Distributed QoS Management for Internet of Things under Resource Constraints

Farzad Samie[1] , Vasileios Tsoutsouras[2] , Sotirios Xydis[2] , Lars Bauer[1] , Dimitrios Soudris[2] , Jörg Henkel[1]

[1] Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany
[2] Microprocessors and Digital Systems Laboratory, ECE, National Technical University of Athens, Greece
{samie, bauer, henkel}@kit.edu, {billtsou, sxydis, dsoudris}@microlab.ntua.gr

ABSTRACT

Internet-of-Things (IoT) envisions an infrastructure of ubiquitous networked smart devices offering advanced monitoring and control services. Current art in IoT architectures utilizes gateways to enable application-specific connectivity to IoT devices. In typical configurations, an IoT gateway is shared among several IoT devices. However, given the limited available bandwidth and processing capabilities of an IoT gateway, the quality of service (QoS) of IoT devices must be adjusted over time not only to fulfill the needs of individual IoT device users, but also to tolerate the QoS needs of the other IoT devices sharing the same gateway.

In this paper, we address the problem of QoS management for IoT devices under bandwidth, battery, and processing constraints. We first formulate the problem of resource-aware QoS tailored to the IoT paradigm and then propose an efficient problem decomposition that enables the adoption of a recurrent dynamic programming approach with reduced execution time overhead. We evaluate the efficiency of the proposed approach with a case study and through extensive experimentation over different IoT system configurations regarding to the number and type of the employed IoT-devices. Experiments show that our solution improves the overall QoS by 50% compared to an unsupervised system while both meet the constraints.

Keywords

Internet of Things, IoT, Resource Management, Computation Offloading, ECG.

1 Introduction

The Internet of Things (IoT) is a novel paradigm in which many of the objects that surround us, such as sensors, actuators, smartphones, and other smart devices, will be networked and connected to the Internet to offer better services in different domains including healthcare monitoring, automotive, smart buildings, etc. [1, 2, 3, 4].

Recent advancements in technology, emerging techniques in embedded systems, wireless communication, and sensors have enabled the design of small-size, ultra-low power, and low cost IoT devices [4]. They sense information from physical phenomena and send the data after pre-processing to a gateway node, which aggregates the streams of sensed data in real-time and sends them to a central server, e.g. a cloud server [5, 6].

CODES/ISSS '16, October 01-07, 2016, Pittsburgh, PA, USA

 ISBN 978-1-4503-4483-8/16/10…$15.00

DOI: http://dx.doi.org/10.1145/2968456.2974005

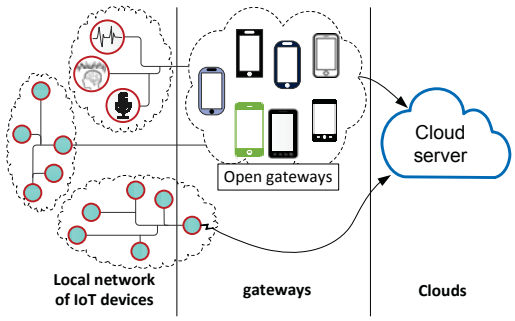

Figure 1: Local networks of IoT devices connected to the Internet via gateways

IoT envisions a model in which the increasingly ubiquitous portable smart devices (e.g. smartphones [7]) provide gateway services [8] , as illustrated in Figure 1. Other IoT devices are able to provide gateway service, too. Especially with the existence of small-size, low-power multi-radio chips which provide multiple communication technologies (e.g. WiFi, Bluetooth, ZigBee, etc.) on a single chip, IoT devices can provide gateway service, if necessary.

The battery-powered IoT devices need to fulfill a specific expected lifetime before the next recharge takes place. To manage the energy consumption under the battery lifetime constraints, each IoT device has generally two control knobs: 1) changing the quality of service (QoS) and 2) changing the on-board processing policy. For instance, in an IoT-based health monitoring device that captures and transmits an Electrocardiogram (ECG) signal to the Internet through a smartphone, when the battery level is low, the device can reduce the sampling rate of data acquisition from 360 Hz to 250 Hz. Alternatively (or conjointly) it can stop performing digital filter for baseline wandering removal [9], and offload it to the gateway. However, the gateway (smartphone) is battery-powered with limited energy sources too. It has limited resources for receiving data and limited processing capability to perform the offloaded computations. Depending on its remaining energy, the gateway should restrict its available bandwidth and processing power offered to other IoT devices. It should be noted that although the gateway might be equipped with a high-bandwidth connection to the Internet (e.g. WiFi), its interface with IoT devices is still a

low-power wireless connection such as Bluetooth Low Energy or ZigBee, which have a low bandwidth [10, 11, 12].

In such a local network, IoT devices aim at reaching the highest overall QoS while meeting their battery lifetime constraints. A mechanism is needed to capture the dynamism in an IoT network and help the IoT devices to choose their best operational configuration using their control knobs (i.e. QoS level and offloading scheme). The QoS management under battery lifetime constraints in IoT needs dynamic online solutions due to the following reasons:

1. The remaining energy of IoT devices varies over time due to consumption or recharge.
2. In case of having a general device as the gateway (e.g. smartphone), the available bandwidth or processing capability may change over time due to other workloads, or due to its available power.
3. The number of IoT devices connected to the gateway may change [13].
4. The gateway may change in an IoT local network [7, 10].

An online solution is necessary to address efficient QoS management in such a dynamic IoT configuration. In relevant approaches, MiLAN [14] is a middleware that manages and allocates network resources for applications that *fuse* data from multiple sensors and need to select optimal set of sensors. However, the QoS of each sensor is fixed. Moreover, MiLAN considers only the bandwidth limitation of network while the processing capability is not modeled. Besides, it does not model the on-board processing and therefore cannot support combination of offloading schemes. In [15], an approach is proposed to minimize the energy consumption of sensor nodes by computation offloading. This approach is used for finding optimal partitions of an application for computation offloading. The output of this approach is an input to our problem.

The novel contributions of this paper are as follows:

- We present a QoS management scheme for IoT systems with constraints on battery, bandwidth and processing power.
- We present an Integer Linear Programming (ILP) formulation for the problem and show that this problem is NP-hard.
- We propose a pseudo-polynomial time scheme which not only reuses sub-solutions of a problem instance, but also reuses the pre-computed solutions for the next problem instance.

Paper structure: in Section 2, we present the problem formulation and then we provide a detailed presentation of our proposed solution in Section 3. Section 4 illustrates the effectiveness of our solution by means of a case study. Experimental results and evaluations are presented in Section 5, while we conclude the paper in Section 6.

