclusions: From our Research, tradeoffs are established between DTLS and IPsec protocol. For DTLS protocol, significant difference w.r.t to CPU utilization, Elapsed time and network overhead is observed while the memory allocated is an exception. For IPsec, significant difference w.r.t. memory utilization, CPU utilization and network overhead is observed but the time taken to complete the tests is almost the same. If, for the user, memory is a constraint, then DTLS protocol is recommended else if power consumption is a constraint, IPsec protocol is recommended.
Keywords: DTLS, IoT, IPsec, Performance, Security.

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to my supervisor Dr. Dragos Ilie for his guidance and continuous support throughout the thesis work. His knowledge and understanding in the field helped me to break the barrier whenever needed during the course. His passion, dedication and energy never cease to amaze me, and I hope that I have gained a bit of those during the Master Thesis. I am thankful to him for providing the opportunity to work under him. His insights and comments were very valuable and helped me in successful completion of my thesis.

I would like to extend my gratitude to Dr. Adrian Popescu, our thesis examiner for conducting and managing the course successfully. I thank him for the lectures and motivational discussions during the course.

I would like to thank my thesis partner Shiva Tarun for his continuous support and strength during the thesis. He stood by me through the thick and thin of times during the course. I surely wouldn't have completed the thesis without him.

Last but not the least, I would like to thank my family who have always been supporting me irrespective of what I did or am willing to do. They have been my pillars of strength right from my childhood and continue to do so even till now and hope forever.

CONTENTS

LIST OF FIGURES

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.

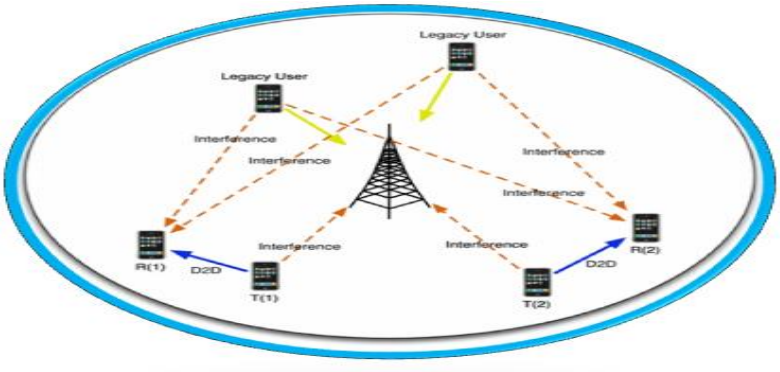


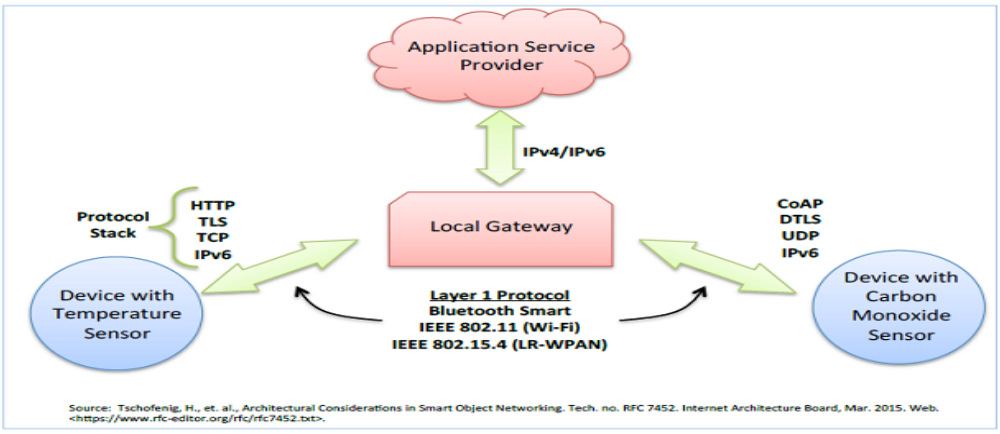
Figure 1 Device to Device Communication

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

Source: Tschofenig, H., et. al., Architectural Considerations in Smart Object Networking. Tech. no. RFC 7452. Internet Architecture Board, Mar. 2015. Web. <https://www.rfc-editor.org/rfc/rfc7452.txt>.

Figure 2 Device to Cloud Communication

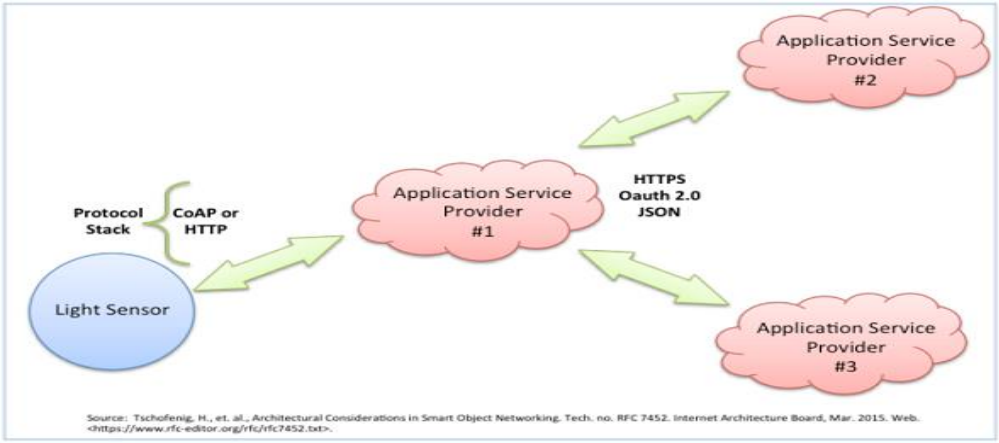
- **Device-to-Gateway Communication**: The application level gateway is being used where gateway acts as an intermediary between the device and the cloud service providing all the functionalities.



Source: Tschofenig, H., et. al., Architectural Considerations in Smart Object Networking. Tech. no. RFC 7452. Internet Architecture Board, Mar. 2015. Web. <https://www.rfc-editor.org/rfc/rfc7452.txt>.

Figure 3 Device to Gateway Communication

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



Source: Tschofenig, H., et. al., Architectural Considerations in Smart Object Networking. Tech. no. RFC 7452. Internet Architecture Board, Mar. 2015. Web. <https://www.rfc-editor.org/rfc/rfc7452.txt>.

Figure 4 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 have open access to all data consumers and controllers while some of them provide multiple features. 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 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. In IoT, we need to provide standards that allow businesses to create devices that are decoupled from services. 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 Framework

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

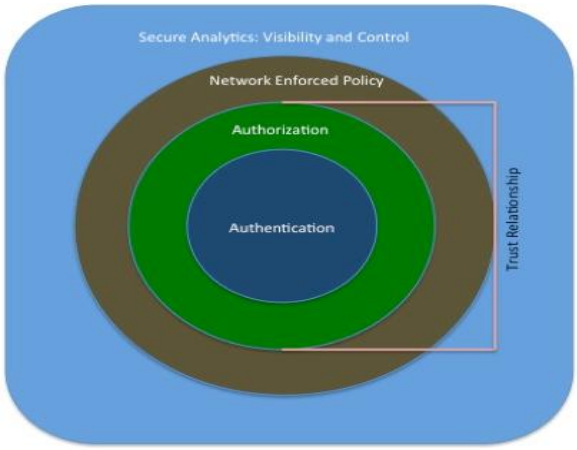


Figure 5 Security Framework

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

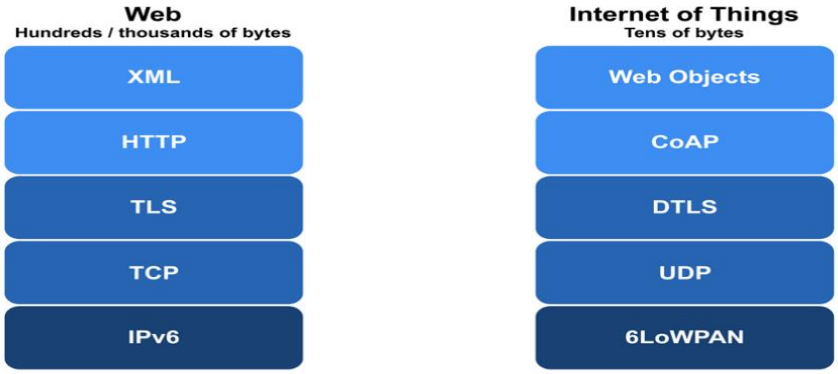


Figure 6 IoT Network Model

Many 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

IP uniquely identifies a device in the current internet architecture, however, IP address serve as a short-term identifier as most devices are portable and change their IP addresses when moved from one network to the other. These IP addresses disrupt long term end-to-end communication. Therefore, mobility and Multihoming are difficult to implement in present internet. The host must prove to its peers that it is the same entity that they have communicated previously. Communication in modern IoT environments is based on the IEEE 802.15.4 standard. IETF has defined a scaled down version called Low Power Wireless Private Area Network (6LoWPAN) which enables interoperability.
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 the case when one communication end is in the 802.15.4 network and the other is placed in the Internet.
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 drawback is that many IoT devices are resource-constrained and thus cannot run full IP stack. The devices have much less support for 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.

1.6.2 Aims and Objectives

The aim of this work is to investigate the performance of end-to-end communication within an IoT environment where the traffic is encrypted with DTLS and IPsec, respectively. The objectives of this project are:

- To build a 6LoWPAN network testbed consisting of the several IoT devices.
- To implement end-to-end DTLS and IPsec, respectively, in an emulated IoT environment.
- To extract performance metrics such as CPU and memory utilization, elapsed time and network overhead for DTLS and IPsec.
- Compare DTLS and IPsec based on protocol features and statistical analysis of the data.

1.6.3 Research Questions

1. What is the performance overhead of implementing end-to-end DTLS compared to plain-text communication in the IoT environment and what are its limitations?
2. What is the performance overhead of implementing end-to-end DTLS compared to plain-text communication in the IoT environment and what are its limitations?
3. Is there a significant difference in performance overhead between DTLS and IPsec?
4. Based on performance overhead and limitations which protocol (IPsec or DTLS) is best suited to provide confidentiality?

1.7 Outline of the document

- Chapter 1 i.e., this chapter consists of the basic overview of the thesis. 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 6LoWPAN, CoAP and IoT as a part of the Literature study.
- Chapter 3 deals with the Background section providing necessary information about 6LoWPAN and CoAP.
- Chapter 4 briefs about the Security protocols considered in the Master Thesis.
- Chapter 5 deals with the methodology adopted in our thesis to build the test bed to perform the tests necessary for the thesis.
- Chapter 6 shows all the results recorded during the experiments performed and statistical analysis of the obtained results.
- Chapter 7 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 8 provides the references which were helpful during the thesis.
- Chapter 9 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	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	Vignesh Kuna 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	Vignesh Kuna
	6.2 Tests Performed	Vignesh Kuna
7.Conclusions and Future Work	7.1 Conclusions	Vignesh Kuna
	7.2 Future Work	Vignesh Kuna

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. Contiki OS has support only for pre-shared key DTLS. 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.

Security for IoT: Effective DTLS with Public Certificates
Employing public certificates makes authentication more robust. In paper [2], it is achieved by issuing digital certificates to client and the server making it a better performer than DTLS with pre-shared key. DTLS has 4 sub protocols Handshake, change cipher spec, alert and application data protocols. In DTLS, some errors are sent as warnings include bad record, overflow or decryption failed which prevents connection termination. As the processing speeds of IoT are increasing, there is a need for proper security standard and the use of a certification authority (CA) as a trusted entity. This paper helped us to use the public key cryptography using digital certificates to authenticate and encrypt the data. The certificates are signed digitally by private key of the CA. Decryption takes place by the public keys of CA on either side. X.509 digital certificates were used for securing DTLS. The RTTs and even the memory consumption was almost the same as DTLS with pre-shared key. They concluded saying a generic configuration of modules is missing and furthermore research needs to be done.

IPv6 Security tools: 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 [3].
They concluded with some of the tools best suitable such as n-map, 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 [4]. They have considered transmission header, connection establishment and processing overhead. They have concluded saying TLS has least number of round-trip times 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 scenarios (i.e., in end-to-end and proxy) in an emulated IoT environment.

Security Analysis of IoT protocols: Focus on CoAP

The authors have done a research on security of CoAP over DTLS providing some solutions in [5]. 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. Different implementations of CoAP such as Californium, Erbium, jCoAP, Libcoap exist which were told. 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 a feasibility study of implementing the security protocols on the existing CoAP implementations.

Securing IP based IoT: HIP and DTLS

In [6], in-order to ensure the security of the devices in the network the interaction of the devices must be regulated during entire life cycle. 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. The IoT network is connected to the public Internet through number of 6LoWPAN border routers (6LBR). There will be a domain manager which manages the IoT networks. Secure network access is how to securely associate a new device in an IOT network. The device that have been authenticated and authorized through a secure network process should be allowed to communicate within the network. The network is protected at link layer by a means of symmetric key which is unknown to the joining device. The joining device authenticates itself to domain manger using link local address by a means of PSK. After successful authentication, the domain manager issues access parameters that would allow the device to access the secure IOT network. Secure communication is achieved by protecting the exchange of CoAP message by means of DTLS record layer. This research has detailed description of implementing the DTLS based security solution in IoT environment.

Cross-platform protocol development based on OMNeT++

In [7], 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 the authors have proposed a new cross-platform OS named CometOS under GPLv3. TinyOS is an open source, BSD-licensed operating system designed for low-power wireless devices, such as those used in sensor networks, ubiquitous computing, personal area networks, smart

buildings, and smart meters. Contiki is an operating system for networked, memory-constrained systems with a focus on low-power wireless Internet of Things devices. Extant uses for Contiki include systems for street lighting, sound monitoring for smart cities, radiation monitoring, and alarms. 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 like 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++. Mesh 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.

6LoWPAN model for OMNeT++

In [8], 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 then 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. This work has provided the basic guideline and instructions in linking the OMNeT++ and Cooja simulator in the Contiki operating system which was the adopted method earlier.

