Trust Assessment in 32 KiB of RAM: Multi-application Trust-based Task Offloading for Resource-constrained IoT Nodes

Matthew Bradbury
Department of Computer Science
University of Warwick
Coventry, UK
M.Bradbury@warwick.ac.uk

Arshad Jhumka
Department of Computer Science
University of Warwick
Coventry, UK
H.A.Jhumka@warwick.ac.uk

Tim Watson
WMG
University of Warwick
Coventry, UK
tw@warwick.ac.uk

ABSTRACT

There is an increasing demand for Internet of Things (IoT) systems comprised of resource-constrained sensor and actuator nodes executing increasingly complex applications, possibly simultaneously. IoT devices will not be able to execute computationally expensive tasks and will require more powerful computing nodes, called *edge* nodes, for such execution, in a process called *computation offloading*. When multiple powerful nodes are available, a selection problem arises: which edge node should a task be submitted to? This problem is even more acute when the system is subjected to attacks, such as DoS, or network perturbations such as system overload. In this paper, we present a trust model-based system architecture for computation offloading, based on *behavioural evidence*. The system architecture provides confidentiality, authentication and non-repudiation of messages in required scenarios and will operate within the resource constraints of embedded IoT nodes. We demonstrate the viability of the architecture with an example deployment of *Beta Reputation System* trust model on real hardware.

CCS CONCEPTS

• **Computer systems organization** → *Sensor networks*; • **Security and privacy** → **Trust frameworks**.

KEYWORDS

Trust, computation offloading, Internet of Things, resource constrained, edge computing

ACM Reference Format:
Matthew Bradbury, Arshad Jhumka, and Tim Watson. 2021. Trust Assessment in 32 KiB of RAM: Multi-application Trust-based Task Offloading for Resource-constrained IoT Nodes. In *The 36th ACM/SIGAPP Symposium on Applied Computing (SAC '21), March 22–26, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3412841.3441898

1 INTRODUCTION

Internet of Things (IoT) devices are being deployed in a variety of contexts including smart farming [16], healthcare [15] and smart cities [22]. IoT devices have typically been deployed as a distributed system to perform sensing of their environment. Recently, there has been interest in these devices to perform more complex tasks (including actuation). An issue is that many of these devices are resource-constrained, with limited processing power, data storage, energy storage and other constraints.

Due to these resource constraints, it will be infeasible for IoT devices[1] to perform computationally expensive tasks, e.g., machine learning. Therefore, these tasks will need to be sent to resource-rich (powerful) devices (or nodes) to execute, in a process called *computational offloading*. These nodes may be accessible via the internet (e.g., a Cloud service) or via edge nodes[2] that exist within the same local network as the resource-constrained devices. For a large class of applications, offloading to edge nodes is preferred, e.g., when latency is important.

To meet demand, multiple edge nodes should be provisioned in the network. However, a selection problem then arises: which edge node should an IoT node choose to submit a task to? This is typically addressed in the literature by *evidence-based behavioural trust* [32, 38], where the incidence of how well an edge node has correctly executed tasks in the past is recorded and used as a predictor of how likely that edge node is to correctly execute future tasks. Trust is typically evaluated at the application level and sufficient storage is needed to record information about each edge node of interest.

In vehicular and cellular networks, the task offloading problem is referred to as Multi-access Edge Computing (MEC) [25]. However, trust models that require a large amount of memory or processing power to compute are not viable for resource-constrained (IoT) devices. Hence, simpler models, e.g., the Beta Reputation System [21] or hidden Markov models (HMM) [14], can be used instead.

While a suitable trust model is vital, its correct deployment is equally important. Common internet infrastructure is typically unsuitable due to the same resource constraints that necessitate task offloading. Therefore, in this paper, we propose and describe a system architecture for trust-based task offloading. The architecture is designed to support arbitrary trust models and multiple applications running on both edge nodes and IoT devices.

We make the following contributions:

(1) We propose a system architecture for performing trust-based task offloading for IoT devices.
(2) We profile cryptographic operations on Zolertia RE-Motes and use this to inform the message protection strategy.
(3) We conduct a small deployment of the Beta Reputation System using Zolertia RE-Motes to demonstrate the efficacy of the system.

[1] By IoT devices, we mean devices that are resource-constrained.
[2] By edge nodes, we mean resource-rich nodes at the edge of the network.

SAC '21, March 22–26, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8104-8/21/03.
https://doi.org/10.1145/3412841.3441898

The rest of this paper is structured as follows: in Section 2 we presented related work. Section 3 describes the system model and Section 4 specifies the problem statement. In Section 5 the individual components of the system architecture are described. Section 6 describes the experimental setup used to obtain the results presented in Section 7. A discussion of the system is presented in Section 8 before concluding in Section 9.

2 RELATED WORK

There has been much work on standardising the fundamentals of IoT device infrastructure. Typically many protocols designed for general internet use-cases are too resource intensive and are designed for high performance and not minimising costs in terms of RAM, flash, computation, and energy. So, alternative protocols for these resource-constrained systems have been developed, such as: uIPv6 [11] for addressing, TSCH [36] for energy-efficient wireless medium access and RPL [1] for packet routing. Higher level protocols have been implemented on top of these, such as CoAP [31] which provides similar functionality to HTTP.

While security protocols such as DTLS can be used to protect UDP traffic, there has been recent effort to standardise security protocols specific to CoAP. OSCORE [28] provides encryption and authentication of messages. Only a subset of headers are protected to facilitate proxying which changes some CoAP fields. A benefit to OSCORE is that it has low overhead compared to DTLS [18, 19] and there remain unaddressed issues with multiple DTLS implementations [17], however, OSCORE does not provide forward secrecy. Other approaches can involve trusted execution environments such as ARM TrustZone [27].

Trust has also been used to solve a variety of problems in IoT applications. Primarily, the security protocols deployed typically need to provide identity trust, where one node can verify the authenticity of a message sent from another node. Trust has been used in a variety of areas, such as routing of messages in wireless sensor networks [9], attack detection (such as intrusion into the network) [6] and localisation [2]. In these areas, trust is evaluated based on observations made about the device's behaviours.

Trust models are important for systems where nodes do not always behave correctly. Nodes may exhibit selective behaviour, where services are correctly performed only some of the time [34]. For example, to save energy a node may choose not to forward all messages that are routed through it. In networks where trust is derived from reputation information, the impact of manipulations of this information needs to be mitigated. An example is when nodes bad-mouth other nodes by lying and indicating they have a low trust value for another node [34].

There has been much work on developing trust models to solve these problems beyond resource-constrained systems [32, 38]. In vehicle and cellular networks the task offloading problem (which we focus on in this paper) is referred to as Multi-access Edge Computing (MEC) (previously Mobile Edge Computing) [25]. A variety of solutions have been proposed for trust-based offloading in MEC systems, typically involving machine learning approaches [26, 37], linear programming (LP) [10], or game theoretic approaches [30]. However, these solutions are typically unsuitable for use in resource-constrained systems. Many machine learning models, though not all, require a large amount of memory or are expensive to compute, and techniques such as LP or game theoretic approaches would require large amounts of data to be sent to a central location to be processed into a schedule which is costly in terms of energy. This typically means that lightweight approaches are needed that are evaluated on resource-constrained IoT devices.

