Edge-centric Resource Allocation for Heterogenous IoT Applications using a CoAP-based Broker

Abstract: The Edge/Fog computing paradigm has been recently advocated for future IoT systems to cope with the capacity and latency constraints of conventional cloud-centric IoT architectures. Fog nodes are not only needed to offload computing tasks from the centralised cloud but also to provide IoT applications with management services that facilitate deployment and improve Quality of Service. Indeed, in large-scale IoT deployments, it is expected that a large number of applications access the same resources (e.g. a sensor or an actuator), most likely hosted on constrained devices. Moreover, applications can have highly heterogeneous QoS requirements, e.g., regarding real-time constraints or frequency with which they desire to receive notifications from the monitored resources. However, IoT applications may be unable to autonomously adapt their access patterns for IoT resources to network dynamics and bandwidth limitations. To address these issues, in this work we design a fog-based broker that regulates the access to IoT resources transparently and effectively. Specifically, we develop an optimisation framework to determine the notification periods that maximise the applications' QoS satisfaction under network-related constraints. Then, we also propose practical algorithms that leverage measurements of the degree of reliability of application transmissions to infer the congestion level of the IoT resources, and adapt the notification periods accordingly. We have developed a software prototype of our broker by exploiting the standard features of the CoAP protocol. Then, we have validated the proposed solution through simulations and real experiments in an IoT testbed. Results show that, as the application demands increase, the proposed approach guarantees better QoS satisfaction, higher throughput and improved energy efficiency than a conventional CoAP proxy. Moreover, the efficacy of the optimal solution heavily depends on accurate estimates of the network capacity, which may be difficult to obtain in real-world IoT deployments.

Keywords: Internet of Things, fog computing, resource brokering, CoAP, emulation, prototype.

1 Introduction

During the last few years, we have witnessed the rapid development and deployment of the Internet of Things (IoT). The most recent projections estimate that 20 billion smart things (excluding smartphones, tablets, and PCs) will be connected to the Internet by 2020 using IoT technologies [1]. Most of these IoT devices are low-cost and resource-constrained devices, endowed with sensing and actuating capabilities, that collect and exchange data related to physical objects, processes and environments. In many existing IoT deployments, a common approach to cope with the limited capabilities of IoT devices regarding computing, memory and energy, is to send all the produced IoT data to a centralised cloud infrastructure, where storage, management, processing and decision-making services are implemented in large data centres [2]. However, it is has become widely recognised that in a cloud-based IoT architecture network bandwidth and communication latency can be severe bottlenecks [3]. Indeed, as the number of IoT devices continues to increase, the volume and velocity of

IoT traffic are expected to grow exponentially [4]. Then, it is very challenging for both existing and future core networks to support the exponential increase of traffic demand due to the exchange of IoT data. Moreover, many IoT applications (e.g., industrial control systems), have stringent requirements regarding time responsiveness and Quality of Service (QoS), which are difficult to satisfy with a cloud-centric model. In the case of intermittent network connectivity (e.g., IoT devices may often be offline), it is also difficult to provide uninterrupted cloud services.

To address the above issues, many research efforts have proposed the edge computing paradigm for future IoT systems [5, 6, 7]. According to this concept, the functions of a centralised cloud are transferred to edge devices of networks, close to end users and things. Different approaches have been proposed to realise edge computing platforms in the IoT domain, but one of the most mature and flexible solutions is fog computing [8]. In a fog-based IoT architecture, resources and services of data management, storage, computing, control and even networking are distributed over many heterogeneous devices, called *fog nodes*, which are distributed from the cloud to the things [9]. Fog integrates with cloud and helps in coping with network bandwidth constraints by reducing the amount of data that needs to be sent to the cloud, and ensuring low latency and location-awareness to applications. However, it is important to emphasise that fog nodes are only used to offload computing from the core cloud, but they are also expected to provide specific services to support heterogeneous QoS requirements of IoT applications. Specifically, large-scale IoT scenarios (e.g., smart cities) entail a large number of applications that concurrently access resources that are available at IoT sensors and actuators [10]. This situation can easily result in scalability issues and QoS degradation, as application messages can get lost due to network congestion and limited memory of IoT devices. Secondly, network congestion would cause an increase in communication and process delays, even for fog nodes. Moreover, applications may have different requirements on how frequently they want to access the same IoT resource or be notified of state changes for a resource. However, IoT applications are typically unaware of network-related constraints (e.g., capacity limitations), and they may be unable to adjust their patterns of IoT data collection to the current network state. Thus, a *resource brokering* service for IoT is needed to handle requests from external applications with the aim of providing load balancing, improving overall network efficiency and performance, and ensuring better QoS [11]. Note that many existing standard IoT architectures, such as ETSI oneM2M [12], advocates the used of so-called M2M gateways to handle application requests on behalf of IoT devices, which can also act as application/resource brokers.

In this work, we focus on the problem of designing a fog-based intermediary entity that can act as a broker for heterogenous IoT applications, regulating the access to IoT resources transparently and effectively. First of all, we formulate an optimisation framework that aims at maximising the overall *QoS satisfaction* of heterogeneous IoT applications that access the same IoT resource, while taking into account network-related constraints. Then, we discuss the limitations of an optimal approach that requires an *a priori* and accurate prediction of the capacity of network links and paths. This analysis leads us to the design of practical algorithms and solutions that can effectively work in the real case where the estimation of network-related constraints is difficult or too costly in terms of network resources. A key feature of our proposed solution is to directly measure the *transmission reliability* of application messages that convey resource state updates, and to leverage this metric to infer if application requests overload an IoT resource. As better explained in the following, we distinguish between localised congestion and network-wide congestion by introducing the concept of *congested routing subtree*, and we define specific rate adaptation mechanisms to

cope with the latter case. Finally, our proposed resource broker implements edge caching to further reduce traffic congestion and response time in the IoT network.

A key challenge in the IoT domain is the fragmentation of existing IoT architectures and platforms, while a root enabler for the IoT wide deployment would be the adoption of open and standardised architectures and protocols. Therefore, we have developed a software prototype of our broker by leveraging the capabilities of the CoAP protocol [13], a specialised web transfer protocol standardised by the IETF for use in resource-constrained devices. Specifically, our prototype exposes standard RESTful APIs to allow applications to discover and access IoT resources. Note that CoAP specification already defines basic proxying and caching capabilities, and our proposed broker is backwards-compatible with the legacy CoAP standard. We have carried out an extensive evaluation through emulation to assess the performance of our solution under controlled and reproducible conditions. As for performance benchmarks we used two alternative approaches: i) the solution of the general optimisation framework, and ii) a conventional CoAP proxy that is unaware of network-related constraints. Results showed that the three schemes perform similarly for low traffic demands, while our proposed approach provides better QoS satisfaction, higher throughput and improved energy efficiency for high traffic demands. Besides, we have validated our prototype implementation using a large-scale experimental IoT testbed. The experimental results confirm the efficacy of the proposed broker also in real-world deployments.

The rest of this paper is organised as follows. Section 2 introduces the reference scenario and the addressed problem. Section 3 describes the general optimisation framework for edge-centric resource brokering. Section 4 describes in detail a practical algorithm to support resource brokering. Section 5 overviews enabling technologies and introduces our prototype implementation. Evaluation results are presented in Section 6. Concluding remarks are finally reported in Section 8.

2 Problem Statement and System Model

As in [11], we assume that an IoT *domain* is any group of IoT devices that are managed by the same IoT *broker* (or M2M gateway following ETSI oneM2M nomenclature [12]). IoT devices are battery-powered devices with limited processing power and memory capacity, which host IoT *resources* to provide sensing and actuating services. The information generated by the IoT resources is typically delivered to a collection point (sink node) through multi-hop network paths. Without loss of generality, we assume that there is a single sink node in the IoT domain and it is deployed together with the border router that interconnects the IoT domain to the core Internet. The IoT broker is a logical entity that, following the edge/fog computing paradigm, is deployed on a server close to the IoT devices. By definition, an IoT broker implements a set of functionalities that are needed to allow client applications to access the IoT resources in the managed domain efficiently. Without loss of generality, we further assume that the broker is deployed in the border router as proximity between the IoT broker and the IoT devices brings many advantages compared to remote brokers, such as more efficient and low-latency communications. The key functionalities that are implemented in the IoT broker are the following: i) resource discovery and lookup to allow applications to locate resources that are hosted in the managed IoT devices, and ii) *proxying* to be able to act as an intermediary between the IoT resources and the applications. In a nutshell, a proxy can forward application requests and relay back responses on behalf of the IoT devices hosting the target resource. In this study, we envisage that the IoT broker

optimally manages the application requests to satisfy application requirements, while taking into account network-related constraints.

