Information Systems 107 (2022) 101860

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

Exploring computation offloading in IoT systems

Sina Shahhosseini [a,*], Arman Anzanpour [b], Iman Azimi [b], Sina Labbaf [a], DongJoo Seo [a,c], Sung-Soo Lim [c], Pasi Liljeberg [b], Nikil Dutt [a], Amir M. Rahmani [a]

[a] University of California Irvine, Irvine, CA, USA
[b] University of Turku, Turku, Finland
[c] Kookmin University, Seoul, South Korea

ARTICLE INFO

Article history:
Received 15 May 2020
Received in revised form 29 November 2020
Accepted 1 July 2021
Available online 10 July 2021
Recommended by Gottfried Vossen

Keywords:
Internet of Things
Fog computing
Computation offloading

ABSTRACT

Internet of Things (IoT) paradigm raises challenges for devising efficient strategies that offload applications to the fog or the cloud layer while ensuring the optimal response time for a service. Traditional computation offloading policies assume the response time is only dominated by the execution time. However, the response time is a function of many factors including contextual parameters and application characteristics that can change over time. For the computation offloading problem, the majority of existing literature presents efficient solutions considering a limited number of parameters (e.g., computation capacity and network bandwidth) neglecting the effect of the application characteristics and dataflow configuration. In this paper, we explore the impact of the computation offloading on total application response time in three-layer IoT systems considering more realistic parameters, e.g., application characteristics, system complexity, communication cost, and dataflow configuration. This paper also highlights the impact of a new application characteristic parameter defined as Output–Input Data Generation (OIDG) ratio and dataflow configuration on the system behavior. In addition, we present a proof-of-concept end-to-end dynamic computation offloading technique, implemented in a real hardware setup, that observes the aforementioned parameters to perform real-time decision-making.

1. Introduction

The Internet of Things (IoT) has become a promising solution to improve efficiency or provide novel solution in many industrial areas, such as automotive, health-care, and home automation. Demand for comprehensive solutions, however, requires more complex IoT systems [1]. For this reason, IoT devices need to interact with cloud servers and other smart IoT devices via the Internet and local networks to offer smart and connected solutions [2]. According to Cisco, 500 billion connected devices are expected by 2030, which represents a substantial increase in the scale and the complexity of existing computing and communication systems [3].

Modern IoT systems comprise three main layers including sensor layer, fog layer, and cloud layer. These layers may include different sensing devices, computational and energy resources, storage capacity, and connectivity infrastructure [1]. The sensor layer is often in charge of data collection, leveraging a local sensor network. The sensor layer is resource-constrained in terms of processing power and energy budget. Some works have addressed such limitations by offloading computations to the upper layers (i.e., fog and cloud) [4]. Offloading computation is based on the assumption that (i) the response time is mostly determined by computation time and (ii) shifting the computation toward upper layers can reduce the total response time. However, this assumption may not always hold, as the migration is often a communication–computation co-optimization problem [5]. More precisely, the response time of an application, in addition to being the function of execution time, is a function of communication cost as well. In other words, any significant increase in the transmission time leads to an increase in the response time [6]. In this paper we only focus on exploring the impact of computation offloading on response time. However, power consumption is another important co-design aspects of computation offloading for IoT systems which needs to be investigated in the future works.

The response time is defined as the time difference between the moment when data is collected for decision making and the moment when the result is delivered to the consumer. Thus, different data flows from sensors to consumers may cause changes in the response time. Therefore, an end-to-end analysis of IoT systems that considers a service from data collection to actuation/ notification is essential. Furthermore, previous studies have neglected the impact of running applications with different characteristics in IoT systems. In other words, each application may

* Corresponding author.
E-mail address: sshahhos@uci.edu (S. Shahhosseini).

https://doi.org/10.1016/j.is.2021.101860

need different input data size and generate different output data size after the execution leading to different communication cost in IoT systems. We define this application characteristic as OIDG ratio (γ) referring to the ratio of output data volume to input data volume generated in an application. Even though there have been recently some efforts to co-optimize communication and computation in a coupled manner, the literature lacks a comprehensive solution that considers system parameters (e.g., application characteristic [OIDG] and dataflow configuration) in a holistic manner. The majority of existing literature only focuses on communication–computation optimization with limited system parameters such as processing capacity and network bandwidth [7–14].

In addition, finding optimal computation offloading policy for an unknown and dynamic system is critical since dynamicity of environment e.g., network condition, workload arrival at computing nodes, user traffic, and application characteristics changes over time. Most current solutions are based on design-time optimization suffering from the condition variation during the runtime [15–25]. A complex system that runs a variety of applications in uncertain environmental conditions requires dynamic control to offer high-performance or low-power guarantees [5].

In this paper, we explore computation offloading through a computation–communication co-optimization lens and in an end-to-end fashion. We highlight the impact of aforementioned system parameters (often neglected) on the response time. In addition, we examine a proof-of-concept dynamic computation offloading strategy in a case study to show IoT systems are required to be dynamically controlled at run-time. In short, we make the following key contributions:

- Explore the impact of the computation offloading on total application response time in three-layer IoT systems considering variety of parameters, e.g., application characteristic, system complexity, communication cost, and dataflow configuration.
- Investigate the impact of the new application characteristic parameter (i.e., OIDG) and the dataflow configuration on system behavior.
- Examine a proof-of-concept end-to-end dynamic computation offloading solution considering the aforementioned influential parameters. We compare the dynamic strategy with three static computing schemes to show the design time models are insufficient for IoT systems with dynamic behavior.

The rest of this paper is organized as follows: Section 2, briefly introduces modern IoT architecture and highlights the advantage of the three-layer IoT architecture and computation offloading technique. In Section 3, we formulate the system behavior in two common Dataflow configurations. In Section 4, we explore an impact of computation offloading on different parameters. In Section 5, we examine the dynamic computation offloading system. Finally, we conclude the paper in Section 6.

2. Background

In this section, we first briefly describe the three-layer IoT architecture. Then, we outline the background of computation offloading in IoT systems.

2.1. Three-layer IoT architecture

An IoT-based system conventionally includes a three-layer architecture. The devices and processing units are located in the three layers, requiring communication methods according to their functions and physical or virtual locations [1]. In general and from the functionality perspective, the lower layer – including sensing, actuating, and informing devices – is entitled as the sensor layer. The upper layer that contains computing and storage devices is the cloud layer, and the intermediate devices is the fog layer (see Fig. 1).

The sensor layer consists of sensor nodes that collect data from the physical world. The devices may also provide actuation control or notification as a result of processing the collected data. The sensing devices are expected to be relatively small, connected to the rest of the system wirelessly. To communicate with the other layers, a sensor node requires a transmission module as well as a small and power-efficient processing unit. Due to the limited battery, the sensing devices are constrained by their energy sources, processing powers, and data transmission capacities.