The seminal example of a lightweight trust model is the Beta Reputation System (BRS) [21]. The BRS maintains two counters which are parameters to the Beta distribution, the number of good events observed (α) and the number of bad events observed (β). A trustor can calculate a trust value about a trustee via the expected value of this distribution (the ratio of good events to the total number of events). These values can be updated with more observations, allowing the belief in the trustee to be refined.

The BRS and other models such as those that use HMMs [14] can allow the representation of trust in a small amount of space. However, for these trust models to be effective in selecting a target for task offloading, they need a suitable system to feed them with observations and then deliver tasks to the chosen node.

3 SYSTEM MODEL

The system is modelled as a graph $G = (V, E)$, where:

- $V = V_R \cup V_C \cup \{\rho\}$ is the set of nodes in the network, made up of edge nodes (V_R), IoT nodes (V_C), and a *root* node ρ,
- $E \subseteq V \times V$ is the set of communication links between nodes in the network.

IoT devices exist within the network to perform sensing and actuation. They are battery-powered with limited CPU power, RAM, energy storage, and potentially no stable storage. For example, the Zolertia RE-Mote [39] has a 32 MHz CPU, 32 KiB of RAM, 512 KiB of programmable flash, a 800 mAh battery, and support for an optional SD card. Communication in these devices is typically performed using IEEE 802.15.4, Bluetooth Low Energy, or LoRaWAN. Edge nodes support the IoT nodes by executing tasks that are either too expensive or require access to data unavailable to IoT devices. The special root node is equipped with similar resources to edge nodes and performs dedicated tasks for the system.

4 PROBLEM STATEMENT

There is a set of applications $\mathcal{A}$ deployed in the network, for each of which there is a variant deployed on an IoT devices $\mathcal{A}_C$ and a variant deployed on edge nodes $\mathcal{A}_R$. There is a bijection $f_{app} : \mathcal{A}_R \to \mathcal{A}_C$ from the edge node applications to IoT device applications. We assume two functions (i) $A_C : V_C \to 2^{\mathcal{A}_C}$ that returns the set of applications on an IoT device and (ii) $A_R : V_R \to 2^{\mathcal{A}_R}$ that returns the set of applications on an edge node.

Tasks generated on IoT device c for application $a \in A_C(c)$ will need to be delivered to an edge node that hosts the corresponding application. The edge nodes that can process these tasks are:

$$V_R^a = \{\, r \mid r \in V_R \wedge \left(\exists a' \in A_R(r)\,, f_{app}(a') = a\right) \,\} \tag{1}$$

Definition 4.1 (Task Offloading). *Given a IoT device $c \in V_C$ and the task for application $a \in A_C(c)$ that c needs to offload, which edge node $r \in V_R^a$ should the job be offloaded to such that: (i) the task will be accepted, (ii) r returns a result within some deadline d, and (iii) the result is correct.*

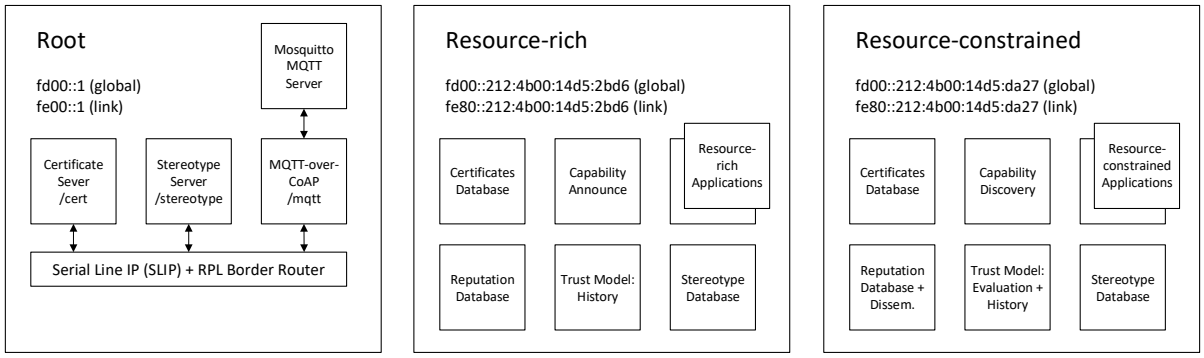


Figure 1: System functionality across the three device classes with example addresses

We assume that application tasks may not always be performed correctly by edge nodes and that there may be failures in any of the three conditions in Definition 4.1 due to edge nodes choosing to intentionally behave badly (e.g., wanting to prefer some capabilities over others) or due to other failures (e.g., transient network failures) that may cause packets to be lost. The deadline of a task is application-specific, some applications may be critical and have an early deadline, whereas non-critical applications will have later (possibly flexible) deadlines.

We propose to use a measure of *trust* as one approach to solve the Task Offloading problem. The metric captures the likelihood of a node to correctly execute an offloaded task. However, in order to capture this behavioural trust, a system must first provide: (i) identity trust and confidentiality, where messages between nodes can be authenticated and protected, (ii) mechanisms to facilitate the discovery of edge nodes and their capabilities, and (iii) mechanisms to submit tasks, receive responses and make observations about these actions. Depending on the trust model in use, it may also be necessary to (iv) provide stereotype information about nodes to bootstrap trust, and (v) facilitate the dissemination of reputation information. This paper does not provide a solution to the Task Offloading problem, but instead presents a system architecture which facilitates the deployment of trust models that do.

5 SYSTEM ARCHITECTURE

In this section we present a high-level description of our architecture to perform trust-based task offloading, before describing in detail the individual components. The system relies upon uIPv6 [11] and RPL [1] for message routing, and CoAP [31] for reliable messaging. As CoAP uses UDP this avoids the RAM cost of including the TCP stack. CBOR [7] is used to encode the contents of CoAP messages. For the security layer, OSCORE [28] provides encryption and authentication of CoAP messages. We plan to use Group OSCORE [33] for messages that require no encryption but require being digitally signed, as no implementation is available yet for the draft standard.

We have performed an implementation[3] using Contiki-NG [12]. The Contiki-NG operating system uses a coroutine-based cooperative scheduling model [13] instead of a multi-threaded model. The impact of this design is that the multiple applications running need to ensure that they behave well to avoid impacting other applications and tasks. For example, they will need to yield often enough to allow other coroutines to execute.

This feature set is not limited to Contiki-NG, and other operating systems (such as RIOT [4], Zephyr[4], OpenThread[5], and others) have a similar set of features. While our implementation is specific to Contiki-NG, the system architecture can be implemented on alternate IoT OSes that support the required features.

A single root node is required as part of Contiki-NG's implementation of RPL. On this single root node, a CoAP server (implemented using aiocoap [3]) will be used to provide services to the network. An overview of the system is shown in Figure 1.

We make the following assumptions as part of the development of this system architecture:

(1) As IoT devices have finite lifetimes, they may be retrieved and have batteries swapped at which point firmware updates may be performed.
(2) The multiple applications running on a single device are assumed to be *mutually trusted* [35], where one application does not intentionally aim to negatively impact another.
(3) A measure of trust in an edge node is evaluated on the IoT devices. While trust could be evaluated elsewhere in the network, there are costs involved with the transmission of observations and the device evaluating trust would need to be assumed to behave well.

