CATComp: A Compression-Aware Authorization Protocol for Resource-Efficient Communications in IoT Networks

Mahmud Hossain, Golam Kayas, Yasser Karim, Ragib Hasan, *Member, IEEE*, Jamie Payton, *Member, IEEE*, and S. M. Riazul Islam, *Member, IEEE*

***Abstract*—The Internet of Things (IoT) devices exchange certificates and authorization tokens over the IEEE 802.15.4 radio medium that supports a maximum transmission unit (MTU) of 127 bytes. However, these credentials are significantly larger than the MTU and are, therefore, sent in a large number of fragments. As IoT devices are resource constrained and battery powered, there are considerable computations and communication overheads for fragment processing both on the sender and receiver devices, which limit their ability to serve real-time requests. Moreover, the fragment processing operations increase energy consumption by CPUs and radio transceivers, which results in shorter battery life. In this article, we propose CATComp—a compression-aware authorization protocol for constrained application protocol (CoAP) and datagram transport layer security (DTLS) that enables IoT devices to exchange small-sized certificates and capability tokens over the IEEE 802.15.4 media. CATComp introduces additional messages in the CoAP and DTLS handshakes that allow communicating devices to negotiate a compression method, which devices use to reduce the credentials' sizes before sending them over an IEEE 802.15.4 link. The decrease in the size of the security materials minimizes the total number of packet fragments, communication overheads for fragment delivery, fragment processing delays, and energy consumption. As such, devices can respond to requests faster and have longer battery life. We implement a prototype of CATComp on Contiki-enabled RE-Mote IoT devices and provide a performance analysis of CATComp. The experimental results show that communication latency and energy consumption are reduced when CATComp is integrated with CoAP and DTLS.**

***Index Terms*—6LoWPAN, authentication, authorization, capability-based access control (CapBAC), compression, constrained application protocol (CoAP), datagram transport layer security (DTLS), Internet of Things (IoT), token.**

Manuscript received January 26, 2021; revised March 31, 2021 and June 2, 2021; accepted June 5, 2021. Date of publication June 24, 2021; date of current version January 24, 2022. This work was supported in part by the U.S. National Science Foundation (NSF) under Grant DGE-1723768, Grant ACI-1642078, Grant CNS-1351038, and Grant CNS-1828363; and in part by the Sejong University Research Faculty Program under Grant 20212023. *(Mahmud Hossain, Golam Kayas, Yasser Karim, Ragib Hasan, Jamie Payton, and S. M. Riazul Islam contributed equally to this work.) (Corresponding authors: Mahmud Hossain; S. M. Riazul Islam.)*

Mahmud Hossain, Yasser Karim, and Ragib Hasan are with the Department of Computer Science, University of Alabama at Birmingham, AL 35294 USA (e-mail: mahmud@uab.edu; yasser@uab.edu; ragib@uab.edu).

Golam Kayas and Jamie Payton are with the Department of Computer and Information Science, Temple University, Philadelphia, PA 19122 USA (e-mail: golamkayas@temple.edu; payton@temple.edu).

S. M. Riazul Islam is with the Department of Computer Science and Engineering, Sejong University, Seoul 05006, South Korea (e-mail: riaz@sejong.ac.kr).

Digital Object Identifier 10.1109/JIOT.2021.3092183

I. Introduction

With the recent revolution of the computing devices along with technological advancement in communication, the Internet of Things (IoT) concept is utilized by several application domains [1], such as smart city [2], smart home [3], intelligent healthcare assistance [4], smart transportation management [5], [6], agriculture [7], and so on. Recent research anticipates that in a year, on an average, around one million new IoT devices will be deployed to different application domains for the next few years [8]. IoT devices are resource constrained and operate on low power and lossy networks [9]. These devices are embedded with limited powered CPUs, few megabytes of storages (RAMs and ROMs), and low data-rate radio transceivers [10]. As such, IoT applications adopt lightweight communication protocols [11].

IoT devices use the Internet protocol version 6 (IPv6) for addressing [12]. They operate in the IPv6 over low-power wireless personal area networks (6LoWPAN) [13], [14] and communicate over the IEEE 802.15.4 low-powered and lossy media, which has limited bandwidth (128 Kbit/s) and a maximum transmission unit (MTU) of 127 bytes [15]. Furthermore, IoT applications use the constrained application protocol (CoAP) [16] for communications as it is designed especially for resource-limited devices and constrained networks. Additionally, in lossy networks, devices adopt the user datagram protocol (UDP) instead of the transmission control protocol (TCP) as the transport layer protocol to avoid communication overheads for large-sized TCP headers and the three-way handshake [17], [18]. As such, in constrained networks, CoAP utilizes datagram transport layer security (DTLS) [19] for end-to-end communication security, such as authentication, integrity, and confidentiality, and the capability-based access control (CapBAC) [20], [21] for authorization.

In DTLS, IoT devices are issued X.509 certifies, which they exchange for mutual authentication. In the CapBAC model, IoT devices are issued authorization tokens, also referred to as CapBAC tokens or capability tokens, which contain information on a device's access rights (capabilities) for particular services and resources provided by an IoT device. A sender device attaches its token with a CoAP request. A receiver device validates the token to ensure that the client is authorized to access the requested services or resources.

Communicating devices exchange certificates and authorization tokens as UDP payloads over IEEE 802.15.4 links. However, the minimum size of an X.509 certificate or a CapBAC token is significantly larger than the 127 bytes MTU of an IEEE 802.15.4 medium (see Section III for details). Therefore, a UDP packet that contains a certificate or an authorization token is sent in multiple fragments. The adaptation layer at the 6LoWPAN protocol stack provides supports for fragmentation and reassembly of UDP packets that do not fit in the MTU. Note that we use the terms capability token, authorization token, and CapBAC token interchangeably in this article.

The sizes of certificates or a tokens vary based on the type and amount of information included in these security credentials. Therefore, the number of fragments increases with the increase in the size of the certificates and tokens. The number of fragments is proportional to fragment processing and delivery time. As IoT devices are resource constrained and battery powered, there are considerable computation and communication overheads for processing these fragments (fragmentation, routing, and reassembly) both on the sender and receiver devices, which limit their ability to serve real-time requests. In [9], [22], and [23], it has been discussed several times that packet fragmentation is susceptible to various types of network attacks, such as fabrication, replay, duplication, and buffer exhaustion attacks. The possibility of fragmentation attacks will increase if the number of fragments is large.

In this regard, we propose CATComp—a compression-aware authorization protocol for CoAP and DTLS that enables IoT devices to compress certificates and authorization tokens using standard compression methods to reduce the size of these security credentials. CATComp proposes additional message flights in the CoAP and DTLS handshakes that allow two communicating devices to negotiate a compression method during certificates and tokens exchange. A sender device uses the negotiated method to compress outgoing certificates at the DTLS layer and authorization tokens at the CoAP layer. CATComp also provides methods that allow a receiver device to identify compressed credentials and decompress them at CoAP and DTLS layers. As the number of packet fragments is reduced for the compressed certificates and tokens, the communicating devices have to spend a fewer amount of time on packet fragmentation, fragment delivery, and fragment reassembly, which enable a service device to respond to requests faster. Furthermore, energy consummations are minimized as the fragment processing overheads are minimized, which results in longer battery life. Additionally, the chances of fragmentation attacks are reduced as devices exchange a fewer number of certificate and token fragments when compression is applied using CATComp.

Contributions: The contributions of this article are as follows.

1) We propose handshakes at the DTLS and CoAP layers that enable IoT devices to apply compression and decompression at the application and transport layers.
2) We design various messages that allow a sender device to negotiate a particular compression method with a receiver device while establishing a DTLS session. While the sender device uses the selected compression method to compress X.509 certificates in the DTLS layer, the receiver device uses the negotiated compression method to decompress incoming certificates. The certificate compression at the DTLS layer reduces the number of packet fragments for X.509 certificates, which results in faster authentication and session establishment.
3) We also introduce multiple headers at the CoAP layer that allow a sender device to instruct the DTLS layer to compress outgoing authorization tokens. The proposed CoAP headers also enable a receiver device to identify compressed tokens and apply decompression accordingly. The compression of the capability tokens reduces the size of the application payloads; as such, requests are delivered, processed, and responded faster.
4) We implement a prototype of CATComp using RE-Mote IoT devices [24] to demonstrate the feasibility of the proposed scheme. We integrate CATComp with TinyDTLS and CoAP libraries [25], [26] of the Contiki operating system [27].
5) We provide experimental evaluations, which show that communication overheads for fragment delivery and energy costs for fragment processing are reduced when CATComp is adopted to exchange CoAP message over DTLS sessions. The results also show that in CATComp, end-to-end message delivery delays are reduced and the throughput of a service device is increased as smaller sized credentials (certificates and tokens) are exchanged.

Organization: The remainder of this article is organized as follows. Section II provides a background on IoT networking and security. The motivation and the details of proposed scheme are presented in Sections III and IV, respectively. The experiments and evaluations are presented in Section V. Section VI provides a comparative discussion on related works. Finally, we conclude this article in Section VII.

II. Background: IoT Networking and Security

A. Secure Communication Phases

There are two phases for secure communications: 1) establishment of a secure session and 2) information exchange over the secure session. The secure session establishment takes place at the DTLS layer. Communicating peers exchange certificates for authentication and session key selection. In the information exchange phase, a client device sends service-access requests and authorization credentials over the secure session. Devices use CoAP for exchanging information over the secure DTLS session. The information exchange phase can be further divided into two phases: 1) request phase and 2) response phase. In the request phase, a client sends a request and authorization credentials to a service device. In

the response phase, a service device validates the credentials and responds to the request.

B. CoAP as Application Layer Protocol

CoAP implements a request and response model. A client sends a CoAP request message over a UDP packet. The recipient replies with a CoAP response message. The details of a CoAP message are presented in Fig. 1. The CoAP message uses a binary format: a fixed size 4-byte header, a variable length Token, a sequence of CoAP options, and payload. The *Ver* indicates CoAP version and *T* indicates the types of a CoAP message. There are four types of CoAP messages: 1) confirmable message ($T = 0$); 2) nonconfirmable message ($T = 1$); 3) acknowledge message ($T = 2$); and 4) reset message ($T = 3$). A confirmable message must be acknowledged by the recipient through an acknowledge message. However, a nonconfirmable message is not acknowledged. A recipient sends a reset message to reject a request. The TKL indicates the length of a token. The code indicates the message type of request or response, i.e., GET (1), POST (2), PUT (3), and DELETE (4). The message ID indicates the identifier of a message. A token is used to differentiate between concurrent requests. A recipient sends back the token and message ID in the response message.

Zero or more option fields may follow a token. CoAP options are similar to the hypertext transport protocol (HTTP) headers. Every CoAP option is assigned an option number, such as Uri-Host (1), Uri-Port (3), Uri-Path (11), Content-Format (12), and Uri-Query (15). HTTP headers are represented in the plain text. In contrast, CoAP options are encoded with delta encoding to minimize the size of a message. The option delta field is the difference between two consecutive options. For example, if a message contains the aforementioned options and the order of their appearance in a request message remains same, then the value of their option delta fields is 1, 2, 8, 1, and 3, respectively. To calculate the option number from an option delta value, the option delta values of current and all previous options before it are summed up. For instance, the option number (12) of the content-format option can be computed by summing up options values 1, 2, 8, and 1. The fields option length and option value indicate the size and value of an option field, respectively. The payload marker (0xFF) indicates the end of the options and the start of the payload.