Security Protocols and privacy issues into 6LoWPAN

The paper [9] 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. This research has stated the various security aspects provided by the IPSec and implementation challenges in the 6LoWPAN networks.

6LoWPAN multi-layered security protocol based on IEEE 802.15.4 security features

Glissa and Medeb in [10] 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 [11], 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 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 tiny-OS 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
Vishwas and Kewal in [12] 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 [13] 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

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. In [14], 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
The authors provide an end-to-end secure communication between IP enabled IoT devices and the internet in [15]. 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 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 [16], 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 Client Hello 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 Server Hello 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 feasibility of compression of DTLS protocol less performance overhead.

Certificate-free Collaborative Key Agreement based on IKEv2 for IoT

In [17], 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 agreement, here this process is modified such that the unconstrained nodes complete the process. Certificate-free Collaborative Key Agreement and Authentication protocol saves communication cost. Implementation was done on 25 constrained nodes and using 10 UC nodes as unconstrained nodes. The number of communication messages for the Security association agreement between CKES and C2A2 was compared where C2A2 was better and scythe, automatic tool for testing security protocols was used which yielded positive results

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

In [18], the research is carried out on 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

The authors in [19] studied the various threats and attacks at the various layers in 6LoWPAN and counter measures to address the needs and recommended some changes to achieve reliability. 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. 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 fact that IPsec also requires another header (AH or ESP) in each packet, it makes its use difficult in 6lowpan environments because IPsec requires two peers communicate to share a secret key which is usually implemented dynamically with IKE (Internet Key Exchange) protocol and the way the protocol is designed to achieve security. 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.

A survey on techniques for securing the 6LoWPAN

Paper [20] discusses about the possible threats in the 6lowpan layer and the mechanisms which is used till now to prevent from these 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 are 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.

End-to-End transport security in IP based IoT

This paper 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 anywhere 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. In [21], architecture can be seen as a typical deployment for buildings to facilitate building automation and control. 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 mainly focuses on the implementation of DTLS and IPsec protocols in securing the COAP. [22] 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.

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 where the number indicates the bits for the length of blocks) 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.

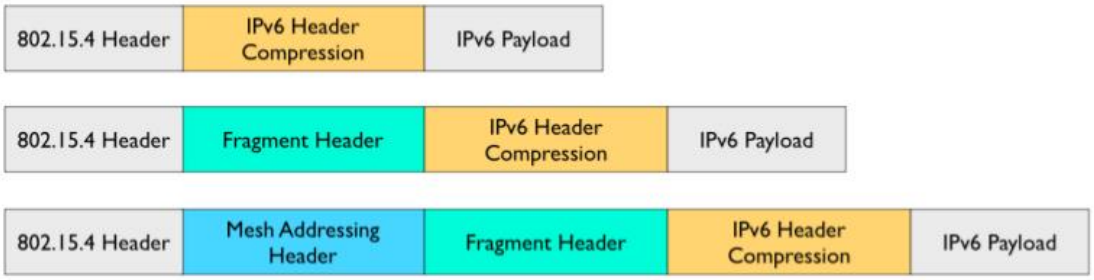


Figure 7 6Lowpan addressing header

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 it's 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.

1. What is the performance overhead of implementing end-to-end DTLS compared to plain-text communication in the IoT environment and what are its limitations?
2. What is the performance overhead of implementing end-to-end DTLS compared to plain-text communication in the IoT environment and what are its limitations?
3. Is there a significant difference in performance overhead between DTLS and IPsec?
4. Based on performance overhead and limitations which protocol (IPsec or DTLS) is best suited to provide confidentiality?

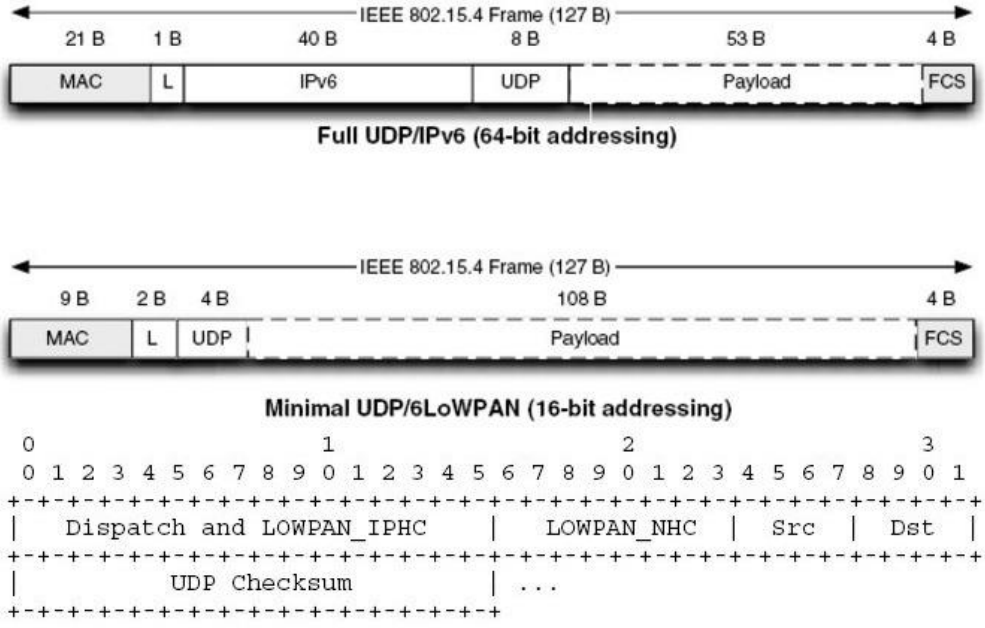


Figure 8 6Lowpan Addressing format

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 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 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 as 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 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

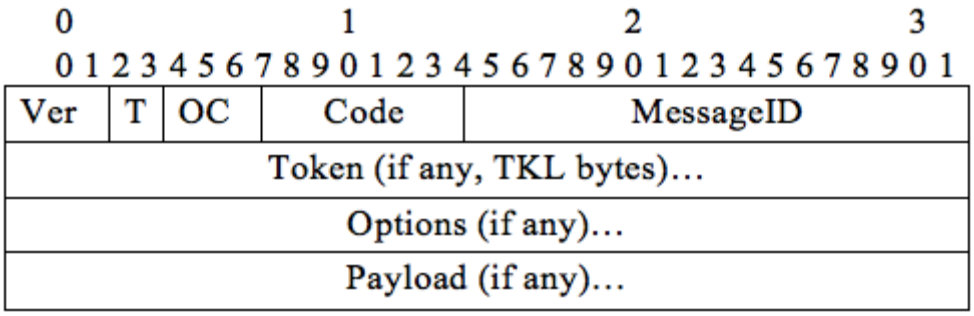


Figure 9 CoAP Message Format

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 a 3-bit class i.e. most significant bits and a 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 Security Protocol

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 (AH)

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.

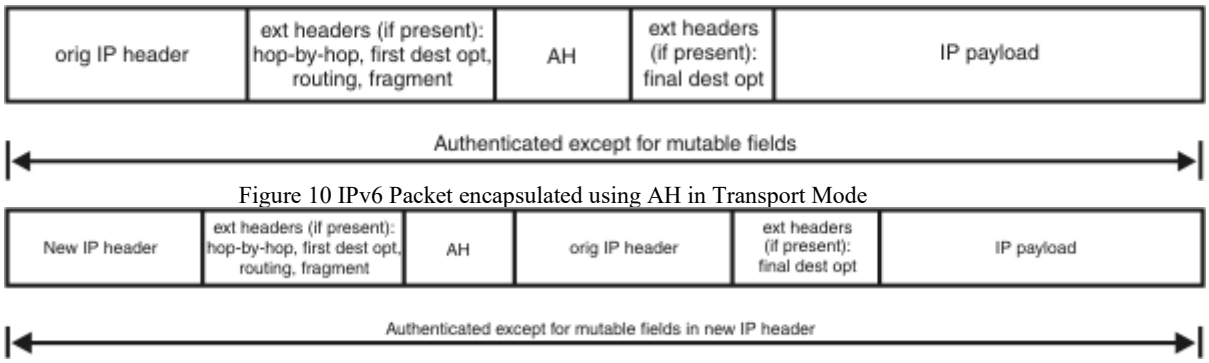


Figure 10 IPv6 Packet encapsulated using AH in Transport Mode

Figure 11 IPv6 packet encapsulated using AH in tunnel mode

4.1.4 Encapsulating Service Payload (ESP)

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.

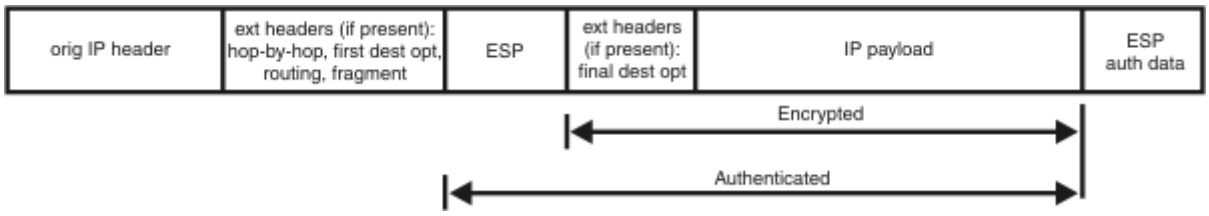


Figure 12 IPv6 encapsulated ESP header in Transport mode

Figure 13 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 infront 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.

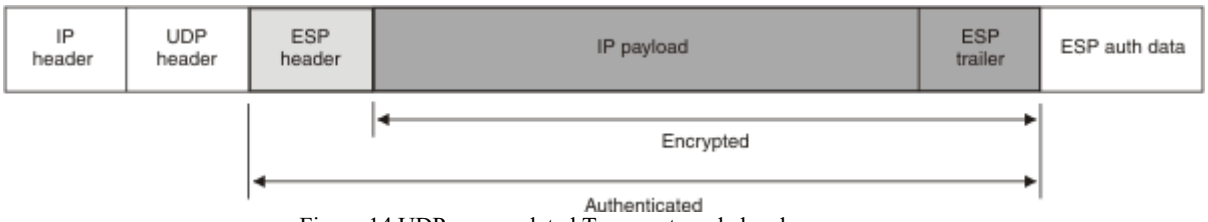


Figure 14 UDP encapsulated Transport mode header

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.

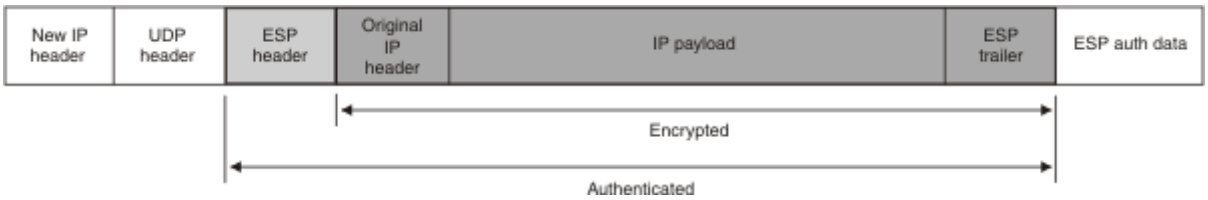


Figure 15 UDP encapsulated tunnel mode header

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 Re-ordering

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.

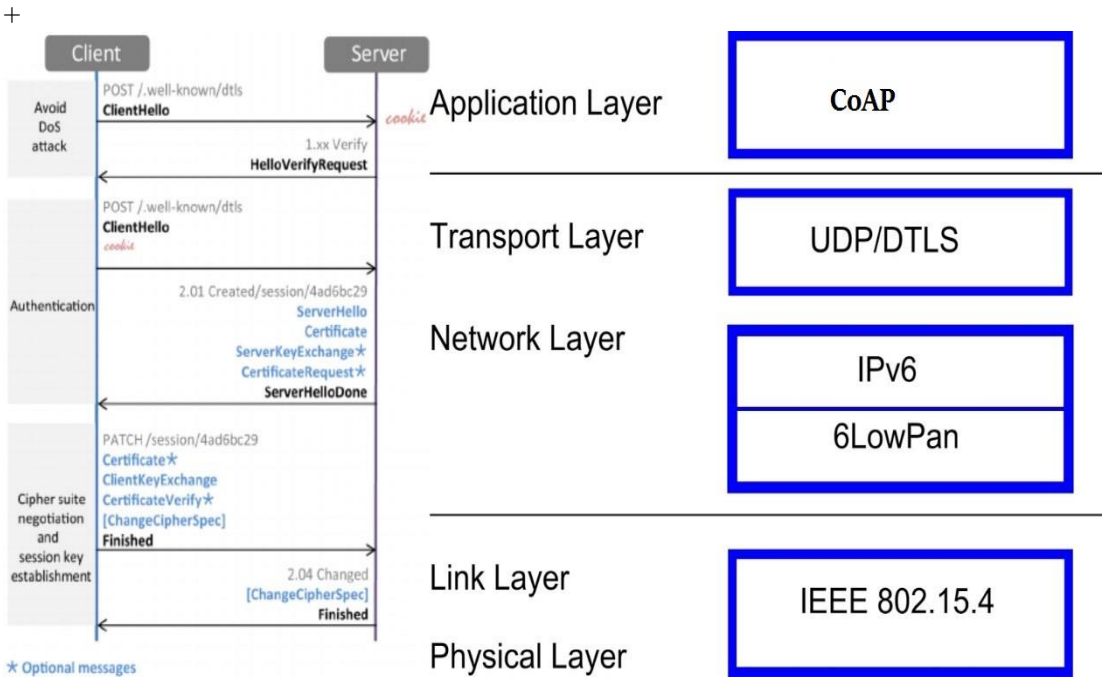


Figure 16 DTLS Architecture and Handshake

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 Denial of Service

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 Handshake 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 Handshake 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.

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. 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.

4.3.3 Advantages of TLS

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

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 throughput, response time, CPU and memory utilization.
- Statistical analysis of the results obtained.

5.1 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. 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.

5.2 Linux-wpan Project

This project is intended to provide support for IEEE 802.15.4 and 6LoWPAN support in mainline support. 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.