5.1 Public Key Infrastructure

This system is primarily focused on evaluating behavioural trust, where the edge node is selected via an evidence-based evaluation of their past behaviour. However, in order to provide a foundation for behavioural-based trust, it is necessary that nodes in the network have trust in the identities of other network nodes.

Each IoT device is pre-deployed with a root certificate, their own certificate, and their secret key. This implementation uses the NIST P-256 elliptic curve (EC) (also know as sepc256r1) for ECC keys because the keys and signatures both take up a small amount of space (64 B) compared to keys required for RSA at comparable bits of security [23]. ECDSA signatures also provide non-repudiation for messages that are digitally signed. However, a downside is that

[3]Implementation source code: https://github.com/MBradbury/iot-trust-task-alloc.

[4]https://zephyrproject.org
[5]https://openthread.io

```
Certificate = [
    tbscertificate  : TBSCertificate,
    signature       : bytes .size 64
]
TBSCertificate = [
    serial_number   : uint,
    issuer          : bytes .size 8,
    validity        : [notBefore: uint, notAfter: uint],
    subject         : bytes .size 8,
    stereotype_tags : StereotypeTags,
    public_key      : bytes .size 64
]
StereotypeTags = [
    device_class    : uint
]
```

Figure 2: CDDL definition of lightweight certificate

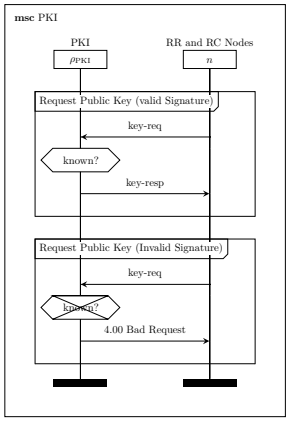


Figure 3: PKI Protocol

ECC operations are time consuming to compute, therefore we aim to minimise the use of ECC operations where appropriate.

Due to the large size of X.509 certificates we use a CBOR-encoded certificate similar to [20], whose contents is shown in Figure 2. While the certificates support including the time at which they are valid, not all systems may be capable of checking the validity. This is because there may be no time-synchronisation protocol in use that allows IoT devices to align their local clock with a global clock. So in this system, certificates can be purged from the root node once IoT devices are expected to have run out of battery.

To minimise the number of ECC operations that IoT devices perform, a shared secret is provided to an OSCORE context so for the majority of operations AES-CCM is used to encrypt and provide authentication of messages. In order for a node n_1 to create an OSCORE context with another node n_2, elliptic curve Diffie–Hellman (ECDH) is first used to generated a shared secret using n_2's public key and n_1's secret key. Therefore, ECC operations are only required when deriving the OSCORE context, digitally signing/verifying specific messages, and verifying certificates.

At network deployment not all edge nodes may be known, this means there is a need for IoT devices to be able to request keys for unknown nodes from a key server on the trusted root node. The protocol for this is shown in Figure 3.

Name	MQTT Topic
announce	edge/+/announce
unannounce	edge/+/unannounce
capability add	edge/+/capability/+/add
capability remove	edge/+/capability/+/remove

Table 1: MQTT Topics

5.2 Resource-rich Capability Discovery

IoT devices need to discover edge nodes and their capabilities (i.e., what applications they are running). Discovery of these capabilities aligns with a publish-subscribe model where IoT devices subscribe to announcements of edge nodes publishing their capabilities. MQTT [5] is a pub-sub protocol designed for IoT devices, however, it has a number of downsides when integrating with this system. Primarily, MQTT uses TCP to provide reliability, which means that there would be an additional RAM cost by including the TCP library. The use of MQTT would also mean that the security mechanisms protecting CoAP messages could not be applied to MQTT messages. To mitigate this overhead on the IoT devices, we instead implement MQTT-over-CoAP, where an application on the root node translates CoAP messages into MQTT messages and vice versa. The MQTT-over-CoAP translator application communicates with a Mosquitto [24] server that provides the MQTT functionality.

There are four phases to resource-rich capability discovery which are shown in Figure 4. The first requires IoT devices to `subscribe` to the four topics in Table 1. The first wildcard entry (represented by +) is the edge node's EUI-64 in hexadecimal and the second wildcard entry is the name of the capability.

The `announce` topic is used for edge nodes to announce themselves to others in the network. Their lightweight certificate is included in the message so receivers do not need to request it. The receivers will validate the certificate upon reception. The `capability add` topic is used for edge nodes to inform subscribers that a specific capability is being provided by that node.

The `unannounce` and `capability remove` topics are used by well-behaved edge nodes to inform subscribers that the node or a capability is unavailable respectively. Malicious nodes may not publish these messages, so IoT devices will need to be able to handle this scenario when encountered.

5.3 Resource-rich Stereotype Request

An issue in trust-based selection is that when the system is starting or a new entrant joins, there is little opportunity for historical data to have been gathered and used to build a trust model. Therefore, in order for facilitate better initial decisions, stereotypes can be provided as a starting point to bootstrap trust models [32].

When an edge node announces itself, the certificate it sends contains a set of tags which provide an abstract description of the node. Once these tags are received, the stereotype for this set of tags is requested from the root node as shown in Figure 5. When choosing which IoT device to submit a task to, the stereotype with the closest set of matching tags may be used in the process of calculating the trust value for that edge node.