2 Problem Formulation

Consider a local network with a set of N IoT devices, where each device is uniquely identified by an integer value $d \in \{1, \ldots, N\}$.

$$I_d = \Big(X_d, R_d, B_d, e_d, U_d(\cdot), S_d(\cdot), T_d(\cdot), C_d(\cdot)\Big) \qquad (1)$$

Each IoT device I_d is specified by a tuple, where

- X_d denotes the set of possible *input data rates* of device I_d. They depend on the sensor sampling frequency and data resolution. An IoT device offers its service at M_d different *QoS levels*, with each level having a different input data rate and thus providing a different service quality. For instance, consider an IoT-based heart monitoring device that can capture ECG signals at multiple discrete sampling rates (e.g. 100 Hz, 200 Hz, 300 Hz, and 1000 Hz).

$$X_d = \big\{x_{d_i} \mid i \in [1, M_d]\big\} \qquad (2)$$

- R_d denotes the set of possible *transmission data rates* of device I_d. They depend on the input data rate x_{d_i} and the computation offloading strategy of the IoT device. Offloading determines how much input data is not processed on the device (on-board processing), but instead transmitted to the gateway (offloaded). An IoT device offers Q_d different *offloading levels*. The transmission data rate $r_{d_{ij}}$ depends on the QoS level i (input data rate) and the offloading level j. It corresponds to the share of input data rate that is offloaded plus the (intermediate) results from the on-board processed data.

$$R_d = \big\{r_{d_{ij}} \mid i \in [1, M_d],\ j \in [1, Q_d]\big\} \qquad (3)$$

  For instance, consider that the above-mentioned IoT-based heart monitoring device had 3 offloading levels. Level 1 could indicate 'no offloading' and thus only transmits a small amount of results (e.g. the features that were extracted from the signal), independent of the input data rate. Level 2 could be used to offload a certain percentage of the input sample rate. For instance, the device could store a certain amount of input samples in a buffer, then process the buffer and offload all incoming samples during this processing to the gateway. Level 3 could perform some pre-processing on the data, e.g. applying a filter on it, and then send the filter output to the gateway. As the filter would not reduce the data rate, the transmission rate would equal the input data rate, but some of the processing is already done. The particular transmission data rates depend on the device and how it is used, which has to be determined by the user and thus is considered as given in this problem formulation.
- B_d denotes the minimum required battery lifetime (i.e. until the next recharge or battery replacement).
- e_d is the remaining energy in the battery of device I_d.
- $U_d(x_{d_i})$ is the utility function that quantifies the utility or quality of service (QoS) provided to the user when the device is capturing input data at rate x_{d_i}.
- $S_d(x_{d_i})$ is the power consumption of the device for sensing and capturing data at rate x_{d_i}.
- $T_d(r_{d_{ij}})$ is the power consumption for transmitting data at rate $r_{d_{ij}}$.
- $C_d(i, j)$ is the power consumption for processing at input data rate x_{d_i} under offloading level j.

The battery lifetime of IoT devices depends on 1) the remaining energy and 2) the total power consumption rate:

$$b_{d_{ij}} = \frac{e_d}{S_d(x_{d_i}) + C_d(i, j) + T_d(r_{d_{ij}})} \qquad (4)$$

where $b_{d_{ij}}$ denotes the expected battery lifetime when the device captures input data at rate x_{d_i}, processes it, and then transmits at rate $r_{d_{ij}}$.

The gateway connects devices to the Internet. It receives data from IoT devices, processes it and transmits the final result to the Internet. The gateway is specified by triple:

$$G = \Big(p(\cdot), R, P\Big)$$

where:

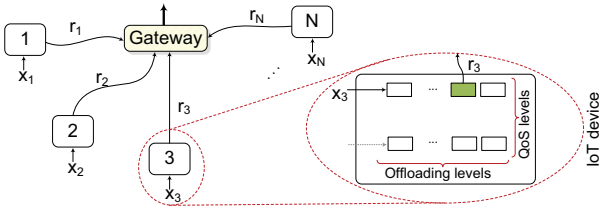

Figure 2: Problem model: IoT devices with different QoS and offloading levels resulting in different transmission data rates. The gateway receives and processes the data.

- $p(r_{d_{ij}})$ shows the required processing capability of the gateway to perform the necessary operations on the received data at rate $r_{d_{ij}}$ and offloading level j.
- R is the total available bandwidth of the gateway to receive data from IoT devices.
- P shows the total processing capability of the gateway.

The effect of environment and surrounding devices (e.g. interference) on the transmission can be modeled in R and $T_d(\cdot)$.

The system is summarized in Figure 2. The problem that is targeted in this paper can be solved by deciding the QoS level i and the offloading level j for each IoT device I_d at runtime, such that the bandwidth, computation, and lifetime constraints are fulfilled (Eq. (5) to (7)) and the overall benefit (Eq. (8)) is maximized.

$$\text{Bandwidth constraint:} \quad \textstyle\sum_{\forall d} r_{d_{ij}} \leq R \tag{5}$$

$$\text{Computation constraint:} \quad \textstyle\sum_{\forall d} p(r_{d_{ij}}) \leq P \tag{6}$$

$$\text{Lifetime constraint:} \quad \forall\, d:\ b_{d_{ij}} \geq B_d \tag{7}$$

$$\text{Optimization goal:} \quad \text{maximize} \textstyle\sum_{\forall d} U_d(x_{d_i}) \tag{8}$$

3 Proposed Solution

3.1 Decomposing the Problem

The targeted problem (see Section 2) has two sets of constraints: one for IoT devices and one for the gateway. The selected configurations for devices I_d (i.e. x_{d_i} and $r_{d_{ij}}$) should meet the lifetime constraint (Eq. (7)). Given the selected configuration for IoT devices, the gateway's constraints to be met are bandwidth and computation (Eq. (5) and (6)).

The device's constraint depends solely on device parameters. To reduce the search space, we can decompose the optimization problem into 1) device's problem and 2) network (gateway) problem. In the device's problem, each IoT device excludes those configurations that violate its lifetime constraint to reduce the search space. Then, the network (gateway) problem is solved by considering the reduced search space.

3.2 Device Problem

3.2.1 CoD Matrix

We use a matrix for each IoT device that shows the possible Configurations of that Device (CoD matrix). Each element $\kappa_d[i,j]$ at the intersection of the QoS level i (corresponding to the input data rate x_{d_i}) and the offloading level j contains a pair of the battery lifetime (see Eq. (4)) and the transmission data rate, i.e. $(b_{d_{ij}}, r_{d_{ij}}), i \in [1, M_d],\ j \in [1, Q_d]$, as shown in Figure 3.

Since the available energy of IoT devices changes over time (due to consumption or re-charge), the expected battery lifetime of each configuration (i.e. the first element of each tuple in the matrix) changes over time, which means this matrix needs to be updated periodically.

$$\overbrace{\quad 1 \quad \cdots \quad Q \quad}^{Q \text{ Offloading levels}}$$
$$M \text{ QoS levels} \begin{cases} x_1 \\ \vdots \\ x_M \end{cases} \begin{pmatrix} (b_{11}, r_{11}) & \cdots & (b_{1Q}, r_{1Q}) \\ \vdots & \ddots & \vdots \\ (b_{M1}, r_{M1}) & \cdots & (b_{MQ}, r_{MQ}) \end{pmatrix}$$

Figure 3: CoD matrix for an IoT device I_d; omitting subscripts d for brevity

Example 1: Consider an IoT device I_d with battery lifetime constraint of $B_d = 40$ that has 5 different QoS levels $X_d = \{100, 200, 300, 400, 1000\}$ and 4 different offloading levels per QoS level. In this example, lets say that each offloading level transmits 20% more data than the previous offloading level, i.e. $r_{i(j+1)} = r_{ij} + 0.2 * x_i$.

$$\overbrace{\quad 1 \quad 2 \quad 3 \quad 4 \quad}^{\text{Offloading levels}}$$
$$\text{QoS levels} \begin{cases} x_1 = 100 \\ x_2 = 200 \\ x_3 = 300 \\ x_4 = 400 \\ x_5 = 1.000 \end{cases} \begin{pmatrix} (49, 20) & (57, 40) & (63, 60) & (74, 80) \\ (44, 40) & (46, 80) & (54, 120) & (61, 160) \\ (31, 60) & (43, 120) & (46, 180) & (52, 240) \\ (24, 80) & (28, 160) & (33, 240) & (40, 320) \\ (18, 200) & (22, 400) & (27, 600) & (31, 800) \end{pmatrix}$$

Figure 4: CoD matrix for Example 1

Figure 4 shows the CoD matrix associated with Example 1. The configurations whose expected battery lifetime is less than the constraint (i.e. $b_{d_{ij}} < B_d$) are not feasible and should be excluded from the search space. The infeasible configurations (those with expected battery lifetime less than 40) are shown in shaded area. All other configurations are feasible as they meet the battery constraint.

