Securing name resolution in the IoT: DNS over CoAP

Martine S. Lenders
Freie Universität Berlin
m.lenders@fu-berlin.de

Christian Amsüss
christian@amsuess.com

Cenk Gündogan
HAW Hamburg
cenk.guendogan@haw-hamburg.de

Marcin Nawrocki
Freie Universität Berlin
marcin.nawrocki@fu-berlin.de

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

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

ABSTRACT

In this paper, we present the design, implementation, and analysis of DNS over CoAP (DoC), a new proposal for secure and privacy-friendly name resolution of constrained IoT devices. We implement different design choices of DoC in RIOT, an open-source operating system for the IoT, evaluate performance measures in a testbed, compare with DNS over UDP and DNS over DTLS, and validate our protocol design based on empirical DNS IoT data. Our findings indicate that plain DoC is on par with common DNS solutions for the constrained IoT but significantly outperforms when additional, CoAP standard features are used such as block-wise transfer or caching. With OSCORE for end-to-end security, we can save more than 10 kBytes of code memory compared to DTLS while enabling group communication without compromising the trust chain when using intermediate proxies or caches. We also discuss a scheme for very restricted links that compresses redundant or excessive information by up to 70%.

CCS CONCEPTS

• **Networks** → **Network protocol design**; Application layer protocols; *Security protocols.*

KEYWORDS

CoAP, DNS, Internet of Things, protocol design, network security

1 INTRODUCTION

The Internet of Things (IoT) progressively achieves deployment, covering a wide range from consumer-grade products such as Smart-TVs, to industrial monitoring and control, to simple environmental sensors. Most IoT operations frequently require access to data or (cloud-)services, commonly accessed via names. Enabling unprotected name resolution on such devices raises concerns regarding security and privacy because names may carry specific semantics and large-scale IoT deployments give rise for large-scale botnets.

Nodes on the IoT are often constrained. These *things* commonly interconnect wirelessly and remain independent of the power grid; they typically operate on batteries or harvest energy from environment. Hardware platforms are kept simple to prolong operation times and reduce unit costs. Even the most powerful devices based on a common IETF classification [6] show orders of magnitude less memory than general-purpose hardware platforms (see Table 1). These devices, however, also require protected name resolution.

Protecting the name resolution of the DNS is an important building block in strengthening privacy and security [55]. Unfortunately, the common use of DNS on top of encrypted transport, DNS over HTTPS (DoH) [17], DNS over TLS (DoT) [19], and DNS over QUIC (DoQ) [21], conflicts with low hardware resources of constrained class 1 and 2 devices.

Table 1: Constraints of potential DoC target platforms [6].

Memory	Class 0	Class 1	Class 2
RAM [kBytes]	$\ll 10$	≈ 10	≈ 50
ROM [kBytes]	$\ll 100$	≈ 100	≈ 250

In this paper, we present secure and privacy-friendly DNS resolution for the constrained IoT. We base our solution on CoAP [49], the Constrained Application Protocol standardized in the IETF. CoAP is a lightweight alternative to HTTP for the IoT and widely available. CoAP is based on UDP but provides transactional message contexts, retransmission mechanisms, and en-route caching on dedicated forward proxies. Security extensions either use DTLS [39] or content object security with OSCORE [47]. DNS over CoAP (DoC) can leverage these security solutions to query the DNS privately, securely, and yet efficiently enough to comply with the low-end IoT.

Designing and implementing DoC is challenging. First, different CoAP methods provide different features and tradeoffs. Second, common DNS answers are large and lead to packet fragmentation, which should be avoided in the IoT. Third, DNS and CoAP provide different independent cache freshness, and caching approaches from DoH cannot directly be applied without leading to larger drawbacks.

In summary, our main contributions read:

(1) We analyze the impact of IoT name features on the design of DNS resolution in the IoT based on an empirical data set of characteristic IoT domain names. We compare with names requested via a large regional Internet eXchange Point. (Section 3)
(2) We design the DoC protocol, which leverages the rich feature set of the CoAP protocol suite for DNS, including end-to-end protection, block-wise transfer, and group communication. (Section 4)
(3) We implement DoC in RIOT, a popular open-source IoT operating system for constrained devices. Our reference software makes DoC available on more than 230 IoT platforms. (Section 5)
(4) A system-level analysis conducted on real IoT hardware reveals that DoC performance is at least on par with generic UDP-based DNS transport. Additional features increase the DoC performance further. (Sections 6 and 7)

(5) We discuss the utility of a potential new media type to transport DNS messages over DoC or DoH. (Section 8)

The remainder of this paper is structured as follows: After providing background on the problem space and related work in Section 2, we present our contributions in Sections 3 to 7, discuss the findings of our evaluation in Section 8, and finish with our conclusion in Section 9.

2 PROBLEM STATEMENT AND RELATED WORK

2.1 The Need for Secure Name Resolution

Threats to Infrastructure Security. The IoT repeatedly takes center stage in large-scale distributed Denial of Service (DDoS) attacks against the Internet infrastructure [4, 16]. Two major reasons make IoT devices a popular target to establish botnets, from which attackers launch amplification attacks based on the DNS [11, 44]. First, the large number of devices facilitates the creation of botnets of substantial sizes, which eases obfuscation of originators. Second, securing IoT devices is a complex task [34], leaving many of the nodes exposed to vulnerabilities [48]. Since IoT devices need to resolve names, most system stacks provide a standard DNS resolver per UDP. The DNS over UDP stack exposes a powerful attack vector, which needs confinement to make amplification attacks more challenging.

Threats to End User Privacy. Many IoT use cases collect, process, and expose sensitive data from the physical world to cloud applications on the Internet. Nevertheless, recent reports [35] show that the majority of device traffic is unencrypted, which has threatening consequences for life and enterprise functions. Unencrypted DNS queries contain hostnames in plain text and potentially leak application-specific information to eavesdroppers. Additionally, plain queries concurrent to (protected) application traffic may disclose the nature of confidential data, reveal behavioral patterns, or uncover hints used for fingerprinting victims [28]. Preserving confidentiality and privacy of DNS queries and responses is thus highly relevant for a sustainable IoT.

2.2 Challenges from the Constrained IoT

The use of DNS over the Transmission Control Protocol (TCP) [12] mitigates DNS-based DDoS attacks, but faces a limited support by many resolvers [29]. TCP itself introduces an increased transport complexity and is only viable for the constrained IoT under restricted functionality and with limited performance [14]. DNS over TLS (DoT) [19], DNS over HTTPS (DoH) [17], and most recently DNS over QUIC (DoQ) [21] are mechanisms to protect the confidentiality and integrity of DNS traffic on the Internet. They employ transport layer security and maintain session state between two endpoints to prevent IP spoofing. The first two approaches build on TCP and their performance significantly drops when network conditions degrade [18]. This reduces their applicability in Low-power and Lossy Networks (LLNs) where links commonly saturate. DoQ uses UDP, but despite its performance advantages over DoT and DoH [24], deployment in low-power regimes is complicated by its use of TLS [41].

DNS over DTLS (DoDTLS) [39] is an alternative that also runs on datagram transport. In addition to the lower protocol complexity compared to the former approaches, this transport does not suffer from head-of-line blocking on the transport, which frequently occurs in LLNs. Nevertheless, DoDTLS faces issues with larger messages exceeding the Path Maximum Transmission Unit (PMTU) [39] and forces applications into fragmentation. Moreover, IoT link layers, *e.g.,* IEEE 802.15.4 or LoRaWAN, provide Service Data Units (SDUs) of a few hundred bytes [22, 27], which is easily reached by rather small queries or responses. Even though the respective adaption layers for IPv6, *e.g.,* 6LoWPAN [33] or SCHC [13], offer fragmentation between the link and network layer, they introduce higher packet loss and higher latencies [26].

Despite the advantages of DoDTLS, there are certain drawbacks with protecting the transport for the IoT use case [15]. (*i*) IoT networks may connect to application gateways that transform between transports, *e.g.,* between UDP and TCP. This burdens the end-to-end protection, since gateways need to be included in trust relationships to re-encrypt the data between endpoints. (*ii*) various IoT scenarios leverage multicast transmissions to efficiently utilize wireless media. DTLS complicates security on the transport layer a multiparty communication, because security contexts establish between individual endpoints. (*iii*) a loose coupling and caching are usually favored techniques in the IoT to deal with mobility and network partitioning, but the established security sessions are deeply rooted on the transport and harden the endpoint paradigm.

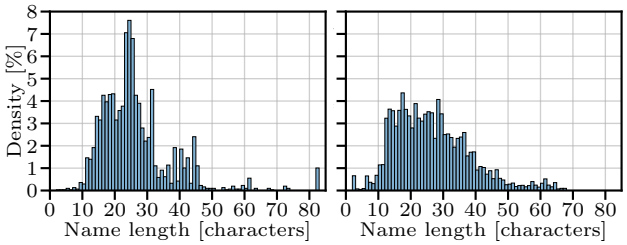
While current standardization efforts for DTLS, *e.g.,* Connection Identifiers (CIDs) [43], help to address the drawbacks, there is another undertaking in the IETF CoRE working group to provide a secured communication. OSCORE [47] protects messages on the object-level instead of the transport-level. It fully integrates with the Constrained Application Protocol (CoAP) [49] ecosystem, ensures end-to-end protection across gateways, allows for a protected multiparty communication, and makes encrypted and authenticated CoAP messages cachable on untrusted proxies. Since CoAP was designed as the HTTP for the IoT, we see it as the proper candidate for protective measures similar to DoH. The CoAP ecosystem may facilitate power-efficient, privacy-friendly DNS queries in the IoT, while it mitigates the size of DDoS attacks through bandwidth reduction and peer authentication.