More formally, let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of n devices that form the IoT domain, and let s_0 denote the sink. Without loss of generality, we assume that each IoT device s_i host a single resource, say r_i, but the formulation could be easily extended to deal with multiple resources per IoT devices. Thus, in the following, we will use the notation s_i and r_i, and the term IoT node and IoT resource interchangeably. Now, let $A = \{a_1, a_2, \ldots, s_m\}$ be a set of m applications that want to access the IoT resources. We envisage that an application typically accesses a single resource, but each resource can be accessed by multiple applications, i.e., $m \geq n$. Therefore, it is also convenient to introduce the set $A_k \subset A$, defined as the set of applications that require to access the same resources r_k. To simplify the notation, in the following we use the subscript index i to refer to an IoT device s_i, the subscript index j to refer to an application a_j and the subscript index k to refer to a resource r_k.

Each application $j \in A$ requires to receive periodic notifications about the current status of the target resource. More precisely, we assume that each application is characterised by a desired range of notification rates, and the higher the notification rate, the higher the satisfaction of the application's QoS requirements. Formally, let $[\alpha_j, \beta_j]$ be the desired range of notification rates for application j and let $g_j(x)$ be the *QoS satisfaction index* associated to application j, which describes the degree of application's QoS satisfaction when it receives updated representation of the target resource with a rate x. Without loss of generality, $g_j(x)$ can be thought as a utility function [14, 15], i.e. an increasing, strictly concave, and continuously differentiable function over the set $x \in [\alpha_j, \beta_j]$, with $g_j(x) = 1$ for $x \geq \beta_j$ and $g_j(x) = 0$ for $x \leq \alpha_j$. In other words, if the broker provides a notification rate to application j greater than β_j, this does not increase the application's QoS satisfaction. The objective of the broker is to assign to each application the maximum feasible notification rate to maximise the total QoS satisfaction. To this aim, the broker collects the notification requirements for a given resource r_k from all the applications $j \in A_k$. Then, the broker periodically polls resource k with a *polling rate* x_k and *caches* the received response. Edge caching at the broker is beneficial to significantly reduce the network traffic in the IoT domain as a single response message can be used to serve multiple applications. Indeed, the broker shall immediately reply the cached response from r_k to application $j \in A_k$ if $x_k \leq \beta_j$, i.e., if the notification rate is within the desired range. Otherwise, the broker should wait to avoid overloading the application.

3 General Optimisation Model for Resource Brokering

In this section, we formulate a general optimisation problem to determine which polling rates should be allocated by the broker to each IoT resource to maximise the overall QoS satisfaction of the active applications, subject to constraints on physical network resources. We cast our problem in the ideal case where a perfect medium access control scheme regulates the access of the shared wireless medium under a simple protocol interference model. The reason of considering an ideal case is to provide a reference scenario and a benchmark to evaluate the proposed algorithms in a real case, in which it might be difficult to obtain reliable statistics about the link capacities (e.g. under time-varying channel and interference conditions).

Let x_k be a continuous decision variable indicating the probing rate that is used by the broker to collect status updates about resource $k \in S$. The following objective function

aims to maximise the overall satisfaction of admitted applications:

$$\max \sum_{k \in S} \sum_{j \in A_k} g_j(x_k) \tag{1}$$

$$x_k \geq \max_{j \in A_k} \{\alpha_j\} \qquad \forall k \in S \tag{2}$$

Constraint (2) enforces that all applications obtain the minimum desired notification period. In principle, the problem of application admission control, namely, to decide whether to allow a new incoming application to access an IoT resource should be considered jointly with the problem of allocating network resources to the admitted applications. For the sake of simplicity, we can assume that the network has sufficient capacity to satisfy the minimum QoS requirements of the client applications (i.e., constraint (2) is always feasible). It is also important to observe that in our architectural model, application requests are sent to the broker rather than directly to the IoT device that hosts the target resource, and the broker can aggregate application requests for the same IoT resource. Such aggregation capability limits the impact of admitting new applications for an already active IoT resource.

The most popular routing protocols for data collection in wireless sensor networks (e.g. CTP [16] and RPL [17]) are based on a routing tree. This means that the sink node is the root of a tree-based routing topology, and each node must forward all the traffic destined for the sink to its parent node in the tree. Thus, all the data generated in the network is collected at the sink node. Finally, according to the flow conservation rule, the traffic flow entering into a node i plus the data generated by node i must be equal to the traffic flow leaving that node. These routing conditions can be conveniently expressed using the following set of constraints:

$$\sum_{k \in S} x_k = \sum_{i \in S} f_i \tag{3}$$

$$x_i + \sum_{h \in S} f_{hi} = \sum_{h \in S} f_{ih} \qquad \forall i \in S \tag{4}$$

$$f_{ih} \leq K_{ih} \qquad \forall i, h \in S \tag{5}$$

where f_i and f_{ih} are variables representing the traffic flow from node i to the sink and node h, respectively. The constant K_{ih} is used to represents the children-parent relationship between nodes in a network path. More precisely, $K_{ih} = 0$ if h is not the parent node for node i, while is chosen to be large enough so that the constraint is always satisfied if h is the parent node for node i (e.g., higher than the maximum link capacity).

Finally, we have to take into account that the available link capacity is limited and must be shared among the IoT nodes that contend for the channel access. More formally, let C_{ih} be the maximum capacity of a link between a pair of nodes (i, h). According to the interference protocol model, given a transmission from node i to node h, none of the nodes in the interference area of i can be a receiver, and none of the nodes in the interference area of h can be a transmitter. In other words, all the nodes within the same interference area share the same transmission channel and therefore, the transmission time should be shared among them. Note that the interference area can depend complexly on several factors, including transmission powers, channel propagation and obstacles. However, under the interference protocol model [18], the interference area of a node is a disk around the node with a radius equal to the interference range. Moreover, node h cannot be a simultaneous receiver for another children node p different from node i. These conditions can be conveniently

expressed by enforcing that the sum of the fractions of total channel time used by each link (i, h) and its interfering links (according to the definition above) is equal or less than one. Formally, the following constraint must hold:

$$\frac{f_{ih}}{C_{ih}} + \sum_{\substack{p \in S \\ p \neq i}} \frac{f_{ph}}{C_{ph}} + \sum_{\substack{p,q \in S \\ d(q,i) \leq I_i}} \frac{f_{pq}}{C_{pq}} + \sum_{\substack{p,q \in S \\ d(p,h) \leq I_h}} \frac{f_{pq}}{C_{pq}} \leq 1 \qquad \forall i, h \in S \tag{6}$$

where $d(i, h)$ is the distance between i and h, and I_i is the interference range of i. Note that the second term in the above sum expresses the constraints that two children nodes of h can not transmit simultaneously.

4 Practical Algorithm for Resource Brokering

The general optimisation framework formulated in the previous section assumes perfect knowledge of both the channel propagation model and the channel access behaviour. While this approach provides a useful benchmark for optimal performance, its applicability to practical cases might be limited due to the hurdles of obtaining such information reliably, especially under time-varying channels and complex interference models. Thus, more practical algorithms and solutions may be needed. In the following, we present the design rationale behind our proposed broker. Then, we describe our practical algorithm for resource brokering if network-related constraints are unknown.

4.1 Design rationale

The broker manages the IoT resources deployed in an IoT domain and makes periodic decisions about the notification rates assigned to the applications that access the IoT resources. We assume that the decision cycle is fixed and equal to T seconds. The m-th decision point is simply given by the time instant $t_m = m \times T$ with $m = 0, 1, \ldots$. The broker's decisions are based on the congestion state of the IoT resource and the degree of transmission reliability for application messages. More precisely, during each cycle, the broker collects measurements about the *transaction loss ratio* (TLR) of each resource, defined as the fraction of transaction losses over a decision cycle, and it leverages these measurements to update the congestion state of IoT resources. Specifically, the transactions between an application and its target IoT resource follow a typical request/response model. For instance, an IoT device can host a temperature resource (sensor), and the resource responds to the application requests with the temperature reading. A transaction loss is defined as a request/response exchange in which the reply does not arrive at the broker before the notification deadline expires. A transaction can get lost due to several reasons. For instance, a burst of channel errors or buffer overflows can cause too many retransmissions at the MAC layer, and the request or response messages get lost. Moreover, network congestion increases the transmission delay of the response messages, which can arrive at the broker when the notification deadline is already expired.