5.3 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 WPAN device contain interface specific settings such as MAC address and PAN id
- **WPAN Ping**: For the network testing of 6LoWPAN nodes.

5.3.1 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.

5.4 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 house can be grouped into one network and all the light sensors into 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 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.

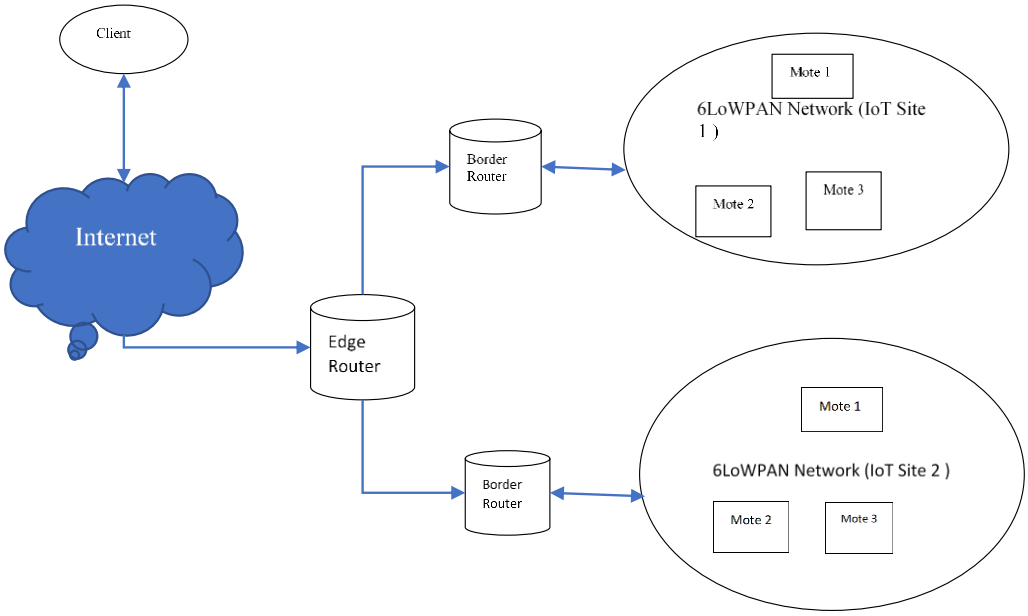


Figure 17 IoT Logical Network Architecture

5.5 Implementation of Protocols

5.5.1 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.

These wpan devices are fake devices which are emulated in the Linux kernel. 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 interfaces, an IoT mote to communicate with other IoT devices within the site and virtual Ethernet interface to communicate with the outside world. For this, we use veth-vpeer pair and enable forwarding 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 in the testbed.

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. Throughput, response time, CPU 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.

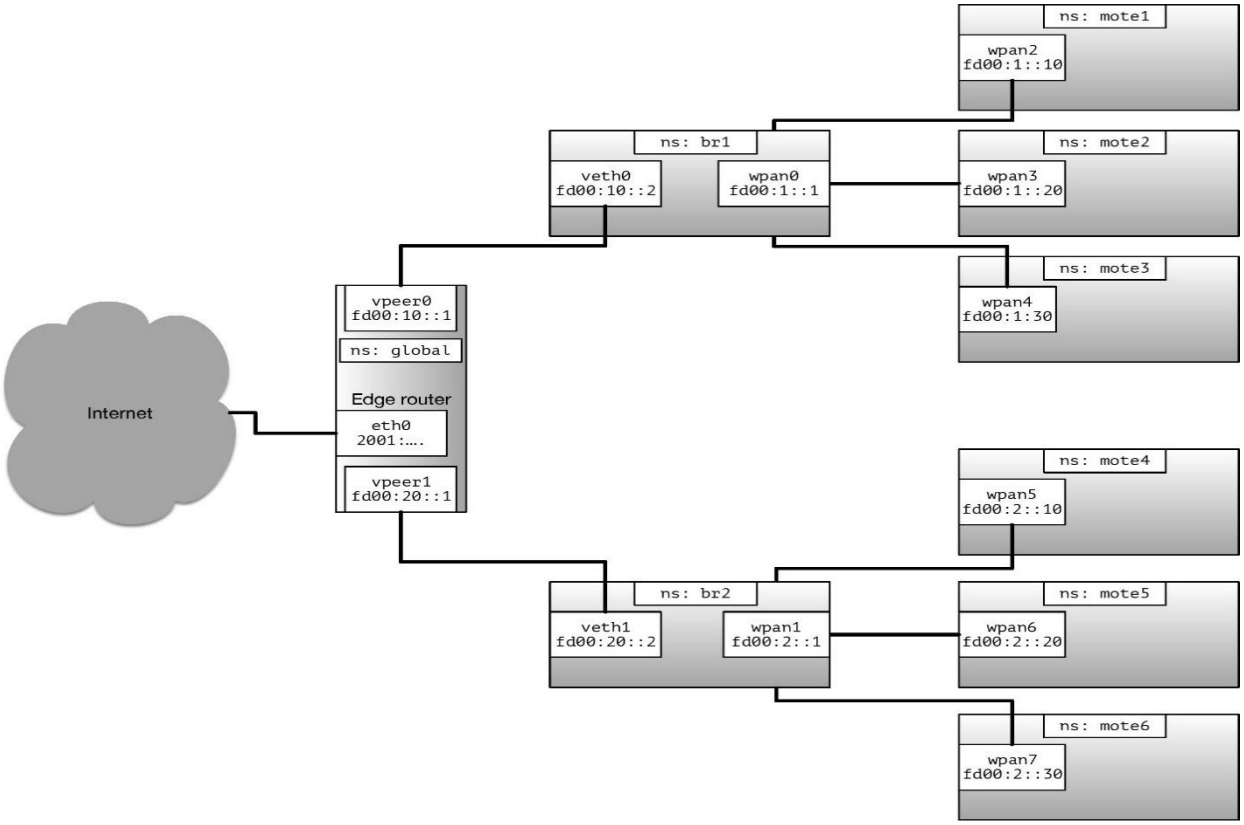


Figure 18 Emulated Testbed

5.5.2 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 individual client connect to network in a remote location, through a secure and encrypted tunnel that allows them to access all resources in that network as if they are inherently connected to the server in that network.

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 emulating the network inside the namespaces, hence strong swan is configured to run multiple instances inside the name spaces. Individual certificates are generated separately for the border route and the client. The client in our scenario runs in the global namespace hence the remote access VPN is setup between the global namespace and border router. The certificate generation for the border router and Client are discussed in the Appendix.

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.

5.5.3 DTLS

Test COAP servers with support for the DTLS are deployed on the 6LoWPAN test nodes which are present in the separate namespaces. COAP clients are deployed in the global namespace. The test COAP servers, test COAP clients 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. The same x.509 certificates which are generated above are used in implementing DTLS protocol.

5.6 Tests Performed

To evaluate the performance of these protocols the following tests are made. The tests below are the default tests that accompany TinyDTLS.

1. **Send a confirmable get request and expect a piggy-backed response**
   Some messages require an acknowledgement to confirm the delivery of the packet which can be termed as confirmable. Each confirmable request elicits exactly one return message. If the response is carried directly in the acknowledgement message that acknowledges the request, then it can be called a "Piggy-backed" response. In the thesis, we send a confirmable get request and expect a piggy-backed response.

2. **Send a confirmable get request and expect a separate response**
   In some cases, the server may need longer time to obtain the representation of the resource requested and in these cases, it is not possible to send piggy-backed responses. Responses to the request are sent separately. Here, acknowledgement to a confirmable request and confirmable response must be

an empty message. In the thesis, we send a confirmable get request and expect a “separate” response.

3. **Send a non-confirmable request**
   Some messages don’t require an acknowledgement which is true for cases which transmit the messages repeatedly for application requirements. In the thesis, we send a non-confirmable get request.

4. **Send 2 block-wise get requests**
   Transfers larger than what can be accommodated in constrained-network link-layer packets can be performed in smaller blocks. Transfer of each block is acknowledged, enabling individual retransmission if required. Size of the block which is variable, whether more blocks are following the preceding block and the relative number of the block within a sequence of blocks with given size is the information in a Block option. In the thesis, we send two Block-wise get requests.

5. **Send 2 block-wise put requests**
   In a Block-wise transfer, Block1 refers to the transfer of the resource representation that pertains to the request, and Block2 refers to the transfer of the resource representation for the response. Due to the desire to avoid IP fragmentation and adaptation-layer fragmentation and the value of maximum datagram size, block-wise transfers can be used. In Block-wise transfer, the advantage being the resulting exchanges are easy to understand for debugging. In this test, we send two block-wise Put requests.

6. **Send 2 non-confirmable get requests**
   Some messages don’t require an acknowledgement which is true for cases which transmit the messages repeatedly for application requirements. Here, we send two non-confirmable get requests.

7. **Send a request with a bad option response**
   We get a response code irrespective of the request/ response result whether it is a success or a failure. These option numbers are predefined in the COAP option number registry. Odd numbers indicate a critical option while an even number indicates an elective option. Here, we send a request with the intention of getting a bad option response.

5.7 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++.

5.8 Contiki OS

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.

5.9 Cooja Simulator

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.

5.10 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.

5.11 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.

6 RESULTS AND ANALYSIS

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. As mentioned above, my work aims to analyze the performance of DTLS and IPsec protocols when the communication is secured end-to-end.

Case-1:
The communication from the client to the server is via the border router and vice versa. The entire data transferred in the network is encrypted with DTLS protocol.

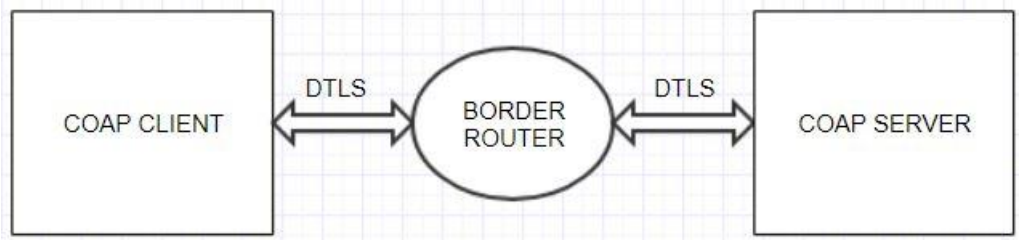


Figure 19 DTLS END-TO-END CASE

Case-2:
Here, IPsec VPNs are established between the clients and the border routers using road warrior configuration as stated earlier. The entire data traffic through the network is encrypted with IPsec protocol.

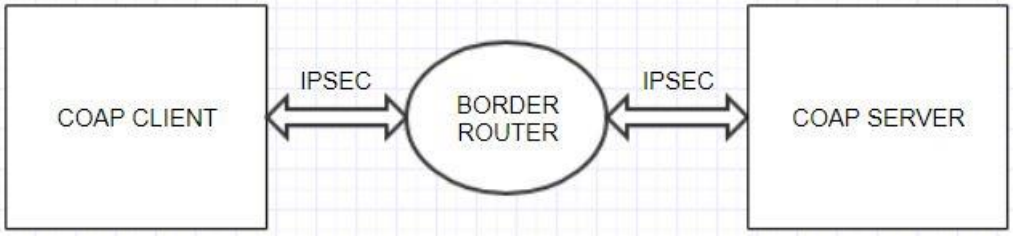


Figure 20 IPSEC END-TO-END CASE

The following is the motivation for considering these performance metrics in the research.

6.1 Performance Metrics

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, impact of adding security on the memory is also important 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.

6.2 Test-1: To Send a Confirmable Get Request and expect a piggy-backed Response

The graphs for CPU utilization, memory utilization, elapsed time and network overhead are shown below. 100 get requests were sent to the server. 30 iterations were performed for DTLS, IPsec and plain text communication respectively and the average values are calculated. The standard deviation and the value for 95% CI are also calculated. The corresponding graphs are plotted.