The fog layer is defined as an intermediate layer to regulate the data transmission and processing loads, since constant direct communication with the server may not be feasible or reasonable for resource-constrained sensing devices. The fog layer consists of one or more gateway devices that are physically close to the sensing devices [4]. This short distance to the data sources significantly reduces the data transmission power consumption of the sensing devices and enables fast responses. The processing power of the gateway devices in the fog layer is higher than the sensing devices, enabling local data analysis before sending the data to the cloud server. This pre/local processing method, which is known as "fog computing", reduces data traffic, the load on the cloud server, and the response time. The gateway and sensing devices are connected to a local network. Therefore, the system is still able to provide the service in case of loss of Internet connection (i.e., connection to the cloud servers) [26–29].

At the cloud layer, the main processing unit of an IoT-based system is the cloud server. The cloud servers are considered as devices with more powerful processing capability [30]. Thanks to a centralized database of the incoming data, heavy data analytic approaches can be performed on a specific source of data, allowing a comparison with other data sources.

In this paper, we focus on the computation offloading in the IoT-based systems, referring to how data is processed and transmitted over the IoT network: the sensor, fog, and cloud layers.

2.2. Computation offloading

Resource allocation is classified into three main categories as resource placement, resource scheduling, and computation offloading. Resource placement is about where and how resources are placed in IoT systems. It aims to find optimal set resources in IoT systems to execute tasks or applications while satisfying QoS (Quality of Service) requirements by optimizing specific objective function (minimizing latency, minimizing energy consumption, etc.). Resource scheduling or scheduling in resource allocation is to determine when and how many resources to allocate in IoT systems. The resource scheduling determines optimal scheduling of tasks, services, or applications to be executed on resources in order to meet QoS requirements [31].

Computation offloading is to determine where and how many resources can be moved to execute tasks or applications. The technique in IoT context is the transfer of resource-intensive computational tasks to a separate external device in the network. The technique of offloading computation over a network can provide computing power and overcome the limitation of an IoT-based device such as computational power, storage, and energy. Optimizing the computation offloading problem can be addressed by finding the best solution for the following questions:

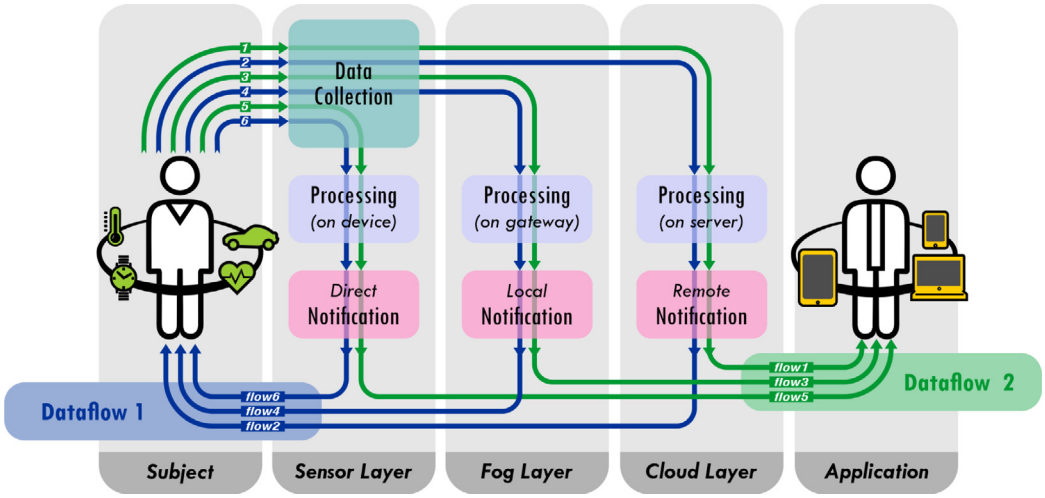

Fig. 1. IoT System Architecture.

Table 1
Notations descriptions.

Notation	Description
A_i	Application i to be executed on S_i which is defined as a tuple $\{D_{in}, D_{out}\}$
α	Decision offloading tuple
β	Packet loss ratio in connectivity between the sensor layer and fog layer
γ	OIDG ratio in application A
T_t^{sg}	Transmission time between the sensor layer to gateway layer
T_t^{gs}	Transmission time between the gateway layer to sensor layer
T_t^{cg}	Transmission time between the cloud layer to gateway layer
T_t^{gc}	Transmission time between the gateway layer to cloud layer
T_t^{ca}	Transmission time between the cloud layer to application layer
T_e^S	Execution time at the sensor node
T_e^G	Execution time for arrival workloads at the gateway device
T_e^C	Execution time for arrival workloads at the cloud device
N	Number of connected sensors
D_{in}	Input data size in A
D_{out}	Output data size in A
T_r^s	Total response time in the sensor computing scheme A
T_r^g	Total response time in the gateway computing scheme A
T_r^c	Total response time in the cloud computing scheme A

(i) where to offload: the scheduler should determine where the computation is offloaded, depending on the variety of parameters such as objectives, availability of resources, and required computation capacity for performing computations. Some studies have addressed this problem with optimally offloading workloads to more capable computing resources [16,23].

(ii) when to offload: the scheduler should determine when the computation is offloaded to upper layers to achieve the required QoS due to many uncertainties in the system and environment such as network congestion, overloaded workload, and device battery. Some studies have addressed this problem with optimal time scheduling of offloading computations [13,32].

(iii) what to offload: the computation scheduler should determine what portion of workloads is offloaded to upper layers. According to this fact, offloading solutions can be classified into two classes: (i) full offloading where the whole workloads are offloaded to external resources such as fog or cloud layers and (ii) partial offloading where workloads are partitioned into parts to be executed locally or externally. Many works have been addressed this optimization problem by the partial offloading method such as [8,14,25,33]. Besides, some studies that have applied the full offloading method in their problems [7,15].

3. System model

In this section, we investigate response time behavior in an three-layer IoT system which uses the computation offloading technique. First, we briefly explain the system components and then we describe the computation offloading model which is applied to our system. Finally, the response time is modeled in two common dataflow configurations. The computing devices in three-layer IoT system are represented by (S,G,C) where S is the sensor layer with n sensor devices; G is a gateway device at the fog layer; C indicates a cloud device. Each sensor run an application in a certain period of time. These applications are represented by a tuple $A_i = \{D_{in}, D_{out}\}$, where D_{in} is the amount of input data required for each application and D_{out} is the output data of each application. All the notations are described in Table 1.

