Received 9 January 2024, accepted 18 January 2024, date of publication 25 January 2024, date of current version 1 February 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3358678

RESEARCH ARTICLE

Design, Implementation, and Evaluation of an Embedded CoAP Proxy Server for 6LoWPAN

ISMAEL AMEZCUA VALDOVINOS[1], **PATRICIA ELIZABETH FIGUEROA MILLÁN**[2], (Member, IEEE), **JUAN ANTONIO GUERRERO-IBÁÑEZ**[1], AND **RAMONA EVELIA CHÁVEZ VALDEZ**[2]

[1]Facultad de Telemática, Universidad de Colima, Colima 28040, Mexico
[2]Tecnológico Nacional de México/I.T Colima, Colima 28976, Mexico

Corresponding author: Patricia Elizabeth Figueroa Millán (patricia.figueroa@colima.tecnm.mx)

This work was supported in part by Programa FRIDA LACNIC (Registro de Direcciones de Internet para América Latina y el Caribe) under the Project Entitled "Implementación de un proxy CoAP en el sistema operativo Contiki-NG para redes 6LoWPAN."

ABSTRACT Proxy servers are widely used in many contexts since they can enhance performance, security, and access control for production networks. The Constrained Application Protocol (CoAP), the de facto standard of Internet of Things (IoT) communications, specifies mechanisms and semantics to applications based on RESTful approach similar to HTTP. The CoAP standard defines an additional operation mode for proxying request, which is supported by several programming languages and platforms. However, there is no current implementation of a CoAP proxy for embedded systems, namely for devices supporting the Contiki-NG operating system. This paper discusses the design, implementation, and evaluation of a forward CoAP Proxy server for 6LoWPAN embedded systems with cache capabilities. We perform simulation and experimental evaluations with topologies involving up to three hops from the proxy to study performance in terms of response times and the number of exchanged packets. In simulation environments results show a 48.03%, 85.39%, and 134.21% in response time reduction at one, two, and three hops away. In experimental environments, results show that the use of our embedded CoAP Proxy server reduces response times in 7.46%, 30.67%, and 37.43% when requests are targeted to servers at one, two, and three hops away from the proxy respectively. In both scenarios, a reduction in the number of exchanged packets of 71.42%, 125%, and 166.66% for requests at each hop is achieved.

INDEX TERMS Cache management, CoAP, embedded proxy, forward proxy, 6LoWPAN.

I. INTRODUCTION

The IoT is considered an important technological paradigm in this last century because of its influence on the shape of the future world. It enables an interconnection of physical devices to the Internet in order to collect, process, and exchange data in a semi-autonomous or fully autonomous manner. Consequently, it also enables many applications for three major fields: society, industry and the environment. For society, IoT applications are focused on health care [1], [2], entertainment [3], building automation [4], [5], transportation systems [6], [7], and security and surveillance [8], [9]. Meanwhile, in the industry field these applications are oriented towards supply chain management [10], [11], aerospace [12], aviation [13], transportation and logistics [14], smart metering [15], [16], warehouse and storage [17]. Lastly, environmental applications are being deployed to achieve smart and precision agriculture [18], [19], [20], natural disaster management [21], pollution control [22], [23], and smart power plants [24].

Wireless sensor networks (WSN) are a key technology in IoT because they provide the basis to sense and collect data from their environment, and wirelessly transmit data to a central location for analysis and processing. Moreover, WSN not only collect data but also respond to changes in the environment by performing actions based on that data,

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Quan.

enabling a closed-loop system where sensor gather data to be analyzed and actuators respond to the analyzed data.

WSNs primarily use IEEE 802.15.4 technology to enable communication between devices in low-rate wireless personal area networks. The IEEE 802.15.4 standard defines the specifications for the physical layer (PHY) and medium access control (MAC) sub-layers to enable low-data-rate wireless communications for fixed or mobile devices with limited energy consumption [25]. For higher layer functions, the Internet Engineering Task Force (IETF) completed two recommendations for the use of IPv6 technology in conjunction with IEEE 802.15.4 radios, namely RFC 4944: Transmission of IPv6 Packets over IEEE 802.15.4 Networks [26] and RFC 6282: Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks (6LoWPAN) [27]. 6LoWPAN defines an adaptation layer with mechanisms for header compression, packet fragmentation, and packet reassembly for networks based on the IEEE 802.15.4 standard. The 6LoWPAN adaptation layer uses three main elements: a header compression mechanism, packet fragmentation and reassembly, and layer-two forwarding. Header compression allows fitting IPv6 datagrams into the 127-byte long IEEE 802.15.4 frame. Whenever large data needs to be segmented into multiple datagrams, packet fragmentation and reassembly are employed. Lastly, layer-two forwarding deals with link-layer and IEEE 802.15.4 header fields to reach the destination address [28].

6LoWPAN is already implemented in several open-source operating systems designed for resource constrained devices to realize the IoT. Contiki-NG is one of such operating systems, which is in compliance with RFC 4944 for enabling connectivity between such constrained devices with 6LoWPAN.

The Constrained Application Protocol (CoAP) is a specialized transfer protocol for resource constrained devices. CoAP is a protocol that sits on top of 6LoWPAN and is based on a similar request/response interaction model as HTTP. The implementation of CoAP [29] in Contiki-NG is based on Erbium. Although the current implementation supports most basic CoAP functions such as options for `Max-Age`, E-Tags, block-wise transfers and observe functionality, it does not support the CoAP proxy mode for its use inside constrained environments, specially inside a border router or in an embedded proxy server.

As a consequence and accordingly with Washizaki et al. [30], 57% of the design patterns and architectures used in IoT systems are not domain-specific, which suggests that most systems are designed using conventional architectures. Due to the vast number and variety of problems IoT systems can solve, their architectures tend to be domain-specific. Some of the problems that such systems can face when using conventional architecture are the difficulty to support device heterogeneity present in most IoT deployments, the low coupling of the different communication technologies, and the support to use technology-bound application protocols. Layer-based architectures with gateways allow inclusion and support of heterogeneity embedded in IoT systems. The latter facilitates the deployment and allows for the design of highly scalable generic systems. The use of network proxies in layer-based architectures play an important role since they can provide adaptation layers to support specific protocols and communication technologies.

As a result, this paper describes the design, implementation, and evaluation of an open-source embedded forward CoAP proxy server with cache management capabilities for the Contiki-NG operating system. Proxies provide an additional layer that can be used to improve security, enable scalability, increase performance, or support heterogeneous protocols. Having a plethora of IoT applications deployed in a variety of fields, the use of CoAP proxies contributes to the reduction of complexity of architectural design. Furthermore, to the best of our knowledge, there is no current implementation of a CoAP proxy for embedded operating systems such as Contiki-NG. Having an embedded implementation of a CoAP proxy allows to addition of such functionality into border routers in the 6LoWPAN. This also reduces the cost and number of devices inside the network. Moreover, we also prove the feasibility of embedded CoAP proxies and demonstrate their effectiveness by evaluating performance in both experimental and simulated environments with different scenarios. Finally, we describe and evaluate a CoAP cache mechanism that enhances performance by reducing response times and the number of exchanged packets.

The rest of the paper is organized as follows: Section II depicts the current state of CoAP proxy implementations for protocol translation, cache management, group message notification optimization, among others. Prior implementations were designed for full-feature operating systems. Section III describes different types of proxies and how CoAP forwarding proxies can be used to enable fast cache responses. In Section IV we present the design and implementation details of our embedded CoAP Proxy server. Section V describes the characteristics and reasoning of our simulated and experimental setups deployed to evaluate the performance of our embedded proxy. And lastly, Section VI discusses our findings and Section VII draws our conclusions, recommendations and future work.

II. RELATED WORK

There are a vast number of scenarios where CoAP proxy servers are found to be useful: from reducing the number of interactions by disseminating messages between CoAP servers and clients to providing frameworks for secure network bootstrapping, name resolution, and mobility management. This section provides a summarized view on different proposals and use cases of CoAP proxy servers, introducing first common implementations and later revising implementations providing support for other technologies.

A common usage of CoAP proxies is to translate between protocols, namely between HTTP and CoAP due to their similarity in syntax and semantics. Sulaeman et al. [31] designed and implemented a cross-protocol reverse proxy with cache support. However, cache is limited to store a single integer value. Castellani et al. [32] define and evaluate different mappings for enhancing HTTP to CoAP support in cross proxies. Esquiagola et al. [33] also propose and evaluate an interception cross proxy focusing on interoperability that uses a broker architecture to translate HTTP to CoAP requests. Mi and Wei [34] provide an architecture for using smartphones as CoAP translation proxies in health care environments as a mean of interaction between the patient's physical condition, remote clinicians, and hospitals. Ludovici and Calveras [35] also propose the design of a cross-proxy CoAP server for the interaction between a CoAP-based sensor network and HTTP-based web applications. Authors also explore observe communication pattern between the server and the CoAP devices to reduce the number of interactions. Mingozzi et al. in [36] propose a container-based proxy hypervisor to interconnect the Web with a CoAP network by translating HTTP requests to CoAP. Authors in these papers do not consider the impact of a multi-hop architecture when measuring performance of their proxies.

A common approach to data dissemination in IP networks is the use of multicast. However, current technologies such as IEEE 802.15.4 and LoRaWAN typically do not provide link-layer support for such mechanisms. There are efforts to extend CoAP to allow one-to-many communication using multicast IP address endpoints on the application layer [37], [38]. However, responses are always carried over unicast, which may introduce high loads to constrained links and reduce the lifetime of battery-powered nodes as well. Gündoğan et al. [39], [40] propose a data-centric approach to aggregate content request and replicate responses by extending existing CoAP components. Individual hops maintain an on-path cache storage to be used for matching requests along the path.

Lai et al. [41] define a group-based message management (GMM) framework that uses a proxy as a means to regulate transmission notifications. A cache mechanism is used to preserve the data as long as the `Max-Age` parameter on the observed data is not fresh. The GMM uses a scheduling module to optimally minimize the number of interactions by setting an observation period selection [42].