C. CapBAC for Authorization

CapBAC schemes [20], [21] protect IoT services and resources from unauthorized accesses. In the CapBAC model, a client's rights, such as read and write privileges, for accessing particular services and resources are mapped to the capabilities of the client. The capabilities of the client determine if the client is authorized to issue commands (e.g., GET, POST, DELETE, and PUT) to an IoT device. Every client of an IoT-based system is issued an authorization token. The access rights of a client are encoded in the token. A trusted issuer signs the token; therefore, it cannot be forged. As shown in Fig. 2, a client device sends a request

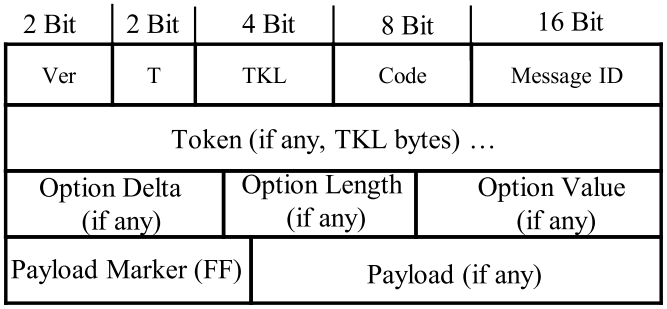

Fig. 1. CoAP headers.

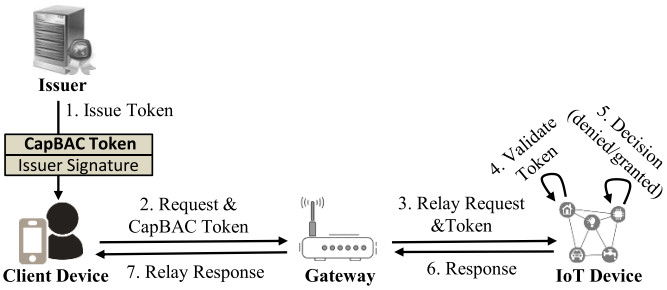

Fig. 2. CapBAC model.

Request: POST /temperature/status
Header: POST (T=CON, Code=1, MID=0x7d34)
Uri-Path: "temperature/status "
Content-Format: "application/json"
Payload: Capability Token

1	CON = 0	0	POST = 2	MID = 0x7d34
OD = 11	OL = 18B	temperature/status		
OD = 1	OL = 2B	application/json (id =50)		
FF	{capability token}			

(a)

Response: "22.3 C"
Header: 2.05 Content (T=ACK, Code=69, MID=0x7d34)
Payload: "22.3 C"

1	ACK=2	0	2.05=69	MID = 0x7d34
"22.3 C" (6 B)				

(b)

Fig. 3. Authorized POST request and ACK response message. OD = option delta. OL = Option length. (a) CoAP request message. (b) CoAP response message.

(e.g., GET or POST) for a particular service (e.g., POST coap://thermostat/temperature/status) to an IoT device. The client attaches its capability token with the request. The IoT device validates the token and ensures that the client is authorized to perform the action. Fig. 3 shows the CoAP messages for an authorized request and response. A CapBAC token is encoded in a JSON format. Fig. 4 shows the content of a CapBAC token. Table I provides the details of fields of a CapBAC token.

```
{
  "UUID" : "9E371587234d004E19AEd5C578702E94",
  "IssueInstant" : "2017-09-15T12:10:47Z",
  "IssuerID" : "admin@users.iha.com",
  "ClientID": "alice@users.iha.com",
  "IssuerPublicKey": "048E3408...684400D314",
  "Signature": "C27D4E5A9...DFFDE8B7E5C42F5203BF9AF",
  "ServiceList": [ {
   "URI": "coap://node786/pacemaker",
   "AccessRight": {
      "ACT": ["GET"," POST "]
   },
   "Obligation": {
    "NotBefore": "08:00:00",
    "NotAfter": "21:00:00"
   },
   "ContextConstraint": {
    "DeviceContext": {
     "OperatingMode": "Energy Saving",
     "BatterySatus": "80%"
    },
    "UserContext": {
     "Location":[ "33.50","86.80"]
    }
   }
  }, {service2}, ...,{servicen-1}, {servicen}]
}
```


Fig. 4. CapBAC token in a JSON envelope [21].

TABLE I
FIELDS OF A CAPBAC TOKEN

Field Name	Description
UUID	A unique random number assigned to every token.
IssueInstant	The time at which the token was issued.
IssuerID	The identity of the issuer.
Signature	The signature of the issuer.
Public Key	The public key of the issuer.
ServiceList	An array of services. A client is granted access to the services mentioned in the list.
URI	An Universal Resource Identifier (URI) assigned to every IoT service.
Access Right	A list of actions. A client is authorized to perform the actions specified in the list.
Obligation	The token must not be accepted before the time specified in NB or after the time mentioned in NA.
Context Constraints	A set of context information used as inputs for token validation.

D. DTLS for Communication Security

In the DTLS [19], IoT devices use X.509 certificates as their identities for authentication [28], [29]. The details of the DTLS handshake for the certificate-based mutual authentication are presented in Fig. 5. *Flight 1:* the client sends the ClientHello record that contains the protocol version and a list of cipher suites and compression methods supported by the client. *Flight 2:* the server replies with the ServerHello record that contains the cipher suite and compression method chosen from the lists offered by the client. The server sends its X.509 certificates (ServerCertificate record), a request (CertificateRequest record) for the client's certificate, and cryptographic materials (ServerKeyExchange record) require to derive a session key. The ServerHello record indicates the

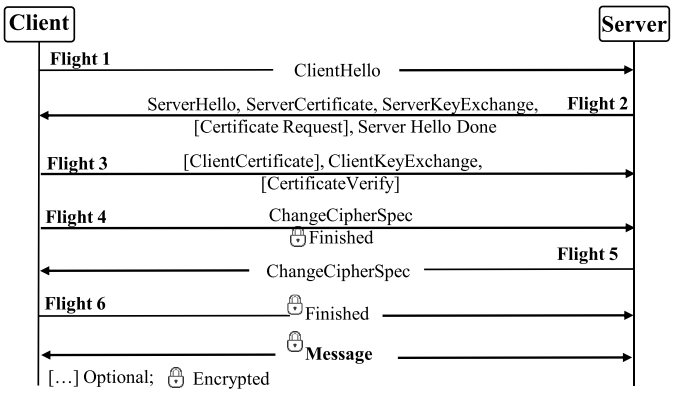

Fig. 5. Fully authenticated DTLS handshake.

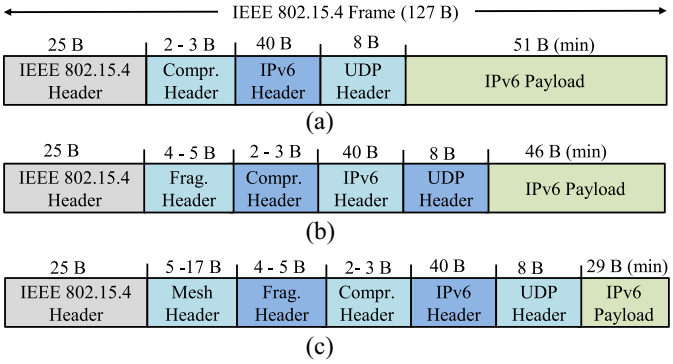

Fig. 6. IEEE 802.15.4 frame. min = minimum. (a) Header Compression. (b) Fragmentation + Header Compression. (c) Mesh Addressing + Fragmentation + Header Compression.

end of Flight 2. *Flight 3:* the client sends its X.509 certificate (ClientCertificate record), a proof of possession of the public key embedded with the certificate (CertificateVerify record), and credentials for deriving a session key (ClientKeyExchange record). *Flight 4–6:* the ChangeCipherSpec record contains the negotiated options (e.g., cipher suite and compression method) for the session. The finish record contains an encrypted message digest of all previous handshake messages.

E. IEEE 802.15.4 as the Communication Medium

6LoWPAN nodes communicate over a low data rate (approximately 250 Kb/s) radio link such as IEEE 802.15.4 [15]. The MTU of the IEEE 802.15.4 medium is 127 bytes. However, the MTU for a IPv6 packet is 1280 octets. Therefore, a full IPv6 packet does not fit in an IEEE 802.15.4 frame. The 6LoWPAN adaptation layer provides functionalities for IPv6 header compression so that an IPv6 packet can fit the 127 bytes MTU. However, if the datagram does not fit the link's MTU, then it is broken into fragments. As shown in Fig. 6, starting from a maximum physical layer packet size of 127 octets and the maximum header overheads of 98 octets, the resultant maximum frame size at the application layer is 29 octets. As a result, application payloads larger than 29 bytes are sent in multiple fragments. Packet fragmentation and reassembly take place in the adaptation layer.

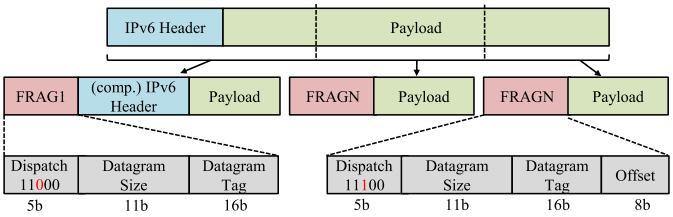

Fig. 7. Packet fragmentation.

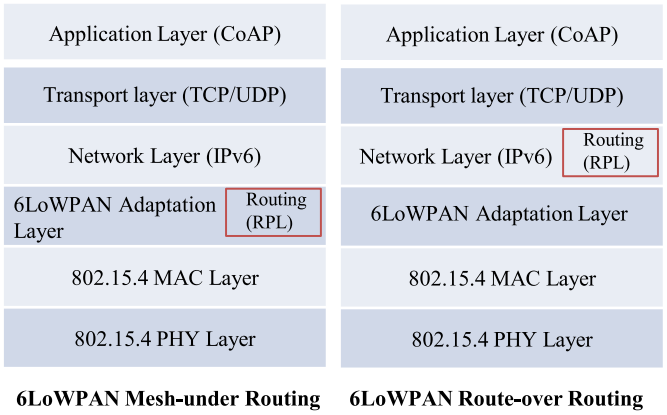

Fig. 8. Position of routing modules.

F. Fragmentation Mechanism

The 6LoWPAN adaptation layer provides functionality for IPv6 header compression and packet fragmentation. In the case of packet fragmentation, each 6LoWPAN fragment contains a fragment header that carries information for in-place reassembly, even for out-of-order fragments. If an entire payload (e.g., IPv6 datagram) fits the MTU of an IEEE 802.15.4 link, then it is unfragmented and the 6LoWPAN encapsulation does not contain a fragmentation header. However, if the datagram does not fit the link's MTU, then it is broken into fragments.

There are two types of fragment header: 1) FRAG1 and 2) FRAGN. The first fragment of a packet contains the first fragment header (FRAG1). In contrast to regular IP fragments, 6LoWPAN fragments only include IP header information in the initial fragment of a packet. The second and subsequent fragments, up to and including the last, contain the second fragment header (FRAGN). As the fragment offset can only express multiples of eight bytes, all fragments for a datagram except the last one must be multiples of eight bytes in length.

As shown in Fig. 7, FRAG1 contains three fields: 1) *dispatch*; 2) *datagram size*; and 3) *datagram tag*. The *dispatch* field is used to distinguish between FRAG1 and FRAGN. The *datagram size* field encodes the size of the entire IP packet before link-layer fragmentation (but after IP layer fragmentation). The value of *datagram size* is the same for all link-layer fragments of an IP packet. The *datagram tag* is a unique per sender and fragmented packet, and is included in each fragment header. In contrast to the FRAG1 header, the FRAGN header contains an additional field *datagram offset*, which indicates the position of the current payload within the original IPv6 packet.