3.1. Computation offloading model

As previously mentioned, sensor devices are supposed to execute application A_i in a certain period of time. Computation offloading model define whether the application should be uploaded to the upper computing resources or be performed at the local resource. The offload decision for the system is represented by a tuple $\alpha = \{\alpha^S, \alpha^G, \alpha^C\}$ where α^j represents offloading decision at layer j. For example, if the sensor layer executes the applications at layer $j \in \{S, G, C\}$ then, $\alpha^j = 1$, otherwise it equals to zero.

3.2. Response time model

In general, response time is the total time it takes to respond to a request for a service [6]. In IoT systems, the response time is defined as the time difference between the moment when enough data is collected for decision-making and the moment when the result is delivered to the consumer [6]. In general, the actuators can be located at different layers in IoT systems which leads to different dataflows from sensors to actuators. In this section, response time model is investigated for two common dataflows as follows:

Dataflow 1: The actuators (i.e., consumers) are located at the sensor layer, so the action response is issued at the processing unit and is performed at the sensor layer. Based on this scenario, the data analysis can be performed at the three layers: the sensor node processing unit, the gateway device processing unit, and the cloud processing unit. Assuming the sensor node is capable of

performing local processing, the response time is obtained from the following equation:

$$T_r^S = T_e^S \tag{1}$$

where T_e^S is the execution time of analyzing raw data at the sensor layer. In contrast, if the sensors offload the analyzing computation to the gateway layer, the total response time will include the data transmission time from the sensor layer to the gateway layer and the notification transmission time from the gateway to the actuators. Therefore, in the case of a two-layer sensor gateway, the total response time is as follows:

$$T_r^G = T_{tran}^{sg} + T_e^G + T_{tran}^{gs} \tag{2}$$

where T_{tran}^{sg} and T_{tran}^{gs} are the sensor-to-gateway and gateway-to-sensor transmission time, respectively. T_e^G is the execution time needed to analyze transmitted data on the gateway. If the computation needs to be performed with a more powerful processing unit, sensors will offload the computation to the cloud layer, making the total response time equal to the round trip transmission time from the sensor layer to the cloud layer. This time can be obtained as:

$$T_r^C = T_{tran}^{sg} + T_{tran}^{gc} + T_e^C + T_{tran}^{cg} + T_{tran}^{gs} \tag{3}$$

where T_{tran}^{sc} and T_{tran}^{cs} are the sensor-to-cloud and cloud-to-sensor transmission time, respectively, and T_e^C is the execution time at the cloud layer.

Dataflow 2: the actuators can be physically distant from the sensor nodes. A network of sensors can collect data to be analyzed through the layers and notify the end-user on the application layer. Tele-monitoring IoT systems usually follow this type of dataflow. In this regard, the result (e.g., notification) is transmitted from the cloud layer to an end user, assuming the end user is connected to the cloud layer. According to where the processing of collected data is performed, the total response time can be obtained from one of the following equations. Assuming the sensor node performs the processing at its local processing unit, the total response time is the following:

$$T_r^S = T_e^S + T_{tran}^{sg} + T_{tran}^{gc} + T_{tran}^{ca} \tag{4}$$

where T_{tran}^{ca} is the cloud-to-end user transmission time. In contrast, if either the gateway or cloud performs the computations, the total response times can be obtained from the following equations, respectively:

$$T_r^G = T_{tran}^{sg} + T_e^G + T_{tran}^{gc} + T_{tran}^{ca} \tag{5}$$

and,

$$T_r^C = T_{tran}^{sg} + T_{tran}^{gc} + T_e^C + T_{tran}^{ca} \tag{6}$$

Therefore, the response time in both dataflows for request from the sensor layer with $\alpha = \{\alpha^S, \alpha^G, \alpha^C\}$ as offload decision tuple can be summarized into the following equation:

$$T_r = \alpha^S.T_r^S + \alpha^G.T_r^G + \alpha^C.T_r^C \tag{7}$$

where only one α^j equals to one while the others are zero. The lowest response time can be obtained by choosing the optimal computing scheme for IoT systems in different system conditions. The offload decision tuple determines the computing scheme for the system to obtain the lowest response time in the design time or dynamic solution.

4. Exploring computation offloading space

A complex Internet of Things system needs to be systematically designed in the initial stages of the design to operate optimally between available design alternatives. This requires an exploration of all influential parameters on system behavior. System behavior depends on several factors that many studies have tended to neglect. In the previous section, we investigated the response time in three-layer IoT systems, and modeled the response time in different dataflow configurations. This section presents a design space exploration for computation offloading in IoT systems, showing impact the following parameters on the response time:

- **Communication:** offloading computations to upper layers in IoT systems requires transmitting data through the layers. Therefore, communication has a significant impact on the total response time. We consider the packet loss ratio (β) as a parameter which impacts communication cost between the devices [34].
- **System Complexity:** execution time might vary depending on what amount of workloads arrives at the computing resource or the number of requested services. Therefore, investigating the response time must include the effect of system complexity at design time or run-time.
- **Dataflow:** the locations of end-users can change the communication cost. Therefore, any change in dataflow from where the data is collected, processed, and delivered can directly impact system behavior. In this paper, we investigate two common dataflows where in Dataflow 1, the actuators (i.e., consumers) are located at the sensor layer, and in Dataflow 2, actuators are physically distant from sensor nodes and located at the application layer.
- **Application Characteristic:** previous studies have neglected the impact of application characteristics in design space explorations when a proper computation layer needs to be determined. Response time not only depends on system characteristics (e.g., data flow) but also on application characteristics. The ratio of generated output data to input data can vary, which can affect the response time. One application characteristic is the OIDG ratio (γ). To show the impact of this application characteristic on system behavior, this study examines the following four manifestations of various OIDG regions: **(i)** an application from the region of extremely low OIDG ratios, such as machine-learning classifiers. A large volume of input data can be classified into a few classes ($\gamma \ll 1$). **(ii)** an application from the region of low OIDG ratios, such as compression or down-sampling ($\gamma < 1$). **(iii)** an application from the region of OIDG ratios almost equal to 1, such as noise filtering where the sample per second is unchanged ($\gamma \simeq 1$). **(iv)** an application from the region of extremely high OIDG ratios, such as encryption or video decoding ($\gamma > 1$).

Table 2
Average bandwidth for download and upload in different networks.

Network	Download speed(Mbps)	Upload speed(Mbps)
3G	2.0275	1.1
4G	13.76	5.85
Wi-Fi	54.97	18.88

In the following subsections, we conduct an experiment to investigate the dependency of IoT systems on the mentioned parameters.

