Content Object Security in the Internet of Things: Challenges, Prospects, and Emerging Solutions

Cenk Gündoğan, Christian Amsüss, Thomas C. Schmidt, and Matthias Wählisch

***Abstract*—Content objects are confined data elements that carry meaningful information. Massive amounts of content objects are published and exchanged every day on the Internet. The emerging Internet of Things (IoT) augments the network edge with reading sensors and controlling actuators that comprise machine-to-machine communication using small data objects. IoT content objects are often messages that fit into single IPv6 datagram. These IoT messages frequently traverse protocol translators at gateways, which break end-to-end transport and security of Internet protocols. To preserve content security from end to end via gateways and proxies, the IETF recently developed Object Security for Constrained RESTful Environments (OSCORE), which extends the Constrained Application Protocol (CoAP) with content object security features commonly known from Information Centric Networking (ICN).**

This paper revisits the current IoT protocol architectures and presents a comparative analysis of protocol stacks that protect request-response transactions. We discuss features and limitations of the different protocols and analyze emerging functional extensions. We measure the protocol performances of CoAP over Datagram Transport Layer Security (DTLS), OSCORE, and the information-centric Named Data Networking (NDN) protocol on a large-scale IoT testbed in single- and multi-hop scenarios. Our findings indicate that (a) OSCORE improves on CoAP over DTLS in error-prone wireless regimes due to omitting the overhead of maintaining security sessions at endpoints, (b) NDN attains superior robustness and reliability due to its intrinsic network caches and hop-wise retransmissions, and (c) OSCORE/CoAP offers room for improvement and optimization in multiple directions.

***Index Terms*—IoT protocol architecture, CoAP, OSCORE, DTLS, ICN, secure networking, protocol evaluation, network experimentation**

I. Introduction

THE Internet design follows an end-to-end principle [1], which strongly shaped its transport layer. Transport sessions shall establish directly between applications without intermediaries. On transport, (datagram) transport layer security (D)TLS augmented end-to-end sessions with security features following the intend to provide secure channels between application endpoints that are transparent to applications. (D)TLS interception [2], [3], however, breaks the end-to-end paradigm from a security perspective. At the same time, a growing number of use cases demands for application layer gateways and transport assistance, which both hinder end-to-end session security.

Currently, the Internet of Things (IoT) emerges with massive deployments of constrained devices that are shielded behind application gateways. These gateways translate between the Constrained Application Protocol (CoAP) [4] over DTLS [5] and HTTPS, or the Message Queuing Telemetry Transport for Sensor Networks MQTT-SN [6] over DTLS and MQTT [7] over TLS, which require re-authentication and re-encryption. In addition, cryptographic overhead burdens the constrained nodes that interact in low-end wireless transmission systems while being challenged by maintaining security sessions for all the small data transfers. The IoT is thus a use case against end-to-end session security.

We are interested in resolving the tussle of provisioning authenticated and eventually encrypted content to users in networks that cannot maintain persistent, bilaterally trusted relationships. For this, we explore generic methods that bind security credentials directly to content objects instead of to transmission channels: content object security.

Content objects security is an orthogonal approach to secure communication on the Internet. It changes the session-centric paradigm by adding authentication and encryption (if desired) to each data chunk—independent of its endpoints. Content object security in turn allows for content caching and transport translation at gateways, while preserving all properties of data security. In spite of deployment facility, it is desirable to provide such security functions at base protocol layers.

Information-centric Networking was first to introduce content object security on the network layer for the sake of ubiquitous caching. Recently, the IETF Core working group released OSCORE [8], which extends the IoT ecosystem around CoAP to content object security. As application layer protocols, CoAP/OSCORE can still be expected to take a basic role in the IoT, since machine-to-machine communication only requires a reduced feature set and less protocol diversity to serve its selected applications.

In this paper, we revisit the current competing IoT secure protocol architectures and qualitatively evaluate their prospects and potentials. Our long-term objective is to work toward a data-centric, resilient Web of Things. We present and comparatively evaluate the full solution space for exchanging secure content objects in the IoT. This extends our previous work [9] by deepening the discussion, considering latest protocol development, as well as evaluating communication resilience and caching.

Starting from a problem statement and related protocol work in Section II, we present a comprehensive set of im-

Cenk Gündoğan and Thomas C. Schmidt are with the Hamburg University of Applied Sciences (HAW), 20099 Hamburg, Germany (e-mail: {cenk.guendogan, t.schmidt}@haw-hamburg.de).

Christian Amsüss is an independent researcher, Austria (e-mail: christian@amsuess.com).

Matthias Wählisch is with the Freie Universität Berlin, 14195 Berlin, Germany (e-mail: m.waehlisch@fu-berlin.de).

plementations within the RIOT [10] networking subsystem in Section III. Theoretical evaluations of the protocol security features follow in Section IV. Section V analyzes the specific features concerning redundancy, resilience, and recovery of protocol functions. Our network experimentation on a large-scale testbed are discussed in Section VI along with various results that indicate significant performance improvements over CoAP/DTLS by OSCORE as well as NDN. Section VII concludes with an outlook.

II. The Problem of Securing IoT Content and Related Protocol Work

A. Problem Statement

The Internet of Things is evolving to connect numerous, often constrained devices that regularly exchange massive amounts of data. Authenticity and possibly confidentiality of information is of vital interest in a wide range of applications. The problem, though, is that low-end devices need to optimize resources and thus need to minimize cryptographic operations and state while (re-)transmitting packets [11].

At the same time, low-power lossy networks frequently experience packet loss and require retransmissions—multihop transfers often significantly challenge these error-prone regimes [12]. Overhead from cryptographic credentials or signaling security sessions consumes additional energy and may quickly become critical for these low-power devices.

Low-end IoT nodes often operate intermittently to save resources [13]. Duty cycling or energy shortages may force devices into deep sleep with little capacities for saving protocol state or security credentials in non-volatile memory modules. Firmware updates, disruptive environments, or intermittent power availability may repeatedly cause unanticipated system resets. Any of these harsh conditions may lead to a loss of state at endpoints. Once lost, session and security state needs reestablishing to continue operation. Methods for replicating protected data, as well as lightweight recovery mechanisms from state loss including an efficient rediscovery and re-association of networked nodes are hence vital for seamless, perpetual operations: they save computational resources, radio cycles, and preserve system energy.

Many IoT scenarios such as multi-destination control messaging or convergecast sensor readings, but in particular over the air (OTA) firmware updates can take advantage of multi-party communication, which in the wireless IoT often pairs with mobility. Multicast mobility is an asymmetric problem [14]. While the movement of receivers is often easy to compensate by local network reconfigurations, the impact of mobile sources on routing and forwarding is complex and requires assisting measures or services. Secure communication contexts require proper group keying and secure group membership management, which require dedicated treatment on the protocol level if bound to communication endpoints.

In a wider context, trust relationships in the IoT are heterogeneous and change with varying deployment settings. While the exchanging endpoints are often widely distributed (*e.g.,* sensors and a cloud), IoT gateways often need to translate between protocols. If translators are required to re-authenticate and re-encrypt, all communicating parties must pre-establish trust with the IoT gateways in place. This will be a major problem in provider-bound deployments such as 5G.

There are several ways to securely transfer content across the constrained IoT. The most common approach builds on securing the transport layer, which establishes confidentiality and trust between end points and remains neutral with respect to application layer protocols. An option for content object security has been newly developed for CoAP communication between IoT nodes, after a longer research period on information-centric IoT networking had worked out the advantages of securing content objects autonomously. We discuss properties, underlying mechanisms, and protocols for these three approaches in the following three subsections.

B. Transport Layer Security in the IoT

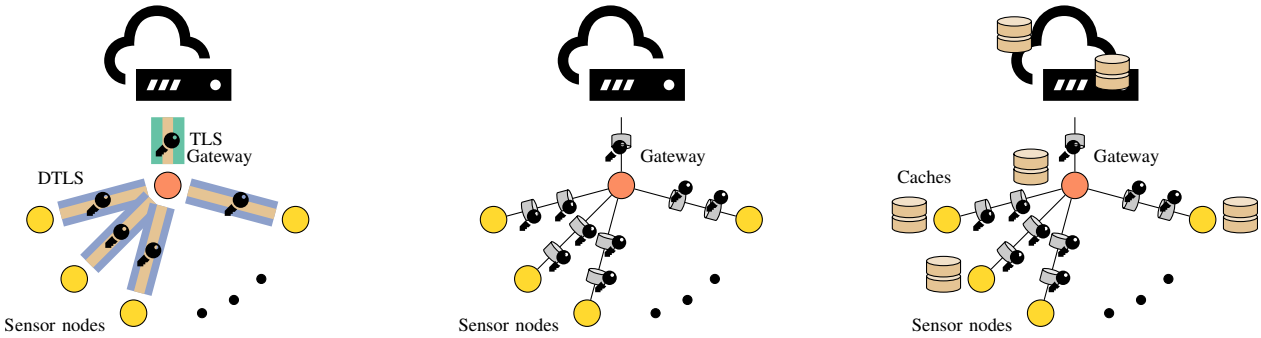
Datagram Transport Layer Security (DTLS) [5] closely follows TLS [15] in terms of protocol behavior and security guarantees. Unlike its stream-oriented relative, DTLS adds facilities to operate in unreliable datagram environments. It contributes a modified record layer that tolerates packet loss and message reordering. To break up inter-record dependencies, DTLS bans stream ciphers and uses explicit sequence numbers in datagrams. A cryptographic context thus spans exactly one record. UDP is the prevailing transport in IoT deployments. Compared to TCP, it exhibits no substantial protocol overhead and allows for implementations with low memory footprints. Utilizing DTLS to secure existing application protocols such as CoAP and MQTT-SN hence appears to be the best logical choice—at least at first glance.

Concerns arose in recent studies that question the applicability of DTLS in large-scale IoT systems. First, certain cryptographic challenges during handshake processes are infeasible. While processing time for cryptographic operations diminish with hardware crypto modules, message sizes are inflated. Asymmetric key ciphers require handshake overhead and large payload sizes, which immensely boost handshake completion times to the order of seconds and minutes in multi-hop deployments due to packet fragmentation [16].

The stateful session characteristic further comes at the cost of multicast capability, since security contexts are identified by the classic 5-tuple between two endpoints. Particularly in scenarios that involve device mobility and multi-homing, a generally accepted effort applies connection identifiers to security channels—independent of the 5-tuple [17]. Fig. 1 (a) illustrates a realistic deployment setup for CoAP over DTLS: End-to-end security commonly terminates at the gateway to allow for protocol conversions, *e.g.,* to HTTPS over TCP.

