Dynamic Computation Migration at the Edge: Is There an Optimal Choice?

Sina Shahhosseini[1], Iman Azimi[2], Arman Anzanpour[2], Axel Jantsch[3], Pasi Liljeberg[2], Nikil Dutt[1], and Amir M. Rahmani[1,3]
[1]University of California Irvine, Irvine, CA, USA
[2]University of Turku, Turku, Finland
[3]TU Wien, Vienna, Austria
{sshahhos, dutt, a.rahmani}@uci.edu, {imaazi, armanz, pakrli}@utu.fi, axel.jantsch@tuwien.ac.at

ABSTRACT

In the era of Fog computing where one can decide to compute certain time-critical tasks at the edge of the network, designers often encounter a question whether the sensor layer provides the optimal response time for a service, or the Fog layer, or their combination. In this context, minimizing the total response time using computation migration is a communication-computation co-optimization problem as the response time does not depend only on the computational capacity of each side. In this paper, we aim at investigating this question and addressing it in certain situations. We formulate this question as a static or dynamic computation migration problem depending on whether certain communication and computation characteristics of the underlying system is known at design-time or not. We first propose a static approach to find the optimal computation migration strategy using models known at design-time. We then make a more realistic assumption that several sources of variation can affect the system's response latency (e.g., the change in computation time, bandwidth, transmission channel reliability, etc.), and propose a dynamic computation migration approach which can adaptively identify the latency optimal computation layer at runtime. We evaluate our solution using a case-study of artificial neural network based arrhythmia classification using a simulation environment as well as a real test-bed.

KEYWORDS

Internet of Things, Fog Computing, Computation Migration

ACM Reference Format:
Sina Shahhosseini, Iman Azimi, Arman Anzanpour, Axel Jantsch, Pasi Liljeberg, Nikil Dutt, and Amir M. Rahmani. 2019. Dynamic Computation Migration at the Edge: Is There an Optimal Choice?. In *Proc. of GLSVLSI (GLSVLSI '19)., May 9–11, 2019, Tysons Corner, VA, USA.* ACM, NY, NY, USA. 6 pages. DOI: https://doi.org/10.1145/3299874.3319336

1 INTRODUCTION

The number of IoT-enabled sensor devices are growing rapidly as we are entering the Internet-of-Things (IoT) era [1]. Today, a variety of sensor devices such as wearables are capable of interacting with the Internet, offering smart and connected solutions [2]. According to Cisco, 500 billion devices will be connected to the Internet by 2030, resulting in an exponential increase in the scale and the complexity of existing computing and communication systems [3].

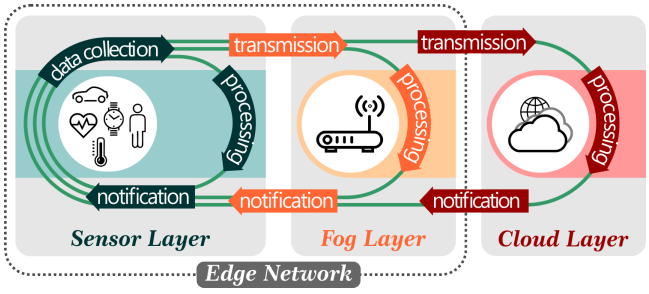

Figure 1: IoT System Architecture

Modern IoT systems often comprise of three main layers [1] as shown in Figure 1. At the first layer, a local sensor network is in charge of collecting sensory data and sending it to the next layer of interconnected smart gateways, also known as Fog layer [4]. This layer is a bridge between the cloud layer (i.e., the third layer) and the sensor network. It provides a set of services at the edge of the network to reduce latency and increase availability. Third is the cloud layer which includes data storage and analytics.

The resource constraints in sensor nodes in terms of computation and energy is often handled by offloading computation from sensor devices to the upper layers (i.e., Fog/Cloud). It has been shown that migrating the computation from sensor nodes to gateways or cloud can help the system to improve performance and energy consumption [4]. This is based on the assumption that the response time is mostly determined by the computation time and shifting the computation towards more powerful units can help reduce the total response time. However, this may not be always the case since such migration is a communication-computation co-optimization problem. More precisely, the response time of an application is not only a function of the execution time but also it depends on the transmission time when the computation of collected data is performed at the upper layers [5]. Therefore, any significant increase in transmission time leads to an increase in response time. Thus, a complex system under continuous change, application interference, environmental uncertainty needs to be dynamically controlled w.r.t. computational assignments of each layer to minimize response time and offer performance guarantees [6].