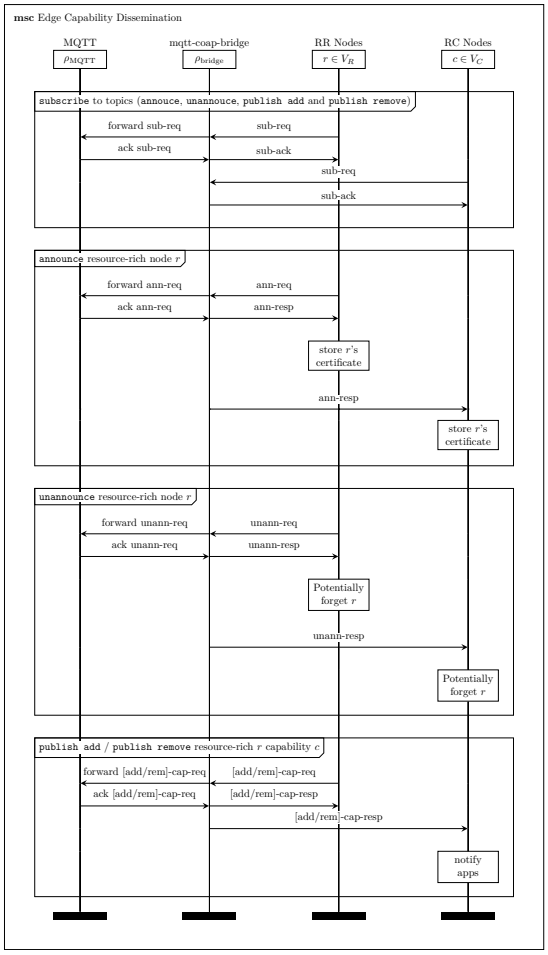


Figure 4: Resource-rich Capability Discovery Protocol

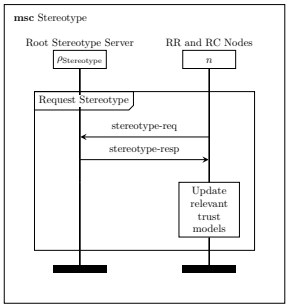


Figure 5: Resource-rich Stereotype Request

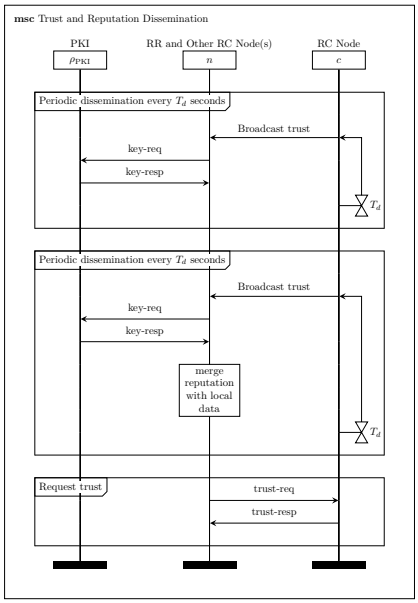


Figure 6: Trust Dissemination Protocol

5.4 Reputation Dissemination

Trust models may incorporate a measure of *reputation* into their evaluation of the trustworthiness of an edge node. The reputation of a trustee is the beliefs held by other trustors in the system and it is stored in the same format as the trust model held by other trustors. When a trust model incorporates reputation, each of the IoT devices need a mechanism to disseminate their beliefs. It is important to provide non-repudiation for messages containing reputation of trustees so nodes cannot claim they had a different trust value in the past. There is also no need for confidentiality, meaning the messages can be signed and sent unencrypted.

There are multiple options for implementing dissemination of reputation information, such as: (i) performing a network-wide multicast, (ii) targeting specific nodes, (iii) performing a 1-hop broadcast, or (iv) allowing nodes to request reputation information. In this implementation we focus on (iii) and (iv) where IoT devices perform a periodic dissemination of trust values and also allow other nodes to request reputation information from arbitrary nodes. Both of these actions are shown in Figure 6.

5.5 Application

The messages that applications on IoT devices send to edge nodes (the task-request and vice versa the task-response), will be protected by the OSCORE layer via encryption and authentication of the message. While, some applications may not require confidentiality, a generic layer will need to encrypt the messages in order to facilitate applications that do require it. An example of the application protocol is shown in Figure 7 for an application that sends a periodic task every T seconds. When a `capability add` for this application is received, the application is started if it is not running.

For the first periodic action, an edge node may be selected which lacks the IoT device's certificate. The edge node will not acknowledge the message and instead request the certificate. Either, when the IoT device retries the certificate will have been retrieved, or the IoT device will eventually timeout. Subsequent actions will not need to repeat this as the edge nodes will cache the certificate.

In the case of failure, applications may choose to resubmit a task to an alternate edge node. This is left up to the application as it will require buffering the task to facilitate retrying it.

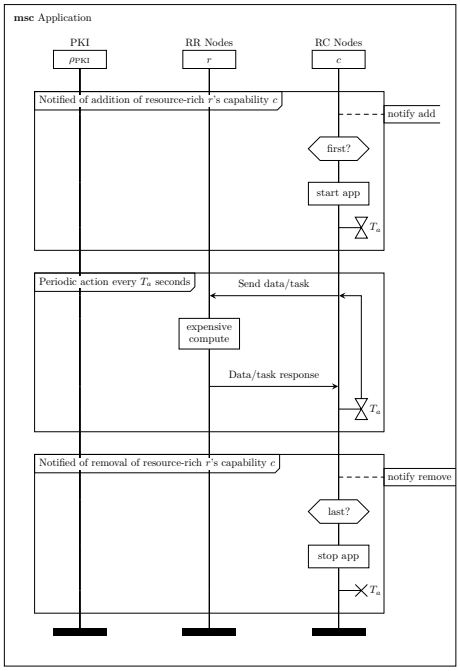


Figure 7: Application Protocol

6 EXPERIMENTAL SETUP

We perform experiments[6] using a deployment of six Zolertia RE-Motes which have hardware acceleration for SHA2, AES-CCM-16-64-128 (used by OSCORE), and 256 bit ECC operations. The key benefit is hardware support for ECC operations which take a long time to compute. Contiki-NG's implementation allows the CPU to execute other (potentially time sensitive) tasks while a message is being signed or verified. Each RE-Mote was attached to a Raspberry Pi which logged output. Two of the Raspberry Pis acted as Edge nodes, performing expensive computation for applications.

The system frequently publishes capabilities (every 2 min), disseminates trust (every 2 min), generates a monitoring task (every 1 min) and generates a routing task (every 2 to 3 min). These rates are higher than typical in order to obtain results in a reasonable time and would need to be adjusted based on the deployment performed.

To avoid memory fragmentation, fixed-sized pools are allocated at compile time. Table 2 shows the default maximum number of different types of objects that can be allocated and their RAM cost. These values would need to be adjusted for different network sizes.

6.1 Example Trust Model

To illustrate the operation of this system we implement an example trust model using the BRS. The trust value for each metric m, edge node r and application a is beta-distributed $\mathcal{T}_m(r,a) \sim \text{Beta}(\alpha,\beta)$ (application-specific) or $\mathcal{T}_m(r) \sim \text{Beta}(\alpha,\beta)$ (application-agnostic) where α is the number of successful interactions and β is the number of unsuccessful interactions. The expected value of the distribution summarises the number of successful events that have occurred.

$$E[X] = \frac{\alpha}{\alpha+\beta} \text{ where } X \sim \text{Beta}(\alpha,\beta) \tag{2}$$

[6]Raw results for these experiments can be found at [8]

Name	Count	Entry (B)	Total Size (B)
Certificates	12	288	3 456
Stereotypes	5	24	120
Edges	4	52	208
Edge Capabilities	12	28	336
Peers	8	32	256
Peer Edges	32	32	1 024
Peer Edge Capabilities	96	16	1 536

Table 2: Configuration constants and RAM cost

For any application-agnostic metric m: $\mathcal{T}_m(r,a) = \mathcal{T}_m(r)$.

Each IoT device c maintains three sets of Beta distributions that summarise interactions with edge node r and application a:

- $\mathcal{T}_{\text{sub}}(r)$ — Did r inform c that a task was received and will be executed?
- $\mathcal{T}_{\text{res}}(r)$ — Did r provide a result for a task?
- $\mathcal{T}_{\text{corr}}(r,a)$ — Was the result that r provided for application a *correct*?

Correctness is an application-specific and best-effort attempt to validate if a result for a task conforms to expected aspects of the result. As c will not execute the task and compare results, there will likely be false positives when evaluating malicious responses.

The overall trust value of an edge node is summarised by a weighted mean over the expected values of these distributions:

$$\mathcal{T}(r,a) = \sum_{m \in M(a)} \varphi_{a,m} E[\mathcal{T}_m(r,a)] \tag{3}$$

where:

- $M(a)$ is the set of metrics that relate to application a.
- $0 \leq \varphi_{a,m} \leq 1$ is the weight that application a gives metric m. Applications use it to specify the relative importance of metrics, with $1 = \sum_{m \in M(a)} \varphi_{a,m}$.

