Oops: Optimizing Operation-mode Selection for IoT Edge Devices

FARZAD SAMIE, Karlsruhe Institute of Technology (KIT), Germany
VASILEIOS TSOUTSOURAS, National Technical University of Athens, Greece
LARS BAUER, Karlsruhe Institute of Technology (KIT), Germany
SOTIRIOS XYDIS, National Technical University of Athens, Greece
DIMITRIOS SOUDRIS, National Technical University of Athens, Greece
JÖRG HENKEL, Karlsruhe Institute of Technology (KIT), Germany

The massive increase of IoT devices and their collected data raises the question how to analyze all that data. Edge computing may provide a suitable compromise, but the question remains: how much processing should be done locally vs. offloaded to other edge devices? The diverse application requirements and limited resources at the edge (i.e. gateways and IoT devices) extend the challenges.

We propose Oops, an optimization framework to decide and adapt the resource management at runtime in a distributed manner. It orchestrates the IoT devices and adapts their operation mode with respect to their constraints and limited shared resources on the gateway. Experimental results show a significantly reduced runtime overhead while even increasing the user utility compared to state-of-the-art.

CCS Concepts: • **Computer systems organization** → **Embedded and cyber-physical systems**; • **Networks** → *Network resources allocation*; Network management; • **Theory of computation** → *Online algorithms*;

Additional Key Words and Phrases: Internet of Things, IoT, Edge Computing, Constrained Devices

ACM Reference Format:
Farzad Samie, Vasileios Tsoutsouras, Lars Bauer, Sotirios Xydis, Dimitrios Soudris, and Jörg Henkel. 2018. *Oops*: Optimizing Operation-mode Selection for IoT Edge Devices. *ACM Trans. Internet Technol.* X, X, Article X (May 2018), 20 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

With the emergence of Internet of Things (IoT), a massive number of embedded devices are connecting to the Internet to provide advanced monitoring and control services [26]. This will lead to a rapid increase in the scale of collected data. Hence, one of the challenges in IoT is to process and analyze a huge amount of data from heterogeneous devices [40]. This challenge has two aspects: 1) large quantities of data, and 2) diverse application requirements of IoT applications [40]. Handling all these collected IoT data with central cloud servers is inefficient and sometimes

Authors' addresses: Farzad Samie, Karlsruhe Institute of Technology (KIT), Haid-und-Neu-Str. 7, Karlsruhe, 76131, Germany, farzad.samie@kit.edu; Vasileios Tsoutsouras, National Technical University of Athens, Greece, billtsou@microlab.ntua.gr; Lars Bauer, Karlsruhe Institute of Technology (KIT), Haid-und-Neu-Str. 7, Karlsruhe, 76131, Germany, lars.bauer@kit.edu; Sotirios Xydis, National Technical University of Athens, Greece, sxydis@microlab.ntua.gr; Dimitrios Soudris, National Technical University of Athens, Greece, dsoudris@microlab.ntua.gr; Jörg Henkel, Karlsruhe Institute of Technology (KIT), Haid-und-Neu-Str. 7, Karlsruhe, 76131, Germany, henkel@kit.edu.

1533-5399/2018/5-ARTX $15.00
https://doi.org/0000001.0000001

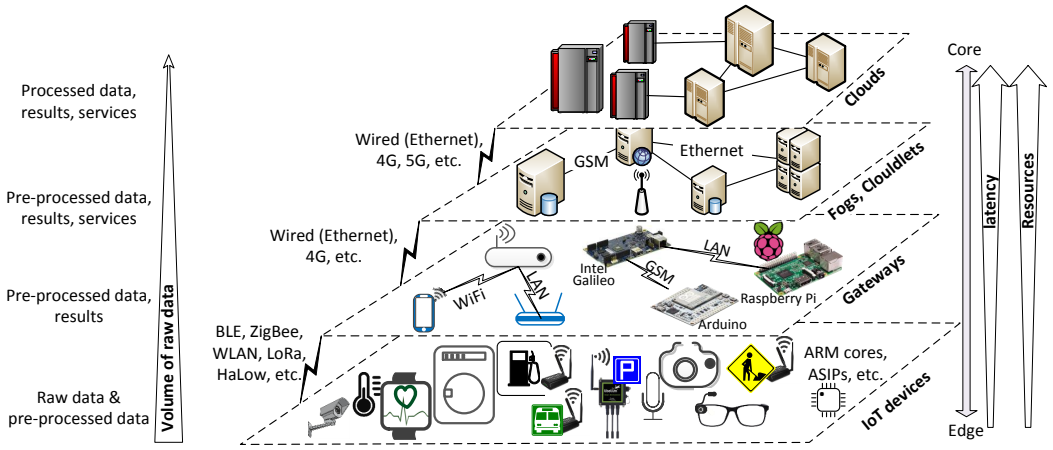


Fig. 1. The hierarchical architecture of IoT system including *things* and smart sensors, gateways, fog nodes, and cloud servers [25].

even unfeasible, because of (i) the overall energy consumption, cost and load on the network to transmit the data, and (ii) unreliable latency due to networking as well as computation workload [7, 30].

To address these issues, the tasks of processing the data and decision making are pushed to the network edges, close to data source [19]. This introduces the new paradigms of fog computing [13] and edge computing [30]. Fog computing extends the traditional cloud by distributing the computation, storage and communication resources geographically closer to the end user. Edge computing further extends the computing architecture of IoT and incorporates gateways and IoT devices into the data processing and decision making. According to a report by International Data Corporation (IDC) Futurescape, around 40% of IoT-generated data will be processed, stored, and acted upon close to the edge of network [24].

Figure 1 shows the hierarchical architecture of computation layers in IoT. IoT devices (smart sensors and *things*) interact with the physical world and collect the data. Due to their limited energy sources, IoT devices are equipped with low-power wireless technologies including Bluetooth Low Energy (BLE), ZigBee, HaLow and Low Power Wide Area Network (LPWAN) like LoRa [28] to communicate with each other or with gateways. Gateways (ubiquitous smart devices) enable seamless integration of low-power wireless networks of IoT devices with other networks (e.g. WiFi, cellular network, LAN, etc.). Gateways may transmit the data to the fogs and cloudlets that are a few network hops away from the edge and have more storage and computation resources for low-latency service of intensive tasks. However, some applications and services may still remain cloud-based due to the substantial or long-term storage and computation requirements.

Even though at the core of the network (i.e. cloud) more computation and storage resources are available, the latency of the services will increase and become non-deterministic [7] as a consequence of the high workload to transmit a large amount of data. To perform the computation and make decisions as close as possible to the data source, edge computing envisions a model in which the IoT gateways provide the local processing services besides their conventional functions to bridge low-power wireless network of IoT devices and the Internet [5, 39]. Thus, the data processing can be partitioned between different IoT devices, gateways and –if needed– fogs and cloudlets. The hierarchical architecture of IoT system offers the opportunity for efficient computation offloading to overcome the resource constraints at the edge.

Besides the computation, the management tasks shall be pushed closer to the edge too. For instance, the management of IoT devices, orchestration and resource allocation of edge devices can be handled at the gateway level, the management of gateways can be handled at the fog level, etc.

Motivation:

IoT gateways are usually located in the vicinity of IoT devices, and therefore, require a small form-factor (e.g. they cannot afford to deploy high-performance processors due to cooling requirements). Therefore in many IoT systems, the gateways are resource-constrained, too [8]. For instance, Intel IoT gateways (DK50 and DK100 series) are based on Intel Quark SoC X1000 and X1020D with an operating frequency of only 400 MHz [2]. Interestingly, the computational competency of the gateways is only one of the critical constraints of gateway-based IoT system. The available communication bandwidth can turn into a significant bottleneck for supporting complex applications and numerous IoT nodes since data transmission is limited to only a few kilobits per second [28]. This stems from the fact that the communication between the gateway and IoT edge devices is via a low-power wireless connection such as BLE, LoRa, HaLow, etc. that have a low throughput [19, 39].

In addition, the portion of computation and communication capabilities of gateways that is allocated to edge processing tasks is constrained, in order to avoid resource starvation for the rest of the tasks of the gateway. These limited resources are shared by all IoT devices that are connected to each gateway and at runtime depending on its workload for performing other management tasks (e.g. operating system), the gateway shall restrict its available resources (communication bandwidth and processing power) offered to the IoT devices (see Figure 2).

From the application point of view, future IoT edge applications are anticipated to offer various different *operation modes*, corresponding to different configuration knobs of the application such as input data quality, the onboard processing policy, etc. Switching between different modes is envisioned to be a dynamic feature of the application, tightly coupled to its runtime functional constraints such as remaining battery capacity. For instance, in the case of an IoT-based fitness & health monitoring device that captures, processes and uploads the Electrocardiogram (ECG) signal to a back-end server infrastructure, when the battery level is low the device can switch to a mode with lower sampling rate for data acquisition (e.g. from 360 Hz to 250 Hz) [29] and thus dynamically reduce its workload. It can alternatively or conjointly avoid the pre-processing (e.g. performing digital filters for baseline wandering removal [17]), and *offload* it to the gateway (see Figure 2).