4.1. Experimental setup

We exploit Raspberry Pi Zero as the sensor device and NVIDIA Jetson TX2 as a gateway device. In addition, AWS c5.4xlarge node is used as a cloud layer. The gateway is equipped with integrated

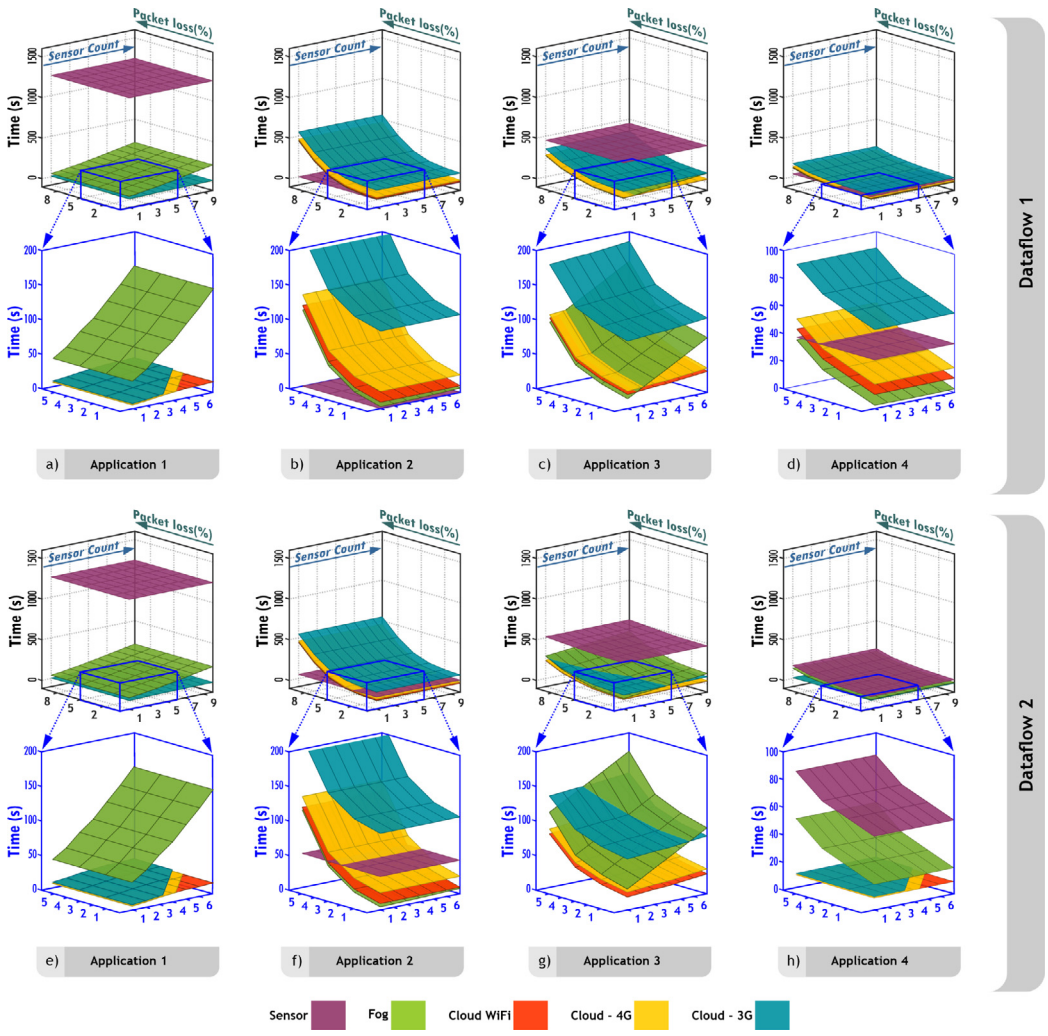

Fig. 2. The Computation Offloading Exploration for different environmental parameters and system complexity for chosen applications in two different dataflows.

Table 3
Applications' Specifications.

Application #	Application name	OIDG ratio	Input size(Byte)	Output size(Byte)	Description
1	YOLO	0.001	164000	161	Image object detector
2	Canny Edge Detector	0.043	14000000	615000	Video edge detector
3	Video-denoising	1	8200000	8200000	Non-local denoising application
4	Video-decoder	95.141	145536	13846510	Image object detector

Table 4
Platforms specifications.

	Raspberry Pi Zero	NVIDIA Jetson TX2	AWS c5.4xLarge
Processor	Single core	Quad-core Cortex-A57	16 Core Intel Xeon 8000
Architecture	ARM	ARM	X86
Speed	1 GHz	2.0 GHz	3.2 GHz
GPU	–	256-core Pascal	—
RAM	512 MB	8 GB	32 GB
External storage	16 GB eMMC	32 GB eMMC	512 GB SSD

256-core NVIDIA Pascal GPU and hex-core ARMv8 CPU, which is substantially more powerful than the sensor node. However, the cloud layer has more powerful computing units, such as a 16-Core Intel Xeon Platinum 8000 processor. Table 4 contains the specifications of all devices. The sensor and gateway use MQTT protocol to communicate with each other [35]. The cloud node, in contrast, runs Gunicorn with Flask, which is a micro web framework written in Python. The gateway is also connected to the AWS instance via cellular network or Wi-Fi. Table 2 lists the average bandwidth for each connectivity protocol [36].

4.2. Evaluation

We conduct experiments to indicate the system with applications selected in different OIDG regions. The described system is investigated through a variety of environmental conditions, application characteristics, dataflow configurations, and system complexity as follows. Four applications with different characteristics are chosen as explained in Table 3. The application's executable file is located at all the computing nodes. Therefore, once the offloading decision determines the computing scheme, the corresponding node executes the application from its local memory. System complexity varies from a single node to ten

nodes in the same network. Dataflows are selected from the configurations, as explained in the System Model section. Communication over a Wi-Fi network is affected by packet loss ratio, which varies within a certain range of 0% to 14% in our investigation. The packet loss ratio is targeted within the range as any ratio more 14% would dramatically increase the communication delay and response time consequently. In addition, communication between the fog layer and the cloud layer is based on 3G, 4G, or Wi-Fi connectivity.

Fig. 2 shows the result of design space exploration considering the mentioned parameters. In the figure, each plot shows an exploration for a specific application type and dataflow configuration. Besides, each plane represents a specific computation offloading policy. For example, the green planes represent the response time (Z-axis) when all sensors offload workloads to the fog node while the yellow planes represent the cloud computing with 4G connectivity to the fog layer. In each plot, X-axis and Y-axis represents the packet loss ratio and number of connected sensors respectively.

The total response time is significantly affected by transmission time across the layers. The system running Application 2 on Dataflow 2 (see Fig. 2.f) is remarkably affected by variation in packet loss ratio. In the low packet loss ratio, fog computing is the optimal solution among other schemes as the green plane is below the others. However, in high packet loss ratio sensor layer computing is the optimal solution. In this case, an increase in packet loss ratio leads to a significant increase in response time showing the advantage of sensor layer computing in some conditions. On the other hand, the packet loss ratio does not affect the optimal solution in the system running Application 1 with both dataflow configurations (see Fig. 2.a and e).