3 EMPIRICAL VIEW ON IOT DNS TRAFFIC

In this section, we motivate our design decisions. To this end, we empirically analyze DNS traffic produced by end-consumer IoT devices and compare to flow samples from a large regional European IXP.

3.1 Data Corpus

Identifying IoT-specific traffic is challenging. We rely on data from three common projects, YourThings [2], IoTFinder [36], and MonIoTr [40], which captured and annotated IoT traffic based on ground truth. YourThings and IoTFinder also provide traffic from desktop computers, phones, tablets, gaming consoles, and Wi-Fi access points. We exclude such traffic. All three IoT data sets include both unicast DNS and multicast DNS (mDNS) [10] traffic. As mDNS is integral to service discovery, namely ZEROCONF, we



(a) IoT devices (b) Internet devices

Figure 1: Distribution of name lengths; names queried by different devices connected via the Internet.

keep mDNS traffic. Overall, the aggregation of all three data sources provides us data from over 90 consumer-grade IoT devices by over 50 manufacturers and contains 0.2 million DNS and mDNS queries and 1.3 million corresponding responses in total. The IoTFinder data only contains responses but we can infer the queries from their question section.

To compare IoT-specific DNS traffic with DNS traffic from common Internet devices, we leverage sFlow [37] samples. Our IXP data contains 1.6 million unicast DNS queries and 2.4 million responses and is based on a sampling rate of 1/16000 packets and packets are truncated to 128 bytes. For privacy regulation compliance, we stripped all names and replaced them with the target analysis data (*e.g.,* name lengths) before exporting them for our analysis.

3.2 Results

How long are names requested by IoT devices? Figure 1 shows a normalized histogram over all queried names seen, for both the IoT data sets (Figure 1a) and the IXP data set (Figure 1b). Furthermore, the statistical key properties for the name lengths of each data set and the aggregated IoT data set are shown in Table 2.

The median of the name lengths is, depending on the IoT data set 23 or 24, which is confirmed by the median of 24 of the IXP data set. Many cloud and CDN names, such as `e123.abcd.akamaiedge.net` (name modified for sake of privacy), gather around this name length. Significantly longer names are typically used for certain mDNS applications, *e.g.,* for reverse DNS or to identify local devices via a UUID. As such, we do not see such longer name lengths at the IXP.

What kind of records are requested? In Table 3, we listed the percentages of a selection of the queried record types seen in the `IN` class in the analyzed data sets. `A` records are in all data sets the most requested records, with `AAAA` records being a close second. With growing deployment of IPv6, these numbers are expected to get ever closer in the future, and at one point even flip. When not accounting for mDNS, these are effectively the only records of significance in the IoT. With mDNS we also see records typically associated with service discovery, namely `ANY`, `PTR`, `SRV`, and `TXT` records [9, 10].

Table 2: Statistical key properties of domain names queried by IoT devices compared to domain names visible at an Internet Exchange Point (IXP). μ denotes the mean, σ the standard deviation, Q_1 the first quartile, Q_2 the second quartile (or median), and Q_3 the third quartile.

Data source	Lengths of Domain Names [chars]						
	min	max	μ	σ	Q_1	Q_2	Q_3
YourThings [2]	2	83	24.5	9.7	18	24	30
IoTFinder [36]	7	82	26.8	10.5	20	24	30
MonIoTr [40]	9	83	27.1	14.7	18	23	30
IoT total	2	83	25.9	11.3	19	24	30
IXP	0	68	26.1	11.7	17	25	33

Table 3: Queried record types in `IN` class.

Record Type	IoT Devices		IXP
	w/ mDNS	w/o mDNS	
`A`	53.6%	75.8%	64.5%
`AAAA`	16.4%	23.5%	17.6%
`ANY`	8.2%	—	1.7%
`HTTPS`	—	—	9.1%
`NS`	—	—	1.0%
`PTR`	19.6%	0.3%	1.8%
`SRV`	1.0%	—	0.3%
`TXT`	1.2%	0.1%	0.5%
Other	< 0.1%	0.3%	3.5%

Besides a great number of other less requested records (3.5% in total), we also mostly see `A` and `AAAA` records at the IXP. The remainder are `NS` and `HTTPS` [46] records.

Other DNS data of interest. In addition to queried name lengths and record types we also looked into the number of entries in each section and the response lengths overall. None of these we depicted, primarily to save space in this paper.

In the analyzed data sets we see numbers of entries in each of the sections that are greater than 255, the overflow point into 2-byte numbers, but the percentage is low and mostly relevant to mDNS. These large numbers most often stem from unrequested `NS` records that advertise name servers in the authority section and the associated `A` or `AAAA` records for these advertised name servers in the additional section. Providing these, seems to be a common practice from cloud and CDN providers, but they altogether optional. The question, on the other hand, section generally only contains 1 entry. In fact, many resolvers ignore or error on queries with a question section length other than 1.

The responses can become very long and contain 400 to 600 bytes, in certain cases even more than 1 kByte, even for the IoT devices. This is again to the long authority and additional sections described above.

4 DESIGN OF DNS OVER COAP (DOC)

In this section, we define DNS over CoAP (DoC), a protocol to query the domain name system and retrieve responses over the

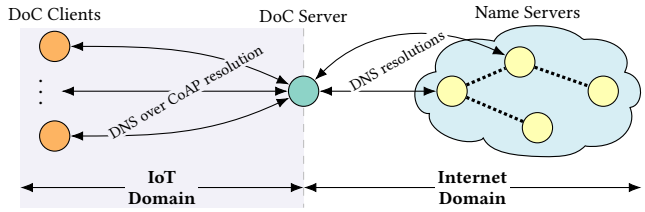
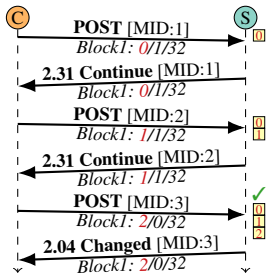


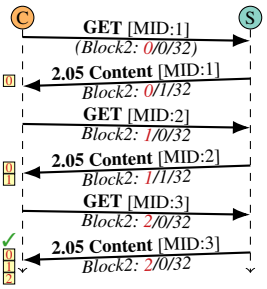
Figure 2: Overview of the DNS over CoAP (DoC) architecture.

Table 4: Comparison of request methods considered for DoC.

	GET	POST	FETCH
Cacheable	✓	✘	✓
Application data carried in body	✘	✓	✓
Block-wise transferable query	✘	✓	✓



(a) Block1 to transfer requests



(b) Block2 to transfer responses

Figure 3: Example of different block-wise transfers of a 96 bytes body between a client C and a server S using 32 byte blocks in CoAP. $n/m/s$ notes the block number, whether more blocks (or not) can be sent, and block size.

Constrained Application Protocol [49] (see Figure 2). We also published this work as an Internet Draft [25]. The goal is to map each DNS query-response pair to a CoAP message exchange, secured on the transport via DTLS [42] or on the object level via OSCORE [47] to ensure message integrity and confidentiality.

Using CoAP for DNS resolution provides the following advantages in the constrained IoT. First, CoAP runs on UDP. UDP does not introduce additional connection setup or states but CoAP still provides reliability on top. Second, CoAP provides block-wise transfer to fragment and reassemble large messages, which are likely in DNS. Third, proxy operations and response caches may help to compensate packet losses in wireless networks. We now discuss how to leverage the CoAP RESTful architecture to improve DNS resolution in the IoT.

4.1 Protocol Overview

For a basic message exchange, mapping DNS queries to CoAP requests and DNS responses to CoAP replies is necessary.

Request mapping. A DoC client can send a DNS query over CoAP by embedding the on-the-wire format of a DNS query into a CoAP message using either a GET, POST, or FETCH [52] request, each of them provide different features.

Using GET, the DNS query needs to be encoded within the URI. This allows for caching of subsequent responses but prevents block-wise transfer and requires a URI Template processor [17] at the constrained side. POST, on the other hand, carries the DNS query in the CoAP body, which reduces complexity since no additional URI processor is needed, but, on the negative side, does not allow for caching because the payload of the request is not taken into account for a cache key. To allow for both caching and block-wise transfer, a DoC client can use FETCH [52]. Even though FETCH is currently not supported by all CoAP implementations, extending them is easy (details see appendix A). Table 4 displays the different benefits and drawbacks of the three CoAP methods.

Response mapping. A DoC server sends a DNS response over CoAP by encoding the on-the-wire format of the DNS response in the payload of a CoAP response.

Block-wise transfer. CoAP POST and FETCH, which carry a DNS query in the body of a request, provide an additional advantage in case the DNS query message size exceeds an MTU: The body can be split into multiple CoAP messages by the block-wise transfer mode. When using the Block1 [8] option, a receiver assembles the full message on successful reception of all blocks (see Figure 3a). In contrast to queries, the Block2 transfer mode [8] allows a client to request a certain block size in the response, but the server may also decide to transfer in blocks proactively, without the Block2 option being present in the initial request (see Figure 3b).

4.2 Response Caching

To enable an efficient utilization of a CoAP cache, we need to tackle three challenges. First, we need to ensure that the CoAP *cache key* is consistent for the same DNS query, since this key is used to determine the existence of cached response copies. Second, we need to align CoAP response caching time with DNS record lifetimes (*i.e.,* TTL) such that a DoC client does not get outdated data or triggers unnecessary data delivery. Third, we need to effectively leverage the cache validation model of CoAP to reduce the number of transmissions for large DNS responses.

Consistent cache keys. When using CoAP FETCH or GET, the original DNS message becomes part of the cache key, either because the key includes the payload (FETCH) or the URI (GET). Since any DNS message carries an ID in the DNS header, which might be different for the multiple queries of the same resource record and hostname, we propose to set this ID always to 0. This yields a deterministic on-the-wire format without introducing additional states at the client side or coordinating this ID between multiple DoC clients.