3.2.2 Properties of CoD Matrix

The CoD matrix of most applications may have the following property. However, we should emphasize that neither the formulated problem nor our presented solution is restricted to this property.

Property 1: For each column of the CoD matrix, the elements are in an increasing order in terms of transmission data rate, but in an decreasing order in terms of estimated battery lifetime.

The nature of this property can be understood intuitively. For a given offloading level (i.e. column), an increased input data rate leads to increased transmission data rate and increased on-board processing requirements. This has negative effects on the power consumption for sensing data S_d, transmitting data T_d, and processing data C_d, all of which have a negative effect on the battery lifetime (see Eq. (4)).

3.2.3 Reducing Problem Size

Although all the elements outside the shaded area meet the battery lifetime constraint and are feasible configurations for the problem, some of them are intuitively inefficient. For instance, all the elements in the first row of the above matrix (Figure 4) provide the same QoS to the user. They are just different in the the amount of computation offloading.

Selecting the element with the smallest transmission data rate (marked with a green circle in Figure 4) would result to the optimal solution as is proved in the following.

Theorem 1. *If two elements from the same row i are feasible, $\kappa(i, j_1)$ and $\kappa(i, j_2)$ where $j_1 < j_2$, then $\kappa(i, j_2)$ never outperforms $\kappa(j, j_1)$ in the optimal solution.*

PROOF. The theorem is intuitively obvious. Since $\kappa(i, j_1)$ and $\kappa(i, j_2)$ offer the same QoS level i and utility $U(x_i)$ their direct contribution to the objective goal (see Eq. (8)) does not differ. However, imposing less computation to the gateway (i.e. less transmitted data) may leave more room for other IoT devices, so that the gateway can possibly choose another feasible configuration with higher utility and correspondingly more computation and offloading. □

Inspired by theorem 1, for each IoT device, we only consider the leftmost feasible element of each row (i.e. the one with the least data transmission rate):

$$\forall\ 1 \leq i \leq M_d \quad r_{d_i} = \min_{b_{d_{ij}} \geq B_d} (r_{d_{ij}}) \tag{9}$$

After building/updating the CoD matrix, each device finds the efficient feasible configurations (EFC), from Eq. (9), and forms a set of pairs containing 1) utility of each EFC, and 2) the transmission data rate r_{d_i} of each EFC. In the previous example shown in Figure 4, let us assume the utility of QoS levels is as $U(100) = 50$, $U(200) = 90$, $U(300) = 110$, $U(400) = 150$, and $U(1000) = 310$. Then the EFC set of this device is shown in Figure 5.

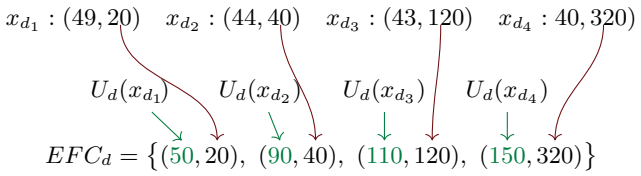

Figure 5: An example of EFC set

Each IoT device periodically checks its remaining energy (i.e. e_d), updates the CoD matrix, and updates the EFC set. In case that the EFC set changes, the device sends the new set to the gateway, where it is used to solve the *Network problem*. Note that the EFC set only contains feasible solutions, i.e. the number of entries in the EFC set of a particular device may change over time.

3.3 Network (Gateway) Problem

3.3.1 Integer Linear Programming (ILP) Formulation

For each IoT device, there is a set of efficient feasible configurations (EFC), showing the offered utility and transmission rate to the gateway. Given all EFC sets of all IoT devices, the gateway needs to solve the QoS management problem. The gateway extends each EFC set by including the processing requirement of the associated transmitted data (i.e. $p(r_d)$). The resulting EFC'_d set is shown in Eq. (10).

$$EFC'_d = \left\{ \left(U_{df},\ r_{df},\ p_{df}\right)_{1 \leq f \leq \left|EFC_d\right|} \right\}$$
$$U_{df} = U_d(x_{d_f}) \text{ // Utility of device } I_d \text{ for } f\text{-th EFC entry} \tag{10}$$
$$r_{df} = r_{d_f} \text{ // Corresponding transmission rate (see Eq. (9))}$$
$$p_{df} = p(r_{d_f}) \text{ // Corresp. gateway processing requirement}$$

The gateway problem is to select one and only one item from each set such that the overall utility is maximized and the gateway's constraints are met. This can be formulated as an Integer Linear Programming (ILP):

$$\max \quad \sum_d \sum_f (U_{df} \times w_{df}) \tag{11}$$
$$\text{subject to} \quad \forall d: \ \sum_f w_{df} = 1 \tag{12}$$
$$\sum_d \sum_f r_{df} \times w_{df} \leq R \tag{13}$$
$$\sum_d \sum_f p_{df} \times w_{df} \leq P \tag{14}$$

where

$$w_{df} = \begin{cases} 1 & \text{if } f\text{-th EFC element from } d\text{-th device is chosen} \\ 0 & \text{otherwise} \end{cases}$$

This optimization problem corresponds to the Multidimensional Multiple-Choice Knapsack Problem (MMKP) and is NP-hard, thus computationally intractable [16]. Since the constraints in Eq. (13) and (14) are inequalities, the proposed technique in [17] for merging multiple constraints does not apply to our problem. In the following, we present a pseudo-polynomial time solution to our problem based on a dynamic programming (DP) approach.

Definition 1: We use the term "instance of problem" to refer to the configurations of the gateway problem (i.e. the input data for Eq. (11) to (14)). Any change in the EFC sets makes it another instance of the problem.

3.4 Dynamic Programming Solution

3.4.1 Intuition

Although we can design and introduce heuristic approaches, a dynamic programming solution seems more appropriate for our problems because:

- We need to solve different instances of the problem where subsequent instances do not differ substantially. This gives the dynamic programming approach the opportunity to possibly reuse some computations that are performed in the previous instance of the problem.
- It provides a quick and optimal solution to the problem.

Since our problem has two dimensions (i.e. two constraints including data rate and processing power), the table to form the basis for the dynamic programming approach has 3 dimensions: one for the number of devices, and two others for the constraints.

3.4.2 Formulation & Example

Let $Z(d, R, P)$ denote the maximum overall utility that we can get from the first d devices while the constraints on the data rate and processing capability of gateway are R and P, respectively.

For the d^{th} device, we need to choose one of its configurations from its EFC set whose size is $\left|EFC_d\right|$. Considering the f^{th} element of the EFC set, its utility, data rate, and processing requirements are U_{df}, r_{df} and p_{df}, respectively. For the f^{th} configuration, we first find the overall utility of a solution with $d-1$ devices whose overall required bandwidth and processing resources are $R - r_{df}$ and $P - p_{df}$, respectively. Then, we add it to the provided utility by f^{th} configuration (i.e. U_{df}). We investigate all the possible configurations of device d and select the one that maximizes the overall utility.