Only the first fragment (FRAG1) contains end-to-end routing information (IPv6 address). However, a receiving node uses the *datagram tag* of the remaining fragments (FRAGN) to correlate them to the FRAG1 to derive IP-based routing or processing decisions for these fragments. Thus, the *datagram tag* enables a receiving node to look up routing information for all the fragments belonging to a fragmented packet after the FRAG1 has been received.

G. Routing in IoT Network

Based on which layer the routing decision—packet or packet fragments forwarding decision—occurs, the 6LoWPAN routing protocols can be classified into two categories [12], [30]—Fig. 8 shows the differences. For the *mesh-under* scheme, the routing decision is taken in the adaptation layer. In this routing scheme, each fragment is perpended with a mesh routing header (see Fig. 6). A forwarding node uses the link layer addresses to derive a routing decision on a per-fragment basis. In contrast, the *route-over* scheme delegates the decision to the network layer on a per-packet basis. An IP packet is fragmented by the adaptation layer and all fragments are sent to the next hop based on the routing table. The next hop reassembles them in order to reconstruct the original IP packet in the adaptation layer when all fragments are received successfully. The reconstruction process starts only when the last fragment arrives. Once reconstructed, the packet is sent to the network layer. Finally, the packet is fragmented again and these fragments are delivered to the next hop.

III. Problem Statement and Motivation

CoAP was designed to avoid fragmentation of a UDP packet. However, the X.509 certificates and authorization tokens are sent in a large number of fragments as they do not fit in a single IEEE 802.15.4 frame. These large number of fragments increase packet delivery time and limit IoT devices to serve a request faster. Additionally, the processing of these fragments—packet fragmentation on the sender end and reassembly on the receiver end—increases energy consumptions by the sender and receiver IoT devices. In this section, we discuss how the sizes of the certificates and authorization tokens can increase communication overheads and energy costs of the IoT devices.

A. Communication Overhead for Certificate and CapBAC Token

The size of a CapBAC token depends on the amount of information it contains on the access rights of a client. Fig. 9 shows a correlation between the number of services to which a client is granted access and the sizes of the authorization tokens issued to the client. From the figure, it can be observed that the size of a token increases when the client is granted access to a higher number of services. This is because the token issued for the higher number of services contains more information on the capabilities of the client than the token issued for the lower number services. It can also be noted that the minimum size of a token is greater than 800 bytes, which

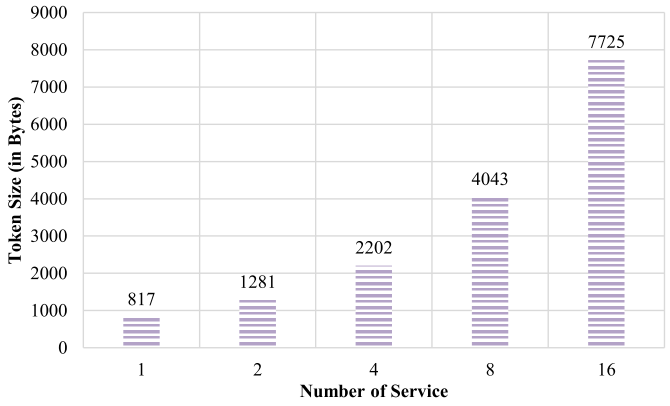

Fig. 9. Service count versus token size.

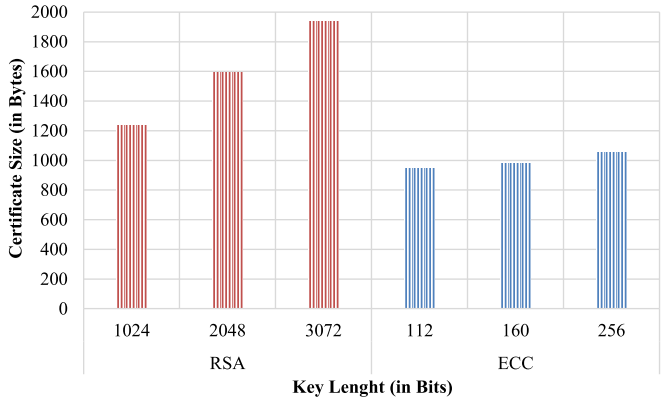

Fig. 10. X.509 certificate size versus public-key length.

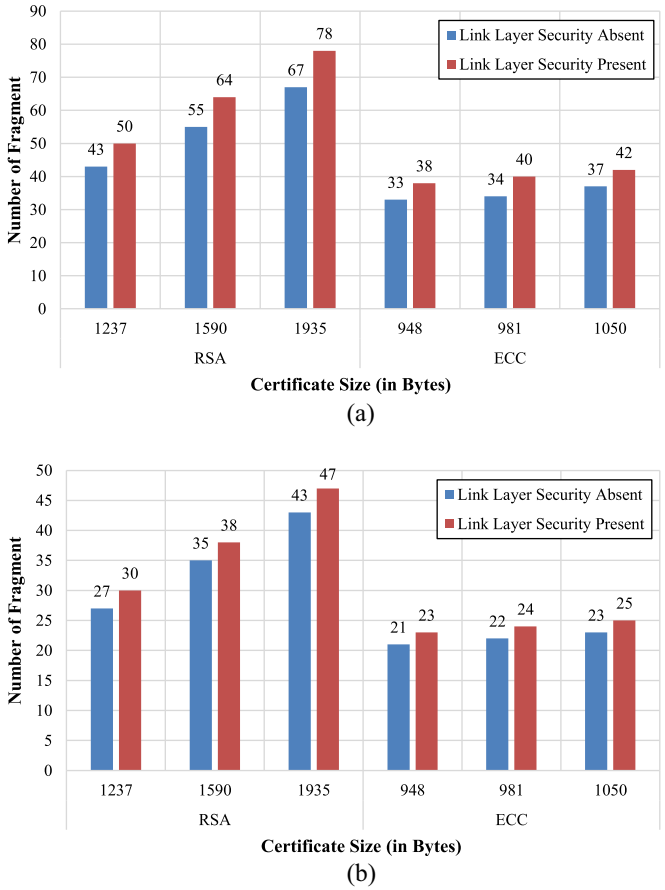

Fig. 11. Analysis of packet fragments for X.509 certificates. (a) Certificate size versus packet fragments (mesh-under routing). (b) Certificate size versus packet fragments (route-over routing).

is larger than the 127 bytes MTU. As a result, a CapBAC token has to be sent in multiple fragments. Additionally, the number of fragments increases with an increase in the token size.

In the DTLS, a client and a server exchange their X.509 certificates for mutual authentication and to set up a secure connection (see Fig. 5). The size of an X.509 certificate varies with the size of the public key attached with the certificate and the type of the public-key cryptography used, such as RSA or elliptic curve cryptography (ECC), used to verify the certificate. As shown in Fig. 10, the size of a certificate increases as the length of a public key increases. From the figure, it can also be noted that the minimum size of an X.509 certificate is larger than 127 bytes MTU. Therefore, certificates are fragmented prior to sending to an IEEE 802.15.4 link.

From the above discussion, it is evident that authorization tokens and X.509 Certificates contribute a notable portion in packet size. In this regard, we posit that authorization tokens and certificates should be compressed to minimize the number of fragments and packet processing delays as well as to enable IoT devices performing mutual authentication and authorization faster.

B. Large Number of Packet Fragments

Uncompressed certificates and capability tokens result in a large number of packet fragments. We provided a correlation between the size of the security materials and the number of fragments in Figs. 11 and 12. From Fig. 11, it can be noted that there is a significant increase in the number of fragments as the size RSA and ECC keys are increased. The number of fragments also increases as a client is given access to a higher number of services (see Fig. 12).

From the figures, it can also be observed that the number of fragments is larger in mesh-under routing than in route-over routing. This is because of the space of an application payload in the IEEE 802.15.4 is reduced by 17 Bytes if Mesh-under routing is adopted—the space for an application payload is 46 and 29 Bytes in mesh-under and route-over routing, respectively (see Fig. 6).

Moreover, the number of packet fragments increases when link-layer security (LLS) is used [31], [32]. The 802.15.4 security layer is handled at the media access control layer, below application control. The 802.15.4 specification defines various security suites for link security. In LLS, an additional 32 bits (4 bytes) message authentication code (MAC) is added to the IEEE 802.15.4 frame. As a result, the room for an application payload is reduced, and the number of fragments is increased. An increasing number of fragments creates significant network overhead. Additionally, these kinds of communications are vulnerable to fragmentation attacks. Even the security layer of

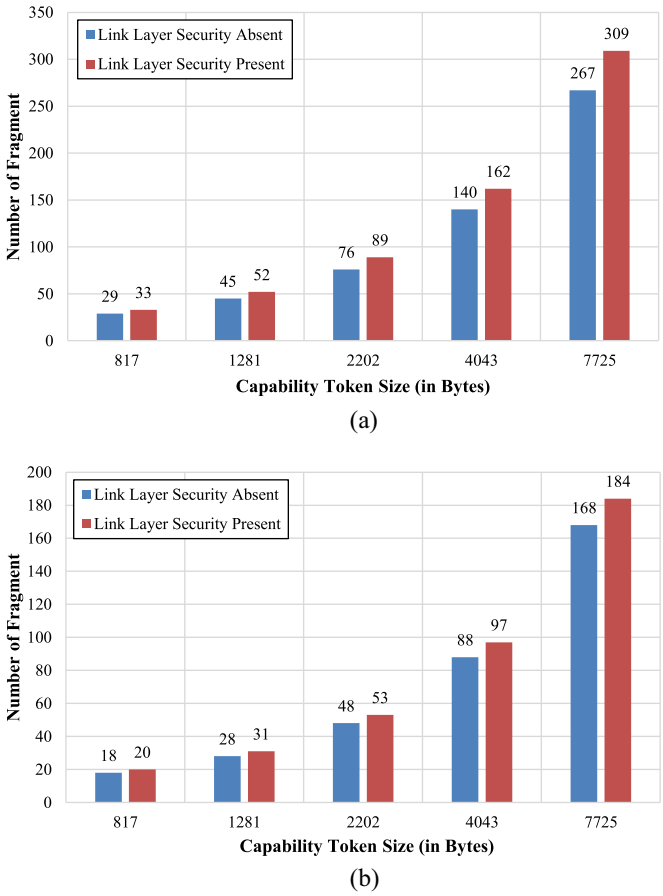

Fig. 12. Analysis of packet fragments for CapBAC tokens. (a) Token size versus packet fragments (mesh-under routing). (b) Token size versus packet fragments (route-over routing).

IEEE 802.15.4 cannot protect the devices from the fragmentation attacks. Therefore, reducing the number of fragments is desirable to minimize the risk of the fragmentation attacks and lower the communication overhead.

C. Increased Request/Response Delivery Delay

The time to deliver a request to a receiver IoT node increases with an increase in the number of packet fragments. A receiver has to spend more time on reassembling the fragments of a request than on preparing a response to the request. Hence, the receiver cannot serve a real-time request. Additionally, the larger the number of fragments, the more the fragment processing overhead on the on-path nodes—the nodes located on the path between a sender and a receiver. The intermediate nodes have to find routes for every fragment. Futhermore, the packet delivery time is larger in route-over routing than in mesh-under routing. In the router-over routing, every on-path node needs to reassemble the fragments to determine the next hop. Additionally, the on-path nodes have to refragment the packet prior to send it to the next hop.