It is quite straightforward to assume that if the TLR value is above a given threshold, the IoT resource should be regarded as congested. An essential characteristic of our proposed solution is that the broker takes into account not only localised congestion conditions but also network-wide congestion conditions to decide whether to increase, decrease or keep

constant the polling rate associated to a resource. Specifically, in a tree-based topology, each node is the root of a subtree. The larger the subtree of a node is, the more packets it has to forward to the sink. Thus, a congested root node of a subtree can also be an indication of traffic congestion generated within the sub-tree. As explained in the following sections, if a subtree is classified as congested the broker aggressively reduces the polling rates of congested resources to recover more rapidly a normal state. On the other hand, if the IoT resource is not congested the broker can tentatively increase the polling rate to test for available network bandwidth. It is important to point out that our broker gives higher priority to applications that have a low QoS satisfaction index to ensure a more balanced usage of the network capacity. Moreover, the adaptive tuning of polling rates is designed in such a way to allow more aggressive rate increases for applications with low QoS satisfaction than applications with already high QoS satisfaction. The broker design is detailed in the following sections.

4.2 Transaction loss ratio

Let us define with $\gamma_k(t_m)$ the TLR value for IoT resource r_k, measured in the m-th decision cycle between t_m and t_{m+1}. The decision cycle should be chosen long enough for observing a reasonable number of application transactions to collect reliable statistics for the instantaneous TLR parameter. To filter out the fluctuations in the measurements and to obtain a long-term forecast of the average TLR value, the broker leverages a classical exponentially weighted moving average approach. Specifically, at the end of the m-th decision cycle the *long-term* TLR value for resource r_k to be used in the $(m+1)$-th decision cycle, say $\hat{\gamma}_k(t_{m+1})$, is computed as follows:

$$\hat{\gamma}_k(t_{m+1}) = \alpha_\gamma \times \gamma_k(t_m) + (1 - \alpha_\gamma) \times \hat{\gamma}_k(t_m) \tag{7}$$

with $0 < \alpha_\gamma < 1$. However, the quick detection of congestion onset due to temporary conditions (e.g., large backoff times or buffer saturation) requires to examine also short-term behaviours. Therefore, the broker also computes the relative difference between consecutive measurements of instantaneous TLR values. Formally, we have that $\Delta\gamma_{k,m} = [\gamma_k(t_m) - \gamma_k(t_{m-1}]/\gamma_k(t_m)$. As explained in the following, the broker leverages both $\hat{\gamma}_k(t_m)$ and $\Delta\gamma_{k,m}$ to assess the degree of reliability and the resulting congestion state of a resource.

4.3 Congestion detection

Let us denote with $\Gamma_{k,m}$ the *reliability degree* of a resource r_k during the m-th decision cycle. The broker defines three possible values for the $\Gamma_{k,m}$ parameter: high reliability (`HR`), low reliability (`LR`), and transitional reliability (`TR`). The two main parameters for the reliability classification are: i) the TLR threshold (Γ_L) that dictates the start to the `LR` region, and ii) the TLR threshold (Γ_H) that determines the stop of the `HR` region. Then, the difference between the two thresholds is the *hysteresis margin* ($HM = \Gamma_L - \Gamma_H$), which defines the size of the transitional region for the TLR. For the sake of clarity, the pseudo-code of the function that estimates the $\Gamma_{k,m}$ parameter is provided in Algorithm 1. The selection of the Γ_L threshold is critical. If it is selected too high, the broker may fail to identify an unreliable resource. On the other hand, if it is too low, the broker may be too conservative and do not fully exploit the available network capacity. It is important to point out that packet losses are very likely in a constrained environment and a small number of transaction losses should be tolerated. In the evaluation section, we show that there is a clear

trade-off on the maximum allowed unreliability and the ability of the broker to fully utilise the network bandwidth. Similarly, if the hysteresis margin is too narrow, the broker may be too sensitive making frequent and unnecessary adjustments to the polling rates. To some extent, the hysteresis margin should be used to cope with the variability of the network and let the broker converge to a stable operating point.

As discussed above, temporary degradations of channel and network conditions play a key role in determining the reliability of application transactions. The onset of this sudden congestion can be detected more effectively by taking into account short-term behaviours rather than long-term averages. For this reason, the broker also considers the variations of TLR values when classifying the resource reliability. If there is a significant TLR decrease (i.e., $\Delta\gamma_{k,m} \leq -\beta_\gamma$), the broker conjectures that the IoT resource is entering a HR state (line 3). Conversely, if there is a noticeable TLR increase (i.e., $\Delta\gamma_{k,m} \geq \beta_\gamma$), the broker conjectures that the IoT resources is entering a LR state (line 2).

Algorithm 1: *Computation of* $\Gamma_{k,m}$.

1: ***function*** ComputeΓ*($\hat{\gamma}_k(t_m)$,$\Delta\gamma_{k,m}$)*

2: ***if*** $(\Gamma_{k,m} \geq \Gamma_L$ **OR** $\Delta\gamma_{k,m} \geq \beta_\gamma)$ ***then return*** LR*;*
3: ***else if*** $(\Gamma_{k,m} \leq \Gamma_H$ **OR** $\Delta\gamma_{k,m} \leq -\beta_\gamma)$ ***then return*** HR*;*
4: ***else return*** TR*;*
5: ***end function***

A straightforward approach to exploit the $\Gamma_{k,m}$ parameter is to assume that an IoT node s_i is *congested* if $\Gamma_{i,m} =$ LR, where r_i is the resource hosted at s_i. We remind that each IoT device hosts a single resource. Thus, the concept of resource and node can be used interchangeably when assessing congestion. However, the root cause of network congestion is not necessarily the node where unreliability manifests itself as each node also forwards to the data collection point traffic that is originated by other nodes in its subtree. Therefore, we also introduce the concept of *congested subtree*. We assume that a subtree is congested if the root of the subtree is congested and a non-negligible fraction of nodes that belongs to the subtree are also congested. For ease of presentation, let us introduce some useful notation. Let $\mathcal{ST}_i$ denote the set of nodes in the subtree rooted at node s_i. Moreover, let $\mathcal{C}_i$ denote the set of children nodes of s_i and let $p(i)$ the parent node of s_i. Finally, let x_k the polling rate allocated to resource r_k. Algorithm 2 presents the pseudo-code of the function that checks whether $\mathcal{ST}_i$ is congested or not. The function checks the reliability degree of all nodes in the subtree and maintains two counters. The first one counts all congested nodes (line 7). The second one counts the fraction of congested nodes that have a polling rate higher than the one assigned to the root of the subtree (line 8). The rationale of this design choice is to weight more congested nodes that use more network bandwidth than the root of the subtree, which has to forward all the traffic generated in the subtree. Then, ω_1 and ω_2 ($\omega_1 > \omega_2$) are the two parameters for fine-tuning the mechanism for subtree congestion detection (line 10). Note that a prerequisite for a subtree to be congested is that the root node is also congested (line 3).

Algorithm 2: *Detection of congested subtrees.*

1: ***function*** CheckCongestedST*($\mathcal{ST}_i$)*

2: $\Gamma_{i,m} \leftarrow \text{Compute}\Gamma\,(\hat{\gamma}_i(t_m), \Delta\gamma_{i,m})$
3: ***if*** $(\Gamma_{k,m} \neq \texttt{LR})$ ***then return*** `False`;
4: $numHL_1 \leftarrow 0$; $numHL_2 \leftarrow 0$
5: ***for all*** $k \in \mathcal{ST}_i$ ***do***
6: $\quad\Gamma_{k,m} \leftarrow \text{Compute}\Gamma\,(\hat{\gamma}_k(t_m), \Delta\gamma_{k,m})$
7: $\quad$***if*** $(\Gamma_{k,m} = \texttt{LR})$ ***then*** $numHL_1 \leftarrow numHL_1 + 1$;
8: $\quad\quad$***if*** $(x_k \geq x_i)$ ***then*** $numHL_2 \leftarrow numHL_2 + 1$;
9: ***end for***
10: ***if*** $(numHL_1 \geq \omega_1 \times |\mathcal{ST}_i|$ **OR** $numHL_2 \geq \omega_2 \times |\mathcal{ST}_i|)$ ***then return*** `True`;
11: ***else return*** `False`;
12: ***end function***

Finally, Algorithm 3 presents a variation of the classical Depth-First Search (DFS) algorithm [19] to recursively traverse the network tree while searching for congested subtrees. The output of the algorithm is the set $\mathcal{O}_1$, defined as the set of nodes (congested or not congested) that belongs to at least one congested subtree. Then, the set of nodes that do not belong to any congested subtree is simply given by $\mathcal{O}_2 = S \setminus \mathcal{O}_1$. Note that, differently from a classical DFS implementation, if a subtree is congested it is not necessary to visit the nodes in the subtree (line 7).