In order to quantify the effectiveness of a gateway-based IoT system, each operating mode of an IoT node is associated with a quality level that affects the user's satisfaction. In the context of resource allocation, the amount of benefit or gain that each user receives under that specific circumstance is called *utility* [20, 32] and has been introduced as a 'soft requirement' of the system. The same term has been described as 'service quality' or 'quality of service' in the literature [18, 29]. In general, the operation modes of devices differ in their energy consumption, resource usage (both on the device and shared gateway), as well as the utility provided to the user.

While the individual building blocks of the system are well defined (i.e. gateways, IoT nodes, application operating modes), a runtime management and configuration infrastructure is crucial in order to take advantage of the full system capabilities in terms of offered user satisfaction. Taking this into account, this article presents a framework for the runtime management of the IoT devices in a gateway based system, which achieves enhanced user satisfaction by determining their operation modes and edge configuration. The framework optimizes the allocation of the constrained shared resources with a dynamic mechanism that is transparent to the end user. We aim at selecting operation modes for each IoT edge device at runtime and efficiently allocating the shared resources of the gateway to them with respect to different runtime constraints of the edge devices and the gateway.

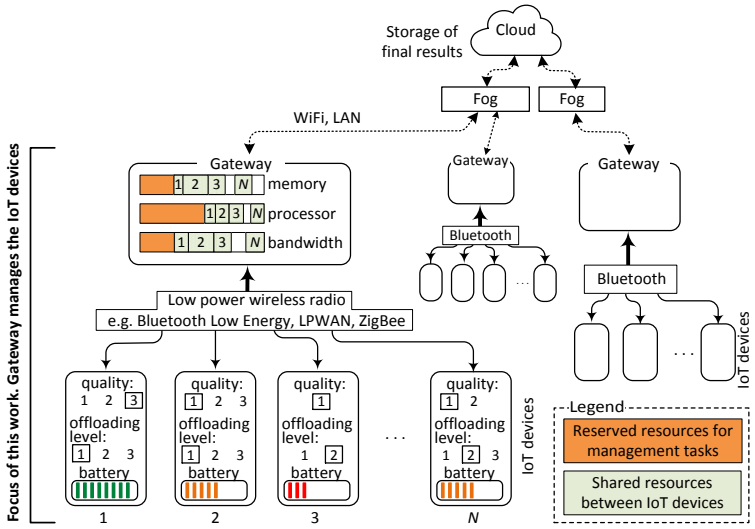


Fig. 2. Problem model: Gateways with their connected IoT devices. Each device has different operation modes (quality & offloading level) and each mode requires different amount of resources.

The shared gateway is used as the orchestrator on the system and thus it is the execution point of the framework. The focus of this article is the edge, where the IoT devices and gateway are located, as shown in Figure 2. Nevertheless, the presented general principles can be employed to orchestrate the connected gateways to the fog infrastructure and its shared resources. The necessity of the dynamic online solutions for such IoT systems in comparison to static offline solutions is due to several reasons, e.g. i) the remaining energy of IoT devices varies over time due to consumption or recharge, ii) the available bandwidth or processing capability of the gateway may change over time due to its available power and workload of other tasks, and iii) at runtime, some devices may become more important and critical to their users. Last but not least, the IoT gateway must be able to dynamically allocate its resources to tasks related to edge, fog and cloud in order to allow for the seamless integration of all these three levels.

We consider IoT systems that opt for reaching fairness among IoT edge devices, e.g. by balancing a design parameter such as energy consumption, utility, resource usage, battery lifetime, etc. [10]. In these systems, a max-min optimization problem (i.e. maximizing the minimum variable) needs to be solved subject to the constraints of devices and gateway. For instance, to balance the battery lifetime of edge devices and prolong the lifetime of the network, the minimum battery lifetime should be maximized. The max-min goal is particularly advantageous for two reasons: (i) it balances the resource usage among the local network and avoids unfair or starvation situation, (ii) it allows supporting devices with different priorities [22]; the designer can prioritize critical devices by assigning a lower utility to their operating modes, which increases the chance that they obtain more resources (because the minimum utility shall be maximized).

The novel contributions of this work are as follows:

- We present a management framework for **o**ptimizing **op**eration-mode **s**election (*Oops*) for IoT edge devices when the objective of system is to reach a max-min fairness among devices. It takes the constraints of the edge devices and the gateway into account and determines an efficient schedule of the operating modes for each device according to the objective of the system. This framework supports a wide range of system optimization goals in the form of max-min (or min-max) objective.

- We formulate the problem of operation-mode selection as a tractable convex optimization problem. Then, we propose a novel approach to identify configurations that are dominated by others to reduce the problem size and consequently decrease the runtime overhead.
- We identify the properties of the problem and leverage them to reduce the complexity of the solution, thus making it more suitable for online usage on resource-constrained devices.

By efficient resource allocation and operation mode selection at the edge, *Oops* provides a transparent management layer on the gateway that integrates the edge devices to the rest of IoT system. We provide experimental validation of our proposed framework using the traces derived from measurements/profiling on actual IoT devices. Results show that the optimization and reduction techniques of *Oops* reduce the runtime overhead for operation-mode selection from more than 100 s to only 1.3 s on average compared to a system with the same optimization solver that does not benefit from the proposed reduction techniques.

The rest of this article is organized as follows. In Section 2, we review related work on resource allocation and edge computing in IoT. Then, we present our system model and the problem formulation in Section 3. We then provide a detailed presentation of our proposed framework and solution in Section 4. Experimental results and evaluations are presented in Section 5, while Section 6 concludes the article.

2 RELATED WORK

Tan et al. [32] studied the utility maximization with resource allocation in wireless networks. They consider the limited bandwidth as the constrained resource, while the user requires a mixture of different traffic types (e.g. soft QoS, hard QoS, etc.). They also provide models for the utility functions. Another utility-based approach presented in [20], studies bandwidth allocation in wireless networks under the constraints of wireless interference. However, these approaches support only one shared resource, which must be a continuous variable. Therefore, they are not applicable to systems with discrete operation modes and multiple shared resources.

Wong et al. [37] propose a two-level solution to allocate the shared bandwidth of home gateway at the network edge. In the first level, the gateways decide how much *credit* to spend in each time interval to receive bandwidth. Then in the second level, each gateway allocates its given bandwidth among its connected devices and applications. It prioritizes its own connected devices based on their pre-determined utility functions such that the overall utility is maximized. Their proposed solution to the first level problem is complementary to ours to determine the bandwidth between gateways and fog. However, the second-level solution only considers one shared resource on the gateway (bandwidth). It cannot be extended to support multiple shared resources such as processing capability of gateway, bandwidth, etc. Another limitation is that the assumption that devices can receive any continuous value as their bandwidth, but IoT devices are anticipated to have a few discrete operation modes.

In [29], we studied the joint problem of bandwidth and quality of service (QoS) management under resource constraints for a network of IoT devices. The allocated bandwidth and computation offloading level of devices are determined to maximize the overall QoS. A fundamental distinction between the optimization strategy used in [29] and our proposed max-min fairness optimization framework is that in [29] optimization delivers a single configuration as the solution. However, usually no single configuration can reach the max-min fairness. The reason is that due to the limited resources, not all the IoT devices can operate in the mode with the utmost utility at the same time. Therefore, our proposed strategy adopts a more adaptive view of the IoT edge delivering system solutions that alternate between several particular configurations, i.e. introducing also a sort of operation-mode scheduling. Another distinction is that [29] aims at maximizing the overall QoS at each step without foresight while one way to increase accumulated QoS over time is to prolong the

battery lifetime of device. Even though in [29] the goal is to maximize the overall QoS, its approach is the closest to ours and we adopted it to the max-min problem and compare with it in Section 5.

MiLAN [12] is a middleware to manage and allocate the limited network resources to low-power devices. It trades the performance of an application (i.e. its utility) for energy cost. However, MiLAN targets systems in which the overall utility must be maximized. While MiLAN only considers the bandwidth limitation of the network, our framework supports multiple scarce resources including the bandwidth, processing capability, memory, etc. Most importantly, MiLAN does not support systems with max-min objective.

Some existing works assume a cloud server as the destination of offloading while in the edge computing paradigm and our system the computation is offloaded from IoT devices to other resource-constrained devices (i.e. gateway) with limited bandwidth, processing power, etc. [6, 14, 38]. Some other research works consider the devices individually and assume that selecting the operating mode (e.g. level of offloading) for a device does not affect the other devices in the setup [16, 21]. However at the edge of IoT, several devices share the limited resources of gateways. Thus, the operating mode of one device would affect the other devices as it determines its usage of the shared resources.

In the aforementioned related work, the proposed solutions do not support the max-min (or min-max) optimization goals [6, 20]. The solution to the max-min objectives often is switching among a combination of different modes (and not one particular mode). To our knowledge, there is no framework for selection of operating modes for edge devices in resource-constrained systems at runtime, in order to achieve max-min (or min-max) optimization goals.

3 SYSTEM MODEL & PROBLEM FORMULATION

3.1 Applications' Operation Modes

Most of IoT applications process a stream of data or signal and they have the following properties: (i) they operate on subsequent buffers or segments of collected data, (ii) they have a pipelined structure of data processing, e.g. filtering, feature extraction, classification, post-processing [11, 25].