GLSVLSI '19, May 9–11, 2019, Tysons Corner, VA, USA

ACM ISBN 978-1-4503-6252-8/19/05.
https://doi.org/10.1145/3299874.3319336

The key contribution of this paper is twofold. We first present an abstract taxonomy of IoT applications and study the impact of full and partial computation migration at the edge on some of application classes to see if there is a static (design-time) optimal solution to this problem. Based on the observations made in the first part, we then propose a proof-of-concept dynamic computation migration approach for the edge sensor nodes and gateways considering the changes in the system characteristics and environment.

2 SYSTEM ARCHITECTURE AND APPLICATION CHARACTERISTICS

IoT devices are typically distributed across three layers [1]. The first layer is the physical layer, which is comprised of sensors that can sense and collect data from the environment. The second is the Fog computing (interconnected smart edge gateways/servers) layer, which is the intermediate computing layer between the cloud and sensor devices that complement the advantages of cloud computing by providing additional services that operate more closely to the sensor layer. Fog computing can perform certain computations mainly to take over some of the computation burden of sensor nodes and/or bringing latency-sensitive cloud analytics to the edge [4]. Another potential advantages of Fog computing is a significant reduction in the volume of data that needs to be transferred to the next layer(s) resulting in reduced transmission latency [7–10]. The third layer is the cloud layer providing unconstrained computing and storage resources. In this paper, we focus on the computation migration at the edge network referring to how data is process in the sensor layer and/or the Fog layer.

Response time is the total time which takes to respond to a request for a service. In a typical IoT architecture, a device in the sensor layer consists of one or more sensors and actuators. The sensors are responsible for collecting a set of data and the actuators are responsible for performing some specific actions (e.g., commands, notifications, etc.) according to collected data. In such systems, the data processing unit needs to analyze the data and decide the corresponding action. Here, the response time is the time difference between the moment when enough data for decision-making is gathered on the sensor node memory and the moment when a node has the actuation command ready-to-execute in hand [5]. Based on this scenario, the processing can happen on the sensor node microprocessor, gateway device processing unit, cloud server processors, or on a combination of these. Assuming that an IoT device is capable of sensing, local processing, and notification (e.g., a smart watches, smart home appliances, surveillance cameras, etc.) and it runs all the processes locally, the total response time is simply the time it needs to process and analyze the collected data:

$$Response\ Time_{total} = T_{comp}^{Sensor} \tag{1}$$

In contrast, if the sensor outsources all the computation to gateway(s) at the Fog layer, the total response time will include the data transmission time from the sensor device to the next layer and vice versa. Therefore, in the case of a two-layer sensor-gateway scenario the total response time is:

$$Response\ Time_{total} = T_{tran}^{stog} + T_{comp}^{Gateway} + T_{tran}^{gtos} \tag{2}$$

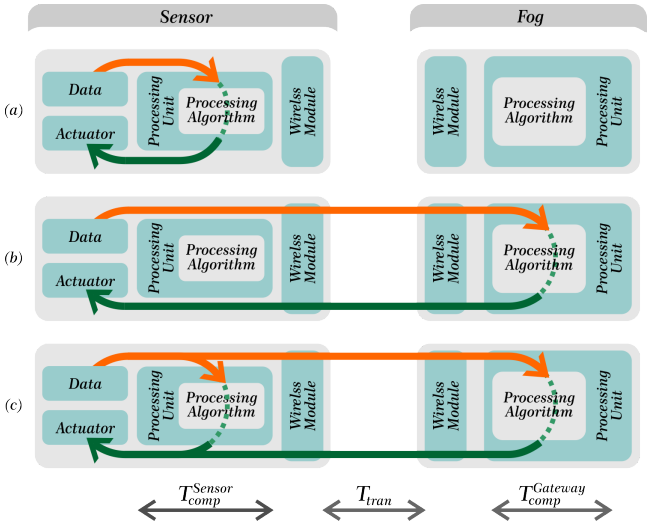

Figure 2: The data flow and response time calculation based on the computation scheme, a) Full computation at the Fog layer, b) Full computation at the sensor layer, and c) Partial computation at both sensor and Fog layer.

where t_{tran}^{stog} and t_{tran}^{gtos} are the sensor to gateway and gateway to sensor transmission time, respectively, and $T_{comp}^{Gateway}$ is the computation time needed on the gateway to complete the task.