If a stereotype $\mathcal{S}_m(r)$ is available for edge node r and metric m, then the trust model for that metric is adjusted before calculating the summarised trust. As these trust models are initialised as $\text{Beta}(1,1)$, 1 is subtracted from α and β.

$$\mathcal{T}'_m(r) = \begin{cases} \mathcal{T}_m(r).\alpha - 1 + \mathcal{S}_m(r).\alpha \\ \mathcal{T}_m(r).\beta - 1 + \mathcal{S}_m(r).\beta \end{cases} \tag{4}$$

The individual distributions are updated as per Algorithm 1, where $f^{\text{opinion}}_{a,m}$ is an application a and metric-specific m function that evaluates the *opinion* IoT device c has about a situation and interaction. The situation details which task was submitted and the interaction contains information about the last interaction with the edge node. For example, a situation may be "Request route from a to b" and the interaction may be "r timed out returning a response".

To choose which edge node to submit a task to, that edge node must support the application that originated the task and also have a sufficiently high trust value. For this example model we implemented a banded approach, where a sufficiently high trust value is any trust value within some distance from the maximum trust value. The chosen edge node is selected randomly from the edge nodes that meet this criteria.

This trust model does not utilise the reputation information disseminated. However, we have included it to demonstrate the cost

Algorithm 1 Update state based on a situation and interaction

▷ a is an application, s is a situation, i is an interaction
function UPDATE(a, s, i)
  for $m \in M(a)$ **do**
    if RELEVANTINTERACTION(a, s, i, m) **then**
      $o \leftarrow f_{a,m}^{\text{opinion}}(s, i)$
      if o = Successful **then**
        $\mathcal{T}_m(e, a).\alpha \leftarrow \mathcal{T}_m(e, a).\alpha + 1$
      else
        $\mathcal{T}_m(e, a).\beta \leftarrow \mathcal{T}_m(e, a).\beta + 1$

$\varphi_{a,m}$	sub	res	corr
Environment Monitoring	1	0	0
Routing	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$

Table 3: Application per-metric weights

that trust models that do utilise it incur, as this system architecture is intended to be trust model agnostic.

6.2 Example Applications

To illustrate the operation of this system we implement two example applications: (i) environment monitoring and (ii) vehicle routing. The environment monitoring application generated a task every 1 min, which involves sending sensor data to an edge node. The routing application generated a task every 2 to 3 min containing source and destination coordinates and expected to receive a route response within 2 min. The routing application performs a correctness check of a task result by checking that the source and destination are the first and last items in the provided path. The trust model weights for these two applications are shown in Table 3.

7 RESULTS

We now present results analysing three key aspects of this system: (i) the RAM and Flash costs, which define the device specifications of IoT devices, (ii) the cost of cryptographic operations, highlighting the trade-offs made, and (iii) the runtime performance of the system with example applications.

7.1 RAM and Flash Usage

The RAM and flash usage of the implementation (shown in Table 4) was generated using `nm` on the compiled binary for IoT devices. This only shows the cost of defined symbols such as static variables and functions, it does not include strings. Symbols have been classified into categories to identify where the RAM and flash costs are incurred. The implementation is limited by the RAM of the IoT hardware. This is because dynamic memory allocation is typically avoided with embedded systems, as long-term use can lead to memory fragmentation which prevents future allocation requests succeeding. So instead fixed-size buffers are chosen at compile time.

In our implementation 64% of the RAM utilisation comes from the buffers required to implement network access (contiki-ng/net), certificate storage and digital signatures (system/crypto), and the trust model (system/trust).

	Flash		RAM	
Category	(B)	(%)	(B)	(%)
applications/monitoring	1 388	1.2	353	1.2
applications/routing	3 868	3.3	474	1.6
contiki-ng	7 280	6.2	846	2.9
contiki-ng/cc2538	14 556	12.4	2 356	8.0
contiki-ng/coap	8 556	7.3	2 388	8.1
contiki-ng/net	26 824	22.9	8 232	27.8
contiki-ng/oscore	5 512	4.7	1 010	3.4
newlib	26 415	22.6	2 534	8.6
system/common	3 188	2.7	37	0.1
system/crypto	6 210	5.3	5 173	17.5
system/mqtt-over-coap	1 490	1.3	503	1.7
system/trust	11 846	10.1	5 659	19.1
Total Used	117 133	100	29 565	100
Total Available	524 288		32 768	

Table 4: IoT device RAM and flash usage

Operation	Mean Cost	Units
SHA256	637 ± 11.6	ns/B
ECC Sign (sepc256r1)	360 ± 0.04	ms
ECC Verify (sepc256r1)	711 ± 0.03	ms
ECDH	344 ± 0.02	ms
AES-CCM-16-64-128 Encrypt	0.94 ± 0.01	µs/B
AES-CCM-16-64-128 Decrypt	1.01 ± 0.01	µs/B

Table 5: Performance of Cryptographic Operations

7.2 Cryptographic Operations Cost

In this section, we perform profiling of the relevant cryptographic operation costs on the Zolertia RE-Mote to understand the trade-offs of using different message protection approaches. The hardware on which these tests were performed has a clock with 32768 ticks per second, which means timers have a resolution of 30.5 µs (3 s.f.). The average costs are shown in Table 5 with 95% confidence intervals.

Results for SHA256 and ECC operations were gathered by generating a random plaintext with a random length from 1 to 1 024 B and then signing and verifying that plaintext. As SHA256 is performed as part of the sign operation, each sign and verify operates on a constant number of bytes, so the results are not shown per byte. Results for AES-CCM encryption and decryption were gathered by generating a random plaintext with a random length from 1 to 1 024 B, a random 35 B of additional authenticated data (maximum supported by OSCORE), a random 16 B key, and a random 13 B nonce. The plaintext was encrypted and a 8 B authentication tag was generated which was then decrypted and authenticated.

These results highlight the cost difference between AES-CCM operations and the ECC operations on this IoT hardware. Performing an AES-CCM operation on a 100 B message is three orders of magnitude faster than an ECC operation. This performance difference is why ECC operations are only used to derive a shared secret for OSCORE and to disseminate signed reputation information, whereas AES-CCM is used to protect all other messages.

7.3 Task Submission

We performed a deployment with three IoT devices (wsn3, wsn4, wsn5) and two edge nodes (rr2 and rr6) where both edge nodes perform the monitoring application correctly, but rr6 always performs incorrectly for the routing application. Incorrect behaviour is randomly chosen from: (i) not sending a response, (ii) sending an invalid response claiming it is correct, or (iii) sending a response indicating a failure. The system was run for long enough for trust values to begin to converge. Results are shown in Figure 8 where (i) the IoT devices evaluate their trust that the edge node will execute the task (lines) and (ii) the number of tasks a IoT device submits to an edge node over a time period (bars).

