Secure and Authorized Client-to-Client Communication for LwM2M

Leandro Lanzieri
HAW Hamburg
leandro.lanzieri@haw-hamburg.de

Peter Kietzmann
HAW Hamburg
peter.kietzmann@haw-hamburg.de

Thomas C. Schmidt
HAW Hamburg
t.schmidt@haw-hamburg.de

Matthias Wählisch
Freie Universität Berlin
m.waehlisch@fu-berlin.de

ABSTRACT

Constrained devices on the Internet of Things (IoT) continuously produce and consume data. LwM2M manages millions of these devices in a server-centric architecture, which challenges edge networks with expensive uplinks and time-sensitive use cases. In this paper, we contribute two LwM2M extensions to enable client-to-client (C2C) communication: (*i*) an authorization mechanism for clients, and (*ii*) an extended management interface to allow secure C2C access to resources. We analyse the security properties of the proposed extensions and show that they are compliant with LwM2M security requirements. Our performance evaluation on off-the-shelf IoT hardware reveals that C2C communication outperforms server-centric deployments. First, LwM2M deployments with edge C2C communication yield a $\approx 90\%$ faster notification delivery and ≈ 8 times higher throughput compared to common server-centric scenarios, while keeping a small memory overhead of $\approx 8\%$. Second, in server-centric communication, the delivery rate degrades when resource update intervals drop below 100 ms.

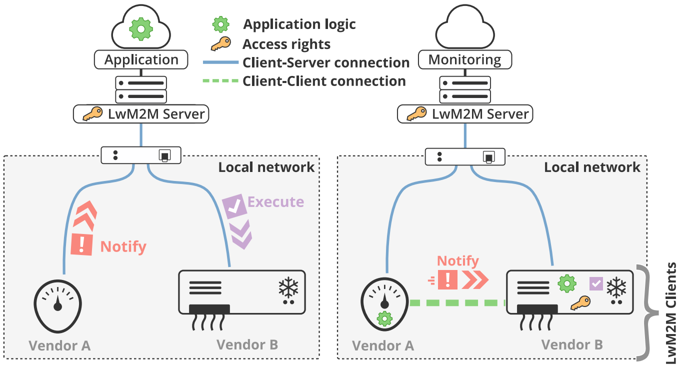
KEYWORDS

Internet of Things; management; security

1 INTRODUCTION

The constant expansion of the Internet of Things (IoT) led to an increased deployment of proprietary ecosystems to interconnect resource constrained "things" via the global Internet. Interoperability between an ever-increasing number of devices and vendors becomes paramount to avoid incompatibility silos. Lightweight Machine to Machine (LwM2M) [32] is a widely deployed protocol that provides device management features, service enablement, and interoperability across vendors, by defining an interaction model between LwM2M servers and clients, which operate on a uniform resource model.

In addition, the need for edge computing in IoT deployments [9] is rising, driven by high data volume and constraints such as intermittent uplink connection to the server and low latency requirements. In these scenarios, autonomous devices, executing distributed application logic are preferred. LwM2M client-to-client (C2C) communication enables this, by allowing edge devices to perform operations directly, while dynamic resource discovery facilitates application logic specification on runtime without human intervention. This requires a mechanism to dynamically distribute credentials and access rights to use resources of other clients securely. LwM2M lacks such a direct client communication because it only allows servers to initiate transactions. Information always flows through servers, see Figure 1a).



a) Server-Centric Communication b) Client-to-Client Communication

Figure 1: Different LwM2M deployment models. This paper introduces b) client-to-client communication.

This work contributes to the research agenda of "building an open, scalable, and secure Internet of Things" that is accessible to all parties via standards. In particular, we fill a design gap by introducing two extensions to the LwM2M core specification and providing an open source implementation on RIOT [2]. Our proposal enables a secure and authorized communication regime between clients, see Figure 1b). In detail, we make the following contributions:

(1) a third party authorization mechanism [17] that allows clients to dynamically request servers to gain credentials and access rights to resources hosted by other clients.
(2) new LwM2M objects and the extension of existing interfaces allowing clients to use them. Both enable direct communication between clients and allow IoT deployment scenarios in which upstream connectivity is limited and local communication preferred.
(3) a security analysis of our proposal, which shows that our approach still complies with LwM2M security requirements. Our analysis considers remote and local attackers separately and covers four common threats.
(4) an empirical performance analysis conducted on real hardware and in different deployment scenarios. Our proposal outperforms a server-centric solution in terms of delay (90%) and goodput (8×).
(5) open-source implementations of LwM2M client-to-client communication, which we make publicly available.

Our extensions are carefully designed such that they reuse existing protocols defined by the LwM2M core specification. This has two advantages. First, our approach seamlessly integrates into the LwM2M ecosystem and, second, it allows for re-utilizing operational knowledge [8] and code, which is particularly important when deploying constrained devices.

The remainder of the paper is organized as follows. Section 2 provides the necessary background about LwM2M. Section 3 and Section 4 introduce our proposal for C2C communication and third party authorization, respectively. Section 5 provides a comprehensive security analysis of our proposed extensions. Our experiments conducted on off-the-shelf IoT hardware are discussed in Section 6, together with results revealing the advantages of C2C communication. We present related work in Section 7 and conclude with a summary and outlook in Section 8.

2 BACKGROUND ON LWM2M

LwM2M [32] is a device management and service provision protocol that provides bootstrapping, access control, semantic data interoperability, and software update features. Clients run on constrained devices and register themselves to one or multiple LwM2M servers. Machine-to-machine applications, which usually run in the cloud, interact with clients via the servers. Server information and credentials are either pre-provisioned on a client, or bootstrapped by a dedicated LwM2M bootstrap-server.

Operation semantics and parameters are first defined generically and then mapped onto the lower layer. LwM2M supports three transport bindings: CoAP [30] (over UDP, TCP, SMS, and other Non-IP transports), HTTP, and MQTT. Interoperability is achieved by (*i*) a uniform resource model and (*ii*) a RESTful interaction model [6]. Objects are the building blocks of the resource model and specify how LwM2M clients group their hosted resources. Occurrences of these groups, called object instances, contain the resources that servers access. Multiple instances of a given object can exist on a client, each with different content but the same data structure. Servers interact with client resources via interfaces that define operations, most of which follow a request-response scheme.

Overhead. In spite of the features provided, LwM2M adds only relatively little overhead compared to CoAP-only applications. When analysing the processing time, our measurements reveal only 3.4% of the total $\approx$ 2570 μs required to compute a LwM2M Read operation (*i.e.,* a GET CoAPS request). In turn, the radio driver and the DTLS layer appeared as the dominant consumers, with 48.1% and 22.4% of the time respectively. When looking at the memory footprint (see Section 6.2), LwM2M represents less than 20% and 33% of total ROM and RAM requirements.

Security. In multi-server scenarios, access control to client resources is required. Each object instance hosted by the client has a corresponding access control object instance, that indicates the server access rights on it. These are organized in access control lists (ACLs), where each element of the list reflects one particular server. A single access control owner server manages the access rights and can modify policies for other servers dynamically.

LwM2M security requirements dictate clients and servers to authenticate each other, communication must be encrypted, and message integrity needs to be protected. The protocol specifies different ways of securing communications (TLS/DTLS [23] and OSCORE [28]), for which clients need information such as URIs to uniquely identify servers, credentials (pre-shared keys, raw public keys, or certificates) and configurations (*e.g.,* security mode, ciphersuite). Two objects organize this information: the server and the security objects, which together reflect a server account. After establishing secure communication, clients register to servers with a unique endpoint name. In contrast to the URI, the endpoint name is independent from the transport binding.

Table 1: Overview of features that are required/provided (●), partly required/provided (◑), or not required/provided (○) in different IoT scenarios/paradigms.

Scenario / Paradigm	Low latency	High bandwidth	Steady connection	Sensitive data	Long range	Local actuation
Smart metering	○	○	○	◑	◑	○
Smart farming	○	○	○	○	●	●
Disaster first response	○	○	○	○	●	●
Smart home	○	◑	●	●	○	●
Smart transportation	◑	◑	○	◑	○	●
Industrial emergency	●	●	◑	○	○	●
Control systems	●	●	●	◑	○	●
server-centric	○	◑	○	◑	●	○
client-to-client	●	●	●	●	◑	●

Shortcomings. Despite the wide deployment of LwM2M, its server-centric paradigm presents shortcomings in certain types of use cases. Table 1 shows typical IoT scenarios, together with usually required features. Applications such as smart agriculture and the tracking of—and interaction with—livestock present deployments at remote locations. On the one hand, they require long-range communication (*e.g.,* to report animal vitals), on the other hand, animals need to interact with local devices (*e.g.,* gate control, food dispensing). LoRaWAN appears as a popular long-range technology choice for this type of applications, but due to its long on-air times it applies strict duty cycles. This quota is easily exhausted by deployments which involve control systems, as all information flows through servers even when—ultimately in many cases—a neighbour node is the recipient.