Aligning expiration timers. TTLs in DNS responses describe how long a resource record should stay in a DNS cache. They are

actively maintained by any DNS cache and, in a DNS response from a cache, reflect the remaining time. When embedded in CoAP messages, TTLs, however, are opaque since DNS responses are treated as any other CoAP payload. DoC clients may thus face two cases that impact the protocol efficiency when cached responses are retrieved. (*i*) DNS TTLs may be expired, although CoAP caches consider them as valid. This leads to outdated name resolutions at DoC clients. (*ii*) Cached CoAP responses may be expired, while the DNS TTLs are still valid. This leads to a reduced cache utilization and unnecessary network overhead.

A CoAP server may include an entity-tag (ETag) in a response to differentiate between response representations. Then, client and proxy caches use ETags to query for the validity of stale cache entries. If a cached object is still valid, the server transmits a *small* confirmation message including the 2.03 response code to reduce the network bandwidth usage. When the entry is outdated (*i.e.,* the payload changed), the server transmits the full response.

We consider the following two approaches to align the DNS record lifetimes with the CoAP response caching and validation model.

Option 1: DoH-like. This approach is inspired by DoH [17, Section 5.1]. A DoC server sets the Max-Age option of CoAP to the minimum TTL of encapsulated DNS resource records. The Max-Age value decreases on intermediate CoAP caches, and DoC clients that receive a cached response use the altered Max-Age to reduce TTLs of included resource records. A drawback of this approach is that changing DNS TTLs, either due to DNS caches or DNS operators, changes the CoAP payload and thereby affects the ETag generation. This results in failed cache re-validations and requires a full DNS response, even though only a TTL changed. Such overhead does not harm in common DoH deployments but reduces performance in the IoT (details see Section 7).

Option 2: EOL TTLs. This approach improves the previous option by increasing success of cache re-validations. In detail, we propose that a DoC server sets the Max-Age CoAP option to the minimum TTL of the resource records in the DNS response and rewrites all TTLs with 0 (see Figure 4). In appendix B, we further describe a supplementing mechanism to improve cache re-validations for the rare occasion of having resource records with varying TTLs. These modifications always ensure identical ETag values for the same resource record set. During name resolution, these responses may be stored in intermediary caches, *e.g.,* on a proxy or on a client. For later requests that result in cache hits, Max-Age values are adjusted accordingly to the CoAP freshness model. When a response arrives at the DoC client, the DoC client copies the Max-Age into the resource records to restore the correctly decremented TTL values when making them available to the local DNS system cache. Requests that hit stale cache entries trigger a cache re-validation (see Figure 4). If only TTLs changed, then the DoC server validates the cache entries, and encodes the new TTL in the Max-Age option.

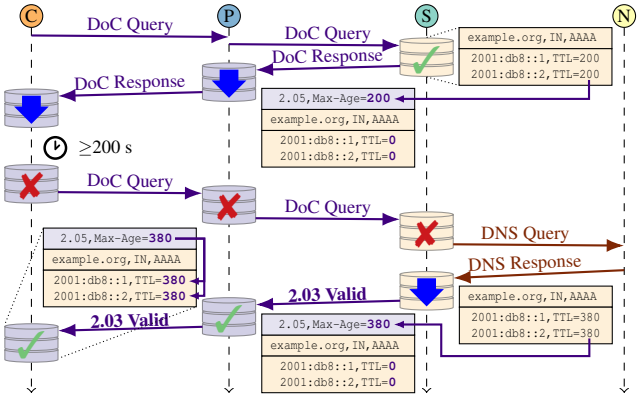


Figure 4: An example of name resolution using the *EOL TTL* caching scheme in DoC. A DoC client C queries `AAAA` records for `example.org` from a DoC server S via a DoC-agnostic CoAP proxy P. Only when a DNS cache miss at S occurs, the DNS name server N is queried.

4.3 Security Modes

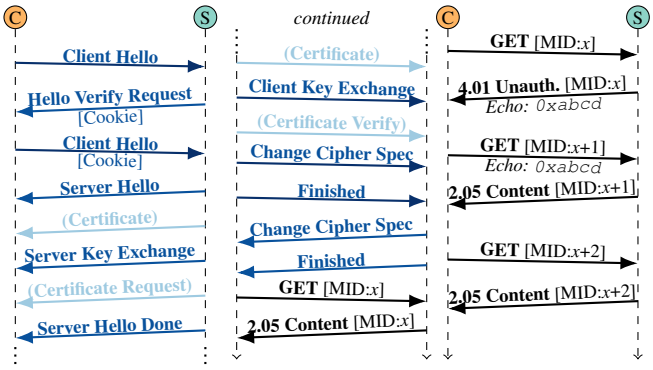
The use of CoAP as transport for DNS enables two security modes to protect the confidentiality and privacy of name resolutions: (*i*) Transport layer security or (*ii*) content object security. Both modes can be used individually, or in combination to attain security features on different protocol layers.

Transport Layer Security. CoAP over DTLS [42] exhibits similar protocol behavior and security guarantees (*i.e.,* confidentiality and protection) as TLS [41], and further contributes a modified record layer that tolerates message reordering and packet loss. One key advantage of this mode compared to DNS over DTLS without CoAP is the block-wise transfer to fragment large DNS queries and responses into smaller CoAP messages. This solves the problem that DoDTLS is facing: The datagram-based DTLS cannot deal with packets that exceed the Path MTU. It relies on EDNS(0) to negotiate a maximum response size during name resolution based on a prior Path MTU discovery [39].

DTLS has three drawbacks, though: First, without any cross-layering, each CoAP retransmission needs to be encrypted and authenticated again before delivery—retransmissions of the encrypted datagram might be rejected by the duplicate detection of the peer. Second, CoAP proxies and caching nodes must be included in the trust relationship to process CoAP messages. Third, DTLS creates only a session between two trusted parties and thus challenges group communication in CoAP [38].

Content Object Security. The second security mode uses OSCORE [47], which is a CoAP protocol extension. It addresses the drawbacks of DTLS. Instead of securing transport sessions between endpoint pairs, it provides integrity, authenticity, and confidentiality on an object level by protecting *entire* CoAP messages. OSCORE transforms the original CoAP message into an authenticated and encrypted CBOR Object Signing and Encryption (COSE) [45] object, and then encapsulates it as a CoAP option in an outer, newly created CoAP header.

To mitigate mismatch and replay attacks, OSCORE constructs a strong message binding between requests and corresponding responses with the use of identical identifiers in their authenticated components, which persist over retransmissions (see Figure 5b). This also leads to less computational effort for encryption and



(a) CoAPS DTLSv1.2 handshake.

(b) OSCORE replay window initialization.

Figure 5: Session creation between a client C and a server S using one of the CoAP security modes discussed in this paper. Blue marks mandatory DTLS messages, lightblue optional DTLS messages, and black CoAP messages.

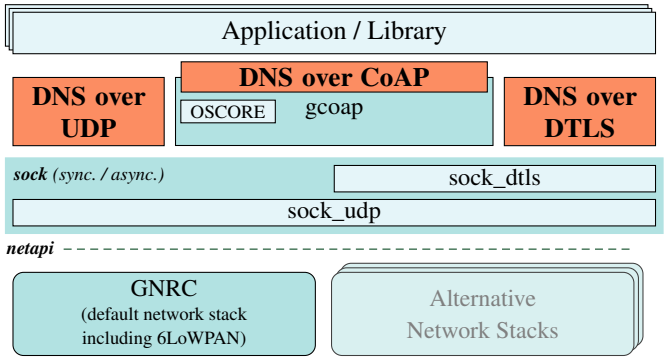


Figure 6: DNS, DoDTLS, and DoC in the RIOT networking subsystem.

authentication, since, compared to CoAP over DTLS, the OSCORE protected messages are stored in the CoAP retransmission buffer.

To enable caching on untrusted nodes, a protocol add-on for OSCORE is proposed [3], thus ensuring end-to-end security via third-party gateways. Likewise, there is a proposal to allow for protected group requests and responses for one-to-many communication [51].

OSCORE initially relies on pre-shared keys or pre-configured certificates. DTLS comes with a built-in key exchange protocol to establish temporary session keys between two endpoints (see Figure 5a). This enables perfect forward secrecy since leaked keys cannot be used to decrypt past correspondences. A lightweight authenticated key exchange for OSCORE (LAKE [53]) is under development, though.

5 IMPLEMENTATION

In this section, we introduce our software framework to run DNS queries in the constrained IoT, including implementations of a DoC prototype, DNS over UDP, and DNS over DTLS. While less RAM (main memory) puts constraints on the size of buffers and concurrent states, less ROM (code space) restricts code complexity. We aim for a lean, yet flexible implementation to enable a consistent comparison of different DNS approaches on the same platform. We make use of the IoT operating system RIOT [5], which provides a well-tested 6LoWPAN network stack, and allows for code portability across a variety of IoT hardware platforms.

Modular network stack integration. The RIOT networking subsystem consists of the following components (see Figure 6). *sock*, a lightweight application programming interface that serves as an entry point for libraries and user applications to perform various networking tasks. *sock* is agnostic to the underlying network stack and allows, among other transports, access abstracted to UDP and DTLS. Individual protocol layers use *netapi* as an interface internal to the networking subsystem. Its unified access allows for a seamless composability of network building blocks, eases implementation complexity, and grants a flexible integration of third-party network stacks.

GNRC is the default network stack of RIOT. It is feature-rich, yet memory-efficient, and implements many of the IETF IoT protocols, including 6LoWPAN [33] and the RPL [54] routing protocol.

The *gCoAP* library provides support for CoAP [49] on top of *sock*. It features the basic request methods GET, POST, PUT, and DELETE, but also the more recent extension including FETCH, PATCH, and iPATCH [52]. *gCoAP* also implements proxying capabilities with support for response caches, utilizing a least recently used (LRU) caching strategy [15]. Block-wise transfer [8] is provided via a slicer class that iteratively fragments a given CoAP body into several block payloads, and reassembles them on reception.