If the processing load is distributed across multiple layers, Equation 2 will change. For instance, for a 2-layer sensor-gateway scenario, the response time will be as follows:

$$Response\ Time_{total} = T_{comp}^{Sensor} + T_{tran}^{stog} + T_{comp}^{Gateway} + T_{tran}^{gtos} \tag{3}$$

where T_{comp}^{Sensor} is the computation time needed on the sensor to complete a portion of a task while $T_{comp}^{Gateway}$ is the time needed to complete the remaining of the task. The ways response time is calculated in all these schemes are shown in Figure 2.

In addition, there exists another application characteristic which is often neglected during design space exploration. In IoT applications, the ratio of a generated data to the input data can vary, which significantly affects the response time. We define this characteristic as Output-Input Data Generation (OIDG) ratio and classify IoT applications based on it as follows:

(1) **Class 1:** Applications with extremely low OIDG ratio such as machine learning inference where a large fraction of input data is classified into often a handful of classes ($OIDG << 1$).
(2) **Class 2:** Applications with low OIDG ratio such as compression, down-sampling, etc. ($OIDG < 1$).
(3) **Class 3:** Applications with OIDG ratio of almost equal to 1, such as noise filtering ($OIDG \approx 1$).
(4) **Class 4:** Applications with high OIDG ratio such as encryption ($OIDG > 1$).

We discuss in the following section how these classes can impact the total response time when mapped to a layered IoT architecture. It should be noted that an IoT application, depending on its structure and use case, can be partitioned using two different ways:

(1) **Data-driven Partitioning:** In this case, data is divided into several partitions and these partitions are distributed among computing units. Each unit keeps the entire application code while computing a portion of data, similar to the scenarios shown in Figure 2. Partitioning of high dimension data presented in [11] can be an appropriate example of such method.
(2) **Task-driven Partitioning:** In this method, a multi-task application is partitioned into (parallel) tasks, and each task or a set of task is mapped to a computation unit. As an example of this method, an approach for partitioning convolutional neural network across edge devices is presented in [12].

In this paper, we focus on data-driven partitioning.

3 STATIC COMPUTATION MIGRATION

There exists a rich literature on how to offload computation from one layer to another or how to partition IoT applications temporally or spatially across different computations layers [12–14]. An example could be the work presented by Xu *et al.* [15] examining the effect of partitioning layers of a deep neural network between a mobile gateway and a battery powered IoT device on the overall response time.

Although executing an IoT application fully in the sensor node or in the Fog layer can both have several advantages and disadvantages, a proper decision cannot be made without considering the characteristics of the communication channel among them. In this section, we assume the inter-layer communication characteristics are static and known a priori and make an attempt to propose a partitioning approach for both full and partial computation migration scenarios. In the next section, we make a more realistic assumption by including the unpredictable nature of network latency in our problem formulation.

3.1 Full Computation Migration

Data is inherently generated at the sensor layer, but when there are constraints to process it locally, the data is migrated to the next layer (e.g., the edge server/gateway). In contrast to *data migration* to the next layer, which moves data to computation, *computation migration* refers to the scenario when computation is moved to data (e.g., to a sensor device). *Computation migration* and *data migration* are complementary which means one of them performs well when other one does not. However, different degrees of network and application latency can determine which scheme is the optimal solution in terms of performance and energy consumption. Assume that computing v bits of data on a sensor takes:

$$T_{comp}^{Sensor}(v) \tag{4}$$

and computing the same v bits of data on a gateway takes:

$$T_{comp}^{Gateway}(v) \tag{5}$$

Since the communication characteristics of the network are assumed to be static, the transmission time of v bits is:

$$T_{tran}(v) \tag{6}$$

where v is the number of transferred bits. Also consider that an application processes v bits raw data and produces v' bits as an output. Taking the above equations, the response time of computing the raw data on the sensor is obtained by the following formula:

$$T_{res}^{Sensor}(v) = T_{comp}^{Sensor}(v) \tag{7}$$

The response time of computing the raw data on the gateway is obtained by following formula:

$$T_{res}^{Gateway}(v, v') = T_{comp}^{Gateway}(v) + T_{tran}(v) + T_{tran}(v') \tag{8}$$

In Class 1 IoT applications, as the volume of generated data is very negligible, it can be removed and the the response time can be estimated using the following formula:

$$T_{res}^{Gateway}(v) = T_{comp}^{Gateway}(v) + T_{tran}(v) \tag{9}$$

In a static full computation migration approach, the lower response time at each side is simply the optimal solution. Therefore, the system determines the computation scheme by identifying the one leading to a shortest response time. For example, if at a given network latency $T_{res}^{Sensor}(v) > T_{res}^{Gateway}(v)$, then, the v bits of raw data are processed at the gateway layer, and the v' bits of output is transferred to the gateway. Full computation migration works well when v' is much less than v (i.e., Class 1 application).

3.2 Partial Computation Migration

Although the previous solution allocates all computations to the sensor-side or the Fog-side, there are specific applications in which dividing the processing between the sensor and gateway results in a lower latency. In such a data-driven partitioning, we process R portion of data on the sensor-side and the rest (1-R) on the Fog-side. For Class 1 applications, the result of processing (v') is very small in size and can be excluded from the transmission traffic. Therefore, the response time would be:

$$T_{res}^{Total}(v) = RT_{comp}^{Sensor}(v) + (1-R)T_{comp}^{Gateway}(v) + (1-R)T_{tran}(v) \tag{10}$$

We take the ratios between the components of this equation into account to have a better view of the solution and find the situation where the partial migration leads to lower response time. We consider sensor computation time to be α times and the transmission time β times longer than the gateway computation time:

$$T_{comp}^{Sensor} = \alpha T_{comp}^{Gateway} \tag{11}$$

$$T_{tran} = \beta T_{comp}^{Gateway} \tag{12}$$

We expect a gateway to process data faster than a sensor, but the transmission time can be shorter or longer than gateway's computation time, therefore $\alpha > 1$ and $\beta > 0$. In this case the total response time would be:

$$T_{res}^{Total} = R\alpha T_{comp}^{Gateway} + (1-R)T_{comp}^{Gateway} + (1-R)\beta T_{comp}^{Gateway} \tag{13}$$

We are interested in finding the relation between α and β, therefore, we simplify Equation 13 by considering $T_{res}^{Total}/T_{comp}^{Gateway}$ as a linear function of β:

$$T_{res}^{Total}/T_{comp}^{Gateway} = (R\alpha + 1 - R) + (1-R)\beta \tag{14}$$

For different R values, the resulting functions are a group of lines intersecting at [α-1, $\alpha\ T_{comp}^{Gateway}$]. As shown in Figure 3 (a), when the transmission time is smaller than α-1 times of the gateway

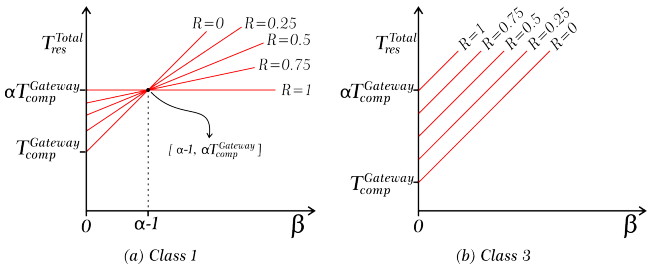

Figure 3: The parametric graph of total response time equation in Class 1 and Class 3 Applications.

computation time, performing a part of the calculation in the sensor node results in a shorter response time.