Similarly, industrial deployments involving closed-loop control systems have low latency requirements (10 − 100 ms delays [10]). Such systems cannot afford a server-centric information flow due to its additional delays. Instead, these scenarios would benefit from distributed applications based on direct local communication between LwM2M clients, which reduces latencies, while still being monitored by central servers. Typically, the distributed logic is installed on the nodes after a resource discovery or a commissioning process. Instead of a central application, nodes follow business rules (*e.g.,* "whenever the light switch of room A is pressed, notify light bulb group 2"). As they only require knowledge of a subset of the whole application, this paradigm is scalable when adding new devices. Changes in the logic are usually performed by management tools from the central servers.

Another example are lossy IoT networks present in smart transportation containers. Given their mobile nature, they have intermittent Internet access. The constant need for communication with the managing server requires to be permanently online. A similar situation is faced by disaster first-response devices, which usually have to build ad-hoc delay-tolerant networks. In these environments a connection to a central server is sporadically available. As an alternative, the deployment of autonomous devices which can communicate with one another would allow keeping local functionalities working even if upstream connectivity is lost.

Even in scenarios with steady connectivity and high bandwidth, a central cloud involvement may raise privacy issues. Constantly utilizing central servers to store and analyse sensor values and to control domestic appliances can reveal usage patterns and disclose personal information. Vendors could leverage LwM2M, and install devices that interact within the household following user-installed policies.

Summarizing, LwM2M acts as a semantics-unifying layer that enables machine-to-machine applications on the cloud not only to manage IoT devices, but also to implement business logic in a vendor-independent fashion (*e.g.,* reading sensed data and activating actuators in consequence), while adding relatively small overheads. In parallel, there is a clear need for direct node-to-node communication over a variety IoT deployments. Consequently, we propose to extend LwM2M to allow clients to operate on each other resources, thus maintaining the benefit of its vendor interoperability and service enablement features.

3 DIRECT LWM2M CLIENT COMMUNICATION

We now want to derive generic requirements for secure C2C communication in the common use cases that were analysed in the previous Section 2. As data flowing between nodes is mostly sensitive, devices are expected to establish a secure communication channel (*i.e.,* providing confidentiality, integrity, and replay protection to the messages), and to authenticate each other prior to any data exchange. It is also desirable that resource owners are able to establish access policies to the resources (which may dynamically change at runtime), thus, some access control mechanism should be in place. Additionally, to cope with changing deployments (*e.g.,* new appliances added to smart homes), flexibility is desired. This implies that nodes need to make use of trusted and authenticated services to securely discover resources of interest, and to obtain the required credentials and rights to access them.

With these requirements in mind, we enable C2C communication in the case of LwM2M, by re-utilizing existing interfaces defined in the core specification, namely (*i*) the device management and service enablement interface, and (*ii*) the information reporting interface. To avoid ambiguity when referring to LwM2M clients utilizing the interfaces, we define: (*i*) *hosting clients* host resources on which operations are performed, and (*ii*) *requesting clients* request the operations on said resources. It is worth noting, however, that nodes will likely play both roles throughout their lifetimes. Using the interfaces requires a secure communication channel, hence, clients need to establish secure transports among each other and to have adequate access rights. For this, we introduce a new *LwM2M client account* to organize the information clients need about each other, including security credentials, URIs and connection configurations. Similarly to LwM2M server accounts, LwM2M clients hold one account per client with which they communicate. Three newly introduced LwM2M objects organize communication and access: (*i*) the *client object*, (*ii*) the *client security object* – a *LwM2M client account* consists of instances of these two objects, and (*iii*) a *client access control object* to determine which operations a requesting client is allowed to perform.

3.1 Client-to-client Objects

To ease code re-utilization and lower the implementation overhead, our newly introduced objects share many resources with existing objects, used to establish client-server communication.

Client Object. An instance of this object holds parameters related to the communication with other clients, including the client ID (an internal reference), the client endpoint name, the account lifetime, default values for observation periods, and the communication binding. To mitigate a potential elevation of privilege when access revocation messages sent by the server do not correctly arrive to the hosting client, the lifetime parameter in this object determines for how long a requesting client account is valid. After expiration, the hosting client disables it, closes existing connections to the requesting client and ignores subsequent operation attempts. Hence, requesting clients access needs periodic refresh, unless disabled by configuring the lifetime to 0.

Client Security Object. An instance of this object holds the URI of a specific client, security configurations, and the DTLS credentials or a reference to an object holding OSCORE credentials, depending on which secure transport is used. Resources of this object exhibit the exact same identifiers and semantics found in the standard LwM2M security object. It is worth noting, that only servers are allowed to operate on client security object instances, to install and modify credentials in a dynamic fashion during the device lifetime. This allows for additional flexibility in contrast to deployments with static configurations. These object instances can be created and modified by servers through the device management and service enablement interface, or bootstrapped by a LwM2M bootstrap-server.

Client Access Control Object. An instance of this object holds the actual access rights which allows hosting clients to keep track of the permitted operations to requesting clients. Each instance is associated to a particular instance of any other object hosted by the client, and indicates an *access control owner* that is the responsible server to manage access rights for this object instance. An instance contains an ACL that specifies which operations each requesting client is allowed to perform on the associated instance. This is indicated using flags (*e.g.,* read, write). For C2C access, we add an explicit 'discover' access flag that controls whether a requesting client can explore resource attributes of an object, increasing the control granularity. This is in contrast to regular server based access, where it is always allowed to discover available objects. Only servers can modify requesting clients access rights.

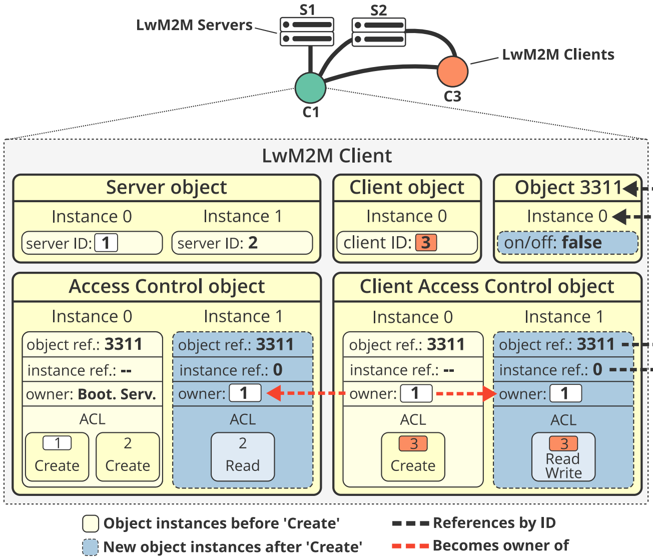


Figure 2: Access control objects in a hosting client, before and after a requesting client instantiates the light control object (ID 3311).

3.2 Extended Interfaces and Access Control

Requesting clients can perform all operations defined by the device management and service enablement interface and the information reporting interface, provided they have the required access rights. They use these interfaces to access resources in hosting clients, via operations like 'read', 'write' and 'create'. Resources may be accessible to multiple requesting clients, and concurrency should be handled the same way as for multiple-server access. As per the LwM2M specification, atomicity is required when performing a 'Write-Composite' operation.

All access control rules that apply to servers also do to clients. This means that for each requesting client explicitly authorized to perform an operation on a resource, a corresponding ACL should be instantiated on the hosting client, otherwise the default access is granted. The assignment of access control owners after a 'create' operation, however, differs for C2C operations. Whenever an object is instantiated via a 'create' operation, a hosting client additionally creates new instances of the access control and client access control objects, to track server and client access rights respectively, for the new object instance. In contrast to regular server operation, a requesting client that creates a new object instance does not become its access control owner, instead, the owner is the server indicated in the client access control object instance which authorized the 'create' operation.

Figure 2 illustrates a subset of the object instances on a hosting client C_1, before (yellow boxes) and after (blue boxes) a requesting client C_3 performs a 'create' operation on the light control object (ID 3311). C_1 hosts two server accounts (S_1 and S_2 with IDs 1 and 2), and one client account for C_3. Instance 0 of the access control object holds access rights for the light control object 3311, and the ACL contains 'create' access rights for both servers, whereas the LwM2M bootstrap-server is the access control owner. Instance 0 of the client access control object holds 'create' access rights for C_3, pointing to S_1 as the access control owner. In both cases, as the 'create' operation is performed on an object and not on a particular object instance, only the *object reference* resource is set, and not the *instance reference.* When C_3 creates a new instance of the light control object on C_1, C_1 also creates locally new instances of both access control objects. Instance 1 of the access control object indicates that S_1 (*i.e.,* the server which allowed the instantiation) is the access control owner for the new instance of the light control object, and it grants 'read' access to S_2. In turn, instance 1 of the client access control object holds S_1 as access control owner respectively, and provides C_3 with 'read' and 'write' access rights.