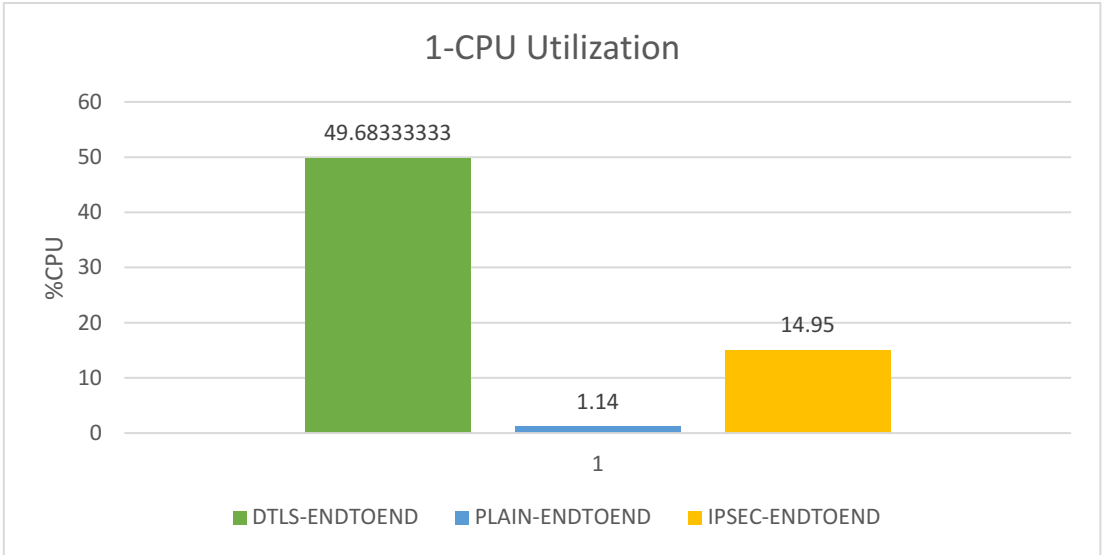


Figure 21 Test-1: CPU Utilization

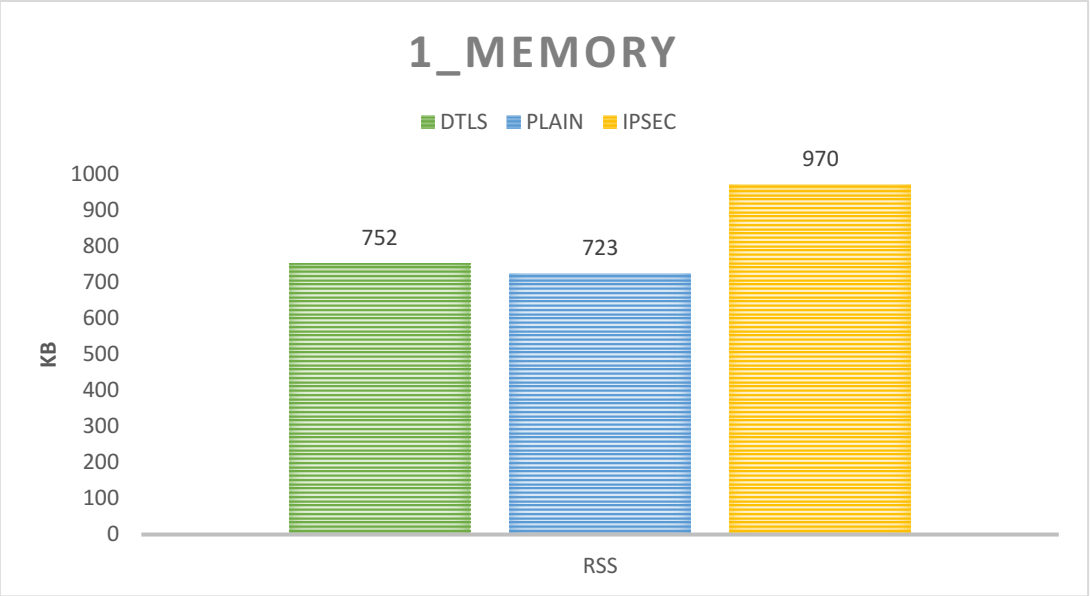


Figure 22 Test-1: Memory Utilization

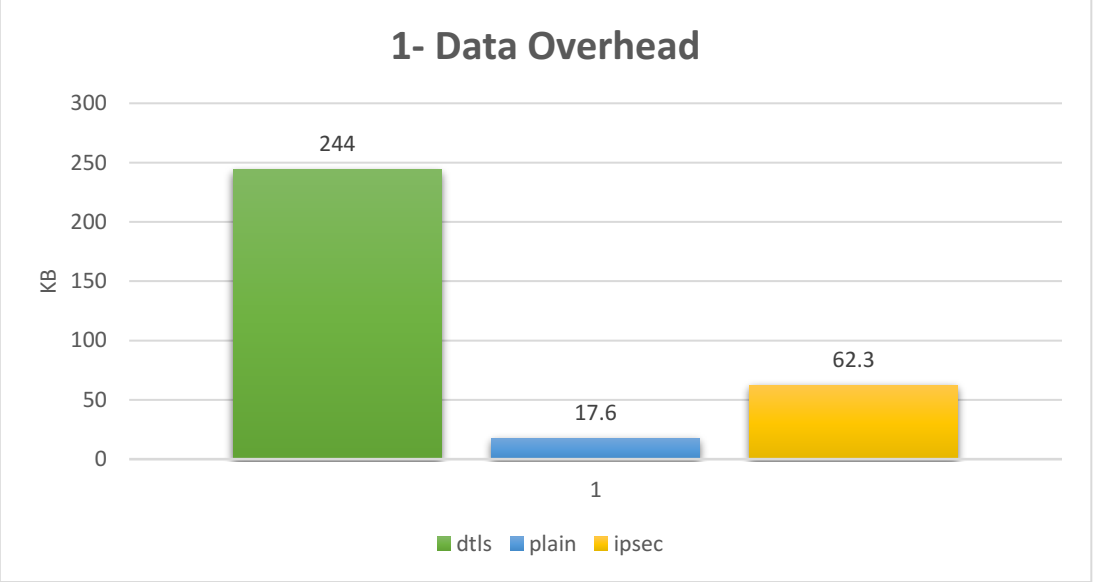


Figure 23 Test-1: Network Overhead

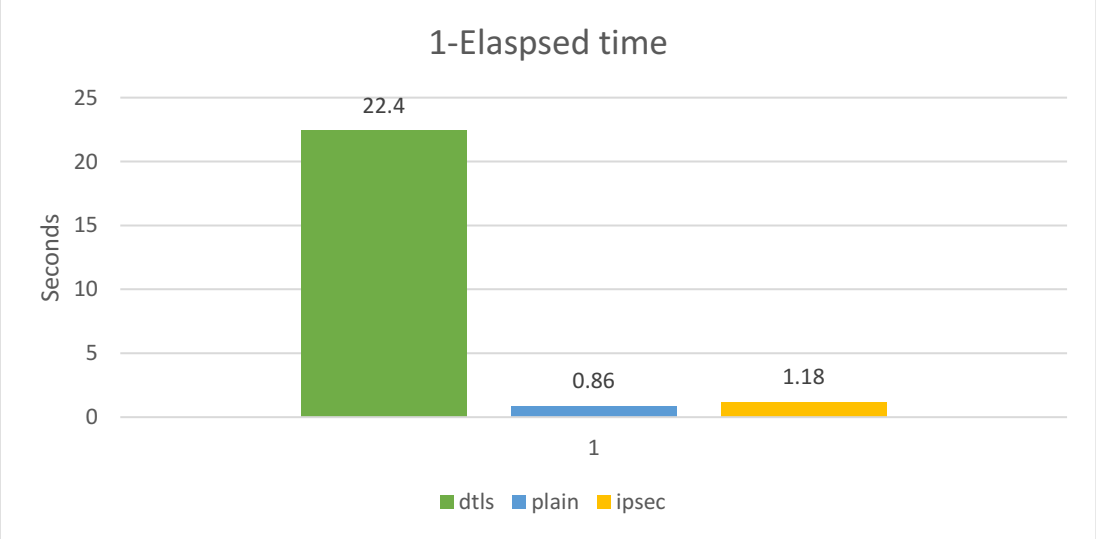


Figure 24 Test-1: Elapsed Time

CPU Utilization: From figure 21, we observe that DTLS protocol generates higher CPU utilization about 50%, IPsec protocol uses 15% and plain-text communication uses about 1%.

Memory Utilization: From figure 22, we observe IPsec protocol uses more memory compared to DTLS and Plain text communication. DTLS has slightly higher memory utilization than plain text communication.

Network Overhead: From figure 23, one can say that DTLS has the highest network overhead followed by IPsec protocol and plain text communication.

Elapsed Time: From figure 24, it is clear, longest time is taken for DTLS to complete the tests. IPsec and Plain-text communication take almost the same time which is much lesser compared to DTLS.

6.3 Test-2: To Send a Confirmable Get Request and expect a Separate Response

The graphs for CPU utilization, memory utilization, elapsed time and network overhead are shown below. 100 get requests were sent to the server. 30 iterations were performed for DTLS, IPsec and plain text communication respectively and the average values are calculated. The standard deviation and the value for 95% CI are also calculated. The corresponding graphs are plotted.

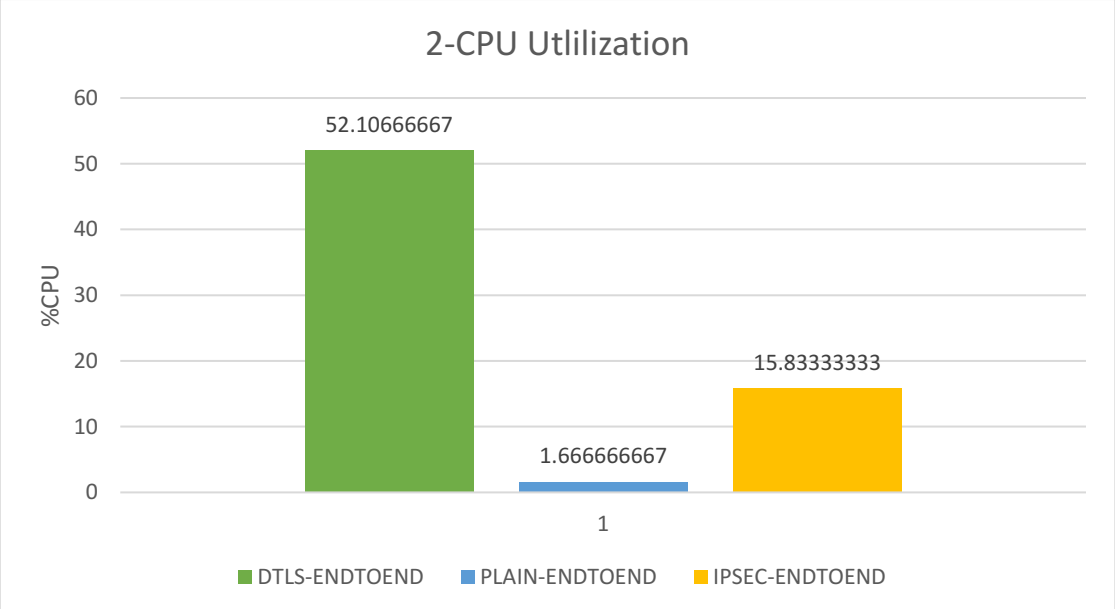


Figure 25 Test-2: CPU Utilization

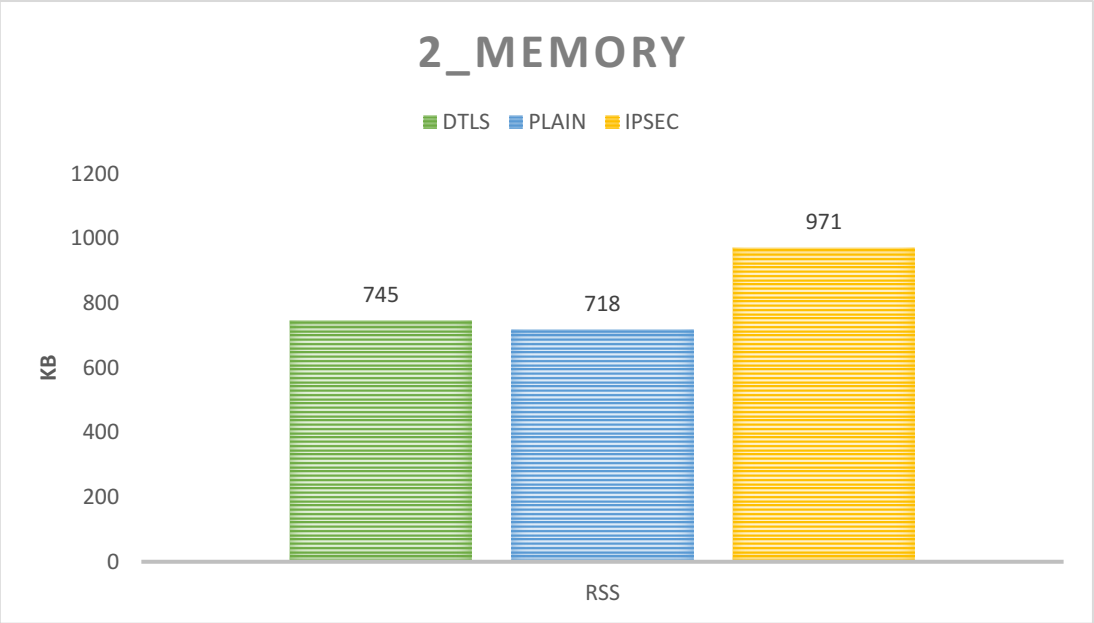


Figure 26 Test-2: Memory Utilization

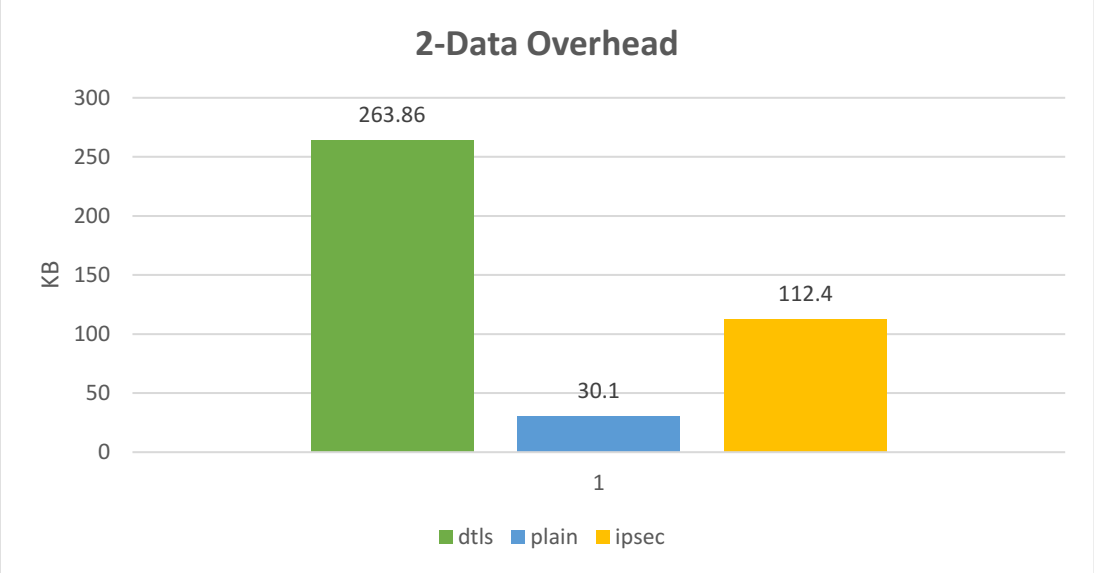


Figure 27 Test-1: Network Overhead

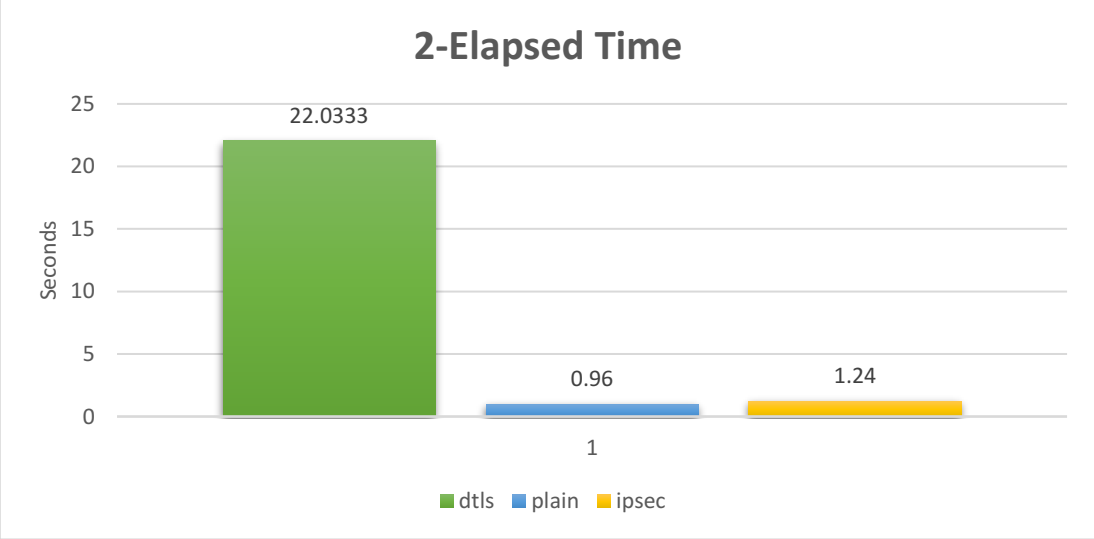


Figure 28 Test-2: Elapsed Time

CPU Utilization: Figure 25, depicts the CPU utilization for a get request with separate response. DTLS protocol has highest CPU, IPsec protocol being the next highest and plain-text communication the last.