Mišic and Mišic [43] investigated the multicast CoAP extension [29] for domain-based data dissemination and its use in CoAP proxies with cache capabilities to ensure data freshness. Authors also developed an analytical model to study the impact of different parameters for cache management and maintenance. The experimental scenario only considers a single hop cluster. Results show that by adjusting the leisure period, which is a time duration that a CoAP server is willing to spend before sending a response, small latency is achieved at the expense of energy consumption. Experimental and simulated results show that content aggregation and notification optimization can reduce response latency whilst increasing success ratios.

In addition to common CoAP proxy implementations described before, proxies can also be considered to further extend or support the integration with other technologies. Banaie et al. [44] propose the use of an intermediary CoAP proxy server between constrained and unconstrained networks to manage client queries and support requests with different levels of Quality of Service (QoS). An analytical model is defined to determine the appropriate data caching interval for reducing response times, optimizing bandwidth utilization and energy consumption. Garcia-Carrillo and Marin-Lopez [45] propose the use of a CoAP proxy server to aid the bootstrapping process of authentication nodes in a secure network using the Extended Authentication Protocol (EAP). Lenders et al. [46] propose a CoAP proxy for secure and privacy-friendly name resolution of constrained nodes. The DNS over CoAP (DoC) involves a server acting as a CoAP forward proxy with cache capabilities. Authors state that name resolution performance is dependent on packet sizes and that a more feasible method for DoC scenarios is FETCH since it can obtain information specified by a number of parameters (much like the SEARCH method from HTTP) [47]. Choi and Koh [48] investigate mobility scenarios for CoAP deployments. The authors propose the placement of a CoAP proxy server for mobility management based on Proxy Mobile IPv6 (PMIPv6).

III. CoAP PROXIES

The CoAP standard defines three modes for proxy operations: reverse, cross and forward proxies [29]. In a reverse proxy, the proxy offers server resources as if they were their own. Different from forward proxies, in a reverse mode the CoAP client is not aware that the resources are placed behind a proxy and thus the client does not need to specify the proxy URI in the request. In a cross proxy, the client makes a request to the proxy with a different protocol, such as HTTP, and the latter translates, if possible, the request to a CoAP message. A cross proxy can also be used as a bridge between two different physical technologies whilst maintaining the same CoAP message. In a forward proxy, a CoAP client explicitly indicates the use of a CoAP Proxy by using the `Proxy-Uri` option defined in CoAP. In this operation mode, the CoAP proxy server will communicate on the CoAP client's behalf with the CoAP server. A CoAP proxy can also be implemented with cache capabilities to reduce the number of messages by responding to a client request with a prior message that satisfies the current request.

One of the main reasons to use a CoAP proxy is to reduce the number of messages exchanged between CoAP clients and servers. This can be achieved by providing cache entries inside the proxy server for rapid response in case a CoAP request has a cache hit with fresh data. The RFC7252 defines a fresh response when the data is within a `Max-Age` value indicated by the CoAP server in its response. The default `Max-Age` value is 60 seconds, but the server can specify any number of seconds depending on the type of sensor and sampling rate. For example, an atmospheric pressure sensor can use longer `Max-Age` values since its information rarely changes during the day. On the other hand, a temperature sensor can use shorter `Max-Age` values since its measurements can change in a matter of minutes. As stated before, the CoAP proxy uses these values to determine the freshness of each resource stored in its cache. Whenever a new cache entry is issued, a timer is started in the CoAP proxy with the `Max-Age` value as limit. When the timer expires, it triggers a function that eliminates the entry from cache immediately to ensure data freshness.

A. REVERSE PROXY OPERATION

A common reverse proxy is generally used to satisfy for requests on behalf of one or more servers. This allows to balance the load of the servers, to reply requests from responses already store in cache, and to translate between protocols or between incompatible protocol versions. Differently from forward proxies, reverse proxies are not explicitly embedded in requests, thus they are transparent to clients. Reverse proxies are widely used in the context of Web servers, databases, among others, to provide scalability to the system. These proxies are generally placed right before the servers.

In CoAP networks however, it is uncommon to find redundant CoAP servers that can benefit from a reverse proxy. According to the CoAP specification, a reverse CoAP proxy can be deployed in two of the following scenarios: 1) a reverse proxy offering several resources as their own, after having learned of their existence through the use of resource discovery mechanisms; and 2) a reverse proxy can be used to define a namespace to provide a specific configurations for the request such as embedding host identifiers and port numbers into the URI path o the resources offered. In the latter case, the reverse proxy can process the given parameters to create specific requests to a resource.

B. CROSS PROXY OPERATION

Cross proxies are more often used to translate requests from one protocol to another or between incompatible protocol versions in order to support a wide range of applications. The most common cross proxy implementation deals with the translation of HTTP requests to CoAP requests. The translation is straightforward since both protocol share the same semantics and verbs. However, the main challenge is the embedding of a CoAP URI into a HTTP request and vice versa.

C. FORWARD PROXY OPERATION

In a common forward CoAP proxy architecture, whenever a client uses the `Proxy-Uri` option available in CoAP, the request is directed to the proxy instead of the destination as shown in Figure 1. This behavior is analog to using a Virtual Private Network (VPN) in the sense that all traffic is rerouted through the proxy for the latter to speak on the client's behalf.

Figure 1 depicts the process of making a request to a CoAP server by using a forward proxy. The client starts by making a request to a known resource, in this case to coap://[fd00::206:6:6:6]/temperature. When creating the request, the client uses the `Proxy Uri` field to indicate it should use a proxy (coap://[fd00::202:2:2:2]) to communicate on the client's behalf by appending the address into the option. Depending on the CoAP client, different options can be used to achieve this task.

Upon receiving the request, the proxy checks the contents of the `Proxy-Uri` option since it uses the complete URI as a unique identifier for searching its cache storage. Two outcomes result from using the proxy: 1) if there is no cache entry for the unique identifier, the proxy creates its own request to the destination URI asking for the specified resource. When the request reaches the server, the latter issues a response to the proxy. The proxy then receives the message and creates a cache entry in memory; 2) However, if there is a cache entry for the identifier, the proxy retrieves the content from memory and immediately issues a response to the client. The first scenario requires two transactions: one between client and proxy and one between proxy and server. The second scenario requires only one transaction between client and proxy. The mechanisms for storing, retrieving, and managing content from cache is totally dependent of the implementation. In the next section we discuss in detail how we designed and implemented our embedded forward CoAP proxy server in the Contiki-NG operating system aimed at simplifying the deployment of networks and at enhancing performance by implementing an embedded cache mechanism.

IV. PROXY DESIGN AND IMPLEMENTATION

The CoAP proxy server implementation is organized into three main elements: 1) the CoAP Proxy module, which implements handling functions for requests and responses; 2) the CoAP Proxy Transactions module, used for storing client and server connection information in memory; and 3) the CoAP Proxy Cache module, which stores and manages fresh responses in memory from CoAP servers. This section describes implementation details by explaining how the proxy processes requests and responses. Figure 2 shows how the modules interact with each other.

According to the authors in [49], the CoAP engine used in Contiki-NG is based on Erbium by Mattias Kovatsh. The engine is responsible for managing CoAP requests and responses. For it to be backwards compatible with the existing implementation, we design our CoAP proxy server to be as transparent as possible. The CoAP proxy only responds whenever the "CoAP Proxy Option Processing" flag is defined in a CoAP node as an option at compilation time. This option indicates that no CoAP endpoints should be installed into the node to avoid the activation of handlers for CoAP resources and thus reducing the memory footprint usage of the node. We purposely limited the capabilities of the CoAP proxy to do only proxy-related tasks since our implementation is deployed and executed in the OpenMote-B development board, which is a resource-constrained embedded device.

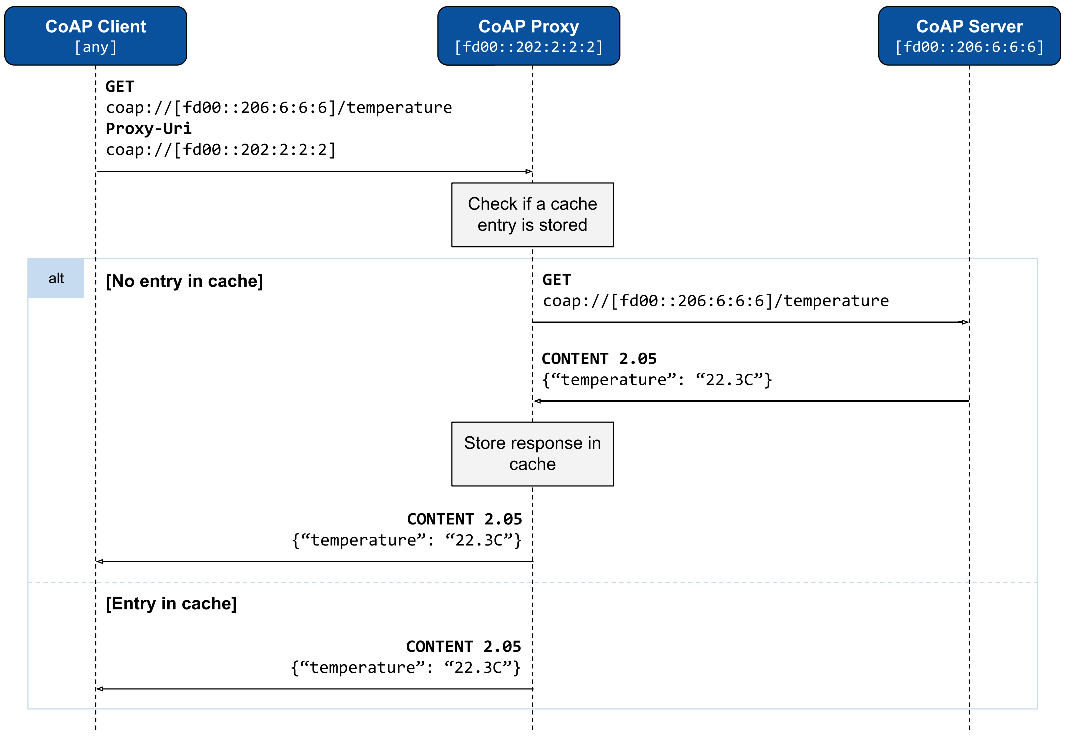

FIGURE 1. **CoAP forward proxy operation. The `Proxy-Uri` option in the request indicates that the messages must be routed through the CoAP proxy instead of going directly to the destination server. When the proxy implements cache management mechanisms, the proxy will immediately send a response with its cache entry if the requested resource is stored in memory, thus reducing the number of exchanged messages between client and server.**

Since efficiency is an important aspect for systems with limited resources, we extracted information about RAM and ROM usage from the OpenMote-B binary file. The CoAP proxy has a ROM size of $56,022$ bytes. The initialized data is $1,680$ bytes and the uninitialized data is $13,858$ bytes. Both memory usage sizes were extracted by using the `size` command. The OpenMote-B platform has $32KB$ in RAM and $512KB$ in flash. Therefore it is fully capable of running the CoAP proxy binary.