C. Content Object Security in the IoT

OSCORE [8] is a protocol extension to CoAP and addresses the terminating security issue at gateways. Instead of securing sessions between endpoints, OSCORE protects entire CoAP messages and provides integrity, authenticity, and confidentiality on an object level. The original CoAP message is thereby encapsulated as an authenticated and encrypted CBOR Object Signing and Encryption (COSE) [18] object by an outer



(a) CoAP over DTLS (b) OSCORE (c) NDN

Fig. 1. Typical deployment setups for CoAP over DTLS, OSCORE, and NDN in the IoT. Validity of session keys terminates at gateways for transport layer security due to transport conversions, *e.g.,* UDP to TCP. Content object security is unaffected by gateway operations and reaches end-to-end.

CoAP option. In addition to cryptographic efforts, the protocol further includes countermeasures to prevent response delay and mismatch attacks. A strong message binding between requests and corresponding responses is constructed with the use of identical identifiers in their authenticated components, which persist over retransmissions. Replay windows allow for rearranged messages to be processed independently. Applications built on it use CoAP mechanisms like If-Match or the Echo [19] option to protect against any ill-effects of rearranged messages.

OSCORE utilizes the request-response semantics of its underlying CoAP layer and an elaborate nonce construction to obtain compact response messages. When combined with CoAP observation (continuous responses to a single request), OSCORE protects the sequence of notifications using its own sequence numbers. When combined with CoAP block-wise transfer, it fragments large resources into pieces small enough for the end points to process in a single cryptographic operation without hindering further block-wise processing by proxies. Unlike DTLS, OSCORE does not come with a built-in key exchange protocol, and relies on pre-shared keys. A lightweight authenticated key exchange (LAKE [20]) is under development as a companion protocol.

A major improvement over the conventional transport layer security concept is the ability to secure multicast messages. CoAP supports a one-to-many group communication [21] when used with UDP. While DTLS fails to perform in multicast environments, the object security characteristic of OSCORE allows for protected requests and responses in these deployments [22].

Fig. 1 (b) illustrates the envisioned deployment option. Messages are cryptographically secured and despite protocol conversions on gateways, their properties stay intact while traversing up to cloud services.

D. Content Security in the Information-Centric IoT

Information-centric Networking [23], [24]—a clean-slate approach of the Future Internet initiatives—abandons the host-centric Internet paradigm in the favor of autonomous content, which allow for an unhindered replication of authenticated data objects. A decade of research has created a variety of ICN flavors that have three principles in common [25]: *Decoupling of named content from hosts*, *universal caching*, and *content object security*.

Named Data Networking (NDN) [26] enjoys significant popularity and has been identified early as a candidate for low-end IoT edge networking [27]. An adaptation layer to the low power lossy wireless exists with ICNLoWPAN [28]. In contrast to the end-to-end stateless packet processing on the Internet, NDN utilizes a stateful, hop-by-hop forwarding fabric that decouples content objects from their locations and enables seamless on-path caching. NDN follows a simple request-response scheme on the network layer using the two message types *Interest* and *Data*, each treated individually by the forwarding state machine. Fig. 1 (c) illustrates the additional content caches that support content replication and local recovery from losses.

NDN supports integrity and authenticity as protocol features by appending cryptographic signatures to data packets. While originally the intrinsic security only applied to data packets, the upcoming NDN protocol version allows for a signature inclusion in Interests. Confidentiality is not supported on the protocol level, but left to the applications, which may encrypt content.

III. Composable Network Stacks for Object Security in Challenged IoT Deployments

The decision for a software platform that can cope with constrained IoT is crucial. As we aim for maintainability and sustainability, we extend existing code bases instead of designing and implementing from scratch. As such, we utilize the open source IoT operating system RIOT [29] and leverage its existing network stack, which follows a cleanly layered architecture [30]. In course of our evaluations, we contribute and upstream improvements to the RIOT integrations of DTLS, OSCORE, and NDN.

A. The RIOT Networking Subsystem

The RIOT networking subsystem displays two interfaces to its externals (see Fig. 2): The application programming interface `sock` and the device driver API `netdev`. Internal

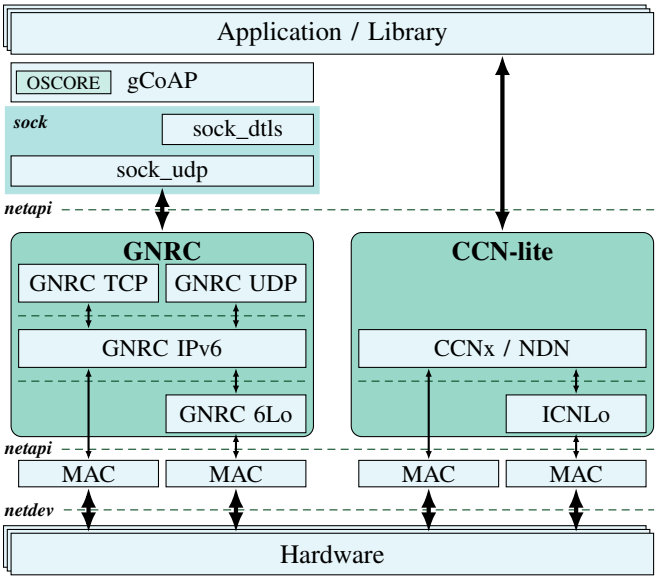


Fig. 2. The RIOT networking subsystem.

to stacks, protocol layers interact via the unified interface `netapi`, thereby defining a recursive layering of a single concept that enables interaction between various building blocks: 6lo with media access control (MAC), IP with routing protocols, transport layers with application protocols, *etc.* This grants enhanced flexibility for network devices that come with stacks integrated at different levels.

B. CoAP Over DTLS

gCoAP is the feature-rich native CoAP implementation of RIOT. It implements the server-side and client-side, it supports the most common methods GET, POST, PUT, DELETE, it handles confirmable messages, and it allows for observing resources. gCoAP further provides the CoAP proxy functionality to redirect client requests and maintains a response cache to reduce round-trip times. The cache is using a least recently used eviction strategy to prioritize responses of the most recent requests. As depicted in Fig. 2, gCoAP uses the sock API. On the north-bound it attaches to `sock_udp` and `sock_dtls`, which makes it completely network stack agnostic. In the default configuration, the native 6LoWPAN [31] network stack of RIOT—Generic (GNRC)—provides the south-bound implementation of sock_udp. The DTLS counterpart is provided by the external package tinyDTLS. It follows a threadless design and depends on events, which are handled by the sock layer within the gCoAP thread. This DTLS setup supports two ciphersuits: (*i*) pre-shared authentication and key exchange using the Advanced Encryption Standard (AES) in Counter with CBC-MAC mode (CCM) [32] with a 128-bit key length, and (*ii*) an Elliptic Curve Cryptography (ECC) based AES-CCM with an Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) key exchange.

C. CoAP With OSCORE

We provide `libOSCORE`[1] as an implementation of the OSCORE [8] model that integrates into RIOT. Unlike other approaches, *e.g.,* a yet to be mainstreamed Contiki implementation[2] and c_OSCORE[3] on top of Zephyr, libOSCORE focuses on portability across different CoAP libraries and provides replay protection.

Distinct features of libOSCORE are its handling of the request-response correlation data and its zero-copy approach. In the former, initialization vectors (`Partial IVs`) [18], which are used to create unique nonces for the underlying authenticated encryption algorithm, are consistently passed by reference. They carry a flag indicating first use and get invalidated by consumers. This allows leveraging OSCORE optimizations for safe representational state transfer (REST) operations. In memory management, libOSCORE expects its user to provide suitable memory locations and provides struct definitions to make that portable. This saves execution time, main memory (RAM) and storage (ROM) at the cost of some implementation complexity on the user side. It allows processing of messages from the receive-buffer in a single reading pass after in-place decryption and without dynamic memory allocation.

In CoAP libraries that build and read their messages in buffers (*cf,* gCoAP), integration of libOSCORE happens in two stages: (*i*) basic integration, and (*ii*) full integration.

The basic integration describes the most elementary way of interacting with libOSCORE. It only requires a mapping of certain CoAP operations and cryptographic primitives. Applications that use this mode directly access OSCORE objects and steer every step of the encryption and decryption process for each packet. Generally, usage of this mode is tedious and error-prone and therefore discouraged for user applications. On the other hand, a basic integration allows for a full control of OSCORE internals, which can be leveraged to perform protocol optimizations by libraries or protocol extensions.

The full integration requires a functional basic integration as prerequisite. At that stage, libOSCORE messages are used as backends for the native CoAP library. Application code is identical for the unprotected and OSCORE-protected case, and thus tightly coupled to the native CoAP implementation. The CoAP library dispatches operations on messages through libOSCORE atop of, or directly through the transport protocol, depending on the application's configuration (*e.g.,* the choice of a security context for a message) or presence of the OSCORE option. This mode of operation is the recommended way of building user applications, as APIs hide security operations and prevent security breaches due to a misuse of OSCORE internals. The stack in Fig. 2 shows the full integration state where applications interact only with gCoAP.

An intermediate integration is available in libOSCORE for cases when full integration is unfeasible with a particular library or simply incomplete. At that level, an additional library provides code to orchestrate and simplify cryptographic

[1] https://gitlab.com/oscore/liboscore
[2] https://github.com/Gunzter/contiki-ng/tree/oscore_12
[3] https://github.com/Fraunhofer-AISEC/c_OSCORE

procedures. This mode is most suited for narrow-purpose helper libraries up to full-fledged REST frameworks, which generally provide their own APIs towards user applications.

We further implement experimental protocol features to explore the design space as discussed in our comparative evaluation (Section VI). In detail, we allow proxy nodes to cache OSCORE responses so they can be served for request retransmissions from the same client.

D. Named Data Networking

CCN-lite [33] is a lightweight NDN forwarder, which supports all primary features: in-network caches, hop-wise retransmissions, request aggregation along paths, and multi-source, multi-destination forwarding. It runs on a variety of hardware platforms—ranging from commodity hardware to embedded devices. While the core forwarder is self-contained and platform independent, adaptors provide access to the system communication API. CCN-lite is integrated into RIOT as an external package. It contributes a RIOT adaptor, which hosts its own thread and translates between CCN-lite messages and netapi packets.

IV. Theoretical Evaluation of Protocol Security

Performance measures such as security properties largely differ for each protocol configuration. In the following, our performance assessment considers protocol design choices and thus provides insights that are independent of specific deployments. We focus on four protocol compositions: (*i*) CoAP (Protected) with encrypted and authenticated response payload as baseline implementation. (*ii*) CoAP over a secured DTLS 1.2 session. (*iii*) OSCORE to provide object security for request and response messages. (*iv*) NDN (Protected) using signed Data messages and encrypted as well as authenticated content.