4 THIRD PARTY AUTHORIZATION OF LWM2M CLIENTS

Clients are authorized by LwM2M servers that handle access rights and credential distribution. LwM2M servers are considered trusted third parties to the clients, as the LwM2M specification requires mutual authentication. We introduce two new mechanisms to enable third party authorization of LwM2M clients. *Owner server hints* allow clients to discover the responsible server which can grant access to a resource. The *access request interface* is utilized by clients to request specific access rights and credentials to this responsible server. If the server accepts the request, it distributes the access rights and credentials through the regular device management interface. Figure 3 presents the complete sequence diagram of an unauthorized requesting client that requests access rights of a resource on a hosting client, utilizing the server as a third party. The server distributes credentials, to establish a secure C2C communication, and access rights, to authorize subsequent operations among clients. The credentials and access rights distribution is usually performed infrequently, this is, before the initial C2C interaction, but its frequency ultimately depends on the application requirements and security policies.

4.1 Owner Server Hints

In multi-server LwM2M deployments requesting clients need to find out which server is in charge of processing a resource access request for a given node. As LwM2M makes no assumptions as to whether servers belong to the same organization (*i.e.,* they may not communicate with one another), this knowledge must be dynamically acquired by the nodes. To achieve this functionality in our LwM2M extension, the hosting client provides server information contained in the owner server hints. It is worth noting, that this discovery mechanism can be avoided in case there is out-of-band information that a deployment has a single server.

In a first step, the requesting client sends an *unauthorized resource request* to the hosting client and attempts to perform an operation on a specific resource ①. This is commonly done on unsecured transport. We further analyse the security implications in Section 5.4. The initial contact information is assumed to be learnt by a discovery mechanism, such as a resource directory [1].

A hosting client that receives a request on an unsecured channel from the unknown requesting client rejects it. It responds with an unauthorized status code, and includes the owner server hints

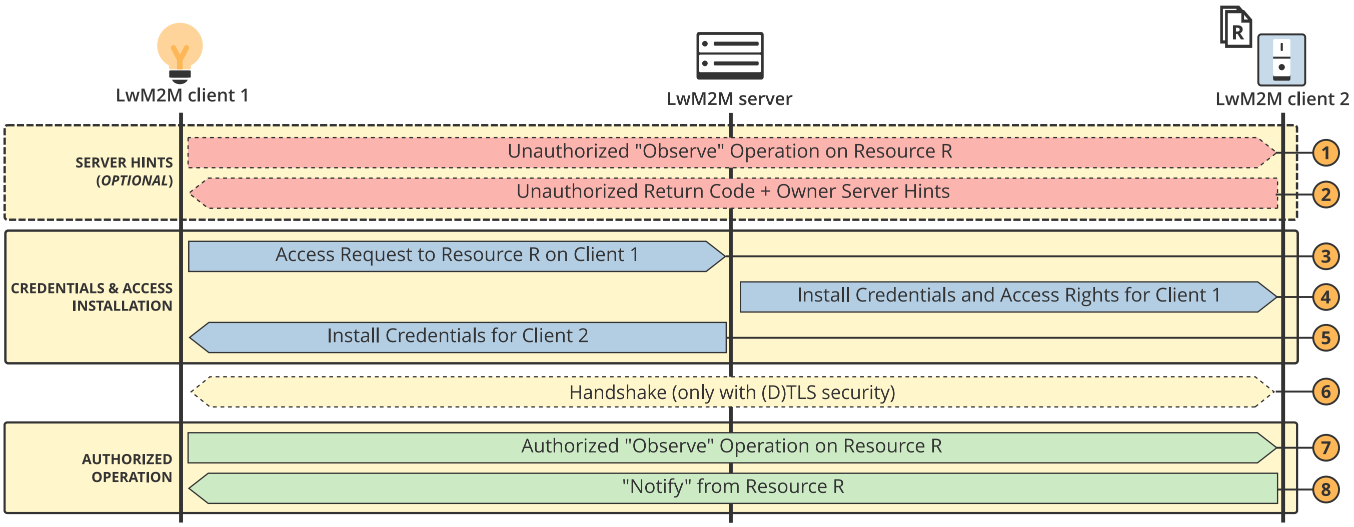

LwM2M client 1
LwM2M server
R
LwM2M client 2
SERVER HINTS (OPTIONAL)
Unauthorized "Observe" Operation on Resource R
1
Unauthorized Return Code + Owner Server Hints
2
CREDENTIALS & ACCESS INSTALLATION
Access Request to Resource R on Client 1
3
Install Credentials and Access Rights for Client 1
4
Install Credentials for Client 2
5
Handshake (only with (D)TLS security)
6
AUTHORIZED OPERATION
Authorized "Observe" Operation on Resource R
7
"Notify" from Resource R
8

Figure 3: Request and installation of LwM2M client access rights and authorized C2C communication.

which contain the URI of the server that owns the resource, responsible to grant access ②. The response optionally contains multiple server URIs that point to separate credential- and access management servers. To prevent disclosing information about whether a resource is hosted by the client, returned codes are kept generic and at least one default owner LwM2M server is included in the response, until enough trust exists with the requesting client and a secured transport is established.

A requesting client must verify that the received server URI represents a known and trusted server with which a secure communication has already been established (*i.e.,* on registration). The existing trust relation to the server is essential, since unsecured server hints could have been tampered by an attacker.

4.2 Access Request Interface

We introduce a new interface that allows clients to request servers for access rights and credentials. The interface consists of a single operation initiated by the client: the *access request.* A client includes the intended access rights, the endpoint name of the hosting client it attempts to access, and a flag to indicate the need for credentials to establish secure communication ③. Utilizing *endpoint client names* [32, Sect. 7.3.1] abstracts from the underlying protocol and allows to identify clients across LwM2M transport bindings. As a counterexample, URIs reveal different structures across transport bindings which complicates interoperability in heterogeneous deployments. It is noteworthy that this requires unique endpoint names for clients that participate in a network.

After reception of the access request, the server verifies if the client is entitled to obtain the requested access rights. This decision depends on the application-logic. Commonly servers follow preinstalled access policies or query the resource owner. On acceptance, the server generates credentials and creates client accounts on both clients. Thereafter, it modifies the client access control object of the hosting client to enable the requesting client to access to the required resource ④ & ⑤. Once clients are in possession of the credentials, if they are using (D)TLS-based security they perform the handshake ⑥, and if they use OSCORE the derive locally the corresponding security contexts. Finally, authorized client-to-client interactions can be performed ⑦ & ⑧.

We present a mapping of the access request interface onto CoAP transport, however, our approach naturally extends to other LwM2M transport protocols. A requesting client performs an access request operation sending a `POST` request on the path */ac* to the LwM2M server. The operation has two parameters passed as URI query strings: (*i*) *ep* is mandatory and holds the endpoint client name of the hosting client. (*ii*) *c* is optional and indicates if the requesting client requires credentials to be installed in order to initiate secure communications with the hosting client.

The requested access rights are included in the payload of the request, encoded in LwM2M CBOR [4] format. A message can contain multiple access requests to multiple objects and instances.

5 SECURITY ANALYSIS

Throughout the design of the proposed LwM2M extensions we followed an iterative threat-driven approach, by performing a methodical four-steps analysis of the security and privacy aspects of the extensions: (*i*) asset identification and security properties assignment (Section 5.1), (*ii*) attacker model building (Section 5.2), (*iii*) attack surfaces analysis (Section 5.3), (*iv*) threat analysis, which lead to protocol improvements. In this section, we present each step of the analysis and the outcome of the threat model, together with mitigations.

5.1 Assets and Security Properties

In this step we enumerate the resources with security properties that should be preserved, and that might be targeted by attackers. We use the well known CIA(A) [21] descriptors as the space of

Table 2: Threat model of the LwM2M client-to-client communication and third party authorization extensions.

No.	Threat description	Asset (§5.1)	Adversary (§5.2)	Surface (§5.3)	CIAA (§5.1)	Mitigation
T0	**Information Disclosure**: Observing unprotected operations attackers can learn which resources may be hosted and which are of interest.	App. config. / Client resources.	Local	Unauthorized request / Server hints.	CO	Sensitive content should be avoided on unauthorized requests.
T1	**DoS**: Open DTLS port on a client is used for message amplification to perform a denial of service attack.	Operational resources.	Remote	Open client DTLS port.	AV	The DTLS server sends out a `HelloVerifyRequest` message during handshake.
T2	**Elevation of privilege**: A server cannot revoke client access to a resource, elevating its privilege, because incoming communication is jammed.	Hosted resources.	Local	Extended device management interface	CO	Lifetime parameter in the client security object defines maximum period of access.
T3	**Tampering**: A requesting client receives invalid server hints, which might point to a rogue or compromised server.	Owner server URI.	Local	Server hints	IN	LwM2M clients only consider for access requests LwM2M servers to which they are already registered.

security properties: Confidentiality (CO), Integrity (IN), Availability (AV), and Authenticity (AU).

Application configuration (CO, IN). Considers the node behaviours and their relations with other clients and servers, *e.g.,* the interest of client C_3 in resource R hosted by client C_1.