Memory Utilization: Figure 26, depicts the memory utilization for a get request with separate response. Highest memory is being utilized by IPsec protocol. DTLS has higher utilization than plain text communication.

Network Overhead: Figure 27, depicts the network overhead for a get request with separate response. Overhead is the most for DTLS protocol almost 271k bytes, succeeded by IPsec almost 115k and the last being plain text communication about 30k.

Elapsed Time: Figure 28, depicts the elapsed time for a get request with separate response. Longest time is taken by DTLS protocol whereas IPsec and Plain text communication take much less time.

6.4 Test-3: To Send a Non-Confirmable Get Request

The graphs for CPU utilization, memory utilization, elapsed time and network overhead are shown below. 100 get requests were sent to the server. 30 iterations were performed for DTLS, IPsec and plain text communication respectively and the average values are calculated. The standard deviation and the value for 95% CI are also calculated. The corresponding graphs are plotted.

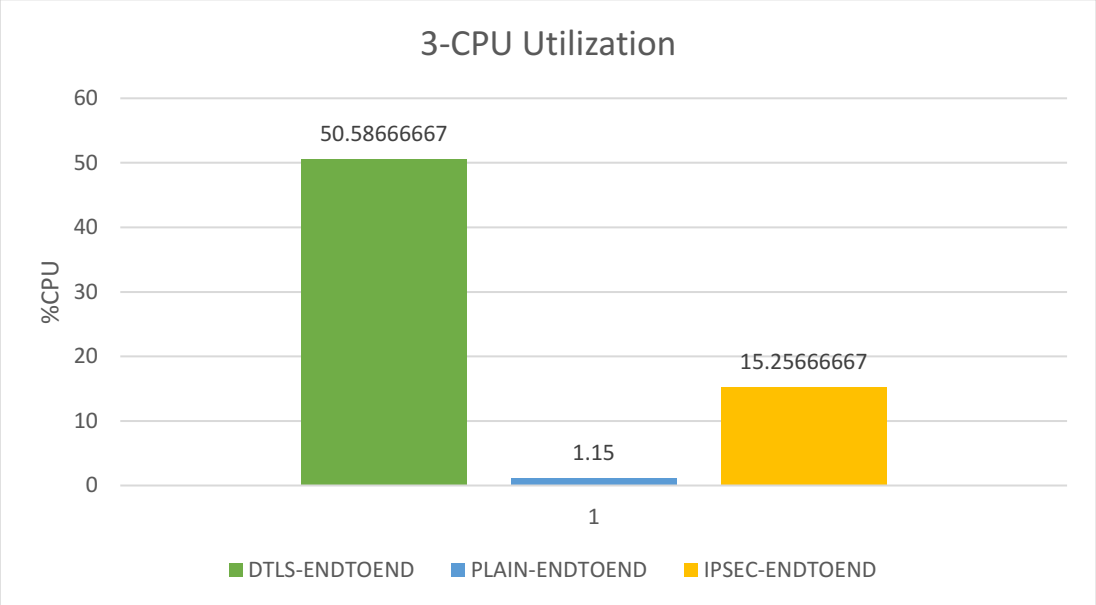


Figure 29 Test-3: CPU Utilization

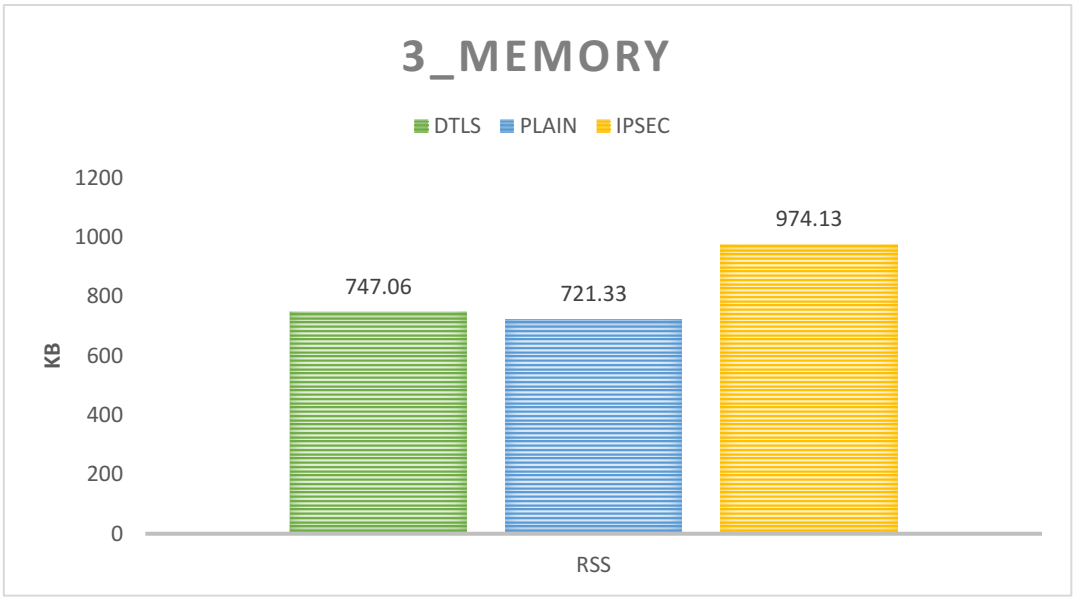


Figure 30 Test-3: Memory Utilization

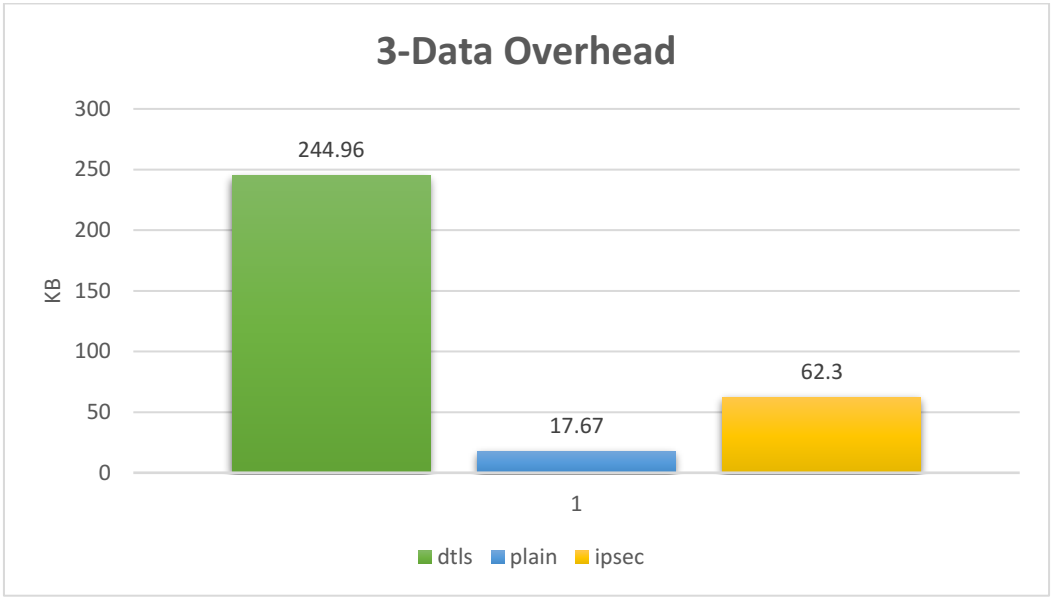


Figure 31 Test-3: Network Overhead

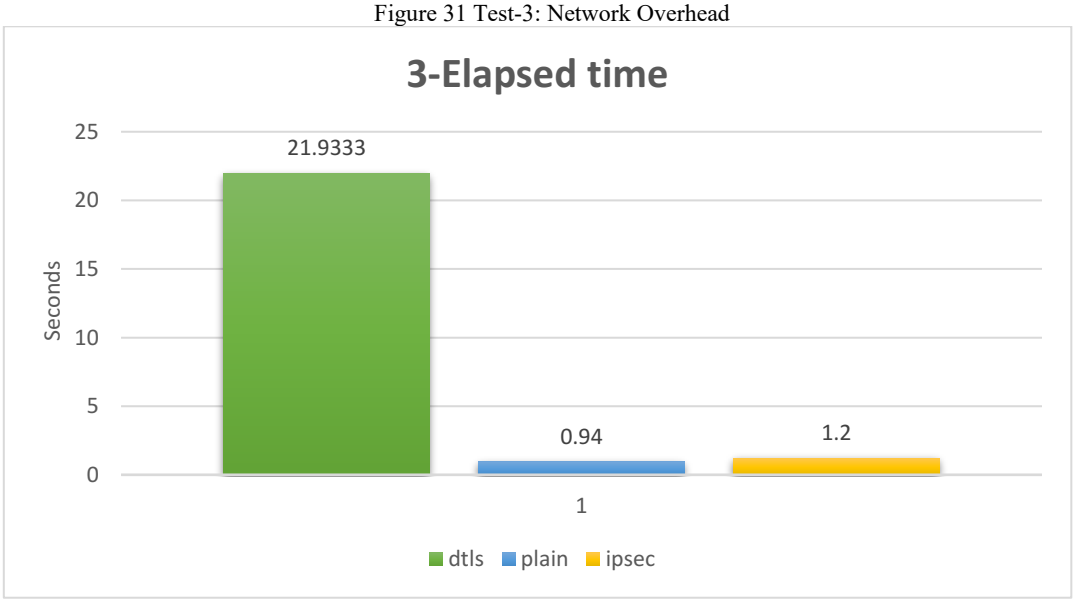


Figure 32 Test-3: Elapsed Time

CPU Utilization: CPU utilization for a non-confirmable get request is showed in figure 29. 15% of CPU is used for IPsec protocol whereas plain text communication uses only 1.15% and 50.5% by DTLS protocol.

Memory Utilization: Memory utilization for a non-confirmable get request is given by figure 30. Least memory is allocated for plain text communication. DTLS protocol is the second least whereas highest memory is allocated for IPsec protocol.

Network Overhead: The network overhead for a non-confirmable get request is presented in figure 31. Descending order of the overhead is DTLS protocol, IPsec protocol and plain text communication.

Elapsed Time: The elapsed time for a non-confirmable get request is displayed in figure 32. Shortest time is taken for plain text communication succeeded by IPsec protocol and DTLS.

6.5 Test-4: To Send 2 Block-wise Get Requests

The graphs for CPU utilization, memory utilization, elapsed time and network overhead are shown below. 200 block-wise get requests were sent to the server. 30 iterations were performed for DTLS, IPsec and plain text communication respectively and the average values are calculated. The standard deviation and the value of 95% CI are also calculated. The corresponding graphs are plotted.