Algorithm 3: *Node classification.*

1: ***function*** DFS($\mathcal{ST}_i$)

2: $\quad$***if*** $i = 0$ ***then*** $\mathcal{O}_1 \leftarrow \emptyset$;
3: $\quad$*mark* s_i *as visited;*
4: $\quad$***for all*** $u \in \mathcal{C}_i$ ***do***
5: $\quad\quad$***if*** *(*u *is not marked as visited)* ***then***
6: $\quad\quad\quad$***if*** *(CheckCongestedST(*ST_u*))* ***then***
7: $\quad\quad\quad\quad\mathcal{O}_1 \leftarrow \mathcal{O}_1 \cup ST_u$;
8: $\quad\quad\quad$***else***
9: $\quad\quad\quad\quad$*recursively call DFS(*$\mathcal{ST}_u$*);*
10: $\quad$***end for***
11: ***end function***

4.4 Tuning of the polling rates

The last component of the broker is responsible for adjusting the polling rates between the broker and the IoT resources. As illustrated in Algorithm 4, first of all, the broker selects the set of resources (say $\mathcal{A}_{dec}$) whose polling rates should be decreased and the set of resources (say $\mathcal{A}_{inc}$) whose polling rates can be tentatively increased. Specifically, all the resources in $\mathcal{O}_1$ with low reliability are added to $\mathcal{A}_{dec}$ (line 3), while all the resources in $\mathcal{O}_2$ with high reliability are added to $\mathcal{A}_{inc}$ (line 7). Note that also a small fraction (say ω_3) of resources in $\mathcal{O}_1$ with high reliability are added to $\mathcal{A}_{inc}$ (line 4) to be tentatively tested for available network bandwidth and speed up the convergence to an optimal operating point. A key characteristic of our rate adjustment policy is that the list of resources in $\mathcal{A}_{inc}$ is sorted in ascending order based on the QoS satisfaction index (line 9). This ranking allows assigning a higher priority to applications that have lower QoS satisfaction, facilitating a fairer usage of network bandwidth and the maximisation of the total QoS satisfaction. Moreover, only

a fraction τ of all eligible resources in $\mathcal{A}_{inc}$ is actually tested for higher notification rates (line 10). The tuning of the τ parameter controls the aggressiveness of the broker in testing for available network bandwidth.

Algorithm 4: *Rate adjustment.*

Require: $\mathcal{O}_1, \mathcal{O}_2$

1: $\mathcal{A}_{inc} \leftarrow \emptyset$; $\mathcal{A}_{dec} \leftarrow \emptyset$
2: ***for all*** $i \in \mathcal{O}_1$ ***do***
3: ***if*** $(\Gamma_{i,m}) = \texttt{LR}$) ***then*** $\mathcal{A}_{dec} \leftarrow \mathcal{A}_{dec} \cup \{r_i\}$
4: ***if*** $(\Gamma_{i,m}) = \texttt{HR}$ **OR** $rand() \geq \omega_3$) ***then*** $\mathcal{A}_{inc} \leftarrow \mathcal{A}_{inc} \cup \{r_i\}$
5: ***end for***
6: ***for all*** $i \in \mathcal{O}_2$ ***do***
7: ***if*** $(\Gamma_{i,m}) = \texttt{HR}$) ***then*** $\mathcal{A}_{inc} \leftarrow \mathcal{A}_{inc} \cup \{r_i\}$
8: ***end for***

9: sort $\mathcal{A}_{inc}$ *based on QoS satisfaction index;*
10: remove the last $(1 - \tau)$ *percent of resources in* $\mathcal{A}_{inc}$*;*

11: ***for all*** $k \in \mathcal{A}_{dec}$ ***do***
12: $x_k \leftarrow computeNewRate(r_k, x_k, , \texttt{dec})$*;*
13: ***end for***
14: ***for all*** $k \in \mathcal{A}_{inc}$ ***do***
15: $x_k \leftarrow computeNewRate(r_k, x_k, \texttt{inc})$*;*
16: ***end for***

A final optimisation of the broker is to perform a more aggressive rate increase if the polling rate is low (i.e. the application's QoS satisfaction is low). This is obtain by partitioning the interval of possible polling rates in subintervals with decreasing length. More formally, let $a_0 = \max_{j \in A_k} \alpha_j$ and $b_0 = \max_{j \in A_k} \beta_j$. Then, the polling rate for resource r_k has to be a real number in range $[a_0, b_0]$. Let us divide this interval into v subintervals with decreasing length. A simple quadratic function can be used to build the subintervals. Specifically, in case of a rate increase a subinterval $d_i = [w_i, w_{i+1}]$ (with $i = 1, 2, \ldots, v$, $w_1 = a_0$ and $w_{v+1} = b_0$) has a length $(v-i)^2(b_0-a_0)/v^2$. Then, the function $computeNewRate(r_k, x_k, \texttt{inc})$ checks the subinterval within which the rate x_k falls, and randomly selects a new rate in the subsequent subinterval. In case of rate decrease, we follows a complementary approach as the broker should perform a more aggressive rate decrease if the polling rate is high. This can be easily achieved by dividing the interval $[a_0, b_0]$ into v subintervals with increasing length. Following the same quadratic approach, this is equivalent to assume that the length of subinterval i is equal to $(i-1)^2(b_0 - a_0)/v^2$.

5 Enabling Technologies and Prototype Implementation

Before detailing the implementation of the software prototype of our proposed broker, we provide a brief overview of the key CoAP features and technologies that we have relied upon.

5.1 CoAP basics

CoAP is a web transfer protocol specified by the IETF CoRE Working Group and optimised for resource-constrained devices [13]. CoAP follows RESTful programming principles and employs a request/response interaction model to access resources hosted by CoAP endpoints (EP). CoAP does not strictly differentiate between clients and servers, because a CoAP EP can generally play both roles. Each IoT resource is made remotely available through a Uniform Resource Identifier (URI). Moreover, an IoT resource can be described using a standard format, called the Link Format [20], which is based on classical Web Linking [21] but extended to include additional attributes that are relevant for IoT resources. Then, CoAP EPs access resources to retrieve or alter their state by sending requests messages that convey one of the four canonical REST commands: GET, PUT, POST and DELETE. To some extent, CoAP is regarded as a lightweight version of HTTP. However, CoAP differs from HTTP in many aspects as it also implements additional services, like *link binding*, or entities, like the *Resource Directory* (RD) and the *reverse-proxy*, which we have exploited for implementing our resource broker.

5.1.1 Proxying

CoAP allows intermediary CoAP EPs, called proxies, to sit in between CoAP clients and CoAP servers and perform tasks on behalf of the CoAP clients. CoAP proxies fall into two categories: forward-proxy and reverse-proxy. With the former, a client directly selects the proxy, while a reverse-proxy is transparent to clients and it is used primarily to provide a common access point to multiple CoAP servers and to cope with intermittently-connected CoAP servers. Due to its feature, a reverse-proxy could be leveraged to implement a resource broker in a fog node since: (i) it can offer internal resources of the IoT domain as if they were its own resources (using namespace translation); and (ii) perform caching of past response to satisfy future CoAP requests.

5.1.2 RD

Legacy CoAP already provides a basic service for resource discovery by defining a non-physical resource, called */.well-known/core*, that a CoAP EP must expose to allow other CoAP EPs to retrieve its list of hosted resources. The RD is a more sophisticated approach for resource discovery under specification in the CORE WG [22], which is based on a resource directory, namely a centralised repository of registrations describing resources hosted on other CoAP EPs. Then, the RD exposes a set of standard REST interfaces which allow CoAP EPs to register and maintain their resource descriptors, as well as to lookup resource descriptors from the RD. Note that the RD can also provide additional information about a target resource, such as the proxy server that can be used to access it. Another option is for the reverse-proxy to directly implement an RD for the CoAP servers on behalf of which it satisfies the requests. As better explained in the following, our broker prototype follows this latter architectural model.

5.1.3 Link Binding

Link Binding is a new concept under specification in CoAP, which defines the rules to create a dynamic link between a source resource and a destination resource so that the state of the two resources can be synchronised [23]. The simplest form of binding is the

`Polling` method, which consists of a destination EP that periodically sends GET requests to a source EP, copying the content of the response message in the destination resource. The `Push` binding method is complementary to the `Polling` method as it is the source node to send PUT requests to the destination resource when a specified binding condition is met. Finally, the `Observe` method is analogous to a publish/subscribe model as notification messages from the source resource (the producer) are simply copied to the destination resource (the consumer). Note that the creation of the observation relationship requires the CoAP Observation mechanism [24]. However, the `Observe` method allows specifying some observation parameters (e.g., maximum notification frequency) or conditional rules to drive the generation of state updates (e.g. threshold levels on resource value). It is important to point out that most constrained devices may be unable to support CoAP observing, as a CoAP observer typically results in high memory usage.