For Class 3 applications, the size of processed and unprocessed data is almost the same. Therefore, the response time would be:

$$T_{res}^{Total}(v) = RT_{comp}^{Sensor}(v) + (1-R)T_{comp}^{Gateway}(v) + T_{tran}(v) \quad (15)$$

Using similar coefficients for the sensor computation time and the transmission time would result in:

$$T_{res}^{Total}/T_{comp}^{Gateway} = (R\alpha + 1 - R) + \beta \quad (16)$$

As shown in Figure 3 (b), in Class 3 applications, any R value between 0 and 1 results in a response time between $T_{comp}^{Gateway}+\beta$ and $\alpha.T_{comp}^{Gateway}+\beta$; but the shortest response time belongs to a setting that performs all computations on the gateway (R = 0).

The aforementioned shows that the total response time is a function of β which is the transmission time coefficient. In other words, in a static partial computation migration scenario, the transmission rate defines the computation migration ratio which is the key to minimize the response time.

4 DYNAMIC COMPUTATION MIGRATION

An important challenge when using static computation migration is the variations in the system's behavior at run-time, in particular w.r.t. communication characteristics. Several sources of variation can affect the system's response latency such as the change in computation time, bandwidth, transmission channel reliability, and transmission latency due to the system's mobility and environmental changes. An alternative approach for such dynamic systems is to dynamically incorporate system's behaviour and context into the computation migration decision-making.

The Observe, Decide, and Act (ODA) control loop paradigm [16] is a proper strategy in this context to leverage real-time observations to dynamically tune system configuration. A classic ODA loop measures the system state and/or context (Observe), performs a decision-making w.r.t. the measurements (Decide), and applies possible changes (Act). In this section, we propose an ODA closed-loop dynamic computation migration approach implemented in the Fog layer to decide the optimal computation layer at runtime.

Our proof-of-concept dynamic computation migration approach is detailed in Algorithm 1. The algorithm assumes two computation status (i.e., C_{status}) as "computation at the sensor node" and "computation at the Fog layer" which is a simple flag indicating where computation is currently performed. If needed, the system

Algorithm 1 The dynamic computation migration algorithm

Initialize:
  $T_{comp}^{sensor} \leftarrow$ estimate computation time at the sensor node
  $C_{status} \leftarrow$ computation at the sensor node
while system is on **do**
  $T_{comp}^{gateway} \leftarrow$ estimate computation time at the Fog layer
  $T_{tran} \leftarrow$ measure transmission time between the sensor node and the Fog layer
  if C_{status} = computation at the sensor node **then**
    if $T_{comp}^{sensor} - \Delta/2 > T_{tran} + T_{comp}^{gateway}$ **then**
      $C_{status} \leftarrow$ computation at the Fog layer
    end if
  else if C_{status} = computation at the Fog layer **then**
    if $T_{comp}^{sensor} + \Delta/2 < T_{tran} + T_{comp}^{gateway}$ **then**
      $C_{status} \leftarrow$ computation on the sensor node
    end if
  end if
end while

dynamically changes the status based on 3 parameters to minimize the total response time. The first parameter is the computation time at the sensor (T_{comp}^{sensor}). This parameter is often fixed and can be measured at design-time. In contrast, the computation time at the Fog layer ($T_{comp}^{gateway}$), as the second parameter, can vary as it relies on the amount of computation capacity available at the moment. Therefore, it needs to be estimated at runtime. The third parameter is the transmission time between the sensor node and the Fog layer (T_{tran}). T_{tran} highly depends on several parameters such as distance, interference, objects in the environment, just to mention a few. There are well-established methods to estimate the network latency at runtime, for instance based on packet loss probability.

Our straightforward proof-of-concept algorithm iteratively measures/estimates $T_{comp}^{gateway}$ and T_{tran} values at runtime and compares their sum against the (T_{comp}^{sensor} value to determine the layer with the lower response time. To avoid unnecessary oscillations of computation migration between layers, it uses a simple threshold-based (i.e., soft margin Δ) strategy. It should be noted that our approach is proposed as a proof-of-concept, and more advanced and efficient control approaches (e.g., predictive modeling) can be used to make smarter decisions.

5 EXPERIMENTAL RESULTS

We present the evaluation of our proposed dynamic full computation migration approach in this section. We first present a case study which requires a rapid response in decision making. Then, we perform the evaluations in two different settings.