D. Vulnerable to Fragmentation Attacks

The 6LoWPAN fragment reassembly mechanism does not provide support to verify the integrity of an individual packet fragment. A receiver has to reassemble an entire packet to confirm whether or not the packet fragments are modified in transit. Adversaries can exploit this vulnerability of the reassembly method to perform fragmentation attacks [9], such as duplication and alteration. In the duplication attack, an on-path adversary sends an additional fabricated fragment along with the legitimate packet fragments. On the other hand, an adversary modifies a legitimate packet fragment in the alteration attacks. The resource consumption of a receiver increases due to such attacks. A receiver reassembles the packet fragments and learns that the one or more fragment was modified in-transit; therefore, it requests the sender to retransmit the packet. Such fragment reassembly and retransmission increase CPU and memory consumption. Adversaries can perform these attacks on large-sized requests (e.g., certificates and capability tokens) to exhaust reassembly buffer or battery of an IoT node. As such, the compression of certificates and capability tokens will reduce the packet size significantly, which will yield less number of fragments. Hence, the possibility of a fragmentation attack will decrease also.

IV. Proposed Scheme

We propose CATComp—a protocol that facilitates certificate and authorization token compression at DTLS and CoAP layers, respectively. During the DTLS session establishment, CATComp enables a sender device to adopt a compression scheme to reduce the size of X.509 certificates that are used authentication. CATComp also provides the ability to compresses authorization tokens that are attached to CoAP requests. The sizes of the certificates and CoAP requests are reduced after the compression, which results in a minimal number of packet fragments for a service request. Furthermore, the fragment processing overheads are reduced both on the sender and receiver ends as devices need to exchange a fewer number of packet fragments for compressed certificates and tokens. As such, service IoT devices can serve requests faster.

A. Certificate Compression

A client and a server negotiate a compression algorithm for a session by exchanging ClientHello and ServerHello messages (see Flights 1 and 2 of Fig. 5). The compression algorithm is used to compress application data. However, we propose a scheme that compresses the client's and server's certificates (or a chain of certificates) by using the negotiated compression method. The operational model of the proposed certificate compression scheme is shown in Fig. 13.

Step 1: The server receives a list of compression method supported by the client through the ClientHello record (see Flight 1 of Fig. 5). If the server does not support the client provided compression methods, it sends its certificate in the plain text (conventional approach) as shown in steps 2.1 and 3.1. Otherwise, the server follows our proposed approach to send its certificate.

Step 2: The server replies with the ServerHello record that contains the compression algorithm chosen by the server.

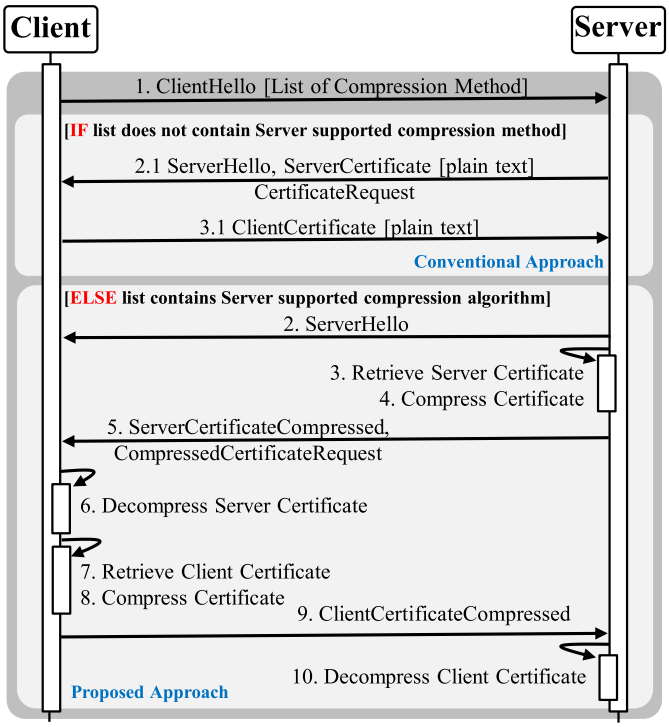

Fig. 13. Operational model of certificate compression procedure.

Algorithm 1: Certificate Compression Procedure

```
Input:  CompessMethod
Input:  certificateChain
Output: compressedCertificate
Output: uncompressLength
if CompessMethod equals NULL or certificateChain equals NULL then
    return NULL ;
else
    struct {
      uncompressedLenght uL;
      compressedCertificateData ccD;
    } certificate_compressed;
    compressedCert ⟵ NULL;
    combinedCert ⟵ EMPTY;
    uncompressLength ⟵ 0;
    for cert in CertificateChain do
        CombinedCert.Append(cert);
        UncompressLength += cert.Length;
    end
    CompressedCert ⟵ CompessMethod.Compress(CombinedCert);
    certificate_compressed.uL ⟵ UncompressLength;
    certificate_compressed.ccD ⟵ CompressedCert;
    if option equals client then
        ClientCertificateCompressed ⟵ certificate_compressed;
        return ClientCertificateCompressed;
    else if option equals server then
        ServerCertificateCompressed ⟵ certificate_compressed;
        return ServerCertificateCompressed;
    else
        return NULL;
    end
end
```

Steps 3–5: The server retrieves the certificate (or the certificate chain) from its memory (ROM or RAM). It is possible that the certificate is already stored in the memory in the compressed format. There are two reasons for which a certificate can be stored in compressed format: 1) to save storage (see storage savings of Section V-B) and 2) to avoid compression during DTLS handshake; thus, to achieve faster authentication and session key establishment. If the certificate is already compressed, then the server moves to step 5. Otherwise, the server provides the certificate and selected compression method as the inputs to Algorithm 1 to compress the certificate. The algorithm first computes the length of the uncompressed certificate and then compresses it. Next the algorithm creates a DTLS handshake record of type *ServerCertificateCompressed*. The server sends the *ServerCertificateCompressed* record and *CertificateRequest* record to the client. We propose two new types of handshakes, such as *ServerCertificateCompressed* and *ClientCertificateCompressed*, to exchange compressed certificates. The DTLS handshake records of type *ServerCertificateCompressed* and *ClientCertificateCompressed* have two fields: 1) *uncompressedLenght* and 2) *compressedCertificateData*. The *unCompressedLength* indicates the length of a certificate before compression and the *compressedCertificateData* indicates the compressed certificate.

Step 6: The client decompresses *ServerCertificateCompressed.compressedCertificateData* and retrieves the certificate in the plain text format. The client discards the certificate if *ServerCertificateCompressed.unCompressedLength* does not match with the length of the uncompressed certificate. The client sends a *bad_certificate* Alert record to the server and aborts the connection.

Steps 7–9: The client also follows the same procedure as described in steps 3–5 to send its compressed certificate. The client generates the *ClientCertificateCompressed* record by executing Algorithm 1 and sends it to the server.

Step 10: The server decompresses *ClientCertificateCompressed.compressedCertificateData* and retrieves the client's certificate in the plain text format. The server aborts the connection if after decompression *ClientCertificateCompressed.uncompressedLength* does not match the actual length.

B. Authorization Token Compression

The current implementation of DTLS provides supports for payload compression. Once the compression method is negotiated between a client and a server, the application payload (CoAP requests and responses) is compressed regardless of their sizes. However, we propose to compress CoAP request

messages (Conformable and Nonconformable) that contain capability tokens. As shown in Fig. 3, the request message is relatively larger than the response message (acknowledge and reset). Unlike the response message, a CapBAC token is added to the request; therefore, the size of the request payload becomes larger than the response message. The response messages are tiny in sizes, typically in between 32 and 64 bytes. For instance, the response messages of wearable medical sensors contain the physical conditions of a patient, such as temperature, pulse, blood pressure, glucose level, respiration level, and so on. This information can be represented in a few numbers of bytes. Similarly, response messages from smart home appliances contain a small number of bytes. As such, a response message can be sent in a single fragment (or a couple of fragments) and do not require to be compressed. Moreover, the compression of such a small response message may even increase the size of the response because compression headers are added to the response payload.

We propose a set of CoAP options for enabling authorization token compression at the CoAP layer (see Table II). The *Token_Offset* is used to compute the starting address of a token in the request payload. The starting address is calculated as *address of payload marker (FF) + Token_offset*. The *Token_Encoding* indicates the method used to compress the authorization token. *Uncompressed_Length* is the length of the token before compression. The *Compression_Flag* is used to inform the DTLS layer whether or not a CoAP request or response should be compressed. The *Compression_Flag* can have two values: 1) zero or 2) one. The DTLS layer skips payload compression when *Compression_Flag* is set to one. Otherwise, an entire CoAP request or response is compressed in the DTLS layer—steps 1–15 of the conventional approach of Fig. 14.

The operational model of the proposed token compression scheme is shown in Fig. 14 (steps 1–8).

Step 1: An authorization token is first compressed and then attached to the CoAP request payload. The proposed CoAP options (see Table II) are also added after the CoAP headers. The client sets the *Compression_Flag* to one.

Steps 2 and 3: The DTLS layer receives the CoAP request and skips payload compressing as it finds that *Compression_Flag* is set to one. The DTLS layer only encrypts the request and sends the encrypted request to the server.

Step 4: The DTLS layer at the server end decrypts the payload and finds that the value of *Compression_Flag* is set to one; therefore, it skips payload decompression and forwards the request to the CoAP layer.

Step 5: The CoAP layer computes the location of the compressed token in the request payload by using the *Token_offset* option and starting address of the payload marker (FF). Afterward, the server decompresses the token using the compression method specified in *Token_Encoding*. The server discards the request if the value specified in *Uncompressed_Length* does not match the token length after the decompression. The server sends *bad_token* reset message to the client.

Step 6: The server sends a CoAP response (e.g., acknowledge message) to the client. In response, the client sets the *Compression_Flag* to one. Hence, the DTLS layers at the server and client ends do not compress the response (see steps 7 and 8).

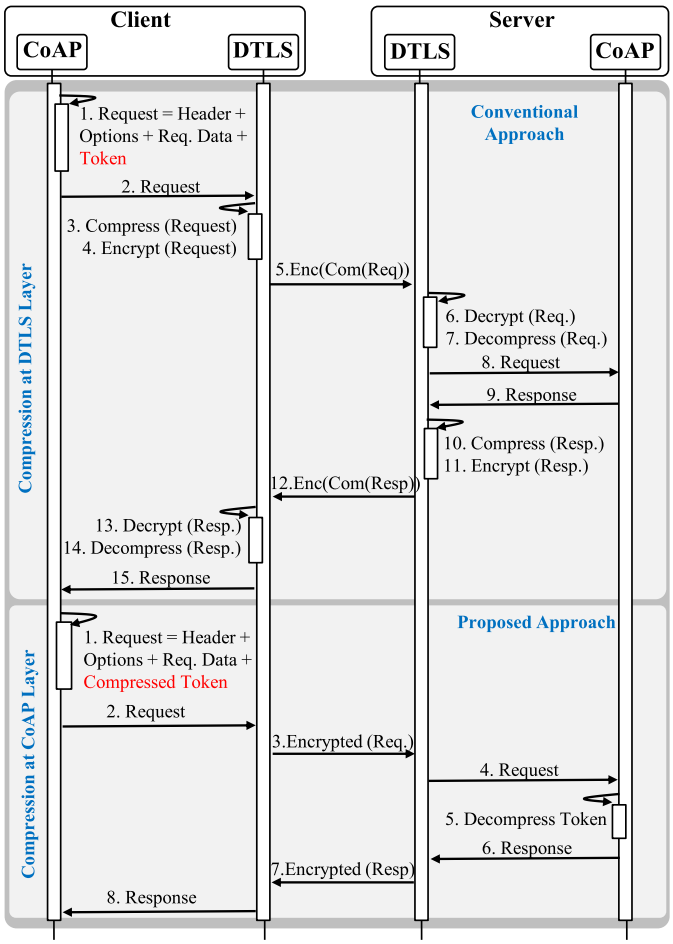

Fig. 14. Operational model of token compression procedure.