Figure 8a shows results for the monitoring application, where trust values start high (due to stereotypes) and remain high. There are instances where trust values decrease, which may be due to transient failures such as edge nodes failing to acknowledge a task submission. The tasks submitted by the three IoT devices are distributed across the two edge nodes as no edge node has a sufficiently low trust value for them to be excluded from task submission.

Figure 8b shows results for the routing application. The trust values begin at a high value due to stereotype information, however, the trust in the two edge nodes quickly diverge due to the differing behaviour. While rr6 has a trust value that is still within the maximum distance from the highest trust value, it can be chosen to execute tasks (as described in Section 6.1). However, after the trust value becomes sufficiently low, rr6 is excluded from being selected and the IoT devices only send tasks to the well behaving rr2.

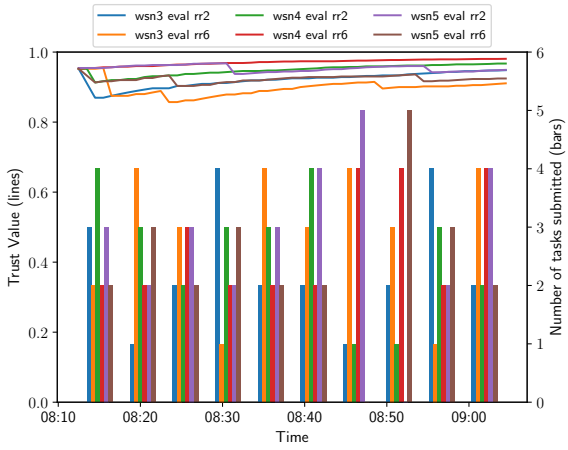
7.4 Message cost

Results showing the number of bytes transmitted and received are shown in Figure 9 and Figure 10 respectively, where messages have been grouped into 5 min windows. The results for IoT devices wsn4 and wsn5 are omitted as they show a similar pattern to wsn3.

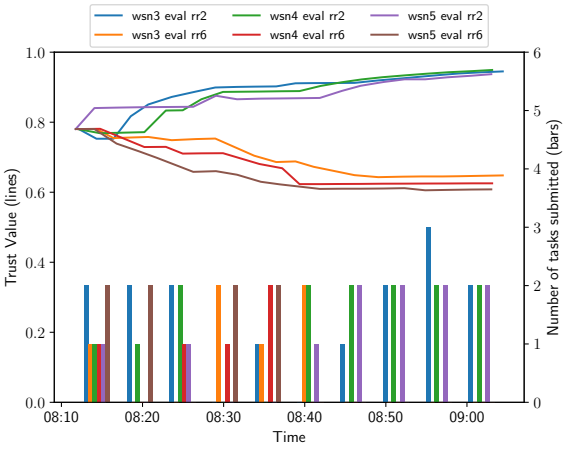
The messages have been categorised where possible. Due to issues with analysis tools not all OSCORE contexts can be decrypted, so valid messages will appear as "oscore". Not all 6LoWPAN fragments could be reassembled, so are shown as "6lowpan-fragment". Packets marked as "oscore" were for a variety of purposes including the two applications and potentially other categories where messages could not be decrypted. "trust-dissem" packets are intentionally not protected with OSCORE, as they need to be signed and not encrypted. The implementation currently manually includes a digital signature, which will be the case until Group OSCORE is supported (as will be described in Section 8.3).

Comparing the two edge nodes rr2 (always good) and rr6 (always bad) shows why an edge node may choose to perform maliciously, as there is a greatly decreased cost in delivering the functionality. Edge node 2 has a higher number of messages sent and received than rr6 because by performing correctly it needs to deliver the result of the application. For the routing application task, this means that a result of 7 600 B needs to be delivered back to the IoT device which involves sending 39 CoAP messages and receiving the same number of acknowledgements in the best-case.

There is a decrease in the number of messages sent and received on rr6 and the wsn3 at 8:30 because all three IoT devices choose to use rr6 for all tasks in this period (as shown in Figure 8b). The same



(a) Monitoring

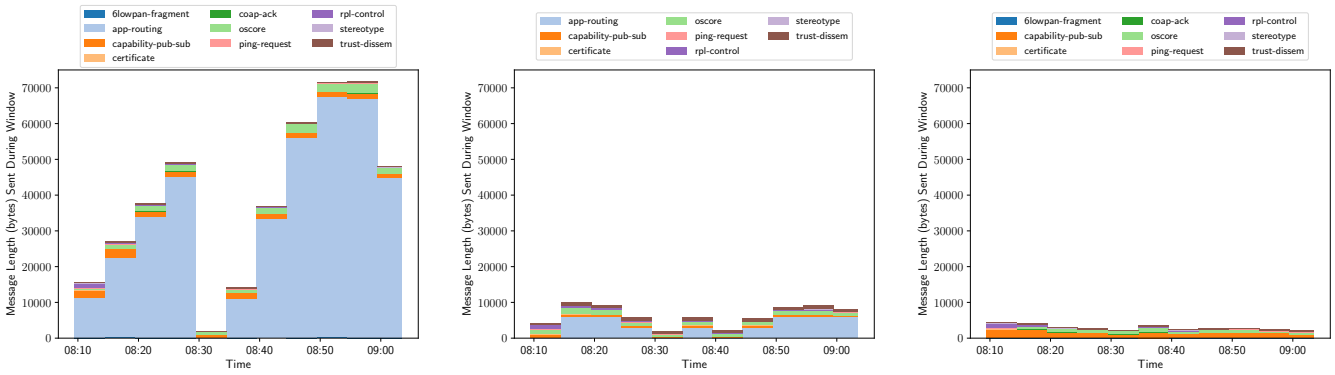


(b) Routing

Figure 8: Trust values over time and nodes selected to execute tasks for two different applications

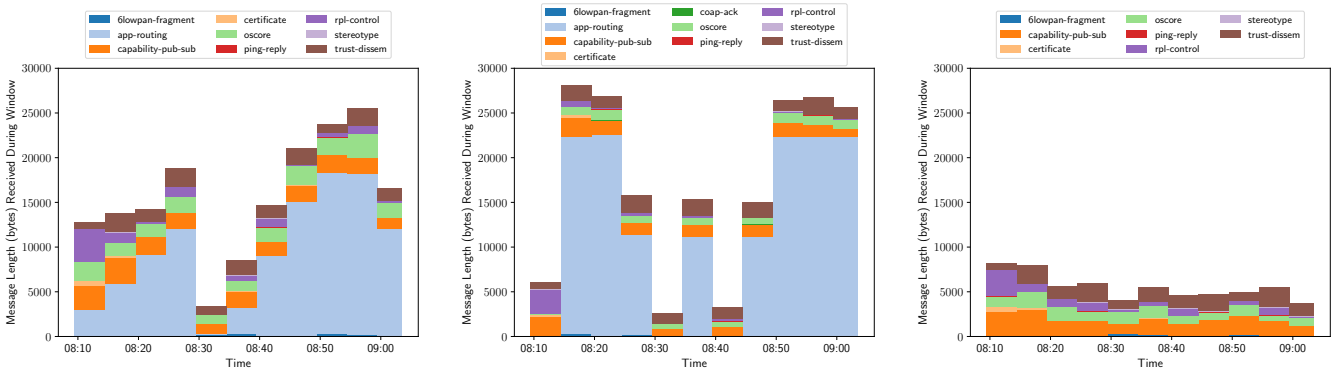
pattern appears at 8:40, but only for wsn3 as the other two edge nodes send routing tasks to rr2. As rr6 behaves well for the monitoring application, IoT devices do not stop submitting monitoring tasks to it. This is why rr6 continues to receive "oscore" messages even after IoT devices stop sending routing tasks to it at 8:45.