DNS over CoAP. Our implementation of DNS over CoAP (DoC) makes use of the generic interface to compose and parse DNS query and response messages. This reduces additional code complexity, eases code maintenance, and ensures consistency across applications. Our DoC implementation can be configured to perform (*i*) blocking requests, in which case the application halts until a response or a timeout is received, or (*ii*) asynchronous requests as provided by *gCoAP*. The latter enables the parallelization of application-specific tasks, and allows for concurrent requests to the same or other resources, on the same or other servers. We also provide a lightweight URI template parser, which the DoC client can use to marshal the packet format of DNS queries into the URI option of CoAP GET requests.

Our DoC implementation runs over a CoAP transport with support for varying security properties: it may be unencrypted but secured on the transport-level with *sock_dtls*, or each CoAP packet may be protected on the object-level using OSCORE [47]. For the latter, *gCoAP* integrates with the external package *libOSCORE*, but since it requires extra code branches for the use of *libOSCORE* we left out the GET implementation for the sake of complexity reduction.

DNS over UDP. The existing DNS client in RIOT builds on the *sock* API (see Figure 6) to interface with the underlying network stack. Message-related operations to compose DNS queries and parse DNS responses follow a modular and reusable design. For code simplicity, this client uses the synchronous version of *sock*, so it

blocks on requests until a response arrives or a timeout occurs. We extend this implementation to support asynchronous calls to *sock* for non-blocking queries. This enhancement improves the reactivity of networked applications and yields a better comparability with the asynchronous CoAP requests. For better comparability with DoC, we also support the same retransmission algorithm as CoAP [49], *i.e.,* four retransmissions using an exponential back-off.

DNS over DTLS. Our DNS over DTLS client interfaces with *sock_dtls* (see Figure 6). A protected session between two peers establishes on-demand before the first packet exchange, and session state is freed on an explicit teardown by closing the DTLS socket. This setup reuses all the network-related code and message buffers from the DNS over UDP setup.

The *sock* interface integrates with the external library *TinyDTLS*, which provides the DTLS state machine, packet parsing, and all relevant cryptographic operations. It supports both Pre-shared Keys (PSK) using AES and Elliptic Curve Cryptography (ECC) with an Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) key exchange.

6 COMPARISON OF LOW-POWER DNS TRANSPORTS

In this section, we evaluate memory usage, packet sizes, and resolution times of DoC and compare with DNS over UDP and DNS over DTLS in different communication setups. Our DoC configurations include the unencrypted use, CoAPS, and OSCORE, using the FETCH, GET, and POST methods.

6.1 Setup

Hardware and Software Platform. We conduct our experiments in the FIT IoT-LAB testbed [1], which supports a variety of IoT hardware environments. We choose nodes from the *Grenoble* site, since the physical stretch makes it a good candidate for multi-hop measurements. Our platform features a Cortex-M3 MCU with 64 kBytes of RAM, 512 kBytes of ROM [50], and an IEEE 802.15.4 radio [31]. The radio is configured to automatically handle link layer retransmissions and acknowledgments.

As the base for our experiments, we use RIOT (`2022.01`) with the software components described in Section 5. We stress-test each of the deployments by using the asynchronous protocol features that allow for concurrently pending queries on a device. We modify a few RIOT configuration parameters to accommodate the number of queries in the air, specifically internal queue sizes to hold multiple packets. Appendix D lists all modified parameters.

We use the current standard DTLSv1.2 [42] in our evaluations. For consistent measurements, we pre-initialize DTLS sessions and OSCORE replay windows on all endpoints before starting experiments. To prevent side effects such as lost requests or prolonged timeouts due to session re-initialization, we increase both the DTLS session timeout and the OSCORE replay window size. This proved useful when measuring the protocol effects during long experiment runs. To improve comparability between protocol implementations of RIOT (clients) and LINUX (resolver), we deactivate the stateful address compression in 6LoWPAN and set the traffic class and flow label IPv6 header fields to `0`, which allows for eliding.

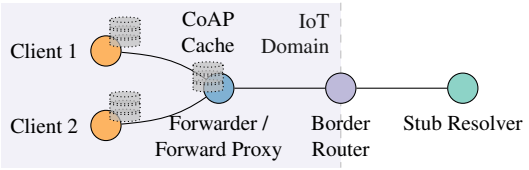


Figure 7: The IoT evaluation scenario in Sections 6 and 7.

In all experiment runs, we measure the actual name resolutions within the IoT network, and exclude the resolution times to external DNS servers.

Topology description. We construct a topology with two wireless hops (see Figure 7), in which two DNS clients communicate with a DNS stub resolver via a forwarder and a border router. The forwarder is either configured as an opaque IPv6 router, or as a CoAP forward proxy with caching capabilities. The border router node is a device identical to the DNS clients and the forwarder. It further connects to the host machine of the DNS resolver via Ethernet that is encapsulated in a TCP-tunneled UART connection, which is the default method to access testbed nodes from a frontend host machine. The DNS resolver consists of a simple Python implementation that uses standard libraries, such as *dnspython* and *aiocoap*. The routes in the wireless domain are constructed using the RPL routing protocol [20].

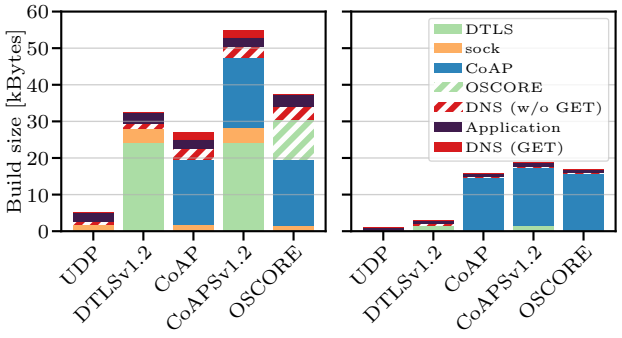
Protocol settings. We evaluate the following DNS transports (short name tags in parentheses): (*i*) DNS over UDP (**UDP**) (*ii*) DNS over DTLS (**DTLSv1.2**), (*iii*) DNS over unencrypted CoAP (**CoAP**), (*iv*) DNS over CoAP over DTLS (**CoAPSv1.2**), and (*v*) DNS over OSCORE (**OSCORE**). We assess CoAP and CoAPSv1.2 with the FETCH, GET, and POST methods, for OSCORE we use only FETCH since our DNS over OSCORE implementation does not support GET (see Section 4). With DTLSv1.2 we use the `AES-128-CCM-8` cipher suite [30] and with OSCORE the `AES-CCM-16-64-128` cipher mode [45], as these are the most comparable options. Both pre-shared key lengths are 9 bytes.

Communication setup. We query 50 names following the pattern `<id>.id.exp.example.org`, where `<id>` is a five-digit number that increments. These names have a length of 24 characters, which is the median of the empirically analyzed names in Section 3 The query rate is Poisson-distributed with $\lambda = 5$ queries / s.

We request `A` and `AAAA` records for each name in separate runs. For the CoAP-based transport, the requested DNS resource is `/dns`. For block-wise transfers, we statically set the block size for both requests and responses to 16, 32, and 64 bytes, respectively. Block size 64 was only used with `AAAA` records, as only the responses for those exceed 64 bytes in the CoAP payload. All runs are repeated 10 times.

6.2 Memory Consumption

We first inspect the memory consumption on our target platform for the DNS requester application of each transport. Since the asynchronous request contexts consume a disproportionate amount of RAM compared to the core functions of each protocol, we limit the maximum number of these contexts to one. We use version



(a) ROM (b) RAM

Figure 8: Memory consumption of each DNS transport. The DNS over CoAP plots are compiled without GET method support, "DNS (GET)" marks the overhead GET introduces.

`9-2019-q4-major` of the *GNU ARM Embedded Toolchain* (which includes GCC v9.2.1)—the recommended toolchain for RIOT `2022.01`. RIOT ships `cosy` to dissect the memory usage of a RIOT firmware image from module level down to function and variable level. As our software platform does not use any dynamic memory allocation, we do not consider heap allocation. For the ROM information we sum up the respective object sizes in the `.text` and `.data` sections of the RIOT image and for the RAM information the `.data` and `.bss` sections.

We group the modules of RIOT according to the following categorization. *Application* contains all the machine code instructions and state information of the experiment application. *DNS* contains the code of each DNS over X implementation, including the shared DNS message parser and composer, as well as URI parsing. As the GET method in DNS over CoAP adds a significant amount of memory for URI template processing, this is separately shown. *OSCORE* contains the code of libOSCORE including its dependencies. *CoAP* contains the code of the gCoAP library and its dependencies. *sock* contains the code of the sock API implementation for the GNRC network stack, as well as the sock_dtls implementation for TinyDTLS. This was included to account for the different build sizes when using DTLS. *DTLS* contains the code of TinyDTLS including its dependencies.

Figure 8 displays the RAM and ROM consumption for the selected protocols. The encrypted transports add a considerable amount of ROM—about 24 kBytes in the case of DTLS and about 11 kBytes in the case of OSCORE—and in the case of DTLS also about 1.5 kBytes of RAM. Notably, the DTLS part of the firmware expects more than twice the memory space of the OSCORE part. This is due to DTLS requiring its own message layer, as well as asymmetric cryptography, to establish a handshake, which is not present in OSCORE.

GET support adds about 2 kBytes of ROM and 173 bytes of RAM to the overall size. About 1 kByte of this ROM contributes the URI template processor. The remainder relates to the different message handling required for the GET request, while the Content-Format option is elided.