A. HANDLING REQUESTS

Whenever the CoAP proxy receives an incoming message through the CoAP engine, it checks whether the `Proxy-Uri` is present in the request. This is important since a CoAP proxy can only act as regular node with forwarding capabilities inside the network. If the option is present, the proxy first searches for an entry for the desired address and endpoint in cache. In our implementation, we use the Memory Blocks library for static and dynamic memory allocations in Contiki-NG to store cache entries in a Linked List, which is also a library for manipulating linked lists in the operating system. Both libraries are available as part of the Contiki-NG development platform. To provide unique identifiers to cache entries, we serialize the complete IPv6 address of the target node along with its resource path. Such functionality is implemented in the CoAP Proxy Cache module.

If a cache entry is available for a particular address and resource, the CoAP proxy retrieves the content from cache and sends a response through the same transaction the request arrived. On the other hand, if there is no entry in cache, the proxy performs a series of procedures. Initially, it creates a new transaction that will be used to communicate with the CoAP server. It is important to store the transactions for incoming and outgoing messages since each transaction carries information about message number ID, and tokens that provide CoAP reliability capabilities. All tasks related to storing and retrieving transactions are performed at the CoAP Proxy Transactions module. Once the transaction is created, the proxy extracts the information from the request to create a new message with the target endpoint, which is composed of the node's IPv6 address and its request path. When the message is already constructed, the proxy sends this new request message to the target server.

B. HANDLING RESPONSES

When an incoming message in the proxy is a response to a previous request from the server, the prior uses the CoAP Proxy Transaction module to retrieve information about the original transaction made by a client and the current transaction with the server. After extracting the proxy transaction information, the proxy checks whether there is a cache entry from the current server response. If there is no entry, the proxy creates a CoAP Proxy Cache entry using the resource

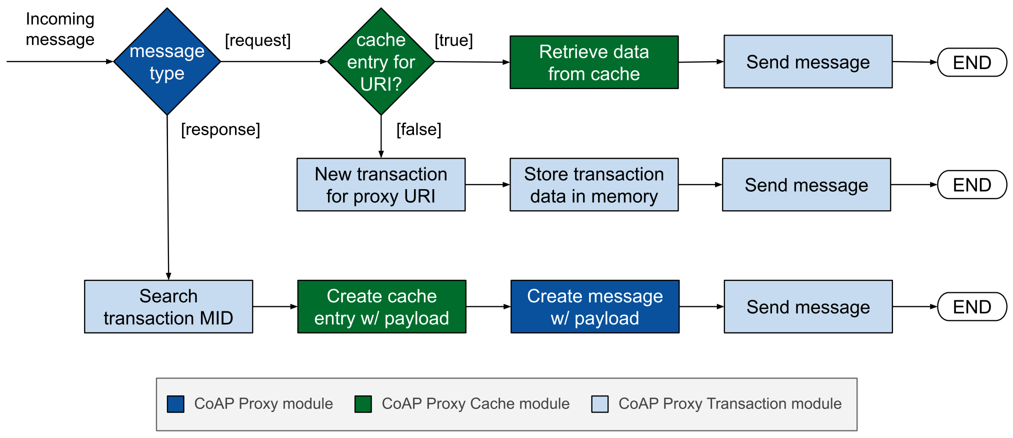

FIGURE 2. CoAP proxy server modules. The CoAP proxy module is responsible for business logic, the CoAP proxy cache module is responsible for data allocation/deallocation and data freshness validation, and finally the CoAP proxy transaction module is responsible for keeping records of the source and target transaction information for requests and responses in the CoAP proxy. The modular design of the CoAP proxy allows faster development, greater flexibility, easier debugging, and extensibility.

URI, the message payload, and the message `Max-Age` option defined by the server. In our current implementation, the only valid cache is text-based data since we commonly use JSON to structure the data for the information exchange. Whenever a cache entry is issued, the CoAP Proxy Cache module initiates a callback timer (`ctimer` in Contiki-NG) that executes a `remove_timed_cache_entry()` function when expiring. Such function removes the cache entry from the proxy cache memory to ensure data freshness at all times.

After checking and creating a cache entry if necessary, the proxy uses the information extracted from the CoAP Proxy transaction to use the same message ID and endpoint to issue a response to the client. It then creates a new message with code `2.05 Content` that indicates a response from the server to the originating client. The message payload is filled with the payload from the incoming message issued from the server. Finally, the proxy sends the message to the client and cleans its buffer from the CoAP Proxy Transaction entry for the corresponding connection.

V. EVALUATION SETUP

In 6LoWPANs, the depth of the network is a key factor to determine performance and energy consumption since they operate without infrastructure, this means that mechanisms for message forwarding must be implemented in order to reach all the nodes inside the network. The IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) is the recommended routing protocol for 6LoWPANs [50]. According to [51], a network with greater depth requires longer space in the packet's header to include source routing header (SRH) information. Moreover, packets generated in the leaf nodes imply the routing of more packets in their parents since they have to forward the sum of all packets from all its children.

We analyze the performance of our CoAP proxy in terms of response time, and number of exchanged messages in both simulated and experimental environments. We define the response time as the elapsed time from the generation of the CoAP request and until a response is received at the client. The number of messages is the sum of all packages generated at each hop of the network. Energy consumption is an important key indicator for 6LoWPAN nodes because maximizing the lifetime of the network is often an important requirement when designing an architecture. Moreover, energy consumption is completely dependent on the hardware platform. According to the CC2538 data sheet [52], the Texas Instrument platform consumes $13mA$ from CPU running at $32MHz$ with flash access, $20mA$ from radio in RX (reception) mode, -50 dBm input power, no peripherals active, CPU idle, and $34mA$ from radio in TX (transmission) mode, $7dBm$ output power, no peripherals active, CPU idle.

Sciancalepore et al. [53] measured the current drawn by the OpenMote-B platform, which uses the CC2538, and concluded that packet reception requires $\approx$ 35 mA ($13mA$ from the CPU, $2mA$ from the LEDs, and $20mA$ from the RX mode for the radio) and packet transmission requires $\approx$ 54 mA ($13mA$ from the CPU, $4mA$ from the LEDs, $3mA$ from the USB UART communication, and $34mA$ from the TX mode for the radio). To evaluate the actual energy consumption, authors removed all factors related to debugging components such as LEDs and the USB UART at both the receiver and transmitter. The RX process consumes approximately $38mA \times 1\ ms = 38\ mJ$ for the radio preparation and $18mA \times 3\ ms = 54\ mJ$ for the actual receiving processing, resulting in $92mJ$ for reception. The transmission procedure consumes $20mA \times 2\ ms = 40\ mJ$ for radio preparation and $27mA \times 3\ ms = 81\ mJ$ for the transmission, resulting in $121mJ$. Overall, considering the remaining CPU cycles and the duration of the slot, each TX slot consumes up to $186mJ$ and each RX slot consumes up to $170mJ$. The number of exchanged messages between nodes is directly related to the energy consumption. However, it is out of the scope of this paper since many factors need to be taken into account to accurately measure energy consumption.

In both our simulated and experimental environments, we placed a Border Router that interconnects a CoAP client with our proxy server and thus with the inside of the 6LoWPAN. The client performs a total of 24 requests to each node inside the 6LoWPAN within 10 second intervals. The number of requests and time interval values were chosen so that the `Max-Age` values at the cache expire at least three times to study its behavior. The client records the time of the departure and arrival of every request. The `Max-Age` parameter for resources at each CoAP server is set to 60 seconds as the default value recommended by CoAP. This means that every 60 seconds the content of cache entries at the proxy expire and a new transaction for new cache entries is issued. As mentioned before, the value of the `Max-Age` parameter is specified according to the type of sensor and the needs of the network. To generate CoAP requests to the inside of the 6LoWPAN we used the well-known `coap-client` application from libcoap.[1]

Due to the limited number of hardware nodes, we also conducted a series of simulations to further study the behavior of the proxy when several more nodes are deployed in the 6LoWPAN.

This section describes two important key performance indicators for proxies, namely response time and number of packets sent. It also depicts the simulation and experimental environments in detail.

A. RESPONSE TIME

Response time measures the total time a transaction takes to complete from the moment a CoAP request is generated up to the moment its corresponding response is received. Specifically in 6LoWPAN, response time may vary due to factors such as node processing power, communication channel bandwidth, current traffic network load, and network depth [54], [55]. Nodes inside a 6LoWPAN may vary in processing power and are generally associated with low power capacity. The lower the processing power, the higher delay is introduced into the response time of the transaction. In addition, the current network link load also affects in the delay of the response. By adding both the delay introduced by the low processing power of the node and the delay introduced by the current load of the network link we can obtain the one-way response time of the transaction. This response time calculation is computed for each hop the packet must perform until it reaches its destination. Therefore, network depth must be taken into the computation of the response time since the more depth a node is deployed, the more delay is introduced into the response time. Moreover, the response time is composed by the time it takes for a request to reach the server plus the time it takes for the response to arrive at the client.