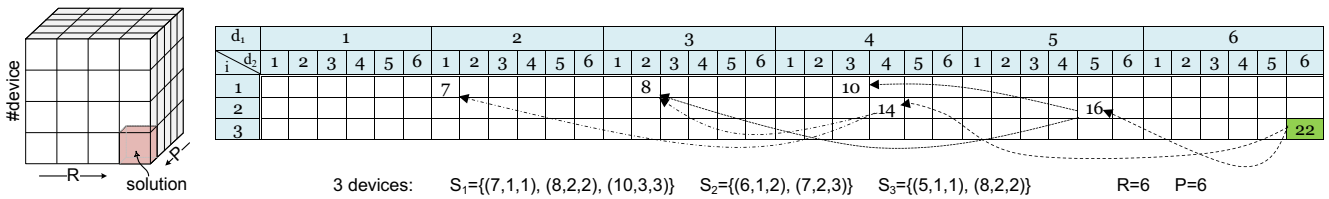

Figure 6: The dynamic programming table (left), and an example solved using our approach (right).

Therefore, this algorithm leads to the global optimum solution. It should be noted that the order of devices does not matter. Equation (15) shows the recurrence relation which is proposed to calculate $Z(d, R, P)$ (as explained above):

$$Z(d,R,P) = \max_{1\le f\le |EFC_d|} \Big\{ Z\big(d-1, R-r_{df}, P-p_{df}\big) + U_{df} \Big\} \quad (15)$$

Example 2: Figure 6 shows an example with 3 devices whose extended EFC sets are $EFC'_1 = \{(7,1,1),(8,2,2),(10,3,3)\}$, $EFC'_2 = \{(6,1,2),(7,2,3)\}$, and $EFC'_3 = \{(5,1,1),(8,2,2)\}$, respectively. The constraints of the gateway are $R = 6$ and $P = 6$. The optimal solution is $Z^* = Z(3,6,6)$. Based on Eq. (15):

$$Z(3,6,6) = max\{5 + Z(2,6-1,6-1), 8 + Z(2,6-2,6-2)\}$$
$$= max\{5+16,\ 8+14\} = 22$$
$$Z(2,5,5) = max\{6 + Z(1,4,3), 7 + Z(1,3,2)\} = 16$$
$$Z(2,4,4) = max\{6 + Z(1,3,2), 7 + Z(1,2,1))\} = 14$$

In this example, the cell $Z(1,3,2)$ is referred twice, which simply shows the benefit of using dynamic programming to avoid recomputing the same sub-problem repeatedly. A top-down dynamic programming is preferred for our approach as some of the sub-problems never get examined at all (see Figure 6). For instance, among all the elements corresponding to device N (3 in the above example), we are just interested in computing $Z(N,R,P)$ (or $Z(3,6,6)$ in our example). Hence, to avoid computing unnecessary cells, we implement a top-down DP that is based on the recursive relation and on *memoizing* the computed sub-problems in a linked list structure, which reduces the memory usage as well as computation run-time.

Algorithm 1 illustrates our top-down dynamic programming approach to find the optimum solution to the QoS management problem based on the recursive relation in Eq. (15).

3.5 Reusing Sub-solutions

Definition 2: Mask the change: Any change in the EFC set of device d that does not change the cells of the table corresponding to device $d+1$ (i.e. $[d+1,*,*]$) is masked, and hence does not propagate to other cells of the table. A masked change does not affect the solution.

As mentioned earlier, the EFC set of each node may change over time as its available energy changes due to consumption or battery re-charge. Some of those changes can be completely masked. However, the unmasked changes still benefit from pre-computed cells of the solution table.

3.5.1 Classifying the Possible Changes

Possible changes in an EFC set can be categorized into three different class:

1. **ADD:** A new item is added: It happens only when the battery is re-charging and the available energy (i.e. e_d) increases. Then, a QoS level that was not present in the EFC set is added to it. For example in Figure 7(a), the highest QoS level (last row) is included as one of its configurations meets the battery lifetime constraint.
2. **REM:** An existing item is removed: It happens when the battery depletes and the available energy decreases. Then, a QoS level that can no longer fulfill the battery lifetime constraint (under the new circumstances) is completely excluded and its corresponding items are removed from the EFC set as shown in the example of Figure 7(b).
3. **CHANGE:** An existing item changes its second entry (i.e. r). It means that the item still offers the same QoS level (the same utility value U), but with a different data transmission rate. In other words, the item is replaced with another item from the same row in the CoD matrix but from a different column as shown in Figure 7(c) and Figure 7(d). It happens under two different circumstances:
   - **DEC:** Due to battery recharge, a new element in the CoD matrix is added to the feasible region of a row (see Figure 7(c)). Therefore, the corresponding item in the EFC set is replaced with a new one which has the same utility (they are both in the same row of CoD) but with less data to transmit (i.e. more on-board processing): $(U, r^*) \rightarrow (U, \hat{r}) : r^* > \hat{r}$.
   - **INC:** Due to the energy consumption, the previous EFC item is not feasible anymore and another element in the CoD matrix from the same row is selected to be in the EFC set (see Figure 7(d)). The first entry (i.e. utility) is the same, but the data rate is increased (i.e. more computation offloading): $(U, r^*) \rightarrow (U, \hat{r}) : r^* < \hat{r}$.

Any change in the EFC set falls into one of the above categories. It should be noted that multiple changes can happen

Algorithm 1: Our proposed approach based on top-down DP

```
1  Inputs : M EFC sets, and constraints P and R
2  Function Z(d, R, P)
3      if R ≤ 0 or P ≤ 0 then
4          return −∞;
       else if d == 0 then
5          return 0;
       end
6      for f ← 1 to |EFC_d| do
7          if r_df ≤ R and p_df) ≤ P then
8              if Tb[d − 1][R − r_df][P − p_df] == −∞ then
                   Tb[d − 1][R − r_df][P − p_df] ←
                     Z(d − 1, R − r_df, P − p_df));
               end
9              if Tb[d − 1][R − r_df][P − p_df] + U_df > Tb[d][R][P]
                then
10                 Tb[d][R][P] ← Tb[d−1][P−r_df][P−p_df]+U_df;
               end
           end
       end
11 call Z(N, R, P)
```

at the same round of updating the CoD matrix.

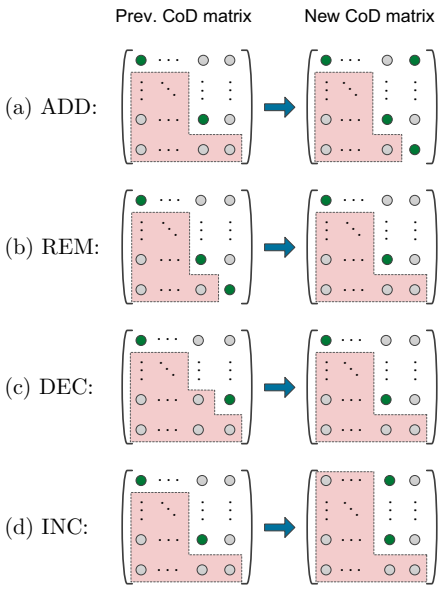

Figure 7: Different possible changes in EFC set of an IoT device

3.5.2 *When and How to Reuse?*

We investigate different classes of changes, and try to propose efficient solutions to reuse the sub-solutions that were computed before the changes. Among all the possible changes, some do not change the optimal solution. Hence, the previous solutions remain the same with no need to solve the problem again. Some cases can reuse the previous solution partially, and others need to rerun the algorithm.