Some IoT applications offer their service at different qualities, ranging from 'low quality' to 'high quality'. Higher service quality increases the *utility* (i.e. user's satisfaction and soft requirements) [20, 32] but at the cost of higher resource usage both on the device and the gateway (e.g. energy consumption, bandwidth, etc.). Different service qualities are usually the result of different quality of input data (e.g. sampling rate). For instance, consider an IoT-based fitness & heart monitoring device that can capture ECG signals at multiple discrete sampling rates (e.g. 125 Hz, 250 Hz, and 360 Hz). The quality of the captured signal allows for processing of higher resolution and thus determines the service quality and affects the utility [29]. When the battery is low, the device can switch to a low quality mode to preserve energy.

Moreover, the pipelined structure of the IoT application enables it (i) to fully process the captured data on the IoT edge device and send only the results to the Internet or (ii) to process the data partially on the IoT edge device and offload the rest of computation to the gateway after one of the application's pipeline stages, or (iii) to offload the whole computation to the gateway by transmitting the raw data [30]. This would lead to different possible computation offloading levels.

Therefore, an IoT application may have multiple operating modes due to:

- a change in input data quality (i.e. sampling rate),
- computation offloading (or on-board processing) schemes,
- or conjunction of the aforementioned control parameters.

At run-time a switch between operating modes is achieved by concluding the processing of the current (undergoing) window of date (a.k.a. segment) and simultaneously buffering the next

segment according to the quality of the selected operating mode (if changed). More precisely, the usually low input data rate of IoT applications in combination to the meticulous, not frequent scheduling of different operating modes results in an not-fully-occupied application's pipeline. The reason is that processing of one segment is finished before the next segment is buffered.

3.2 Gateway Model

The IoT gateways need to be distributed and located in the vicinity of IoT devices, and hence, require small form-factor (e.g. cannot afford deploying high performance processors due to cooling requirements). Due to these limitations, the processing power, memory and bandwidth of the IoT gateway to communicate with IoT edge devices, are constrained. After reserving the required resources for the core functionality of the gateway (operating system, networking, management tasks), the surplus of its resources is available for sharing among the connected IoT devices.

$\mathcal{R}$ is the vector showing the resources on the gateway that are available for sharing between devices (after deducting the resources required for management tasks). $\mathcal{R}[j]$ shows the total shareable amount of a certain resource type $j \in \{1, \cdots, g\}$ on the gateway, e.g. the communication bandwidth ($j=1$), the processing capability ($j=2$), and internal memory ($j=3$).

3.3 IoT device Model

We consider a total of N IoT edge devices, $\mathcal{D} = \{D_1, \cdots, D_N\}$, connected to the gateway as a local IoT network (see Figure 2). Each edge device is described by the following parameters:

- Device D_d, $1 \leq d \leq N$, has M_d operating modes (e.g. the sampling rates and offloading levels in Section 3.1). At any point in time, the device is operating at one mode $m \in \{1, \cdots, M_d\}$.
- $P_d(m)$ is the average power consumption of the device d in mode m. It includes the power consumption for sensing, on-board computation and communication[1]. We assume that the average power consumption of a mode does not change frequently. In practice, the changes in the device parameters (e.g. getting recharged, physical property of battery, ambient temperature, etc.) are neither severe nor sudden. Consequently, even a period of change in the parameters equal to tens of minutes would be a valid assumption.
- $R_d(m)$ denotes the resource vector and amount of each resource that the edge device d requires from the gateway under the operating mode m. $R_d(m)[j]$ is the amount of j-th resource on the gateway that is consumed or required while the device d is working under mode m. In our example, the gateway shares 3 different resource types (i.e. $g=3$), i.e. communication bandwidth ($j=1$), processing power ($j=2$), and internal memory ($j=3$). The required resources depend on the operational parameters of each mode (e.g. input data rate, service quality and computation offloading strategy of the IoT device).
- $u_d(m)$ is the utility of the device d when operating under mode m. The utility functions are provided by the system designer or application owner [32].
- B_d denotes the required battery lifetime for the device, e.g. 24 hours (i.e. until the next recharge or battery replacement).
- $e_d(t)$ shows the remaining energy in the battery of device d at time t ($t = 0$ when device is recharged or turned on).
- $b_d(t, m)$ is the estimated battery lifetime of device d at time t if it operates at mode m. At any given point t in time, the expected battery lifetime of the IoT device depends on (i) the remaining energy at time t and (ii) the average power consumption in mode m:

[1]The effect of surrounding devices (e.g. wireless interference) can be modeled in this parameter. Upon a change in packet loss, the device updates its communication cost. However, modeling the interference effect is beyond the scope of this work.

$$b_d(t,m) = \frac{e_d(t)}{P_d(m)} \tag{1}$$

It is worth noting that these parameters might change at runtime, however we assume that they do not change before one instance of the mode selection problem is solved.

3.4 Problem Statement

In general, our framework supports a wide class of optimization problems for the local networks of IoT edge devices in the form of max-min (or min-max) objective goals. The problem that is targeted in this article (as a proof-of-concept) is as follows. Let U_d denote the accumulated utility of the edge device d over time, $U_d = \sum(u_d(m) \times t_m)$, where t_m shows the total time that device d has operated in mode m so far. Now we need to select the operating mode for each IoT device d at runtime, such that the constraints of shared resources on the gateway (Eq. (3)) and on the devices (Eq. (4)) are fulfilled and the minimum of accumulated utility among devices is maximized (Eq. (2)).

Our optimization goal is to maximize the minimum accumulated utility among edge devices. Formally, the optimization problem at time t is as follows:

$$\text{Optimization goal:} \quad \text{maximize } \left(\min_{\forall d} U_d\right) \tag{2}$$

$$\text{Resources constraint:} \quad \sum_{\forall d} R_d(m_d) \leq \mathcal{R} \tag{3}$$

$$\text{Battery constraint:} \quad b_d \geq (B_d - t) \tag{4}$$

4 PROPOSED FRAMEWORK: OPTIMIZED OPERATION-MODE SELECTION (*OOPS*)

4.1 Problem Reformulation

The accumulated utility is calculated over time and therefore depends on the uptime. Imagine an operation mode with high utility but also high power consumption. Prioritizing this mode would result in early depletion of battery and less accumulated utility. Therefore to achieve the goal of optimization problem in Eq. (2), we define $\bar{u}_d(m) = \frac{u_d(m)}{p_d(m)}$, called *normalized utility*, and aim to maximize the minimum accumulated $\bar{u}$ among devices.

From hereon, we present our proposed solution and the properties of our problem along with an example. Let us assume our systems consists of three devices, $N = 3$, each of which has three operating modes representing *low quality* (and therefore low-power), *normal quality* and *high quality* which means $M_1 = M_2 = M_3 = 3$. For the sake of simplicity, in this example we assume that $\bar{u}_d(m) = m$ for $d, m = 1, 2, 3$.

4.2 Addressing the Constraints

4.2.1 Constraints of Devices. Each device is in charge of ensuring its constraints (i.e. battery lifetime). The rationale is that each device has all the required information to check its own constraints (e.g. power consumption of each operation mode, its residual energy, its battery model, etc.). Hence, the device can temporarily exclude those operation modes that violate its battery lifetime constraint. This also reduces and distributes the overhead of the gateway for checking the devices' constraints. Let T denote the time period after which the system solves the operation modes selection problem again and potentially changes the modes for IoT devices. Device D_d excludes its operation mode m if the following condition holds:

$$\frac{e_d(t) - T \times P_d(m)}{\min\limits_{1 \leq l \leq M_d} \{P_d(l)\}} < (B_d - t) - T \tag{5}$$

The condition indicates that when operating at mode m for duration T, then afterwards, even if working at the mode with the lowest power consumption, the device cannot meet its battery lifetime constraint anymore. After checking this constraint and excluding infeasible modes, devices send the information of the remaining operation modes including utility and resource usage of each mode to the gateway.

4.2.2 Constraints of Shared Resources on the Gateway. Once the gateway receives the parameters of the operation modes from the devices, it must select the mode for each device such that the constraints of the shared resources (e.g. bandwidth, memory, processing power of gateway) are met and the minimum accumulated utility among the devices is maximized.

Definition 1: System Configuration (SC) shows the operating mode of all devices as a vector of size N. For instance $SC = [1, 1, 1]$ is a system configuration where there are 3 IoT devices (i.e. $N = 3$) and all are operating at their first mode, i.e. low quality ($m_1 = m_2 = m_3 = 1$). $SC = [3, 3, 3]$ is another system configuration in which all three devices are operating at their third mode (i.e. high quality). The set of all possible SCs is the Cartesian product of the operation modes of devices.

$$\text{SC set} = \{1, \cdots, M_1\} \times \cdots \times \{1, \cdots, M_N\} \tag{6}$$

In our example, the set of SCs is:

$$\text{SC set} = \{[3, 3, 3], [3, 3, 2], [3, 2, 3], [2, 3, 3], \cdots, [1, 1, 1]\}$$

Definition 2: For resource vectors $\mathcal{R}_1$ and $\mathcal{R}_2$, we define $\mathcal{R}_1 \preceq_v \mathcal{R}_2$ if and only if for all types of resources $j = 1, \cdots, g$, it holds that $\mathcal{R}_1[j] \leq \mathcal{R}_2[j]$.