Furthermore, simultaneous requests to a computation unit can severely interrupt request processing, which affects response time. Therefore, in complex systems, as the number of connected sensors increases, there is a possibility of significant change in system behavior. As our experiments show in Fig. 2, the system running Application 3 on Dataflow 1 (see Fig. 2.c) is remarkably affected by increasing the number of connected sensors. The fog computing scheme is the optimal solution when number of connected sensor equals to 1. However, it becomes the cloud computing scheme for the system with more connected sensors. In the case of simultaneous requests, the fog layer is more sensitive than the cloud layer, which leads to changing the system behavior in a more complex system.

In addition, the consumer's location affects the response time in our experiment. The system running Application 4 has different behavior in both dataflows. In this case, cloud-3G computing is the worst solution for Dataflow 1 (see Fig. 2.d), while it can be the optimal solution with Dataflow 2 (see Fig. 2.h), implying that, for the same environmental parameters and system complexity, a different flow of data can change the optimal solution. In another case, the system running Application 2 with Dataflow 1 (see Fig. 2.e) performs computation optimally in the sensor layer. However, that might not be always true for Dataflow 2.

The OIDG ratio (γ) as the application characteristic remarkably affects system behavior. In other words, a system with a higher ratio has more communication costs delivering the outputs to the consumer. For example, running Application 2 with Dataflow 1 (see Fig. 2.b) over the sensor layer is more suitable while running Application 4 with Dataflow 1 (see Fig. 2.d) on the upper layers is faster. In addition, the system behavior is more sensitive to packet loss when OIDG is high. Consequently, various parameters affect IoT systems, and they need to be considered to design the model.

Algorithm 1 The dynamic computation offloading decision-making in the *Decide* component

Initialization in design time:
  $\vec{\alpha} \leftarrow \{1, 0, 0\}$
  ▷ $\vec{\alpha}$ represents the decision offloading tuple for all sensors: {1,0,0} (in sensor), {0,1,0} (in gateway) and {0,0,1} (in cloud)
while system is on **do**
  Mode ← Dataflow mode ▷ i.e., 1 or 2 selected by the user
  if *Mode* = 1 **then**
    From the *Observe*:
      Collect T_e^S, T_e^G, T_e^C, T_{tran}^{sg}, T_{tran}^{gc}, T_{tran}^{cg}, and T_{tran}^{gs}
  else if *Mode* = 2 **then**
    From the *Observe*:
      Collect T_e^S, T_e^G, T_e^C, T_{tran}^{sg}, T_{tran}^{gc}, T_{tran}^{ca}
  end if
  Calculate T_r^S, T_r^G, and T_r^C
  Update $\vec{\alpha}$ based on $\{\min(T_r^S, T_r^G, T_r^C) - \Delta\}$
  To the *Act*:
    Send $\vec{\alpha}$ for the actuation
end while

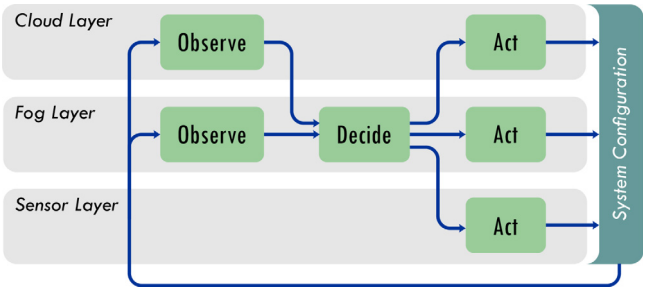

Fig. 3. The ODA control loop in the three-layer IoT system [37].

5. A proof-of-concept dynamic computation offloading technique

In the previous section, we examined the response times of IoT systems, showing how communication and computational limitations could influence different applications. Our experiment demonstrated the behavior of IoT systems leveraging different computational resources in three-layer architecture. Such an investigation is necessary for system evaluation in design time and the provision of optimal computation schemes for stationary conditions; however, it is insufficient in the dynamic runtime environment.

The behavior of IoT-based systems dramatically changes during runtime due to several variations in status and context. For example, degraded Internet connection and uncertainty in network latency have an impact on transmission latency. Moreover, the computation time of an application varies considerably due to varying computation loads on different system nodes (e.g., gateway devices).

5.1. System design

In this sub-section, we present and examine a simple proof-of-concept end-to-end dynamic computation offloading technique tailored for IoT-based systems. The proposed approach selects the near-minimum computation offloading scheme according to the system's condition. As mentioned earlier, there are two dataflow modes in these systems, since the end-user can be considered in the vicinity of the sensor layer or in the cloud layer (see Fig. 1). Therefore, in each iteration, the proposed method is performed to minimize the latency of mode 1 or mode 2. The proposed dynamic

computation migration approach employs a closed-loop control strategy to minimize response latency. We exploit the Observe, Decide, and Act (ODA) control loop in this regard to leverage the system's conditions into the computation migration decision-making [37]. A conventional ODA paradigm includes three major components. The *Observe* component captures the system status, enabling self- and context-awareness in the system. The *Decide* component performs the decision-making according to the measurements obtained in the Observe component. Finally, the *Act* component fulfills the decision, making the (required) changes in the system accordingly [38].

The ODA paradigm, as the backbone of the proposed computation migration approach, is positioned in the three-layer IoT systems, as shown in Fig. 3. The *Observe* component is distributed into the fog and cloud layers, as the transmission and computation loads are dynamic and should be collected continuously. Note that computation time at all nodes is measured in the design time to be used for the run-time decision making. The *Decide* component is fully positioned at the fog layer, enabling local computation migration decision-making. The actuation of this approach is carried out in all three layers, since the computation can be performed in the sensor network, gateway devices, or cloud server. Therefore, the *Act* component is partitioned into the three layers.

The computation offloading decision-making, performed in the *Decide* component, is indicated in Algorithm 1, which indicates how the computation is dynamically switched between a sensor node, a gateway device, and a cloud server in our system. In design time, $\vec{\alpha}$ is initialized as $\{1, 0, 0\}$ for all sensors, and the computation time of the sensor, T_e^S, is measured. When the system is on (i.e., runtime), the algorithm first checks the dataflow mode selected by the user. As mentioned earlier, the response time of the system is different in the two modes. According to the selected mode, the required values are collected from the *Observe* component. Then, the response time of the three layers, T_r^S, T_r^G, and T_r^C, are calculated via Eqs. (1)/(4), (2)/(5), and (3)/(6). By obtaining the minimum response time, $\vec{\alpha}$ is updated and sent to the *Act* component for actuation in the next iteration. The possible permutations for $\vec{\alpha}$ are limited to $\{1, 0, 0\}$, $\{0, 1, 0\}$, $\{0, 0, 1\}$. Therefore, the computational complexity for the decision making is $O(1)$. It is worth mentioning that the system might oscillate between two layers if the response time values are proximal. To avoid such unnecessary oscillations, we consider a simple threshold strategy by adding a soft margin (i.e., Δ) in the computation of the minimum layer's response time. The soft margin is defined based on the application sensitivity and the quality of the communication. In this work, we selected 10% of the current response time as the soft margin. Therefore, α is changed if the response time in the layer with computation subtracted by Δ is greater than the other layers' response times (line 10 in Algorithm 1). The runtime dynamic computation offloading system leverages the ODA loop to minimize the response time during the runtime. The system dynamically assigns the computation to the layers by observing environmental parameters, system complexity, etc.

