Chapter 1
Introduction

1.1 Introduction:

In IoT, Wireless Sensor Networks (WSNs) are supposed to interact and exchange information and data with objects outside their own internal network. So interoperability is required on that purpose. This interoperability requirement is forcing to migrate from proprietary protocol stacks to adopt and standardized solutions, getting benefit of the use of IPv6 through the Low power Wireless Personal Area Network (6LoWPAN) and IEEE 802.15.4 protocols [1][2].

This architecture enables sensor nodes in a WSN to be directly addressed and acted as producers of data consumed by users on the standard Internet or by users in the IoT. As WSN uses public medium such as air, that makes the data vulnerable to being modified, injected and eavesdropped. Therefore, security is always a concern, especially in application where personal and public security data is transported. IPv6 over Low-power Wireless Personal Area Network (6LoWPAN) enables the use of IP in IEEE 802.15.4 low power and lossy wireless networks such as wireless sensor network (WSN). Such IP-connected sensor devices are becoming part of the Internet hence forming the Internet of Things (IoT) or strictly speaking IP-connected IoT.

Security is particularly important for the things as they are connected to the untrusted and unreliable Internet. For the IoT, UDP is mostly used and a new connection-less lightweight Constrained Application Protocol (CoAP) [3] is proposed. CoAP proposes to use Datagram Transport Layer Security (DTLS) as a security protocol for automatic key management and data encryption and authentication. CoAP with DTLS support is termed as secure-CoAP (CoAPs).

The 6LoWPAN border router acts as a bridge between 6LoWPAN and the conventional Internet. CoAPs enabled devices can securely communicate with Internet hosts such as laptops, smartphones, etc. that support DTLS.

The Maximum Transmission Unit (MTU) for 802.15.4 is 127 bytes. Also, in low-power WSN sending and receiving bits is much more expensive than local processing. IoT devices can use the 6LoWPAN protocol to compress the long IP layer headers. DTLS was designed for the Internet and not for resource constrained IoT devices. We can exploit 6LoWPAN compression capabilities to compress the DTLS headers and messages. In this paper we provide 6LoWPAN compression mechanisms for the DTLS protocol. To the best of our knowledge this is the first proposal that enables a lightweight DTLS support for the IoT.

1.2 Motivation:

We are getting closer to a smart world with the help of IoT. Now a days wireless devices are becoming more popular and comfortably bearable.But in this sector, security is an important issue. In IOT, we have many heavyweight protocols which are used for high powered devices. But we got interest to compress heavyweight protocols for devices those have less memory ,low cost and low power. And we also want to secure those devices. That's why we are motivated to do this work.

1.3 Problem Statement:

1.To adapt the DTLS for 6LoWPAN devices.

2. To implement adapted DTLS to practical devices.

1.4 Contribution:

DTLS can be compressed using 6LoWPAN-GHC compression mechanism. We are aiming to compress DTLS for 6LoWPAN devices so that DTLS can be adapted for low-powered devices.

Adapting for low-power devices and following 6LoWPAN compression mechanism we will be able to implement DTLS to practical devices.

1.5 Thesis Organization:

This Report has been organized into five chapters. Each chapter gives distinct concept.

Chapter 2 (Background and Related Work): Here we will discuss background and related works to the proposed work.

Chapter 3 (Proposed Work): What we intend to do and which requirements will be needed for our work.

Chapter 4 (Implementation): Our expecting result on the basis of our challenge is the main concern of our work.

Chapter 5 (Concluding Remarks): This chapter will give the concluding words of our proposed work.

Chapter 2

Background and Related Work

2.1 Background:

Before working on this topics we have found out some of our security issues and we have approached to solve these issues. In our proposed work we have many related terms like IPv6, 6LoWPAN, DTLS, CoAP, UDP etc. Besides many security issues some of them are discussed here:

2.1.1 Security Concerns:

The IoT has to protect against attacks from the following categories: authentication, access control, confidentiality, integrity, and availability.

Authentication involves the mutual verification of routing peers before they share route information and ensures shared data origin is accurate.

Access control is the prevention of unauthorized node use, i.e. making sure nodes are not compromised.

Confidentiality is the protection of information, especially when shared over a publicly accessible medium such as air for wireless.

Integrity involves the protection of data and confirms no unauthorized modifications occur.

Availability, which is specific to IoT, ensures that information is available when required [4].

2.1.1.1 Physical Layer:

Attackers can seize and extract security information, keys, etc. from the device. They may even re-program the device for their own needs. If a group key is used throughout the network, this sort of attack can compromise the entire network. If unique keys are used, this attack is not as damaging. Networks can also experience denial-of-service attacks at a physical layer if the attacker uses jamming or interference equipment. This sort of attack aims to disrupt communications and can be hard to detect [4].

2.1.1.2 Network Layer:

The routing protocols used in the network layer of IoT are similar to the network layer of standard Internet; however, the network layer of IoT is specified towards low-power and lossy networks [4].

2.1.1.3 Failure to Authenticate Attacks:

Node impersonation occurs when an attack gains access to a network as a legitimate node. It would be able to carry attacks, which involve reporting false data or readings, provide bad control messages, or control/affect the traffic flow of the network.

Node resource spam occurs when an attacker continuously joins a network to drain the resources of the network. The attacker would aim to fill up storage memory and potentially take down a portion of the network [4].

2.1.1.4 Confidentiality Attacks:

At the network layer, confidentiality attacks aim to expose routing information or routing exchange data. A deliberate exposure attack happens when a routing entity allows the information to be exposed to an outside entity either due to misconfiguration or by an attack.

Passive wiretapping attacks listen in on information being sent between nodes. These attacks can be countered by encrypting all data used for routing. It is mandatory to implement Advanced Encryption Standard (AES)-128 in Counter with CBC-MAC (CCM) mode for low-power lossy networks. CCM combines the counter mode for encryption and the cipher block chaining message authentication code technique for authentication.

This attack relies on the fact that data link layer and network layer routing information has to remain unencrypted. By analyzing source and destination addresses, attackers can map the network and flow patterns [4].

2.1.1.5 Integrity Attacks:

Unauthorized modification attacks are when attackers change information in a message or in stored data.

Over claiming and disclaiming attacks aim to change the topology and routing data by creating false routes.

Identity attacks, also known as spoofing, happen when an attacker tries to gain access to a device by masquerading as someone else.

Sybil attacks are when the attacker has multiple of these identities. The attacker can create false routing information and disrupt normal routing operations. Byzantine routing information attacks are when a node is compromised by an attacker but still contains a valid identity and security credentials [4].