5.2 Prototype implementation in detail

Our broker prototype is built using the Californium library [25], a popular Java-based open-source CoAP implementation. The Californium framework already includes modules that implement the reverse-proxy and the RD entities. Our software implementation of the proposed broker extends the RD module that is included in Californium by adding a customised reverse-proxy and integrating a new module, called the *brokering engine*. The RD module exposes REST APIs that allows CoAP clients to register their hosted resources. The registration message contains the EP name (e.g., *ep=node1*) and a "base" URI (optional), namely the path to resolve any relative references given in the registration (e.g., *</sensors/temp>*). If the base URI is not provided, the RD generates the reference path using the IP address and protocol port of the hosting node. If the resource registration is successful the RD generates a unique identifier (e.g. */rd/4521*) for the location of the associated resource descriptor in its local database. This location is returned to the registrant EP in the response message and must be used by the EP for any further operation on the registered resources (e.g., update and removal).

The RD exposes a REST API to allow other CoAP EPs to discover the resources registered with the RD. In the existing implementation, usually, output of a lookup operation on the RD is a list of links equal to the ones submitted to the RD (special cases are not considered in this work, and the interested reader is referred to the draft RD specification [22]). However, our prototype modifies this normal behaviour to ensure resource "shadowing", i.e. to force the client applications to access the registered resources only through the Proxy component of the broker. Specifically, the reverse-proxy performs namespace translation of the registered resources and creates equivalent *virtual* URIs for the resources by concatenating: (i) the reverse-proxy URI address (e.g., */rp/*), (ii) the location path of the EP hosting the resources, and (iii) the resource URI-reference. This design choice yields a fully backwards-compatible CoAP URI, and a transparent redirection of client requests from real resources to the reverse-proxy.

The link binding, resource polling and application notification operations are illustrated with the example in Figure 1. After a discovery operation, an application knows the virtual URIs of the registered resources. A CoAP GET request is sent by the CoAP client to establish a resource binding relation. The reverse-proxy can resolve the real URI associated with the target resource by simply extracting from the virtual URI the location path of the resource descriptor in the RD. We have also extended the basic `Observe` binding to include the support for periodic conditional observing. Specifically, we have introduced in

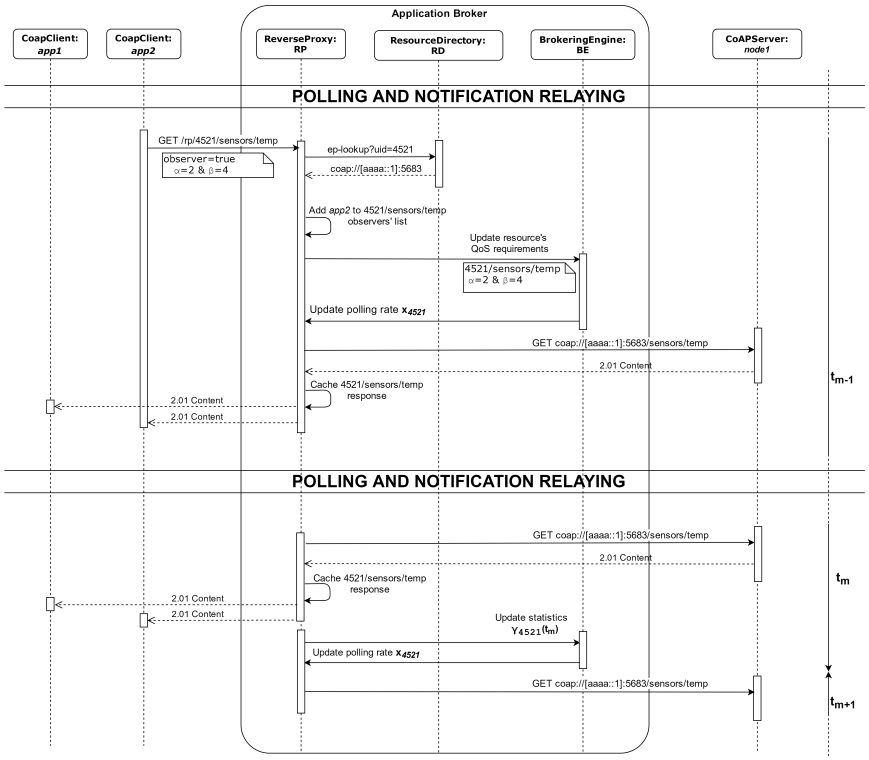


Figure 1 Sequence diagram of the observe binding, resource polling and application notification.

the conditional GET two additional options to specify the α_j and β_j parameters. Then, the reverse-proxy registers these parameters in the broker engine, that updates the polling rate for the target resource. As new client applications register for the same target resource, the broker engine also updates the polling rate at the end of each decision cycle. The reverse-proxy maintains a list of notification deadlines for the application registered to the same resources and notifies the cached contents before deadline expiration.

Finally, the broker engine maintains information about the routing topology. This is achieved by requiring that each CoAP EP in the IoT domain exposes a resource with URI reference */RP-info/*, which returns the IP address of the node's parent. Then, we extended the EP descriptor in the RD to include a *parent* attribute, which is updated by the EP itself with the content of the RP-info resource. This last modification allows the broker engine to obtain complete topological information by merely querying the RD module.

6 Performance Evaluation

In this section, we describe the results obtained from both a simulation study and an IoT testbed. Simulations are important to perform experiments in controlled and reproducible conditions. In our case, they are also needed to better isolate the impact of the channel behaviours on the efficiency of the optimisation framework. Before presenting the results, we detail the benchmarks used for performance comparison.

6.1 Benchmarks

The first benchmark, referred to as PROXY, is a conventional CoAP reverse-proxy that allows applications to register their interest in a resource by sending an extended GET request that specifies the QoS requirements (i.e. α_j and β_j). Then, the PROXY establishes a link binding with that resource using the Polling method [23], choosing as polling frequency the maximum notification rate requested by the clients bound to the same resource. This ensures that all the applications attain that maximum QoS satisfaction. The second benchmark, called OPT, is still based on a CoAP reverse-proxy that establishes polling bindings, but the polling frequency is chosen according to the optimal solution of the optimisation problem described Section 3. As far as the $g_j(x)$ function (i.e., the QoS satisfaction index for application j) is concerned, we select an S-shaped function, which is commonly used in network utility maximisation problems for resource allocation [15]. More formally, we have that

$$g_j(x) = \begin{cases} 0 & x \leq \alpha_j \\ 2\left(\frac{x-\alpha_j}{\beta_j-\alpha_j}\right)^2 & \alpha_j \leq x \leq x^0 \\ 1-2\left(\frac{x-\alpha_j}{\beta_j-\alpha_j}\right)^2 & x^0 \leq x \leq \beta_j \\ 1 & x_k > \beta_j \end{cases}, \tag{8}$$

where $x^0 = (\alpha_j + \beta_j)/2$. Clearly, our proposed solution, called BROKER for brevity, adopts the formulation in (8) for resource ranking, as described in Algorithm 4. The Matlab software was used to solve the optimisation model.

6.2 Simulation study

First, we introduce the simulation tool used for the evaluation. Secondly, we explain how simulations have been conducted, and finally, we present the simulation results.

6.2.1 Simulation setup

For the simulation study, we used Cooja, a cross-level network simulator that allows simulating networks of Contiki motes, which can also be emulated at the hardware level [26]. Contiki (http://www.contiki-os.org) is a popular open-source operating system for IoT systems, which provides full support for Internet standards (6lowpan, RPL, CoAP).

The simulated network scenario consists of a regular 6x6 grid topology, with the length of a grid cell equal to 10 meters. The sink is deployed at one of the corners of the grid, and it also behaves as the border router interconnecting the IoT domain to external applications. RPL is used for building the network paths between the IoT devices and the sink. To avoid that routing instabilities caused by RPL [27] interfere with CoAP performance, before activating the applications we let the routing algorithm to run until it converges to a stable topology (5 minutes) and, then, we freeze the network paths.

At the PHY and MAC layers, IoT nodes use the IEEE 802.15.4 standard. No radio duty-cycling mechanism is used at the MAC layer. To model real-world propagation and interference conditions we use the MRM model included in Cooja, which accounts for multi-path and fading [28]. We set up the model parameters to obtain a 100% success rate at 10 meters, and an interference range of 20 meters. It is important to point out that the use of a deterministic interference range allows to precisely define the $d(i,h)$ distance in the

Table 1 Parameters for the broker.