5.2. Evaluation

This subsection evaluates the dynamic system under various uncertain complexity and environmental conditions. In the envisaged scenario that executes over 24 h, packet loss varies between 0% and 14% and the number of sensors connected to our network changes between 1 and 10. According to the Exploring Computation Offloading Space section, the system which executes Application 3 in Dataflow 2 behaves more sensitive to the mentioned influential parameters (see Fig. 2.g). Therefore, in our evaluation, the system is supposed to execute Application 3 during the experiment within Dataflow 2. The dynamic system assigns the computation to one of the layers using Algorithm 1, as shown in Fig. 4. On the other hand, other computing schemes are static over the run-time where variations in the system parameters such as packet-loss or systems complexity do not change the computing scheme. The response time is evaluated with both the static and dynamic approaches. In the scenario, the dynamic solution begins with performing computation at the cloud layer. However, once the packet loss and the number of connected sensors changes, the dynamic system assigns the computation to the fog layer. On the other hand, the static approaches do not consider the condition variations where it leads to run the IoT system inefficient.

As a result, the system executes the application at the sensor, fog, and cloud layers with 289 s, 321 s, and 512 s average response time respectively and through dynamic solution with 268 s average response time, which demonstrates that better performance can be achieved through the dynamic solution. Fig. 4 shows that the sensor layer computing is sometimes better than the others, due to the packet loss and system complexity. On the other hand, cloud layer computing is the optimal computing scheme where communication cost is low. These findings verify that static computing scheme would perform inefficient in compare with dynamic solution. Therefore, many parameters affect the IoT system, and they need to be considered. The proof-of-concept 24 h test highlights the importance of considering the mentioned parameters for dynamic systems. It shows the design time models are insufficient for IoT based systems with dynamic behavior as the context and condition change. Self-aware optimization methods are required at run time, considering such dynamic and unpredictable conditions of IoT systems.

6. Related work

This section illustrates related works in the computation offloading technique in IoT systems. Computation offloading is a technique to transfer compute-intensive tasks or applications from the resource constrained IoT devices to more computing capable nodes to enhance QoS specially for IoT latency-sensitive applications. In this section we explain efforts to address the problem. Even though there have been recently some efforts to co-optimize communication and computation in a coupled manner, the literature lacks a comprehensive solution that highlight the impact of various system parameters (e.g., application characteristic [OIDG] and dataflow configuration).

Zhang et al. [15] design an energy-efficient computation offloading algorithm. The work formulates an optimization problem to minimize energy consumption of the offloading system by considering computation task offloading and its data transmission in the offloading decision. The optimization problem is reduced with a simplification method to polynomial complexity and solved in numerical method. Cao et al. [16] investigate a joint computation and communication user cooperation for Mobile Edge Computing (MEC) where a nearby helper node is enabled to share its resource to improve QoS. The work proposes a energy-efficient framework to minimize the total energy consumption at both the user and the helper using convex optimization. You et al. [23] propose a design for mobile cooperative computing that enables a user to exploit non-causal CPU-state information shared by offloading computation to a nearby helper. It helps to fully utilize random computation resources at the helper with minimum energy consumption.

Sheng et al. [24] present a joint optimization problem of computation and communication in the MEC system to optimally partition, offload and execute tasks between sensor nodes and edge nodes to minimize energy consumption. Kao et al. [39]

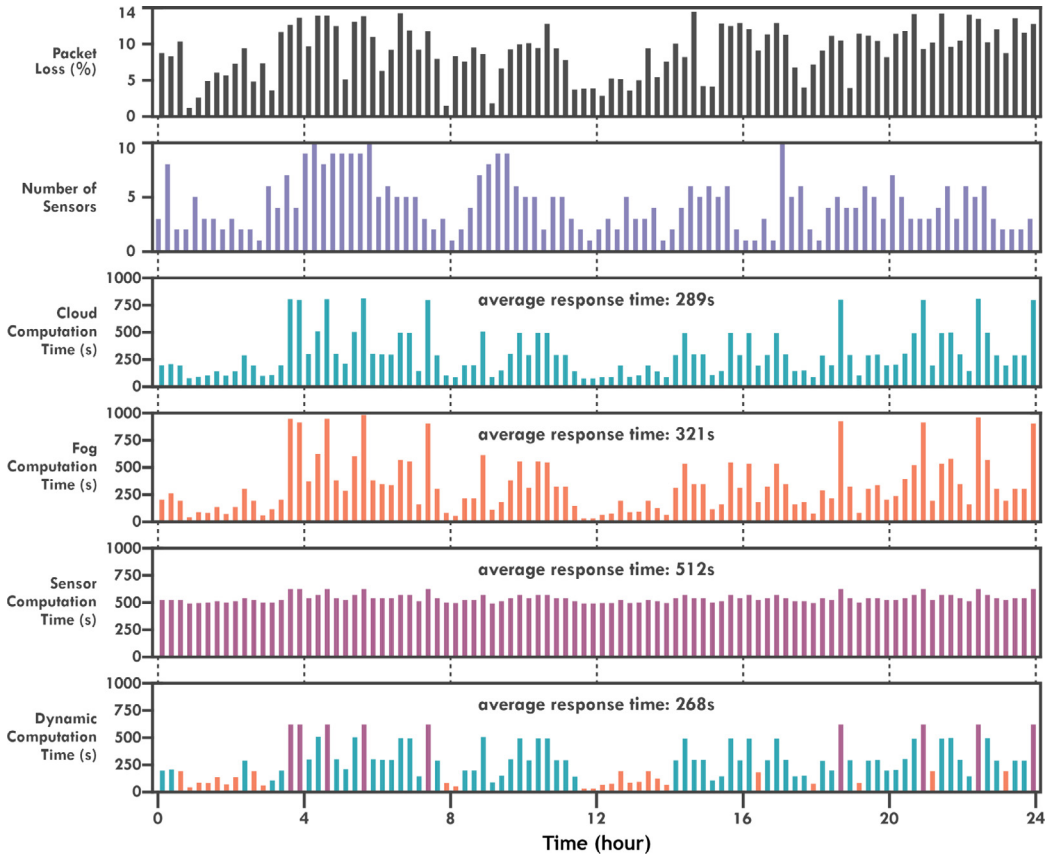

Fig. 4. Dynamic system behavior over a scenario.

formulate a task assignment in multiple resource constrained mobile devices and provide a fully polynomial time approximation algorithm to find the optimal offloading scheme that balances latency and energy consumption of mobile devices. An efficient one-dimensional search algorithm is proposed [25] to find optimal task scheduling policy for MEC by formulating a power-constrained delay minimization problem.