5.1 Case Study: Neural Network-based Arrhythmia Classification

We use a real-time arrhythmia classification approach as an exemplar for a time-critical health application, where the state of a patient's health deterioration is estimated via Electrocardiogram (ECG) signals. The approach is enabled via an IoT-based system to perform ubiquitous health monitoring in everyday settings for

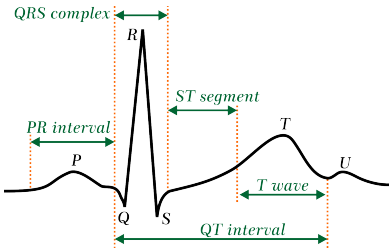

Figure 4: An ECG cycle with the 5 features

	Raspberry Pi 3 Model B	HP Compaq 8200 Elite
Processor	Quad-core Broadcom BCM2837 64bit	Quad-core Core i3 2100
Architecture	ARMv8-A	Intel Core
Speed	1.2 GHz	3.10 GHz
RAM	1 GB	16GB
External Storage	16 GB fast eMMC	250GB SATA HDD

Table 1: The device specifications

patients suffering from cardiovascular diseases. Such a system continuously acquires health data (i.e., ECG signal) from a user, implements a decision making (i.e., the classification approach), and sends the health decision to the user. These situations demand a rapid response time to alert the possibility of deterioration in health state, which in turn requires a reduction in response time.

We use "Long-Term ST Database" available on Physiobank [17], including normal ECG signals (patient with a normal health condition) and ECG signals showing arrhythmia (patient with a critical health condition). A binary classification is performed on the data, by which the normal ECG signals and ECG signals with arrhythmia are distinguished. The classifier in this approach is an Artificial Neural Network (ANN) method with one hidden layer and rectified linear unit (ReLU) as the activation function. The classifier is trained exploiting 12 hours of ECG signals, i.e., 6 hours of normal ECG and 6 hours of ECG with arrhythmia. In this regard, we, first, extract 5 different attributes from the ECG signal as QRS complex duration, ST segment duration, T wave duration, PR interval and QT interval [18]. Figure 4 shows an ECG cycle with the 5 features. The feature extraction is implemented via a cross-correlation between the ECG signal and a Triangular signal. Afterwards, the features are fed to the classifier.

In the following, the case study is evaluated first in a real-hardware setting and then in a more comprehensive simulation-based environments to assess the efficiency of our proposed dynamic full computation migration approach.

5.2 Evaluation on a Real Testbed

We conduct an experiment to indicate the dynamic full computation migration in a real-time health monitoring system. The system includes a sensor node and a gateway device, each of which is able to implement the decision-making approach. In the following, the details of our setup is presented.

5.2.1 Setup. We exploit a Raspberry Pi 3 as the sensor node and an HP Compaq 8200 Elite Linux machine as the gateway device. The latter device is equipped with a Quad-core Corei3 2100 CPU and 16GB RAM, which is considerably more powerful than the sensor node. The specifications of both devices are indicated in Table 1. Both the sensor and gateway devices run Apache web servers with PHP interpreters installed and are interconnected with a WiFi network. The reply duration of an HTTP request is used to measure the data transmission time.

Data collection is emulated in this setup, leveraging the existing dataset (i.e., "Long-Term ST Database"). The pre-recoreded data with 250 Hz sampling frequency are stored in files in the MicroSD card of the Raspberry Pi 3. Each file includes a 5-second window of (ECG) signal. The files are sequentially analyzed at the sensor node or transmitted to the gateway device. For the decision making, the binary classification is performed on the data (each segment).

5.2.2 Measurements. The dynamic computation migration is carried out to minimize the system's response time. In this regard, two different settings can be selected to perform the computation (i.e., decision making): at the sensor node or at the gateway device.

In the first setting, the sensor node performs the data collection and decision making (i.e., binary classification). Then, the decision is delivered to the user. As the senor node is attached to the user, the response time only includes T_{comp}^{sensor}: the computation time of the decision making at the sensor node. Running the classifier in 1000 iterations, α equals to 86.67 ± 3.61 ms.