Parameter	Value
T	90 seconds
α_γ	0.2
$HM = \Gamma_H - \Gamma_L$	0.1
$\mu = (\Gamma_H + \Gamma_L)/2$	{ 0.1, 0.2, 0.3}
β_γ	0.4
$\{\omega_1, \omega_2, \omega_3\}$	{ 0.2, 0.1, 0.1}
τ	0.8
v	10

optimisation framework. As already pointed out, the most challenging task for the optimal approach is to estimate the network link capacity C_{ij} as seen at the application layer. It is straightforward to recognise that the C_{ij} value is lower than the data transmission rate as various protocol overheads (e.g., packet headers and acknowledgements) should be taken into account. Thus, we experimentally evaluate the C_{ij} value by running an experiment with only two nodes, a CoAP client and a CoAP server, exchanging CoAP messages. Then, the C_{ij} value is approximated with the maximum transaction frequency that guarantees a negligible average TLR ($\leq 1\%$). Under our settings, we have that C_{ij} is equal to about 60 kbps.

It is important to point out that Cooja is not only a network simulator, but it also allows running real software to test it before deployment. Moreover, it also allows injecting real traffic into the simulator. Thus, in our simulations, both the broker prototype and the CoAP clients are Java applications that run as independent processes on the local machine on which Cooja runs. To bridge the CoAP traffic between the CoAP clients, the broker and the RPL network simulated in Cooja we use the Tunslip utility provided in Contiki [29].

The evaluation focuses on the impact that an increasing application demand has on three performance metrics:

- *application goodput*, measured as the average number of resource state updates that are successfully received by the CoAP clients;
- *total TLR*, i.e., the percentage of transactions losses over the entire simulation;
- *QoS satisfaction*, evaluated as the distribution of the average QoS satisfaction of each CoAP client;
- *energy efficiency*, measured as the ratio between the total energy consumption and the total number of successful CoAP messages.

Table 1 lists the value of the main parameters for the broker. Note that the parameter μ, defined as the arithmetic mean of the γ_L and γ_H thresholds, is equivalent to the target reliability that the broker tries to maintain, as the transitional reliability region is centred around μ.

Finally, each simulation lasts 30 minutes, and the 95% confidence intervals are computed by replicating each simulation ten times. Confidence intervals are represented by vertical errorbars in the graph (often in the graphs small errorbars collapse to a point).

6.2.2 Simulation results

In the following evaluation we assume that there are ten CoAP clients registered to each resource. Without loss of generality we assume that $\alpha_j = \alpha$ for all $j \in A$, while each CoAP client randomly selects a β_j value within the range $[\alpha, \alpha + \Delta\alpha]$. Clearly, the higher the α value, and the higher the minimum notification rate that the CoAP server hosting that resource should guarantee. Moreover, the higher the $\Delta\alpha$ value, and the higher the application heterogeneity. To fairly compare BROKER to PROXY and OPT we introduce the *maximum total load* as the $\sum_{k \in R} \max_{j \in A_k} \{\beta_j\}$. In other words, the maximum observing load expresses the total number of state updates that should be received at the proxy to satisfy the maximum QoS requirements of all the client applications.

Figure 2 shows the application goodput as a function of the maximum observing load for the BROKER and the two benchmarks. We consider three cases for the target reliability μ, namely 10%, 20% and 30%. Several observations can be derived from the reported results. First, all the considered schemes perform similarly if the network is not congested. In this case, the QoS requirements of CoAP clients are fully satisfied. Second, when the network is deeply congested, both PROXY and OPT suffer from goodput degradation, which is more severe for PROXY than OPT. The root cause of performance degradation for PROXY is that it is unaware of network-related constraints and it continues to poll the IoT resources with the maximum notification rate that is demanded by the CoAP clients. Although OPT implements a rate adjustment policy, the rate decisions are not efficient due to over-optimistic estimate of channel capacities in congested scenarios. On the other hand, BROKER can attain a stable goodput since it promptly reacts to the onset of network congestion and it reduces the notification rate as the TLR value increases. It is important to point out that there is a clear trade-off between the TLR thresholds used to detect the start and the end of LR state, and the efficiency of the BROKER scheme. Specifically, if μ is too low (e.g., 0.1), the broker starts reducing the notification rate too early and this prevents the broker to fully utilise the available network bandwidth Clearly, this behaviour is more evident in the transitional region of TLR values, i.e., under intermediate conditions for traffic congestion. Interestingly, we can observe that $\mu = 0.3$ provides the best performance for BROKER, which obtains up to three times more goodput than PROXY.

To confirm the above conjecture, Figure 3 shows the total TLR in the same configurations of Figure 2. The results demonstrate that our solution is very effective in controlling the transmission reliability by limiting the notification rate. On the contrary, both PROXY and OPT suffers from an increasing number of transaction losses as the congestion level rises. To get a deeper insight into the broker's decision process, we plot the box-plots of the clients' QoS satisfaction index for two relevant cases in Figure 4. Specifically, case A (shown in Figure 4a) corresponds to the scenario in which PROXY obtains the maximum goodput. In such scenario, there are no remarkable differences between the algorithms, except for BROKER with $\mu = 0.1$ which underutilise the network. This is also reflected in smaller values for the QoS satisfaction index of the CoAP clients. Case B (shown in Figure 4b) corresponds to scenario in which OPT performs for the first time worse than BROKER. The results show that BROKER is able to guarantee higher QoS satisfaction values on average to most CoAP clients as it avoids that CoAP clients requesting higher notification rates gets an unfair share of the network resources.

Finally, Figure 5 shows the energy efficiency values as a function of the maximum observing load for the BROKER and the two benchmarks. Interestingly, we can observe that not only PROXY and OPT obtain a lower goodput when the network is congested, but

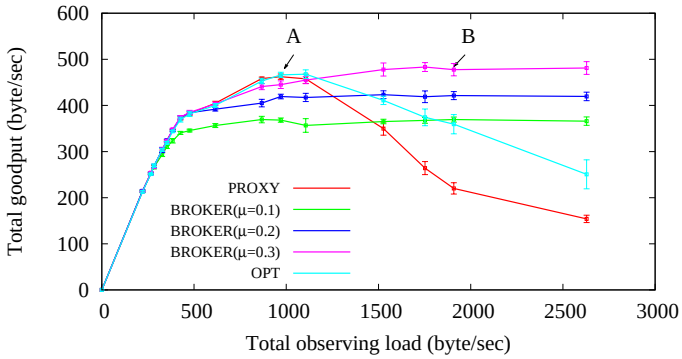


Figure 2 Total goodput versus the maximum observing load (ten clients per IoT resource).

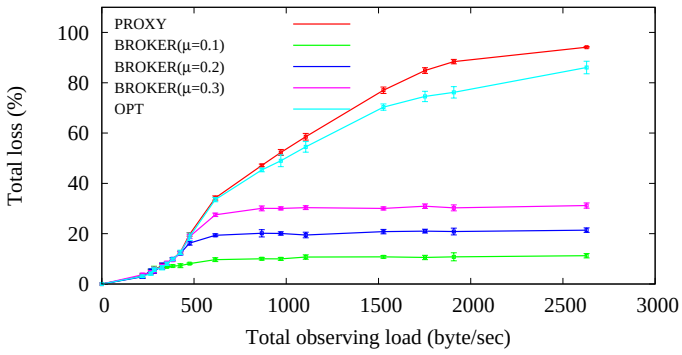
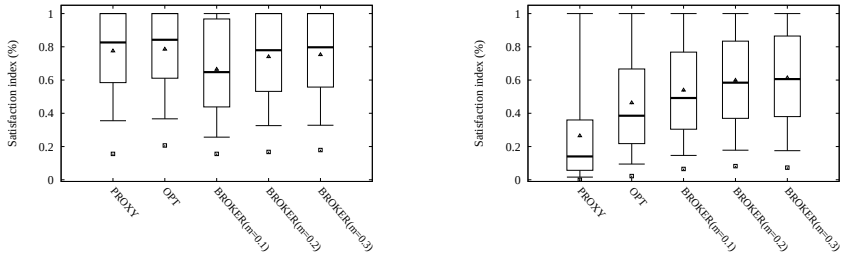


Figure 3 TLR versus the maximum observing load (ten clients per IoT resource).



(a) $case = A$ (b) $case = B$

Figure 4 Boxplots of clients' QoS satisfaction values for two representative cases.

they also consume more energy to complete the CoAP message exchanges. This can be explained by observing that BROKER avoids overloading the network, thus reducing the energy wasted in packet retransmissions and packet losses due to buffer overflows.