Skarlat et al. [7] model the service placement problem for IoT applications over fog resources as an optimization problem and propose a genetic algorithm to solve the problem. Latency-minimization problem in a multi-user time-division multiple access for mobile-edge computation offloading is investigated by [8]. The work formulates a convex optimization problem for partial compression problem and develops an optimal joint communication and computation resource allocation algorithm to solve it. Kouloumpris et al. [40,41] uses a mathematical programming based framework to optimally allocate tasks while satisfying operational constraints.

Kwak et al. [9] jointly optimize dynamic cloud offloading policy and CPU scaling in dynamic mobile network environment. The work uses the Lyapunov optimization technique to minimize energy consumption for given delay constraints. Partial computation offloading modeling in MEC is presented to minimize energy consumption and latency by jointly optimizing the computational speed (DVFS) of mobile devices and communication power consumption and offloading ratio [33]. The work formulates the problem and offers the globally optimal solutions in closed-form which shows that total offloading could not be optimal when the smart mobile device has the capability of computation scaling. A message-passing framework is proposed to reduce complexity of partial computation offloading by formulating the problem as a mixed integer program [10]. An online algorithm for partial computation offloading is proposed based on Lyapunov optimization to minimize energy and latency. The agent decides the CPU-cycle frequencies for local execution and the bandwidth allocation and transmit power for computation offloading [17].

Chen et al. [11] aim to find the overall optimal decision on partial computation offloading in a general mobile cloud computing system to minimize a weighted total cost of energy, computation, and delay of all users. The work formulates the problem as a non-convex quadratically constrained quadratic program and proposes an efficient algorithm to solve the problem. Nan et al. [18] present an adaptive online decision-making algorithm which uses the Lyapunov optimization technique to distribute the incoming data to the corresponding tiers to minimize response time and application loss number. Liu et al. [12] formulate a multi objective optimization problem to minimize the energy consumption and delay performance in mobile devices. The work optimizes the offloading probability and transmission power by using an Interior Point Method based algorithm.

Zhao et al. [14] provide an offloading algorithm to minimize energy consumption with consideration of latency tolerance and the transmission power of mobile devices. The work solves the optimization problem using Non-linear fractional programming dynamically and makes a decision to offload computing by adjusting transmission power. Chamola et al. [19] propose an optimal task assignment in a network of connected cloudlets which provide services to mobile devices to minimize the latency. Chang et al. [20] investigate the problem of energy optimization for a fog computing system. The work proposes a scheme that optimizes both offloading probability and transmit power to minimize the energy consumption by using nonlinear optimization and ADMM-based method. The majority of existing solutions only focus on communication–computation optimization considering limited system parameters such as processing capacity and network bandwidth. In contrast, we highlight the impact

of variety of parameters (e.g., application characteristics, system complexity, communication cost, and dataflow configuration) on IoT-based systems (see Fig. 2). Furthermore, finding optimal computation offloading policy for dynamic systems is critical since dynamicity of environment changes over time. In this paper, we examine a proof-of-concept dynamic computation offloading strategy to show IoT-based systems are required to be dynamically controlled at run-time.

7. Conclusions

This paper investigated various co-design aspects of computation offloading for IoT systems with typical three-layer architecture. We explored the impact of dataflows, communication, system complexity, and application characteristics on the response time in IoT systems. Then, we examined an IoT-based system enabled by an end-to-end dynamic computation offloading. The system minimized the response time of the application by (1) observing the system condition, (2) performing computation migration decision-making, and (3) carrying out the computation in the sensor, fog, or cloud layer. Experiments showed the optimal solution varies based on the network latency, dataflow configuration, application characteristic, and the number of connected sensors to the fog layer. We showed it in a scenario with run-time variations in system complexity and packet loss ratio. The dynamic computation offloading system reduced the average response time with %8, %17, %48 in comparison with only the cloud or fog or sensor computing scheme respectively.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This material is based upon work supported partially by the US National Science Foundation (NSF) WiFi US grant CNS-1702950 and Academy of Finland grants 311764 and 311304 and MSIT(Ministry of Science, ICT), Korea, under the High-Potential Individuals Global Training Program (2019-0-01607) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation).