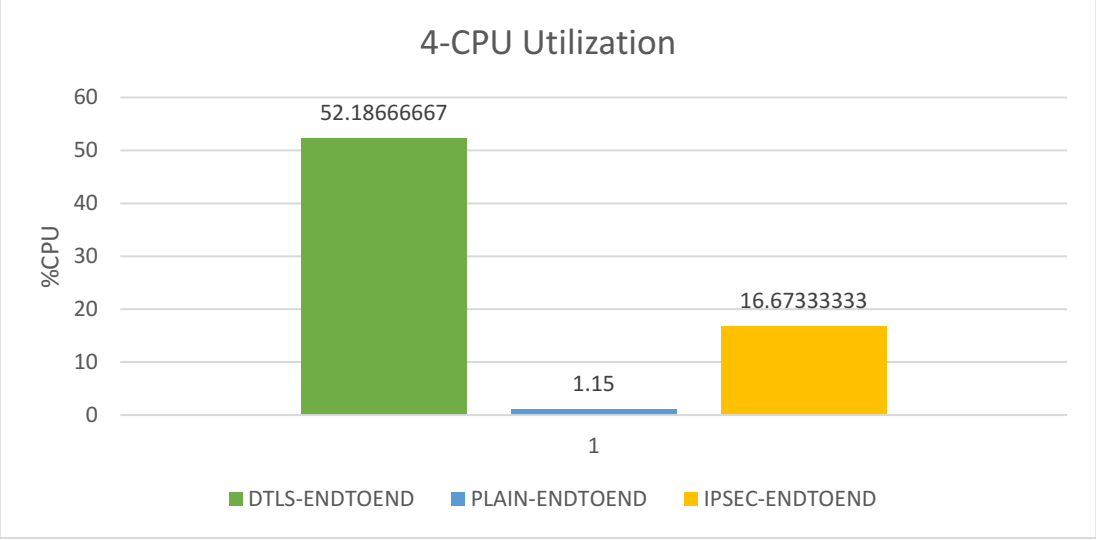


Figure 33 Test-4: CPU Utilization

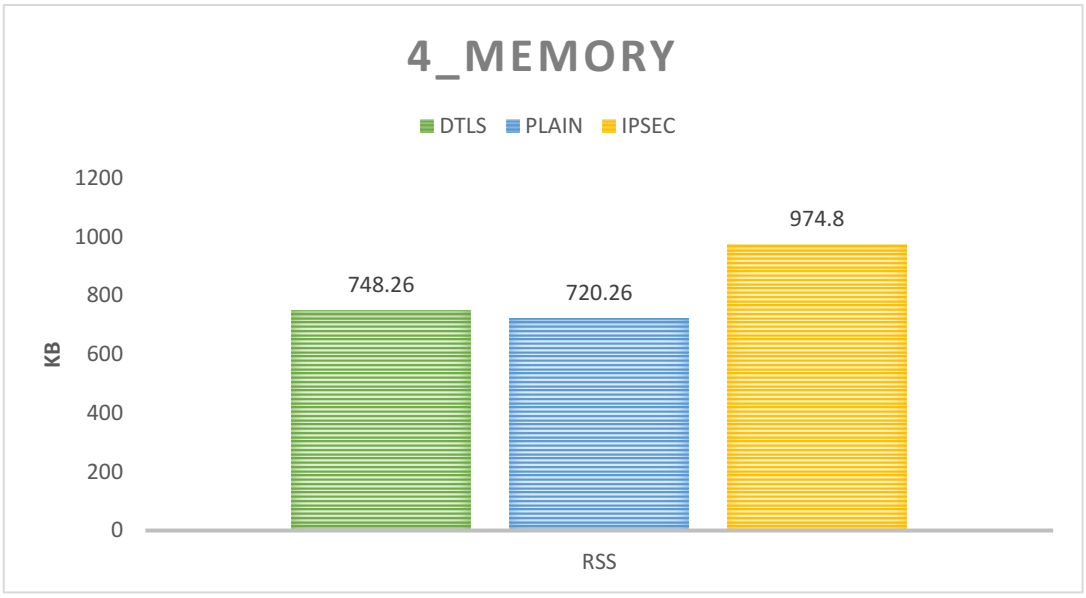


Figure 34 Test-4: Memory Ulitization

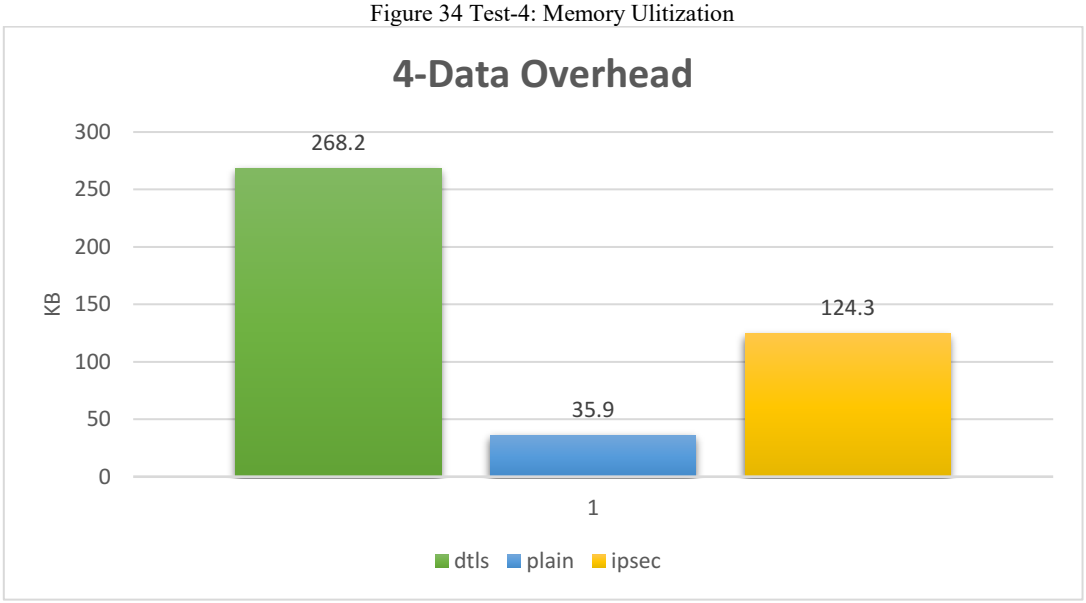


Figure 35 Test-4: Network Overhead

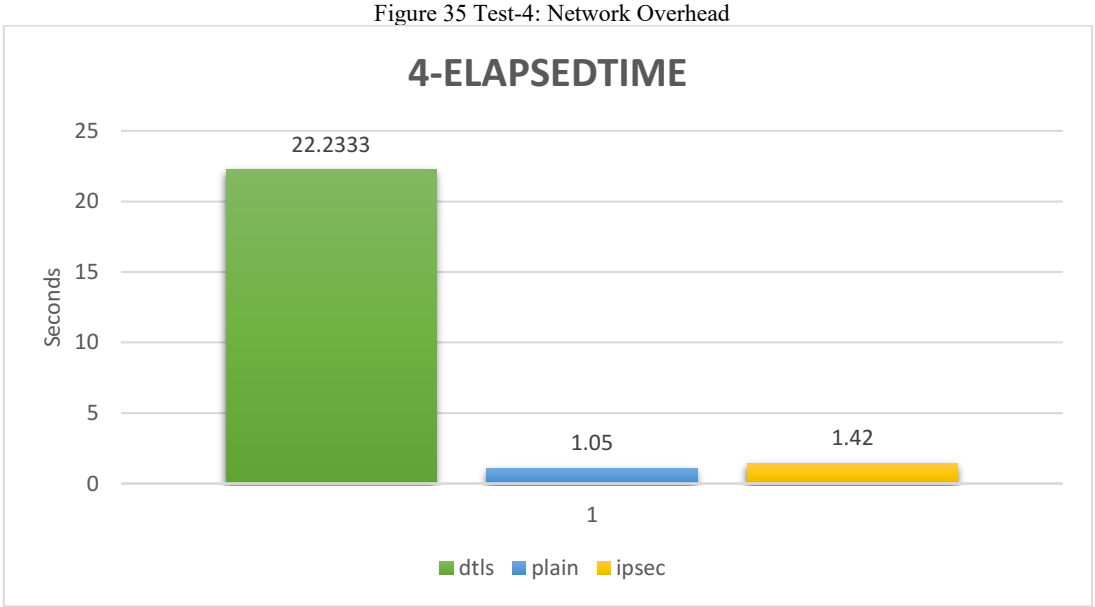


Figure 36 Test-4: Elapsed Time

CPU Utilization: DTLS uses three times the CPU that is consumed for IPsec and 40 times the CPU that is consumed for plain text communication which can be observed in figure 33.

Memory Utilization: DTLS only uses 2/3rds of the memory that is used by IPsec and Plain text communication uses slightly less memory compared to DTLS protocol, demonstrated in figure 34.

Network Overhead: DTLS uses almost 8 times the data consumed for plain text communication otherwise and 2 times the data in the case of IPsec which is clearly visible from figure 35.

Elapsed Time: DTLS takes 22 times the time taken for plain text communication otherwise and IPsec takes 1.5 times the time taken for plain text communication depicted in figure 36.

6.6 Test-5: To Send 2 Block-wise Put Requests

The graphs for CPU utilization, memory utilization, elapsed time and network overhead are shown below. 200 block-wise Put requests were sent to the server. 30 iterations were performed for DTLS, IPsec and plain text communication respectively and the average values are calculated. The standard deviation and the value of 95% CI are also calculated. The corresponding graphs are plotted.

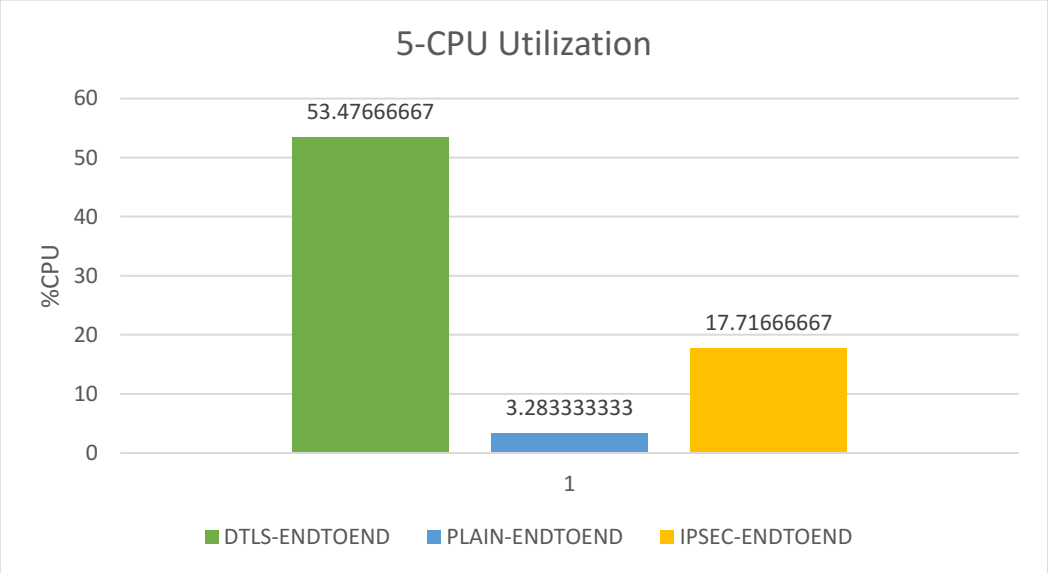


Figure 37 Test-5: CPU Utilization

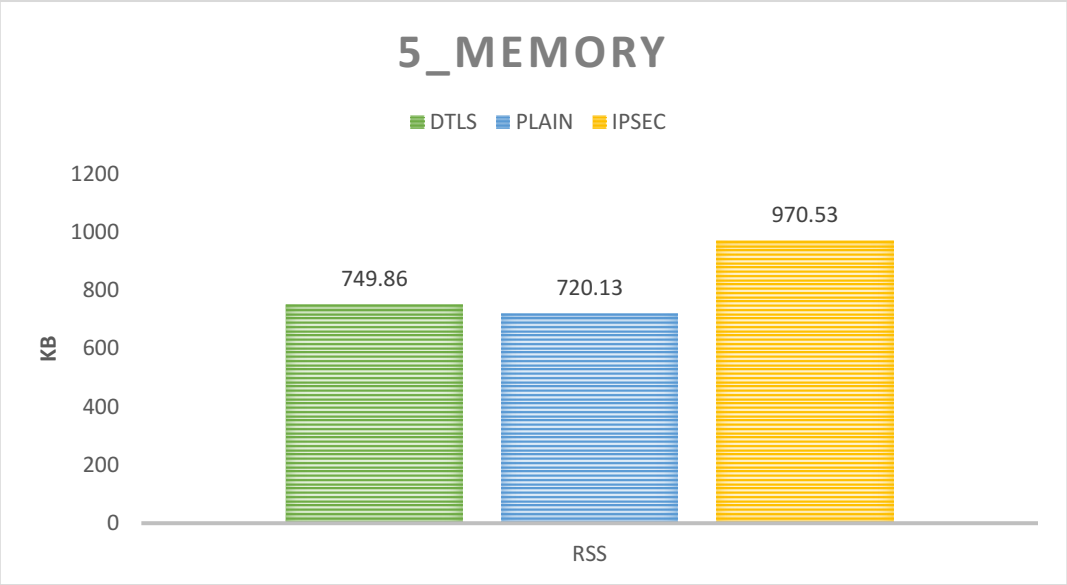


Figure 38 Test-5: Memory Utilization

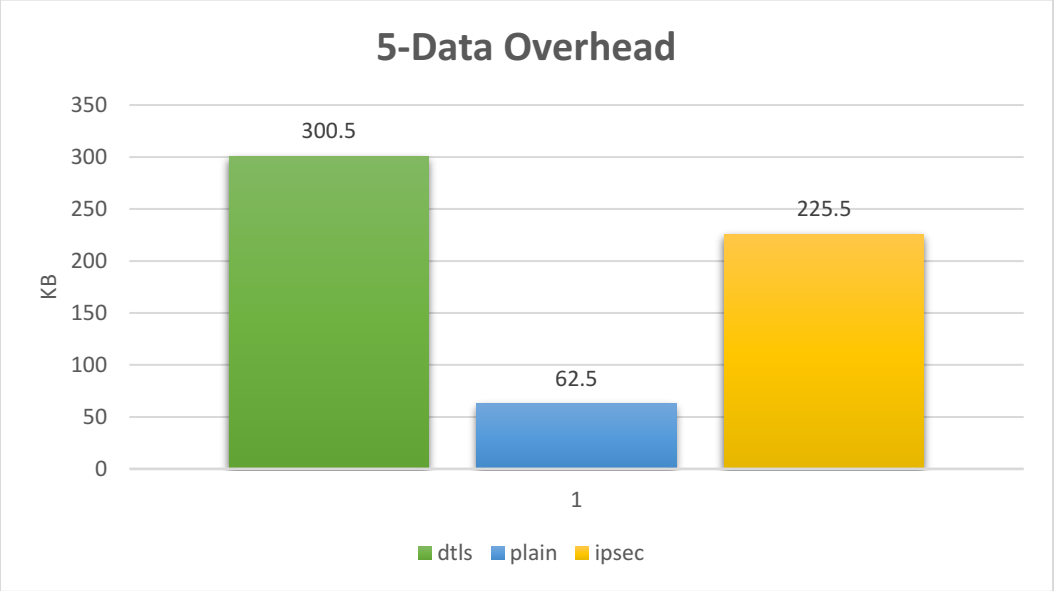


Figure 39 Test-5: Network Overhead

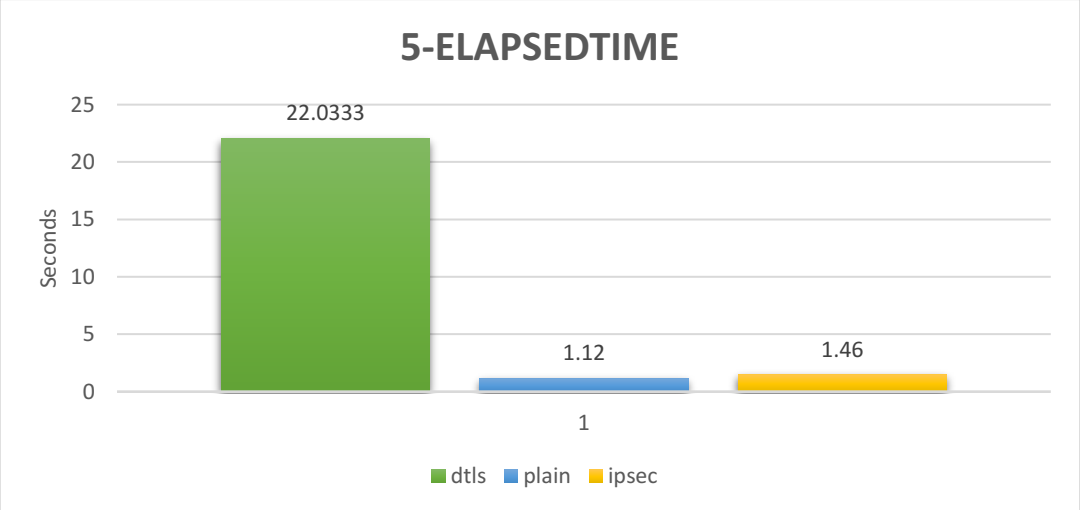


Figure 40 Test-5: Elapsed Time

CPU Utilization: Ascending order of the CPU utilization is plain text communication, IPsec protocol and DTLS protocol respectively observed from figure 37.