- ADD: In case that the change is an 'ADD', the previously computed cells of table are completely reused, but some new cells need to be computed. For instance, assume that in Example 2, a new item is added to the EFC set of device 2, i.e. $EFC_2' = EFC_2' \cup \{(8,3,4)\}$. It affects computing of $Z(2,5,5)$ and $Z(2,4,4)$ as following:

$$Z(2,5,5) = max\{6+Z(1,4,3), 7+Z(1,3,2), 8+Z(1,2,1)\} = 16$$

$$Z(2,4,4) = max\{6+Z(1,3,2), 7+Z(1,2,1), 8+\boxed{Z(1,1,0)}\} = 14$$

  As it is shown, only one new table cell is needed to be computed (i.e. $Z(1,1,0)$) and all the other required cells are reused from previously computed cells.
- REM:
  1. If the removed item was not the item that was selected as the solution in the previous setup, then the optimal solution does not change and there is no need to re-run the algorithm. This case can be checked and masked at the IoT device without the need to inform the gateway.
  2. Based on Property 1, the removed item had the highest utility and data rate. In this case, the solution to the new setup is already in the computed table, as the new instance of the problem is a subset of the previous instance.
- INC: The updated item in the EFC set is either the selected item in previous instance or a non-selected item.
  1. In the former case, if the amount of increase in bandwidth (i.e. $\Delta r = r^* - \hat{r}$) and processing (i.e. $\Delta p = p(r^*) - p(\hat{r})$) are less than available resources that are left (i.e. $R - \sum r_d^*$ and $P - \sum p(r_d^*)$), then the solution does not change.
  2. In the latter case, the optimal solution does not change, hence there is not need to re-run the algorithm to find it.
- DEC: If this change happens for device d_1 ($1 \leq d_1 \leq N$), only the precomputed cells of devices $1 \leq d_2 < d_1$ can be reused, and the other referred cells need to be updated. Let us assume that in Example 2, the EFC set of device 2 witnesses a 'DEC' change as the second item becomes $(7,2,2)$. All the cells containing $Z(1,*,*)$ remain unchanged, but the others are updated as following:

$$Z(2,5,5) = max\{6+Z(1,4,3), 7+Z(1,3,3)\} = 17$$
$$Z(2,4,4) = max\{6+Z(1,3,2), 7+Z(1,2,2)\} = 15$$
$$Z^* = Z(3,6,6) = max\{5+17, 8+15\} = 23$$

This shows that in our proposed solution not only the sub-problems of each instance can be reused to avoid re-computation (by means of dynamic programming), but different instances of the problem (i.e. after changes in the setup) can still benefit from previously computed sub-solutions.

Figure 8 shows a simplified flow of the solution including the device flow and gateway solution flow.

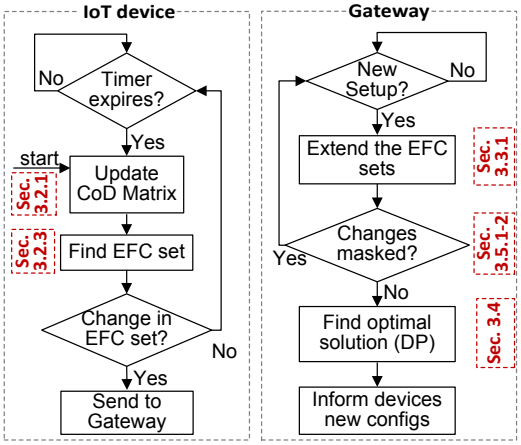

Figure 8: Simplified flow of QoS management

4 Use case: IoT in Healthcare Monitoring

Healthcare has been recently emerged to a promising but demanding IoT application paradigm [18, 19]. IoT-based portable devices provide continuous personal health monitoring, e.g. electrocardiography (ECG), electroencephalogram (EEG), Electromyogram (EMG), blood pressure (BP), etc., aiming to reduce the costs of hospitalization and/or support early disease detection, fitness and wellness. In this paper, we focus on ECG arrhythmia detection, serving as our driver application around which we formulate a realistic use case for evaluating the efficiency of the proposed IoT resource management solution.

4.1 ECG Analysis & Arrhythmia Detection

ECG provides essential information about the status of the heart which is critical for prevention, early diagnosis, and treatment of cardiovascular diseases [20] as well as wellness applications [21]. In this work, ECG signals are used to detect heart arrhythmia, i.e. irregularly fast or slow heart beats which may lead to strokes or heart failure [22]. On the IoT device, we implement the ECG analysis flow of Figure 9,

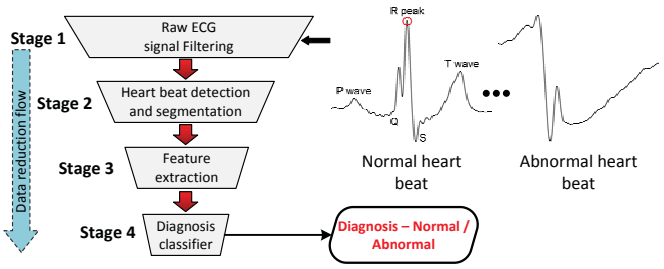

Figure 9: ECG analysis flow

which receives raw ECG data and detects arrhythmia (if any).

1. **Filtering:** The initial step includes signal acquisition and filtering. A band-pass FIR filter is used to remove baseline wander (< 1 Hz) and power line noise (50 Hz).
2. **Segmentation and heart beat detection:** For segmentation of the ECG signal, we first need to detect the R peak (see the annotated ECG signal in Figure 9). The periodicity of the heart beat is not constant and varies due to different reasons such as physical activity, stress level, etc. Hence, a window of samples are examined to locate the peak. The window size depends on the sampling rate. The detected R peak is used as the start of a new segment.
3. **Feature extraction:** Features extraction of the heart beat is performed through Discrete Wavelet Transformation (DWT) which is very popular for ECG signal processing due to the fact that it is lightweight and capable of providing time and frequency information simultaneously [23, 24]. This is essential when analyzing signals whose frequency response varies in time, such as the ECG signal, and thus time localization of the frequency spectral components is required.
4. **Diagnosis:** For the final stage, a Support Vectors Machine (SVM) classifier is used to capture non-linear relationships of the feature space representing the target classification problem [11]. The execution of the classifier concludes whether the processed heart beat exhibits any signs of arrhythmia. Figure 9 depicts an example ECG signal exhibiting normal and arrhythmic heart-beats.

The examined IoT scenario is located at clinical ward which welcomes a large number of patients that should all be monitored simultaneously. Wearable ECG analysis devices communicate with the gateway through low-power low proximity wireless communication which in our deployed scenario is Bluetooth Low Energy.

4.2 QoS and offloading levels

The presented design is pipelined in the sense that it produces valid results at the end of every processing stage. For example, if the device is instructed to execute up to the feature extraction process then the output of the flow is the Discrete Wavelet Decomposition of the input signal. This result can be used in subsequent processing if transmitted to other devices.

This property of the flow enables the system to support offloading of processing to the gateway. All of the aforementioned pipeline stages can be executed on the gateway. Therefore, processing can be performed up to an arbitrary pipeline stage on the IoT node, then transmit its output to the gateway, and resume the execution of pipeline there. The gateway decides the level of offloading for each IoT node of the system by solving the QoS management problem.