A. Cryptographic Algorithms

Cryptographic primitives quickly touch critical resource limits in low-power, wirelessly connected regimes, as computational complexity and memory consumption strongly vary with algorithms and implementations. The limited amount of main memory and slow CPU clock speeds in embedded IoT devices can push completion times of resource-exhaustive operations by orders of magnitude beyond a reasonable performance on commodity hardware—notably for asymmetric cryptography. While long computations reduce sleep cycles and drain batteries much faster, they also affect the responsiveness of the overall—commonly single-threaded—system. Hardware-assisted cryptography can largely improve on resource consumption, but necessary crypto μchips are not always integrated into hardware platforms due to economical reasons. For a detailed analysis of crypto-primitives on low-end IoT devices we refer to [34].

In our following protocol selection, we specifically focused on low-complexity modes in crypto using pre-shared keys to not burden the protocol assessment with disproportionally long intra-stack delays. Since CoAP appoints an authenticated encryption using AES with a block size of 128 bits in CCM mode and an 8-byte authentication tag as mandatory-to-implement [4, Section 9.1.3.1], we configured this choice for all protocol deployments.

B. Security Properties

CoAP (Protected) exhibits the weakest security properties in our comparison: While it uses an authenticated encryption for the *payload*, it does not provide any security measures for the actual CoAP messages to protect CoAP signaling. Protocol headers are prone to tampering and messages are susceptible to interception as well as packet delay attacks. These shortcomings make the binding of requests to correct responses fragile. The inability to map responses to particular requests is especially dangerous in cases when resources publish mutable content [19], [35]. Consequently, even in the case when the payload is secured, delayed and replayed messages can affect the state machine on the client and server. Since the message headers are not protected against confidentiality attacks, this configuration easily leads to privacy concerns. Plaintext requests will contain resource URIs, which typically help to identify sensitive application information and therefore potentially leak private data. Responses may not include resource URIs, but included tokens unambiguously identify potentially intercepted requests and thus their resource URIs.

CoAP over DTLS is the common method for securing message transmissions in an IoT network. DTLS provides integrity, authenticity, and confidentiality for UDP datagrams within sessions based on pre-established private keys. It operates below the application layer and inherently takes CoAP requests as well as responses into consideration. A drawback from this layering, however, is that the DTLS record layer is not aware of CoAP semantics. This introduces a twofold problem: First, this configuration suffers from the same request-response binding issues when messages are delayed and replayed [35] while recent mitigations [19] are not deployed yet. Second, end-to-end security terminates at gateways in usual IoT setups when protocol conversions from CoAP to HTTP take place. Minimal DTLS implementations commonly provide the lightweight DTLS cipher suite TLS_PSK_WITH_AES_128_CCM_8 [36], which does not provide perfect forward secrecy. Adaptations [37], allow for the combination of existing cipher suites with the Ephemeral Elliptic Curve Diffie-Hellman key agreement protocol.

OSCORE achieves a secured communication by protecting request and response messages on CoAP level. This is in contrast to CoAP over DTLS that establishes secure channels between endpoints. OSCORE provides integrity, authenticity, and confidentiality by nesting the actual CoAP message as an authenticated and encrypted payload, interleaving information relevant to routing and retransmission in the unprotected outer parts. This layer hides sensitive information, such as the resource path and the CoAP method of the original message. Furthermore, the security of inner messages stays intact across protocol translations on gateways (*e.g.,* from CoAP to HTTP/S). OSCORE provides a strong request-response binding with mechanisms like sequence counters and sliding

TABLE I
SUMMARY OF SECURITY PROPERTIES FOR EACH PROTOCOL CONFIGURATION. (✓) INDICATES OPTIONAL SPECIFICATIONS, WHICH ARE UNAVAILABLE IN THE USED IMPLEMENTATIONS.

	CoAP			NDN
	Protected	DTLS	OSCORE	Protected
Request Message				
Integrity	✘	✓	✓	(✓)
Authenticity	✘	✓	✓	(✓)
Confidentiality	✘	✓	✓	✘
Response Message				
Integrity	✘	✓	✓	✓
Authenticity	✘	✓	✓	✓
Confidentiality	✘	✓	✓	✘
Attack Resiliency				
Replay Insensitivity	✘	(✓)	✓	✓
Perfect Forward Secrecy	✘	(✓)	✘	✘

windows, which renders many attacks ineffective. The original specification is missing a key exchange protocol and thus does not provide perfect forward secrecy. Adaptations [38] allow for an Ephemeral Diffie-Hellman over COSE.

NDN authenticates response messages between arbitrary endpoints without the need for session state. While application payload can be encrypted, NDN does not provide confidentiality for message headers. Moreover, NDN reduces security features to response messages only[4]. Names are an integral part of the NDN forwarding fabric and may contain sensitive application information. Thus, privacy concerns arise from plaintext names in NDN messages. An encryption or obfuscation of names inevitably affects the routing system and adds an exhaustive overhead. Unlike the CoAP variants, NDN follows the principle of immutable content: A specific name invariably points to the same content object. This property reduces the attack surface and desensitizes applications to delayed and replayed messages.

We summarize the observed advantages and drawbacks of the discussed protocol schemes in TABLE I, with a strong focus on the actual protocol behavior rather than on application payload security.

C. Security Message Overhead

In all protocol configurations, security extensions add message overhead and consequently affect transmission times. Notably for IEEE 802.15.4, inflated messages easily increase media access times by a few milliseconds, whereas computational overhead in common IoT network stacks is in the range of microseconds [30]. We now quantify the overhead in terms of packet size which is introduced by the different security extensions. In Section IV-D, we will put this into perspective with respect to the common CoAP and NDN packets.

CoAP (Protected) and NDN (Protected) do not add any message overhead to requests. All configurations other than CoAP (Protected) add a structural overhead related to security. DTLS includes 11 bytes for the DTLS 1.2 record layer in all datagrams, excluding the epoch field. The NDN packet format uses flexible Type-Length-Value (TLV) fields to encode message headers. Security related TLVs similarly account for 11 bytes overhead. OSCORE exploits implicit information that results from a strong request-response binding and further utilizes a concise binary object representation (CBOR). This nets to a structural overhead of four and three bytes.

[4]Specification v0.3 is in progress and adds security features to Interests

TABLE II
MESSAGE OVERHEAD OF SECURITY MEASURES IN BYTES. OVERHEAD DOES NOT APPLY TO COAP AND NDN REQUESTS.

	CoAP						NDN	
	Protected		DTLS		OSCORE		Protected	
	Req	Resp	Req	Resp	Req	Resp	Req	Resp
Structure	–	0	11	11	4	3	–	11
Context ID	–	2	2	2	1	0	–	1
Nonce	–	2	8	8	1	0	–	0
MAC	–	8	8	8	8	8	–	40

The security context identifier consists of two bytes for CoAP (Protected) and CoAP over DTLS. In the former scenario, contexts are identified by the 2-byte key identifier within our payload, while the latter scenario uses a 2-byte epoch field in the record layer to denote a secured session. OSCORE and NDN are able to reduce the length of small context identifiers. OSCORE omits the security context in response messages and requesting devices must deduce it from the request state.

For AES in CCM mode, the same nonce is required for encryption and decryption. The nonce is of variable length and usually ranges between 7–13 bytes [32]. We design our experiment to use partially implicit nonces [39]. Two bytes of the nonce are encoded into messages, while the remaining bytes are deduced implicitly, *e.g.,* from the hash of a resource URI. This allows 2^{16} messages per resource until a refresh of established security contexts is advisable. OSCORE repeatedly encodes smaller values in a single byte and CoAP (Protected) uses a 2-byte representation. In responses, OSCORE uses the same nonce to protect objects and thus omits the nonce. CoAP over DTLS uses eight bytes as a result of concatenating the epoch and sequence number fields. The remaining four bytes of the DTLS nonce are implicit and generated as part of the handshake process [36]. NDN benefits from the immutable content property: Since names always map to the respective content, its hash is used as nonce.

We use a message authentication code of eight bytes as defined by the TLS AES-CCM cipher suites [36]. NDN appends another 32-byte hash-based message authentication code (HMAC) signature that envelops the complete response packet. TABLE II summarizes the message overhead for the discussed protocols.

D. Security Overhead in Comparison to Basic CoAP and NDN Messages

We now dissect each message of the protocols under comparison in detail and relate the basic CoAP and NDN packet sizes to the security extension (see Fig. 3). Our analysis distinguishes between requests and responses and includes all handshake messages for DTLS. We assume that a response payload includes a 2-byte temperature value.

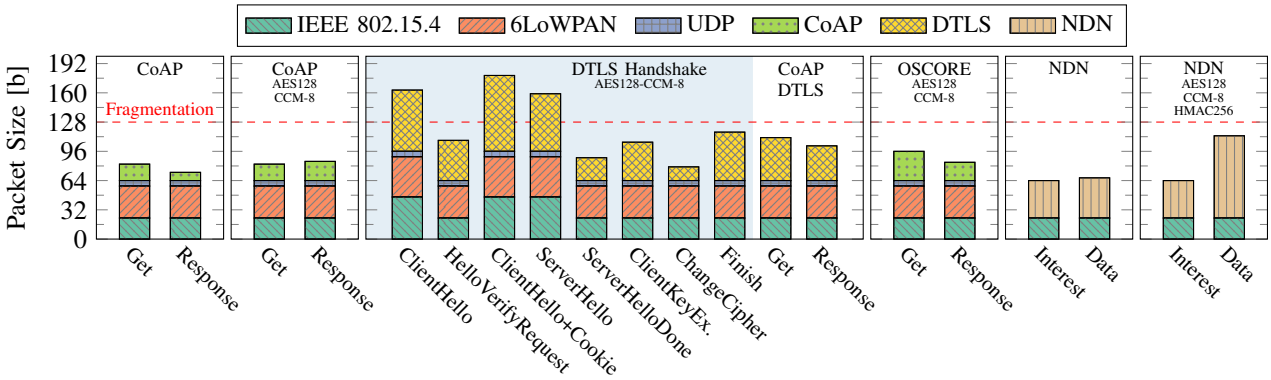


Fig. 3. Packet structures of control- and data-plane packets for each protocol configuration.

IEEE 802.15.4 admits a maximum physical layer packet size of 127 octets. Assuming a typical configuration of 8-byte source and destination hardware addresses, considering a given 2-byte frame control field, 1-byte sequence number, 2-byte personal area network (PAN) identifier, and a 2-byte frame check sequence, the total MAC header overhead adds up to 23 bytes for each protocol. This leaves 104 bytes for upper layer headers and user data.