[1]https://libcoap.net

TABLE 1. Simulation environment parameters.

Parameter	Value
Simulation time	112 minutes
Total number of nodes	17 nodes
Distance between nodes	$\approx$ 40 meters apart
Node type	Cooja nodes
Node radio	IEEE 802.15.4

B. NUMBER OF PACKETS SENT

6LoWPANs are characterized by two fundamental elements: embedded devices generally use low-power and lossy links and networks can be formed without infrastructure. To cover large areas, the IEEE 802.15.4 standard allows nodes to create networks by using routing protocols such as RPL to maintain and provide paths for packets to reach their destination. The number of packets sent is closely related to depth of the network. By using proxy cache mechanism we argue that we can achieve a reduction in the number of exchanged messages between nodes inside the 6LoWPAN since the proxy can send responses to requests when content is fresh and available in cache. In both experimental and simulation environments, we consider the number of packets or exchanged messages traversing the 6LoWPAN from the Border Router and into the 6LoWPAN. The number of hops outside the 6LoWPAN are not considered.

C. SIMULATION ENVIRONMENT

The simulation environment is useful to perform evaluations with several nodes. Moreover, simulations provide an interference-free environment and other connection issues associated with real-world deployments. This allows to measure only the performance of our CoAP Proxy server in terms of response time and number of packets exchanged. Also, the number of nodes that can be deployed in a network is virtually unlimited at low costs. Figure 3 shows the topology of our simulation scenario.

Every leaf node in the topology has two endpoints: a humidity and temperature sensor resourcea. The proxy uses its cache management mechanism when the `Proxy-Uri` option is present in the request message. Otherwise, it acts as a simple RPL relay node when the option is not carried in the message. The Border Router allows the interconnection of the 6LoWPAN with the Internet. The client issues requests to each server inside the IoT network and records response times and the number of messages exchanged. Table 1 shows the simulation parameters.

The CoAP client issues 24 requests every 10 seconds to each server in the topology. A first run is performed without the CoAP Proxy option enabled and a second run is performed with the option enabled. This results in an 112 minute simulation time overall. There are 17 nodes in total from which 14 are servers, one is a client that issues requests to each server, one is a Border Router that creates a bridge between the 6LoWPAN and the Internet, and one is our proxy server. Each of the 6LoWPAN nodes are created

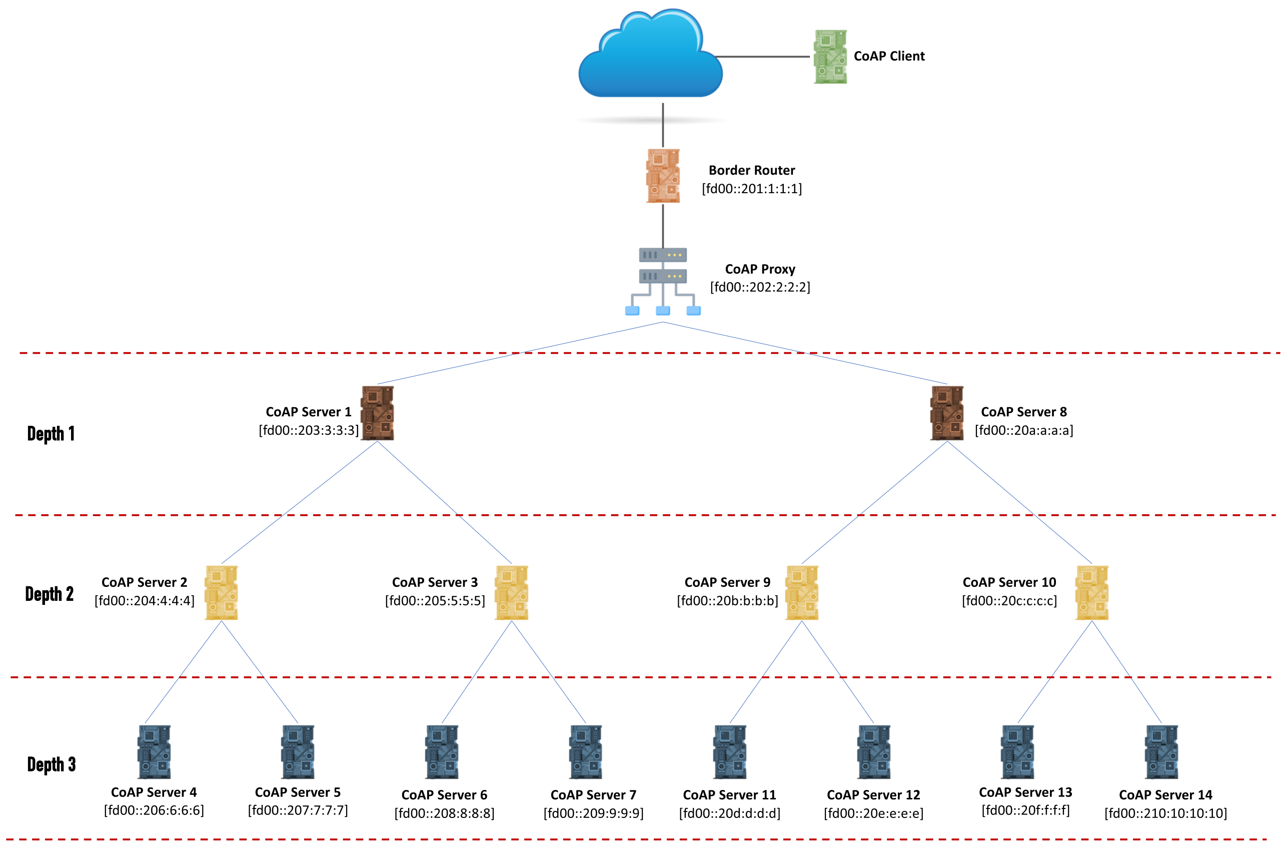

FIGURE 3. Simulation environment topology. In the setup, the depth of a given node is defined as the number of hops a packet performs from the CoAP proxy node to its target. Nodes 1 and 8 have a depth of 1. Nodes 2, 3, 9, and 10 have a depth of 2. And nodes 4, 5, 6, 7, 11, 12, 13, and 14 have a depth of 3.

as Cooja nodes and the distance between each CoAP server is approximately 40 meters apart to create a hierarchical topology.

We performed our simulations with the Cooja Simulator included in the Contiki-NG operating system. One important feature of this simulator is that it can emulate different hardware platforms. Therefore, the same code developed for the simulations were used in real hardware with the OpenMote-B devices. The hardware components running the simulator are a 12th Gen Intel Core i7-1255U CPU with 12 cores and 24 threads with 32GB of RAM and a 512GB NVMe disk drive running Fedora Linux 37 (Workstation Edition).

D. EXPERIMENTAL ENVIRONMENT

In our experimental environment we used the OpenMote-B hardware platform which uses an ARM Cortex-M3 with an IEEE 802.15.4 radio. Since we are studying the performance in terms of response time and the number of packets exchanged between a CoAP client and a server in a 6LoWPAN, we defined a network topology with a network depth of three hops between the CoAP proxy and the servers. The topology is shown in Figure 4. The Border Router (BR) is connected to a computer through the USB serial port to route requests from the CoAP Client (CC). The client creates requests to each of the servers (CS1, CS2, and CS3) in the network and records the response times for each request. The CoAP Proxy (CP) is located next to the BR and it is used to route requests to the sever nodes. Every CoAP server in the network is programmed with two endpoints: a humidity and a temperature sensor resources.

To measure performance, we developed a script that issues as records requests to each CoAP server in the network every 10 seconds. The same script is used for gathering results in our experimental and simulation environments. A first set of experiments only uses the CoAP Proxy as a routing node since the CoAP Proxy Option is not present in the requests. A second set of experiments uses the CoAP Proxy Option to enable the cache functionality in the proxy. We then compared the results of using the CoAP Proxy with and without the cache functionality. Table 2 shows the experimental setup parameters.

TABLE 2. Experimental environment parameters.

Parameter	Value
Simulation time	24 minutes
Total number of nodes	5 nodes
Distance between nodes	$\approx$ 7.5 meters apart
Node type	OpenMote-B nodes
Node radio	IEEE 802.15.4

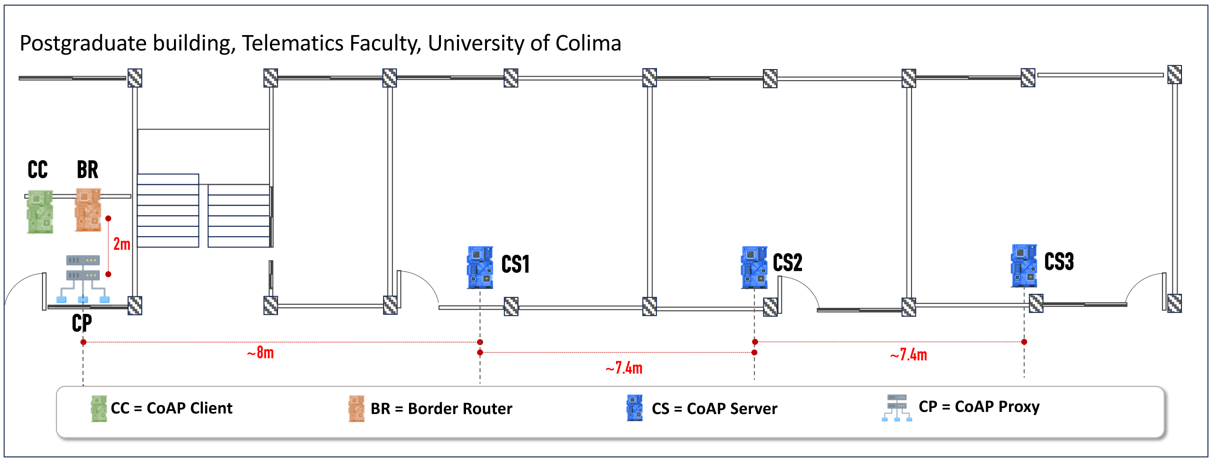

FIGURE 4. Experimental environment topology. The CoAP Client (CC) makes requests to each of the CoAP Servers (CS1, CS2, and CS3 respectively). The border router (BR) is connected directly to a computer. The CoAP Proxy (CP) is located close to the BR and in front of the CoAP Servers.