6.3 Packet Sizes

We now measure the packet sizes of the DNS messages on all transports by capturing the IEEE 802.15.4 frames using the `sniffer_aggregator` tool of the FIT IoT-LAB testbed. Figure 9 displays the packet dissection for each packet type, segmented into the individual communication layers. Both IPv6 and UDP headers are compressed within the 6LoWPAN header, which we group with the IEEE 802.15.4 MAC header for the sake of simplicity. The maximum Service Data Unit (SDU) of IEEE 802.15.4 is marked in each plot by a red dashed line. 6LoWPAN fragments larger IEEE 802.15.4 frames, producing additional MAC and 6LoWPAN headers for each generated fragment. We represent each additional fragment with its headers above the red marker line.

We see three distinct sizes of DNS messages in our experiments. DNS queries requesting either an `A` or `AAAA` record from the DNS resolver. These queries are identical in size and only differ in their query types (`A` vs. `AAAA`). Respective responses contain either an `A` or `AAAA` record, which vary in size due to the IP address lengths.

Figure 9 further includes packet sizes of the DTLSv1.2 handshake and the OSCORE replay window initialization. We observe that DTLS—both with DTLSv1.2 and CoAPSv1.2—is at a disadvantage as the handshake messages alone already cause fragmentation and multiply the likelihood for packet loss during the session establishment.

DNS queries are `base64`-encoded within the GET method. This inflates requests to a size that is approximately 1.5 times larger than binary FETCH or POST queries. As such, with either CoAP-based transport a DNS query using GET will always be fragmented. Likewise, when `AAAA` records are requested the response will always be fragmented. For CoAPSv1.2, little room is left for the DNS message itself in either message format before reaching the fragmentation limit. The same, however, is also true for OSCORE, if the Echo option required for the replay window initialization is carried in the request. Appendix E adds supplementary dissections for queries using GET and for block-wise transfer.

6.4 Name Resolution Times

Next, we evaluate the name resolution times for each protocol. For this, we measure the time from issuing the query by the DNS client until the IP address is parsed in the response.

Figure 10 summarizes the distributions of resolution times for our protocols. We observe that the different transports form distinct groups in their temporal distributions due to the different packet sizes and the resulting 6LoWPAN fragmentation. For UDP requesting an `A` record no packet is fragmented and names resolve fastest. 85% of the queries resolve in less than 250 ms, all complete within 20 s. Requesting an `AAAA` record, though, plain UDP compares to unencrypted CoAP with the FETCH or POST method. The query is not fragmented, but the response is. For these transports and methods, only 65–70% of the names resolve below 250 ms, but 99% of names resolve after 20 s. The last group consists of those transports and methods, for which both queries and responses fragment. Unencrypted CoAP with GET, as well as DTLSv1.2, CoAPSv1.2, and OSCORE perform all within approximately 7% from each other, with 42–49% of `A` records and 37–45% of `AAAA` records resolve below 250 ms. All require at most 41–44 s to resolve 99% of names.

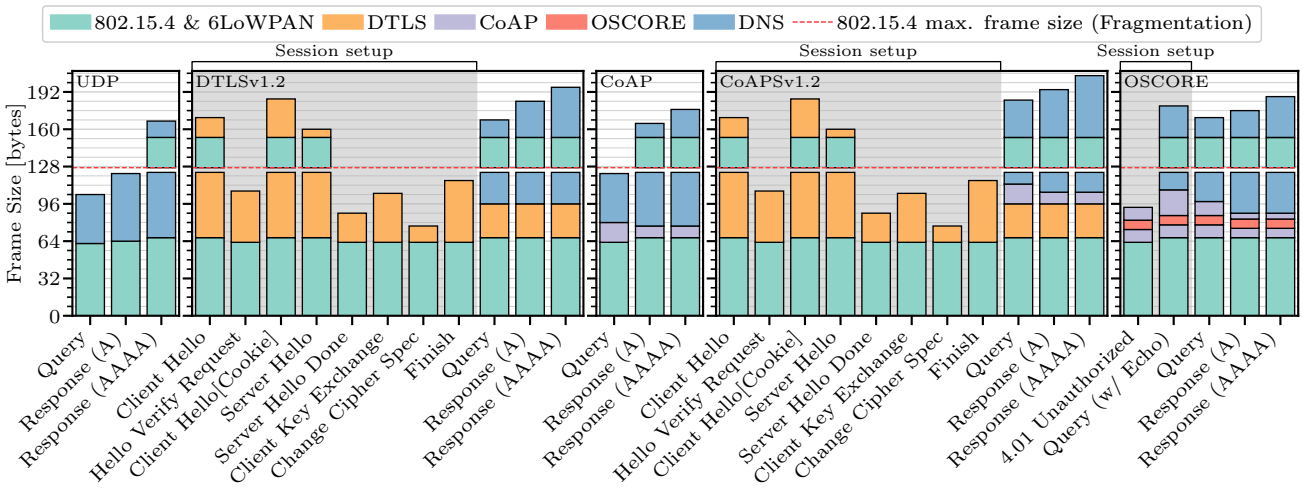
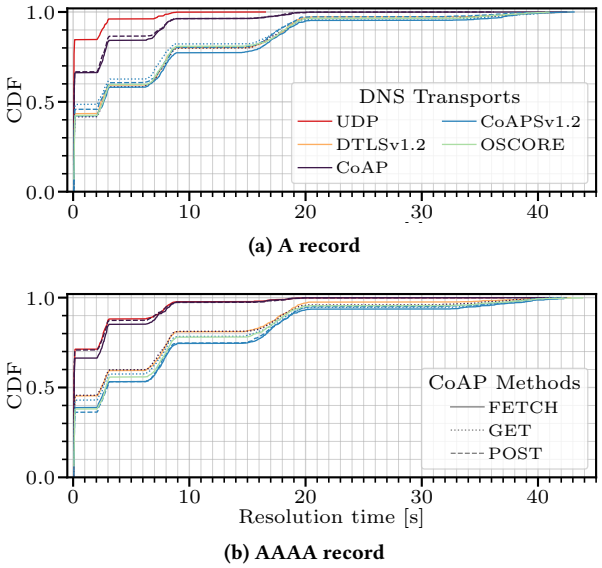


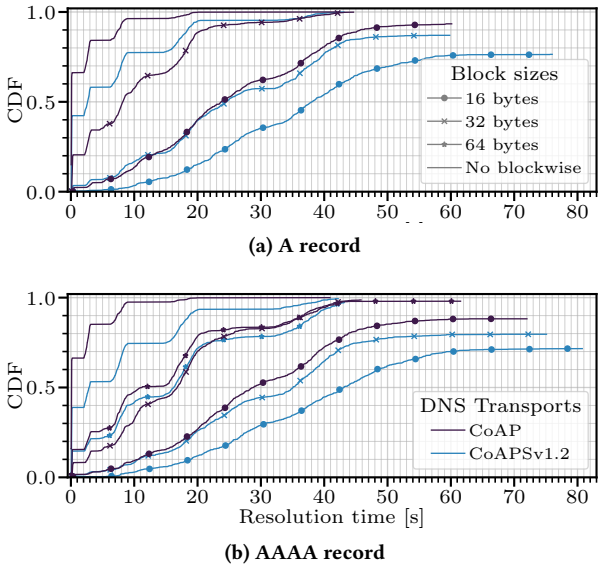
Figure 9: Maximum link layer packet sizes for each transport for the name resolutions of name `XXXXX.id.exp.example.org` (24 characters) for a single record (`A` and `AAAA` respectively).



(a) A record

(b) AAAA record

Figure 10: Resolution times for 50 queries (Poisson distributed with $\lambda = 5$ queries / s).



(a) A record

(b) AAAA record

Figure 11: Resolution times for 50 queries using FETCH with block-wise transfer. Block size 64 was only used with `AAAA` records, as DNS responses for `A` record stay below 64.

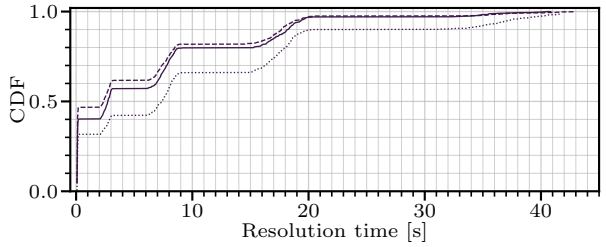
We plot the temporal distributions for `A` and `AAAA` records with block-wise transfer using FETCH requests for CoAP and CoAPSv1.2 in Figure 11. For easier comparison, we include the distributions of Figure 10, which do not utilize the block-wise transfer, but we emphasize the increased range of the x-axis. We observe that the performance decreases with smaller block sizes. With a block size of 16 bytes, only ≈ 90% and 60–70% of name resolutions complete in total for CoAP and CoAPSv1.2, respectively. This is due to congestion emerging in the wireless medium, which increases the probability of packet loss for transfers with higher block counts.

7 EVALUATION ON CACHING SCHEMES FOR DOC

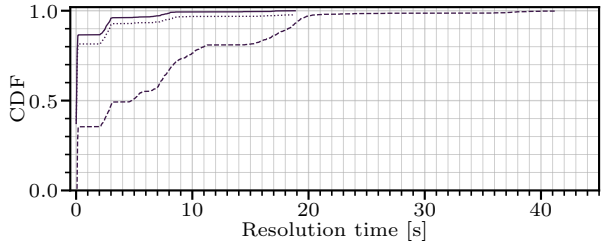
In this section, we perform a comparative assessment of caching as introduced in Section 4 using a multihop network.

7.1 Setup

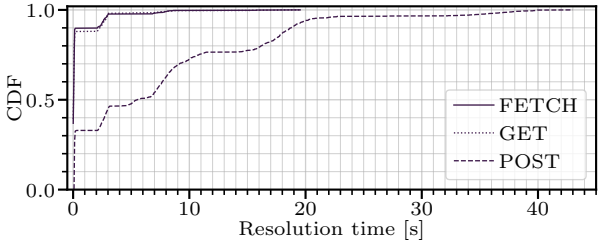
We base our evaluation on the setup described in Section 6.1, except that clients now query 50 `AAAA` records of eight distinct names to showcase the cache utilization.