In CoAP setups, the 6LoWPAN header occupies 35 bytes because it accommodates three 6LoWPAN dispatch bytes and two IPv6 addresses. Moreover, each packet counts six bytes for the compressed UDP header.

Special consideration is required for *ClientHello* and *ServerHello* packets in a DTLS handshake. In contrast to previous calculation, they surpass the maximum physical packet size and trigger a hop-wise 6LoWPAN fragmentation. While the MAC header overhead is therefore doubled, the 6LoWPAN overhead increases by only nine bytes for the inclusion of fragmentation dispatches in both fragments.

In contrast to unprotected CoAP responses, CoAP (Protected) messages inflate by 12 bytes to include the context id, nonce, and message authentication code of AES-CCM (see Section IV-C). CoAP over DTLS emits 29 and 27 more bytes for requests and responses, respectively, due to the DTLS record layer. OSCORE messages display similar but extenuating effects: requests increase by 14 and responses by only 11 bytes. The primary explanation for this surprisingly smaller increase is a reduced header overhead of OSCORE compared to the DTLS 1.2 record layer. Nonces are further omitted from responses to decrease their header overhead.

In contrast to CoAP, where responses display smaller packet sizes than requests, NDN data packets exhibit larger sizes than Interests. This is a result of names being fully included in returning data packets. NDN data packets increase by an 8-byte AES-CCM MAC and an 32-byte HMAC signature, compared to unsecured NDN packets with an overall packet size of 64 bytes. Since Interest messages do not contain any security measures, their packet sizes remain unaffected.

V. Redundancy, Resilience, and Recovery

We now discuss prospective protocol features that are under early discussion in the IETF, or could be utile in the near future. The focus of this section is on protocol resilience against unanticipated service interrupts, fast and lightweight recovery, as well as on potential benefits from multi-destination content replication and caching.

A. Resilience to Loss of Security State

Nodes in the constrained IoT have frequent reasons to power down or even reboot. Protocol state which frequently updates such as session keys, nonces, or sequence numbers usually remains in main memory to reduce the number of energy-intensive I/O operations on non-volatile memory modules, and is therefore endangered to be lost. In distributed system settings such loss of state causes difficulties with synchronizing protocol behavior.

Security protocols usually manage state with varying demands on state persistence. Keying material is deployed statically at device bootstrap, dynamically exchanged, or derived from a key establishment scheme. These keys are considered immutable for the lifetime of a single security context and are persisted into non-volatile memory. Reorder and replay protection mechanisms require per-message state in the form of sequence counters, which update continuously with byte flow in a specific context. While these counters are essential for an efficient operation and remain effective across spurious system reboots, it strains energy and memory lifetime to preserve all individual state changes.

In this evaluation, we review the protocol behavior on unexpected security state loss for our selected protocol ensemble.

CoAP over DTLS deployments perform session establishment between endpoints to negotiate keying material and to agree on suitable cryptographic ciphers. Derived keys are valid for the lifetime of the session and are renegotiated on rare occasions, in which case the session epoch number gets incremented. The epoch number identifies delayed packets that did not properly transition to the new ciphers yet. Session keys and epoch fields easily persist on non-volatile storage to make them available throughout system reboots.

The sequence number in the DTLS record layer allows for detecting reordered and replayed messages below the CoAP layer. For this, each session endpoint individually advances a sliding window following its received sequence numbers.

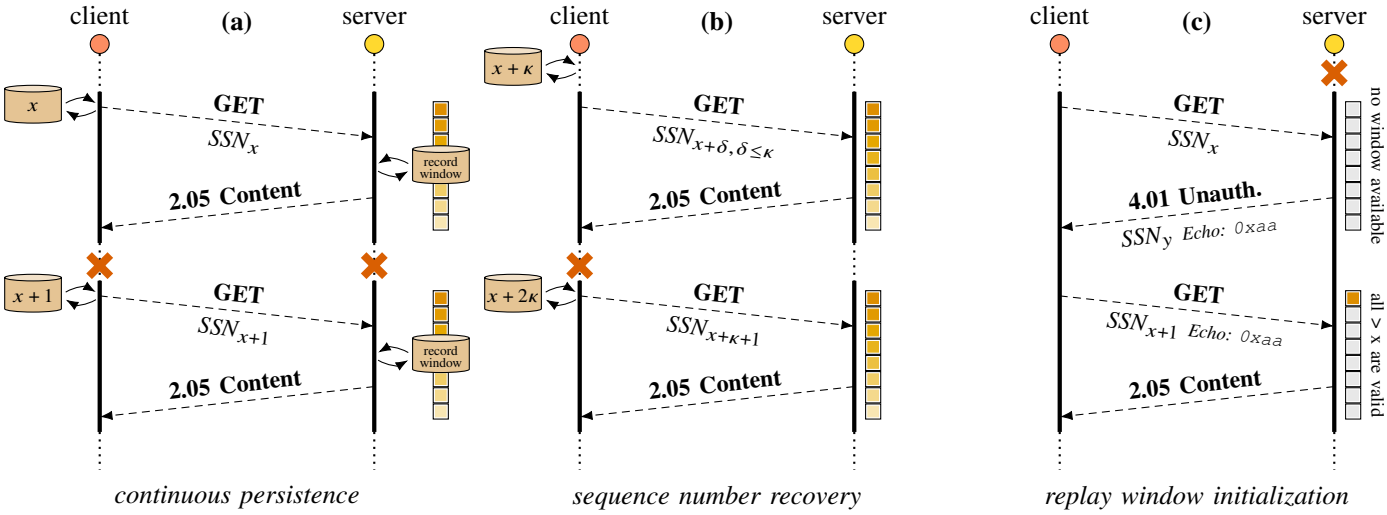


Fig. 4. Protocol behavior due to loss of mutable OSCORE state after unanticipated client and server shutdowns with and without optimizations [8].

Since sequence numbers and the window bitmap are frequently updated, persisting them on a storage module is infeasible. A state loss due to unanticipated fail on either side thus requires a full handshake of six to ten message flights to re-establish the session. An optimization is the ticket-based session resumption feature [40], which reduces the handshake down to three flights. Tickets are created on the initial handshake and can be stored in non-volatile memory.

CoAP with OSCORE shares some DTLS basic properties, but has different recovery options. Keying material of a pre-established security context is infrequently updated and therefore demands only a minimal amount of persistent storage I/O. Here as well, message sender sequence numbers (SSN) and sliding windows are employed to provide a replay protection mechanism. Only CoAP server nodes advance a sliding replay window for request messages.

Like in DTLS, an unexpected shutdown renders even persisted key material unusable when no recovery options are available. This hits OSCORE, however, harder than DTLS, for while DTLS can restart with a full handshake, OSCORE needs external mechanisms to arrive at a fresh security context. A simple but inefficient mechanism to avoid that is to persist the volatile state on each message, which adds to round trip times and memory wear. As shown in Fig. 4 a, every single sequence number and each advancement of the replay window is persisted to maintain a consistent state beyond an unanticipated system reset.

The appendix B.1 of the OSCORE specification [8] introduces two mechanisms to reduce the persistence operations to a few writes per reboot and takes the writes out of the request-response latency path, while introducing little or no randomness requirements. We distinguish between loss of volatile state on the client and the server side. On the client side (depicted in Fig. 4 b), sequence numbers are leased in chunks of K numbers, and the last value of the leases is persisted. On system boot, or when the pool of leased numbers is near exhaustion, another chunk is leased and the persisted number increased. The interval provides a trade-off between costly write operations and the amount of sequence numbers that are lost after this procedure until a rekeying becomes necessary, and can be configured or adjusted automatically at runtime.

Fig. 4 c depicts the behavior when a CoAP server loses the sliding replay window state. As the server does not persist a replay window, it cannot determine that any request is not a replay, and rejects it. Along with the rejection, it sends an encrypted fresh Echo value [19]. (While using a random nonce is an option, our implementation draws from the previously established sequence number pool). A client with the correct key material can repeat this request and mirrors the echo tag back to the server. On success, the server then accepts this request and initializes its replay window starting at a sequence number it knows to be fresh.

NDN IoT deployments typically use pre-shared keys or a complementary public key infrastructure to protect Interest and data messages. For deployments with asymmetric cryptography, content producers use key material to provide message authenticity and integrity by digitally signing content objects, while consumers require the corresponding public key counterparts to perform origin authentication on incoming data messages. A static configuration of necessary keys may work for a small set of devices in a network, but becomes unmanageable if content origins dynamically leave and join trust relationships. A trust schema [41] can be leveraged to configure automatic decisions for content producers and consumers to dynamically create, choose, and receive the correct key material based on the content names. In addition, temporary session keys can be derived from key exchange protocol extensions, such as OnboardICNg [42], or LASeR [43]. OnboardICNg builds on the authenticated key exchange protocol (AKEP2) [44] and LASeR is inspired by the extensible authentication protocol with session-key derivation using a pre-shared key (EAP-PSK) [45]. Both generate temporary security contexts that are valid for the lifetime of a session and the resulting keys can be used to maintain forward secrecy across consecutive sessions.

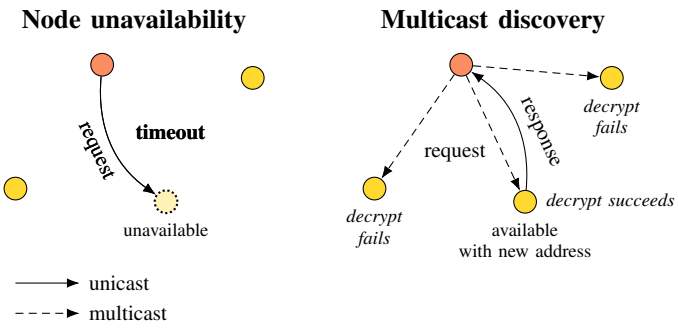


Fig. 5. Node discovery with OSCORE using a scoped multicast request.

Similar to DTLS and OSCORE, persisting pre-shared secrets or session keys on device bootstrapping or session establishing allows for continuing a secured communication after unexpected system reboots. NDN does not include sequence numbers in messages, nor does it maintain frequently changing security state on devices, since the hop-wise content replication and immutability of named objects already reduce the attack vector for reordering and replay attacks (see Section IV-B).

B. Protected Multicast Device Discovery

Multicast on the network layer is a scalable mechanism for contacting large groups of possibly unknown devices. In fixed IPv6 networks it is an essential protocol feature for establishing communication relations (*e.g.,* using neighbor discovery [46]). In low-power, wireless regimes devices running 6LoWPAN face additional challenges.