2.1.1.6 Availability Attacks:

Selective forwarding attacks affecting routing paths and aim to disrupt communications. If the node drops all the packets it receives, it is called a black hole attack .

Wormhole attacks are when two malicious/compromised nodes advertise having a very short path between them.

Sinkhole attacks use a compromised node to advertise good links to attract traffic.

Overload attacks are another type of denial-of-service attack where a malicious node fills the network with random traffic.

HELLO Flood and ACK spoofing attacks are different ways of achieving the same result by leading nodes to believe routes exist when they do not [4].

2.1.2 IPv6:

Internet Protocol version 6 (**IPv6**) [5] is the most recent version of the Internet Protocol (IP), the communications protocol that provides an identification and location system for computers on networks and routes traffic across the Internet.

IPv6 uses a 128-bit address, theoretically allowing 2^{128}, or approximately 3.4×10^{38} addresses. The actual number is slightly smaller, as multiple ranges are reserved for special use or completely excluded from use. The total number of possible IPv6 address is more than 7.9×10^{28} times as many as IPv4, which uses 32-bit addresses and provides approximately 4.3 billion addresses [5].

2.1.3 Comparison between IPv6 and IPv4:

On the Internet, data is transmitted in the form of network packets. IPv6 specifies a new packet format, designed to minimize packet header processing by routers. Because the headers of IPv4 packets and IPv6 packets are significantly different, the two protocols are not interoperable. However, in most respects, IPv6 is an extension of IPv4.

Larger address space:

The main advantage of IPv6 over IPv4 is its larger address space. The length of an IPv6 address is 128 bits, compared with 32 bits in IPv4. The address space therefore has 2^{128} or approximately 3.4×10^{38} addresses.

In IPv4, complex Classless Inter-Domain Routing (CIDR) methods were developed to make the best use of the small address space. The standard size of a subnet in IPv6 is 2^{64} addresses, the square of the size of the entire IPv4 address space. Thus, actual address space utilization rates will be small in IPv6, but network management and routing efficiency are improved by the large subnet space and hierarchical route aggregation.

Multicasting:

IPv6 does not implement traditional IP broadcast, i.e. the transmission of a packet to all hosts on the attached link using a special *broadcast address*, and therefore does not define broadcast addresses. In IPv6, the same result can be achieved by sending a packet to the link-local *all nodes* multicast group at address ff02::1, which is analogous to IPv4 multicasting to address 224.0.0.1. IPv6 also provides for new multicast implementations, including embedding rendezvous point addresses in an IPv6 multicast group address, which simplifies the deployment of inter-domain solutions.

Stateless address auto configuration (SLAAC):

IPv6 hosts can configure themselves automatically when connected to an IPv6 network using the Neighbor Discovery Protocol via Internet Control Message Protocol version 6(ICMPv6) router discovery messages. When first connected to a network, a host sends a link-local router solicitation multicast request for its configuration parameters; routers respond to such a request with a router advertisement packet that contains Internet Layer configuration parameters.

If IPv6 stateless address auto-configuration is unsuitable for an application, a network may use stateful configuration with the Dynamic Host Configuration Protocol version 6(DHCPv6) or hosts may be configured manually using static methods.

Network-layer security:

Internet Protocol Security (IPsec) was originally developed for IPv6, but found widespread deployment first in IPv4, for which it was re-engineered. IPsec was a mandatory specification of the base IPv6 protocol suite, but has since been made optional.

Simplified processing by routers:

In IPv6, the packet header and the process of packet forwarding have been simplified. Although IPv6 packet headers are at least twice the size of IPv4 packet headers, packet processing by routers is generally more efficient, because less processing is required in routers. This furthers the end-to-end principle of Internet design, which envisioned that most processing in the network occurs in the leaf nodes.

The packet header in IPv6 is simpler than the IPv4 header. Many rarely used fields have been moved to optional header extensions.

IPv6 routers do not perform IP fragmentation. IPv6 hosts are required to either perform path MTU discovery, perform end-to-end fragmentation, or to send packets no larger than the default Maximum transmission unit (MTU), which is 1280 octets.

The IPv6 header is not protected by a checksum. Integrity protection is assumed to be assured by both the link layer or error detection and correction methods in higher-layer protocols, such as TCP and UDP. In IPv4, UDP may actually have a checksum of 0, indicating no checksum; IPv6 requires a checksum in UDP. Therefore, IPv6 routers do not need to recompute a checksum when header fields change, such as the time to live (TTL) or hop count.

The *TTL* field of IPv4 has been renamed to *Hop Limit* in IPv6, reflecting the fact that routers are no longer expected to compute the time a packet has spent in a queue.

Privacy:

Like IPv4, IPv6 supports globally unique IP addresses by which the network activity of each device can potentially be tracked. The design of IPv6 intended to re-emphasize the end-to-end principle of network design that was originally conceived during the establishment of the early Internet. In this approach each device on the network has a unique address globally reachable directly from any other location on the Internet.

Network prefix tracking is less of a concern if the user's ISP assigns a dynamic network prefix via DHCP. Privacy extensions do little to protect the user from tracking if the ISP assigns a static network prefix. In this scenario, the network prefix is the unique identifier for tracking and the interface identifier is secondary.

It is not a requirement for IPv6 hosts to use address auto-configuration, however. Yet, even when an address is not based on the MAC address, the interface's address is globally unique, in contrast to NAT-masqueraded private networks. Privacy extensions for IPv6 have been defined to address these privacy concerns, although Silvia Hagen describes these as being largely due to "misunderstanding". When privacy extensions are enabled, the operating system generates random host identifiers to combine with the assigned network prefix. These ephemeral addresses are used to communicate with remote hosts making it more difficult to track a single device.

In addition to the "temporary" addresses mentioned above, there are also "stable" addresses: Interface Identifiers are generated such that they are stable for each subnet, but change as a host moves from one network to another. In this way it is difficult to track a host as it moves from network to network, but with-in a particular network it will always have the same address so that network access controls and auditing can be potentially be configured.

Privacy extensions do not protect the user from other forms of activity tracking, such as tracking cookies or browser fingerprinting.