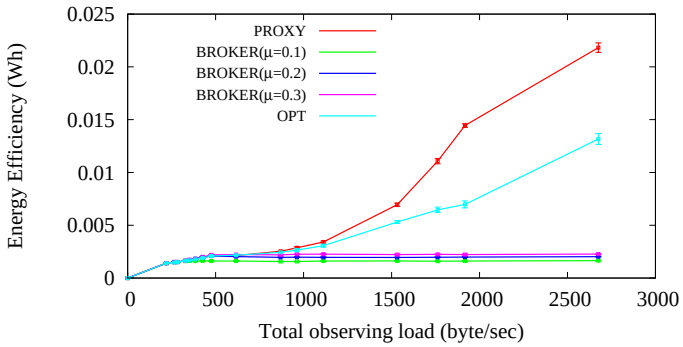


Figure 5 Energy efficiency versus the maximum observing load (ten clients per IoT resource).

6.3 Experimental study

In the following we first introduce the testbed where experiments have been conducted, and finally we present the experimental results.

Figure 6 Physical deployment of the Strasbourg testbed.

6.3.1 Experimental methodology

To carry out our experimental study, we exploited the FIT IoT-LAB platform [30]. IoT-LAB provides a very large scale infrastructure for IoT experimentation that consists of 1823 wireless sensors nodes deployed over six different testbeds across France. We conducted our experiments in the Strasbourg testbed, which is located at ICube laboratory. Different boards are available in this testbed, but we fused the M3 and A8 open motes as they are capable of running Contiki. More precisely, M3 nodes are based on an ARM Cortex M3 microcontroller, have a 64kB RAM, and a radio interface with an 802.15.4 PHY Layer. Instead, A8 nodes are powerful nodes based on an ARM M8 micro-controller, which allows running high-level OSs like Linux. Therefore, each A8 also embeds an M3 node to support 802.15.4 communications. In the Strasbourg testbed, nodes are deployed on wooden poles forming a two-floor grid topology (see Figures 6). Node's distance is approximately 2 meters on both X and Y axes, while nodes on the same pole are distant 90 centimetres. For our evaluation, we selected 39 nodes (seven A8 nodes and 32 M3 nodes) on the same topology layer, as shown in Figure 7. Our broker is deployed in the red node in the figure, which is a powerful A8 node. Due to the vicinity of nodes if was necessary to fine-tune the transmission power of the 802.15.4 radio interfaces in order to build meaningful network topologies. After an extensive testing, we set the antenna TX power level to -3dBm and the RX sensitivity to -60dBm, which ensured to have a network topology with an average node degree equal to six.

In the following evaluation, we tried to replicate the same scenarios of the simulation study. However, due to time limitations (IoT Lab is a crowdsourced experimental platform and reserved experiments are queued based on testbed availability and resources), we focus our tests on the congested regime. Furthermore, results have been obtained only for `PROXY` and `BROKER` as it is challenging to obtain accurate estimates of channel capacities and link interferences in a real testbed.

6.3.2 Experimental results

The experimental evaluation confirms the findings and trends that were highlighted in Section 6.2.2. Specifically, Figure 8 confirms that `BROKER` is able to provide a stable goodput even under deep network congestion, while `PROXY` experiences a goodput collapse. Interestingly, the performance differences between the three `BROKER` variants are less notable than in the simulated scenarios. In other words, in real-world conditions, the performance of our proposed solution seems less sensitive to the choice of the

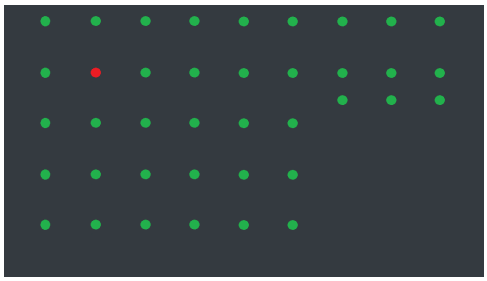


Figure 7 Topology of the network used in the experiments.

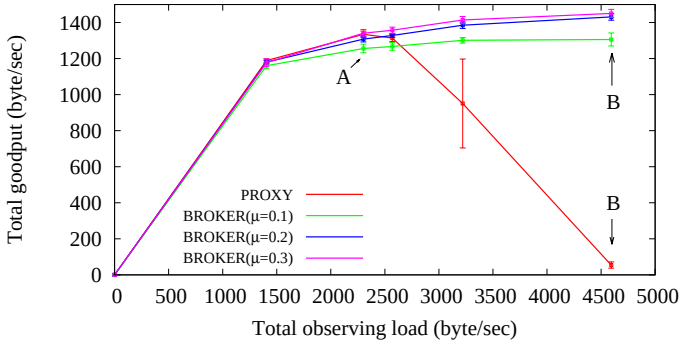


Figure 8 Total goodput versus the maximum observing load (ten clients per IoT resource).

μ parameter. Furthermore, PROXY experiences a severe collapse of performance when network congestion is high. Figure 9 shows the TLR measured in the experiments, and the results confirm the ability of BROKER to bound the percentage of CoAP transaction losses to a target value. Finally, we have also analysed the distribution of the QoS satisfaction index for two relevant cases. The former compares the algorithms when network experience moderate congestion, while the latter corresponds to a state of deep congestion. Again, we obtain similar results as the one shown in Section 6.2.2. Case A (shown in Figure 10a) confirm that a legacy proxy and a more sophisticated broker behave similarly if there is no traffic congestion. Case B (shown in Figure 10b) highlights the superiority of BROKER over PROXY to satisfy applications' QoS requirements with high congestion.

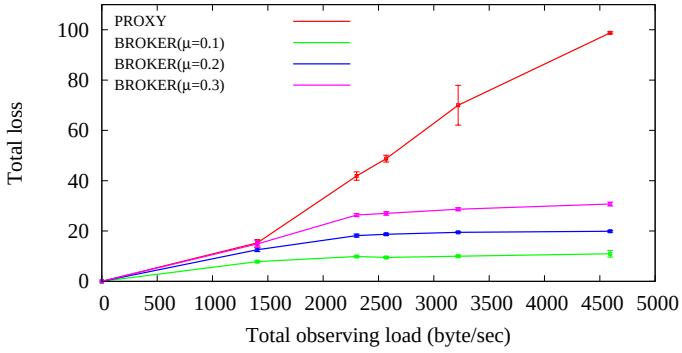
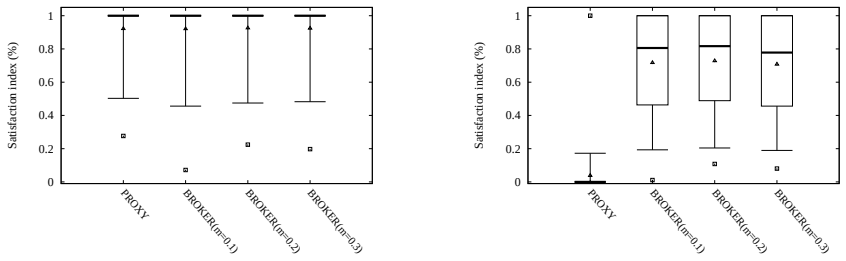


Figure 9 TLR versus the maximum observing load (ten clients per IoT resource).



(a) $case = A$ (b) $case = B$

Figure 10 Boxplots of clients' QoS satisfaction values for two representative cases.

7 Related Work

Several IoT platforms and solutions have been developed to support resource management in IoT domains, which implement services for the monitoring of resource usage, fair resource allocation, and resource conflict resolution [3].

A class of solutions have been proposed that leverages virtualisation technologies to allow multiple applications to share the same IoT network and devices. For instance, Maté [31] is a virtual machine (VM)-based architecture for constrained devices that provides support for adaptability and resource management (e.g., modifying sampling rate of sensors or dynamically installing data aggregation functions). UMADE is another example of a management platform proposed in [32], which defines a utility-based sensor selection scheme for heterogeneous applications that dynamically reallocate applications based on network dynamics and applications' QoS requirements. More recently, an optimisation framework and greedy-based heuristic have been proposed in [33] for solving the problem of application admission and physical resource leasing in sensor networks with multiple concurrent applications with heterogeneous requirements and constraints. A QoS-aware service selection algorithms for cloud-based IoT systems is proposed in [34] to minimise network energy consumption while guaranteeing that all service invocations are entirely executed before their deadline.