LoWPAN nodes are often subject to disruptive events, *e.g.,* lossy links and device mobility, which make a perpetual connectivity between endpoints virtually impossible. Not uncommonly, reappearing nodes may configure new endpoint addresses due to expiring DHCP leases or privacy extensions [47] for stateless address auto-configuration schemes that limit the validity of addresses to a few hours or days depending on the scenario setup. A CoAP deployment thus usually requires a complementary infrastructure like the CoRE resource directory [48], which allows to discover available resource endpoints. Nevertheless, in high mobility scenarios, propagations of frequently updating topologies to a central registry can introduce an infeasible communication overhead. A scoped multicast CoAP request is a lightweight alternative for (re-)discovering CoAP endpoints in close vicinity.

CoAP over DTLS prohibits the use of multicast addresses, because security sessions are bilaterally established between endpoint pairs. A multicast CoAP request is thus not protected by DTLS, which renders this lightweight alternative impractical for deployments with reasonable security demands.

OSCORE detaches the security context from endpoint-specific information. A request can be sent as a CoAP multicast message with OSCORE protection (in regular mode, without the Group OSCORE [22] extension) with its encryption and most privacy properties intact. Although such messages are potentially received by a group of nodes (see Fig. 5), only a single server that holds the corresponding security context can decrypt and respond. The response possibly returns on a unicast link to the client node. On a successful transaction, subsequent requests can use this newly discovered unicast address.

To stay reachable after an address change, an OSCORE server could inform its peers of an identifier that is usable for longer than its network addresses. This may happen by explicit announcement, or as additional information when the OSCORE context is set up. As such an identifier can be used to track a device across address changes and possibly across different OSCORE keys, its privacy implications need to be considered before employing such a scheme.

Carefully designing group memberships in wireless deployments with multicast support is pivotal to reduce energy expenditure and excessive media utilization. Following the IPv6 address architecture [49] (see Fig. 6), a 16-byte multicast address decomposes into a 2-byte address classifier and a 14-byte group identifier. The `ff02` classifier identifies multicast addresses that are link-local.

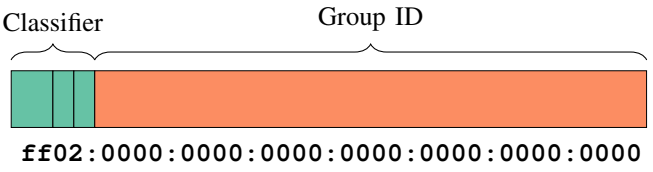


Fig. 6. IPv6 link-local multicast address format [49] using a 2-byte classifier and a 14-byte group id.

We propose a scheme that maps OSCORE state into the least significant bytes of an IPv6 multicast address and corresponding nodes configure these addresses on their network interfaces. This optimization utilizes scoped multicast messages that are already filtered on the network layer and only recipients with likely-to-match OSCORE contexts will receive and process them.

We propose link-local identifiers that map to one or more OSCORE contexts. These identifiers could be derived from the OSCORE key ids, or could be distributed via an external mechanism, *e.g.,* during a session establishment process, or could utilize global node identifiers, if available.

Choosing a suitable length within the multicast address is a trade-off between two extremes: With too short identifiers, collisions may force many nodes to receive and try to decrypt messages. Long identifiers possibly consume too much of the available address space. To find an advisable id length, we follow the design decision for the `solicited-node` multicast address [49], which is used in the neighbor discovery protocol of IPv6 [46]. The address has the prefix ff02::1:ff00:0/104 and allows for a 3-byte (24-bit) variability in the low-order.

We use the same amount of variability by using three bytes long identifiers, but configure another multicast prefix to not interfere with the neighbor discovery. An example prefix could be the currently unallocated ff02::3:ff00:0/104 address (see Fig. 7), but any serious deployment would need to check with an up-to-date IANA registration to sustain interoperability with coexisting protocols.

`ff02:0000:0000:0000:0000:0003:FFXX:XXXX`

Fig. 7. Link-local multicast addresses for OSCORE context discovery.

The base of 2^{24} addresses makes it unlikely that a device needs to process an OSCORE request intended for another device. More precisely, the probability of collisions is known from the birthday paradox. Equation 1 describes this probability for a year of d 'days' and n 'people'. In our case, $d = 2^{24}$ is the given address space and n is the number of multicast addresses in use. Evaluating equation 1 yields a collision probability $< 1\%$ for 575 multicast addresses, which must be considered a large number of security groups on a single link.

Still, in case of a collision, the full key id that is part of any request serves as a further distinction. Eventually, the authenticated encryption of OSCORE filters out remaining collisions.

$$p(n;d) = 1 - \prod_{k=1}^{n-1} \frac{d-k}{d} \tag{1}$$

NDN allows for secured multicast messages similarly to OSCORE, provided a proper name to MAC address mapping [50] is in place. In contrast to OSCORE, a multicast request discovers content and not devices. Since content is matched by exact and immutable names, no additional mechanisms to protecting against re-ordering or replay are needed, even though a single request can lead to returning responses from several content producers or in-network caches. NDN deduplicates multiple data messages of the same request on the forwarding plane by serving only the first incoming data and thereafter discarding all replicated messages.

C. Protected Multi-Source and Multi-Destination Messages

Scalable group communication is essential for supporting key IoT use cases: Multi-source readings of uniform sensors are most efficiently implemented as a convergecast following a multi-destination read request. Groups of actuators are often coordinated via multi-destination control messages. Over the air (OTA) firmware updates are vital for device maintenance and hard to implement without efficient multicast flows.

Multicast communication is an inherent property of the information-centric NDN architecture. Endpoint information is absent from the addressing scheme, which allows for direct multi-source and multi-destination access by applications mapping to a multicast scheme. In NDN, a multi-source communication is enabled as follows: nodes on intersecting request paths aggregate requests and returning responses fan out on the interfaces to the corresponding requesters. Multi-destination requests, on the other hand, are supported by on-path caches and multiple fan-outs in the forwarding information base for the same name prefix.

Responses that traverse a request path consume the request state. Thus, responses are deduplicated on path intersections when arriving from multiple destinations, and only the first response is forwarded. Since requests and responses are protected on an object level, the security measures of NDN are equally effective in unicast and multicast communication.

The classic CoAP request-response model enables multi-destination requests by leveraging IP multicast groups with response messages returning via unicast. In contrast to NDN, a reliable multicast communication using acknowledgments and retransmissions is unsupported, since the number of endpoints within a multicast group is unknown in IP multicast. CoAP proxies cache requests from multiple clients for the same resource, and aggregate observations by fanning out single responses to multiple clients using unicast messages. Sending a request to a multicast address does not preclude caching, but practical deployment via a proxy depends on experimental extensions [51] and the client awareness of the origin being a multicast location. Setups where the client is unaware of that and a reverse proxy requests from multicast to return the first response are conceivable, but we are not aware of any existing implementation or experimentation.

With OSCORE, multicast requests are covered by Group OSCORE [22], necessitating a client awareness of the multicast context. Work on distributing responses to multiple clients is highly experimental [52], [53]. The approaches are centered around the clients obtaining or building identical requests to which the responses can be bound. Such request-response mappings are necessary because knowing the request is essential for understanding the response. They promise to transfer both the caching and aggregation abilities of CoAP over to OSCORE, and even to extend the base CoAP mechanism to allow multicast distribution of responses. We further evaluate the impact of group communication with caching in our experiment report in Section VI-E.

VI. Evaluation in the Testbed

In this section, we compare the different protocol configurations based on real implementations deployed in a testbed.

A. Experiment Setup

Scenarios & Parameters. We want to quantify the performances of a protected CoAP and NDN communication in a typical IoT data collection scenario with multiple sensor nodes. For this, subsequent requests periodically traverse a gateway into an IoT stub network. Each sensor device is requested 1000 times at an (randomly jittered) interval of $2 \pm 0.5s$ and returns a 2-byte temperature reading. To allow for comparison of pull-based NDN with CoAP, we limit CoAP methods to confirmable GET.

We align our experiments with respect to retransmission and timeout configurations. All protocols employ the same retransmission strategy: On failures, nodes wait two seconds before retransmitting the original request. In NDN, retransmissions are performed hop-by-hop, while CoAP performs them end-to-end. At most four retransmissions will occur for each data, which is the default configuration for CoAP [4, Section 4.8]. Remaining protocols do not have standard defaults and we align with CoAP.

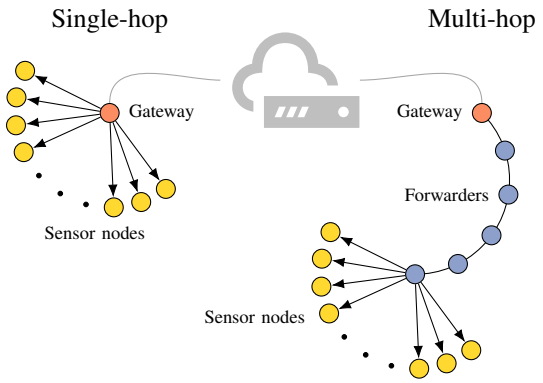


Fig. 8. Topologies for single-hop and multi-hop experiments.

We do not consider congestion from external cross-traffic in this work. However, each individual transmission experiences self-induced background traffic from on-going requests and retransmissions. The jittered request interval further mixes the event space and allows a greater exploration of the state space. On average, this cross-traffic is constant per experimental run.

Software & Hardware Platform. All devices run RIOT version 2019.10. NDN deployments are based on CCN-lite, and CoAP experiments use the default GNRC network stack of RIOT including libOSCORE and tinyDTLS (*cf,* Section III).

We conduct all experiments on the FIT IoT-LAB [54] testbed. The hardware platform consists of class 2 devices [55] featuring an ARM Cortex-M3 MCU with 64 kB of RAM and 512 kB of ROM. Each device is equipped with an Atmel AT86RF231 [56] transceiver to operate on the IEEE 802.15.4 radio. The testbed provides access to several sites with varying properties. We perform our experiments on the *grenoble* site in a single-hop and multi-hop configuration. Our single-hop setup consists of one gateway node and ten sensor nodes in broadcast range as illustrated in Fig. 8. In the multi-hop configuration, we use one gateway node, ten sensor nodes, and five forwarder nodes. Forwarding states are statically configured on each node to form the topology depicted in Fig. 8.

Protocol Configurations & Start-Up Conditions. In all setups, we use AES in CCM mode with a 128-bit key and limit the resulting message authentication code to eight bytes as described in [36]. Each configuration also contains a 1-byte key identifier where applicable. The NDN (protected) setup further includes a hash-based message authentication code (HMAC) salted with a pre-shared key. We limit the number of security contexts on the gateway to ten and on each sensor node to one. As a consequence, sensor devices maintain only one DTLS session concurrently and all secured content objects from a particular sensor device use a single security context. As we do not evaluate key management schemes, we compare all security protocols with pre-shared keys. Context related variables such as sequence counters are set to default on device start-up.