As far as different QoS levels are concerned, they correspond to different sampling frequencies of the ECG signals. Signals sampled at a higher frequency offer more detailed description of the monitored ECG. The increased signal resolution enhances further analysis and diagnosis by medical experts thus leading to increased QoS for the patient. Combinations of QoS levels and offloading levels (i.e. after which pipeline stage computation is offloaded to the gateway) result in different data rates for input (x_d in Eq. (2)) and output (r_d in Eq. (3)) of the device. Table 1 summarizes these values for combinations of QoS Levels and offloading levels stages for our ECG monitoring prototype.

Table 1: Input data rates and transmission data rates for different QoS levels and offloading levels

			Transmission rate r_d [B/s] for offloading after a certain pipeline stage			
QoS level	Sampling freq. [Hz]	Input data rate x_d [B/s]	Stage 1	Stage 2	Stage 3	Stage 4
1	180	720	720	360	104	1
2	360	1440	1440	720	192	1
3	720	2880	2880	1440	372	1
4	1440	5760	5760	2880	564	1
5	2000	8000	8000	4000	1024	1

Due to the increased sampling frequency of higher QoS levels we observe an increase in input and output data rate of each stage. For example, if our window of data analysis W is 256 points wide at sampling frequency of 360 Hz, then the corresponding window rises to 512 data points at double the sampling frequency. Inevitably, this affects all other pipeline stages given that they operate on greater amount of data. The only exception is the result of the analysis flow (Stage 4), which is always one value that corresponds to the diagnosis label of the processed heart beat.

Stages 1 to 3 of the flow have been designed to operate on a variable-sized input data window while a classifier model (stage 4) was trained for each QoS level. Therefore, there is an instance of the pipeline for each QoS Level, which operates on different amount of data. To comprehend how this fact affects the resources needed for the execution of each combination of QoS level and pipeline stage, we profiled the execution of the flow on the target IoT device. Figure 10 summarizes the percentage of execution of each processing stage over one minute for increasing QoS levels.

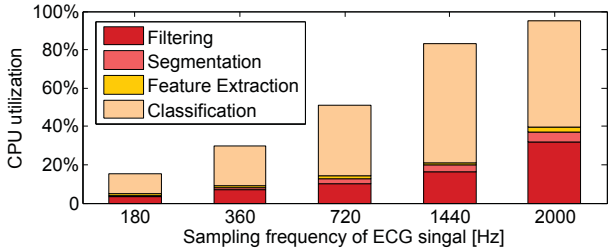

Figure 10: CPU utilization of ECG analysis stages

A expected, we observe that a higher QoS level comes at the price of increased computational requirements. Figure 10 also shows the computational effort that is offloaded to the gateway for the different offloading levels, as the breakdown for the computational complexity of all pipeline stages is shown. In all cases, the most computationally intensive

stage is the diagnosis part due to its complex structure in an effort to provide accurate predictions. On the contrary, beat segmentation and DWT do not occupy the CPU for prolonged period. The rest of the time the CPU remains idle, which is the major reason of power consumption variations over different QoS levels.

5 Evaluation and Results

5.1 Experimental Setup

To construct our ECG analysis flow, we use actual patient ECG data records from *MIT-BIH Arrhythmia Database* [25]. The annotated signals by medical experts are used for training our machine learning tools. Original signals were sampled at 360 Hz. Down- and up-sampling has been used to generate ECG signals of differing sampling rates.

Experimental analysis has been performed by simulating IoT network topologies of up to 10 nodes. To efficiently model the characteristics of our case study for different QoS and Offloading levels, we profile the real execution of the ECG analysis flow on an Intel Quark SoC, already proposed and used for wearable IoT devices [26, 27]. The outcome of this profiling campaign summarizes the computational requirements, expressed in CPU utilization, of each combination of QoS- and offloading-level.

To conduct the experiments, a combination of experimentally derived data enhanced with nominal data from datasheets of commercial devices is used for the model parameters values. Regarding the available energy of the IoT device (e_d), a battery consumption model of each IoT node is composed based on the instrumented CPU utilization. More specifically, energy consumption of ECG acquisition $S_d(x_{d_i})$ was calculated based on [28]. Bluetooth Low Energy (BLE) is used for communication between IoT devices and gateway. Power consumption value of data transmission (i.e. $T_d(r_d)$) is $0.153\,\mu W$ based on [29] and transmission latency is $4\,\mu s/bit$ [30]. Since BLE exploits an adaptive frequency hopping mechanisms, the probability of interference is very low. To complete the battery model of the IoT nodes, we choose a rechargeable Lithium-Ion coin cell battery with a nominal capacity of up to $420\,mAh$ [31]. We use a realistic discharge model for the battery using [32] for various values of discharge currents to evaluate the available energy for Eq. (4).

An ARM Cortex-M3 device is considered as the gateway [33]. The energy consumption values were acquired by profiling the execution of the ECG analysis flow for all combinations of QoS levels and processing stages to measure the values of our parameters, e.g. $C_d(\cdot)$, $p(r_{d_{ij}})$, etc.

The final key component of the system model is to determine the utility functions of each device. The QoS value of each combination of QoS level and ECG processing stage was set proportional to the ratio of the sampling frequency of the ECG signal divided by the maximum available ECG sampling frequency (2 kHz). We also enable the creation of more complex profiles of IoT devices by allowing the user to specify a factor of how important high QoS levels are for this device.

5.2 Overhead Analysis

We implemented our DP approach on the ARM Cortex-M3 microcontroller that is used as our gateway platform. We measured the number of CPU cycles that our proposed solution needs to calculate the optimal result and compare it against the brute-force (BF) method in Figure 11. As the number of devices increases, the algorithm execution time for

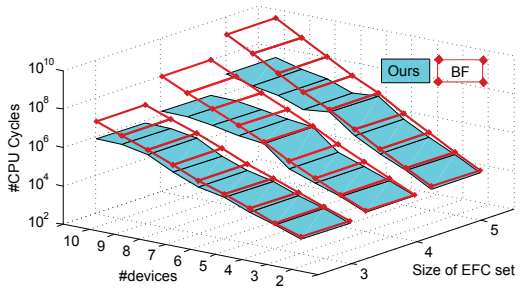

Figure 11: The execution time of our proposed method compared to the BF method for different number of devices and different sizes of EFC sets.

BF increases exponentially whereas the execution time of our algorithm increases moderately (note the logarithmic scale of the Z-axis). For instance, having ten IoT devices with each one having five feasible configurations, our algorithm finds the optimal solution in 0.4 seconds when the gateway is running at 100 MHz (i.e. $\frac{39{,}480{,}530\,[cycles]}{10^8\,[Hz]} \simeq 0.4\,s$), while the BF methods takes around 56 seconds (i.e. $\frac{5{,}587{,}270{,}875\,[cycles]}{10^8\,[Hz]} \simeq 56\,s$). The generally short execution time of our proposed solution and its good scalability show its suitability for online and dynamic scenarios of IoT networks where the number of active devices and their feasible configurations change over time repeatedly.

As explained in Section 3.4.2, our proposed solution is based on a recursive function and its sub-problems may occur repeatedly. We achieve execution time reduction by *memoizing* the answers of sub-problems to avoid recomputing them over and over. In Figure 12, we show a detailed analysis of our proposed algorithm with $N = 7, \cdots, 10$ devices and $Q = 3, \cdots, 5$ QoS levels. It shows the average number of recursive function calls if the sub-problems are not stored (named 'Total'), and the average number of recursive function calls in our approach. Note that the values on the X-axis are presented in logarithmic scale. The Y-axis shows the recursion level where the function call happened. To make it clear, see Figure 13 which shows the function calls as a tree. The root is at the N^{th} level, where N is the number of IoT devices (see Section 2). For instance, in Figure 13 the number of function calls at level 9 is equal to 4. Now consider the green node which is labeled with (1). This node is called four times, however, in our proposed solution it is called only once and the result is stored in the table for later references. The number of function calls at the N^{th} level is always 1 and is not shown in Figure 12.