Memory Utilization: Ascending order of the memory being utilized for 2 block-wise put requests are plain text communication, DTLS protocol and IPsec Protocol displayed in figure 38.

Network Overhead: Ascending order of the network data being carried extra after securing the data for 2 block-wise put requests are plain text communication, IPsec protocol and DTLS protocol visible in figure 39.

Elapsed Time: Ascending order of the time taken for 2 block-wise put requests are plain text communication with a slight increase for IPsec protocol and drastic increase for DTLS protocol which can be seen in figure 40.

6.7 Test-6: To Send 2 Non- Confirmable Get Requests

The graphs for CPU utilization, memory utilization, elapsed time and network overhead are shown below. 200 non-confirmable get requests were sent to the server. 30 iterations were performed for DTLS, IPsec and plain text communication respectively and the average values are calculated. The standard deviation and the value of 95% CI are also calculated. The corresponding graphs are plotted.

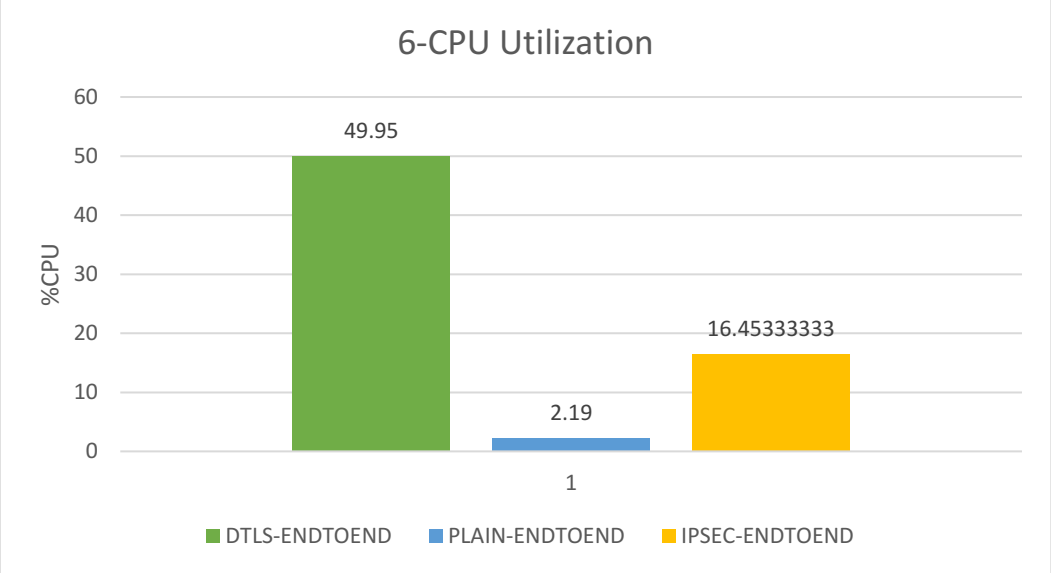


Figure 41 Test-6: CPU Utilization

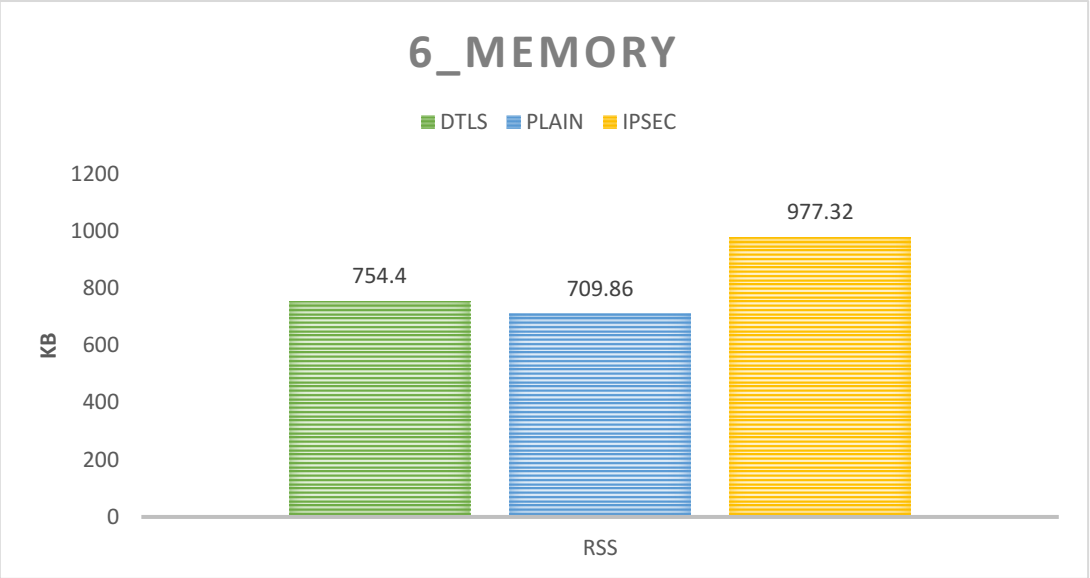


Figure 42 Test-6: Memory Utilization

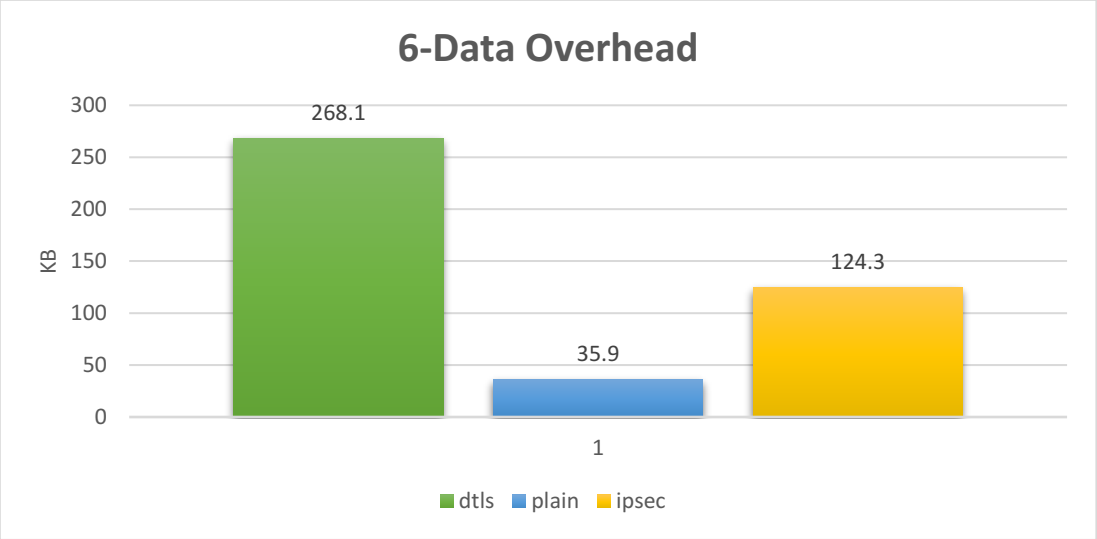


Figure 43 Test-6: Network Overhead

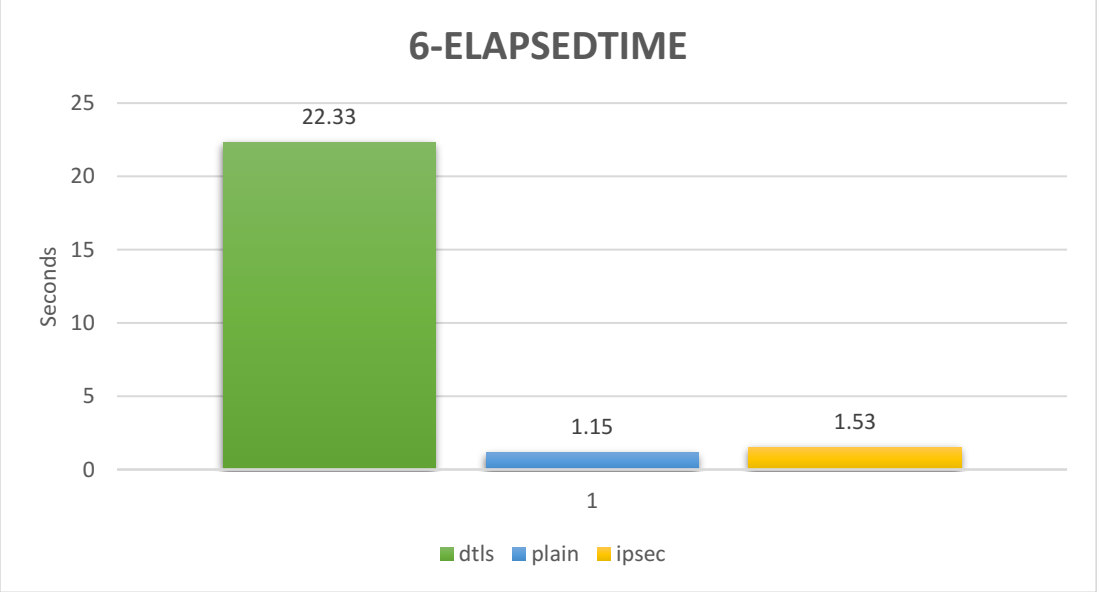


Figure 44 Test-6: Elapsed Time

CPU Utilization: When compared to plain text communication, both IPsec and DTLS use higher CPU load. Among IPsec and DTLS, DTLS uses higher CPU load than IPsec protocol, can be viewed in figure 41.

Memory Utilization: When compared to IPsec communication, DTLS and plain text communication have lesser memory overhead and among DTLS and plain text communication, DTLS has slightly higher memory overhead than plain text communication clearly shown in figure 42.

Network Overhead: When compared to plain text communication, among the overhead of IPsec and DTLS protocols, DTLS has higher overhead than IPsec protocol depicted in figure 43.

Elapsed Time: When compared to the time taken by DTLS for completing 2 non-confirmable requests, the time taken by IPsec is very less and that by plain text communication is even lesser showed in figure 44.

6.8 Test-7: To Send a request to receive a Response with a bad option value.

The graphs for CPU utilization, memory utilization, elapsed time and network overhead are shown below. 200 non-confirmable get requests were sent to the server. 30 iterations were performed for DTLS, IPsec and plain text communication respectively and the average values are calculated. The standard deviation and the value of 95% CI are also calculated. The corresponding graphs are plotted.