Definition 3: Feasible system configuration (FSC) is an SC which fulfills the resource constraints of the gateway. For instance the system configuration $[2, 3, 1]$ is an FSC if $R_1(2) + R_2(3) + R_3(1) \preceq_v \mathcal{R}$.

The set of FSCs is not known a priori and may change at runtime due to several reasons:

- The remaining energy in the battery may change by recharging or harvesting energy.
- The power consumption of devices at a certain mode may change over time. Particularly, the cost of data transmission does not only depend on the device, but also on the data transmission rate of other devices and external interference from surrounding wireless devices.
- The available resources on the gateway may change, e.g. due to the execution of other management tasks.

Hence, the set of FSCs needs to be determined and updated at runtime.

However, in general (as explained in Section 2), no single FSC would fulfill the optimization goal (maximizing the minimum variable). But the combination of particular FSCs and alternating between them would do. The rationale is that FSCs practically have unbalanced elements (as shown in Example 1 below). Hence having the set of FSCs, our problem is reduced to *"selecting and scheduling a subset of FSCs such that the minimum accumulated utility among the IoT devices is maximized"*.

4.3 Reduced Problem

We assume that T is the time interval during which the system state does not change. We refer to it as a *round*. For one round, we need to schedule the FSCs, i.e. determine the share of time T in which the system should operate at each FSC.

Let Q denote the number of FSCs ($Q \leq \Pi_{d=1}^{N} M_d$). Then, the array $C_{Q \times N}$ is the normalized utility of devices for the given FSCs and is formed as follow:

$$C = \begin{pmatrix} c_1 & c_2 & \cdots & c_Q \end{pmatrix}^T \quad c_i[d] = \bar{u}_d\big(FSC_i[d]\big) \qquad 1 \leq i \leq Q \quad 1 \leq d \leq N \tag{7}$$

Example 1: Let us assume that for our considered example, the FSC set has 6 SCs and equals $\{[3,2,1],[2,3,1],[2,1,1],[1,3,1],[1,2,3],[1,1,1]\}$, thus:

$$C = \begin{pmatrix} \bar{u}_1(3) & \bar{u}_2(2) & \bar{u}_3(1) \\ \bar{u}_1(2) & \bar{u}_2(3) & \bar{u}_3(1) \\ \bar{u}_1(2) & \bar{u}_2(1) & \bar{u}_3(1) \\ \bar{u}_1(1) & \bar{u}_2(3) & \bar{u}_3(1) \\ \bar{u}_1(1) & \bar{u}_2(2) & \bar{u}_3(3) \\ \bar{u}_1(1) & \bar{u}_2(1) & \bar{u}_3(1) \end{pmatrix} \tag{8}$$

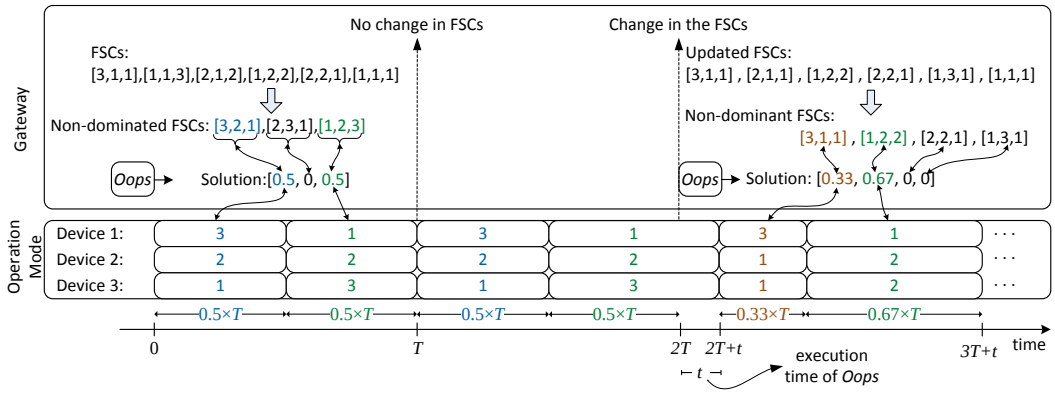


Fig. 3. An example scenario with 3 devices. Our framework, *Oops*, determines the efficient system configuration (efficient mode for edge devices) and the time interval for each of them.

The reduced problem is defined formally as follows:

$$maximize \quad \min_{d=1,\cdots,N} (U_d) \tag{9}$$

subject to:

$$[w_1 \ \cdots \ w_Q] \times (c_1 \dots c_Q)^T = [U_1 \ \cdots \ U_N] \tag{10}$$

$$\sum_{1}^{Q} w_i = 1 \tag{11}$$

$$0 \leq w_i \leq 1 \quad i = 1, \cdots, Q \tag{12}$$

where w_i –that we call the *weight factor* for each FSC– is needed to be determined by this optimization problem. The constraints of the optimization problem (Eq. (9) to (12)) are linear and the objective is convex [3]. Hence, it can be solved using the interior point methods such as Newton's method [33] which starts from an initial guess and converges to the solution after several iterations.

Once the weight factor (i.e. w_i) for each FSC is determined, the system schedules the configurations such that it operates at FSC_i for the time $T \times w_i$. At the end of each round, the problem needs to be solved again with the updated inputs. If there is no change in the inputs of the system (i.e. the same FSC set), it keeps operating with this schedule until a change in the system occurs (i.e. the FSC set changes). See Figure 3 for an example.

Example 2: The optimal solution to the problem in Example 1 is $w = [\frac{1}{2}, 0, 0, 0, \frac{1}{2}, 0]$. Therefore, the system should spend $\frac{T}{2}$ with configuration $[3, 2, 1]$, and $\frac{T}{2}$ with $[1, 2, 3]$. The accumulated utility will be $U_1 = \frac{3}{2} + \frac{1}{2} = 2$, $U_2 = \frac{2}{2} + \frac{2}{2} = 2$ and $U_3 = \frac{1}{2} + \frac{3}{2} = 2$. Figure 3 illustrates the timeline of the system and scheduled FSCs over time for the corresponding example.

4.3.1 Analysis and Properties of the Problem. The overhead of computation to select the efficient mode for edge devices, especially for large FSC set, is large. We present some properties of our problem and leverage them to reduce the computation overhead by shrinking the search space.

Definition 4: Domination Given two feasible system configurations, FSC_i dominates FSC_j, if and only if $\bar{u}_d(FSC_i[d]) \geq \bar{u}_d(FSC_j[d])$ for $d = 1, \cdots, N$. For instance in our example, $[3, 2, 1]$ dominates $[2, 1, 1]$ and $[1, 1, 1]$. None of $[3, 2, 1]$ and $[2, 3, 3]$ dominate the other. Moreover, all the FSCs can dominate $[1, 1, 1]$.

The dominated FSCs can be excluded from the problem without affecting the solution. In other words, the weight factor (w_i in equation Eq. (10)) for dominated FSCs is zero in the optimal solution. Note that some of SCs are excluded due to resource constraints of the gateway and battery constraints of devices (resulting in only FSCs). In other words, $FSC \neq SC$ in practice. Otherwise the solution is trivial as there is one SC which dominates all other elements ($[3, 3, 3]$ in our example) and would be the solution.

We illustrate the domination relation of SCs in our example as a tree in Figure 4. Each sub-tree is dominated by its root (shown by an arrow). Each child is different from its parent in the operating mode of only one device and only by one level lower. For instance, $SC = [3, 2, 3]$ is the child of $SC = [3, 3, 3]$ and only the operating mode of the second device is one level lower.

Starting from the third level of the tree, some SCs might appear in different sub-trees, i.e. duplicate nodes in the same level of the tree (shown with the same color). For instance, the $SC = [3, 2, 2]$ is a child of $SC = [3, 3, 2]$ as well as $SC = [3, 2, 3]$. Moreover, in level 4, there are 24 nodes from which only 7 are unique. For the sake of better visualization, at each level we first show all the child nodes, and then summarize them by only showing the unique (non-redundant) SC nodes to continue to the next level. At the bottom of the tree, there is the $SC = [1, 1, 1]$ which is dominated by all nodes from the upper levels.

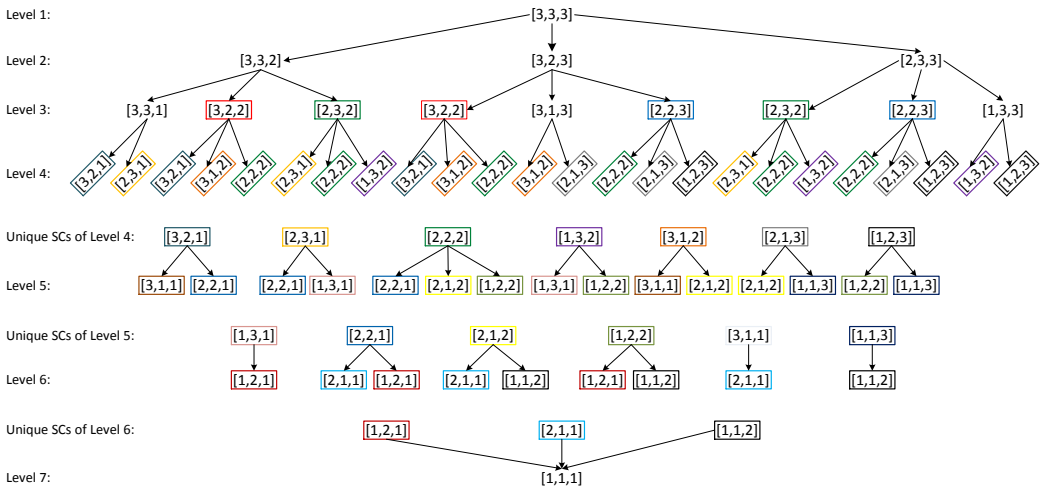


Fig. 4. An example of domination tree for three devices each of which has three operation modes.