In Figure 14, we show the time interval between two successive re-executions of our algorithm (i.e. time between two problem instances), which is triggered after a change in the set of feasible configurations. As we increase the number of devices in our case study, the average time between re-executions decreases (i.e. it is needed more frequently).

5.3 Comparison to unsupervised devices

We also compare our solution to the system that operates with no QoS management by the gateway (called 'unsupervised'). We conducted experiments with two scenarios for the unsupervised system.

In the first scenario, devices operate at the highest QoS level and with no offloading to the gateway. We consider a battery lifetime constraint of 20 hours (1200 minutes) and assume that designers can choose a battery out of three

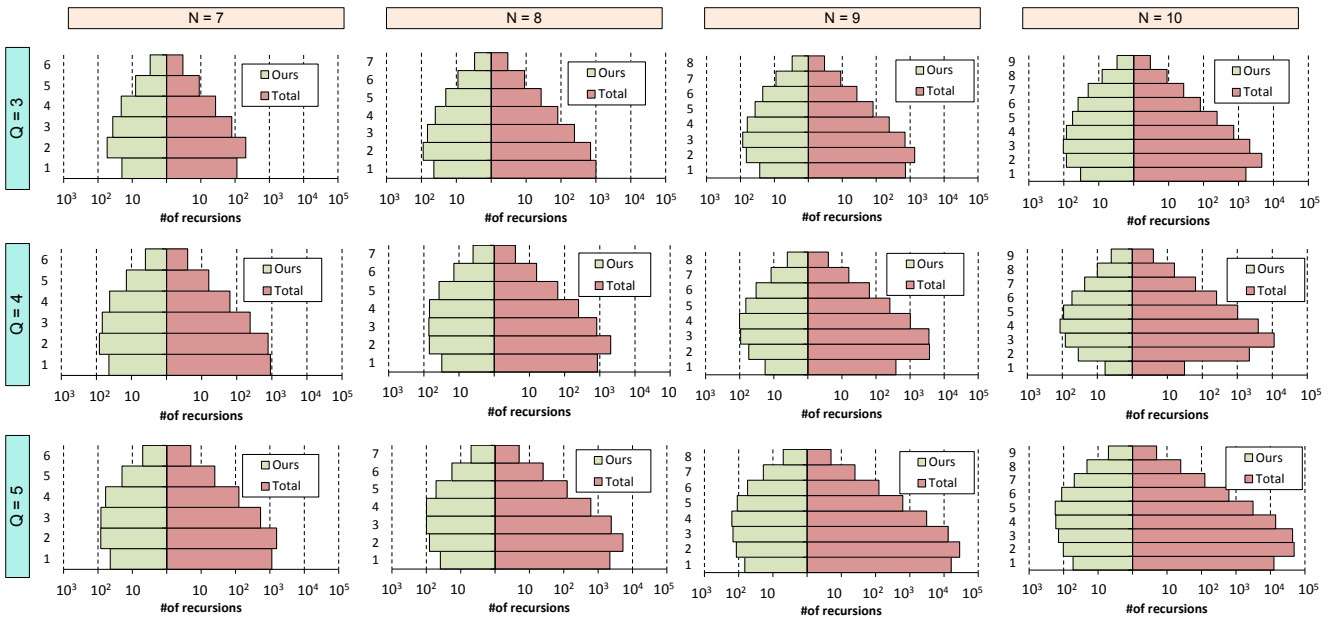

Figure 12: Total number of recursive calls of function (see Section 3.4.2), and number of recursions in our algorithm at different levels of the tree

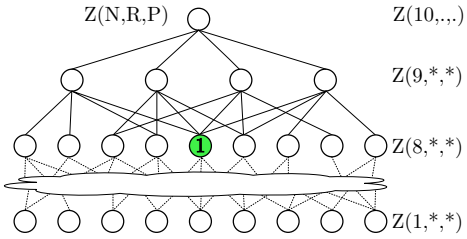

Figure 13: function calls tree

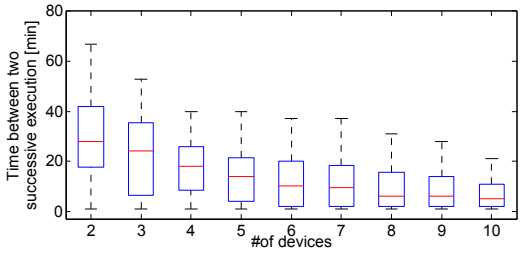

Figure 14: The time intervals between successive re-execution of algorithm in the case study

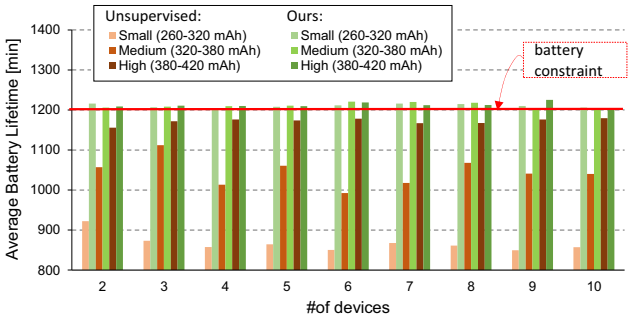

Figure 15: The average battery lifetime of devices in our system and the unsupervised system for different battery sizes

ranges with small (260–320 mAh), medium (320–380 mAh) and high (380–420 mAh) capacity. For a varying number of IoT devices in a range of 2 to 10, we compare the achieved battery lifetime of the unsupervised system and our solution. Figure 15 shows the average achieved battery lifetime. The unsupervised system fails to meet the constraint even when the battery capacity is high, while ours always respects it. In some cases, the battery lifetime of our system may exceed the battery constraint (i.e. device operates a bit longer). The reason is that as the number of devices increases, our solution offloads more data and the constraints of the gateway force the devices to decrease their QoS level such that all devices can benefit from offloading. For instance, in Figure 4 instead of $x_4 = 400$ and $r_{4,4} = 320$, our solution has to select $x_3 = 300$ and $r_{3,2} = 120$ in order to meet the gateway's constraints. This leads to lower energy consumption and longer battery lifetime.

In the second scenario, the IoT devices in the unsupervised system select a low QoS level to make sure that the battery lifetime constraint is met. Figure 16 presents the average achieved QoS with our proposed solution compared to the unsupervised system. In a system with only a few devices (i.e. 2 or 3), the gateway resources are not scarce and therefore, our solution selects a high QoS level for the IoT devices. As the number of devices increases, the gateway's resources become saturated and thus our solution assigns a low QoS level to IoT devices. By employing our solution, the overall QoS of devices is at least 50% more than unsupervised system.