An alternative approach is adopted in the management frameworks that rely on IoT gateways to directly handle the interactions between applications and IoT resources, providing efficient services for resource discovery and access. In cloud-centric IoT systems, such IoT gateways are typically deployed in the cloud infrastructure [35]. For instance, the DNS-SD protocol uses a central repository to facilitate service discovery for IoT applications [36]. To deal with scalability issues, most recent IoT platforms proposes the use of "local" gateways deployed on network nodes in proximity to the IoT network. For instance, the oneM2M standard proposes a functional architecture in which Common Services Entities (CSEs) form a common service layer between applications and things, providing several service functions, including discovery, data and application management [12]. Then, CSEs can be hosted in the cloud or on the home gateway. The OMA Lightweight M2M (LWM2M) protocol is a COAP-based application protocol for remote IoT-device management that defines LWM2M clients and servers, the former deployed on end devices and the latter on local gateways, monitoring and managing the devices' resources [37]. A COAP-based distributed architecture that provides a common standard interface to applications for global discovery and access of IoT resources is proposed in [38] by federating different IoT gateways implemented in fog nodes. Similarly to our work, a local reverse-proxy is deployed with the RD in the IoT gateway for handling application requests on behalf of constrained devices. However, the proposed broker simply relays the application requests without taking into account applications' QoS requirements. The same authors address in another work the problem of designing a CoAP proxy that supports multiple CoAP observers with QoS requirements by selecting the optimal notification period that minimises the number of messages sent by the server [39]. However, limitation of network capacity is not considered in the proposed optimisation framework. A similar problem is considered in [40], where a greedy-based heuristic is designed to allow a CoAP proxy to approximate a max-min fair allocation of notification periods of multiple CoAP observers. The formulation in [39] allows the applications to specify the required values for both the minimum and maximum notification periods, while the formulation in [40] allows the applications to specify only the minimum desired notification period.

8 Conclusions

In this work, we have proposed an edge-centric solution for resource brokering in IoT systems, which deploys a CoAP proxy and RD on a local gateway to seamlessly and transparently handle application requests. The broker regulates the access to IoT resources in such a way to maximise the applications' QoS satisfaction while taking into account network-related constraints. We have shown how resource brokering can be cast to an optimisation problem, and we have discussed the limitations of this approach when network capacity is uncertain or difficult to estimate. Thus, we have also proposed practical algorithms for the selection of the polling rates from the broker to the IoT resources that leverage direct measurements of the degree of reliability of application transmissions. The efficacy of the proposed solution has been extensively evaluated through simulations and experiments in an IoT testbed. Performance is compared to a conventional CoAP proxy and the optimal approach.

As future work, we plan to extend the problem formulation to consider that multiple resources can be hosted in the same IoT device and a conflict resolution mechanism between co-located resources is needed. Furthermore, cooperation between brokers of geographically collocated IoT domains is also a future avenue of research.

References

[1] Mark Hung, editor. *Leading the IoT*. Gartner, 2017.

[2] Eleonora Borgia. The Internet of Things vision: Key features, applications and open issues. *Computer Communications*, 54:1–31, December 2014.

[3] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke. Middleware for Internet of Things: A Survey. *IEEE Internet of Things Journal*, 3(1):70–95, February 2016.

[4] CISCO. Cisco Visual Networking Index: Forecast and Trends, 2017–2022, November 2018.

[5] N. Hassan, S. Gillani, E. Ahmed, I. Yaqoob, and M. Imran. The Role of Edge Computing in Internet of Things. *IEEE Communications Magazine*, 56(11):110–115, November 2018.

[6] Yuan Ai, Mugen Peng, and Kecheng Zhang. Edge computing technologies for Internet of Things: a primer. *Digital Communications and Networks*, 4(2):77–86, 2018.

[7] G. Premsankar, M. Di Francesco, and T. Taleb. Edge Computing for the Internet of Things: A Case Study. *IEEE Internet of Things Journal*, 5(2):1275–1284, April 2018.

[8] M. Mukherjee, L. Shu, and D. Wang. Survey of Fog Computing: Fundamental, Network Applications, and Research Challenges. *IEEE Communications Surveys Tutorials*, 20(3):1826–1857, thirdquarter 2018.

[9] M. Chiang and T. Zhang. Fog and IoT: An Overview of Research Opportunities. *IEEE Internet of Things Journal*, 3(6):854–864, December 2016.

[10] J. Santos, T. Vanhove, M. Sebrechts, T. Dupont, W. Kerckhove, B. Braem, G. V. Seghbroeck, T. Wauters, P. Leroux, S. Latre, B. Volckaert, and F. D. Turck. City of Things: Enabling Resource Provisioning in Smart Cities. *IEEE Communications Magazine*, 56(7):177–183, July 2018.

[11] G. Tanganelli, C. Vallati, and E. Mingozzi. Edge-centric Distributed Discovery and Access in the Internet of Things. *IEEE Internet of Things Journal*, 2018.

[12] oneM2M. Functional Architecture. TS-0001-V3.12.0, July 2018.

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

[14] Frank Kelly. Charging and rate control for elastic traffic. *Emerging Telecommunications Technologies*, 8(1):33–37, January-February 1997.

[15] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

[16] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, Maria Kazandjieva, David Moss, and Philip Levis. CTP: An Efficient, Robust, and Reliable Collection Tree Protocol for Wireless Sensor Networks. *ACM Trans. Sen. Netw.*, 10(1):16:1–16:49, December 2013.

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

[18] Xiao Long Huang and Brahim Bensaou. On Max-min Fairness and Scheduling in Wireless Ad-hoc Networks: Analytical Framework and Implementation. In *Proc. of ACM MobiHoc '01*, pages 221–231, 2001.

[19] T.H. Cormen, C E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, third edition, 2009.

[20] Z. Shelby. Constrained RESTful Environments (CoRE) Link Format. IETF RFC 6690, September 2012.

[21] M. Nottingham. Web Linking. IETF RFC 4287, October 2010.

[22] Z. Shelby, M. Koster, C. Bormann P. van der Stok, and C. Amsuess. CoRE Resource Directory. Internet Draft (Expires: July 15, 2019), January 2019.

[23] Z. Shelby, M. Koster, C. Groves, J. Zhu, and B. Silverajan. Dynamic Resource Linking for Constrained RESTful Environments. Internet Draft (April 25, 2019), October 2018.

[24] K. Hartke. Observing Resources in the Constrained Application Protocol (CoAP). IETF RFC 7641, September 2015.

[25] https://www.eclipse.org/californium.

[26] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-Level Sensor Network Simulation with COOJA. In *Proc. of IEEE LCN'06*, pages 641–648, November 2006.

[27] E. Ancillotti, R. Bruno, and M. Conti. Reliable Data Delivery With the IETF Routing Protocol for Low-Power and Lossy Networks. *IEEE Transactions on Industrial Informatics*, 10(3):1864–1877, Aug 2014.

[28] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-Level Sensor Network Simulation with COOJA. In *Proc. of IEEE LCN'06*, pages 641–648, November 2006.

[29] https://anrg.usc.edu/contiki/index.php/rpl_border_router.

[30] https://www.iot-lab.info.

[31] Philip Levis and David Culler. MatÉ: A Tiny Virtual Machine for Sensor Networks. *SIGARCH Comput. Archit. News*, 30(5):85–95, December 2001.

[32] S. Bhattacharya, A. Saifullah, C. Lu, and G. Roman. Multi-Application Deployment in Shared Sensor Networks Based on Quality of Monitoring,. In *Proc. of IEEE RTAS'10*, pages 259–268, 2010.

[33] C. Delgado, M. Canales, J. Ortín, J. R. Gállego, A. Redondi, S. Bousnina, and M. Cesana. Joint Application Admission Control and Network Slicing in Virtual Sensor Networks. *IEEE Internet of Things Journal*, 5(1):28–43, 2018.

[34] G. Tanganelli, C. Vallati, and E. Mingozzi. Energy-Efficient QoS-aware Service Allocation for the Cloud of Things. In *Proc. of IEEE CloudCom'14*, pages 787–792, December 2014.

[35] S. K. Datta, R. P. F. Da Costa, and C. Bonnet. Resource discovery in Internet of Things: Current trends and future standardization aspects. In *Proc. of IEEE WF-IoT'15*, pages 542–547, 2015.

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

[37] Open Mobile Alliance. Lightweight Machine to Machine Technical Specification, Version 1.0.2, February 2018.

[38] G. Tanganelli, C. Vallati, and E. Mingozzi. Edge-Centric Distributed Discovery and Access in the Internet of Things. *IEEE Internet of Things Journal*, 5(1):425–438, 2018.

[39] G. Tanganelli, C. Vallati, E. Mingozzi, and M. Kovatsch. Efficient proxying of CoAP observe with quality of service support. In *Proc. of IEEE WF-IoT'16*, pages 401–406, 2016.

[40] R. Bruno and S. Bolettieri. Design and Implementation of a COAP-Based Broker for Heterogeneous M2M Applications. In *Proc. of IEEE ICIOT'18*, pages 1–8, 2018.