C. Relevant Issues

Correlation Between Datagrams: There exists a correlation between datagrams that can be used for compression. Conversely, the proposed framework facilities the reduction of resource overheads for authentication in session establishment and authorization in information exchange. It, therefore, does not need to consider the correlation between datagrams, as the current specification of DTLS supports datagram compression. However, there is no specification in the existing DTLS protocol to compress the credentials exchanged in session establishment and information exchange. The proposed

TABLE II
COAP OPTIONS FOR COMPRESSING AUTHORIZATION TOKENS

Option No.	Name	Format	Values
2048	Token_Offset	uint16	0 - 65535
2049	Token_Encoding	uint8	enum { LZSS (0), (127) } comprsn_method
2050	Uncompressed_Length	uint16	0 - 65535
2051	Compression_Flag	1 bit	0 or 1

scheme fills this gap by proposing a compression-aware authorization framework based on DTLS and CoAP protocols.

CoAP Transfer Modes: In the request phase (see Fig. 3(a)) our framework allows a request and authorization credentials to get delivered to a service device faster as the scheme proposes additional headers in the CoAP, which can be used to compress and reduce the size of access-control credentials. In the response phase (see Fig. 3(b)), a service device responds to a request after a successful authentication and authorization. CoAP transfer methods such as block and observation modes are designed for the response phase, where a service device can send bulk data or periodical updates to a client device. Especially, the block wise transfer (BWT) [52] has been introduced to enable a client/server to send a large message in multiple blocks so that the size of each block is small enough to fit in a single MTU of the IEEE 802.15.4 to avoid fragmentation. However, BWT only works on the application layer and does not deal with the credentials (certificates) used in the DTLS layer, which can still result in fragmentation issue at the 6LoWPAN layer. Moreover, the authorization credentials, such as CapBAC tokens, are considered as the application payload. If the tokens do not fit in the MTU of 6LoWPAN, they will be sent in different blocks in BWT. The CATComp works on the authentication/session establishment phase (DTLS layer) and request phase (authorization over CoAP). Thus, CATComp reduces the communication overheads for exchanging certificates in the DTLS layer and the number of blocks in the application layer by compressing the authorization token. Furthermore, CATComp enables faster authentication by compressing the certificates in the DTLS layer, which eventually results in faster request processing in BWT. Moreover, CATCom improves the performance of the CoAP transfer methods, such as BWT and observations modes, in the request and response phases by using compressed tokens.

V. EXPERIMENT AND EVALUATION

In this section, we provide an analysis of resource efficiency of the proposed approach in terms of fragment saving, storage saving, throughput, communication overhead, and energy cost.

A. Prototype Implementation

We implemented a prototype of CATComp on IoT devices that run the Contiki operating system [27]. We created a 6LoWPAN network using RE-Mote IoT devices [24] and a Weptech [33] border router (see Fig. 15). We integrated CATComp with the TinyDTLS library [25] of Contiki. We also

TABLE III
MEMORY CONSUMPTION TO COMPRESS AND DECOMPRESS

Component	RAM (Bytes)	Flash/ROM (Bytes)
Contiki	11407	41168
COAP	14091	47753
TinyDTLS	14202	47932
Compression	13862	44001

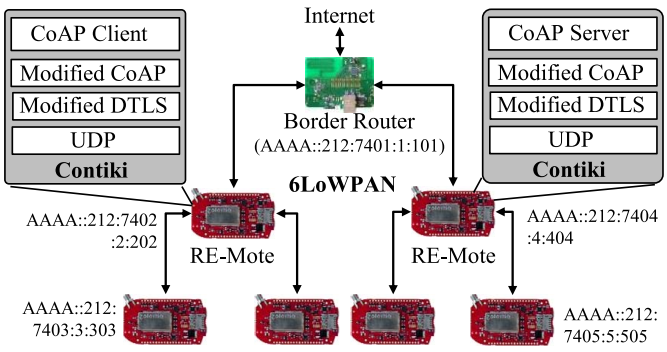

Fig. 15. Experimental setup. Modified CoAP = CoAP + CapBAC Token Compression. Modified DTLS = DTLS + Certificate Compression.

incorporated the proposed CoAP options (see Section IV-B) to the Contiki CoAP library [26]. We used a lossless compression algorithm to compress and decompress certificates and tokens. In this regard, we added the Heatshrink [34] library to Contiki, which implements the LZSS compression algorithm. Note that LZSS does not necessarily have to be the only option. The proposed framework can be integrated with any other lossless compression algorithm, such as prediction by partial matching and run-length encoding. We used LZSS as a proof-of-concept algorithm because it is popular among embedded systems. For instance, the IoT operating system RIOT [35] provides a built-in support for the LZSS compression. Table III presents the memory footprint of our implementation. We used Contiki's memory utility libraries to record the memory usage of the compression methods.

B. Evaluation

Compression Ratio: We computed the compression ratio (cr) for a certificate or a capability token as $cr = (s_u/s_c)$, such that s_u and s_c represent the size of an uncompressed and a compressed certificate or token, respectively. The results of the compression ratio for various sized certificates and token are presented in Fig. 16. From Fig. 16(a), we can observe that the compression ratios for the certificates are approximately 1.34. A certificate includes various fields, such as the issuer's public key, the issuer's details (e.g., name, e-mail, and location), the subject's public key, the subject's details (e.g., country, organization, location, and e-mail address), signature algorithm, and so on. Although the values of the public key fields are random, there are redundancies in the issuer and subject details fields. As such, these low entropy fields contribute to achieve a compression ratio grater than one.

From Fig. 16(a), it can be noted that compression ratios for the variable-sized certificates are almost same (1.38 and 1.3 for RSA and ECC certificates, respectively). Although

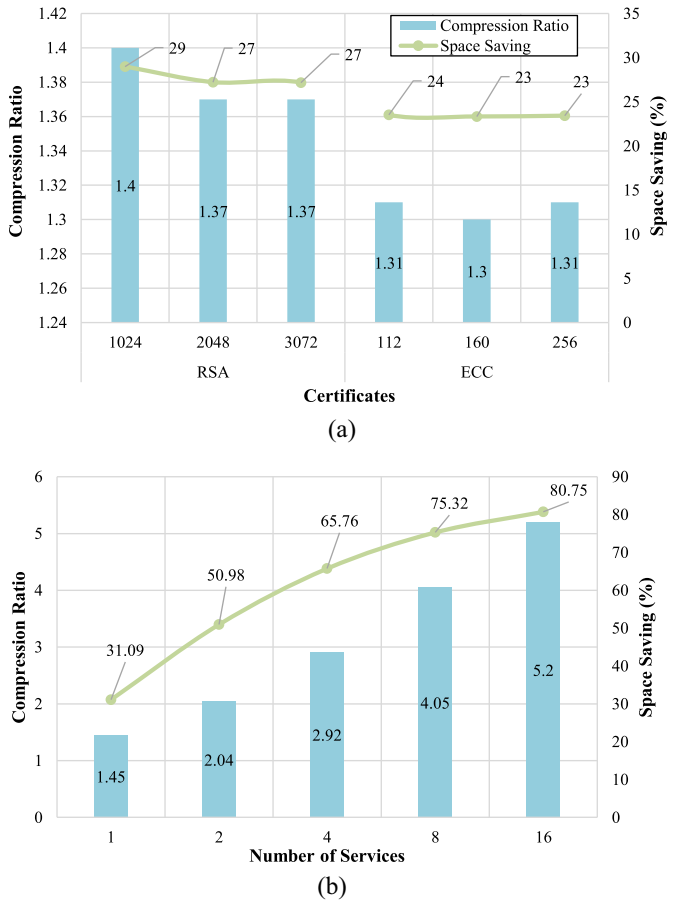

Fig. 16. Analysis of compression ratio and space savings for certificates and tokens. (a) Certificates. (b) Capability tokens.

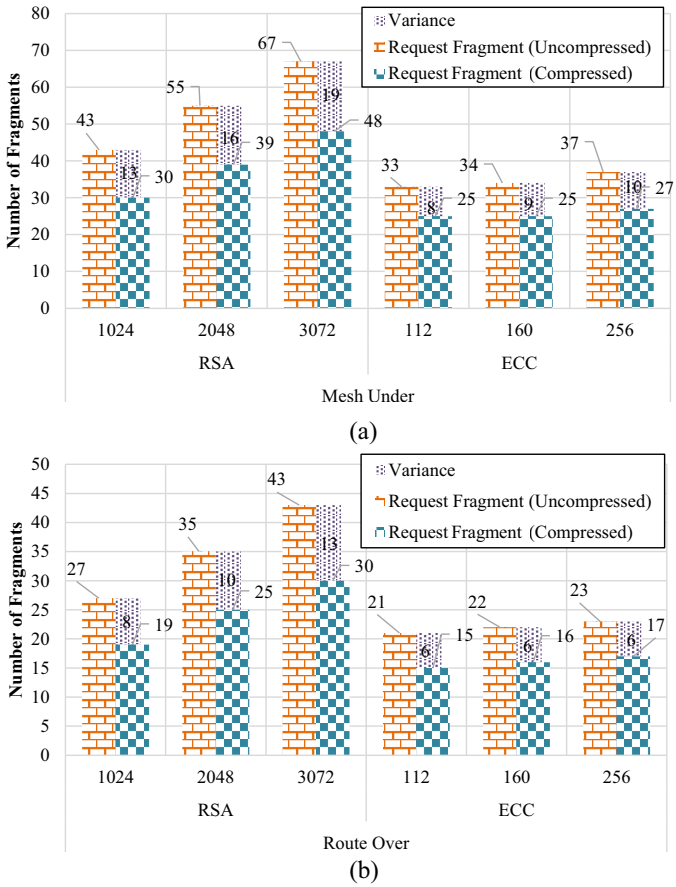

Fig. 17. Fragment savings for certificate compression. (a) Mesh-under routing. (b) Route-over routing.

the variable-sized certificates share information on issuers and subject details, the public keys included in the certificates are generated randomly. Because of the randomness, the public-key fields have high entropy, and the ratios are almost the same. On the other hand, as shown in Fig. 16(b), the compression ratio for the capability tokens increases as the size of the tokens increases. This is because the tokens share information about capability details (see Fig. 4). The JSON fields defining the capability of a client are the same for multiple services. As such, the tokens have low entropy and have better compression ratios than the certificates.

Storage Saving: The storage saving (ss) was computed as $ss = 1-(s_c/s_u)$. As shown in Fig. 16(a), on an average 27.62% and 23% ROM or RAM storage can be saved for X.509 RSA and ECC certificates, respectively, if these are stored in the memory in the compress format. Furthermore, from Fig. 16(b), we can note that the space saving increases in between 31% and 80% with an increase in the size of capability tokens.

Fragment Saving: We calculated the fragment saving (f_s) as the difference between the total number of uncompressed fragments (f_u) and compressed fragments (f_c). As such, the fragment saving is expressed as $f_s = f_u - f_c = (s_u/s_{ap}) - (s_c/s_{ap})$.

Here, s_{ap} denotes the size of an application payload and its values are determined as follows:

$$s_{ap} = \begin{cases} 29 \text{ bytes}, & \text{if Mesh-under Routing} \\ 25 \text{ bytes}, & \text{if Mesh-under} \wedge \text{Link Layer Security} \\ 46 \text{ bytes}, & \text{if Route-over Routing} \\ 42 \text{ bytes}, & \text{if Route-over} \wedge \text{Link Layer Security}. \end{cases}$$

The results of fragment savings are presented in Figs. 17 and 18. From the figures, we can observe that the number of packet fragments is reduced significantly if capability tokens added to the CoAP payload and certificates used in the DTLS handshake are compressed. It can also be noted that the larger the size of a capability token, the higher the fragment savings.