Owner server URI (IN). Identity of a resource owner, *e.g.,* the server that assigns access rights. It is worth noting, however, that confidentiality of the owner server URI is not expected, as it is usually sent over un-protected communication channels between clients.

Client access rights (CO, IN). Access grants of requesting clients to resources on hosting clients, *e.g.,* permissions that the owner server grants a client C_3 on resources hosted by client C_1.

Client credentials (CO, IN, AU). Key material used for secure communication, *e.g.,* the pre-shared key of a client.

Hosted client resources (CO, IN). LwM2M resources on a hosting client, *e.g.,* the status of a light control object.

Device operational resources (CO, IN, AV, AU). The preservation of networking-, computational-, battery-, and memory resources, *i.e.,* a device that hosts/operates on a resource R remains in operation.

In addition to preserving the aforementioned asset properties, we must consider the LwM2M security requirements defined in [33, Sect. 5.1], which apply across all transport bindings. They state that (*i*) messages must be replay protected, (*ii*) requests and responses must be bound, (*iii*) freshness must be verifiable for certain operations, (*iv*) secure fragmentation must be supported, (*v*) data from clients and servers must be encrypted and integrity protected and (*vi*) clients and servers must be authenticated prior to data exchange.

5.2 Attacker Model

Our attacker model assumes that adversaries are not in possession of valid credentials required for mutual authentication with the server or clients. We identify two attacker groups:

Remote attackers access nodes remotely through the network. They may be capable of eavesdropping messages transmitted between clients and servers but have no access to the local client network. These attackers try to learn internals and use this information to compromise a device under attack. Therefore, they impersonate hosting clients, requesting clients or LwM2M servers by sending malicious messages via the LwM2M interfaces. Remote attackers usually leverage protocol or software vulnerabilities to manipulate sensitive processing tasks.

Local attackers have direct access to the local network. Additionally to the capabilities of a remote attacker, local ones may intercept, modify and replay messages among clients on the local area network (usually wirelessly). Attackers who apply advanced hardware access techniques to manipulate secret information directly from the silicon are not in the scope of this analysis.

On peer-to-peer attacks. LwM2M deployments are commonly composed of heterogeneous embedded devices with specific capabilities, resources and tasks that are all part of a single administrative domain. Even though we analyse direct communication between LwM2M clients, these networks are not equivalent to traditional peer-to-peer (P2P) systems, which run on independent responsibilities. Consequently, we consider typical P2P attacks out of the scope for this analysis. As an example, rational attacks would not apply because nodes participating in the LwM2M deployment are naturally cooperating (*i.e.,* they expose their resources and share via the LWM2M server).

5.3 Attack Surfaces

Attack surfaces are potential entry points that can be leveraged by adversaries to perform attacks against the system assets. Here, we analyse the attack surfaces of the LwM2M extensions.

Open DTLS port on client. To allow secure DTLS-based transport among clients, these need to accept incoming session handshakes, similarly to LwM2M servers. It is noteworthy, however, that OSCORE [28] as the alternative secure transport requires no handshake.

Extended device management interface. Not only servers, but also other clients, can access client resources. The increased number of accessing devices, which can be potentially compromised, enlarges this attack surface compared to server-centric LwM2M deployments.

Unauthorized resource request. In order to learn which server owns access rights to a resource, requesting clients may perform unauthorized requests, which often occur over non-secured transports.

Server hints. In response to an unauthorized request, a hosting client responds with the owner server hints, which may be sent in clear text prior to establishing a secure transport.

5.4 Threat Model

Table 2 presents a series of threats that we identify based on the former analysis of assets, adversaries, and attack surfaces. To classify the threats we follow the STRIDE framework [14] which defines six categories of security threats: Spoofing identity (S), Tampering with data (T), Repudiation (R), Information disclosure (I), Denial of service (D) and Elevation of privilege (E).

T0 describes a threat in which an attacker, who eavesdrops an unprotected unauthorized resource request, acquires information about a possible resource hosted by the LwM2M client and the interest of the requesting client on it. An attacker learning this information may raise a privacy issue, thus, requesting clients should avoid sending sensitive payload on unprotected unauthorized request (*e.g.,* only perform read operations), and hosting clients should keep response codes generic. In cases where particular resource URIs must not be revealed, a requesting client can perform an initial request to a non-sensitive resource to get the owner server hints, from which it can request initial credentials. After the secure channel has been established, the sensitive request can be performed.

T1 is identified as a threat introduced by the extensions, because in a server-centric LwM2M deployment clients would not play the server role during a DTLS handshake, and could be configured to simply ignore them, when using DTLS-based security. One possible mitigation is to use the `HelloVerifyRequest` message with a stateless cookie, making the usage of spoofed IP addresses for DoS attacks difficult. Another strategy is to establish a limit for incoming requests.

T2 and T3 are mitigated by design in the proposed LwM2M extensions. T2 considers a situation in which an access right revocation message does not arrive to the hosting client, either because an attacker blocks it or because of the lossy nature of the networks. This results in an elevation of privileges for the requesting client, who keeps the access beyond the intended period. By assigning a lifetime to the distributed credentials the impact of such an attack is reduced, at the cost of an increased traffic generated by periodic authorization requests. T3 considers the case of invalid server hints, which are sent to a requesting client (*e.g.,* injected by a local attacker) and could point to a compromised or rouge LwM2M server. A requesting client should only consider known servers as valid owners to request access credentials.

Now we analyse the compliance of the extensions with the LwM2M security requirements (listed in Section 5.1). As C2C communication is not considered in the specification, we give the requirements a broad scope to consider data exchanged among clients as well. Requirements (*i*) through (*iv*) are fulfilled by the underlying transport bindings, as we utilize the same ones as in standard LwM2M deployments for C2C communication. Messages exchanged over the extended device management and access request interfaces are encrypted and integrity protected (requirement (*v*)) by both OSCORE and DTLS. Moreover, these protocols also provide mutual authentication (requirement (*vi*)) to clients when performing operations on the extended interface. The only messages sent prior to mutual authentication, and that are not encrypted nor integrity protected, are the initial unauthorized resource request and the server hints. We have already described the impact of this and provided mitigations to reduce the exposure through this surface. Only the URIs of the requested resource and LwM2M servers would be sent over unprotected transport, and no critical information would be disclosed. In the case when a particular application cannot afford such disclosure, LwM2M clients can be pre-provisioned with credentials and still establish a C2C communication. Thus, we conclude that the extensions comply with the LwM2M security requirements.

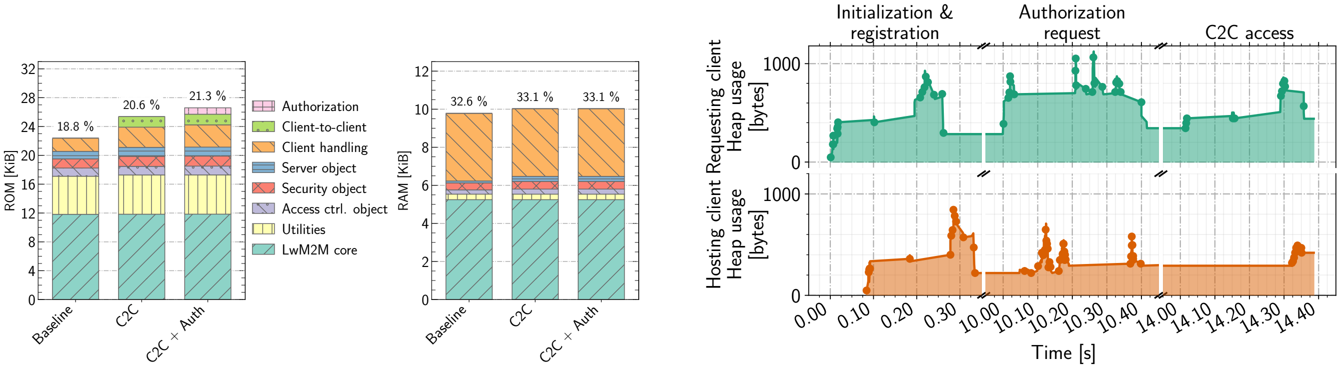
6 PERFORMANCE EVALUATION

In this section, we compare our proposal, client-to-client communication and authorization, with the current client-server architecture in LwM2M. We analyze memory consumption, transmission delays, and maximum goodput, based on experiments on real hardware. Our experiments are guided by the use cases of edge processing and distributed application logic in single- and multi-hop deployments.

6.1 Experiment Setup

Hardware and Software. We conduct our experiments by deploying real implementations on the FIT IoT-LAB testbed, using off-the-shelf class 2 IoT devices [3] that feature ARM Cortex-M3 MCUs running at 72 MHz, with 64 KiB of RAM and 512 KiB of ROM, equipped with IEEE 802.15.4-compatible Atmel AT86RF231 transceivers.