(a) Opaque forwarder



(b) DoH-like (w/ caching)



(c) EOL TTLs (w/ caching)

Figure 12: Resolution times for four AAAA record queries (Poisson-distributed with $\lambda = 5$ queries / s) with and without CoAP forward proxy and caching.

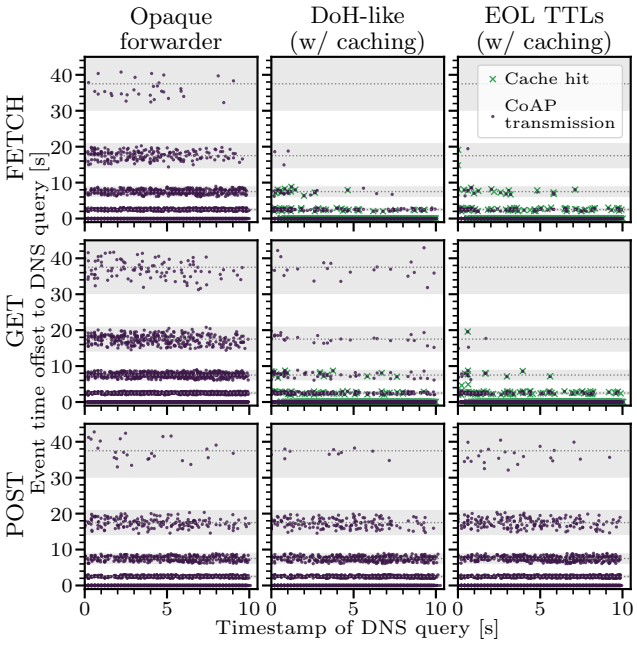


Figure 13: CoAP events of message (re-)transmissions at the client in relation to the time of the initial DNS query. Retransmissions follow an exponential back-off and scatter within the time bounds marked by the gray areas. Green crosses mark cache hits for DNS queries at client and proxy.

We compare three scenarios, each consists of ten runs for each scheme. In the *opaque forwarder* scenario, the forwarder is an IPv6 router. This baseline scenario does not deploy caches. The two other scenarios reflect caching, *i.e.,* the forwarder is configured as a CoAP forward proxy, which provides a CoAP response cache and the clients also provision a CoAP cache. In the *DoH-like* caching scenario, we configure the DoC server operations compliant with [17], *i.e.,* we set the Max-Age option in the CoAP header to the minimum value of all TTLs in the DNS response. In the third *EOL TTLs* caching scenario, we configure the DoC server to set the Max-Age option as proposed in Section 4.

We only evaluate unencrypted CoAP since any security overhead increases latencies and packet loss, which would sidetrack from the cache analysis. The DNS resolver returns four AAAA records for each name query. This causes 6LoWPAN fragmentation with three fragments. All four records use the same TTL picked uniformly random between 2 and 8 s.

7.2 Name Resolution Time

Figure 12 shows the distributions of resolution times over all runs for each scenario and each CoAP method. We observe a pronounced staircase pattern for all methods in the *opaque forwarder* scenario, which is a result of packet loss due to the congested wireless medium. Since all 6LoWPAN fragments are required for packet reassembly, the error probabilities accumulate for CoAP responses. Nevertheless, CoAP retransmissions recover almost all losses, although at a price of increased latency.

For the *caching scenarios*, the resolution times for the GET and FETCH methods significantly decrease compared to the *opaque forwarder* scenario. As responses to POST requests are not cachable, performance degrades for this method due to the increased complexity of the intermediary proxies.

The *EOL TTLs* scenario shows a better performance than the *DoH-like* scenario as it benefits from the validation mechanism of CoAP (see Section 4). ≈ 10% more queries are received within 2 s for the GET method, in which the queries are fragmented (see Figure 9). With FETCH, ≈ 5% more queries are received in less than 2 s.

7.3 Transport Retransmissions and Cache Hits

In our last experiment, we want to quantify the link stress that clients generate due to corrective actions in our three scenarios. We track the timestamp of each event at which a client initiates a CoAP transmission during the experiment duration in Figure 13. Likewise, we track the timestamp for cache hits including re-validations of stale entries at the clients and the proxy. For all events, we

calculate the time offset to the start time of the respective DNS query. The original requests have a negligible offset in the range of microseconds, and since retransmissions follow a random exponential back-off mechanism [49, Section 4.2], their time offsets scatter within the pre-calculated gray areas in Figure 13. We mark cache events by green crosses.

For the *opaque forwarder* scenario, we observe about 50% more retransmissions for both GET and FETCH than for the *caching scenarios*. With GET, the retransmissions in the third and fourth iteration even increase by 7% compared to POST and FETCH. This is a result of 6LoWPAN fragmenting the query (see Section 6.3).

When utilizing client and proxy caches, the number of retransmissions reduces considerably. In the *EOL TTLs* scenario, cache hits are able to complete requests without requiring more than one retransmission for the majority of DNS queries. For both caching scenarios, POST requests do not utilize response caches, which degrades their performance to the level of the *opaque forwarder*.

8 DISCUSSION AND OPTIMIZATION POTENTIALS

With the core DoC protocol at hand, a series of potential optimizations and protocol enhancements become attainable, which—along with open questions and shortcomings—we discuss in the following.

How to reduce the DNS packet overhead? DNS messages account for the largest parts of the packets in DoC. Hence, DNS compression schemes beyond the generic methods of 6LoWPAN [33] or SCHC [32] promise enhanced efficiency.

One evident enhancement arrives from operators preferring shorter host names in communication with constrained IoT devices. Nevertheless, further compression of superfluous DNS header fields is required. Klauck and Kirsche [23] proposed a compression for mDNS/DNS-SD messages for 6LoWPAN, but their approach focuses on compatibility with the DNS wire format. DoC offers the opportunity to use different message formats via its use of a new Content-Format. Specifically, the Concise Binary Object Format (CBOR) [7] offers a standardized, structured, and space-efficient encoding.

In addition, CoAP messages carry a transactional context that matches a reply to its request. Exploiting this, we argue for the following practices to reduce packet overhead. A DoC query could be encoded as a CBOR array, containing up to three entries: the name (as text string), an optional record type (as unsigned integer), and an optional record class (as unsigned integer). If record type and class are elided, DoC implies `AAAA` and `IN`, respectively. A DoC response can be matched to the request. Hence, the encoding could use only one CBOR array, which contains the DNS answer section. This could be nested for several, separate answer sections. An answer section includes a name, TTL, and a class using the space-efficient encoding of CBOR. For an answer with two arrays, DoC additionally identifies the question section (formatted as in the DoC query).

In our evaluation, we could verify that the wire-format of an `AAAA` response packet compresses from 70 bytes down to 21 bytes—a reduction by 70% (see Figure 9).

How to protect the integrity of the DNS TTLs? DoC depends on the CoAP Max-Age option to track elapsed caching time, which a DoC client then uses to decrement DNS TTLs. The integrity of the Max-Age option, however, cannot be guaranteed, because it is altered on—potentially untrusted—intermediaries. An adversary with malicious intent, or a faulty proxy behavior may impair TTLs on the client by using incorrect Max-Age values.

For *EOL TTLs*, a potential mitigation is to include a second Max-Age value that is protected by OSCORE. A DoC client compares both Max-Age values, deduces inconsistent modifications, *e.g.,* larger values than the original TTLs, and discards the response on a failed consistency check. For the *DoH-like* caching scheme, responses include the original TTLs, which can be used to perform consistency checks instead of including an additional Max-Age value. This approach mitigates the use of outdated DNS records, but still allows for unauthorized reduction of TTLs, which affects the caching performance.

How to ensure an efficient cache re-validation? A naïve ETag generation calculates a hash over the CoAP message payload to identify a specific DNS response. However, apart from changing TTLs, DNS resolvers often rearrange resource records within responses, *e.g.,* for load balancing reasons. This modifies the binary representation of DNS messages, and thus their resulting ETag values. We argue that a more comprehensive ETag generation approach is beneficial. One approach is to sort incoming records at the DoC server and select a random record at the DoC client to achieve similar load balancing effects.

9 CONCLUSION

The current constrained IoT lacks a protocol for privacy-friendly, secure name resolution such as DoH for the regular Internet. In this paper, we presented DNS over CoAP (DoC), which leverages the rich feature set of the CoAP protocol suite to provide an energy-efficient and end-to-end protected name resolution for constrained networks. In a comprehensive analysis we compared DNS over various transports including UDP, CoAP, DTLS, CoAP over DTLS, and CoAP with OSCORE in experiments based on full-featured implementations.

Our findings indicate that name resolution performance is primarily driven by packet sizes. While CoAP has a space-efficient protocol encoding, the choice of the request method largely impacts the packet overhead and the choice of additional CoAP features. The FETCH method is preferred over GET and POST as it allows for block-wise transfer of queries and for caching of responses. GET, the other cachable alternative, on the contrary requires an ASCII-based `base64` encoding of the query, which increases the packet size and in turn leads to higher loss rates. Further, GET significantly increases code complexity, as a `base64` encoder, a URI template processor, and the inclusion of further options to the CoAP header become necessary. OSCORE is preferred over CoAPS for protecting DoC, since OSCORE seamlessly integrates with the semantics of CoAP and its header encoding, leading to smaller packets and less memory usage. Transforming DNS responses into a deterministic form favors the cache validation model of CoAP and reduces bandwidth needs and loss rates further.

Future work shall optimize the coding efficiency by defining a comprehensive compression scheme for DNS messages. This will

enfold impact with less fragmentation and higher reliability in low-power and lossy regimes. To enable service discovery, we will also focus on a DoC integration for mDNS protected by Group OSCORE.