References

[1] L. Atzori, et al., The internet of things: A survey, Comput. Netw. 54 (15) (2010).
[2] F. Firouzi, et al., Internet-of-things and big data for smarter healthcare: From device to architecture, applications and analytics, Future Gener. Comput. Syst. (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] Sina Shahhosseini, Iman Azimi, Arman Anzanpour, Axel Jantsch, Pasi Liljeberg, Nikil Dutt, Amir M Rahmani, Dynamic computation migration at the edge: Is there an optimal choice?, in: Proceedings of the 2019 on Great Lakes Symposium on VLSI, ACM, 2019, pp. 519–524.
[6] M.R. Nakhkash, et al., Analysis of performance and energy consumption of wearable devices and mobile gateways in IoT applications, in: COINS, 2019.
[7] Olena Skarlat, Matteo Nardelli, Stefan Schulte, Michael Borkowski, Philipp Leitner, Optimized IoT service placement in the fog, Serv. Orient. Comput. Appl. 11 (4) (2017) 427–443.
[8] Jinke Ren, Guanding Yu, Yunlong Cai, Yinghui He, Latency optimization for resource allocation in mobile-edge computation offloading, IEEE Trans. Wireless Commun. 17 (8) (2018) 5506–5519.
[9] Jeongho Kwak, Yeongjin Kim, Joohyun Lee, Song Chong, DREAM: Dynamic resource and task allocation for energy minimization in mobile cloud systems, IEEE J. Sel. Areas Commun. 33 (12) (2015) 2510–2523.
[10] Shahrouz Khalili, Osvaldo Simeone, Inter-layer per-mobile optimization of cloud mobile computing: A message-passing approach, Trans. Emerg. Telecommun. Technol. 27 (6) (2016) 814–827.
[11] Meng-Hsi Chen, Ben Liang, Min Dong, Joint offloading and resource allocation for computation and communication in mobile cloud with computing access point, in: IEEE INFOCOM 2017-IEEE Conference on Computer Communications, IEEE, 2017, pp. 1–9.
[12] Liqing Liu, Zheng Chang, Xijuan Guo, Shiwen Mao, Tapani Ristaniemi, Multiobjective optimization for computation offloading in fog computing, IEEE Internet Things J. 5 (1) (2017) 283–294.
[13] Liqing Liu, Zheng Chang, Xijuan Guo, Socially aware dynamic computation offloading scheme for fog computing system with energy harvesting devices, IEEE Internet Things J. 5 (3) (2018) 1869–1879.
[14] Xiaohui Zhao, Liqiang Zhao, Kai Liang, An energy consumption oriented offloading algorithm for fog computing, in: International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness, Springer, 2016, pp. 293–301.
[15] Ke Zhang, Yuming Mao, Supeng Leng, Quanxin Zhao, Longjiang Li, Xin Peng, Li Pan, Sabita Maharjan, Yan Zhang, Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks, IEEE Access 4 (2016) 5896–5907.
[16] Xiaowen Cao, Feng Wang, Jie Xu, Rui Zhang, Shuguang Cui, Joint computation and communication cooperation for mobile edge computing, in: 2018 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), IEEE, 2018, pp. 1–6.
[17] Yuyi Mao, Jun Zhang, S.H. Song, Khaled Ben Letaief, Power-delay tradeoff in multi-user mobile-edge computing systems, in: 2016 IEEE Global Communications Conference, GLOBECOM, IEEE, 2016, pp. 1–6.
[18] Yucen Nan, Wei Li, Wei Bao, Flavia C Delicato, Paulo F Pires, Albert Y Zomaya, A dynamic tradeoff data processing framework for delay-sensitive applications in cloud of things systems, J. Parallel Distrib. Comput. 112 (2018) 53–66.
[19] Vinay Chamola, Chen-Khong Tham, G.S.S. Chalapathi, Latency aware mobile task assignment and load balancing for edge cloudlets, in: 2017 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops, IEEE, 2017, pp. 587–592.
[20] Zheng Chang, Zhenyu Zhou, Tapani Ristaniemi, Zhisheng Niu, Energy efficient optimization for computation offloading in fog computing system, in: GLOBECOM 2017-2017 IEEE Global Communications Conference, IEEE, 2017, pp. 1–6.
[21] Wei Bao, Wei Li, Flavia C. Delicato, Paulo F. Pires, Dong Yuan, Bing Bing Zhou, Albert Y. Zomaya, Cost-effective processing in fog-integrated internet of things ecosystems, in: Proceedings of the 20th ACM International Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems, 2017, pp. 99–108.
[22] Ajay Kattepur, Harshit Dohare, Visali Mushunuri, Hemant Kumar Rath, Anantha Simha, Resource constrained offloading in fog computing, in: Proceedings of the 1st Workshop on Middleware for Edge Clouds & Cloudlets, 2016, pp. 1–6.
[23] Changsheng You, Kaibin Huang, Exploiting non-causal CPU-state information for energy-efficient mobile cooperative computing, IEEE Trans. Wireless Commun. 17 (6) (2018) 4104–4117.
[24] Zhengguo Sheng, Chinmaya Mahapatra, Victor C.M. Leung, Min Chen, Pratap Kumar Sahu, Energy efficient cooperative computing in mobile wireless sensor networks, IEEE Trans. Cloud Comput. 6 (1) (2015) 114–126.
[25] Juan Liu, Yuyi Mao, Jun Zhang, Khaled B. Letaief, Delay-optimal computation task scheduling for mobile-edge computing systems, in: 2016 IEEE International Symposium on Information Theory, ISIT, IEEE, 2016, pp. 1451–1455.
[26] R. Roman, et al., A survey and analysis of security threats and challenges, Future Gener. Comput. Syst. 78 (2018).
[27] D. Amiri, et al., Edge-assisted sensor control in healthcare IoT, in: IEEE GLOBECOM, 2018.
[28] I. Azimi, et al., HiCH: Hierarchical fog-assisted computing architecture for healthcare IoT, ACM Trans. Embed. Comput. Syst. 16 (5) (2017).
[29] I. Azimi, et al., Empowering healthcare IoT systems with hierarchical edge-based deep learning, in: IEEE/ACM CHASE, 2019.
[30] Blesson Varghese, Rajkumar Buyya, Next generation cloud computing: New trends and research directions, Future Gener. Comput. Syst. 79 (2018) 849–861.
[31] Li Lin, Xiaofei Liao, Hai Jin, Peng Li, Computation offloading toward edge computing, Proc. IEEE 107 (8) (2019) 1584–1607.
[32] Jie Xu, Shaolei Ren, Online learning for offloading and autoscaling in renewable-powered mobile edge computing, in: 2016 IEEE Global Communications Conference, GLOBECOM, IEEE, 2016, pp. 1–6.
[33] Yanting Wang, Min Sheng, Xijun Wang, Liang Wang, Jiandong Li, Mobile-edge computing: Partial computation offloading using dynamic voltage scaling, IEEE Trans. Commun. 64 (10) (2016) 4268–4282.

[34] Neal Cardwell, Stefan Savage, Thomas Anderson, Modeling TCP latency, in: Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064), Vol. 3, IEEE, 2000, pp. 1742–1751.
[35] Kasem Khalil, Khalid Elgazzar, Magdy Bayoumi, A comparative analysis on resource discovery protocols for the internet of things, in: 2018 IEEE Global Communications Conference, GLOBECOM, IEEE, 2018, pp. 1–7.
[36] 2019 Speedtest U.S. mobile performance report by Ookla,
[37] Iman Azimi, Arman Anzanpour, Amir M. Rahmani, Tapio Pahikkala, Marco Levorato, Pasi Liljeberg, Nikil Dutt, Hich: Hierarchical fog-assisted computing architecture for healthcare iot, ACM Trans. Embed. Comput. Syst. 16 (5s) (2017) 174.
[38] H. Hoffmann, et al., SEEC: A Framework for Self-Aware Computing, Technical report, MIT, 2010.
[39] Yi-Hsuan Kao, Bhaskar Krishnamachari, Moo-Ryong Ra, Fan Bai, Hermes: Latency optimal task assignment for resource-constrained mobile computing, IEEE Trans. Mob. Comput. 16 (11) (2017) 3056–3069.
[40] Andreas Kouloumpris, Theocharis Theocharides, Maria K. Michael, Metis: Optimal task allocation framework for the edge/hub/cloud paradigm, in: Proceedings of the International Conference on Omni-Layer Intelligent Systems, 2019, pp. 128–133.
[41] Andreas Kouloumpris, Maria K Michael, Theocharis Theocharides, Reliability-aware task allocation latency optimization in edge computing, in: 2019 IEEE 25th International Symposium on on-Line Testing and Robust System Design, IOLTS, IEEE, 2019, pp. 200–203.