End to End Delay Analysis: The average end-to-end delay (EED_{avg}) was defined as the mean time required for delivering multiple requests under the experiment time. EED_{avg} was calculated as [(Total EED)/(Total Number of Requests Delivered)]. The EED for a request was calculated as the sum of the time to establish a DTLS session as shown in Fig. 13 and the time to send a CoAP request as shown in Fig. 14. For every request, two communicating peers performed DTLS certificate-based authentication, and a capability token was attached to a request. We considered an RSA certificates of key length 1024 bits and size 1237 bytes as well as an ECC certificate of key length 256 bits and size 1050 bytes. Additionally, we

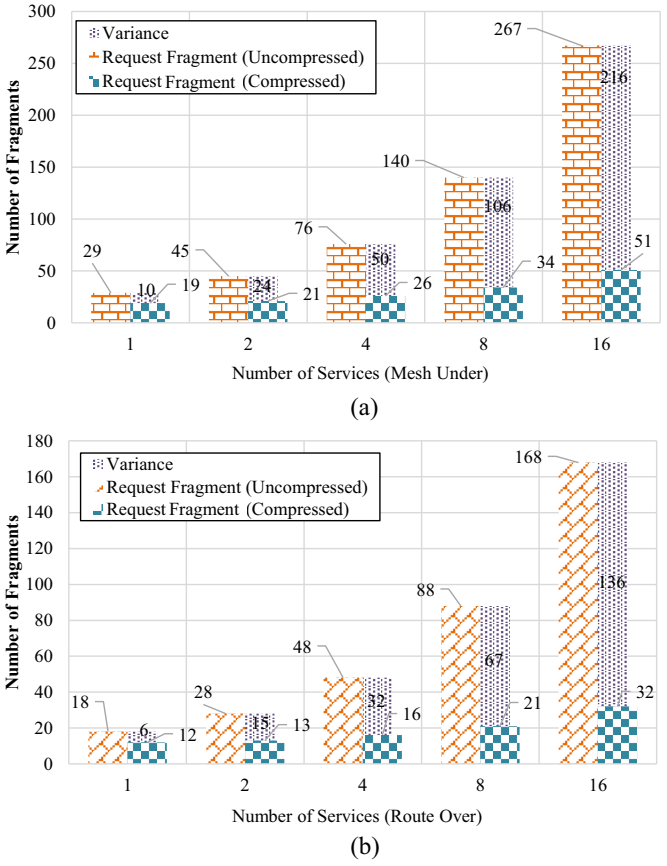

Fig. 18. Fragment savings for token compression. (a) Mesh-under routing. (b) Route-over routing.

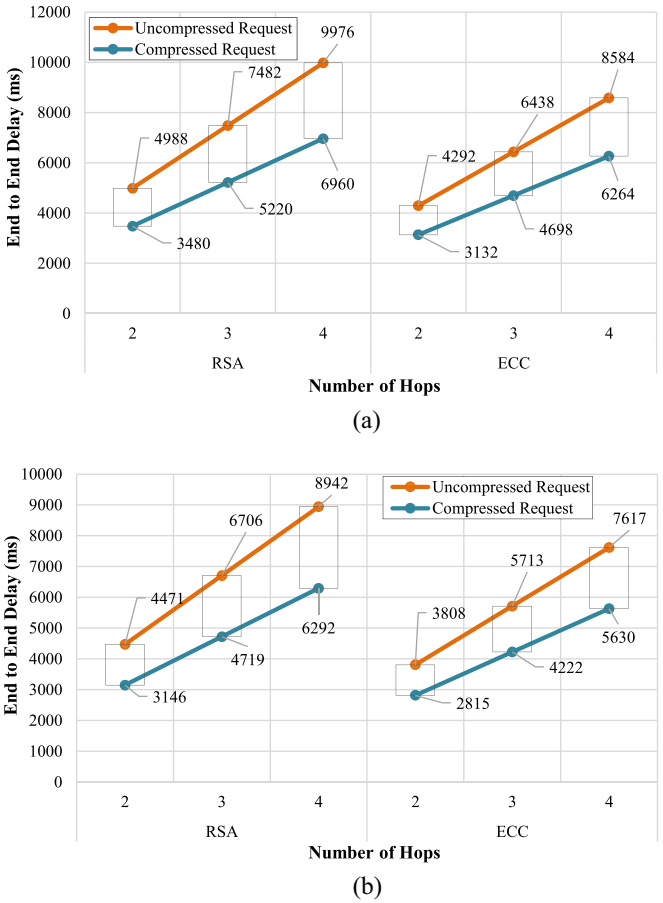

Fig. 19. Analysis of end-to-end delay for certificates. (a) Mesh-under routing. (b) Route-over routing.

considered authorization tokens that contained information on capabilities for two, four, and eight services. We varied the number of intermediate nodes between a sender and a receiver from one to three and recorded the end-to-end delay for variably sized requests. We measured the delays by using the Contiki clock library [36].

The results are presented in Figs. 19 and 20. From the figures, it can be observed that there was a significant reduction in the EED when certificates and tokens were sent in the compressed format. The total number of fragments was reduced due to the compression. As a result, a request was delivered much faster when CATComp was used compared to the conventional approach, although an additional four bytes of CoAP options proposed in Table II were added to a request, and an additional couple of bytes were exchanged through the ClientHello and ServerHello message (steps 1 and 2 of Fig. 13) to negotiate a compression method

Runtime Analysis: We analyzed the CPU time for compression and decompression operations. Table IV shows the execution time required to compress and decompress certificates with variable length keys and authorization token with variable number of service details. It can be noted from Table IV that compression and decompression operations contributed a very small computation time to the EED when CATComp was adopted.

TABLE IV
RUNTIME ANALYSIS

	Certificate			
	ECC-128	**ECC-160**	**RSA-1024**	**RSA-2048**
Compression	28 ms	30 ms	31 ms	36 ms
Decompression	19 ms	21 ms	21 ms	25 ms
	Authorization Token (Number of Services)			
	Service 1	**Service 2**	**Service 4**	**Service 8**
Compression	25 ms	26 ms	28 ms	31 ms
Decompression	17 ms	18 ms	19 ms	22 ms

Throughput Analysis: To calculate the throughput, we divided the total number of served requests by the total time taken for serving the requests. We generated one request every 3 s and recorded the total number of requests served by a receiver node under the experiment time. An analysis and comparison of throughput improvement (t_i) are presented in Fig. 21. The throughput improvement was calculated as $t_i = 1 - (t_c/t_u)$. The terms t_c and t_u represent throughput for compressed and uncompressed requests, respectively. We considered a fixed sized token of 817 bytes that contained capability description for a single service while calculating throughputs for the authorization token. Additionally, we considered an ECC certificates of key length 256 and of size 1050 bytes for computing throughputs for the certificate. From Fig. 21, we can note that on an average, throughputs increased

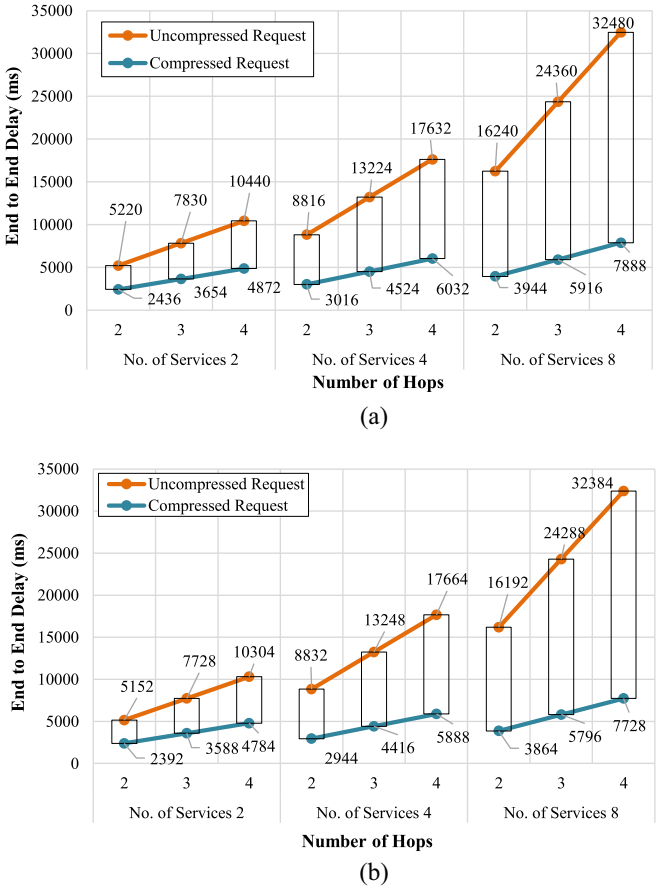

Fig. 20. Analysis of end-to-end delay for tokens. (a) Mesh-under routing. (b) Route-over routing.

by 50% and 39% for RSA and ECC certificates, respectively, when the certificates were compressed. From the figure, it can also be observed that throughputs were increased on an average 187% when we compressed the capability tokens.

Analysis of Communication Energy Consumption: We recorded the amount of energy consumed by the radio transceivers of the communicating devices to set up a DTLS session using the certificate-based authentication and exchange capability tokens over the established session. Furthermore, on the sender node, we recorded the CPU energy consumption for packet fragmentation and fragment transmission. On the receiver node, the CPU energy consumption for receiving and reassembling the fragments was recorded. The total energy consumption associated with communications was computed as the sum of the CPU energy consumption for processing packet fragments and the radio-transceiver energy consumption for sending and receiving the fragments. We used Contiki energy library [37] to measure the amount of energy consumed by the CPU for packet fragmentation and reassembly and by the radio transceiver for sending and receiving fragments.

As shown in Fig. 22, there was a significant drop in the energy consumption when the certificate and authorization token were exchanged in the compressed format. Although additional four bytes of CoAP options (Table II) were added to a request and a couple of additional bytes (steps 1 and 2 of Fig. 13) was exchanged between the peers to negotiate a compression method, the total number of fragments was reduced, as shown in Figs. 17 and 18, with the decrease in the sizes of the certificates and tokens, as presented in Fig. 16. As the communicating devices had to process and deliver a few numbers of fragments in CATComp, there was a drop in the over all energy consumption.

Analysis of Computation Energy Consumption: There was an additional energy consumption by the CPU to compress and decompress the certificates and the tokens. However, the additional energy was very insignificant compared to the energy required to receiving and reassembling the fragments. From Fig. 23, it can be observed that the CPU energy consumption for the compression and decompression operations was only 0.2%–0.5% and 0.3%–0.5% of the total energy consumption, respectively, of the total energy consumption.