IPv6 packet

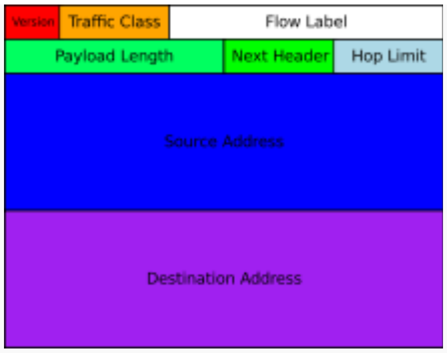


Fig 1: IPv6 packet header [5]

An IPv6 packet has two parts: a header and payload.

The header consists of a fixed portion with minimal functionality required for all packets and may be followed by optional extensions to implement special features.

The fixed header occupies the first 40 octets (320 bits) of the IPv6 packet. It contains the source and destination addresses, traffic classification options, a hop counter, and the type of the optional extension or payload which follows the header. This *Next Header* field tells the receiver how to interpret the data which follows the header. If the packet contains options, this field contains the option type of the next option. The "Next Header" field of the last option, points to the upper-layer protocol that is carried in the packet's payload.

Extension headers carry options that are used for special treatment of a packet in the network, e.g., for routing, fragmentation, and for security using the IPsec framework.

Without special options, a payload must be less than 64KB. With a Jumbo Payload option (in a *Hop-By-Hop Options* extension header), the payload must be less than 4 GB.

Address representation:

The 128 bits of an IPv6 address are represented in 8 groups of 16 bits each. Each group is written as four hexadecimal digits and the groups are separated by colons (:). An example of this representation is 2001:0db8:0000:0000:0000:ff00:0042:8329.

For convenience, an IPv6 address may be abbreviated to shorter notations by application of the following rules.

- One or more leading zeroes from any groups of hexadecimal digits are removed; this is usually done to either all or none of the leading zeroes. For example, the group *0042*is converted to *42*.
- Consecutive sections of zeroes are replaced with a double colon (::). The double colon may only be used once in an address, as multiple use would render the address indeterminate. RFC 5952 recommends that a double colon must not be used to denote an omitted single section of zeroes.

Address uniqueness:

Hosts verify the uniqueness of addresses assigned by sending a neighbor solicitation message asking for the Link Layer address of the IP address. If any other host is using that address, it responds. However, MAC addresses are designed to be unique on each network card which minimizes chances of duplication.

The Manage bit, that indicates whether or not the host should use DHCP to obtain additional addresses

The Other bit, that indicates whether or not the host should obtain other information through DHCP. The other information consists of one or more prefix information options for the subnets that the host is attached to, a lifetime for the prefix, and two flags:

On-link: If this flag is set, the host will treat all addresses on the specific subnet as being on-link, and send packets directly to them instead of sending them to a router for the duration of the given lifetime.

Address: This is the flag that tells the host to actually create a global address.

Link local address:

All interfaces of IPv6 hosts require a link-local address. A link-local address is derived from the MAC address of the interface and the prefix fe80::/10. The process involves filling the address space with prefix bits left-justified to the most-significant bit, and filling the MAC address in EUI-64 format into the least-significant bits. If any bits remain to be filled between the two parts, those are set to zero.

The uniqueness of the address on the subnet is tested with the Duplicate Address Detection (DAD) method.

Global addressing:

The assignment procedure for global addresses is similar to local address construction. The prefix is supplied from router advertisements on the network. Multiple prefix announcements cause multiple addresses to be configured.

2.1.4 The IPv6-connected Internet of Things :

The introduction of 6LoWPAN compressed IPv6 in WSNs, resource constrained devices can be connected to the Internet. This hybrid network of the Internet and the IPv6 connected constrained devices form the IoT. Unlike the Internet where devices are mostly powerful and unlike typical WSN where devices are mostly resource constrained, the things in the IoT are extremely heterogeneous. An IoT device can be a typical sensor node, a light bulb, a microwave oven, an electricity meter, an automobile part, a smart phone, a PC or a laptop, a powerful server machine or even a cloud. Hence the number of potential devices that can be connected to the IoT are in hundreds of billions. This requires the use of IPv6 [7] , a new version of the Internet Protocol that increases the address size from 32 bits to 128 bits (2128 unique addresses). Also, a number of protocols are being standardized to fulfill the specific needs of the IoT [6].

2.1.5 6LoWPAN:

6LoWPAN integrates IP-based infrastructures and WSNs by specifying how IPv6 packets are to be routed in constrained networks such as IEEE 802.15.4 networks [8]. To achieve this, the 6LoWPAN standard proposes context aware header compression mechanisms: the IP Header Compression (IPHC) for the IPv6 header, and Next Header Compression (NHC) for the IPv6 extension headers and the User Datagram Protocol (UDP) header. Due to the limited payload size of the link layer in 6LoWPAN networks, the 6LoWPAN standard also defines fragmentation and reassembly of datagram. 6LoWPAN defines a fragmentation scheme in which every fragment contains a reassembly tag and an offset. When security is enabled or for big application data size, the IEEE 802.15.4 frame size may exceed the Maximum Transmission Unit (MTU) size of 127 bytes; in that case additional fragment(s) are needed. In order to allow compression of header like structures in the UDP payload and the layers above, an extension to the 6LoWPAN header compression, called Generic Header Compression (GHC) [9] is also defined . 6LoWPAN networks are connected to the Internet through the 6LoWPAN Border Router (6BR) that is analogous to a sink in a WSN. The 6BR performs compression/decompression and fragmentation/assembly of IPv6 datagram.

2.1.6 CoAP :

Due to the low-powered and lossy nature of wireless networks in the IoT, connection-less UDP, instead of stream-oriented TCP, is mostly used in the IoT. The synchronous Hyper Text Transfer Protocol (HTTP) is designed for TCP and is infeasible to use in the UDP-based IoT. Therefore, the Constrained Application Protocol (CoAP) , a subset of HTTP is being standardized as a web protocol for the IoT. CoAP is tailored for constrained devices and for machine-to-machine communication [6]. Sensor nodes are resource-constrained devices with limited storage and processing capabilities, are battery powered, and are connected through lossy links. The Internet Protocol (IP) is also proposed for WSN ; until recently IP has been assumed to be too heavyweight protocol to be used in WSN, as additional 40 bytes of IPv6 header are added in each packet . However, IP offers interoperability, scalability, easy of programming, has ready to use hardware, eliminates the need of complex gateways, and has pool of readily available experts. Considering these advantages, IPv6 over low-powered Personal Area Network (6LoWPAN) is standardized. With the advent of 6LoWPAN, it is possible to use IP in resource-constrained WSNs in an efficient way; such networks are called 6LoWPAN networks.