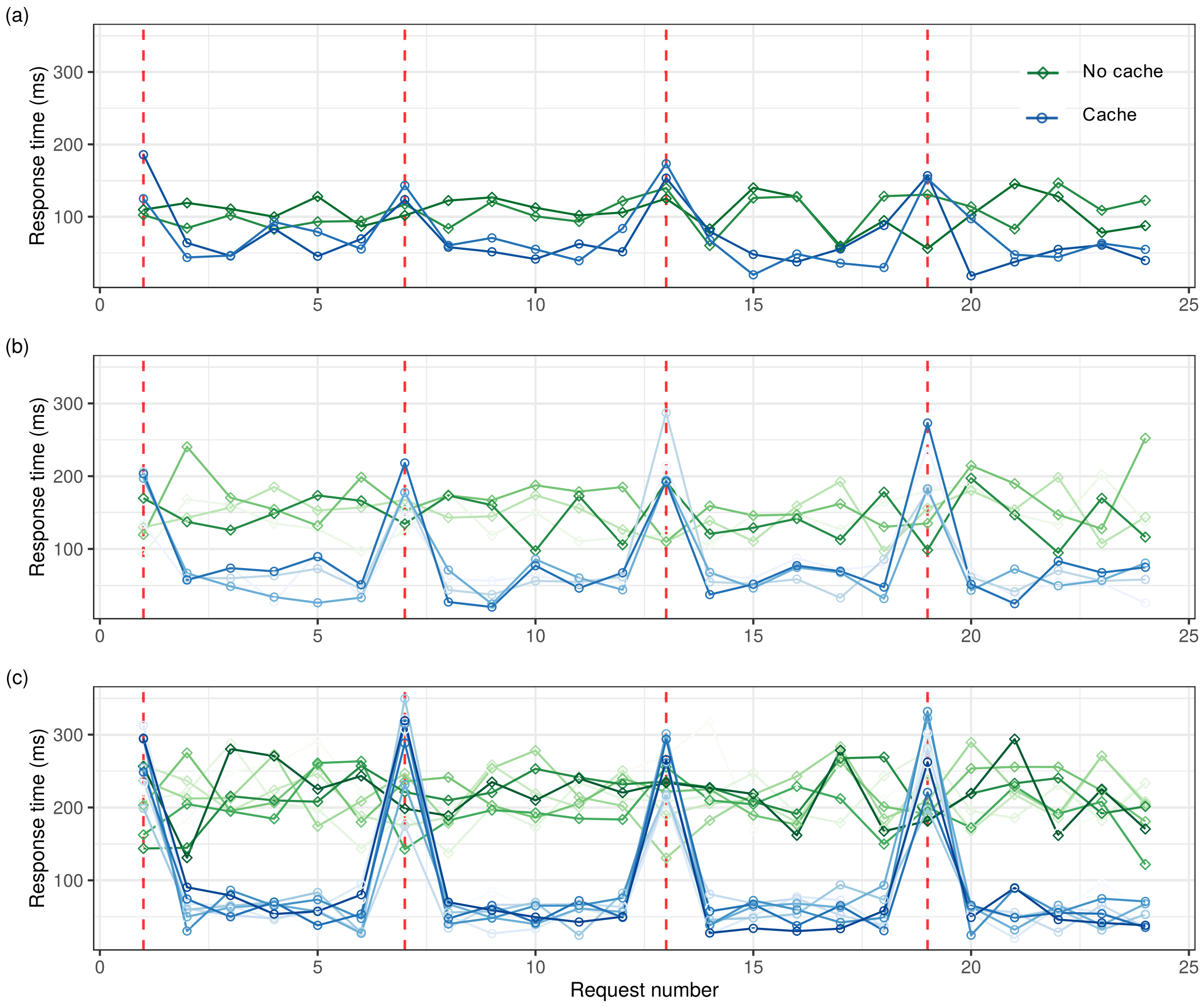

FIGURE 5. Simulation response time series for CoAP requests for (a) one hop, (b) two hops, and (c) three hops away from the CoAP Proxy server for requests without cache (green lines) and with cache responses (blue lines). Response times when cache is available improves as the number of hops between server and proxy increases.

The experimental setup allows realistic view of the proxy's performance since simulations are not affected mostly by interference and path loss.

VI. RESULTS EVALUATION

This section shows our results and discusses our findings and is organized into three subsections: time response results

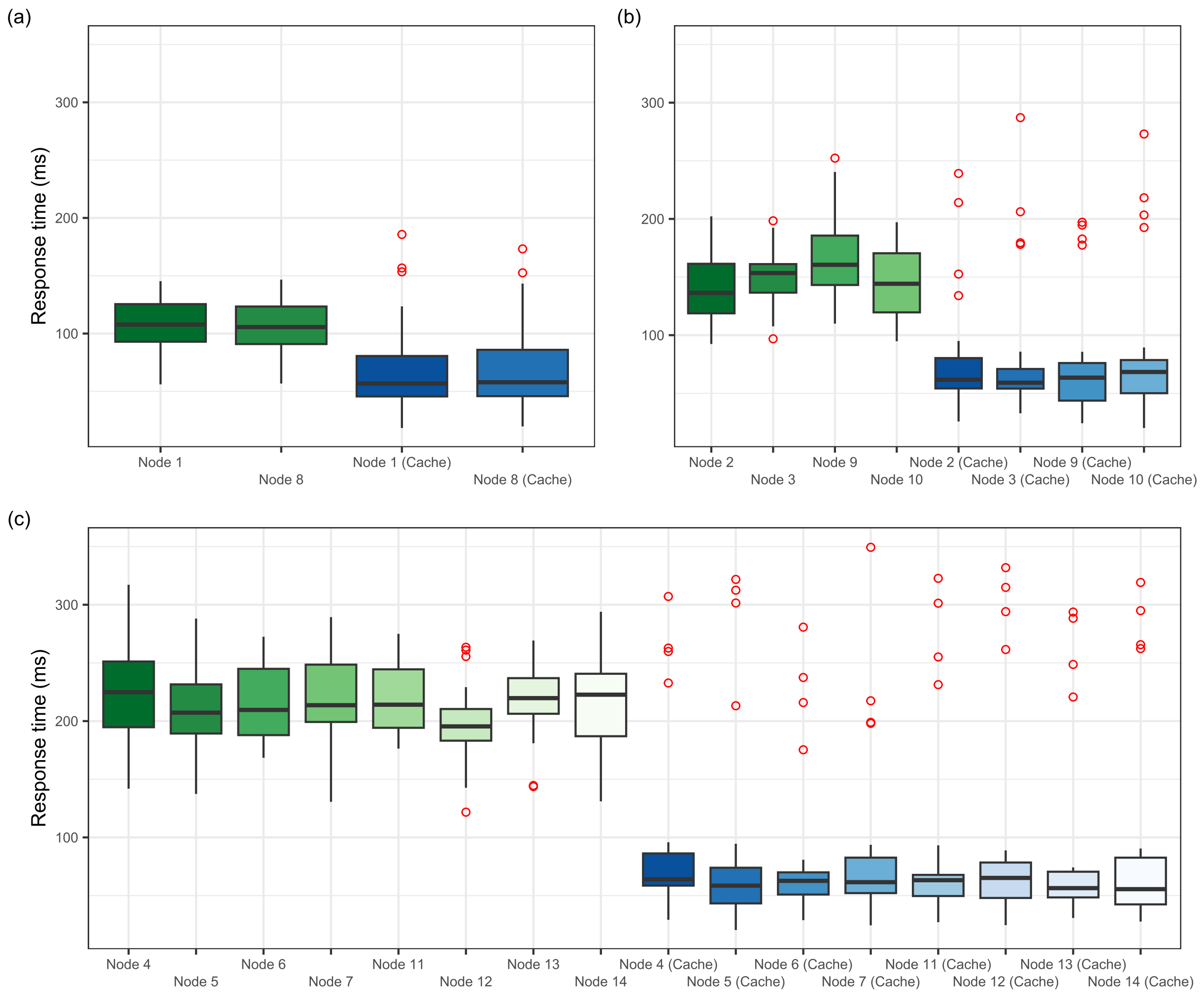

FIGURE 6. **Response time statistics for CoAP requests for (a) one hop, (b) two hops, and (c) three hops away from the proxy server. The outliers presented as red circles mostly represent an increase in response times whenever a CoAP resource is requested and no cache entry is available at the proxy server.**

for the simulation environment, time response results for the experimental environment, and number of packets exchanged results for both environments.

As mentioned in the previous section (Section V, we performed a set of CoAP requests without using the CoAP Proxy and another set using the proxy in both simulation and experimental environments. When using the proxy, the `coap-client` application uses the `-p` parameter to configure the proxy IPv6 address.

The time to reach a node inside the 6LoWPAN is mostly influenced by the number of hops a request experiences on its way to the destination. Therefore, is important to differentiate between performance measurements for one-hop, two hops, and three-hops depth nodes.

A. RESPONSE TIME RESULTS FOR THE SIMULATION ENVIRONMENT

Response times in our simulated environment are depicted in Figure 5 for requests at (a) one hop, (b) two hops, and (c) three hops away from the proxy. The associated delay introduced by the proxy when there is no cache entry available in memory to create a response is smaller compared to the experimental results as will be discussed later. This is because simulated Cooja nodes have no memory and processing restrictions since the node is compiled as a native process in the operating system. However, there is still a small delay introduced due to the new transaction that is triggered when no cache entry is found in memory.