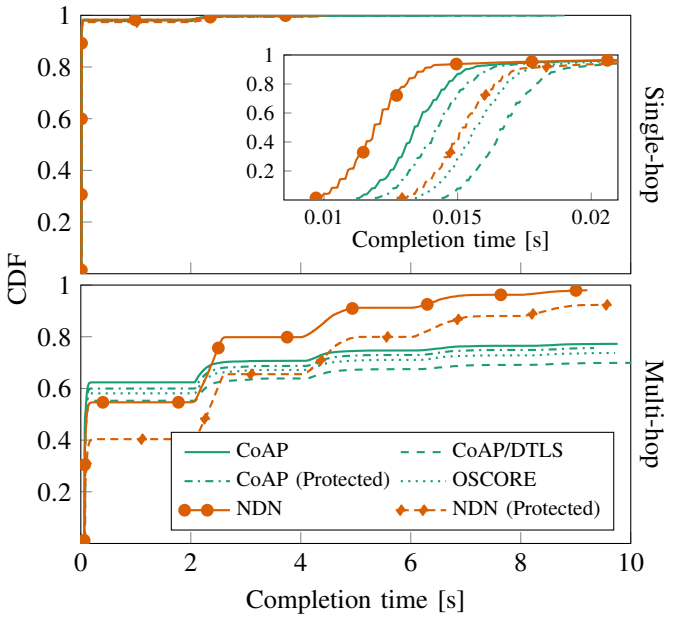


Fig. 9. Temporal distributions of content arrival times.

B. Time to Content Delivery

We examine the delays measured between content requests and content arrivals at the gateway. Fig. 9 combines the results for CoAP and NDN configurations in the single-hop and multi-hop setup. We first observe that protocol families are in rough accordance for the single-hop case. Temporal performances indicate a sub-second completion time for close to 100% of all transmissions across the protocols. The unprotected NDN configuration displays the fastest operation with 50% of transmissions finishing below 11 ms. Combining this observation with our previous result that NDN transmits the smallest request and response messages (see Fig. 3), we can conclude that unprotected NDN succeeds in quickly exchanging its small messages. In the unprotected CoAP configuration, 50% of transmissions finish below 13 ms. The protected protocol versions follow closely, whereby CoAP over DTLS is on the slow end with 50% of transmissions finishing only below 16 ms.

Next, we consider the more challenging multi-hop scenarios. Overall, results reveal much slower protocol operations. This reflects the common experience in low-power regimes that radio interferences and individual error probabilities accumulate over several hops and decrease reliability for the entire subnet. The staircase pattern visible for all protocols is based on request retransmissions at the configured interval of 2 s per retransmission. On the slower end, stairs show attenuating effects due to an accumulating jitter for each retransmission. We observe that all CoAP variants operate in agreement. Roughly 55–60% of content requests complete in the sub-second range without requiring retransmissions. Corrective actions, *i.e.,* request retransmissions, delay the completion time, but are able to increase the number of successful responses at the gateway to 70–77%.

The effects of inflated messages also become apparent.

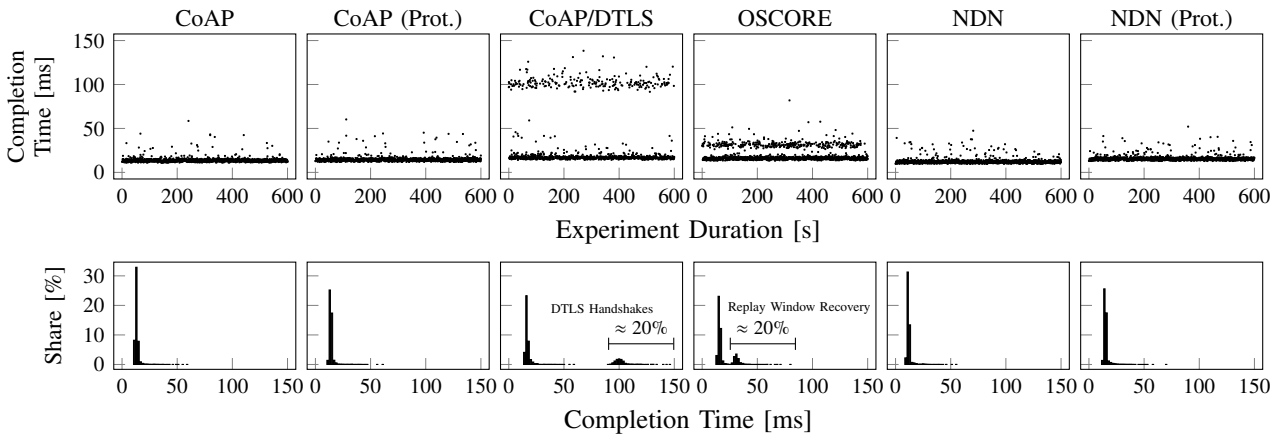


Fig. 10. Content arrival times and their percental distribution during the evolution of an experiment in a single-hop scenario, with simulated reboots after 20% of all exchanges. DTLS and OSCORE clearly reflect these events as delayed handshakes or reply window recovery, respectively.

CoAP keeps packets smallest and reveals a better performance than CoAP over DTLS, which requires the largest packets. The delay distributions for NDN show surprising results. Only 40–55% of all responses arrive in the sub-second range at the gateway and therefore seem to indicate an inferior performance of NDN compared with the CoAP variants. This discrepancy is nevertheless due to the different retransmissions strategies of CoAP and NDN. NDN implements hop-wise retransmission, which stepwise increases the packet numbers on the forwarding path and makes interference with parallel content requests on the wireless links more likely. Hop-wise retransmission with in-network caching, however, advances packets toward receivers in each step and converges more easily to successful content delivery. Hence, corrective actions are able to boost the overall reliability of the NDN family to 92–97%.

C. Security Overhead

We now inspect the case, in which an endpoint repeatedly connects to the IoT stub network to retrieve sensor readings. This setup follows the previous single-hop scenario with one minor change: The endpoint at the gateway simulates an unexpected reboot by losing any volatile cryptographic state after 20% of all exchanges. (With OSCORE, the simulated unexpected reboot happens at the sensor node instead, as an unexpected reboot of the gateway would have no visible effect at all). Fig. 10 summarizes the evolution of content arrival times throughout an experiment duration of ten minutes. In the top row, we measure times to completion during the ongoing experiments for all protocols, while the bottom row visualizes the distributions of the completion times as measured for each protocol. The supplementary TABLE III delivers an in-depth view on the statistical key properties of the arrival time distributions in Fig. 10, including the total average μ, standard deviation σ, the first quartile Q1 (25%), third quartile Q3 (75%), and the median.

Mostly the completion times reflect the results already presented in Fig. 9. Neither our protected or unprotected CoAP deployments, nor the NDN counterparts use volatile security state. The distributions of all transmission times of the experiment thus accumulates at around ≈12–15 ms, even for the case of unexpected reboots.

Protocol	μ [ms]	σ [ms]	Q1 [ms]	median [ms]	Q3 [ms]
CoAP	13.93	2.83	12.82	13.48	14.43
CoAP (Prot.)	14.72	2.94	13.57	14.22	15.18
CoAP/DTLS	34.09	34.00	16.29	17.27	23.07
OSCORE	16.42	2.72	15.23	16.08	17.00
NDN	12.44	2.91	11.28	11.95	12.89
NDN (Prot.)	15.79	3.15	14.52	15.33	16.15

TABLE III
STATISTICAL KEY PROPERTIES OF CONTENT ARRIVAL TIMES IN MILLISECONDS FOR SUCCESSFUL REQUESTS IN A SINGLE-HOP SCENARIO WITH SIMULATED REBOOTS.

An obvious exception to this is CoAP over DTLS. After a loss of cryptographic state at the gateway, a session handshake must precede the initial request. Such handshake for the configured cipher suite requires ten DTLS packet transmissions in total. Fig. 3 depicts the composition of these packets and clearly shows their considerable sizes. In some cases, handshake messages are much larger than the actual authenticated and encrypted user packets. Our evaluation shows that DTLS handshakes complete after around 100 ms, whereas in rare cases they even require up to 150 ms. Since these handshakes are measured on a single hop, they clearly serve as a good lower bound for DTLS negotiations in more complex scenarios.

A proper DTLS session resumption [40] and Connection IDs [17] could reduce the effects of handshakes after deep sleep or address changes, but are not available in tinyDTLS and require the persistence of context information that scales with the number of connected sensor devices.

The OSCORE recovery process also displays a delay introduced by the state loss. The effect, though, is much less pronounced as state recovery completes within a single round-trip as depicted in Fig. 4 c, and does not establish a fresh security context. The messages involved are not depicted in Fig. 3 for lack of a visible difference to the regular OSCORE

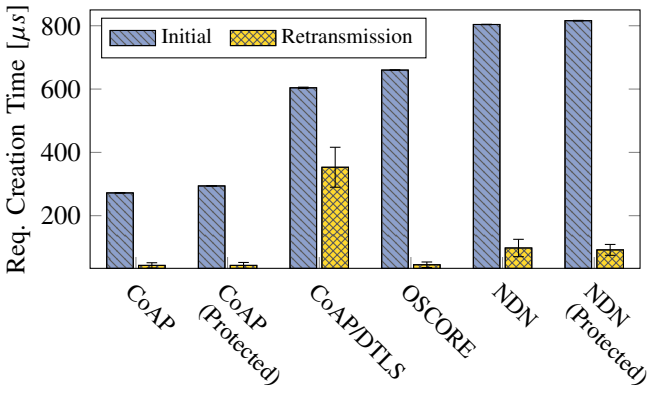


Fig. 11. Creation time for initial and retransmitted requests.

messages, as their sizes differ only by up to 4 bytes throughout the tests.

D. Request Creation Time

Our temporal protocol measurements finalize with evaluating the message creation time of requests. We start our measurement when an application triggers a request and stop the time as soon as the packet is passed to the lower layer, which is UDP in the CoAP variants and the link-layer in NDN. Fig. 11 visualizes the results using two bar plots for each protocol. The first bar shows average creation times for initial requests and the second bar denotes average creation times for request retransmissions.

CoAP in its unprotected and protected configurations exhibits the lowest creation times at around 280 μs. Since the protected version only affects responses, equal times for requests are expected. In both protocol versions, retransmissions are built much quicker in around 43 μs, since they already exist in retransmission buffers. NDN behaves similarly, but creation times increase to $\approx$ 810 μs for initial requests and $\approx$ 95 μs for request retransmissions. The latter is mainly due to complex (TLV) header structures, which require string parsing, and significantly less optimized implementations of packet processing in CCN Lite.