2.1.7 UDP:

User Datagram Protocol (**UDP**) is part of the Internet **Protocol** suite used by programs running on different computers on a network. **UDP** is used to send short messages called datagram but overall,it is an unreliable, connectionless protocol.

2.1.8 Others:

Low-power and Lossy Networks (LLNs) consist largely of constrained nodes (with limited processing power, memory, and sometimes energy when they are battery operated or energy scavenging). These routers are interconnected by lossy links, typically supporting only low data rates.

In order to be useful in a wide range of LLN application domains, RPL separates packet processing and forwarding from the routing optimization objective. Examples of such objectives include minimizing energy, minimizing latency, or satisfying constraints.

Header compression is a mechanism that compresses the IP **header** in a data packet before the packet is transmitted. **Header compression** reduces **network** overhead and speeds up the transmission of Real-Time Transport Protocol (RTP) and Transmission Control Protocol (TCP) packets.

2.2 Related Works:

There are some works previously done related to this topic. Here will focus on some of the previous work [10].

2.2.1 6LoWPAN And DTLS:

DTLS is a protocol used to secure datagram traffic for client/server applications. DTLS is composed of a Record Protocol that carries other protocols such as Handshake, Alert, and application data.The initial Handshake authenticates the server and optionally the client using a Public Key Infrastructure. The 6LoWPAN standard defines IP Header Compression for the IP header and Next Header Compression for the IP extension headers and the UDP header. 6LoWPAN also defines fragmentation schemes for the packets that do not fit in 127 bytes MTU.In order to use heavyweight security protocols, such as DTLS, for the resource constrained IoT we need to apply 6LoWPAN header compression mechanisms to these security protocols as well. A 6LoWPAN extension for IP security is also proposed and evaluated.

2.2.2 Integrating Compressed DTLS into 6LoWPAN:

To cope with the size limitations of IEEE 802.15.4 link layer frames to compress the DTLS headers.The 6LoWPAN standard does not provide ways to compress the UDP payload and the layers above. However,recently 6LoWPAN-GHC has been proposed as a plug-in for 6LoWPAN which can be used to compress the UDP payload. DTLS as being part of the UDP payload can be compressed using 6LoWPAN-GHC. The proposed ID bits in the NHC for UDP-GHC are used to differentiate NHC for UDP from NHC for UDP-GHC. The ID bit 11010 in the NHC for UDPGHC, as defined in the 6LoWPAN-GHC,indicates that the UDP payload is compressed with 6LoWPAN-GHC 1. Once the UDP-GHC is defined we can compress DTLS by providing GHC for the DTLS messages. When the 6LoWPAN-GHC for DTLS is completed we use the STOP code, set to 10010000 in the 6LoWPAN-GHC draft , to indicate that the DTLS header compression is completed. In the following section we define 6LoWPAN-GHC for the Record header, Handshake header, ClientHello message, and ServerHello message. We can use the same compression techniques to compress the other Handshake messages [11].

2.2.3 6LoWPAN-GHC for the DTLS Record and Handshake:

We propose 6LoWPAN-GHC for compressing the Record and Handshake headers. We have two cases here. In the first case we compress both the Record header and the Handshake header using a single encoding byte. In other words the 6LoWPAN-GHC for Record+Handshake (6LoWPAN-GHCRHS) byte defines an encoding for both the Record header and the Handshake header. I n the second case (when the Handshake protocol is completed) we define 6LoWPAN-GHC for the Record header (6LoWPAN-GHC-R) where the fragment field in the Record header is an application data rather than a Handshake header as in the first case.

· The first four bits in the 6LoWPAN-GHC-RHS represent the ID field. We set these bits to 1000. These are needed to comply with 6LoWPAN-GHC encodings. In case of 6LoWPAN-GHC-R we set the ID bits to 1001.

· V: If 0, the version will be DTLS latest version and the field is elided. Currently the latest version is 1.2. If 1, the version field is carried inline.

· EC: If 0, an 8 bit epoch is used and the left most 8 bits are elided. If 1, all 16 bits of the epoch are carried inline. In most cases the actual epoch is either 0 or 1.

· SN: If 0, a 16 bit sequence number is used and the left most 32 bits are elided. If 1, all 48 bits of the sequence number are carried inline. In case of 6LoWPAN-GHC-R, as shown in Figure 2, we use two bits for SN and can more effectively compress the sequence number field. Here if SN is 00, a 16 bit sequence number is used and the left most 32 bits are elided. If 01, a 32 bit sequence number is used and the left most 16 bits are elided. If 11, all 48 bits of the sequence number are carried inline. SN bits 10 is reserved and are not used.

· F: If 0, the handshake message is not fragmented and the fields fragment offset and fragment length are elided. This is a common case when the handshake message is not bigger than maximum record size. If 1, the fields fragment offset and fragment length are carried inline. In the Record header, content type field is always carried inline; also, msg type and message seq fields in the Handshake header are always carried inline. The length filed in the Record and Handshake headers are always elided as they can be inferred from the lower layers: either from the 6LoWPAN fragmentation header or the IEEE 802.15.4 header.

2.2.4 6LoWPAN-GHC for ClientHello

We propose 6LoWPAN-GHC for the ClientHello message (6LoWPAN-GHC-CH). During the handshake process the ClientHello message is sent twice - with cookie and without cookie. Figure 3 shows 6LoWPAN-GHC encoding for the
ClientHello message.

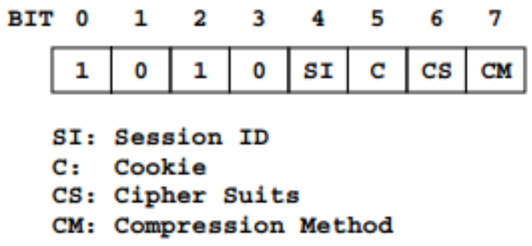


Fig 2: 6LoWPAN-GHC header for ClientHello [10]

The function of each compressed header field is described below:

· The first four bits in the 6LoWPAN-GHC-CH represent the ID field. For 6LoWPAN-GHC-CH we set the ID bits to 1010.

· SI: If 0, the session id is not available and this field and 8 bits of prefixed length field are elided. The actual session id field in the ClientHello contains 0 to 255 bits; however, it is always prefixed with an 8 bit field that contains the size of the session id. The ClientHello message uses session id only if the DTLS client wants to resume the old session. If SI is 1, the session id field is carried inline.

· C: If 0, the cookie field is not available and this field and its prefixed 8 bits length field are elided. The actual cookie field in the ClientHello contains 0 to 255 bits; however, it always has an 8 bit length field that contains the size of the cookie. If C is 1, the cookie field is carried inline.