The routing application has the largest proportion of bytes sent (>52%) and received (>72%) for the three IoT devices. Trust dissemination and subscribing to capabilities are also expensive, costing 10–13% of bytes sent and 7–10% of bytes received. Packets that our analysis tools could not process (marked as "oscore") took up 15–17% of bytes transmitted and 4–5% of bytes received. This indicates a worst case 50% overhead in transmitted bytes and 28% overhead in received bytes to facilitate trust-based task offloading. In reality these overheads will be lower, as some application packets were categorised as "oscore". The overhead will differ depending on the frequency of reputation and capability dissemination, frequency of tasks, and the payload sizes of tasks and their responses. Deployments would need to adjust the rate at which reputation and capability information is disseminated based on application needs.



(a) rr2 (b) wsn3 (c) rr6

Figure 9: Length of messages sent over 5 min windows



(a) rr2 (b) wsn3 (c) rr6

Figure 10: Length of messages received over 5 min windows

8 DISCUSSION

A number of design decisions were made due to limitations in the software libraries available or to simplify implementation aspects. We now discuss two considerations with using this architecture.

8.1 Use of MQTT Retain Flag

An optimisation to reduce the cost of announce and capability messages would be to use the MQTT retain flag. This means when a message is published, that message is saved and delivered to nodes that subscribe in the future. However, a retained message may contain outdated information (e.g., an edge crashed or a capability becomes unavailable). Therefore, in this work we have chosen to be conservative and have edges periodically publish information.

8.2 Forward Secrecy via ECHDE

We have proposed pre-deploying a public/private key pair to each IoT device lasting the device's lifetime. This simplifies key management, reducing the cost of managing and exchanging keys. A downside is that using ECDH to derive a shared secret once (i.e., used for the lifetime of the devices) does not provide forward-secrecy. If required, then future standards (such as EDHOC [29]) that facilitate ECDHE will be necessary to setup the OSCORE context.

8.3 Implementation Limitations

Due to the use of recently published standards there are some features of the libraries being depended upon that are not yet implemented. Firstly, the implementation of OSCORE for Contiki-NG does not yet implement RFC 8613 Appendix B.1 [28], which means that when nodes reset they will be unable to restart communication via OSCORE. Secondly, the Group OSCORE draft standard [33] does not yet have a working implementation so signed and unencrypted trust packets cannot be protected by Group OSCORE. To work around this, the payload is signed, however, this will not protect the CoAP headers that Group OSCORE protects.

9 CONCLUSIONS

We have presented a system for facilitating trust-based task offloading for multiple applications on IoT devices. Through two case studies and an example trust model, we show how a suitable edge node is selected as the destination for offloading. In our example, it took 6 rounds of task submissions over under 30 min for a permanently bad node to be excluded and at worst a 50% overhead in transmitted bytes and 28% overhead in received bytes. The implementation applies recent IoT security standards such as OSCORE and will make use of future standards such as Group OSCORE to

provide security guarantees. We have also developed the building blocks to enable the use of more complex trust models that involve the use of disseminated reputation information and stereotypes. For future work, we plan to perform a threat modelling of this system to guide the development of resilient trust models.

DATA STATEMENT

The software used to generate these results can be found at https://github.com/MBradbury/iot-trust-task-alloc. The data gathered and presented in this paper can be found at [8].

ACKNOWLEDGMENTS

This work was supported by the PETRAS National Centre of Excellence for IoT Systems Cybersecurity [EPSRC Grant EP/S035362/1]. The authors would like to thank Martin Gunnarsson and Krzysztof Mateusz Malarski at RISE for assistance with their WIP OSCORE implementation.

REFERENCES

[1] Roger Alexander, Anders Brandt, J. P. Vasseur, Jonathan Hui, Kris Pister, Pascal Thubert, P. Levis, Rene Struik, Richard Kelsey, and Tim Winter. 2012. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550. https://doi.org/10.17487/RFC6550

[2] J. G. Alfaro, M. Barbeau, and E. Kranakis. 2009. Secure Localization of Nodes in Wireless Sensor Networks with Limited Number of Truth Tellers. In *Seventh Annual Communication Networks and Services Research Conference*. IEEE, Moncton, NB, Canada, 86–93. https://doi.org/10.1109/CNSR.2009.23

[3] Christian Amsüss and Maciej Wasilak. 2013–. aiocoap: Python CoAP Library. http://github.com/chrysn/aiocoap/

[4] E. Baccelli, C. Gündoğan, O. Hahm, P. Kietzmann, M. S. Lenders, H. Petersen, K. Schleiser, T. C. Schmidt, and M. Wählisch. 2018. RIOT: An Open Source Operating System for Low-End Embedded Devices in the IoT. *IEEE Internet of Things Journal* 5, 6 (Dec 2018), 4428–4440. https://doi.org/10.1109/JIOT.2018.2815038

[5] Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta (Eds.). 2019. *MQTT Version 5.0.* OASIS Standard, Burlington, MA, USA. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

[6] F. Bao, I. Chen, M. Chang, and J. Cho. 2012. Hierarchical Trust Management for Wireless Sensor Networks and its Applications to Trust-Based Routing and Intrusion Detection. *IEEE Transactions on Network and Service Management* 9, 2 (June 2012), 169–183. https://doi.org/10.1109/TCOMM.2012.031912.110179

[7] Carsten Bormann and Paul E. Hoffman. 2013. Concise Binary Object Representation (CBOR). RFC 7049. https://doi.org/10.17487/RFC7049

[8] Matthew Bradbury, Arshad Jhumka, and Tim Watson. 2020. *Dataset for: Trust Assessment in 32 KiB of RAM: Multi-application Trust-based Task Offloading for Resource-constrained IoT Nodes.* https://doi.org/10.5281/zenodo.4312801

[9] I. Chen, F. Bao, M. Chang, and J. Cho. 2014. Dynamic Trust Management for Delay Tolerant Networks and Its Application to Secure Routing. *IEEE Transactions on Parallel and Distributed Systems* 25, 5 (2014), 1200–1210.

[10] M. Chen and Y. Hao. 2018. Task Offloading for Mobile Edge Computing in Software Defined Ultra-Dense Network. *IEEE Journal on Selected Areas in Communications* 36, 3 (2018), 587–597.

[11] A. Dunkels, J. Eriksson, N. Finne, F. Österlind, N. Tsiftes, J. Abeillé, and M. Durvy. 2012. Low-power IPv6 for the Internet of Things. In *9th International Conference on Networked Sensing (INSS)*. IEEE, Antwerp, Belgium, 1–6.