4.4 Further Overhead Reduction

We introduce binary vectors to indicate if an SC is FSC or not. We name this vector V-FSC. The size of this vector equals the number of all possible SCs (i.e. $Q = \Pi_{d=1}^{N} M_d$). If an SC is FSC (i.e. it fulfills the constraints of the gateway and the battery lifetime of the device), then the value of its corresponding element in V-FSC is '1', otherwise '0'. We introduce V-DSC, another binary vector of

size Q, to indicate if an FSC is dominated and excluded (='0') or is not dominated and therefore remained (='1'). Formally, the V-DSC[q]=1 if the V-FSC[q]=1 **and** none of its dominators is '1':

$$\begin{aligned} \text{V-DSC}[q] = \text{V-FSC}[q] \ \wedge \ \neg\Big(\text{V-FSC}[k_1] \vee \cdots \vee \text{V-FSC}[k_I]\Big) \\ \forall\ \text{FSC}_{k_1}, \cdots, \text{FSC}_{k_I} \text{ dominate } \text{FSC}_q \end{aligned} \tag{13}$$

As an example of domination with 6 FSCs, $\{[3,2,1],[2,3,1],[2,1,1],[1,3,1],[1,2,3],[1,1,1]\}$, according to Def. 4, $[2,1,1]$, $[1,3,1]$ and $[1,1,1]$ are dominated by other FSCs. Therefore, the corresponding element of these FSCs are changed to '0' in the V-DSC vector. Consequently, the size of the C matrix in Eq. (8) is reduced from 6 rows to 3 rows.

Some of the SCs are dominated by several other SCs. Hence, the number of dominators in Eq. (13), i.e. k_I, might be large. For instance, in our example $SC = [1,1,1]$ is dominated by all other SCs. Therefore, the domination function contains a large NOR operand with $k_I = 26$ inputs. This value increases exponentially with the number of devices and modes. For instance, if the number of devices increases to 4 in the previous example, the size of the NOR function in the domination relation for $SC = [1,1,1,1]$ increases to $k_I = 80$. The domination functions can be implemented in hardware or software. In either case, having too many inputs for the NOR operation is inefficient. Using the following property we can reduce the complexity of the domination function for each V-DSC element.

Property 1: The dominance relation for SCs is transitive: If SC_1 dominates SC_2, and SC_2 dominates SC_3, then SC_1 dominates SC_3 as well. The proof follows immediately from Def. 4. Using this property, we can reduce the complexity of finding the dominant SCs as follows: we only check SCs that are only one level above the SC in the domination tree (i.e. immediate parents). For example in Figure 4, the immediate parents of $SC = [1,1,2]$ are $SC = [1,1,3]$, $SC = [1,2,2]$ and $SC = [2,1,2]$. Therefore, we only consider these SCs to calculate the V-DSC. After applying Property 1, the maximum size of the NOR function is reduced to the number of devices (i.e. N). For instance, the size of the NOR function for the domination relation of $SC = [1,1,1,1]$ is reduced from $k_I = 80$ to 4.

Given the number of devices and their number of operation modes, the domination relation can be determined at design time (i.e. offline). However at runtime, the V-FSC vector changes according to the status of the devices (e.g. residual energy, available resources on the gateway, etc.). Once the V-FSC vector changes, we update the V-DSC vector to obtain the dominated SCs. After the feasible SCs are reduced to the DSCs, we can solve the optimization problem in Eq. (9) to (12) using a smaller C matrix with less rows (see Section 5 and Figure 7a for the detailed results).

After reducing the problem size, we incorporate a *Sequential Least SQuares Programming (SLSQP)* method based on [15] which is a gradient-based numerical optimization solver for constrained problems to solve Eq. (9) to (12).

4.5 Complexity Analysis of *Oops*

The proposed reductions in *Oops* come at the cost of memory overhead to keep the domination relations. The number of unique SCs in the domination tree is $\Pi_{d=1}^{N} M_d$. As described in Section 4.4, we only need to keep at most N dominators for each node of tree. Assuming that the index of each node requires 2 Bytes, and all the IoT devices have M operation modes, the required memory to keep the domination relations is less than $M^N \times N \times 2Byte$. If a gateway serves 10 devices with 3 operation modes it would require 1.1 MB of memory to store the domination relations. This is less than 0.5% of the available memory of typical IoT gateways, e.g. *Intel Quark* [2] and *Raspberry Pi*.

It is important to note that in the envisioned hierarchical architecture of IoT (see Figure 1), the number of IoT edge devices connected to a single gateway is bounded. The reasons lie on practical limitations or application properties: 1) In typical applications, spatial density of the IoT devices is

limited. For instance, the number of wearable IoT devices carried by a user cannot grow arbitrarily. 2) The limited capabilities of a gateway do not allow it to serve large numbers of connected devices. For instance, the communication bandwidth or computation competency of the gateway is only enough to support a couple of devices [11, 40]. 3) Some existing IoT technologies limit the gateway from serving many IoT devices, e.g. Bluetooth protocol supports up to 7 devices connected to a single transceiver. Therefore, the problem size for our operation mode selection does not grow, and hence, the absolute overhead of the solution is more important than its complexity (i.e. order of the function). In other words, the overhead of the solution in the range of typical problem sizes matters more than the growth rate.

4.6 Memoization and Caching

According to our observations (see Section 5), some input vectors occur repeatedly. We exploit a memoization scheme to store the results of our optimization solver to avoid the expensive computation when the same inputs occur again. The results associated with each input are stored in a look-up table. The subsequent optimization solver calls retrieve and reuse the stored results, thus avoiding repeating computation. The memoization scheme reduces the execution time for finding the efficient solution, at the cost of memory usage.

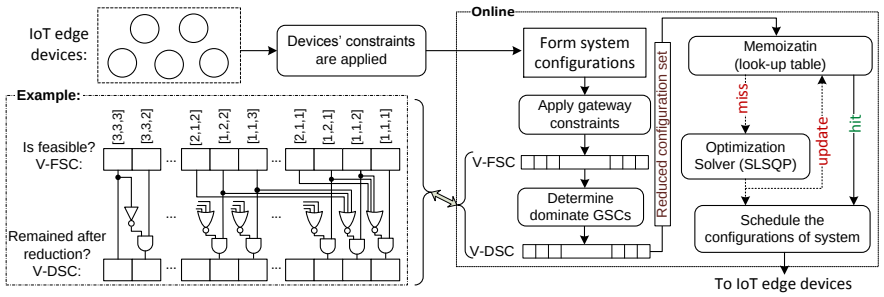


Fig. 5. The overview of our proposed framework

Figure 5 illustrates the overview of the proposed framework, *Oops*. After applying the devices and gateway constraints, the vector V-DSC is derived based on the domination relation and V-FSC. If the look-up table contains the solution to the vector (i.e. memory hit), the associated solution is used for the schedule. If there is no solution entry for this vector in the look-up table (i.e. memory miss), the optimization solver (using SLSQP) is executed to calculate the schedule. This solution is then stored in the look-up table for subsequent calls.

5 EVALUATION AND RESULTS

5.1 Experimental Setup

Our experimental analysis includes measurements on the actual IoT devices and trace-driven network simulation to study the behavior of the whole system under our proposed mechanism. The profiling of the IoT applications for edge devices is performed on an *Intel Quark* SoC, which has already been proposed and used for wearable IoT devices. For the gateway, we use a *Raspberry Pi Zero* (RPi0) module.While we implement the task of gateway and *Oops* on our RPi0 module, the network-level behavior of the system and the interaction between IoT edge devices and the gateway is modeled in a network simulator. While a physical deployment of IoT network is feasible from application's perspective, the evaluation of battery lifetime is a challenge due to the following reasons: 1) Using batteries in the practical setup restrains the capability to replicate the experiments and compare different techniques. 2) The vast majority of IoT platforms come as development

boards, e.g. the *Intel Quark* SoC is available on the *Intel Galileo* board. These boards include many peripherals and components that not all of them will be available on the final products. Even though the power consumption of a processor can be *measured* on the board individually, other components still will drain a considerable share of the total power from battery. 3) Using batteries in the practical setup will severely complicate the development and debugging of the system. These reasons led us to the use of a simulated network environment, augmented with the information of profiled applications.