· CS: If 0, the default (mandatory) cipher suites for the CoAP that supports automatic key management is used and this field and the prefixed 16 bits length field are elided. In the current CoAP draft [12] TLS ECDHE ECDSA WITH AES 128 CCM 8 is a mandatory cipher suit. The actual cipher suites field contains 16 to 216 − 16 bits and is always prefixed with a 16 bit field that contains the size of the cipher suites. If CS is 1, the cipher suites field is carried inline.

· CM: If 0, the default compression method, i.e., COMPRESSION NULL is used and this field and the prefixed 8 bit length field are elided . The actual compression methods field contains 8 to 28 − 8 bits; it is always prefixed with an 8 bits field that contains the size of the compression methods. If CM is 1, the compression methods field is carried inline. The *random* field in the ClientHello is always carried inline whereas the version field is always elided.

2.2.5 6LoWPAN-GHC for ServerHello

We propose 6LoWPAN-GHC for the ServerHello message (6LoWPAN-GHC-SH). Figure 4 shows the 6LoWPAN-GHC encoding for the ServerHello message.

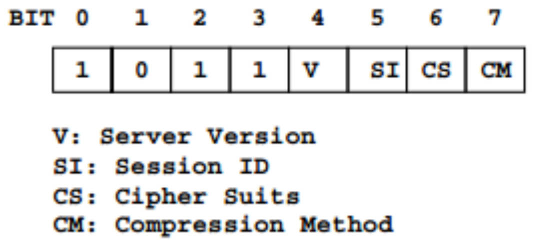


Fig 3: 6LoWPAN-GHC header for ServerHello [10]

The function of each compressed header field is described below:

· The first four bits in the 6LoWPAN-GHC-SH represent the ID field. For 6LoWPAN-GHC-SH we set the ID bits to 1011.

· V: If 0, the version will be DTLS 1.0 and the field is elided. In order to avoid version negotiation in the initial handshake, the DTLS 1.2 standard suggests that the server implementation should use DTLS version 1.0. However the DTLS clients must not assume that the server does not support higher versions. If V is 1, the version field is carried inline.

· SI: If 0, the session id is not available and this field and 8 bits of the prefixed length field are elided. If SI is 1, the session id field is carried inline.

· CS: If 0, the default (mandatory) cipher suites for CoAP that supports automatic key management is used and this field is elided . If CS is 1, the cipher suites field is carried inline.

· CM: If 0, the default compression method i.e. COMPRESSION NULL is used and this field is elided. If CM is 1, the compression methods field is carried inline.
The *random* field in the ServerHello is always carried inline.

2.3 Discussion:

Generic Header Compression compress generic headers and header-like payloads, without a need to define a new header compression scheme for each such new header or header-like payload. The basic approach of GHC's compression function is to define a bytecode for LZ77-style compression. The bytecode is a series of simple instructions for the decompressor to reconstitute the uncompressed payload.

We can see from the above discussion that, previous approach for compressing DTLS headers using GHC, header size can be reduce significantly. But implementation of this compressed DTLS is not mentioned and still a lot of work to be done. More details of performance evaluation is needed for further study.

Table 1 shows how much bits has been compressed in our previous work and how much we are estimating to improve.

DTLS Header	Without Compression	With Compression	% Saving	% Saving Estimation for Our Proposed Work
Record	104	40	62	62
Handshake Header	96	24	75	75
ClientHello	336	264	23	25
ServerHello	304	264	14	20

Table 1: DTLS compression estimation

Chapter 3

Compressed DTLS for CoAP over the 6LoWPAN and Header Compression Approach

3.1 Proposed Work:

We are proposing a compressed DTLS implementation for CoAP over the 6LoWPAN. We aim to compress DTLS by compressing its header so that, this can be implemented into 6LoWPAN for CoAP.

3.2 Solution Approach:

Exploiting 6LoWPAN-GHC compression mechanisms it is possible to reduce DTLS headers sizes significantly. In this work abstract we will work on 6LoWPAN-GHC for the Record, Handshake, ClientHello, and ServerHello.
There will be some number of additional bits we send during DTLS message exchanges with and without 6LoWPAN-GHC compression.

Though some related works discussed before worked on that topic, compressed DTLS is yet to be implemented into 6LoWPAN. The aim of our work is to compress DTLS header following 6LoWPAN header compression mechanism.

DTLS record and Handshake headers are too large to fit into 6LoWPAN. DTLS record DTLSPlaintext structure is as follows:

```
struct {
    ContentType type;
    ProtocolVersion version;
    uint16 epoch;
    uint48 sequence_number;
    uint16 length;
    opaque fragment[DTLSPlaintext.length];
  } DTLSPlaintext;
```

And the structure of Handshake protocol is as follows:

```
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 server_hello:  ServerHello;
      case hello_verify_request: HelloVerifyRequest;  // New field
      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;
```

The 6LoWPAN standard defines IP Header Compression for the IP header and Next Header Compression for the IP extension headers and the UDP header. We can use this 6LoWPAN header compression mechanism to the DTLS Record and Handshake protocol header.

This compression can be applied to ClientHello and ServerHello headers as well. DTLS ClientHello header structure is as follows:

```
struct {
  ProtocolVersion client_version;
  Random random;
  SessionID session_id;
  opaque cookie<0..32>;
  CipherSuite cipher_suites<2..2^16-1>;
  CompressionMethod compression_methods<1..2^8-1>;
} ClientHello;

struct {
  ProtocolVersion server_version;
  opaque cookie<0..32>;
} HelloVerifyRequest;
```

Compressing DTLS headers, overall processing can be reduced. It also makes the protocol lightweight for the low-powered devices. That makes the DTLS suitable for integrating into 6LoWPAN and we can implement that to the practical devices.

Chapter 4
Implementation Details

4.1 Contiki OS:

For the Internet of Things, Contiki [13] is an open source operating system. Contiki is an operating system for networked, memory-constrained systems with a focus on low-power wireless Internet of Things devices. It connects tiny low-cost, low-power microcontrollers to the Internet. Contiki is a powerful features for building complex wireless systems [14].

4.1.1 Features:

Contiki supports per-process optional preemptive multithreading, inter-process communication using message passing through events, as well as an optional graphical user interface (GUI) subsystem with either direct graphic support for locally connected terminals or networked virtual display with Virtual Network Computing (VNC) or over Telnet. Contiki can be run in VMWires.