VI. Related Works and Comparative Discussion

From the practical viewpoint, the new trends in transport layer security are the uses of DTLS due to its lightweight nature. Leading industries preferred DTLS over `secure sockets layer (SSL)` for better performance without compromising security features [50]. Similarly, CoAP is also gaining popularity over its predecessor `HTTP` in resource-constrained IoT network for the lightweight design. In recent days, integrating DTLS with CoAP has gained the attention of the research community, and many exciting research efforts are found that improve the efficiency of DTLS-CoAP frameworks. Carrillo *et al.* [46] proposed a multihop bootstrapping with intermediaries in IoT networks using the `extensible authentication protocol (EAP)` over the CoAP layer. The authors utilized the `REST` architecture of CoAP to perform the bootstrapping and allegorically show the effectiveness of the scheme in a similar approach to our work. A secure multicast architecture for CoAP is proposed by Park [47]. The study presented a design to secure both group communication and pairwise communication between CoAP clients and servers via exchanging multicast messages. The work introduced a key management scheme that can replace `DTLS handshake` with fewer number of messages and offers better performance. In [48], the IoT resource caching to a broker in the application layer was discussed. Therein, the authors proposed a traffic load balancing mechanism among the brokers in a CoAP-based resource-constrained IoT network to reduce the energy consumption in IoT servers. The method demonstrated that not every IoT resource is suitable for caching to gain energy efficiency. Moreover, the work introduced an approach to find and monitor the popular IoT resources, which are suitable for caching. The research also showed that obtaining the optimal load balancing among the brokers in a resource-constrained network is `NP-hard` and therefore, introduced an approximation algorithm. The evaluation through simulated experiments showed that the approximation algorithm performs better than the prior options in terms of average resource utilization and delay. Roselin *et al.* [50] analyzed the remote CoAP server access support and found out that the inherent CoAP implementation

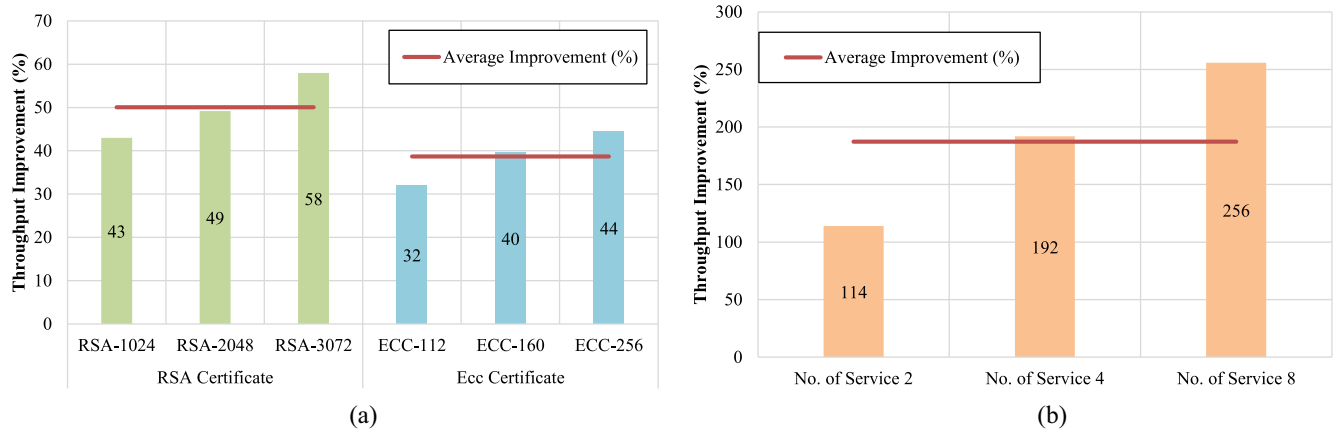

Fig. 21. Throughput improvement for compressed credentials. (a) Compressed certificates. (b) Compressed tokens.

is vulnerable to a potential off-path attack. The work showed the feasibility of the attack using a setup with real devices and proposes machine learning-based solutions to mitigate the attacks. Similar to our work, all of the aforesaid researches focus on improving the operations of DTLS-CoAP frameworks. Depending on the nature of the problems of interests, these works presented appropriate analysis illustrating the feasibility and efficiency of the work using either mathematical models or experiment-based procedural frameworks like our work.

The existing research works related to compression predominantly focused on the compression of packet headers at different layers of the protocol stack. Several prior research works proposed schemes for the compression of packet headers, such as record header, handshake header, or authorization header. The X.509 certificates and the authorization tokens have not been considered for compression in previous research works.

In the context of IoT, some compression mechanisms have been proposed, which are protocol specific. Raza *et al.* [38] proposed 6LoWPAN header compression for DTLS. They introduced compression of DTLS record and handshake header. In another work [39], the compression of IPsec's authentication header(AH) was proposed in the 6LoWPAN stack. Raza *et al.* [40] also presented an integration of DTLS and CoAP for IoT, which includes a scheme for DTLS header compression. In comparison with our work, these works significantly focused on DTLS header compression. On the other hand, we proposed a scheme for the payload compression.

Hummen *et al.* [41] proposed Slimfit, a compression layer for HIP DEX. HIP DEX is an end-to-end security protocol designed for constrained network environments in the IP-based IoT. This layer works just under the HIP DEX layer. The Slimfit layer compresses expendable HIP DEX protocol information. Mainly, the HIP header was the central focus. As we can see, this scheme does not consider payload for compression where our proposed module significantly addresses the compression of payload.

The robust header compression (ROHC) framework [42] addressed protocol-specific compression profiles. It provided an efficient and flexible header compression concept. Also,

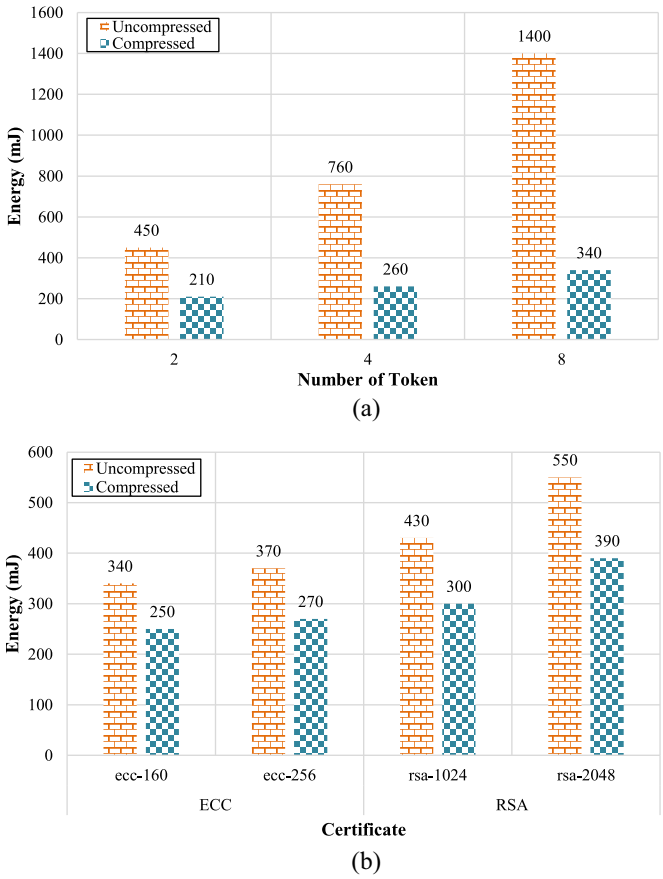

Fig. 22. Communication energy consumption. Communication energy consumption for (a) authorization tokens and (b) certificates.

there is proposed standardization for 6LoWPAN generic compression of headers and header-like payloads [51]. Both of these schemes made suggestion about a header compression. Although 6LoWPAN GHC presented techniques for header-like payload but these are not CoAP payload. Granja *et al.* [44] proposed and evaluated the usage of compressed security headers for the network layer with IoT. Granjal *et al.* [45] also observed that payload space scarcity would be problematic

TABLE V
COMPARATIVE ANALYSIS WITH THE PRIOR SURVEY WORKS. RD = REAL DEVICE, SS = STORAGE SAVING, FR = FRAGMENTATION REDUCTION, ER = END-TO-END DELAY REDUCTION, AND TI = THROUGHPUT IMPROVEMENT

Properties / Schemes	Compression at DTLS		Compression at CoAP		Evaluation				
	Header/Payload	Certificate	Header/Payload	Authorization Token	RD	SS	FR	ER	TI
Raza(2012) [39]	✓	✗	✗	✗	✗	✗	✗	✗	✗
Raza(2011) [40]	✓	✗	✗	✗	✗	✗	✗	✗	✗
Lithe et al. [41]	✓	✗	✓	✗	✗	✗	✗	✗	✗
Hummen et al. [42]	✓	✗	✗	✗	✓	✗	✓	✓	✓
ROHC [43]	✓	✗	✗	✗	✗	✗	✗	✗	✗
Bormann et al. [44]	✓	✗	✓	✗	✗	✗	✗	✗	✗
Granja (2010) [45]	✓	✓	✗	✗	✓	✓	✗	✗	✗
Granja (2015) [46]	✓	✓	✗	✗	✗	✗	✗	✗	✗
Carrillo et at. [47]	✗	✗	✗	✗	✗	✗	✗	✓	✓
Park et al. [48]	✗	✗	✗	✗	✓	✗	✓	✓	✓
Sun et al. [49]	✗	✗	✗	✗	✗	✗	✗	✗	✓
Roselin et al. [50]	✗	✗	✗	✗	✓	✗	✗	✗	✗
CATComp (Proposed)	✓	✓	✓	✓	✓	✓	✓	✓	✓

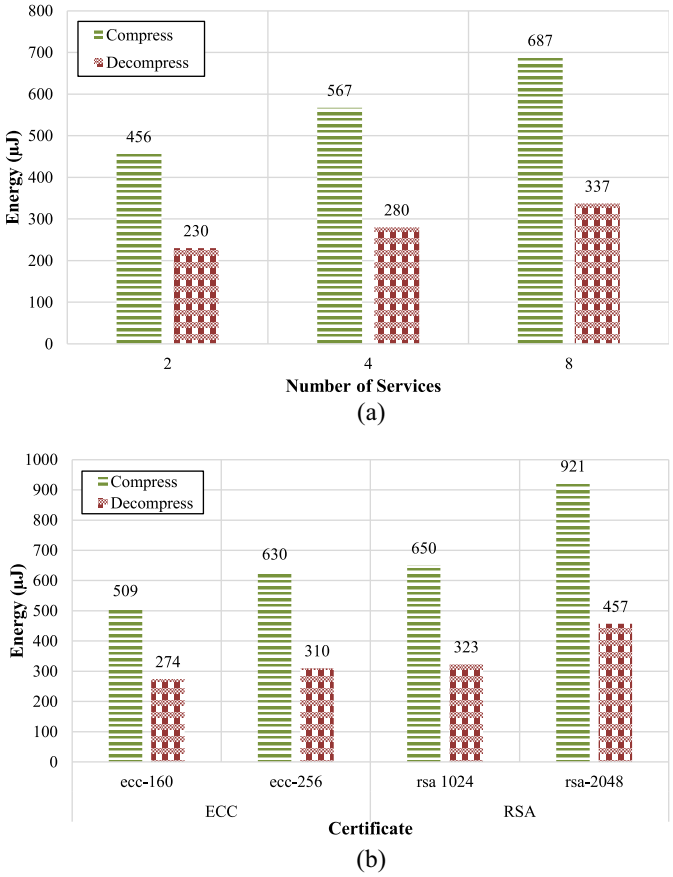

Fig. 23. Computation energy consumption. CPU energy consumption to compress and decompress the (a) tokens and (b) certificates.

with the application which requires larger payloads. They suggested employing security at other layers such as compressed IPSec.

From our analysis, we can observe that existing research work predominantly focused on the compression of packet headers at different layers of the protocol stack. It is quite comprehensible that in the early stage of the communication packet, headers consume a significant portion of the total packet sizes. However, as the communication goes on, the payload becomes the main portion of a packet and compression of the payload can significantly improve end-to-end delay and communication overheads. Our proposed model centers around compression of the certificates and authorization tokens. We summarize the comparative analysis of the proposed model with the prior related works in Table V.