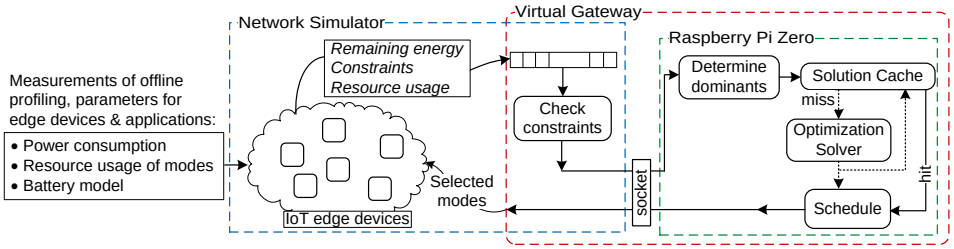


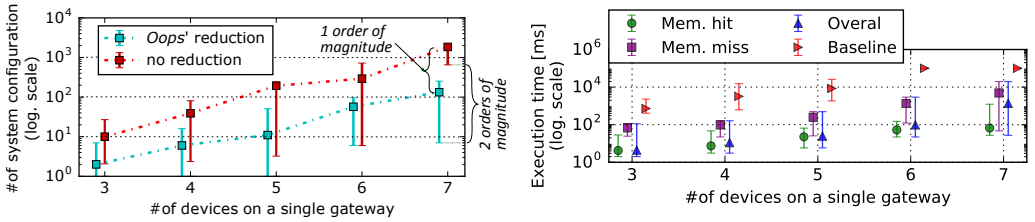
Fig. 6. The overview of our setup for network-level simulation

Figure 6 shows the overview of our setup. We use the profiled (measured) data from the IoT devices, their application and their operation modes, and then form a local network in our simulator to capture the dynamic behavior of the system including the energy consumption of devices over time, discharging of the batteries, etc. The tasks of the gateway are implemented partially on the simulator, but the optimization framework (i.e. *Oops*) runs on the RPi0 hardware module. Bluetooth protocol is considered along with TCP/IP for the communication model. The communication latency is modeled according to relevant models in literature [31], as well as in-house experimental evaluations [23]. The frequency-hoping technique of Bluetooth minimizes the interference between the IoT nodes and other wireless devices. We examine a varying number of connected IoT devices, ranging from 3 to 7. *Oops* needs to be executed only when run-time conditions (e.g. battery reduction) result in changes in V-FSC vector (see Section 4.3). The system simulation continues until the battery of all IoT devices is depleted.

5.2 Results

5.2.1 Overhead Analysis. The first set of experiments focuses on the properties of *Oops* on the gateway, in order to investigate the effectiveness of its components as well as their associated overhead. Figure 7a shows the total number of feasible system configurations (FSCs) compared to the non-dominated FSCs (logarithmic scale) for different number of devices. The former is the problem size before applying our reduction techniques, while the latter is the problem size in *Oops* (after excluding the dominated SCs). *Oops* reduces the number of FSCs (i.e. problem size) by up to 2 orders of magnitude. In the average case, the reduction shows 1 order of magnitude improvement.

Figure 7b shows the execution time overhead of the baseline, which does not benefit from our reduction techniques compared to *Oops*. For *Oops* we show the overhead in the following cases: (i) the solution is previously found and stored, i.e. memory hit, (ii) the solution needs to be computed and then stored, i.e. memory miss, and (iii) overall which shows the average including both memory hit and miss cases. The Figure illustrates the minimum, maximum and average execution time for each case in the experiments. On average, the execution time of *Oops* is not longer than 1.3 s while the baseline (without applying dominance and transitive reduction) takes more than 100 s (we terminate the execution after this time limit and consider it as timeout). These results are for the devices with 3 operation modes. If the number of operation modes increases, the difference between overhead of *Oops* and baseline is expected to further increase.



(a) Maximum, minimum and average number of total FSCs without reduction compared to *Oops*' reduction technique

(b) Maximum, minimum and average of execution time of *Oops* for memory hits, memory misses, and in overall

Fig. 7. The problem size and *Oops*' overhead with respect to the number of connected devices

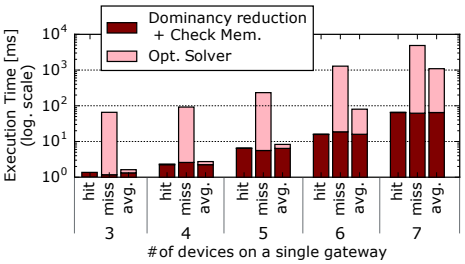


Fig. 8. The breakdown of *Oops*' overall execution time: checking dominancy, reduction of problem size, look-up table operation (index calculation, checking table), and optimization solver (in case of memory misses)

Figure 8 illustrates the breakdown of *Oops*' overall execution time. *Oops* consists of three main steps: 1) reduction using dominancy, 2) checking look-up table which includes calculating the index of the problem and then searching the look-up table for the solution, and 3) the optimization solver in case the solution has not been found in the look-up table. Note that the Y-scale is logarithmic. The optimization solver accounts for most of the execution time, which confirms again that having the memoization is beneficial.

In this use-case the number of connected devices to the gateway may change at runtime, e.g. when a user enters or leaves a room. When a device is removed from the network, the domination tree for the remaining devices is a sub-tree of the previous tree. When a new IoT device is added to the network, the existing domination tree will be a sub-tree of the new one. However, reusing the existing domination trees would make *Oops* complicated, and therefore we determine the new domination tree based on the new setup. According to our experiment, the overhead of *Oops* to determine the domination relations is less than 1.2 s on average.

5.2.2 Overhead of Operation Mode Switching. Once the device is notified to change its operation mode, at least one of these two parameters changes: 1) service quality due to quality of input, 2) offloading level. Both changes are performed in software and can be implemented in such a way that do not require lengthy reconfiguration of the device. Service quality is tightly coupled to the sampling frequency of the input signal, which only requires alternating the expiration interval of a timer. The small number of different service quality levels allows the hard-coding of system parameters for the rest of execution flow, e.g. the coefficients of the digital filter for the pre-processing stage. Consequently, the change of service quality corresponds to the dynamic choice of a different execution flow, which is implemented in a "switch case" manner. Taking these

facts into account, the dynamic adaptation of service quality imposes negligible overheads both to the performance and power of the IoT device.

The second case, i.e. a dynamic adaptation of the offloading level is even more straightforward. Again in software, this corresponds to changing a condition variable, which designates the exact processing stages that needs be executed on the IoT device. After a certain stage (which is determined by the offloading level), the intermediate results are transmitted to the gateway and the rest of processing is performed there [28]. Therefore, the overhead for the change of an offloading level is negligible (i.e. checking an "if-else" condition at the end of each stage) both in performance and power. Note that the power cost of processing and transmitting the data is already taken into account when the operating mode is designated (i.e. $P_d(m)$ in Section 3.3).

Lastly, the new operation mode of the device is effective for the next segment of input data. This means that the processing of the current segment must be completed first before the mode adaptation takes place [25]. In practice, this delay is constrained by its worst case, i.e. the execution latency for processing of an entire segment. Even for computation intensive applications (e.g. Epileptic seizure prediction [27]) this latency is very small, and typically is in the order of hundreds of milliseconds. Due to the aforementioned design aspects, the overhead for changing the operation mode of application is negligible. Moreover in our experiments, the number of times that the operation mode of devices change is small compared to their expected lifetime (18 times in 950 min.).

5.2.3 Battery Lifetime, QoS & Utility improvement. Moving to a complete system perspective, we exploit a use-case to present runtime effectiveness of *Oops*. We consider a scenario revolving around a future smart home (presented in Figure 9), where a number of IoT devices operate within the premises of this smart home. We exclude smart household appliances that are plugged into the power network and thus do not have battery constraints. On the contrary, we examine portable devices that are used in an assisted living setup for elderly (e.g. wearable healthcare monitoring devices, fall detection devices, smart watches, etc.). Each device senses its surroundings, performs a series of processing tasks on the input (e.g. filtering, features extraction, classification), and eventually assesses the status of its monitoring environment. This structure has been seen in other IoT applications such as the IoT-based ECG monitoring application proposed and utilized in [29].

The last key component of the smart home system is the gateway, which is responsible for instructing the operation of devices in such a way that the provided utility to the user is maximized, while the expected lifetime constraints of each individual device are respected. As detailed in Section 3.1, each individual IoT device is characterized by its operation modes each of which uses a different computation offloading level to the gateway, different service quality, and different utility for the user. Furthermore, the user dictates the expected battery lifetime for each device. The objective of our proposed *Oops* is to dynamically designate the operation mode of the devices with respect to the runtime changes in available battery, number of devices, user requirements, etc.

The gateway is characterized by a limited amount of computational and communication resources, expressed by its CPU and bandwidth capacity. The number of connected IoT devices to the gateway ranges from 3 to 7. This range is typical for a local IoT network with a single gateway (e.g. Bluetooth supports up to seven different slave devices). The utility values are chosen from the range of $[0-1]$ using an iso-elastic model (with diminishing marginal returns), which is widely used to model the utility functions in resource allocation problems [9].