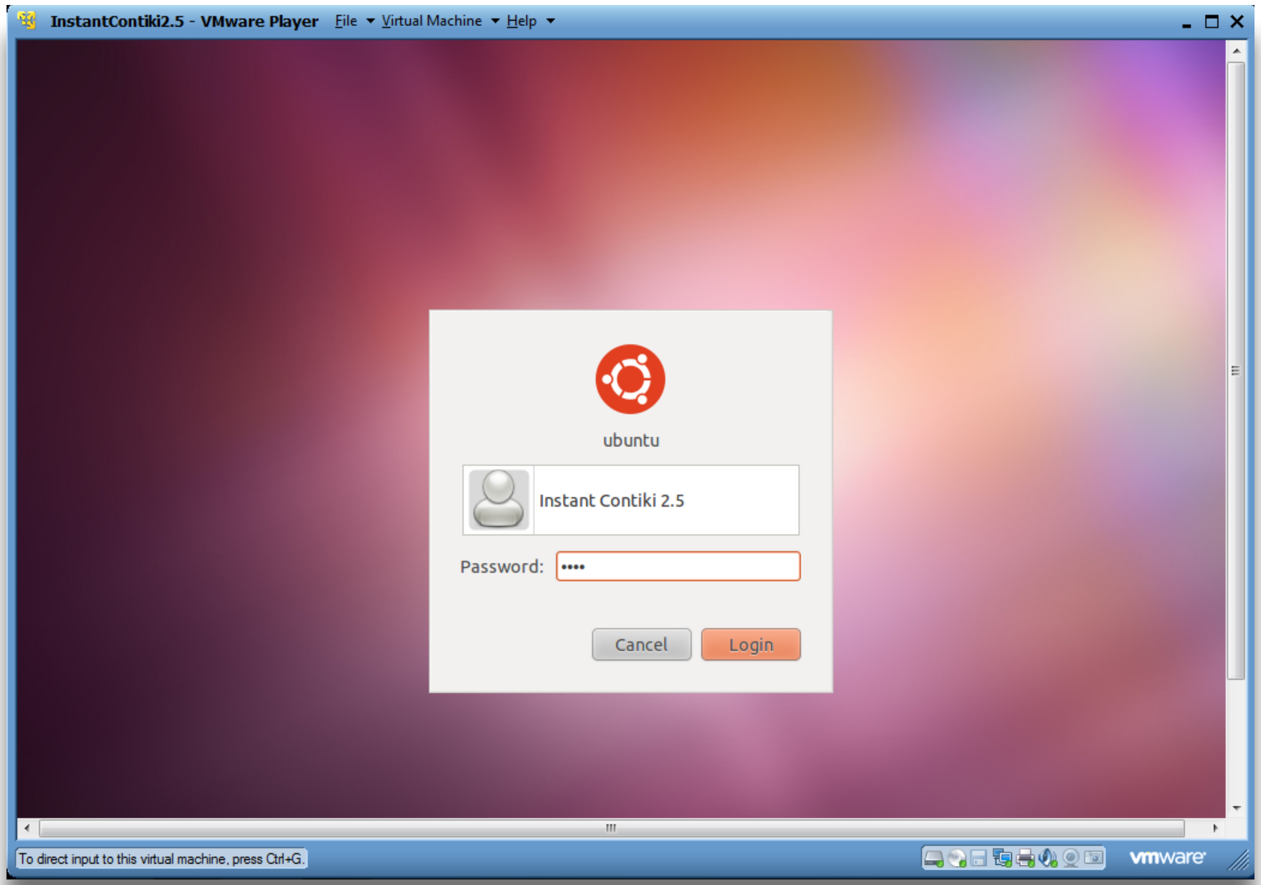

InstantContiki2.5 - VMware Player
File
Virtual Machine
Help
ubuntu
Instant Contiki 2.5
Password:
Cancel
Login
To direct input to this virtual machine, press Ctrl+G.
vmware

Fig 4: Instant Contiki OS running on VM [15]

Some of it's features that are contextual to our proposed work are discussed below:

- Contiki is designed for tiny systems, having only a few kilobytes of memory available. Contiki is therefore highly memory efficient and provides a set of mechanisms for memory allocation: memory block allocation *memb*, a managed memory allocator *mmem*, as well as the standard C memory allocator *malloc*.

- 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.

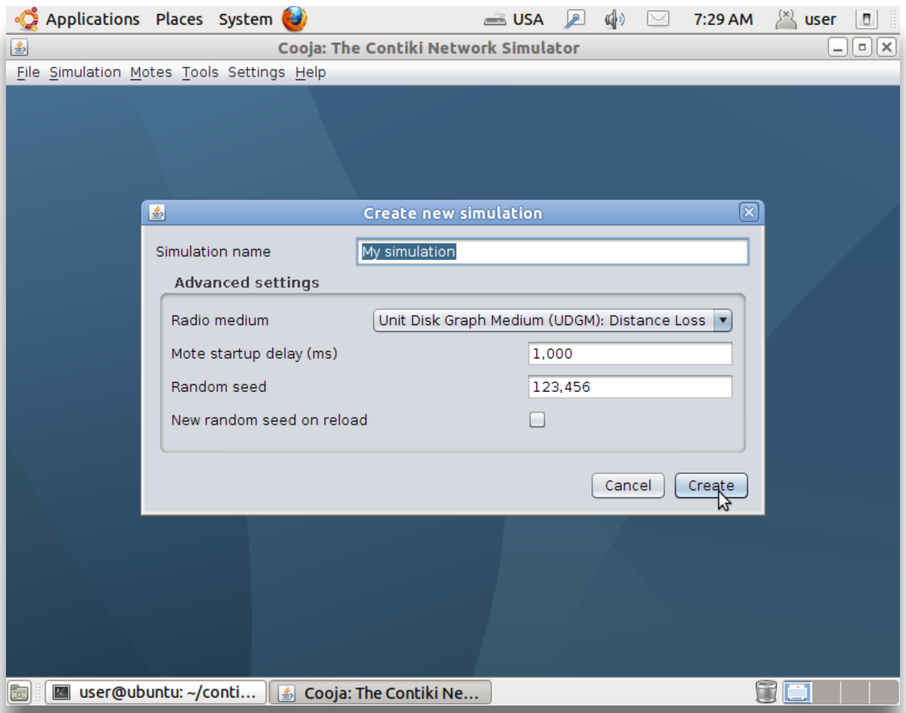
- Contiki provides a full IP network stack, with standard IP protocols such as UDP, TCP, and HTTP, in addition to the new low-power standards like 6lowpan, RPL, and CoAP. The Contiki IPv6 stack, developed by and contributed to Contiki by Cisco, is fully certified under the IPv6 Ready Logo program.