[12] Adam Dunkels, Björn Grönvall, and Thiemo Voigt. 2004. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *29th Annual IEEE International Conference on Local Computer Networks*. IEEE, Florida, USA, 455–462. https://doi.org/10.1109/LCN.2004.38

[13] Adam Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. 2006. Protothreads: Simplifying Event-driven Programming of Memory-constrained Embedded Systems. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems* (Boulder, Colorado, USA) *(SenSys '06)*. ACM, New York, NY, USA, 29–42. https://doi.org/10.1145/1182807.1182811

[14] Ehab ElSalamouny, Vladimiro Sassone, and Mogens Nielsen. 2010. HMM-Based Trust Model. In *Formal Aspects in Security and Trust*, Pierpaolo Degano and Joshua D. Guttman (Eds.). Springer, Berlin, Heidelberg, 21–35.

[15] A. Elsts, X. Fafoutis, P. Woznowski, E. Tonkin, G. Oikonomou, R. Piechocki, and I. Craddock. 2018. Enabling Healthcare in Smart Homes: The SPHERE IoT Network Infrastructure. *IEEE Communications Magazine* 56, 12 (2018), 164–170.

[16] M. S. Farooq, S. Riaz, A. Abid, K. Abid, and M. A. Naeem. 2019. A Survey on the Role of IoT in Agriculture for the Implementation of Smart Farming. *IEEE Access* 7 (2019), 156237–156271.

[17] Paul Fiterau-Brostean, Bengt Jonsson, Robert Merget, Joeri de Ruiter, Konstantinos Sagonas, and Juraj Somorovsky. 2020. Analysis of DTLS Implementations Using Protocol State Fuzzing. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Boston, MA, 2523–2540.

[18] C. Gündoğan, C. Amsüss, T. C. Schmidt, and M. Wählisch. 2020. IoT Content Object Security with OSCORE and NDN: A First Experimental Comparison. In *IFIP Networking Conference (Networking)*. IEEE, Paris, France, 19–27.

[19] Martin Gunnarsson, Joakim Brorsson, Francesca Palombini, Ludwig Seitz, and Marco Tiloca. 2020. Evaluating the Performance of the OSCORE Security Protocol in Constrained IoT Environments. *Internet of Things* 13 (2020), 100333. https://doi.org/10.1016/j.iot.2020.100333

[20] Joel Höglund, Samuel Lindemer, Martin Furuhed, and Shahid Raza. 2020. PKI4IoT: Towards public key infrastructure for the Internet of Things. *Computers & Security* 89 (2020), 101658. https://doi.org/10.1016/j.cose.2019.101658

[21] Audun Jøsang and Roslan Ismail. 2002. The Beta Reputation System. In *15th Bled Electronic Commerce Conference*. University of Maribor Press, Bled, Slovenia, 14 pages.

[22] L. U. Khan, I. Yaqoob, N. H. Tran, S. M. A. Kazmi, T. N. Dang, and C. S. Hong. 2020. Edge Computing Enabled Smart Cities: A Comprehensive Survey. *IEEE Internet of Things Journal* 7, 10 (2020), 10200–10232. https://doi.org/10.1109/JIOT.2020.2987070

[23] Arjen K. Lenstra and Eric R. Verheul. 2001. Selecting Cryptographic Key Sizes. *Journal of Cryptology* 14, 4 (2001), 255–293. https://doi.org/10.1007/s00145-001-0009-4

[24] Roger A. Light. 2017. Mosquitto: server and client implementation of the MQTT protocol. *Journal of Open Source Software* 2, 13 (2017), 265. https://doi.org/10.21105/joss.00265

[25] P. Mach and Z. Becvar. 2017. Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Communications Surveys Tutorials* 19, 3 (2017), 1628–1656.

[26] S. Pan, Z. Zhang, Z. Zhang, and D. Zeng. 2019. Dependency-Aware Computation Offloading in Mobile Edge Computing: A Reinforcement Learning Approach. *IEEE Access* 7 (2019), 134742–134753.

[27] Carlos Segarra, Ricard Delgado-Gonzalo, and Valerio Schiavoni. 2020. MQT-TZ: Hardening IoT Brokers Using ARM TrustZone. In *39th International Symposium on Reliable Distributed Systems (SRDS 2020)*. IEEE, Shanghai, China, 256–265. arXiv:2007.12442 [cs.CR]

[28] Göran Selander, John Mattsson, Francesca Palombini, and Ludwig Seitz. 2019. Object Security for Constrained RESTful Environments (OSCORE). RFC 8613. https://doi.org/10.17487/RFC8613

[29] Göran Selander, John Preuß Mattsson, and Francesca Palombini. 2020. *Ephemeral Diffie-Hellman Over COSE (EDHOC).* Internet-Draft draft-ietf-lake-edhoc-01. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-lake-edhoc-01 Work in Progress.

[30] Ali Shakarami, Ali Shahidinejad, and Mostafa Ghobaei-Arani. 2020. A review on the computation offloading approaches in mobile edge computing: A game-theoretic perspective. *Software: Practice and Experience* 50, 9 (2020), 1719–1759. https://doi.org/10.1002/spe.2839

[31] Zach Shelby, Klaus Hartke, and Carsten Bormann. 2014. The Constrained Application Protocol (CoAP). RFC 7252. https://doi.org/10.17487/RFC7252

[32] Phillip Taylor, Lina Barakat, Simon Miles, and Nathan Griffiths. 2018. Reputation assessment: a review and unifying abstraction. *The Knowledge Engineering Review* 33 (2018), e6. https://doi.org/10.1017/S0269888918000097

[33] Marco Tiloca, Göran Selander, Francesca Palombini, and Jiye Park. 2020. *Group OSCORE - Secure Group Communication for CoAP.* Internet-Draft draft-ietf-core-oscore-groupcomm-09. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-groupcomm-09 Work in Progress.

[34] B. Wang, M. Li, X. Jin, and C. Guo. 2020. A Reliable IoT Edge Computing Trust Management Mechanism for Smart Cities. *IEEE Access* 8 (2020), 46373–46399. https://doi.org/10.1109/ACCESS.2020.2979022

[35] Jun Wang, Xi Xiong, and Peng Liu. 2015. Between Mutual Trust and Mutual Distrust: Practical Fine-Grained Privilege Separation in Multithreaded Applications. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference* (Santa Clara, CA) *(USENIX ATC '15)*. USENIX Association, USA, 361–373.

[36] Thomas Watteyne, Maria Rita Palattella, and Luigi Alfredo Grieco. 2015. Using IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the Internet of Things (IoT): Problem Statement. RFC 7554. https://doi.org/10.17487/RFC7554

[37] D. Wu, G. Shen, Z. Huang, Y. Cao, and T. Du. 2019. A Trust-Aware Task Offloading Framework in Mobile Edge Computing. *IEEE Access* 7 (2019), 150105–150119.

[38] H. Yu, Z. Shen, C. Leung, C. Miao, and V. R. Lesser. 2013. A Survey of Multi-Agent Trust Management Systems. *IEEE Access* 1 (2013), 35–50.

[39] Zolertia. 2016. *Zolertia RE-Mote Revision B Internet of Things hardware development platform, for 2.4-GHz and 863-950MHz IEEE 802.15.4, 6LoWPAN and ZigBee® Applications.* Datasheet ZOL-RM0x-B. Barcelona, Spain. V1.0.0.