VII. CONCLUDING REMARKS

In this article, we proposed CATComp—a compression-aware protocol that enables IoT devices to exchange compressed X.509 certificates and authorization tokens. In 6LoWPAN networks, often, certificates and authorization tokens contribute a significant portion in a communication packet. Therefore, the number of packet fragments varies with the size of these certificates and authorization tokens. CATComp enables communicating devices to compress X.509 certificates and authorization tokens at the DTLS and CoAP layers before sending them over the low-powered and lossy networks. Thus, CATComp enables devices to minimize the number of packet fragments significantly. We implemented a prototype of CATComp on Contiki-enabled RE-Mote IoT devices and provided a performance analysis of CATComp in terms of communication and energy efficiency. The experimental results showed that sizes of the DTLS and CoAP payloads were reduced significantly by compressing certificates and authorization tokens. The smaller sized payloads resulted in decreasing the number of packet fragments, which yielded less communication overhead and energy consumption for fragment processing. As such, devices could exchange messages faster and experience longer battery life.

REFERENCES

[1] R. Petrolo, V. Loscri, and N. Mitton, "Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms," *Trans. Emerg. Telecommun. Technol.*, vol. 28, no. 1, 2017, Art. no. e2931.

[2] A. S. Deese *et al.*, "Long-term monitoring of smart city assets via Internet of Things and low-power wide-area networks," *IEEE Internet Things J.*, vol. 8, no. 1, pp. 222–231, Jan. 2021.

[3] Y. Meng, W. Zhang, H. Zhu, and X. S. Shen, "Securing consumer IoT in the smart home: Architecture, challenges, and countermeasures," *IEEE Wireless Commun.*, vol. 25, no. 6, pp. 53–59, Dec. 2018.

[4] M. Hossain, S. M. R. Islam, F. Ali, K.-S. Kwak, and R. Hasan, "An Internet of Things-based health prescription assistant and its security system design," *Future Gener. Comput. Syst.*, vol. 82, pp. 422–439, May 2018.

[5] F. Zhu, Y. Lv, Y. Chen, X. Wang, G. Xiong, and F.-Y. Wang, "Parallel transportation systems: Toward IoT-enabled smart urban traffic control and management," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 10, pp. 4063–4071, Oct. 2020.

[6] H. Sedjelmaci, M. Hadji, and N. Ansari, "Cyber security game for intelligent transportation systems," *IEEE Netw.*, vol. 33, no. 4, pp. 216–222, Jul./Aug. 2019.

[7] O. Elijah, T. A. Rahman, I. Orikumhi, C. Y. Leow, and M. N. Hindia, "An overview of Internet of Things (IoT) and data analytics in agriculture: Benefits and challenges," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3758–3773, Oct. 2018.

[8] Forbes. (2017). *Roundup of Internet of Things Forecasts*. [Online]. Available: https://goo.gl/iVf5uz

[9] R. Hummen, J. Hiller, H. Wirtz, M. Henze, H. Shafagh, and K. Wehrle, "6LoWPAN fragmentation attacks and mitigation mechanisms," in *Proc. 6th ACM Conf. Security Privacy Wireless Mobile Netw.*, 2013, pp. 55–66.

[10] Y. Luo and L. Pu, "Practical issues of RF energy harvest and data transmission in renewable radio energy powered IoT," *IEEE Trans. Sustain. Comput.*, early access, Jun. 4, 2020, doi: 10.1109/TSUSC.2020.3000085.

[11] M. Hossain and R. Hasan, "P-HIP: A lightweight and privacy-aware host identity protocol for Internet of Things," *IEEE Internet Things J.*, vol. 8, no. 1, pp. 555–571, Jan. 2021.

[12] T. Winter, "RPL: IPv6 routing protocol for low-power and lossy networks," Internet Eng. Task Force, Fremont, CA, USA, RFC 6550, 2012.

[13] A. M. Efendi, A. F. P. Negara, O. S. Kyo, and D. Choi, "A design of 6LoWPAN routing protocol border router with multi-uplink interface: Ethernet and Wi-Fi," *Adv. Sci. Lett.*, vol. 20, no. 1, pp. 56–60, 2014.

[14] N. Kushalnagar, G. Montenegro, and C. Schumacher, "IPv6 over low-power wireless personal area networks (6LoWPANs): Overview, assumptions, problem statement, and goals," Internet Eng. Task Force, Fremont, CA, USA, RFC 4919, 2007.

[15] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 packets over IEEE 802.15.4 networks," Internet Eng. Task Force, Fremont, CA, USA, RFC 4944, 2007.

[16] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (CoAP)," Internet Eng. Task Force, Fremont, CA, USA, RFC 7959, 2016.

[17] M. Masirap, M. H. Amaran, Y. M. Yussoff, R. Ab Rahman, and H. Hashim, "Evaluation of reliable UDP-based transport protocols for Internet of Things (IoT)," in *Proc. IEEE Symp. Comput. Appl. Ind. Electron. (ISCAIE)*, 2016, pp. 200–205.

[18] G. A. Akpakwu, G. P. Hancke, and A. M. Abu-Mahfouz, "CACC: Context-aware congestion control approach for lightweight COAP/UDP-based Internet of Things traffic," *Trans. Emerg. Telecommun. Technol.*, vol. 31, no. 2, 2020, Art. no. e3822.

[19] E. Rescorla and N. Modadugu, "Datagram transport layer security version 1.2," Internet Eng. Task Force, Fremont, CA, USA, RFC 6347, 2012.

[20] S. Gusmeroli, S. Piccione, and D. Rotondi, "A capability-based security approach to manage access control in the Internet of Things," *Math. Comput. Model.*, vol. 58, no. 5, pp. 1189–1205, 2013.

[21] J. L. Hernandez-Ramos, A. J. Jara, L. Marın, and A. F. Skarmeta, "Distributed capability-based access control for the Internet of Things," *J. Internet Services Inf. Security (JISIS)*, vol. 3, nos. 3–4, pp. 1–16, 2013.

[22] H. Kim, "Protection against packet fragmentation attacks at 6LoWPAN adaptation layer," in *Proc. Int. Conf. Converg. Hybrid Inf. Technol.*, 2008, pp. 796–801.

[23] M. Hossain, Y. Karim, and R. Hasan, "SecuPAN: A security scheme to mitigate fragmentation-based network attacks in 6LoWPAN," in *Proc. CODASPY*, 2018, pp. 307–318. [Online]. Available: https://doi.org/10.1145/3176258.3176326

[24] Re-Mote. (2017). *Z1 6LoWPAN IoT Device*. [Online]. Available: http://zolertia.io/z1

[25] O. Bergmann. (Feb. 15, 2013). *Tinydtls*. [Online]. Available: http://tinydtls.sourceforge.net/Visited

[26] Contiki-CoAP. (2017). *Contiki CoAP Library*. [Online]. Available: https://github.com/contiki-os/contiki/tree/master/apps/er-coap

[27] Contiki. (2016). *Contiki OS: An Open Source Operating System for the Internet of Things*. [Online]. Available: http://www.contiki-os.org/

[28] R. Hummen, J. H. Ziegeldorf, H. Shafagh, S. Raza, and K. Wehrle, "Towards viable certificate-based authentication for the Internet of Things," in *Proc. 2nd ACM Workshop Hot Topics Wireless Netw. Security Privacy*, 2013, pp. 37–42.

[29] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, and G. Carle, "DTLS based security and two-way authentication for the Internet of Things," *Ad Hoc Netw.*, vol. 11, no. 8, pp. 2710–2723, 2013.

[30] A. H. Chowdhury *et al.*, "Route-over vs mesh-under routing in 6LoWPAN," in *Proc. Int. Conf. Wireless Commun. Mobile Comput. Connect. World Wirelessly*, 2009, pp. 1208–1212.

[31] J. Arkko, V. Devarapalli, and F. Dupont, "Using IPsec to protect mobile IPv6 signaling between mobile nodes and home agents," Internet Eng. Task Force, Fremont, CA, USA, RFC 3776, 2004.

[32] S. Praptodiyono, M. I. Santoso, T. Firmansyah, A. Abdurrazaq, I. H. Hasbullah, and A. Osman, "Enhancing IPsec performance in mobile IPv6 using elliptic curve cryptography," in *Proc. 6th Int. Conf. Electr. Eng. Comput. Sci. Inform. (EECSI)*, 2019, pp. 186–191.

[33] Weptech. (2017). *6LOWPAN IoT Gateway* [Online]. Available: https://www.weptech.de/en/6lowpan/gateway-saker.html

[34] Heatshrink. (2017). *An Implementation of the LZSS Compression Method*. [Online]. Available: https://github.com/atomicobject/heatshrink

[35] RIOT. *Lightweight Compression Library*. Accessed: Oct. 8, 2020. [Online]. Available: http://doc.riot-os.org/group_pkg_heatshrink.html#details

[36] Contiki. *Contiki Clock Library*. Accessed: May 8, 2017. [Online]. Available: http://www.eistec.se/docs/contiki/a02184.html

[37] Contiki. (2017). *Contiki APIS for Measuring Energy Consumption*. [Online]. Available: http://contiki.sourceforge.net/docs/2.6/a00452_source.html

[38] S. Raza, D. Trabalza, and T. Voigt, "6LoWPAN compressed DTLS for CoAP," in *Proc. IEEE 8th Int. Conf. Distrib. Comput. Sens. Syst. (DCOSS)*, 2012, pp. 287–289.

[39] S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig, "Securing communication in 6LoWPAN with compressed IPsec," in *Proc. Int. Conf. Distrib. Comput. Sens. Syst. Workshops (DCOSS)*, 2011, pp. 1–8.

[40] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt, "Lithe: Lightweight secure CoAP for the Internet of Things," *IEEE Sensors J.*, vol. 13, no. 10, pp. 3711–3720, Oct. 2013.

[41] R. Hummen, J. Hiller, M. Henze, and K. Wehrle, "Slimfit—A HIP DEX compression layer for the IP-based Internet of Things," in *Proc. 9th Int. Conf. Wireless Mobile Comput. Netw. Commun. (WiMob)*, 2013, pp. 259–266.

[42] C. Bormann *et al.*, "Robust header compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed," Internet Eng. Task Force, Fremont, CA, USA, Rep. RFC 3095, 2001.

[43] C. Bormann, "Guidance for light-weight implementations of the Internet protocol suite," Internet Eng. Task Force, Fremont, CA, USA, Internet-Draft draft-bormann-lwig-guidance-01, 2013.

[44] J. Granjal, E. Monteiro, and J. S. Silva, "Enabling network-layer security on IPv6 wireless sensor networks," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, 2010, pp. 1–6.

[45] J. Granjal, E. Monteiro, and J. S. Silva, "Security for the Internet of Things: A survey of existing protocols and open research issues," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1294–1312, 3rd Quart., 2015.

[46] D. Garcia-Carrillo and R. Marin-Lopez, "Multihop bootstrapping with EAP through CoAP intermediaries for IoT," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 4003–4017, Oct. 2018.

[47] C.-S. Park, "Security architecture for secure multicast CoAP applications," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 3441–3452, Apr. 2020.

[48] X. Sun and N. Ansari, "Traffic load balancing among brokers at the IoT application layer," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 1, pp. 489–502, Mar. 2018.

[49] A. G. Roselin, P. Nanda, S. Nepal, X. He, and J. Wright, "Exploiting the remote server access support of CoAP protocol," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 9338–9349, Dec. 2019.

[50] Fortinet. (2020). *Technical Note: Using DTLS to Improve SSL VPN Performance*. [Online]. Available: https://kb.fortinet.com/kb/documentLink.do?externalID=FD38162

[51] C. Bormann, "6LoWPAN generic compression of headers and header-like payloads," Internet Eng. Task Force, Fremont, CA, USA, RFC 7400, 2013.

[52] C. Bormann and Z. Shelby, "Block-wise transfersin the constrained application protocol (CoAP)," Rep. RFC 7959, Aug. 2016. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7959