The firmware that runs on the constrained IoT devices is based on the operating system RIOT, version `2021.04`. The LwM2M client is implemented using Wakaama version `1.0`, the heap, needed by Wakaama, with the Two-Level Segregated Fit (TLSF) allocator. In terms of security support, we consider both OSCORE (the current main branch of uOSCORE) and DTLS (current development branch



(a) ROM (left) and RAM (right) requirements of LwM2M client modules in a default configuration (Baseline), with client-to-client (C2C) and third party authorization (Auth) extensions. Relative values relate to the complete firmware image size.

(b) Heap usage of a LwM2M requesting client (top) and a LwM2M hosting client (bottom) during initialization and registration, authorization request, and C2C access.

Figure 4: Memory requirements of LwM2M client modules (4a) and heap usage by requesting client and hosting client (4b).

of tinyDTLS). On the LwM2M server side, we use the implementation Leshan, version `1.3.1`, which runs on a Dell PowerEdge R6525 server with two AMD EPYC 7702 processors and 512 GB of RAM.

Configuration and Startup. In all DTLS and OSCORE deployments, we use AES in CCM mode with a 128-bit key, an authentication tag of 8 bytes, and pre-shared keys (PSK) suites. The testbed hardware does not feature a hardware random number generator. To generate the initial CoAP message ID, we use the SRAM-based physically unclonable function (PUF) [13] as entropy source to seed the RIOT pseudo random number generator.

Deployment Scenarios. To compare the performance of C2C versus server-centric communication, we deploy multiple topologies running three scenarios: (*i*) server-centric, (*ii*) C2C using DTLS security, (*iii*) C2C using OSCORE security. In all scenarios, a hosting client produces a 5-bytes data every second, which should reach a requesting client. We focused our experiments on the observation of resource updates, as it is, for the typically deployed low-power sleepy devices, a more common approach than constant polling. In scenario (*i*), a centralized application interacts with the LwM2M server via an HTTP API and observes the sensor resource, writing new values to another client upon update notifications. In scenarios (*ii*) & (*iii*), the application logic is decentralized, *i.e.,* the requesting client observes the sensor resource in the hosting client, thus, receiving periodic notifications directly.

6.2 Firmware Size

Figure 4a presents memory requirements for three configurations of the LwM2M client firmware: (*i*) baseline (no extensions), (*ii*) C2C extension enabled, and (*iii*) C2C + Auth extensions enabled. Measurements are separated into ROM, which considers the code segment and variables initial values (`text` + `data` segment), and RAM, which includes (un-)initialized global variables (`bss` + `data` segment). ROM and RAM consumption of the LwM2M core module remain unaffected across configurations and require ≈ 11 KiB ROM and 5 KiB RAM, including the RAM memory pool used by the heap allocator. Similarly, utilities modules are fundamentally constant, however, our Auth extension adds 460 bytes of ROM for introducing CBOR encoding which would also benefit the pure Wakaama baseline implementation.

Due to the functional similarities between the newly introduced LWM2M objects and their existing counterparts, we are able to reutilize most of the code (*i.e.,* no extra C object files are compiled for the new LwM2M objects), only with slight size increments to accommodate the extra logic. The size of security, access control, and server objects increase by ≈ 100 bytes, 110 bytes, and 200 bytes in ROM, and ≈ 40, 40 and 170 bytes in RAM, due to the additional states to handle client security, client access control, and client objects. The client handling module is responsible for connections and requesting client operations, which adds 970 bytes in ROM within the C2C extension, and additional 310 bytes with the Auth extension, for additional logic of connection handling and client credential management. The C2C and Auth modules use 1230 and 910 extra bytes of ROM, while no extra RAM in needed, since connection states are stored in the security and server objects. Memory requirements that correspond to the OS, network stack, and drivers have a constant memory offset across configurations (not displayed in Figure 4a), however, we indicate the percentage of our LwM2M client modules to the total firmware size. In summary, the LwM2M proportions conform ≈ 20 % of the ROM and ≈ 33 % of the RAM in comparison to the total image, which is around 125 KiB in ROM and 31 KiB in RAM.

We conclude that the pure overhead of our C2C extension increases total ROM image size by only ≈3.3 % and RAM by 0.9 %, while the Auth extension requires additional ≈ 5.0 % of ROM and no extra RAM. This is in line with our goal to maximize code reutilization.

6.3 Heap Usage

Figure 4b illustrates heap usage on two clients, separated into three phases: Initialization & registration, authorization request, and C2C access. The top half of the figure corresponds to a LwM2M requesting client, and the bottom half to a LwM2M hosting client.

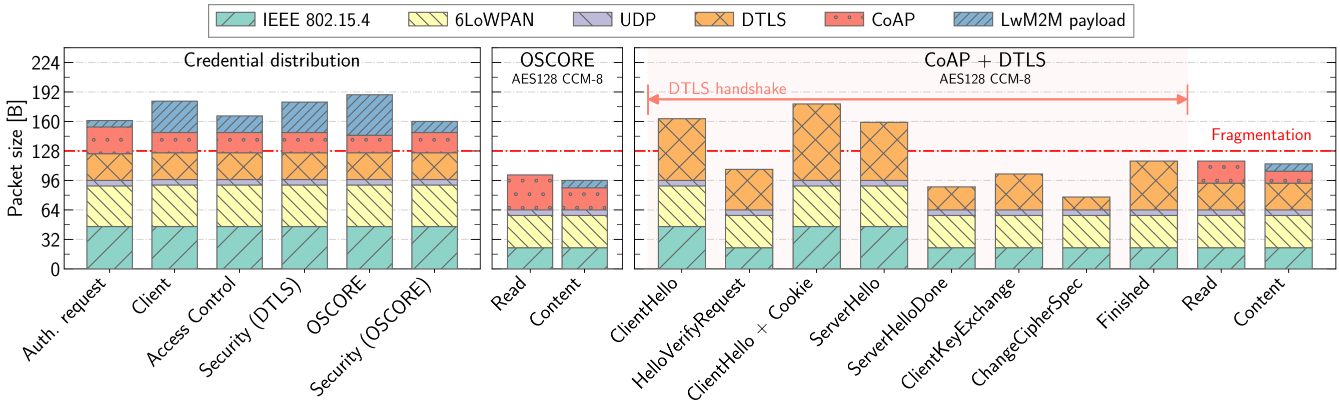


Figure 5: Packet structures of credential distribution and client-to-client read operation, using DTLS and OSCORE.

We exclude the LwM2M server memory consumption, since it does not suffer from memory constraints. Overall, heap requirements range from ≈ 300–1100 bytes on constrained nodes.

The usage pattern during initialization & registration (00.00–00.35 seconds) is similar for both clients, however, the requesting client requires additional ≈ 80 bytes to allocate a structure that holds initial information about the LwM2M hosting client. This memory is required for every hosting client to which the requesting client connects. In contrast, the host has no prior knowledge of a requesting client. At 10 seconds, the requesting client initiates an authorization request to the server which requires memory for the state of a CoAP request. A spike in the graph at 10.03 seconds corresponds to temporary memory used for the CBOR message encoding. Between 10.10–10.38 seconds, spikes in both graphs correspond to read and write operations on the resources, performed by the server, which installs credentials and access rights. After the authorization request has finalized, at 10.40 seconds, the requesting client de-allocates its state and memory usage returns to a new baseline slightly higher than the previous one, because of the new client information that Wakaama needs to allocate internally. The requesting client starts a DTLS handshake with the hosting client at 14.02 seconds. An increment of ≈ 100 bytes reflects an active DTLS connection, that has to be allocated and kept for every host, until the connection closes. Finally, at 14.28 seconds, a C2C read operation allocates ≈ 200 bytes state on the requesting client. Upon reception, at 14.34 seconds, the hosting client allocates heap memory for the new DTLS connection (≈ 100 bytes similarly to the requesting client on handshake) and utilizes temporary memory to format the response message.

6.4 Packet sizes

Now we dissect the packets that constitute the authorization request flow and a C2C read operation, using both DTLS and OSCORE security, as shown in Figure 5.

The maximum data unit size of the IEEE 802.15.4 2.4 GHz physical layer is 127 bytes. Considering the sizes of the 8-bytes destination and source hardware addresses, 2-bytes frame control field, 1-byte sequence number, 2-bytes personal area network (PAN) ID, and 2-bytes frame check sequence (FCS), the MAC header adds to 23 bytes, which allows up to 104 bytes to be transmitted by the upper layers. A total of 41 bytes are used by the 6LoWPAN layer, as it requires 2 bytes to accommodate the IP header compression, 1 for the hop limit, 32 for both IPv6 addresses, and 6 to encode the compressed UDP header.