In the second setting, the sensor node is in charge of the data collection and sending the data to the gateway device. The gateway performs the classification and sends the decision back to the user. The response time contains (1) data transmission time from the sensor node to the gateway device, (2) $T_{comp}^{gateway}$: computation time of the decision making at the gateway device , and (3) data transmission time from the gateway device to the sensor node. We consider (1) and (3) as the T_{tran}. Data flows in the two settings are indicated in Figure 2 (a) and (b). $T_{comp}^{gateway}$ equals to 18.39 ± 0.63 ms in this setup (measured in 1000 iterations). T_{tran} is not a fixed value and highly depends on the distance between the sensor node and gateway device. T_{tran} equals to 31.88 ± 4.92 ms when the sensor node is close to the gateway (i.e., < 2 meters). The value rises when the distance increases. High packet loss ratio in data transmission in long distances is one of the main reasons for such changes.

The response time of the two settings is indicated in Figure 5. When the two devices are close, the response time will be lower if the classification is performed at the gateway. However, when the distance exceeds 2.5 meters, the response time will be larger then the one of the first setting. The gateway dynamically generates a procedure for the system to reduce the response time, determining if the decision making should be performed at the sensor layer or at the gateway device. In this regard, a service is implemented at the gateway device, selecting one of the settings based to Algorithm 1.

5.3 Simulation-based Evaluation

We conduct an experiment to evaluate the dynamic full computation migration method in a more flexible simulation environment. To this end, we use open-source Cooja Simulator [19]. For our simulations, a Sky mote as a sensor device and an ARM A9-Cortex processor as a gateway device are used. The behavior of a gateway sensor is modeled and simulated in MATLAB with Star topology. In

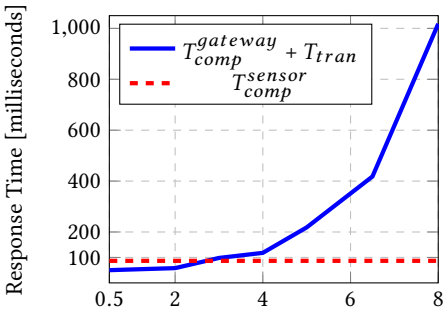

Figure 5: Response time of the system when the classification is performed at the gateway and at the sensor node.

the Star topology, all nodes are connected to a central connection point, such as a gateway.

The experiment shows a sensor-gateway pair running a neural network application (emulating our case study) using two scenarios: running on the sensor node or running on the gateway. The response time is measured with different packet loss ratio and the results are shown in Figure 6. As can be seen from the figure, the response time is almost constant when the application is computing fully at sensor layer, and increasing the packet loss ratio does not significantly increase the response time. However, as the gateway is more computationally powerful then the sensor node, when the packet loss is negligible the gateway response time is lower than sensor response time at the same packet loss ratio.

The experiments also evaluate the response time of a more complex system with many connected sensors to a gateway. Increasing the number of sensors can lead to a rise in response time in the Fog computing scheme if the we consider limited resources at the Fog layer. Whereas, increasing the number of sensors does not affect the response time in the sensor computing scheme. Thus, as shown in Figure 6, by increasing the number of sensors connected to the gateway, the Gateway response time exceeds the sensor response time. A dynamic computation migration solution can adapt to these variations at runtime by estimating the response time and finding the layer for computation which leads to the shortest response time.

6 CONCLUSIONS

This paper investigated the communication-computation co-design aspect of computation migration for the edge network of a typical 3-layer IoT architecture. Since the total response time does not depend only on the computational capacity of each layer, we presented a communication-computation co-optimization formulation to minimize the total response time. Furthermore, due to the uncertainty in network parameters and the number of IoT sensors, an absolute computation scheme is not an optimal solution. Therefore we also proposed a dynamic solution for computation migration at the edge. We evaluated our proposed scheme through several experiments using a real-time health monitoring application for arrhythmia classification from real-time ECG signals. Experiments show the optimal solution varies based on the network latency and the number of connected sensors to the Fog layer.