The behavior of CoAP over DTLS differs. Initial request creation times escalate to around 600 μs due to the authenticated encryption. In particular, request retransmissions reveal outlying results. Requests exist in CoAP retransmission buffers and reduce overall creation times, but they still require to pass through the DTLS layer. Hence, retransmission creation times spend on average around 353 μs and therefore eight times as long as other CoAP setups. This problem arises from layering independent protocols, which is not present in the OSCORE. Protection takes place on the CoAP layer and retransmission buffers already contain protected messages. Creation times for retransmissions reside at around 45 μs and are thus comparable to the protected and unprotected CoAP composition.

E. Impact of On-Path Caching

In our previous evaluation of temporal performance (see Section VI-B) we could observe that hop-wise

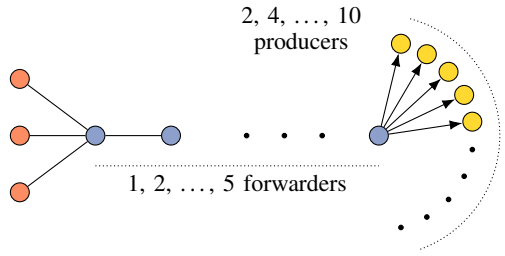


Fig. 12. Topology with three consumers and a varying number of forwarders as well as producers to gauge the effects of hop-wise caching.

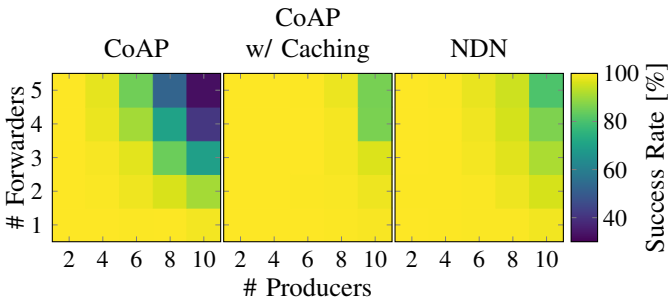


Fig. 13. Success rates for plain CoAP compared to alternative deployments with hop-wise caching capabilities.

(re-)transmission with content caching of NDN yielded higher success rates than the host-centric protocols in this low-power and lossy wireless regime. Notably, CoAP already brings the architectural feature of CoAP proxies with caches, which can be used to construct a deployment that is hop-centric like NDN.

Content object security integrates seamlessly into hop-by-hop deployments with untrusted CoAP proxies, whereas transport layer security (i) would require proxies to be included in trust relationships, and (ii) would cache content as unencrypted plain data. We envision OSCORE a natural candidate and a potential enabler for protected CoAP deployments that align with information-centric concepts such as on-path caching. This can increase the robustness of CoAP in networks with intermittent connectivity.

In a preview of future prospects and developments, we examine the effectiveness of content replication on request paths. For demonstration purposes, we consider an extended topology (Fig. 12) that consists of three consumer nodes, a number of forwarders varying from one to five, and a number of content producers increasing from two to ten. All producers host ten different temperature readings and all three consumers request every 2–3 seconds a random content item out of these ten sensor readings for a period of 100 requests per producer. Content caches for the extended CoAP deployment and for NDN are dimensioned to hold 30 content objects.

We compare CoAP with and without caching and NDN in Fig. 13. Success rates of data delivery are averaged over the three consumers. In all three deployments, we record success rates of close to 100% for simpler topologies. For the more complex topologies starting at three forwarders and six producers, we observe success rates that quickly collapse

down to ≈ 40% for the plain CoAP deployment, while the by caches extended CoAP and NDN setups are able to remain operationally successful at rates of ≈ 80% and above. The increased robustness in the presence of on-path caching has two reasons. First, caching shortens request paths for retransmissions of the same request, and second, content is pulled closer to the consumers, such that subsequent requests to the same CoAP resource do not need to fully traverse the request path. The small fluctuations in the success rates between NDN and CoAP with caching for the setups with more than six forwarders is due to different qualities of their implementations and their varying behavior under increased network stress. Furthermore, the open testbed infrastructure is shared by multiple users and interferences are unpredictable and not suppressible.

These phenomena have been known from the information-centric network world since years [57]. Content object security enabled by OSCORE may introduce these network optimizations soon into the CoAP ecosystem.

VII. Conclusions and Outlook

In this paper, we explored the vision of content object security on the network protocol level for the IoT. We analyzed current protocols and prospective developments that aim for securing content independent of network transport, and measured different configurations of CoAP, OSCORE, and NDN in a real-world testbed of constrained nodes. With this comprehensive study, we spanned the full solution space from end-to-end security sessions that act on transport channels to approaches that secure each data chunk individually. Our findings indicate that in challenging environments those protocols that can hop-wise transfer and cache content objects significantly outperform protocols which cannot. Smoothly integrated into the CoAP mechanics, OSCORE identified itself as a promising candidate for the former.

We further identified security overheads and the burdens from volatile and non-volatile protocol state that warrants session persistence. As explicit replay and reorder protection in NDN are unnecessary, energy-expensive I/O operations to non-volatile storage can be minimized and caching simplifies. In contrast, DTLS and OSCORE require up-to-date sequence numbers and sliding replay windows to prevail even after unexpected system resets, which marks these protocol elements as candidates for future optimization.

As we show both the impact of overheads and of hop-by-hop retransmissions, the present results help to justify, guide and (in future work) evaluate further improvements in the compared protocols: The upcoming DTLS 1.3 will optimize the record layer footprint [58]. OSCORE extensions that allow for (re-)establishing a security context (LAKE [20], EDHOC [59]) will need to keep extra round-trips to a minimum. Cacheable group observations [52] enables NDN-like multicast features in CoAP, which would be beneficial as discussed in this paper. We plan to analyze these emerging approaches in future work—toward architecting an information-centric, restful Web of things [60].

Acknowledgment

This work was supported in part by the German Federal Ministry for Education and Research (BMBF) within the project *PIVOT – Privacy-Integrated design and Validation in the constrained IoT*, and the Hamburg *ahoi.digital* initiative. The `libOSCORE` library was made with financial support from Ericsson AB.

A Note on Reproducibility. We fully support reproducible research [61], [62] and perform all our experiments using open source software and an open access testbed. Code and documentation will be available on Github at https://github.com/inetrg/tnsm-icn-coap-objectsecurity-2021.

References

[1] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-End Arguments in System Design," *ACM Trans. Comput. Syst.*, vol. 2, no. 4, pp. 277–288, Nov 1984.

[2] X. de Carnavalet and M. Mannan, "Killed by Proxy: Analyzing Client-end TLS Interception Software," in *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2016.

[3] R. Holz, T. Riedmaier, N. Kammenhuber, and G. Carle, "X.509 Forensics: Detecting and Localising the SSL/TLS Men-in-the-Middle," in *Computer Security – ESORICS 2012*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 217–234.

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

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

[6] A. Stanford-Clark and H. L. Truong, "MQTT For Sensor Networks (MQTT-SN) Version 1.2," IBM, Protocol Specification, November 2013. [Online]. Available: http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf

[7] A. Banks and R. G. (Eds.), "MQTT Version 3.1.1," OASIS, OASIS Standard, October 2014. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

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

[9] C. Gündogan, C. Amsüss, T. C. Schmidt, and M. Wählisch, "IoT Content Object Security with OSCORE and NDN: A First Experimental Comparison," in *Proc. of 19th IFIP Networking Conference*. Piscataway, NJ, USA: IEEE Press, June 2020, pp. 19–27. [Online]. Available: https://ieeexplore.ieee.org/document/9142731

[10] E. Baccelli, O. Hahm, M. Günes, M. Wählisch, and T. C. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," in *Proc. of the 32nd IEEE INFOCOM. Poster*. Piscataway, NJ, USA: IEEE Press, 2013, pp. 79–80.

[11] C. Bellman and P. C. van Oorschot, "Analysis, Implications, and Challenges of an Evolving Consumer IoT Security Landscape," in *17th International Conference on Privacy, Security and Trust (PST)*. IEEE, August 2019, pp. 1–7.

[12] C. Gündogan, P. Kietzmann, M. S. Lenders, H. Petersen, M. Frey, T. C. Schmidt, F. Shzu-Juraschek, and M. Wählisch, "The Impact of Networking Protocols on Massive M2M Communication in the Industrial IoT," *IEEE Transactions on Network and Service Management (TNSM)*, 2021. [Online]. Available: https://doi.org/10.1109/TNSM.2021.3089549

[13] J. Hester and J. Sorber, "The Future of Sensing is Batteryless, Intermittent, and Awesome," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 1–6.

[14] T. Schmidt, M. Waehlisch, and G. Fairhurst, "Multicast Mobility in Mobile IP Version 6 (MIPv6): Problem Statement and Brief Survey," IETF, RFC 5757, February 2010.

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

[16] H. Kwon, J. Park, and N. Kang, "Challenges in Deploying CoAP Over DTLS in Resource Constrained Environments," in *Information Security Applications*. Cham, Switzerland: Springer, 2016, pp. 269–280.

[17] E. Rescorla, H. Tschofenig, T. Fossati, and A. Kraus, "Connection Identifiers for DTLS 1.2," IETF, Internet-Draft – work in progress 13, June 2021.

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

[19] C. Amsuess, J. Mattsson, and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing," IETF, Internet-Draft – work in progress 13, July 2021.

[20] M. Vucinic, G. Selander, J. Mattsson, and D. Garcia-Carillo, "Requirements for a Lightweight AKE for OSCORE," IETF, Internet-Draft – work in progress 04, June 2020.

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

[22] M. Tiloca, G. Selander, F. Palombini, J. Mattsson, and J. Park, "Group OSCORE - Secure Group Communication for CoAP," IETF, Internet-Draft – work in progress 12, July 2021.

[23] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A Survey of Information-Centric Networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, July 2012.

[24] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos, "A Survey of Information-Centric Networking Research," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 2, pp. 1024–1049, 2014.

[25] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, "Information-Centric networking: Seeing the Forest for the Trees," in *Proc. of the 10th ACM HotNets Workshop*, ser. HotNets-X. New York, NY, USA: ACM, 2011.

[26] V. Jacobson, D. K. Smetters, J. D. Thornton, and M. F. Plass, "Networking Named Content," in *5th Int. Conf. on emerging Networking Experiments and Technologies (ACM CoNEXT'09)*. New York, NY, USA: ACM, Dec. 2009, pp. 1–12.