- Contiki is designed to operate in extremely low-power systems: systems that may need to run for years on a pair of AA batteries. 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.
- With Contiki, development is easy and fast: Contiki applications are written in standard C, with the Cooja simulator Contiki networks can be emulated before burned into hardware, and Instant Contiki provides an entire development environment in a single download.

4.2 Cooja Simulator:

Contiki devices often make up large wireless networks. Developing and debugging software for such networks is really hard. Cooja, the Contiki network simulator, makes this tremendously easier by providing a simulation environment that allows us to both see our applications run in large-scale networks or in extreme detail on fully emulated hardware devices.

Some applications of Cooja simulator are shown with following images:


Applications Places System
USA 7:29 AM user
Cooja: The Contiki Network Simulator
File Simulation Motes Tools Settings Help
Create new simulation

Simulation name: My simulation

Advanced settings

Radio medium: Unit Disk Graph Medium (UDGM): Distance Loss

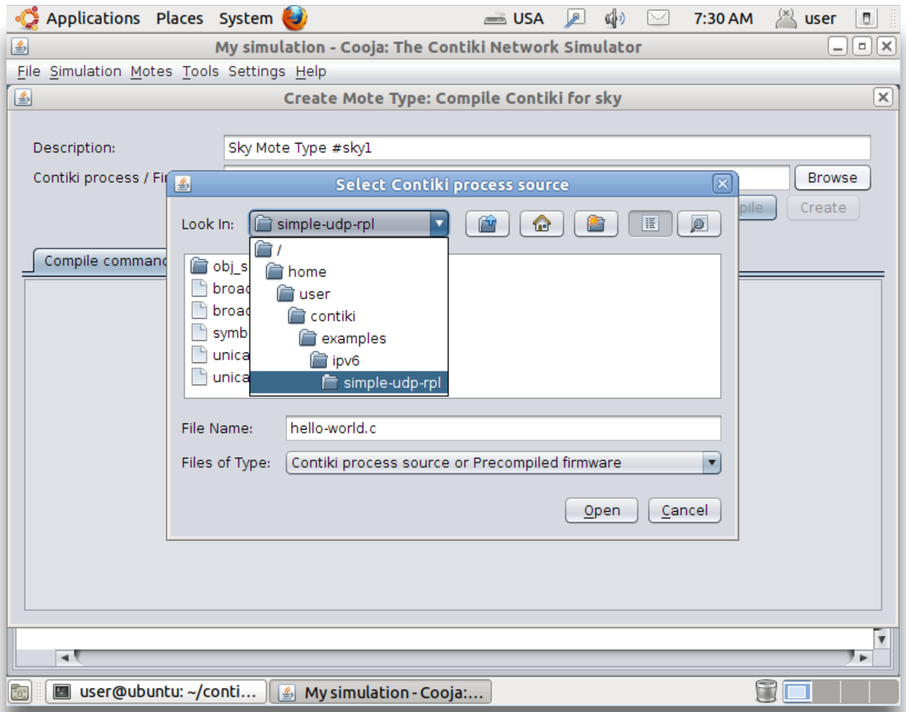
Mote startup delay (ms): 1,000

Random seed: 123,456

New random seed on reload: ☐

Cancel
Create
user@ubuntu: ~/conti...
Cooja: The Contiki Ne...

Fig 5: Create New Simulator [15]


Applications Places System
USA 7:30 AM user
My simulation - Cooja: The Contiki Network Simulator
File Simulation Motes Tools Settings Help
Create Mote Type: Compile Contiki for sky

Description: Sky Mote Type #sky1

Contiki process / Fi
Browse
pile
Create
Compile command
Select Contiki process source

Look In: simulation-udp-rpl

```
/
 home
  user
   contiki
    examples
     ipv6
      simple-udp-rpl
```

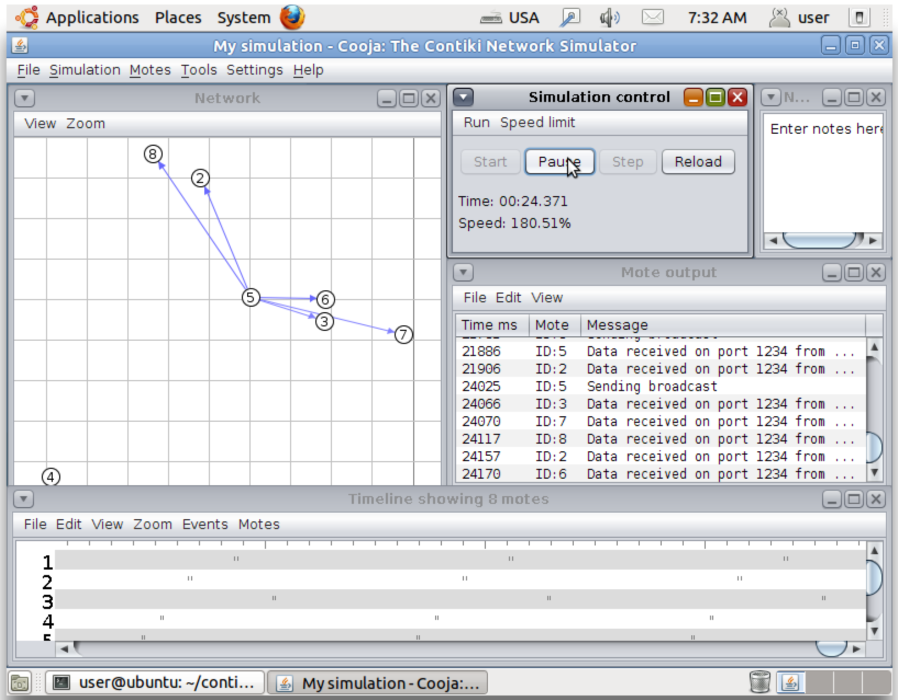
obj_s
broa
broa
symb
unica
unica

File Name: hello-world.c

Files of Type: Contiki process source or Precompiled firmware

Open
Cancel
user@ubuntu: ~/conti...
My simulation - Cooja:...