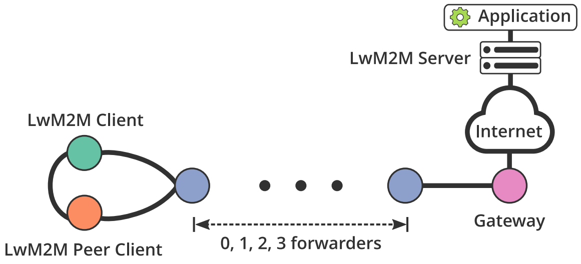
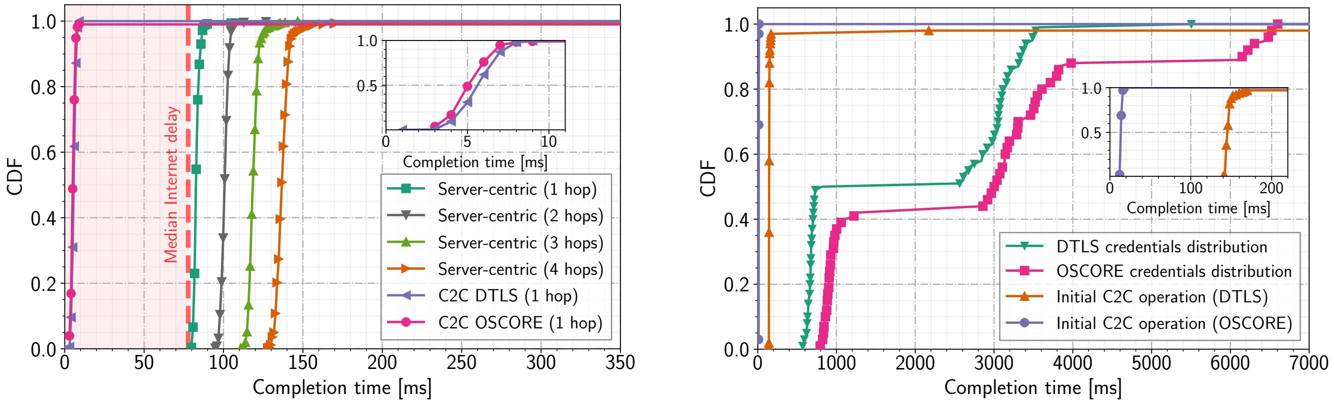


Figure 6: A variable number of forwarders to determine the impact of multihop on server-centric LwM2M deployments.

The authorization request is for read access on one object instance, encoded in CBOR as detailed in Section 4.2. During credential distribution the authorization request, client and access control object instantiation messages are common across transports. On the other hand, the content of the security object instantiation message only holds credentials when using CoAP over DTLS (DTLS security), as OSCORE credentials are distributed separately in its own OSCORE object. All messages in this process trigger 6LoWPAN fragmentation as they are bigger than the physical data unit. The object instantiations and the content messages are encoded in LwM2M TLV, because of the current support in Wakaama. The OSCORE read and content packets are respectively 15 and 21 bytes smaller than the DTLS counterpart, due to the bigger size of the DTLS record layer compared to the OSCORE header.

6.5 Time to resource update

We analyse the times between the generation of a new resource value in a hosting client and its arrival at a requesting client on all deployment scenarios. The topology for this experiment consists of LwM2M clients connected to the LwM2M server through a gateway and a varying number of intermediate forwarder nodes, as shown



(a) Notification arrival times in server-centric and C2C deployments.

(b) Authorization request and initial C2C operation completion times.

Figure 7: Temporal distributions of notification arrival (7a) and authorization request followed by first C2C operation (7b).

in Figure 6. We vary forwarders to quantify the impact of extra hops when using the server-centric LwM2M deployment. Figure 7a shows the results for C2C and server-centric communication for different amount of hops between the gateway and the clients. To reduce the impact of the variable delay introduced by the Internet connection between the gateway and the LwM2M server, we first subtracted this time from the server-centric measurements, and then offset them by the median Internet delay of across all experiments (≈ 78 ms), which was measured by timestamping the packets from and to the gateway. We observe a reduction of ≈ 90% in the notification delay when using C2C communication compared to the server-centric single-hop scenario. The extra time required in the latter scenario is explained by the overhead of the 'write' operations from the LwM2M server to the client and the delays of the connection between the gateway and the server. The impact of additional hops is of ≈ 15 ms delay per hop when communicating through the server, which is consistent with typical 6LoWPAN times.

Now we consider a second setup where the requesting client first performs an authorization request to the LwM2M server and then a C2C read operation. The entire credential distribution is composed of 5 operations: (*i*) authorization request, instantiation of (*ii*) client object, (*iii*) client security object and (*iv*) client access control object in hosting client, (*v*) update of the client security object instance in requesting client.

Figure 7b shows that ≈ 50% of the DTLS credential distributions are completed in less than 1 second without needing CoAP retransmissions, while within 4 seconds most of the DTLS credentials distributions are successful. We observe that distributing OSCORE credentials takes slightly longer and needs a second retransmission for ≈ 15% of the cases, due to the instantiation of one more object compared to DTLS credentials (the OSCORE object). We can observe a stair-case pattern caused by CoAP retransmissions that reflects the default configuration of a 2-seconds ACK timeout [30]. Once credentials are distributed, the initial C2C read operation using OSCORE takes ≈ 10 ms, while the initial DTLS handshake raises this time to between 140 and 160 ms.

Next, we look at the effective goodput achieved across deployment scenarios, summarized in Figure 8. For this experiment, a hosting client sends 5.000 notifications at varying intervals, configuring the radios at 250 Kbit/s and 2000 Kbit/s (minimum and maximum available values respectively). The resulting goodput measurements are depicted using box plots, next to the theoretical optimum (dashed lines). For each interval the rate of successfully delivered notifications is plotted as well. We can observe an almost optimal behaviour in both C2C scenarios, with a steady delivery rate close to 100%, which only starts to degrade when approaching 10 ms intervals. This is in line with the times shown in Figure 7a. On the other hand, the server-centric scenario reveals a maximum LwM2M payload goodput of ≈ 50 B/s, and a degradation of the delivery rate for intervals bellow 100 ms, which is in concordance with the notification arrival times we observed before. When utilizing a higher radio data rate we observe a slight improvement in the delivery rates and less dispersion in the goodput values. We attribute this to a lower probability of packet collision.

Finally, to simulate a more realistic and less controlled topology, we construct six topologies of 20 randomly selected nodes each. The constraints for the selection algorithm are a minimum distance of 2.2 m and a maximum of 6.6 m between nodes, which has resulted in a sufficiently reliable communication for our measurements. For each topology, the hosting client and the requesting client are also randomly chosen. Figure 10 depicts the randomly built topologies and the number of hops between the nodes of interest. We measure the notification completion times for the three previously-described deployment scenarios, and observe that C2C performs between 60% and 90% faster than server-centric communication.

6.6 Energy consumption

For the topologies depicted in Figure 6 we now measure the energy consumption while sending notifications to a requesting client. Figure 9 shows the energy consumption aggregated through all nodes but the gateway, which was not considered energy-constrained due to its wired connection. We observe that the main impact in energy consumption occurs when increasing the number of intervening nodes, and that C2C deployments, at ≈ 42.9 J, pay no energy

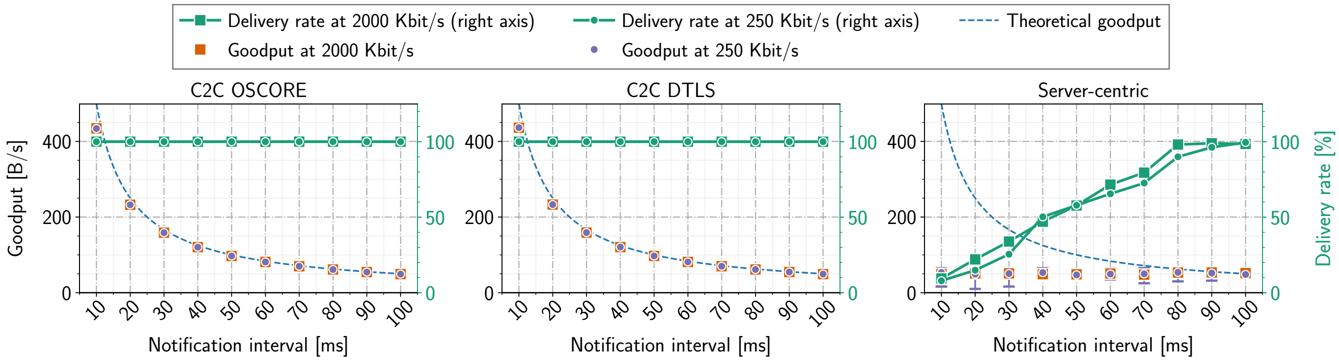


Figure 8: Goodput of client-to-client (C2C) and server-centric deployments using different notification intervals.

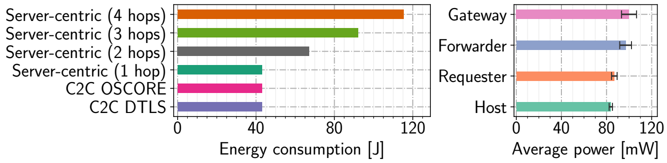


Figure 9: Total energy consumption for different deployments (left), and the average with the standard deviation of power requirements for different node types (right).