REFERENCES

[1] Cedric Adjih, Emmanuel Baccelli, Eric Fleury, Gaetan Harter, Nathalie Mitton, Thomas Noel, Roger Pissard-Gibollet, Frederic Saint-Marcel, Guillaume Schreiner, Julien Vandaele, and Thomas Watteyne. FIT IoT-LAB: A large scale open experimental IoT testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 459–464, Piscataway, NJ, USA, Dec 2015. IEEE Press.

[2] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. Sok: Security evaluation of home-based iot deployments. In *IEEE S&P 2019*, pages 1362–1380, 2019.

[3] Christian Amsuess and Marco Tiloca. Cacheable OSCORE. Internet-Draft – work in progress 04, IETF, March 2022.

[4] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the Mirai Botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC, August 2017. USENIX Association.

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

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

[7] C. Bormann and P. Hoffman. Concise Binary Object Representation (CBOR). RFC 8949, IETF, December 2020.

[8] C. Bormann and Z. Shelby. Block-Wise Transfers in the Constrained Application Protocol (CoAP). RFC 7959, IETF, August 2016.

[9] S. Cheshire and M. Krochmal. DNS-Based Service Discovery. RFC 6763, IETF, February 2013.

[10] S. Cheshire and M. Krochmal. Multicast DNS. RFC 6762, IETF, February 2013.

[11] Hesselman Cristian, Kaeo Merike, Chapin Lyman, Claffy Kimberly, Seiden Mark, McPherson Danny, Piscitello Dave, McConachie Andrew, April Tim, Latour Jacques, and Rasmussen Rod. The DNS in IoT: Opportunities, Risks, and Challenges. *IEEE Internet Computing*, 24(4):23–32, 2020.

[12] J. Dickinson, S. Dickinson, R. Bellis, A. Mankin, and D. Wessels. DNS Transport over TCP - Implementation Requirements. RFC 7766, IETF, March 2016.

[13] O. Gimenez and I. Petrov. Static Context Header Compression and Fragmentation (SCHC) over LoRaWAN. RFC 9011, IETF, April 2021.

[14] C. Gomez, J. Crowcroft, and M. Scharf. TCP Usage Guidance in the Internet of Things (IoT). RFC 9006, IETF, March 2021.

[15] Cenk Gündogan, Christian Amsüss, Thomas C. Schmidt, and Matthias Wählisch. Content Object Security in the Internet of Things: Challenges, Prospects, and Emerging Solutions. *IEEE Transactions on Network and Service Management (TNSM)*, 19(1):538–553, March 2022.

[16] Hang Guo and John Heidemann. Detecting IoT Devices in the Internet. *IEEE/ACM Transactions on Networking*, 28(5):2323–2336, October 2020.

[17] P. Hoffman and P. McManus. DNS Queries over HTTPS (DoH). RFC 8484, IETF, October 2018.

[18] Austin Hounsel, Kevin Borgolte, Paul Schmitt, Jordan Holland, and Nick Feamster. Comparing the Effects of DNS, DoT, and DoH on Web Performance. In *Proceedings of The Web Conference 2020*, WWW '20, pages 562–572, New York, NY, USA, 2020. ACM.

[19] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman. Specification for DNS over Transport Layer Security (TLS). RFC 7858, IETF, May 2016.

[20] J. Hui and JP. Vasseur. The Routing Protocol for Low-Power and Lossy Networks (RPL) Option for Carrying RPL Information in Data-Plane Datagrams. RFC 6553, IETF, March 2012.

[21] C. Huitema, S. Dickinson, and A. Mankin. DNS over Dedicated QUIC Connections. RFC 9250, IETF, May 2022.

[22] IEEE 802.15 Working Group. IEEE Standard for Low-Rate Wireless Networks. Technical Report IEEE Std 802.15.4™–2015 (Revision of IEEE Std 802.15.4-2011), IEEE, New York, NY, USA, 2016.

[23] Ronny Klauck and Michael Kirsche. Enhanced dns message compression - optimizing mdns/dns-sd for the use in 6lowpans. In *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 596–601, 2013.

[24] Mike Kosek, Trinh Viet Doan, Malte Granderath, and Vaibhav Bajpai. One to Rule Them All? A First Look at DNS over QUIC. In *Proc. of PAM*, volume 13210 of *LNCS*, pages 537–551, Cham, 2022. Springer.

[25] Martine S. Lenders, Christian Amsüss, Cenk Gündogan, Thomas C. Schmidt, and Matthias Wählisch. DNS over CoAP (DoC). IETF Internet Draft – work in progress 04, IETF, July 2022.

[26] Martine S. Lenders, Thomas C. Schmidt, and Matthias Wählisch. Fragment Forwarding in Lossy Networks. *IEEE Access*, 9:143969 – 143987, October 2021.

[27] LoRa Alliance. RP002-1.0.0 LoRaWAN™Regional Parameters. Technical report, LoRa Alliance, November 2019.

[28] Chaoyi Lu, Baojun Liu, Zhou Li, Shuang Hao, Haixin Duan, Mingming Zhang, Chunying Leng, Ying Liu, Zaifeng Zhang, and Jianping Wu. An End-to-End, Large-Scale Measurement of DNS-over-Encryption: How Far Have We Come? In *Proc. of ACM IMC*, pages 22–35, New York, NY, USA, 2019. ACM.

[29] Jiarun Mao, Michael Rabinovich, and Kyle Schomp. Assessing Support for DNS-over-TCP in the Wild. In *Proc. of PAM*, volume 13210 of *LNCS*, pages 487–517, Cham, 2022. Springer.

[30] D. McGrew and D. Bailey. AES-CCM Cipher Suites for Transport Layer Security (TLS). RFC 6655, IETF, July 2012.

[31] Microchip. *Low Power 2.4 GHz Transceiver for ZigBee, IEEE 802.15.4, 6LoWPAN, RF4CE, SP100, WirelessHART, and ISM Applications (AT86RF231)*, September 2009. Rev.8111C.

[32] A. Minaburo, L. Toutain, C. Gomez, D. Barthel, and JC. Zuniga. SCHC: Generic Framework for Static Context Header Compression and Fragmentation. RFC 8724, IETF, April 2020.

[33] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944, IETF, September 2007.

[34] B. Moran, H. Tschofenig, D. Brown, and M. Meriac. A Firmware Update Architecture for Internet of Things. RFC 9019, IETF, April 2021.

[35] Palo Alto Networks. 2020 Unit 42 IoT Threat Report. https://unit42.paloaltonetworks.com/iot-threat-report-2020/, 2020. Retrieved 2021-10-23.

[36] Roberto Perdisci, Thomas Papastergiou, Omar Alrawi, and Manos Antonakakis. IoTFinder: Efficient Large-Scale Identification of IoT Devices via Passive DNS Traffic Analysis. In *IEEE EuroS&P 2020*, pages 474–489, 2020.

[37] P. Phaal, S. Panchen, and N. McKee. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. RFC 3176, IETF, September 2001.

[38] A. Rahman and E. Dijk. Group Communication for the Constrained Application Protocol (CoAP). RFC 7390, IETF, October 2014.

[39] T. Reddy, D. Wing, and P. Patil. DNS over Datagram Transport Layer Security (DTLS). RFC 8094, IETF, February 2017.

[40] Jingjing Ren, Daniel J. Dubois, David Choffnes, Anna Maria Mandalari, Roman Kolcun, and Hamed Haddadi. Information Exposure for Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach. In *Proc. of the Internet Measurement Conference (IMC)*, 2019.

[41] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, IETF, August 2018.

[42] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347, IETF, January 2012.

[43] E. Rescorla, H. Tschofenig, T. Fossati, and A. Kraus. Connection Identifier for DTLS 1.2. RFC 9146, IETF, March 2022.

[44] Christian Rossow. Amplification Hell: Revisiting Network Protocols for DDoS Abuse. In *Proc. of NDSS*. Internet Society, 2014.

[45] J. Schaad. CBOR Object Signing and Encryption (COSE). RFC 8152, IETF, July 2017.

[46] Benjamin Schwartz, Mike Bishop, and Erik Nygren. Service binding and parameter specification via the DNS (DNS SVCB and HTTPS RRs). Internet-Draft – work in progress 10, IETF, May 2022.

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

[48] Senrio. 400,000 Publicly Available IoT Devices Vulnerable to Single Flaw. https://blog.senr.io/blog/400000-publicly-available-iot-devices-vulnerable-to-single-flaw, July 2016. Retrieved 2022-04-12.

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

[50] STMicroelectronics. *High-density performance line ARM©-based 32-bit MCU with 256 to 512KB Flash, USB, CAN, 11 timers, 3 ADCs, 13 communication interfaces (STM32F103REY)*, July 2018. DS5792 Rev 13.

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

[52] P. van der Stok, C. Bormann, and A. Sehgal. PATCH and FETCH Methods for the Constrained Application Protocol (CoAP). RFC 8132, IETF, April 2017.

[53] Malisa Vucinic, Goeran Selander, John Mattsson, and Dan Garcia-Carillo. Requirements for a Lightweight AKE for OSCORE. Internet-Draft – work in progress 04, IETF, June 2020.

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

[55] Liang Zhu, Zi Hu, John Heidemann, Duane Wessels, Allison Mankin, and Nikita Somaiya. Connection-Oriented DNS to Improve Privacy and Security. In *Proc. of IEEE Symposium on Security and Privacy*, pages 171–186, Piscataway, NJ, USA,

May 2015. IEEE.

A FETCH SUPPORT IN COAP IMPLEMENTATIONS

The FETCH method was not part of the original CoAP specification in RFC 7252 [49] but introduced in RFC 8132 [52]. To understand which common CoAP implementations support FETCH and as such allow deployment of our proposed DoC design (see Section 4), we reviewed CoAP implementations listed in https://coap.technology/impls.html. We excluded all proprietary implementations and those that were not updated after April 2021, and added CoAP implementations of major IoT operating systems such as RIOT or Zephyr.