From Figure 5, we observe the response time delay increase when there is no cache entry in the proxy's memory at requests 1, 7, 13, and 19. However, consequent requests for the same node and resource are quickly dispatched at the CoAP Proxy server as requests between that timeframe are lower than responses without using the proxy. The pattern becomes more evident as the network depth and number of nodes increase in Figure 5 (b) and (c) sections. Moreover, the difference in response times from when using the proxy also become more prominent as network node depth increases.

At one-hop depth, results show an average response time of 106.08 *ms* without using the proxy server and an average of 71.66 *ms* using the proxy server's cache mechanism. This

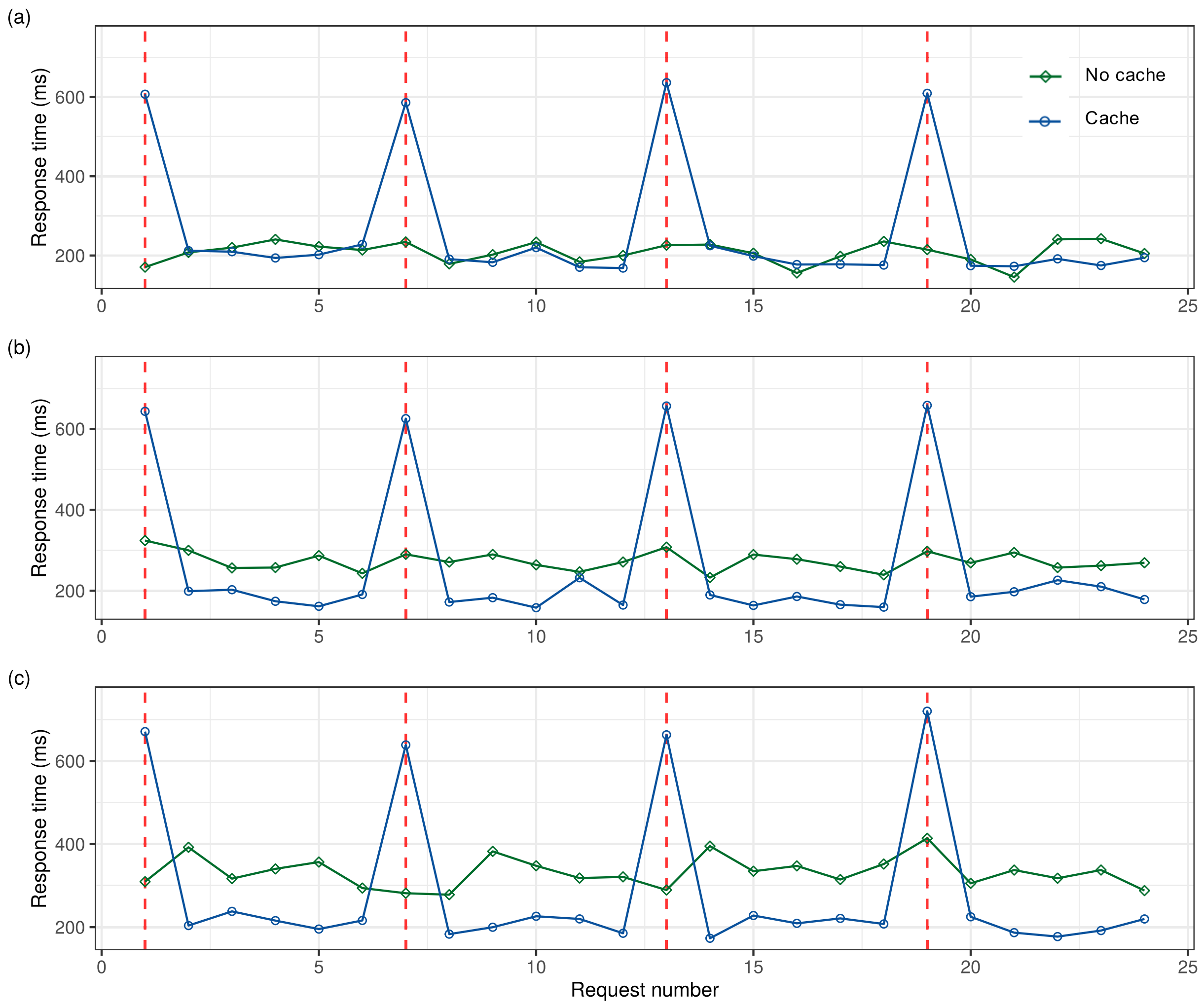

FIGURE 7. Experimental response time series for CoAP requests for (a) one hop, (b) two hops, and (c) three hops away from the CoAP proxy server. Green-colored lines in the graph are the requests made without the use of cache at the CoAP proxy server. Blue-colored lines are the requests made with the use of the CoAP proxy server. Red lines indicate when the proxy performs a request to populate its cache. Response times decrease when the number of hops between proxy and servers increases.

is plausible since the distance between the CoAP client and the CoAP server is just one hop away from the proxy. The median is placed at 106.71 *ms* and 57.29 *ms* for requests without and with cache responses respectively. There is a time reduction of 48.03% whenever the CoAP client used our CoAP Proxy server and a difference of 34.42 *ms* between averages. Figure 6 (a) shows detailed statistics for response times.

At two-hop depth, results shown an average response time of 150.05 *ms* without using the CoAP server and an average of 80.93 *ms* when using the proxy server as shown in Figure 6 (b). This represents a response time reduction of 85.39% and a difference of 69.12 *ms* between averages.

At three-hop depth, results show an average response time of 215.2 *ms* when proxy server does not employ cache management, and an average of 91.88 *ms* whenever the proxy with cache is being used. This imposes a response time reduction of 134.21% and a time difference of 123.32 *ms* between averages. Figure 6 (c) shows a detailed representation of such results.

B. RESPONSE TIME RESULTS FOR THE EXPERIMENTAL ENVIRONMENT

The response time indicator allows to quantify the performance of our 6LoWPAN when using the CoAP proxy server with cache management enabled and without cache. Figure 7 shows the results for (a) requests to a CoAP server at one-hop depth, (b) two-hop depth, and (c) three-hop depth in the network. In the graph, the green-colored lines correspond to the requests performed without cache management at the proxy. The blue-colored lines correspond to requests performed with the CoAP Proxy Option enabled to use cache management at the proxy.

The default `Max-Age` option for cache management as defined in CoAP is 60 seconds. To avoid adding complexity to the environment, each CoAP resource located at the

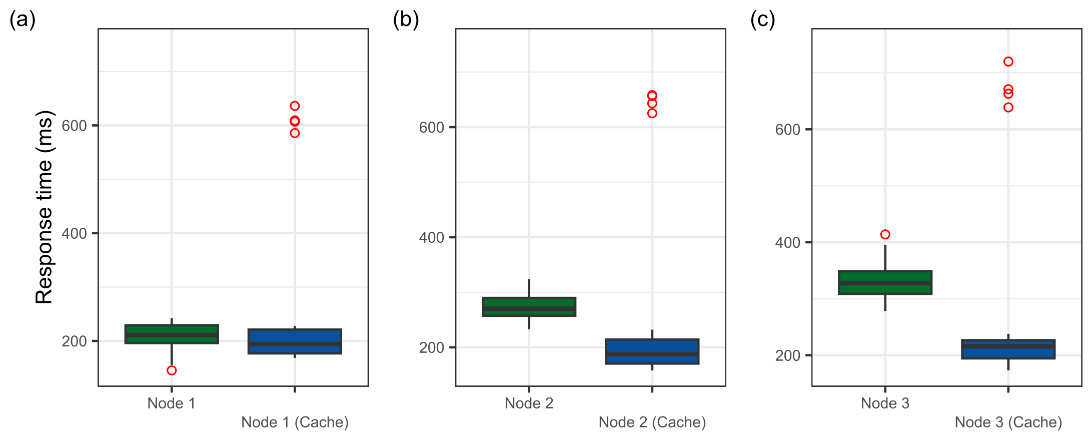

FIGURE 8. Experimental response time statistics for CoAP requests for (a) one hop, (b) two hops, and (c) three hops away from the CoAP proxy server. The outliers presented as red circles represent an increase in response times whenever a CoAP resource is requested and no cache entry is available at the CoAP proxy server. Response times are lower depending on the number of hops when cache at the proxy is employed.

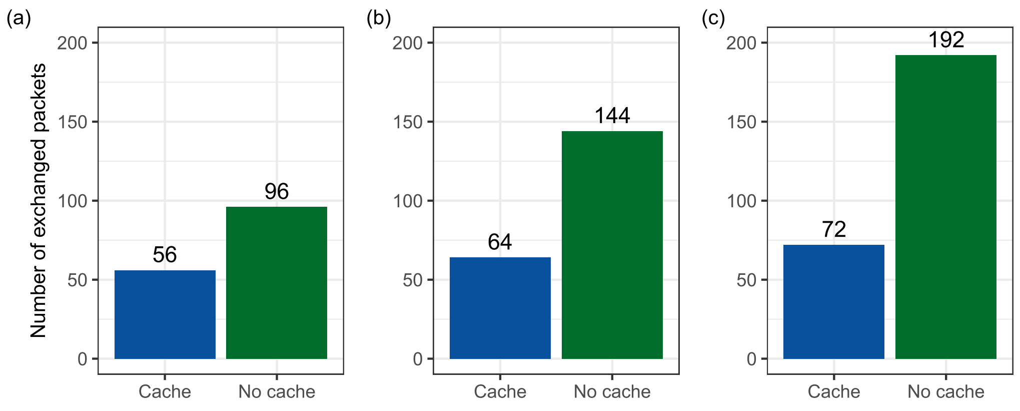

FIGURE 9. Number of messages exchanged at (a) one hop, (b) two hops and (c) three hops away from the CoAP Proxy Server.

server nodes is configured with the default `Max-Age` value. Therefore, our proxy server parses this value from the response and initiates a timer to flush the entry from the cache whenever the timer expires.