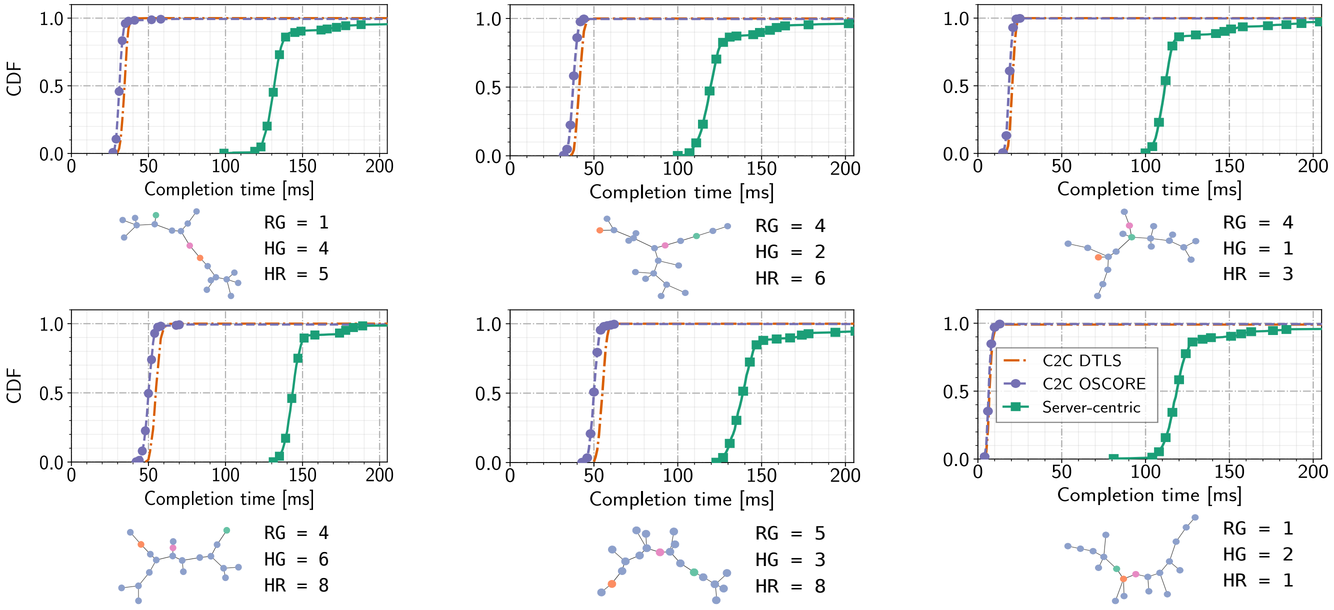
overhead for the features, nor for the simultaneous utilization of DTLS and OSCORE. Moreover, the right side of Figure 9 shows that forwarder nodes require ≈ 12% more power than hosts and requesters. We can conclude that C2C communication helps relaxing the overall energy requirements in LwM2M deployments by reducing the amount of intermediate forwarder nodes.

7 RELATED WORK

Edge computing and device-to-device communication. IoT deployments have shown a shift from centralized cloud computing paradigms towards distributed architectures that augment the edge of the network [31] by producing and consuming data locally in an autonomous fashion. Edge Mesh [25] proposes a paradigm where decision-making and task distribution are moved to edge devices. Costa *et al.* [5] propose a middleware to share a tuple space among wireless sensors. Following the same direction, Whitehouse *et al.* [39] and Lachenmann *et al.* [16] present neighborhood programming abstractions for wireless sensor networks to share state among them. Shang *et al.* [29] show how named data networking architecture allows building IoT applications with local trust management and inter-vendor interoperability, while staying independent of constant cloud connectivity. Although these proposals focus on enabling decentralized IoT deployments, they define their own interaction and data models. In this paper, we focus on LwM2M, as it is a highly deployed management protocol. Tracey *et al.* [35] propose a peer-to-peer architecture based on a distributed hash table, which they further develop in [37]. Although they integrate the developed tuple-based library in a LwM2M implementation [36], no analysis is performed on the performance impact of such integration on a LwM2M deployment.

LwM2M extensions. The interoperability provided by the LwM2M protocol makes it an appealing solution for heterogeneous IoT deployments, thus, multiple extension proposals have been made to expand its capabilities and increase its performance. Given the lossy nature of IoT networks and the reduced energy availability, there are proposals to reduce the traffic between clients and servers. Karaagac *et al.* [12] define a LwM2M object that allows LwM2M servers to perform batched operations on clients, thus, reducing the amount of sent messages. In addition, different LwM2M proxy entities [24] [22] have been proposed. They can perform group operations across multiple clients, cache and aggregate responses and apply compression mechanisms to the messages. We argue that the addition of intermediate proxies in LwM2M networks increases deployment costs and complexity. Although these solutions reduce the bytes transferred between client and server on certain scenarios, they do not provide direct interaction among clients, thus, they do not enable autonomous deployments with distributed IoT applications, which is described by Jimenez in [11] as a lacking feature in the LwM2M specification.

Authentication in the IoT. With an increasing direct communication between constrained IoT devices, the need for mutual authentication and authorization [26] arises. Markmann *et al.* [18] propose a lightweight federation scheme that binds device authentication to network attachment. Vučinić *et al.* [38] propose the OSCAR architecture, based on object security and the distribution of access secrets to request resources from other nodes. However, the existence of a secret per access group imposes high memory requirements when a fine-grained access control is required. Moreover, the usage of the same secret access across consumers complicates access revocation. AoT [19, 20] proposes a suite of protocols to perform attribute-based access control and authentication throughout the life-cycle of IoT devices. Their analysis suggests that the imposed communication overhead may not be well-suited for limited bandwidth networks, as it would produce a big amount a packet fragmentation, which could lead to packet loss. Xi *et al.* [40] propose an authentication and key agreement mechanism to enable device-to-device communication, which allows deriving common secrets based on the radio environment. The IETF is developing ACE-OAuth [27], an authentication and authorization framework based on OAuth 2.0 and



RG: hops between requester & gateway **HG:** hops between host & gateway **HR:** hops between host & requester

Figure 10: Temporal distributions of notification arrival times for randomly generated topologies.

CoAP, where devices request access tokens and credentials from authorization servers, which are used to access resources on other devices. This delegates access control and policies to centralized servers. Although integrating such mechanisms into LwM2M may be feasible, in this paper we focus on reutilizing the already existing key distribution and access control mechanisms in LwM2M.

8 CONCLUSION AND OUTLOOK

In this paper, we started from the observation that device management and provisioning is challenging but required in the constrained IoT. Popular solutions such as LwM2M avoid this challenge by involving servers when sensors need to communicate with actuators. This triangular message forwarding requires upstream connectivity where local communication is sufficient.

We designed and implemented client-to-client communication. Instead of starting completely from scratch, our solution purposefully extends LwM2M. We provided a detailed security analysis, including an attacker and threat model, which showed that our proposal complies with LwM2M security requirements but abandons the server-centric perspective. Our performance analysis, conducted in a multi-hop testbed, showed that client-to-client communication leads to shorter data arrival times (up to ≈ 90% on single-hop topologies) and higher and reliable goodput (≈ 8× when notifying resource updates) compared to the server-centric communication, but only introduces little overhead (≈ 8% in ROM and ≈ 0.9% in RAM). Our findings indicated that client-to-client communication may lead to almost optimal data delivery.

In the future, we plan to apply the principles derived in this work on other management protocols. We also aim for further improvements of our proposal. Link Bindings [15] allow to dynamically link state updates between resources, allowing to define distributed behaviours. By utilizing Group OSCORE [34] or its data-centric variants [7] a LwM2M client could potentially reduce the number of outgoing notifications when multiple observations exist on the same resources. The emerging lightweight authorization and authentication framework ACE OAuth may be deployed in parallel to the existing key distribution and access control mechanisms of LwM2M.

ARTIFACTS

We conducted all our experiments based on open source software and an open access testbed. Source code and documentation are available on Github: http://github.com/inetrg/ipsn-2022-lwm2mc2c.

ACKNOWLEDGMENTS

We would like to thank our anonymous shepherd and reviewers for their valuable feedback. This work was partly supported by the German Federal Ministry of Education and Research (BMBF) within the project PIVOT.

REFERENCES

[1] Christian Amsuess, Zach Shelby, Michael Koster, Carsten Bormann, and Peter van der Stok. 2020. *CoRE Resource Directory.* Internet-Draft – work in progress 26. IETF.

[2] Emmanuel Baccelli, Cenk Gündogan, Oliver Hahm, Peter Kietzmann, Martine Lenders, Hauke Petersen, Kaspar Schleiser, Thomas C. Schmidt, and Matthias Wählisch. 2018. RIOT: an Open Source Operating System for Low-end Embedded Devices in the IoT. *IEEE Internet of Things Journal* 5, 6 (December 2018), 4428–4440. http://dx.doi.org/10.1109/JIOT.2018.2815038

[3] C. Bormann, M. Ersue, and A. Keranen. 2014. *Terminology for Constrained-Node Networks.* RFC 7228. IETF.

[4] C. Bormann and P. Hoffman. 2013. *Concise Binary Object Representation (CBOR).* RFC 7049. IETF.