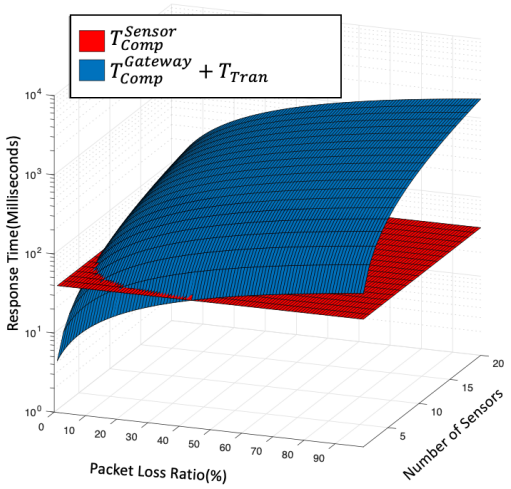

Figure 6: Response Time for different number of sensors and packet loss ratio (plotted based on the interpolation of a large set of discrete simulation results)

7 ACKNOWLEDGMENTS

This material is based upon work supported partially by the US National Science Foundation (NSF) WiFiUS grant CNS-1702950 and Academy of Finland grants 311764 and 311304.

REFERENCES

[1] L. Atzori et al. The internet of things: A survey. *Computer networks*, 54(15), 2010.
[2] F. Firouzi et al. Internet-of-Things and big data for smarter healthcare: From device to architecture, applications and analytics. *FGCS*, 2018.
[3] CISCO. *Internet of Things At a Glance*, 2016. https://www.cisco.com/c/dam/en/us/products/collateral/se/internet-of-things/at-a-glance-c45-731471.pdf.
[4] A.M. Rahmani et al. *Fog Computing in the Internet of Things - Intelligence at the Edge*. Springer, 2017.
[5] M.R. Nakhkash et al. Analysis of Performance and Energy Consumption of Wearable Devices and Mobile Gateways in IoT Applications. In *COINS*, 2019.
[6] C. Perera et al. Context Aware Computing for The Internet of Things: A Survey. *IEEE Communications Surveys Tutorials*, 16(1):414–54, 2014.
[7] R. Roman et al. A survey and analysis of security threats and challenges. *Future Generation Computer Systems*, 78, 2018.
[8] D. Amiri *et al.* Edge-Assisted Sensor Control in Healthcare IoT. In *IEEE GLOBECOM*, 2018.
[9] I. Azimi *et al.* Hich: Hierarchical fog-assisted computing architecture for healthcare iot. *ACM Transactions on Embedded Computing Systems*, 16(5), 2017.
[10] I. Azimi et al. Empowering healthcare iot systems with hierarchical edge-based deep learning. In *IEEE/ACM CHASE*, 2019.
[11] J. Zhou et al. An efficient multidimensional fusion algorithm for IoT data based on partitioning. *Tsinghua Science and Technology*, 18(4):369–78, 2013.
[12] S. Dey et al. Partitioning of CNN Models for Execution on Fog Devices. In *1st ACM Int. Workshop on Smart Cities and Fog Computing*, pages 19–24, 2018.
[13] T.N. Gia et al. Fog Computing in Body Sensor Networks: An Energy Efficient Approach. In *IEEE Int. Body Sensor Networks Conference*, 2015.
[14] A.M. Rahmani et al. Exploiting smart e-health gateways at the edge of healthcare Internet-of-Things: a Fog computing approach. *Future Generation Computer Systems*, 78(2):641–58, 2018.
[15] M. Xu et al. Enabling cooperative inference of deep learning on wearables and smartphones. *CoRR*, abs/1712.03073, 2017.
[16] H. Hoffmann et al. SEEC: A Framework for Self-aware Computing. Technical report, MIT, 2010.
[17] F. Jager et al. Long-term st database: a reference for the development and evaluation of automated ischaemia detectors and for the study of the dynamics of myocardial ischaemia. *Medical & Biological Engineering & Computing*, 2003.
[18] J.T. Catalano. *Guide to ECG Analysis*. Lippincott Williams & Wilkins, 2002.
[19] Osterlind, F. and others. Cross-level sensor network simulation with COOJA. *Proceedings. 2006 31st IEEE Conf. on Local Computer Networks*, 2006.