From Figure 7 (a), (b), and (c) we can observe that the proxy introduces a significant delay when there is no cache entry available in memory to create a response. This is mainly because the proxy has to create a new transaction request to the server in order to recollect the data to store it in cache memory. In Figure 7 (a) we also observe that response times when there is a cache entry in the proxy and when no proxy is used are very similar. Therefore, using a proxy with cache capabilities in networks with one hop topology does not provide performance gains in terms of response times since the overall average is 261.59 *ms* when using the cache compared to 208.14 *ms* not using the cache. However, when we compare the average times for server at two and three hops away from the proxy, we observe a decrease in response times: 261.77 *ms* with cache and 273.15 *ms* without cache for two hops, and 286.30 *ms* with cache and 332.22 *ms* without cache for three hops.

If we pay close attention to the requests in between the ones made when a new cache entry is required (which are values close to the median), we find that the response times are lower for servers at every hop. In Figure 8 we observe such behavior. The average times for requests at one, two, and three hops are: 192.01 *ms*, 184.98 *ms*, and 208.93 *ms* when using the cache respectively compared to 207.49 *ms*, 266.82 *ms*, and 333.93 *ms* when not using the cache. This results in a decrease in time responses of 7.46%, 30.67%, and 37.43% for requests at one, two, and three hops respectively. We can infer that, once the CoAP proxy has a cache entry in memory, better performance can be expected if we increase the number of hops in the network.

C. NUMBER OF EXCHANGED PACKETS

Energy consumption in 6LoWPANs is a key component when designing and deploying a network. Every time a node uses its IEEE 802.15.4 radio its consumes energy from its source. After energy depletion of the first node, the performance of the network degrades [56]. One approach to energy saving is the use of cache inside the network to alleviate load in battery-powered nodes since the number of packets or exchanged messages between nodes can be reduced significantly. Figure 6 shows the number of messages exchanged with each hop.

In Figure 9 we can see the number of exchanged messages between nodes in our simulation results. The difference between using and not using our CoAP Proxy at one hop away is 40 packets with, resulting in a 71.42% of packet exchange decrease. At two hops, the difference is 80 packets with a 125% packet exchange decrease. Finally, at three hop the difference is 120 fewer packets exchanged and a 166.66% packet exchange decrease. Increasing the `Max-Age` parameter will also decrease the number of exchanged packets since cache entries stay fresh for longer periods of time.

VII. CONCLUSION

The use of CoAP proxies with cache capabilities in 6LoWPANs can increase performance with respect of response times and energy consumption. We designed, implemented, and evaluated an embedded CoAP proxy for the Contiki-NG operating system. The design is based on a modular architecture to allow fast integration and testing of new features. Moreover, the implementation followed recommended coding conventions and best practices provided by the Contiki-NG project. We performed a simulation evaluation using the Cooja simulator included with Contiki-NG anbd an experimental evaluation using the OpenMote-B hardware platform. We defined network topologies with a three-hop network depth. We studied the effect of using a CoAP Proxy cache management in terms of response time and the number of packets exchanged at different depths. We performed tests by sending 24 CoAP requests every 10 seconds from a client to each server in the network. The definition of the number of requests and sending interval is directly related to the `Max-Age` parameter of the CoAP resources which indicates the number of seconds a cache entry should be valid since we want entries to expire in order to study its impact in response times. Simulation results show a reduction in response times of 48.03%, 85.39%, and 134.21% for requests to servers at one, two, and three hops away respectively. The nodes in the simulation environment have no memory and processing restrictions since the node is compiled as a native process in the operating system. Experimental results further confirm our simulation results showing a reduction in response times of 7.46%, 30.67%, and 37.43% for requests to servers at one, two, and three hops away from the proxy server respectively when using cache. These results could vary depending on the hardware platform due to memory and CPU capabilities. In both experimental and simulation environments, results show a packet exchange reduction of 71.42%, 125%, and 166.66% for requests at one, two, and three hops respectively.

In this paper we proved the feasibility and evaluated the performance in both simulation and experimental environments of an embedded CoAP proxy server. To the best of our knowledge, this is the first embedded CoAP Proxy server implemented for the Contiki-NG operating system. We recommend to carefully select the `Max-Age` parameter according to the requests intervals to further optimize response times. Future work is planned to evaluate and study of optimization mechanisms for observable CoAP requests to reduce response time in managing cache entries, to broadening the scenarios to include different network setups, varying device densities, different cache expiration periods, and device mobility to provide more comprehensive results. Also the study of cooperative cache algorithms to further improve performance in large-scale IoT networks is desirable.

REFERENCES

[1] M. N. Bhuiyan, M. M. Rahman, M. M. Billah, and D. Saha, "Internet of Things (IoT): A review of its enabling technologies in healthcare applications, standards protocols, security, and market opportunities," *IEEE Internet Things J.*, vol. 8, no. 13, pp. 10474–10498, Jul. 2021.

[2] S. M. R. Islam, D. Kwak, M. D. H. Kabir, M. Hossain, and K. S. Kwak, "The Internet of Things for health care: A comprehensive survey," *IEEE Access*, vol. 3, pp. 678–708, 2015.

[3] D.-K. Choi, J.-H. Jung, S.-J. Koh, J.-I. Kim, and J. Park, "In-vehicle infotainment management system in Internet-of-Things networks," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2019, pp. 88–92.

[4] D. Minoli, K. Sohraby, and B. Occhiogrosso, "IoT considerations, requirements, and architectures for smart buildings—Energy optimization and next-generation building management systems," *IEEE Internet Things J.*, vol. 4, no. 1, pp. 269–283, Feb. 2017.

[5] A. Verma, S. Prakash, V. Srivastava, A. Kumar, and S. C. Mukhopadhyay, "Sensing, controlling, and IoT infrastructure in smart building: A review," *IEEE Sensors J.*, vol. 19, no. 20, pp. 9036–9046, Oct. 2019.

[6] M. B. Mollah, J. Zhao, D. Niyato, Y. L. Guan, C. Yuen, S. Sun, K.-Y. Lam, and L. H. Koh, "Blockchain for the Internet of Vehicles towards intelligent transportation systems: A survey," *IEEE Internet Things J.*, vol. 8, no. 6, pp. 4157–4185, Mar. 2021.

[7] 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.

[8] L. Hu and Q. Ni, "IoT-driven automated object detection algorithm for urban surveillance systems in smart cities," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 747–754, Apr. 2018.

[9] K. Muhammad, R. Hamza, J. Ahmad, J. Lloret, H. Wang, and S. W. Baik, "Secure surveillance framework for IoT systems using probabilistic image encryption," *IEEE Trans. Ind. Informat.*, vol. 14, no. 8, pp. 3679–3689, Aug. 2018.

[10] F. M. Bencic, P. Skocir, and I. P. Žarko, "DL-tags: DLT and smart tags for decentralized, privacy-preserving, and verifiable supply chain management," *IEEE Access*, vol. 7, pp. 46198–46209, 2019.

[11] S. Mondal, K. P. Wijewardena, S. Karuppuswami, N. Kriti, D. Kumar, and P. Chahal, "Blockchain inspired RFID-based information architecture for food supply chain," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5803–5813, Jun. 2019.

[12] Z. Qu, G. Zhang, H. Cao, and J. Xie, "LEO satellite constellation for Internet of Things," *IEEE Access*, vol. 5, pp. 18391–18401, 2017.

[13] N. Hossein Motlagh, T. Taleb, and O. Arouk, "Low-altitude unmanned aerial vehicles-based Internet of Things services: Comprehensive survey and future perspectives," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 899–922, Dec. 2016.

[14] Y. Song, F. R. Yu, L. Zhou, X. Yang, and Z. He, "Applications of the Internet of Things (IoT) in smart logistics: A comprehensive survey," *IEEE Internet Things J.*, vol. 8, no. 6, pp. 4250–4274, Mar. 2021.

[15] D. Alahakoon and X. Yu, "Smart electricity meter data intelligence for future energy systems: A survey," *IEEE Trans. Ind. Informat.*, vol. 12, no. 1, pp. 425–436, Feb. 2016.

[16] R. Morello, C. De Capua, G. Fulco, and S. C. Mukhopadhyay, "A smart power meter to monitor energy flow in smart grids: The role of advanced sensing and IoT in the electric grid of the future," *IEEE Sensors J.*, vol. 17, no. 23, pp. 7828–7837, Dec. 2017.

[17] K. Zhao, M. Zhu, B. Xiao, X. Yang, C. Gong, and J. Wu, "Joint RFID and UWB technologies in intelligent warehousing management system," *IEEE Internet Things J.*, vol. 7, no. 12, pp. 11640–11655, Dec. 2020.

[18] N. Ahmed, D. De, and I. Hussain, "Internet of Things (IoT) for smart precision agriculture and farming in rural areas," *IEEE Internet Things J.*, vol. 5, no. 6, pp. 4890–4899, Dec. 2018.

[19] M. Ayaz, M. Ammad-Uddin, Z. Sharif, A. Mansour, and E. M. Aggoune, "Internet-of-Things (IoT)-based smart agriculture: Toward making the fields talk," *IEEE Access*, vol. 7, pp. 129551–129583, 2019.

[20] M. S. Farooq, S. Riaz, A. Abid, K. Abid, and M. A. Naeem, "A survey on the role of IoT in agriculture for the implementation of smart farming," *IEEE Access*, vol. 7, pp. 156237–156271, 2019.

[21] S. A. Shah, D. Z. Seker, S. Hameed, and D. Draheim, "The rising role of big data analytics and IoT in disaster management: Recent advances, taxonomy and prospects," *IEEE Access*, vol. 7, pp. 54595–54614, 2019.

[22] S. Dhingra, R. B. Madda, A. H. Gandomi, R. Patan, and M. Daneshmand, "Internet of Things mobile–air pollution monitoring system (IoT-mobair)," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5577–5584, Jun. 2019.

[23] F. K. Shaikh, S. Zeadally, and E. Exposito, "Enabling technologies for green Internet of Things," *IEEE Syst. J.*, vol. 11, no. 2, pp. 983–994, Jun. 2017.