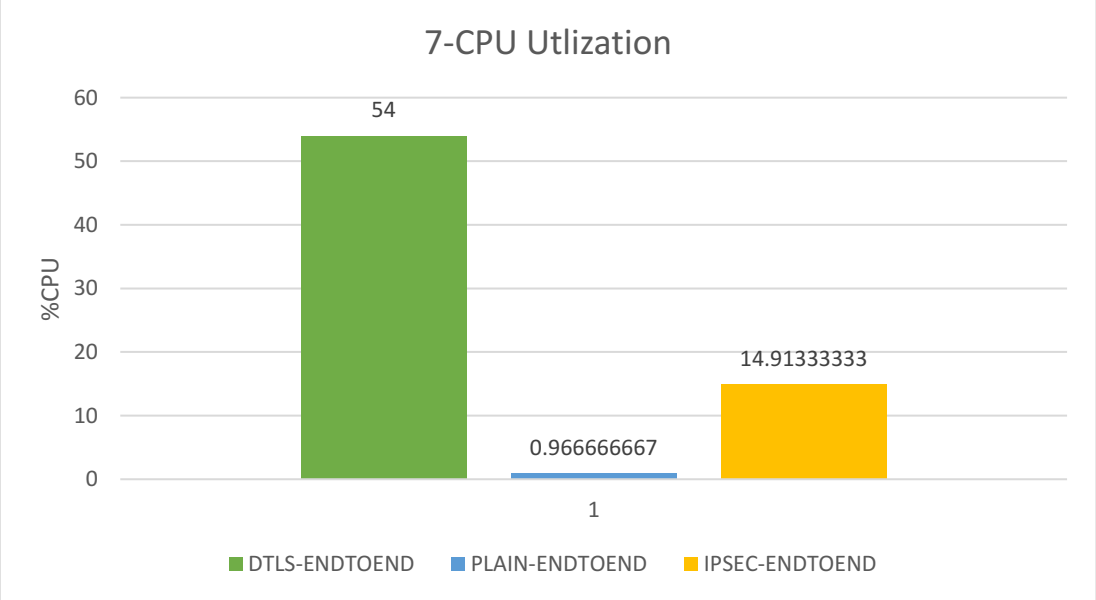


Figure 45 Test-7: CPU Utilization

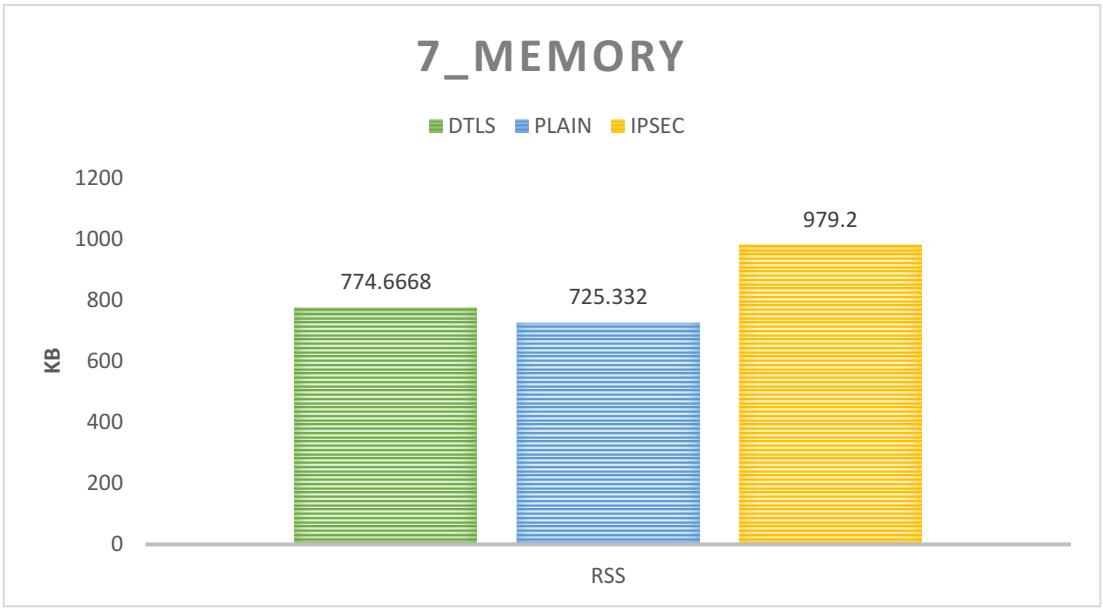


Figure 46 Test-7: Memory Utlization

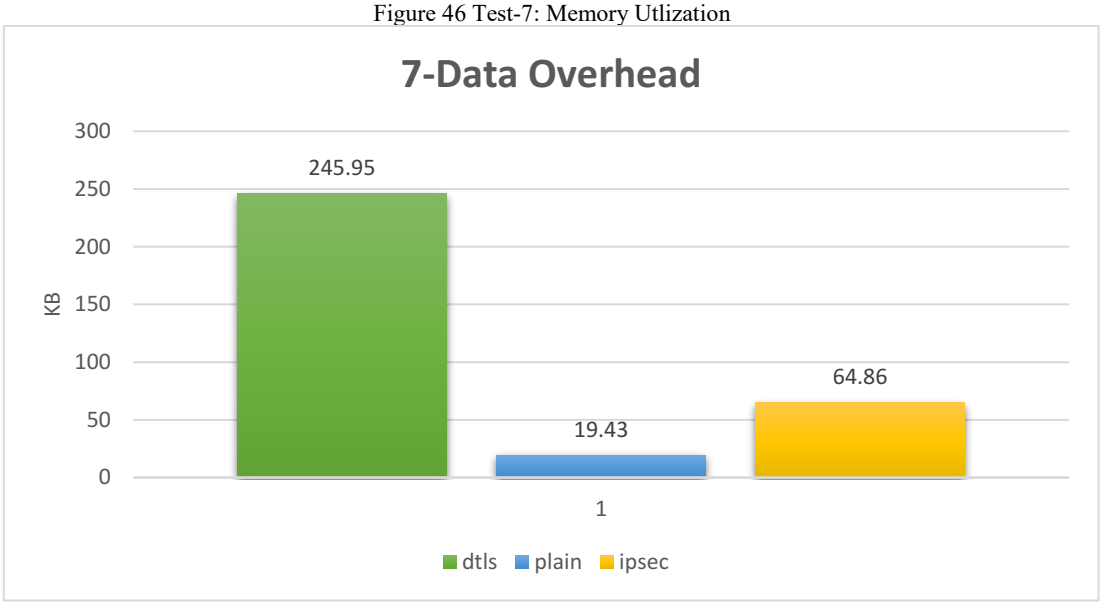


Figure 47 Test-7: Network Overhead

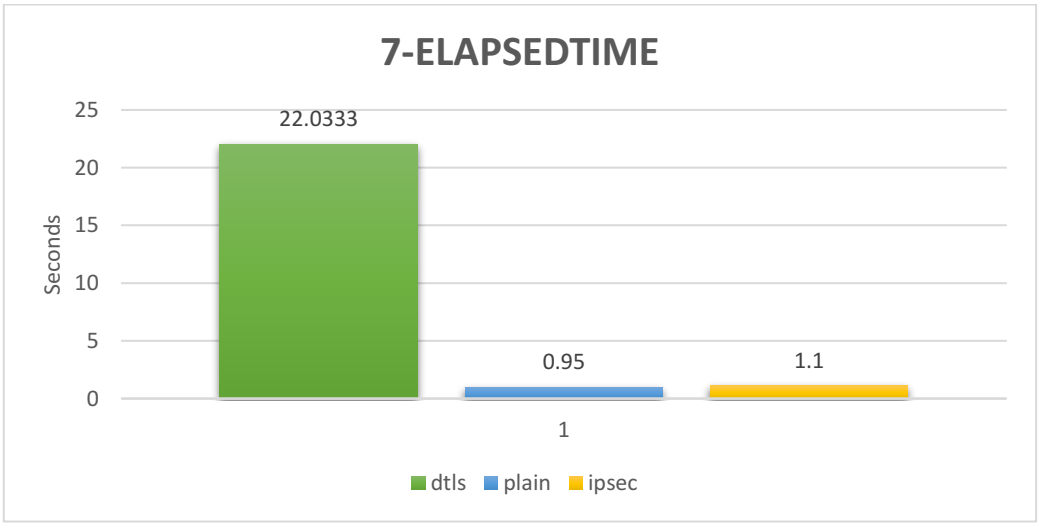


Figure 48 Test-7: Elapsed Time

CPU Utilization: It is evident from the graph corresponding to figure 45 that the highest amount of the CPU is used by DTLS protocol followed by IPsec and then by plain text communication.

Memory Utilization: For plain text communication if the memory allocated is taken as a degree for comparison, then DTLS has little more memory being allocated for the protocol while IPsec has much more being allocated for it, noticeable in figure 46.

Network Overhead: DTLS has the most overhead up to 8 times the value for plain text communication and 4 times the value for IPsec communication which is pointed in figure 47.

Elapsed Time: The time taken by plain text communication and IPsec protocol is in the difference of 0.15 seconds in the range of 1 second while it is about 22 seconds for the DTLS protocol to complete the tests to send a request to receive a bad option response.

6.9 Previous Works on DTLS and IPsec Protocols

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

Comparison of TLS, DTLS and IPSec protocols [4]	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

Field	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

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

Securing IP based communication with IoT [6]	Communication Overhead
802.15.4 Headers	168 Bytes
6LoWPAN Headers	480 Bytes
UDP Headers	96 Bytes
Application	487 Bytes

6.10 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 because DTLS protocol has its own validation scheme for ensuring reliability and so does CoAP which leads to separate implementations. 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.

6.11 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. 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 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 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 mandatory.
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.
[23]–[33]

7 CONCLUSION AND FUTURE WORK

7.1 CONCLUSIONS

RQ:3- Is there a significant difference in performance overhead between DTLS and IPsec?

From our detailed experimentation and analysis, we conclude that there is a significant difference in performance overhead between DTLS and IPsec protocols. For all the tests conducted under the research, the performance metrics, CPU Utilization, Network Overhead and Elapsed time when the communication is secured with DTLS are very high compared to the metrics obtained on securing the communication with IPsec. For Memory utilization, IPsec secured communication consumes higher memory than the memory consumed when secured by DTLS communication. Based on the results, tradeoffs can be established between the protocols.

RQ:4-Based on performance overhead and limitations which protocol (IPsec or DTLS) is best suited to provide confidentiality?

Based on the performance overhead and limitations after the research is done, tradeoffs are established about the protocol to be used to provide confidentiality. If memory is a constraint, DTLS protocol is best suited to provide confidentiality and if computational power is a constraint, then IPsec protocol is the best to provide confidentiality.

Other conclusions from the thesis are, there have been works done previously done both on IPsec and DTLS but the performance metrics considered in their research were different from my research and there has been a contribution to science by the research where it was concluded that DTLS is better suited to provide confidentiality when memory is a constraint and IPsec is better suited if energy is a constraint. The contribution to science is valid according to me, one could choose from the results of the thesis which protocol to use according to his requirements from IPsec and DTLS to encrypt the data. The protocols are the most widely used to provide security and confidentiality, so comparison of the metrics would be a fair contribution to science. 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, since the 6LBR typically does not perform any authentication.

7.2 FUTURE WORK

The Research mainly focuses on the impact of enabling security in the constrained networks w.r.t CPU Utilization, Memory Utilization, Elapsed Time and Network Overhead. As Future work, one could consider the impact of security on constrained devices w.r.t. other metrics such as energy consumption, throughput. Implementation on real devices is also an interesting topic to work on and even other protocols like Host Identity Protocol could also be considered.

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] M. Panwar and A. Kumar, "Security for IoT: An effective DTLS with public certificates," in *2015 International Conference on Advances in Computer Engineering and Applications*, 2015, pp. 163–166.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] M. Kirsche and J. Hartwig, "A 6LoWPAN Model for OMNeT++: Poster Abstract," in *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, ICST, Brussels, Belgium, Belgium, 2013, pp. 330–333.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.
[18] 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.
[19] 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.
[20] 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.
[21] 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.
[22] 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.
[23] "DTLS - The Wireshark Wiki." [Online]. Available: https://wiki.wireshark.org/DTLS. [Accessed: 05-Mar-2017].
[24] "RFC 6282 - Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks." [Online]. Available: https://tools.ietf.org/html/rfc6282. [Accessed: 12-Sep-2017].
[25] N. Modadugu and E. Rescorla, "Datagram Transport Layer Security." [Online]. Available: https://tools.ietf.org/html/rfc4347. [Accessed: 05-Mar-2017].
[26] "RFC 6347 - Datagram Transport Layer Security Version 1.2." [Online]. Available: https://tools.ietf.org/html/rfc6347. [Accessed: 12-Sep-2017].
[27] "RFC 4347 - Datagram Transport Layer Security." [Online]. Available: https://tools.ietf.org/html/rfc4347. [Accessed: 12-Sep-2017].
[28] "RFC 7925 - Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things." [Online]. Available: https://tools.ietf.org/html/rfc7925. [Accessed: 12-Sep-2017].
[29] "RFC 7252 - The Constrained Application Protocol (CoAP)." [Online]. Available: https://tools.ietf.org/html/rfc7252. [Accessed: 12-Sep-2017].
[30] "RFC 7967 - Constrained Application Protocol (CoAP) Option for No Server Response." [Online]. Available: https://tools.ietf.org/html/rfc7967. [Accessed: 12-Sep-2017].
[31] M. Mavani and K. Asawa, "Experimental study of IP spoofing attack in 6LoWPAN network," in *2017 7th International Conference on Cloud Computing, Data Science Engineering - Confluence*, 2017, pp. 445–449.
[32] "IPsec," *Wikipedia*. 01-Jan-2017.
[33] "RFC 4301 - Security Architecture for the Internet Protocol." [Online]. Available: https://tools.ietf.org/html/rfc4301. [Accessed: 12-Sep-2017].

APPENDIX

CPU UTILIZATION

Table 1 Average, Standard deviation and 95% CI for CPU Utilization

		DTLS	PLAIN	IPSEC
TEST-1	Average	49.68	1.14	14.95
	Standard Deviation	4.64	0.80	0.53
	95% CI	1.73	0.30	0.19
TEST-2	Average	52.10	1.66	15.83
	Standard Deviation	1.65	0.73	0.59
	95% CI	0.61	0.27	0.22
TEST-3	Average	50.55	1.15	15.25
	Standard Deviation	3.04	0.51	0.71
	95% CI	1.13	0.19	0.26
TEST-4	Average	52.18	1.15	16.67
	Standard Deviation	2.28	0.51	0.68
	95% CI	0.85	0.19	0.25
TEST-5	Average	53.47	3.28	17.71
	Standard Deviation	0.75	0.90	0.80
	95% CI	0.28	0.33	0.30
TEST-6	Average	49.95	2.19	16.45
	Standard Deviation	2.97	0.87	0.66
	95% CI	1.11	0.32	0.24
TEST-7	Average	54	0.96	14.91
	Standard Deviation	1.71	0.26	0.15
	95% CI	0.63	0.09	0.05

MEMORY UTILIZATION

Table 2 Average, Standard deviation and 95% CI for RSS

		DTLS	IPSEC	PLAIN
TEST-1	Average	752.53	723.73	970.13
	Standard Deviation	33.13	10.50	11.24
	95% CI	12.37	3.92	4.19
TEST-2	Average	745.06	718.4	971.86
	Standard Deviation	7.97	8.72	8.85
	95% CI	2.97	3.25	3.30
TEST-3	Average	747.06	721.33	974.13
	Standard Deviation	9.45	7.87	9.68

	95% CI	3.52	2.94	3.61
TEST-4	Average	748.26	720.26	974.80
	Standard Deviation	8.23	8.55	9.82
	95% CI	3.07	3.19	3.66
TEST-5	Average	749.86	720.13	970.53
	Standard Deviation	7.44	8.84	8.01
	95% CI	2.78	3.30	2.99
TEST-6	Average	754.40	709.86	977.33
	Standard Deviation	8.00	9.13	10.81
	95% CI	2.98	3.41	4.03
TEST-7	Average	774.66	725.33	979.2
	Standard Deviation	8.30	8.94	10.05
	95% CI	3.10	3.34	3.75

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.