[5] Paolo Costa, Luca Mottola, Amy L. Murphy, and Gian Pietro Picco. 2006. TeenyLIME: Transiently Shared Tuple Space Middleware for Wireless Sensor Networks. In *Proceedings of the International Workshop on Middleware for Sensor Networks* (Melbourne, Australia) *(MidSens '06)*. Association for Computing Machinery, New York, NY, USA, 43–48.

[6] Roy T. Fielding and Richard N. Taylor. 2000. Principled Design of the Modern Web Architecture. In *Proc. of the 22Nd International Conference on Software Engineering* (Limerick, Ireland) *(ICSE '00)*. ACM, New York, NY, USA, 407–416.

[7] Cenk Gündogan, Christian Amsüss, Thomas C. Schmidt, and Matthias Wählisch. 2021. Group Communication with OSCORE: RESTful Multiparty Access to a Data-Centric Web of Things. In *Proc. of the 46th IEEE Conference on Local Computer Networks (LCN)* (Edmonton, Canada). IEEE Press, Piscataway, NJ, USA, 399–402. https://doi.org/10.1109/LCN52139.2021.9525000

[8] Cenk Gündogan, Peter Kietzmann, Martine S. Lenders, Hauke Petersen, Michael Frey, Thomas C. Schmidt, Felix Shzu-Juraschek, and Matthias Wählisch. 2021. The Impact of Networking Protocols on Massive M2M Communication in the Industrial IoT. *IEEE Transactions on Network and Service Management (TNSM)* 18, 4 (Dec. 2021), 4814–4828. https://doi.org/10.1109/TNSM.2021.3089549

[9] Jungha Hong, Yong-Geun Hong, Xavier de Foy, Matthias Kovatsch, Eve Schooler, and Dirk Kutscher. 2022. *IoT Edge Challenges and Functions*. Internet-Draft – work in progress 04. IETF.

[10] International Society of Automation. 2011. *Wireless Systems for Industrial Automation: Process Control and Related Applications*. Technical Report Standard ISA-100.11a-2011. ISA.

[11] Jaime Jimenez. 2016. *CoAP functionality expected in a LWM2M system*. Internet-Draft – work in progress 00. IETF.

[12] Abdulkadir Karaagac, Matthias VanEeghem, Jen Rossev, Bart Moons, Eli DePoorter, and Jeroen Hoebeke. 2018. Extensions to LwM2M for Intermittent Connectivity and Improved Efficiency. In *2018 IEEE Conference on Standards for Communications and Networking (CSCN)*. IEEE, Paris, France, 1–6.

[13] Peter Kietzmann, Thomas C. Schmidt, and Matthias Wählisch. 2021. A Guideline on Pseudorandom Number Generation (PRNG) in the IoT. *ACM Comput. Surv.* 54, 6 (July 2021), 112:1–112:38. https://dl.acm.org/doi/10.1145/3453159

[14] Loren Kohnfelder and Praerit Garg. 1999. *The threats to our products*. Technical Report. Microsoft. https://adam.shostack.org/microsoft/The-Threats-To-Our-Products.docx

[15] Michael Koster and Bill Silverajan. 2021. *Dynamic Resource Linking for Constrained RESTful Environments*. Internet-Draft – work in progress 14. IETF.

[16] Andreas Lachenmann, Pedro Marrón, Daniel Minder, Olga Saukh, Matthias Gauger, and Kurt Rothermel. 2007. Versatile Support for Efficient Neighborhood Data Sharing. *In: Langendoen, Koen (ed.); Voigt, Thiemo (ed.): Proceedings of the 4th European Conference on Wireless Sensor Networks (EWSN 2007), pp. 1-16* 4373.

[17] Leandro Lanzieri, Peter Kietzmann, Thomas C. Schmidt, and Matthias Wählisch. 2021. Poster Abstract: Third Party Authorization of LwM2M Clients. In *Proc. of ACM/IEEE Int. Conf. on Internet of Things Design and Implementation (IoTDI '21)* (Virtual). ACM, New York, NY, USA, 263–264. https://doi.org/10.1145/3450268.3453512

[18] Tobias Markmann, Thomas C. Schmidt, and Matthias Wählisch. 2015. Federated End-to-End Authentication for the Constrained Internet of Things using IBC and ECC. In *Proc. of ACM SIGCOMM, Poster Session* (London). ACM, New York, 603–604. http://dx.doi.org/10.1145/2785956.2790021

[19] Antonio L. Maia Neto, Yuri L. Pereira, Artur L. F. Souza, Italo Cunha, and Leonardo B. Oliveira. 2018. Demo Abstract: Attributed-Based Authentication and Access Control for IoT Home Devices. In *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. 112–113.

[20] Antonio L. Maia Neto, Artur L. F. Souza, Italo Cunha, Michele Nogueira, Ivan Oliveira Nunes, Leonardo Cotta, Nicolas Gentille, Antonio A. F. Loureiro, Diego F. Aranha, Harsh Kupwade Patil, and Leonardo B. Oliveira. 2016. AoT: Authentication and Access Control for the Entire IoT Device Life-Cycle. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM* (Stanford, CA, USA) *(SenSys '16)*. Association for Computing Machinery, New York, NY, USA, 1–15.

[21] NIST. 2004. *Standards for Security Categorization of Federal Information and Information Systems*. Technical Report FIPS-199. National Institute of Standards and Technology, Gaithersburg, MD, US.

[22] Martina Pappalardo, Giacomo Tanganelli, and Enzo Mingozzi. 2020. Enhanced Support of LWM2M in Low Power and Lossy Networks. In *2020 IEEE International Conference on Smart Computing (SMARTCOMP)*. 344–349.

[23] E. Rescorla and N. Modadugu. 2012. *Datagram Transport Layer Security Version 1.2*. RFC 6347. IETF.

[24] Maria Ines Robles, Domenico D'Ambrosio, Jaime Jimenez Bolonio, and Miika Komu. 2016. Device group management in constrained networks. In *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. IEEE, Sydney, NSW, Australia, 1–6.

[25] Yuvraj Sahni, Jiannong Cao, Shigeng Zhang, and Lei Yang. 2017. Edge Mesh: A New Paradigm to Enable Distributed Intelligence in Internet of Things. *IEEE Access* 5 (2017), 16441–16458.

[26] L. Seitz, S. Gerdes, G. Selander, M. Mani, and S. Kumar. 2016. *Use Cases for Authentication and Authorization in Constrained Environments*. RFC 7744. IETF.

[27] Ludwig Seitz, Goeran Selander, Erik Wahlstroem, Samuel Erdtman, and Hannes Tschofenig. 2021. *Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)*. Internet-Draft – work in progress 46. IETF.

[28] G. Selander, J. Mattsson, F. Palombini, and L. Seitz. 2019. *Object Security for Constrained RESTful Environments (OSCORE)*. RFC 8613. IETF.

[29] Wentao Shang, Zhehao Wang, Alexander Afanasyev, Jeff Burke, and Lixia Zhang. 2017. Breaking out of the Cloud: Local Trust Management and Rendezvous in Named Data Networking of Things. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation* (Pittsburgh, PA, USA) *(IoTDI '17)*. Association for Computing Machinery, New York, NY, USA, 3–13.

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

[31] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.

[32] OMA SpecWorks. 2020. *Lightweight Machine to Machine Technical Specification: Core v1.2*. Technical Report. Open Mobile Alliance.

[33] OMA SpecWorks. 2020. *Lightweight Machine to Machine Technical Specification: Transport Bindings v1.2*. Technical Report.

[34] Marco Tiloca, Goeran Selander, Francesca Palombini, John Mattsson, and Jiye Park. 2022. *Group OSCORE - Secure Group Communication for CoAP*. Internet-Draft – work in progress 14. IETF.

[35] David Tracey and Cormac Sreenan. 2013. A Holistic Architecture for the Internet of Things, Sensing Services and Big Data. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. 546–553.

[36] David Tracey and Cormac Sreenan. 2017. OMA LWM2M in a holistic architecture for the Internet of Things. In *2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC)*. 198–203.

[37] David Tracey and Cormac Sreenan. 2019. Using a DHT in a Peer to Peer Architecture for the Internet of Things. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. IEEE, Limerick, Ireland, 560–565.

[38] Mališa Vučinić, Bernard Tourancheau, Franck Rousseau, Andrzej Duda, Laurent Damon, and Roberto Guizzetti. 2015. OSCAR: Object security architecture for the Internet of Things. *Ad Hoc Networks* 32 (2015), 3–16.

[39] Kamin Whitehouse, Cory Sharp, David Culler, and Eric Brewer. 2004. Hood: A Neighborhood Abstraction for Sensor Networks. *MobiSys 2004 - Second International Conference on Mobile Systems, Applications and Services*.

[40] Wei Xi, Chen Qian, Jinsong Han, Kun Zhao, Sheng Zhong, Xiang-Yang Li, and Jizhong Zhao. 2016. Instant and Robust Authentication and Key Agreement among Mobile Devices. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 616–627.