Fig 6: Simple UDP-RPL communication simulation [15]

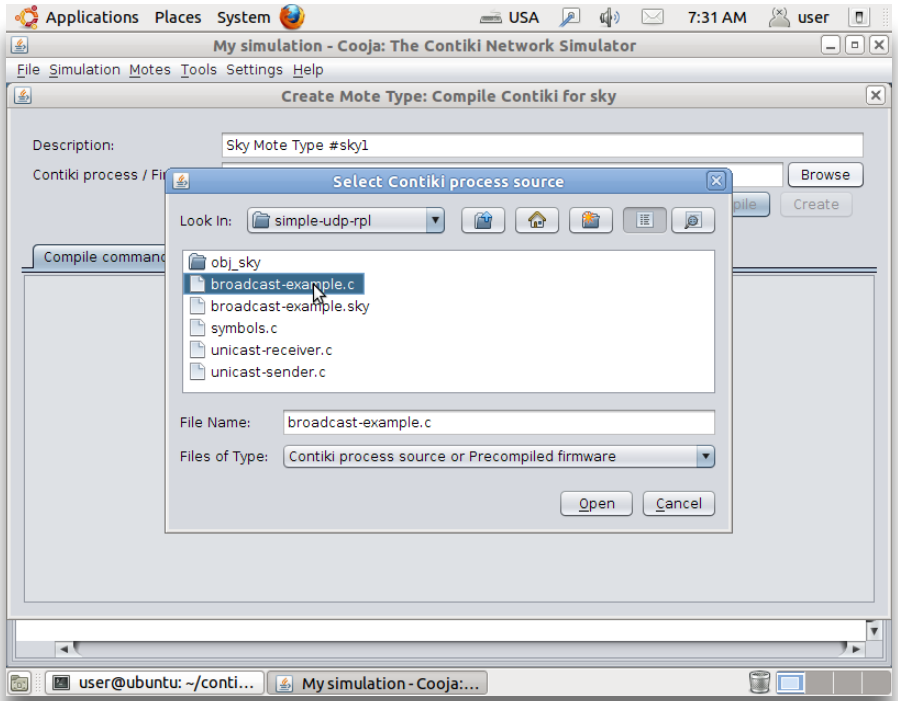

Applications Places System
USA 7:32 AM user
My simulation - Cooja: The Contiki Network Simulator
File Simulation Motes Tools Settings Help
Network
View Zoom
Simulation control
Run Speed limit
Start Pause Step Reload
N...
Mote output
File Edit View
Timeline showing 8 motes
File Edit View Zoom Events Motes
user@ubuntu: ~/conti...
My simulation - Cooja:...

Enter notes here

Time: 00:24.371
Speed: 180.51%

```
Time ms  Mote  Message
21886    ID:5  Data received on port 1234 from ...
21906    ID:2  Data received on port 1234 from ...
24025    ID:5  Sending broadcast
24066    ID:3  Data received on port 1234 from ...
24070    ID:7  Data received on port 1234 from ...
24117    ID:8  Data received on port 1234 from ...
24157    ID:2  Data received on port 1234 from ...
24170    ID:6  Data received on port 1234 from ...
```

Fig 7: Running Simulation [15]


Applications Places System
USA 7:31 AM user
My simulation - Cooja: The Contiki Network Simulator
File Simulation Motes Tools Settings Help
Create Mote Type: Compile Contiki for sky
Browse
Create
Compile command
Select Contiki process source
Open Cancel
user@ubuntu: ~/conti...
My simulation - Cooja:...

Description: Sky Mote Type #sky1

Contiki process / Fi

Look In: simple-udp-rpl

obj_sky
broadcast-example.c
broadcast-example.sky
symbols.c
unicast-receiver.c
unicast-sender.c

File Name: broadcast-example.c

Files of Type: Contiki process source or Precompiled firmware

Fig 8: Cooja works on C programming language [15]

Chapter 5
Concluding Remarks

CoAP enabled hosts will be an integral part of the Internet of Things. DTLS is a standard protocol to secure standard internet. But DTLS is a heavyweight protocol that's why it is difficult to implement in low power and lossy networks as we are concerned about low powered devices. 6LoWPAN is the compressed version of IPv6. If we can compress the DTLS protocol using 6LoWPAN-GHC mechanisms for CoAP we can get better performance from lossy power network nodes and it is also possible to secure communication between sensor nodes and hosts in the Internet.

References

[1] N. Kushalnagar, G. Montenegro, and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals," *Network Working Group RFC 4919*, August 2007. [Online]. Available: http://tools.ietf.org/html/rfc4919

[2] "Approved Draft Amendment to IEEE Standard for Information technology-Telecommunications and information exchange between systems-PART 15.4:Wireless Medium Access Control (MAC) and Phys- ical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs): Amendment to add alternate PHY (Amendment of IEEE Std 802.15.4)," *IEEE Approved Std P802.15.4a/D7, Jan 2007*, pp. –, 2007.

[3] Z. Shelby, K. Hartke, C. Bormann, and B. Frank. Constrained Application Protocol (CoAP), March 2012.

[4] Medaglia, C. M., & Serbanati, A. (2010). An overview of privacy and security issues in the internet of things. In The Internet of Things (pp. 389-395). Springer New York.

[5] IPv6. (2016, August 3). Retrieved from https://en.wikipedia.org/wiki/IPv6; last accesed: 2016, August 3.

[6] Raza, S. (2013). Lightweight security solutions for the internet of things (No. 64). Mälardalen University, Västerås, Sweden.

[7] S. Kent and R. Atkinson. Security architecture for the internet protocol, 1998. http://www.ietf.org/rfc/rfc2401.txt.

[8] IEEE Computer Society. Ieee std. 802.15.4-2006, 2006.

[9] C. Bormann. 6LoWPAN Generic Compression of Headers and Header- like Payloads, September 2012. http://tools.ietf.org/html/draft-bormann- 6lowpan-ghc-05.

[10] Raza, S., Trabalza, D., & Voigt, T. (2012, May). 6LoWPAN compressed DTLS for CoAP. In 2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems (pp. 287-289). IEEE.

[11] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard), January 2012.

[12] Shelby, Z., Hartke, K., & Bormann, C. (2014). The constrained application protocol (CoAP) (No. RFC 7252).

[13] Contiki: The Open Source OS for the Internet of Things. Retrieved from http://www.contiki-os.org/. Last accessed 2016, August 3.

[14] Contiki. (2016, July 22). Retrieved from https://en.wikipedia.org/wiki/Contiki. Last accessed 2016, August 3.

[15] Get Started with Contiki. Retrieved from http://www.contiki-os.org/start.html. Last accessed 2016, August 2.