[27] E. Baccelli, C. Mehlis, O. Hahm, T. C. Schmidt, and M. Wählisch, "Information Centric Networking in the IoT: Experiments with NDN in the Wild," in *Proc. of 1st ACM Conf. on Information-Centric Networking (ICN-2014)*. New York: ACM, September 2014, pp. 77–86. [Online]. Available: http://dx.doi.org/10.1145/2660129.2660144

[28] C. Gündogan, P. Kietzmann, T. C. Schmidt, and M. Wählisch, "ICNLoWPAN – Named-Data Networking in Low Power IoT Networks," in *Proc. of 18th IFIP Networking Conference*. Piscataway, NJ, USA: IEEE Press, May 2019, pp. 1–9. [Online]. Available: http://dx.doi.org/10.23919/IFIPNetworking.2019.8816850

[29] E. Baccelli, C. Gündogan, O. Hahm, P. Kietzmann, M. Lenders, H. Petersen, K. Schleiser, T. C. Schmidt, and M. Wählisch, "RIOT: an Open Source Operating System for Low-end Embedded Devices in the IoT," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4428–4440, December 2018. [Online]. Available: http://dx.doi.org/10.1109/JIOT.2018.2815038

[30] M. Lenders, P. Kietzmann, O. Hahm, H. Petersen, C. Gündogan, E. Baccelli, K. Schleiser, T. C. Schmidt, and M. Wählisch, "Connecting the World of Embedded Mobiles: The RIOT Approach to Ubiquitous Networking for the Internet of Things," Open Archive: arXiv.org, Technical Report arXiv:1801.02833, January 2018. [Online]. Available: https://arxiv.org/abs/1801.02833

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

[32] D. Whiting, R. Housley, and N. Ferguson, "Counter with CBC-MAC (CCM)," IETF, RFC 3610, September 2003.

[33] C. Tschudin, C. Scherb *et al.*, "CCN Lite: Lightweight implementation of the Content Centric Networking protocol," 2018. [Online]. Available: http://ccn-lite.net

[34] P. Kietzmann, L. Boeckmann, L. Lanzieri, T. C. Schmidt, and M. Wählisch, "A Performance Study of Crypto-Hardware in the Low-end IoT," in *International Conference on Embedded Wireless Systems and Networks (EWSN)*. New York, USA: ACM, February 2021.

[35] J. Mattsson, J. Fornehed, G. Selander, F. Palombini, and C. Amsuess, "Controlling Actuators with CoAP," IETF, Internet-Draft – work in progress 06, September 2018.

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

[37] J. Mattsson and D. Migault, "ECDHE_PSK with AES-GCM and AES-CCM Cipher Suites for TLS 1.2 and DTLS 1.2," IETF, RFC 8442, September 2018.

[38] G. Selander, J. Mattsson, and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)," IETF, Internet-Draft – work in progress 01, March 2020.

[39] D. McGrew, "An Interface and Algorithms for Authenticated Encryption," IETF, RFC 5116, January 2008.

[40] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State," IETF, RFC 5077, January 2008.

[41] Y. Yu, A. Afanasyev, D. Clark, kc claffy, V. Jacobson, and L. Zhang, "Schematizing Trust in Named Data Networking," in *Proceedings of the 2nd ACM Conference on Information-Centric Networking*, ser. ICN '15. New York, NY, USA: ACM, 2015, pp. 177–186.

[42] A. Compagno, M. Conti, and R. Droms, "OnboardICNg: a Secure Protocol for On-boarding IoT Devices in ICN," in *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, ser. ICN '16. New York, NY, USA: ACM, 2016, pp. 166–175.

[43] T. Mick, R. Tourani, and S. Misra, "LASeR: Lightweight Authentication and Secured Routing for NDN IoT in Smart Cities," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 755–764, 2018.

[44] M. Bellare and P. Rogaway, "Entity Authentication and Key Distribution," in *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '93. Berlin, Heidelberg: Springer, 1993, pp. 232–249.

[45] F. Bersani and H. Tschofenig, "The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) Method," IETF, RFC 4764, January 2007.

[46] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)," IETF, RFC 4861, September 2007.

[47] T. Narten, R. Draves, and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6," IETF, RFC 4941, September 2007.

[48] C. Amsuess, Z. Shelby, M. Koster, C. Bormann, and P. van der Stok, "CoRE Resource Directory," IETF, Internet-Draft – work in progress 28, March 2021.

[49] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture," IETF, RFC 2373, July 1998.

[50] P. Kietzmann, C. Gündogan, T. C. Schmidt, O. Hahm, and M. Wählisch, "The Need for a Name to MAC Address Mapping in NDN: Towards Quantifying the Resource Gain," in *Proc. of 4th ACM Conference on Information-Centric Networking (ICN)*. New York, NY, USA: ACM, September 2017, pp. 36–42.

[51] M. Tiloca and E. Dijk, "Proxy Operations for CoAP Group Communication," IETF, Internet-Draft – work in progress 04, July 2021.

[52] M. Tiloca, R. Hoeglund, C. Amsuess, and F. Palombini, "Observe Notifications as CoAP Multicast Responses," IETF, Internet-Draft – work in progress 04, November 2020.

[53] C. Amsuess and M. Tiloca, "Cacheable OSCORE," IETF, Internet-Draft – work in progress 02, July 2021.

[54] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne, "FIT IoT-LAB: A large scale open experimental IoT testbed," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, Dec 2015, pp. 459–464.

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

[56] Atmel, *Low Power 2.4 GHz Transceiver for ZigBee, IEEE 802.15.4, 6LoWPAN, RF4CE, SP100, WirelessHART, and ISM Applications*, Atmel Corporation, September 2009. [Online]. Available: http://www.atmel.com/images/doc8111.pdf

[57] C. Gündogan, P. Kietzmann, M. Lenders, H. Petersen, T. C. Schmidt, and M. Wählisch, "NDN, CoAP, and MQTT: A Comparative Measurement Study in the IoT," in *Proc. of 5th ACM Conference on Information-Centric Networking (ICN)*. New York, NY, USA: ACM, September 2018, pp. 159–171. [Online]. Available: https://doi.org/10.1145/3267955.3267967

[58] J. Mattsson and F. Palombini, "Comparison of CoAP Security Protocols," IETF, Internet-Draft – work in progress 01, March 2018.

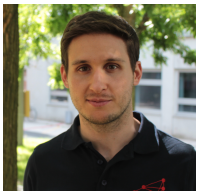
[59] F. Palombini, M. Tiloca, R. Hoeglund, S. Hristozov, and G. Selander, "Combining EDHOC and OSCORE," IETF, Internet-Draft – work in progress 01, November 2020.

[60] C. Gündogan, C. Amsüss, T. C. Schmidt, and M. Wählisch, "Toward a RESTful Information-Centric Web of Things: A Deeper Look at Data Orientation in CoAP," in *Proc. of 7th ACM Conference on Information-Centric Networking (ICN)*. New York: ACM, September 2020, pp. 77–88. [Online]. Available: https://doi.org/10.1145/3405656.3418718

[61] ACM, "Result and Artifact Review and Badging," http://acm.org/publications/policies/artifact-review-badging, Jan., 2017.

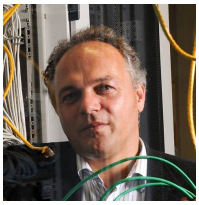
[62] Q. Scheitle, M. Wählisch, O. Gasser, T. C. Schmidt, and G. Carle, "Towards an Ecosystem for Reproducible Research in Computer Networking," in *Proc. of ACM SIGCOMM Reproducibility Workshop*. New York, NY, USA: ACM, August 2017, pp. 5–8.

AUTHOR BIOGRAPHY



Cenk Gündoğan received the M.Sc. degree in computer science from the Institut für Informatik, Freie Universität Berlin, Germany, in 2016. Currently, he is pursuing the Ph.D. degree with the Internet Technologies Group, Hamburg University of Applied Sciences, Germany, and explored within the I3 project—Information Centric Networking (ICN) for the Industrial Internet—routing, QoS, and resilience in ICN-based and IoT tailored networks. Recently, Cenk Gündoğan put focus on a data-centric Web of Things deployment option by applying ICN principles to the IETF envisioned IoT network stack. He is one of the core developers and maintainer of RIOT.

Christian Amsüss is an independent network architect and software developer. His involvment in commercial projects (in recent years primarily around the Internet of Things) gives him a firm grasp of practical requirements, while his participation in Free Software projects allows him to gain a wider perspective, and to explore new technologies in early stages. Christian is active in IETF working groups surrounding Constrained Restful Environments. His long term goal is to shape components of the Internet of Things towards an ecosystem where interoperability can be exepected, the user is in control, and which is secure by default.



Thomas C. Schmidt is professor of Computer Networks and Internet Technologies at Hamburg University of Applied Sciences (HAW), where he heads the Internet Technologies research group (iNET). Prior to moving to Hamburg, he was director of a scientific computer centre in Berlin. He studied mathematics, physics and German literature at Freie Universitaet Berlin and University of Maryland, and received his Ph.D. from FU Berlin in 1993. Since then he has continuously conducted numerous national and international research projects. He was the principal investigator in a number of EU, nationally funded and industrial projects as well as visiting professor at the University of Reading, U.K.. His continued interests lie in the development, measurement, and analysis of large-scale distributed systems like the Internet. He serves as co-editor and technical expert in many occasions and is actively involved in the work of IETF and IRTF. Together with his group he pioneered work on an information-centric Industrial IoT and the emerging data-centric Web of Things. Thomas is a co-founder of several large open source projects and coordinator of the community developing the RIOT operating system - the friendly OS for the Internet of Things.

Matthias Wählisch is an Assistant Professor of Computer Science at Freie Universität Berlin where he heads the Internet Technologies Research Lab. He received his Ph.D. in computer science with highest honors from Freie Universität Berlin. His research and teaching focus on efficient, reliable, and secure Internet communication. This includes the design and evaluation of networking protocols and architectures, as well as Internet measurements and analysis. His efforts are driven by improving Internet communication based on sound research.. Matthias is the PI of several national and international projects, supported by overall 4.7M EUR grant money. He published more than 150 peer-reviewed papers (*e.g.,* at ACM HotNets, ACM IMC, The Web Conference). Since 2005, Matthias is active within IETF/IRTF, including eight RFCs and several Internet drafts. His research results have been distinguished multiple times. Amongst others, he received the Young Talents Award of Leibniz-Kolleg Potsdam for outstanding achievements in advancing the Internet, as well as the Excellent Young Scientists Award (10,000 EUR) for his contributions to the Internet of Things and their prospective entrepreneurial practice. He co-founded some successful open source projects such as RIOT, where he is still responsible for the strategic development.