Since the expected battery lifetime of IoT devices is the crucial runtime constraint, we examine three scenarios for lifetime expectations in order to evaluate the efficiency of our proposed framework. These scenarios assume lifetime constraints in the range of *all day battery life* with respectively 950, 1050, and 1150 minutes. When the expected lifetime of IoT nodes is high, the intensity of the constraints in the optimization problem increases. Thus the operation mode selection

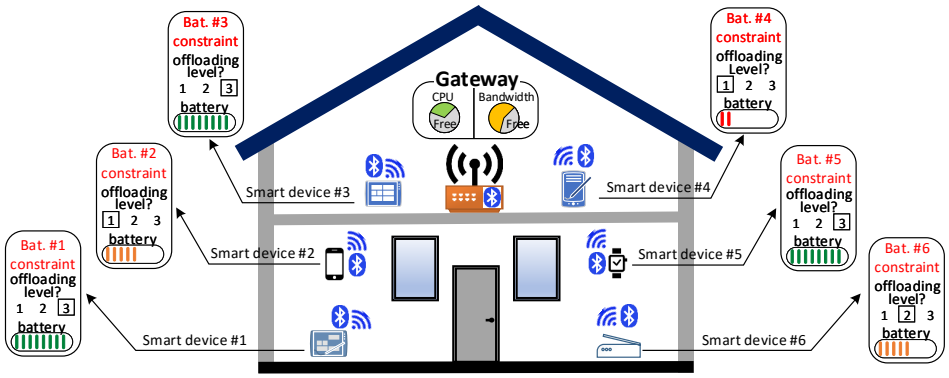


Fig. 9. Envisioned future Gateway-governed Smart Home setting

becomes more critical as it must determine the efficient operating modes for devices in order to preserve the energy more.

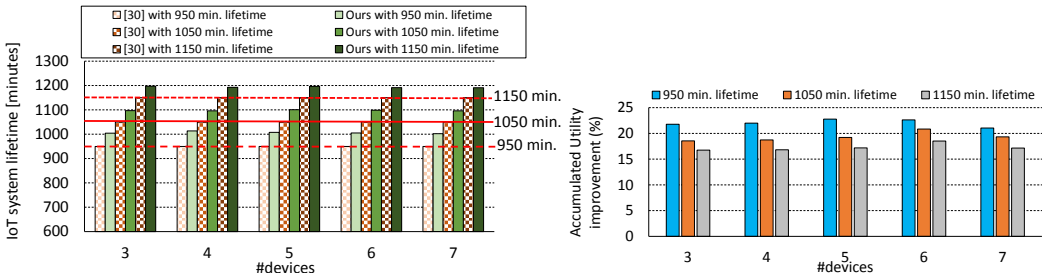
Furthermore, we consider a CR2430 coin-cell battery as the power source of devices as it is already used in other IoT products [41]. Each IoT device has a different initial battery capacity, randomly sampled in the range of [280, 320] mAh according to [1, 4]. For fairness of comparison, the initial battery capacity conditions are maintained constant throughout the examined scenarios.

In Figure 10a, we present the minimum achieved lifetime of the IoT devices for 3 different lifetime expectations. The expected lifetime of IoT devices is depicted using dashed red lines. The effectiveness of *Oops* is compared against a similar setup, but the algorithm for runtime operation mode selection of IoT devices is based on the algorithm presented in [29], where the operation modes with the highest utility are prioritized. Results show that both algorithms are capable of satisfying the expected lifetime constraints for all cases (different examined numbers of IoT devices and lifetime requirements). However, *Oops* increases the lifetime of devices further (longer than the constraint) by prioritizing the operation modes that offer higher utility at the cost of a given energy consumption value. Execution of *Oops* accounts for less than 1% of the achieved lifetime rounds for 3-5 devices and less than 2% of the rounds for scenarios of 6 and 7 devices.

The advantage of *Oops* is that it extends battery lifetime providently by prioritizing operation modes with higher utility per energy. Hence by extending the lifetime, it increases the accumulated utility of the system. Figure 10b illustrates the improvement of accumulated utility in *Oops* compared to the state-of-the-art [29]. When the expected lifetime constraints are low (short expected lifetime), the optimization algorithm has the flexibility to choose between more operation modes and as a consequence, utility improvements surpass 20% compared to the [29]. Hence, the advantage of *Oops* is more prominent in this scenario.

Tighter expected lifetime requirements result in less flexibility for the mode selection algorithms to instruct the IoT devices, i.e. there are few options for selection that can still meet battery lifetime expectation. This affects the utility improvement over the state-of-the-art (see Figure 10b). The lowest accumulated utility improvement belongs to the last scenario where battery lifetime expectation is high (i.e. 1150 minutes in our experiment). This is because there are very few feasible configurations for each device, since these operation modes are more energy consuming and their adoption would result in violation of battery lifetime constraint.

The achieved aggregated utility usually decreases as the number of connected devices increases. The reason is that due to higher resource contention on the gateway, the operation modes with less utility will be selected. In our experiment, the aggregated utility in *Oops* and state-of-the-art [29] decreases when the number of connected devices increases. However, the proportion remains



(a) Lifetime of examined IoT systems for different constraints

(b) Utility improvement over [29] for different lifetime constraints

Fig. 10. The battery lifetime and utility improvement of *Oops* compared to [29]

almost the same. Therefore, the utility improvement that is reported in Figure 10b changes slightly when increasing the number of connected devices.

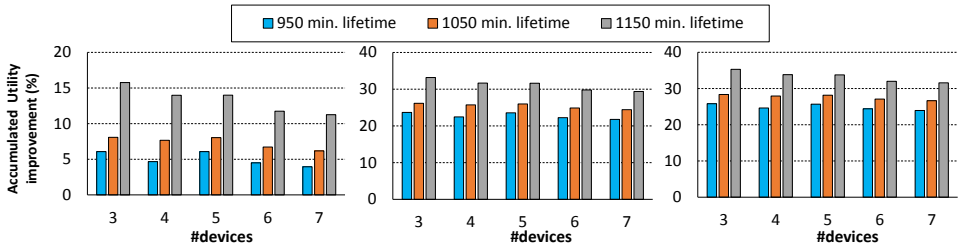
Figure 10b also provides another observation and insight about the relation of application's requirements and defining operation modes at the design time; Application requirements such as battery lifetime constraint, service quality constraint, etc. restrain the range of acceptable input quality, offloading levels, etc. These requirements are often contradictory. They form a design space from which the operation modes must be selected wisely. Defining efficient operation modes for the application at design time is still an open problem and requires more attention.

It must be noted that the operation modes do not necessarily need to meet all system's requirements *individually*. The reason is that the operation mode selection algorithm, e.g. *Oops*, are able to combine and schedule the operation modes at runtime such that all the requirements are met eventually. However, the operation modes must be defined –at design time– in a way that could provide a rich portfolio and offer enough options to *Oops*. In other words, the operation mode selection algorithms need to have the flexibility in choosing the feasible modes at runtime. Otherwise (if the feasible options are few) the advantage of *Oops* descends when the constraints and requirement are tight (e.g. the gray bar in Figure 10b).

5.3 Comparison with Cloud-based Solutions

To compare *Oops* with a cloud-based solution we consider a setup where IoT devices are connected to the cloud server without the assistance of a gateway. The IoT devices are equipped with broadband cellular network transceivers such as 2G [35], 3G [36] and 4G-LTE1 [34] (*u-blox* modules) that provide high-bandwidth and direct connection to the Internet. The bandwidth of the transceivers is adequate for the executed applications.

Figure 11 shows the achieved utility with *Oops* compared to the cloud-based solution. The IoT devices offload all tasks to the cloud instead of a local gateway. The power consumption of data transmission using cellular transceivers is higher compared to the Bluetooth in gateway-based setup of *Oops*. The energy consumption overhead is calculated according to profiling of IoT applications [29] and the power consumption profile of the transceivers as provided by their manufacturer [34–36]. Hence to meet the battery lifetime requirements, the cloud-based approach must set the IoT devices to operate at a mode with lower utility (e.g. lower input quality). This effect is more crucial in newer generation of cellular networks, where power consumption is higher to provide more bandwidth and reliability. Consequently, the advantage of *Oops* over cloud-based approach is more prominent in 3G and 4G. *Oops* shows 17.5%, 20% and 26% improvement in accumulated utility on average for the cases of low, medium and high lifetime expectancies, respectively.



(a) Connect to Cloud by 2G (b) Connect to Cloud by 3G (c) Connect to Cloud by 4G

Fig. 11. The utility improvement of *Oops* compared to cloud-based solution

6 CONCLUSIONS

While the task of data processing is being pushed to the edge of the IoT network, efficient techniques are required to manage the limited and shared resources of low-power devices and gateways. In this article, we propose an optimization framework to orchestrate the IoT devices and manage the constrained shared resources at the edge. It determines the efficient operation mode of IoT devices at runtime with respect to the devices' constraints and gateway's shared resources. This framework supports a wide range of optimization goals in the form of max-min (or min-max). Experimental results show a significant reduction in runtime overhead of operation-mode selection from more than 100 s to only 1.3 s on average. *Oops* also increases the user utility by more than 15% compared to the state-of-the-art.

REFERENCES

[1] [n. d.]. Coin Cell / Button Cell Battery Guide. Online: https://www.batteries.com/pages/coin-cell-button-cell-battery-guide, Visited on 15.March.2018. ([n. d.]).