11 out of 15 implementations support FETCH (see Table 5). Given this majority, we argue that using FETCH is not an artificial design choice to enable both caching and block-wise transfer in parallel. Regarding the implementations that miss FETCH, adding FETCH support would be easy. The FETCH CoAP header needs to be defined and, for CoAP caches, the cache key calculation must be updated to include the body of a FETCH request. A CoAP implementation in Zephyr that was recently extended to support FETCH needed 19 additional lines of code.

Table 5: Overview of common CoAP implementations and their support of CoAP FETCH in April 2022.

CoAP implementation	FETCH Support
aiocoap	✓
Californium	✓
Californium (Leshan)	✓
coap-lite	✓
gcoap (RIOT)	✓
libcoap	✓
Lobaro CoAP	✓
nanocoap (RIOT)	✓
node-coap	✓
Waher.Networking.CoAP	✓
Zephyr CoAP	✓
cantcoap	✘
Erbium (Wakaama)	✘
Go-CoAP (go-ocf)	✘
mbed-CoAP	✘

B EOL TTLS EXTENSION FOR MULTIPLE RRSETS

A DNS response may include multiple resource record sets (RRsets) with varying TTLs. Instead of setting all TTLs to zero (see Section 4.2), we provide an alternative method that preserves their differences: Max-Age is set to the minimum TTL of all resource records, and this value is subtracted from each TTL.

$$\text{Max-Age} = \min(\text{TTLs})$$
$$\text{TTL}_{\text{new}} = \text{TTL}_{\text{old}} - \text{Max-Age}$$

This will result in the original minimum TTL to be zero, so if all TTLs are equal, we get the same message as proposed in Section 4.2. To reverse, the DoC client adds the (potentially decremented) Max-Age to all TTLs in the DNS message:

$$\text{TTL}_{\text{new}} = \text{TTL}_{\text{old}} + \text{Max-Age}$$

The advantage of this method is a better utilization of CoAP cache re-validations, since the binary representation remains identical for responses where all TTLs change with the same offset.

C DNS MESSAGE FORMAT

Throughout our evaluation in Section 6, we expected the message size for a queried name of 24 characters to be known: A query in our setup is always of size 42 bytes, a response for an `A` record always of size 58 bytes, and a response of an `AAAA` record always of size 70 bytes. In Figure 14 we show how these messages are comprised and where each byte stems from.

D COMPILE-TIME PARAMETERS IN RIOT

For our evaluation in Sections 6 and 7, we changed certain compile-time configuration parameters of RIOT from their default value, using RIOT's Kconfig integration Depending on the transport. The parameters in question can be seen in Table 6.

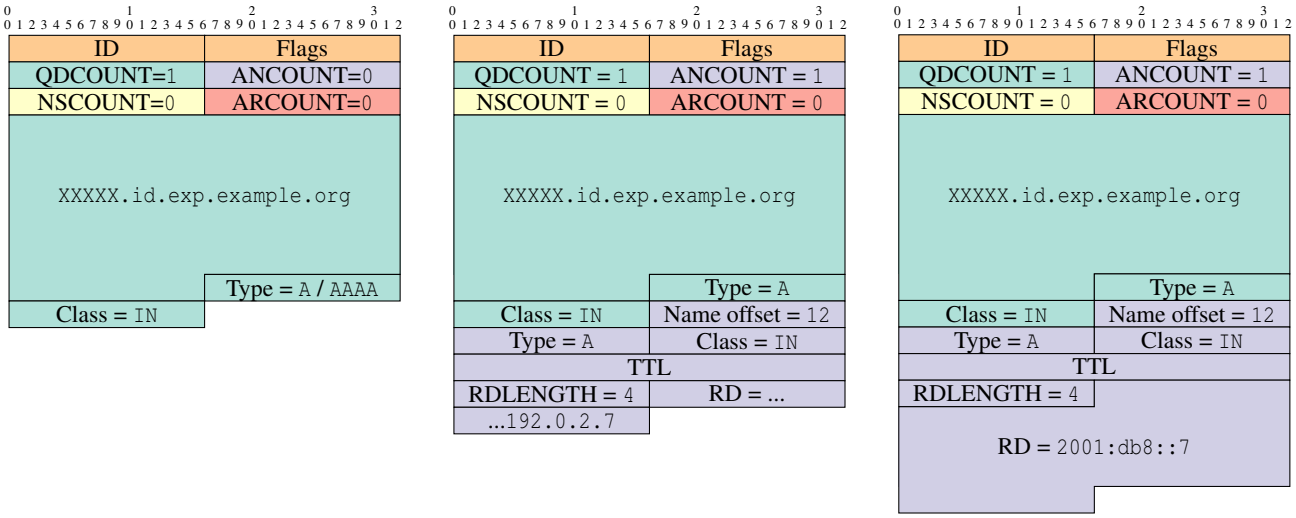
E ANALYSIS OF COAP PACKET SIZES

In Section 6.3 we evaluated the packet size, but did not go into detail for the different CoAP message types. In Figure 15 we show the packet sizes with block-wise transfer for DNS over CoAP.

Block-wise transfer offers a solution for the CoAP-based transports to prevent fragmentation on the link layer, allowing for confirmable segmentation into blocks within the application layer. However, only the payload can be transferred in blocks. As such, using the GET method, we are unable to send our DNS query in blocks, as it is carried, encoded in `base64` within the CoAP URI-Query option. Consequently, the GET request stays the same in all the block-wise transfer modes shown in Figure 15. With block-wise transfer, we are able to reduce the overall packet size enough to drop below the fragmentation line of 6LoWPAN. Compared to fragmentation in 6LoWPAN, CoAP block-wise transfer provides us with a recovery mechanism, so even if a message is lost, we can recover from that on a block-level, so we do not have to send

Table 6: Changed compile-time parameters in RIOT 2022.01. An asterisk (*) denotes configuration for the proxy. A plus (+) only applies to the dedicated block-wise runs. All other configurations refer to configuration of the clients.

Parameter	Value
`CONFIG_GNRC_PKTBUF_SIZE`	3072
`CONFIG_GCOAP_DNS_BLOCK_SIZE`	8^+ / 16^+ / 32^+ / 64^+
`CONFIG_GCOAP_REQ_WAITING_MAX`	60 / 51^*
`CONFIG_GCOAP_RESEND_BUFS_MAX`	60 / 51^*
`CONFIG_GNRC_IPV6_NIB_NUMOF`	8^*
`CONFIG_NANOCOAP_CACHE_ENTRIES`	50^*
`CONFIG_SOCK_DODTLS_TIMEOUT_MS`	2000
`CONFIG_SOCK_DODTLS_RETRIES`	4
`CONFIG_DTLS_PEER_MAX`	2



(a) DNS query wire-format

(b) DNS response wire-format (A record)

(c) DNS response wire-format (AAAA record)

Figure 14: The DNS message format throughout our evaluation. While `XXXXX.id.exp.example.org` consists of 24 characters in its string representation, in the wire format it comprises of 26 bytes (20 characters, 5 length bytes and 1 terminating 0-byte).

the whole request or response again, in case one single block or fragment gets lost.

For the setup evaluated, a block size of 32 bytes is ideal: 16 bytes makes the blocks smaller and more numerous than necessary and 64 already leads to 6LoWPAN fragmentation.

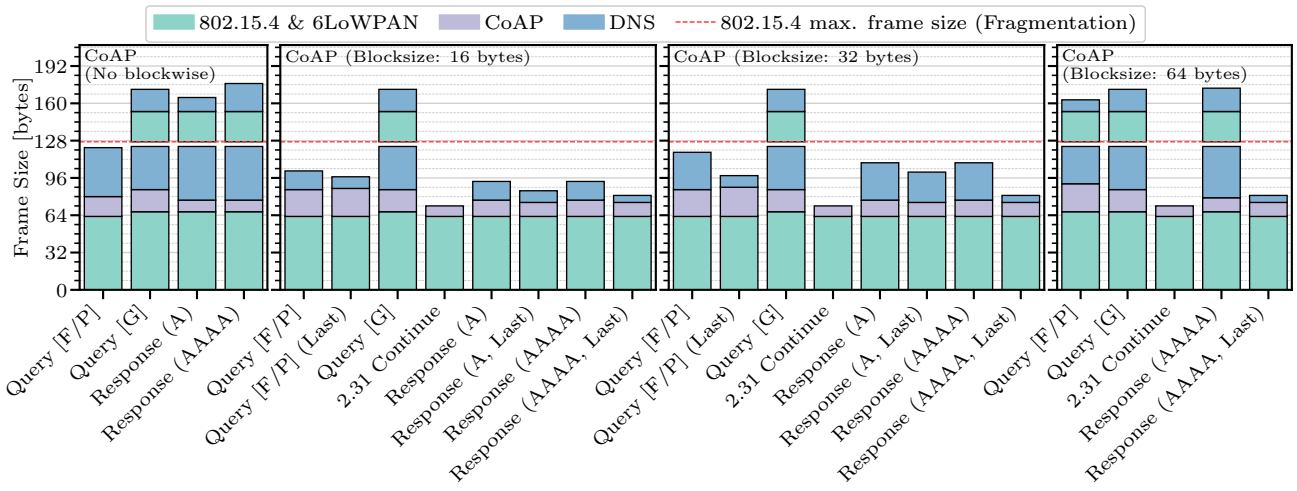


Figure 15: Maximum link layer packet sizes for each transport for the resolution of name `XXXXX.id.exp.example.org` (24 characters) for a single record (`A` and `AAAA` respectively) for different CoAP methods (F = FETCH, G = GET, P = POST) and block sizes. "Last" denotes the size of the last block with block-wise transfer.