[24] S. Pattar, R. Buyya, K. R. Venugopal, S. S. Iyengar, and L. M. Patnaik, "Searching for the IoT resources: Fundamentals, requirements, comprehensive review, and future directions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2101–2132, 3rd Quart., 2018.

[25] *IEEE Standard for Low-Rate Wireless Networks*, IEEE Standard 802.15.4-2020 (Revision of IEEE Standard 802.15.4-2015), 2020, pp. 1–800.

[26] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, *Transmission of IPv6 Packets Over IEEE 802.15.4 Networks*, document RFC 4944, Sep. 2007.

[27] J. Hui and P. Thubert, *Compression Format for IPv6 Datagrams Over IEEE 802.15.4-Based Networks*, document RFC 6282, Sep. 2011.

[28] J. W. Hui and D. E. Culler, "Extending IP to low-power, wireless personal area networks," *IEEE Internet Comput.*, vol. 12, no. 4, pp. 37–45, Jul./Aug. 2008.

[29] Z. Shelby, K. Hartke, and C. Bormann, *The Constrained Application Protocol (CoAP)*, document RFC 7252, Jun. 2014.

[30] H. Washizaki, S. Ogata, A. Hazeyama, T. Okubo, E. B. Fernandez, and N. Yoshioka, "Landscape of architecture and design patterns for IoT systems," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 10091–10101, Oct. 2020.

[31] A. B. Sulaeman, F. A. Ekadiyanto, and R. F. Sari, "Performance evaluation of HTTP-CoAP proxy for wireless sensor and actuator networks," in *Proc. IEEE Asia–Pacific Conf. Wireless Mobile (APWiMob)*, Sep. 2016, pp. 68–73.

[32] A. P. Castellani, T. Fossati, and S. Loreto, "HTTP-CoAP cross protocol proxy: An implementation viewpoint," in *Proc. IEEE 9th Int. Conf. Mobile Ad-Hoc Sensor Syst. (MASS)*, Oct. 2012, pp. 1–6.

[33] J. Esquiagola, L. Costa, P. Calcina, and M. Zuffo, "Enabling CoAP into the swarm: A transparent interception CoAP-HTTP proxy for the Internet of Things," in *Proc. Global Internet Things Summit (GIoTS)*, Jun. 2017, pp. 1–6.

[34] Z. Mi and G. Wei, "A CoAP-based smartphone proxy for healthcare with IoT technologies," in *Proc. IEEE 9th Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Nov. 2018, pp. 271–278.

[35] A. Ludovici and A. Calveras, "A proxy design to leverage the interconnection of CoAP wireless sensor networks with Web applications," *Sensors*, vol. 15, no. 1, pp. 1217–1244, Jan. 2015.

[36] E. Mingozzi, G. Tanganelli, and C. Vallati, "CoAP proxy virtualization for the web of things," in *Proc. IEEE 6th Int. Conf. Cloud Comput. Technol. Sci.*, Dec. 2014, pp. 577–582.

[37] A. Rahman and E. Dijk, *Group Communication for the Constrained Application Protocol (CoAP)*, document RFC 7390, Oct. 2014.

[38] M. Tiloca and E. Dijk, "Proxy operations for CoAP group communication," Internet-Draft Draft-Tiloca-Core-Groupcomm-Proxy-07, IETF Secretariat, Tech. Rep., Sep. 2022. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-core-groupcomm-proxy/00/bibtex/

[39] C. Gündoğan, C. Amsüss, T. C. Schmidt, and M. Wählisch, "Toward a RESTful information-centric web of things: A deeper look at data orientation in CoAP," in *Proc. 7th ACM Conf. Inf.-Centric Netw. (ICN)*, New York, NY, USA, 2020, pp. 77–88.

[40] C. Gündoğan, C. Amsüss, T. C. Schmidt, and M. Wählisch, "Group communication with OSCORE: RESTful multiparty access to a data-centric web of things," in *Proc. IEEE 46th Conf. Local Comput. Netw. (LCN)*, Oct. 2021, pp. 399–402.

[41] W.-K. Lai, Y.-C. Wang, and S.-Y. Lin, "Efficient scheduling, caching, and merging of notifications to save message costs in IoT networks using CoAP," *IEEE Internet Things J.*, vol. 8, no. 2, pp. 1016–1029, Jan. 2021.

[42] G. Tanganelli, C. Vallati, E. Mingozzi, and M. Kovatsch, "Efficient proxying of CoAP observe with quality of service support," in *Proc. IEEE 3rd World Forum Internet Things (WF-IoT)*, Dec. 2016, pp. 401–406.

[43] J. Mišic and V. B. Mišic, "Proxy cache maintenance using multicasting in CoAP IoT domains," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 1967–1976, Jun. 2018.

[44] F. Banaie, J. Misic, V. B. Misic, M. H. Yaghmaee Moghaddam, and S. A. Hosseini Seno, "Performance analysis of multithreaded IoT gateway," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3143–3155, Apr. 2019.

[45] 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.

[46] M. S. Lenders, C. Amsüss, C. Gündoğan, T. C. Schmidt, and M. Wählisch, "Securing name resolution in the IoT: DNS over CoAP," in *Proc. CoNEXT Student Workshop (CoNEXT-SW)*, New York, NY, USA, 2021, pp. 11–12.

[47] P. Van der Stok, C. Bormann, and A. Sehgal, *PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)*, document RFC 8132, Apr. 2017.

[48] S.-I. Choi and S.-J. Koh, "Use of proxy mobile IPv6 for mobility management in CoAP-based Internet-of-Things networks," *IEEE Commun. Lett.*, vol. 20, no. 11, pp. 2284–2287, Nov. 2016.

[49] G. Oikonomou, S. Duquennoy, A. Elsts, J. Eriksson, Y. Tanaka, and N. Tsiftes, "The Contiki-NG open source operating system for next generation IoT devices," *SoftwareX*, vol. 18, Jun. 2022, Art. no. 101089.

[50] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*, document RFC 6550, Mar. 2012.

[51] Y.-S. Yu, C.-C. Huang, and C.-H. Ke, "Analysis of maximum depth of wireless sensor network based on RPL and IEEE 802.15.4," in *IoT as a Service*, Y.-B. Lin, D.-J. Deng, I. You, and C.-C. Lin, Eds. Cham, Switzerland: Springer, 2018, pp. 234–239.

[52] *CC2538 Powerful Wireless Microcontroller System-on-Chip for 2.4-GHz IEEE 802.15.4, 6LoWPAN, and ZigBee Applications datasheet*, Texas Instruments, Dallas, TX, USA, Dec. 2012.

[53] S. Sciancalepore, G. Oligeri, and R. Di Pietro, "Strength of crowd (SOC)—Defeating a reactive jammer in IoT with decoy messages," *Sensors*, vol. 18, no. 10, p. 3492, 2018.

[54] P. Pinto, A. Pinto, and M. Ricardo, "End-to-end delay estimation using RPL metrics in WSN," in *Proc. IFIP Wireless Days (WD)*, Nov. 2013, pp. 1–6.

[55] P. Pinto, A. Pinto, and M. Ricardo, "RPL modifications to improve the end-to-end delay estimation in WSN," in *Proc. 11th Int. Symp. Wireless Commun. Syst. (ISWCS)*, Aug. 2014, pp. 868–872.

[56] S. M. M. H. Daneshvar, P. Alikhah Ahari Mohajer, and S. M. Mazinani, "Energy-efficient routing in WSN: A centralized cluster-based approach via grey wolf optimizer," *IEEE Access*, vol. 7, pp. 170019–170031, 2019.

ISMAEL AMEZCUA VALDOVINOS received the B.S. degree in computer science from Universidad de Colima, Mexico, in 2007, and the M.S. and Ph.D. degrees in computer science from Tecnológico de Monterrey, Mexico, in 2009 and 2013, respectively. He is currently a Professor and a Researcher with Facultad de Telemática, Universidad de Colima. His research interests include wireless sensor networks (WSN), the industrial Internet of Things (IIoT), vehicular networks, and software-defined networks (SDN).

PATRICIA ELIZABETH FIGUEROA MILLÁN (Member, IEEE) received the B.Sc. degree in computer science from Universidad de Colima, Mexico, in 2007, and the Ph.D. degree from Tecnológico de Monterrey, Campus Estado de México, in 2017. She was involved in developing application protocols for 6LoWPAN wireless sensor networks with Tecnológico de Monterrey, Campus Estado de México. She is currently a Professor with Tecnológico Nacional de México, Campus Colima, Mexico, at graduate and undergraduate levels. Her research interests include wireless sensor networks, the Internet of Things, emerging technologies, and software engineering.

JUAN ANTONIO GUERRERO-IBÁÑEZ received the Ph.D. degree from the Polytechnic University of Catalonia, Barcelona, Spain, in 2008. He is currently a Tenured Professor with the Faculty of Telematics, University of Colima, Colima, Mexico. He is also the Leader of the Networks and Telecommunications Research Group, Faculty of Telematics. His research interests include autonomous cars, vehicular communication, the Internet of Things, networking and quality of service provision, and wireless sensor networks.

RAMONA EVELIA CHÁVEZ VALDEZ received the degree in computer science and the master's degree in information technology management from Universidad Tecmilenio, Monterrey, Mexico. From August 2009 to April 2022, she held compatible administrative positions with Instituto Tecnológico de Colima, including a Career Coordinator, the Head of the Department of Systems and Computing, and the Division of the Graduate Studies and Research. Since 2017, she has been recognized as a Desirable Profile, and since November 2019, she has been a part of the ''Emerging Technologies'' Academic Group. She is currently a Professor with Tecnológico Nacional de México, Campus Colima. She is also a Professor at graduate and undergraduate levels. She has published articles in refereed and indexed journals and participated in the development of computer systems and technological prototypes. She has published scientific technical articles in peer-reviewed journals and conferences. Her research interests include software engineering and web development.