[2] [n. d.]. Intel IoT Gateway. Online: http://www.intel.de/content/dam/www/public/us/en/documents/product-briefs/gateway-solutions-iot-brief.pdf. ([n. d.]). Visited on April 3, 2016.

[3] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex optimization*. Cambridge university press.

[4] Energizer Brands. [n. d.]. Lithium Coin, Handbook and Application Manual. Online: http://data.energizer.com/pdfs/lithiumcoin_appman.pdf, Visited on 15.March.2018. ([n. d.]).

[5] Luca Catarinucci, Danilo De Donno, Luca Mainetti, Luca Palano, Luigi Patrono, Maria Laura Stefanizzi, and Luciano Tarricone. 2015. An IoT-aware architecture for smart healthcare systems. *IEEE Internet of Things Journal* 2, 6 (2015).

[6] Xu Chen. 2015. Decentralized computation offloading game for mobile cloud computing. *IEEE Transactions on Parallel and Distributed Systems* 26, 4 (2015), 974–983.

[7] Mung Chiang and Tao Zhang. 2016. Fog and IoT: An overview of research opportunities. *IEEE Internet of Things Journal* 3, 6 (2016), 854–864.

[8] Soumya Kanti Datta, Christian Bonnet, and Navid Nikaein. 2014. An IoT gateway centric architecture to provide novel M2M services. In *World Forum on Internet of Things (WF-IoT'14)*. 514–519.

[9] Mostafa Dehghan, Laurent Massoulie, Don Towsley, Daniel Menasche, and Yong Chiang Tay. 2016. A utility optimization approach to network cache design. In *International Conference on Computer Communications (INFOCOM'16)*. 1–9.

[10] Kai-Wei Fan, Zizhan Zheng, and Prasun Sinha. 2008. Steady and fair rate allocation for rechargeable sensors in perpetual sensor networks. In *ACM Embedded network sensor systems (SenSys'08)*. 239–252.

[11] Tuan Nguyen Gia, Mingzhe Jiang, Amir-Mohammad Rahmani, Tomi Westerlund, Pasi Liljeberg, and Hannu Tenhunen. 2015. Fog computing in healthcare Internet of Things: A case study on ECG feature extraction. In *International Conference on Computer and Information Technology*. 356–363.

[12] Wendi B Heinzelman, Amy L Murphy, Hervaldo S Carvalho, and Mark A Perillo. 2004. Middleware to support sensor network applications. *IEEE Network* 18, 1 (2004), 6–14.

[13] Pengfei Hu, Sahraoui Dhelim, Huansheng Ning, and Tie Qiu. 2017. Survey on fog computing: architecture, key technologies, applications and open issues. *Journal of Network and Computer Applications* (2017).

[14] Dejan Kovachev, Tian Yu, and Ralf Klamma. 2012. Adaptive computation offloading from mobile devices into the cloud. In *International Symposium on Parallel and Distributed Processing with Applications*. IEEE, 784–791.

[15] D Kraft and K Schnepper. 1989. SLSQP-A Nonlinear Programming Method with Quadratic Programming Subproblems. *DLR, Oberpfaffenhofen* (1989).

[16] Jeongho Kwak, Yeongjin Kim, Joohyun Lee, and Song Chong. 2015. DREAM: dynamic resource and task allocation for energy minimization in mobile cloud systems. *IEEE Journal on Selected Areas in Communications* 33, 12 (2015).
[17] SangJoon Lee, Jungkuk Kim, and Myoungho Lee. 2011. A real-time ECG data compression and transmission algorithm for an e-health device. *IEEE Transactions on Biomedical Engineering* 58, 9 (2011), 2448–2455.
[18] Wei Li, Flávia C Delicato, Paulo F Pires, Young Choon Lee, Albert Y Zomaya, Claudio Miceli, and Luci Pirmez. 2014. Efficient allocation of resources in multiple heterogeneous Wireless Sensor Networks. *J. Parallel and Distrib. Comput.* 74, 1 (2014), 1775–1788.
[19] Jie Lin, Wei Yu, Nan Zhang, Xinyu Yang, Hanlin Zhang, and Wei Zhao. 2017. A survey on Internet of Things: architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal* (2017).
[20] Qicheng Ma, David C Parkes, and Matthew D Welsh. 2007. A utility-based approach to bandwidth allocation and link scheduling in wireless networks. In *International Workshop on Agent Technology for Sensor Networks (ATSN'07)*.
[21] Yuyi Mao and Jun Zhang. 2016. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE Journal of Solid-State Circuits* 51, 3 (2016), 712–723.
[22] Peter Marbach. 2002. Priority service and max-min fairness. In *International Conference on Computer Communications (INFOCOM'02)*. 266–275.
[23] Dimosthenis Masouros, Ioannis Bakolas, Vasileios Tsoutsouras, K Siozior, and Dimitrios Soudris. 2017. From edge to cloud: Design and Implementation of a Healthcare Internet of Things Infrastructure. In *International Workshop on Power And Timing Modeling Optimization and Simulation (PATMOS'17)*. 1–6.
[24] Chris Raphael. 2016. Why Edge Computing Is Crucial for the IoT. Online: http://www.rtinsights.com/why-edge-computing-and-analytics-is-crucial-for-the-iot/. (2016).
[25] Farzad Samie. 2018. *Resource Management for Edge Computing in Internet of Things (IoT)*. Ph.D. Dissertation. Karlsruhe Institute of Technology (KIT).
[26] Farzad Samie, Lars Bauer, and Jörg Henkel. 2016. IoT Technologies for Embedded Computing: A Survey. In *CODES+ISSS*.
[27] Farzad Samie, Sebastian Paul, Lars Bauer, and Jörg Henkel. 2018. Highly Efficient and Accurate Seizure Prediction on Constrained IoT Systems. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'18)*.
[28] Farzad Samie, Vasileios Tsoutsouras, Sotirios Xydis, Lars Bauer, Dimitrios Soudris, and Jörg Henkel. 2016. Computation Offloading and Resource Allocation for Low-power IoT Edge Devices. In *World Forum on Internet of Things (WF-IoT'16)*.
[29] Farzad Samie, Vasileios Tsoutsouras, Sotirios Xydis, Lars Bauer, Dimitrios Soudris, and Jörg Henkel. 2016. Distributed QoS Management for Internet of Things under Resource Constraints. In *CODES+ISSS*.
[30] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* (2016).
[31] Matti Siekkinen, Markus Hiienkari, Jukka K Nurminen, and Johanna Nieminen. 2012. How low energy is bluetooth low energy? comparative measurements with ZigBee/802.15.4. In *IEEE Wireless Communications and Networking Conference Workshops (WCNCW'12)*.
[32] Liansheng Tan, Zhongxun Zhu, Fei Ge, and Naixue Xiong. 2015. Utility maximization resource allocation in wireless networks: Methods and algorithms. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45, 7 (2015), 1018–1034.
[33] Tamás Terlaky. 2013. *Interior point methods of mathematical programming*. Vol. 5. Springer Science & Business Media.
[34] u blox. [n. d.]. LARA-R2 Data Sheet. Online: https://www.u-blox.com/sites/default/files/LARA-R2_DataSheet_(UBX-16005783).pdf, Visited on 1.April.2018. ([n. d.]).
[35] u blox. [n. d.]. SARA-G3 Data Sheet. Online: https://www.u-blox.com/sites/default/files/SARA-G3_DataSheet_(UBX-13000993).pdf, Visited on 1.April.2018. ([n. d.]).
[36] u blox. [n. d.]. SARA-U2 Data Sheet. Online: https://www.u-blox.com/sites/default/files/SARA-U2_DataSheet_(UBX-13005287).pdf, Visited on 1.April.2018. ([n. d.]).
[37] Felix Ming Fai Wong, Carlee Joe-Wong, Sangtae Ha, Zhenming Liu, and Mung Chiang. 2015. Improving user QoE for residential broadband: Adaptive traffic management at the network edge. In *International Symposium on Quality of Service (IWQoS'15)*. 105–114.
[38] Changjiu Xian, Yung-Hsiang Lu, and Zhiyuan Li. 2007. Adaptive computation offloading for energy conservation on battery-powered systems. In *International Conference on Parallel and Distributed Systems*, Vol. 2. 1–8.
[39] Thomas Zachariah, Noah Klugman, Bradford Campbell, Joshua Adkins, Neal Jackson, and Prabal Dutta. 2015. The Internet of Things Has a Gateway Problem. In *Mobile Computing Systems and Applications (HotMobile'15)*. 27–32.
[40] Ben Zhang, Nitesh Mor, John Kolb, Douglas S Chan, Nikhil Goyal, Ken Lutz, Eric Allman, John Wawrzynek, Edward Lee, and John Kubiatowicz. 2015. The cloud is not enough: saving IoT from the cloud. In *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'15)*. 21–21.
[41] Haibin Zhang, Jianpeng Li, Bo Wen, Yijie Xun, and Jiajia Liu. 2018. Connecting Intelligent Things in Smart Hospitals using NB-IoT. *IEEE Internet of Things Journal* (2018).

Received December 2017; revised April 2018; accepted May 2018