6 Conclusions

In this paper we studied the problem of QoS management in IoT systems, where the IoT devices can provide different QoS levels and can offload a share of their workload. The QoS management has to fulfill constraints for the battery lifetime of IoT devices, communication bandwidth to the gateway, and processing capability of the gateway (for offloading). We present an ILP formulation for this problem and decompose it into separate device and gateway problems. This allows

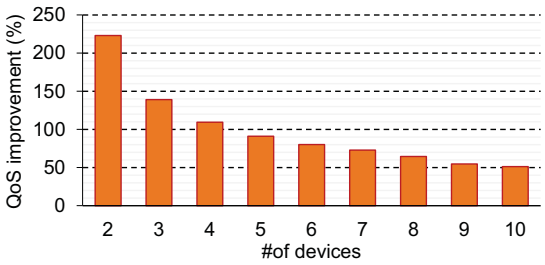

Figure 16: The accumulated QoS in our system compared to the unsupervised system for different number of devices

to reduce the search space and to distribute a part of the problem calculation to the IoT devices. The proposed solution benefits not only from reusing its sub-solutions (based on dynamic programming), but also from previous instances of the problem. We demonstrate the effectiveness of our proposed approach by using a case study of ECG processing in a personal healthcare monitoring application. The experiments show that our solution improves the overall QoS by 50% compared to an unsupervised system while both meet the constraints.

Acknowledgements

Authors would like to thank Mr. Sebastian Paul for his help in implementing the DP solution. This research has been partially supported by the E.C. funded program AEGLE under H2020 Grant Agreement No: 644906.

References

[1] Farzad Samie, Lars Bauer, and Jörg Henkel. IoT Technologies for Embedded Computing: A Survey. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2016.

[2] Shancang Li, Li Da Xu, and Shanshan Zhao. The internet of things: a survey. *Information Systems Frontiers*, 17(2), 2014.

[3] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012.

[4] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[5] Geng Wu, Shilpa Talwar, Kerstin Johnsson, Nageen Himayat, and Kevin D Johnson. M2M: From mobile to embedded internet. *IEEE Communications Magazine*, 49(4):36–43, 2011.

[6] Qian Zhu, Ruicong Wang, Qi Chen, Yan Liu, and Weijun Qin. IoT gateway: Bridging wireless sensor networks into internet of things. In *Embedded and Ubiquitous Computing (EUC)*, 2010.

[7] Ben Zhang, Nitesh Mor, John Kolb, Douglas S Chan, Nikhil Goyal, Ken Lutz, Eric Allman, John Wawrzynek, Edward Lee, and John Kubiatowicz. The cloud is not enough: saving IoT from the cloud. In *USENIX Conf. on Hot Topics in Cloud Computing*, pages 21–21, 2015.

[8] Farzad Samie, Lars Bauer, and Jörg Henkel. An approximate compressor for wearable biomedical healthcare monitoring systems. In *CODES+ISSS*, pages 133–142, 2015.

[9] SangJoon Lee, Jungkuk Kim, and Myoungho Lee. A real-time ECG data compression and transmission algorithm for an e-health device. *IEEE Transactions on Biomedical Engineering*, 58(9):2448–2455, 2011.

[10] Thomas Zachariah, Noah Klugman, Bradford Campbell, Joshua Adkins, Neal Jackson, and Prabal Dutta. The internet of things has a gateway problem. In *Mobile Computing Systems and Applications (HotMobile)*, pages 27–32, 2015.

[11] Moeen Hassanalieragh, Alex Page, Tolga Soyata, Gaurav Sharma, Mehmet Aktas, Gonzalo Mateos, Burak Kantarci, and Silvana Andreescu. Health monitoring and management using Internet-of-Things (IoT) sensing with cloud-based processing: Opportunities and challenges. In *IEEE International Conference on Services Computing (SCC)*, pages 285–292, 2015.

[12] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of Things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4), 2015.

[13] Sungwook Kim. Nested game-based computation offloading scheme for mobile cloud IoT systems. *Journal on Wireless Communications and Networking*, 2015(1):1–11, 2015.

[14] Amy Murphy and Wendi Heinzelman. Milan: Middleware linking applications and networks. Technical Report #795, University of Rochester, Jan. 2003.

[15] Z. Sheng, C. Mahapatra, V. Leung, M. Chen, and P. Sahu. Energy efficient cooperative computing in mobile wireless sensor networks. *IEEE Transactions on Cloud Computing*, 2015.

[16] Martin Moser, Dusan P Jokanovic, and Norio Shiratori. An algorithm for the multidimensional multiple-choice knapsack problem. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 80(3):582–589, 1997.

[17] David Pisinger. *Algorithms for knapsack problems*. PhD thesis, University of Copenhagen, 1995.

[18] Vandana Milind Rohokale, Neeli Rashmi Prasad, and Ramjee Prasad. A cooperative internet of things (IoT) for rural healthcare monitoring and control. In *Int'l Conf. on Wireless Communication*, pages 1–6, 2011.

[19] Nicola Bui and Michele Zorzi. Health care applications: a solution based on the internet of things. In *Int'l Symp. on Applied Sciences in Biomedical and Communic. Technologies*, 2011.

[20] Xin Liu, Yuanjin Zheng, Myint Wai Phyu, FN Endru, V Navaneethan, and Bin Zhao. An ultra-low power ECG acquisition and monitoring ASIC system for WBAN applications. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2(1):60–70, 2012.

[21] Daniele Bortolotti, Mauro Mangia, Andrea Bartolini, Riccardo Rovatti, Gianluca Setti, and Luca Benini. An ultra-low power dual-mode ECG monitor for healthcare and wellness. In *DATE*, pages 1611–1616, 2015.

[22] Victor Shnayder, Bor-rong Chen, Konrad Lorincz, Thaddeus RF Fulford Jones, and Matt Welsh. Sensor networks for medical care. In *SenSys*, volume 5, pages 314–314, 2005.

[23] Rubén Braojos, Giovanni Ansaloni, David Atienza, and Francisco J Rincón. Embedded real-time ECG delineation methods: A comparative evaluation. In *International Conference on Bioinformatics & Bioengineering (BIBE)*, pages 99–104, 2012.

[24] Roshan Joy Martis, U Rajendra Acharya, and Lim Choo Min. ECG beat classification using PCA, LDA, ICA and discrete wavelet transform. *Biomedical Signal Processing and Control*, 8(5):437–448, 2013.

[25] George B Moody and Roger G Mark. The impact of the MIT-BIH arrhythmia database. *IEEE Engineering in Medicine and Biology Magazine*, 20(3):45–50, 2001.

[26] Lisa Avila and Mike Bailey. The wearable revolution. *IEEE Computer Graphics and Applications*, 35(2):104–104, 2015.

[27] Shahab Ardalan, Siavash Moghadami, and Samira Jaafari. Motion noise cancelation in heartbeat sensing using accelerometer and adaptive filter. *IEEE Embedded Systems Letters*, 7(4):101–104, 2015.

[28] Matthew W. Hann. Ultra low power, 18 bit precision ecg data acquisition system, June 2013.

[29] Phil Smith. Comparing low-power wireless technologies. Tech Zone, Digikey Online Magazine, Digi-Key Corporation, 2011.

[30] Matti Siekkinen, Markus Hiienkari, Jukka K Nurminen, and Johanna Nieminen. How low energy is bluetooth low energy? comparative measurements with ZigBee/802.15.4. In *WCNCW*, pages 232–237, 2012.

[31] Thomas Dittrich, Chen Menachem, Y Herzel, and A Lou. Lithium batteries for wireless sensor networks. Technical report, Tadiran Batteries, 2012.

[32] Cong Zhu, Xinghu Li, Lingjun Song, and Liming Xiang. Development of a theoretically based thermal model for lithium ion battery pack. *Journal of Power Sources*, 223:155–164, 2013.

[33] Luca Mainetti, Luigi Patrono, and Antonio Vilei. Evolution of wireless sensor networks towards the internet of things: A survey. In *International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–6, 2011.