A Unified Architecture for Integrating Energy Harvesting IoT devices with the Mobile Edge Cloud

V. Balasubramanian*, N. Kouvelas*, K. Chandra*, R.V. Prasad*, A.G. Voyiatzis†, W. Liu ‡
*Embedded Software, EEMCS, TU Delft, Delft, the Netherlands
†SBA Research, Vienna, Austria
‡Auckland University of Technology, New Zealand
(v.balasubramanian, n.kouvelas, k.chandra, r.r.venkateshaprasad)@tudelft.nl,
avoyiatzis@sba-research.org, william.liu@aut.ac.nz

***Abstract*—Recently, the edge resource provisioning schemes were defined considering the low-latency Mobile Edge Computing (MEC) paradigm. Most of these models only consider battery-powered devices like smart-phones, thus are agnostic to the energy harvesting techniques that achieves a green MEC system. Further, most of the studies on MEC assume unlimited edge resources which is not the case as it is with the conventional data-centers (public clouds). Hence, unrestricted use of edge resources is not ideal. This work mainly considers two problems: (1) the offloading of data traffic from the Internet of Things (IoT) devices that rely on energy harvesting to the MEC entities and (2) assignment of the resources at the MEC. The novelty of this paper lies in the energy scavenging based architecture that is developed over the Contiki OS. Secondly, saving the energy for computations to maximize the lifetime of the sensing nodes by performing the execution of the computationally-intensive tasks at the edge which is a single hop away. The proposed architecture uses the ambient triggers to form the sensor network and establish links with computationally capable resources located at the edge. Further, a mathematical model to manage the resources at the edge is proposed. Finally, we evaluate a threshold-policy for optimizing the resources participating in an edge computation service for an IoT scenario and discuss the improvements achieved.**

***Index Terms*—Mobile Edge Computing, Integer Linear Programming, Internet of Things, Sensor Network.**

I. Introduction

The popularity of Mobile Cloud Computing (MCC) has rapidly increased due to the exponential growth of mobile devices and applications. However, in many scenarios, data needs to be processed and decisions must be made in real time. The established MCC paradigm is not able to meet the latency requirements [1]. For example, it takes about 30-100 ms round trip time (RTT) for an online gaming traffic with high resource demands [1]. To overcome this, the concept of *Mobile Edge Computing* – where the computing elements are placed closer to the end user devices – is proposed [2]. Figure 1 shows the edge ecosystem with the computational elements collocated with the mobile edge network. Owing to the benefits of providing a faster service by a resource entity closer to the mobile devices, edge computing and its varieties like Mobile Edge Computing (MEC) have received much attention from academic and industrial research communities. In this work we focus on MEC.

978-1-4673-9944-9/18/$31.00© 2018 IEEE

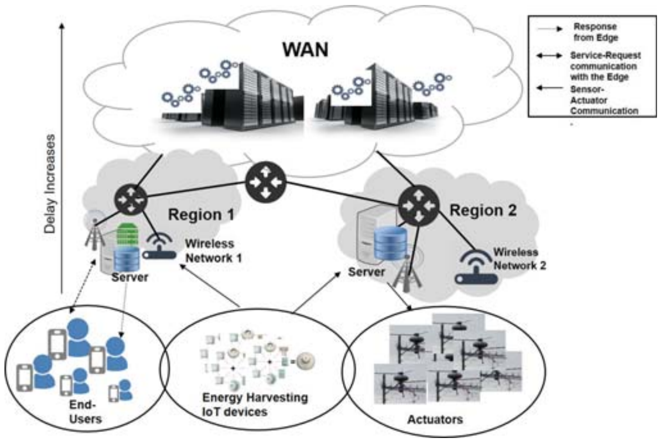

Fig. 1. The Edge Ecosystem

According to [3], *"MEC is specified to provide a multi-tenant hosting environment for 4G RAN edge applications, where MEC is collocated with the aggregated eNodeBs"*. Various implementations that tap into this association of MEC servers with the Radio Access Networks for enabling a proactive computation are described in [2]. Most of the current research considers the presence of an all-powerful *"cloud-like"* entity at the edge of the network (like Cloudlets [4]). Contrarily, an MEC server cannot be as resourceful as the conventional cloud due to obvious reasons of limited capacity. Most of the prior research, like [5] and [6] investigate the execution cost at the edge cloud or assume a single edge cloud usage without much deliberation on the life of the devices during and after the computation. Further, as the edge services are becoming increasingly important for the IoT ecosystems, we have to also consider the limiting constraints of these low-cost sensor devices. Apart from the limited storage and computation capabilities, the most important constraint of low-cost sensor nodes is the limited battery capacity. To overcome this, energy harvesting sensor nodes are widely being used for IoT applications such that a perpetual network operation can be ensured. However, the stochastic nature of harvested energy results in unreliable operation of EH-IoT nodes. Thus, it is highly important that sensor data gathered at EH-IoT nodes is offloaded to the MEC before the nodes die. In this paper, we, focus on two challenging aspects considering EH-IoTs and MEC as discussed via the use-cases below.

A. Problems

1) *Computation Offloading and Communication*: In typical IoT applications, like triggering context events in critical areas [7], the precise context determination needs large amount of computational resources to process the data gathered from large scale deployment of sensors. The individual sensor nodes are not suitable for such computation as only partial information is available at each node. The increased use of energy harvesting (EH) sensor nodes further aggravate this problem as the available node energy is highly fluctuating due to the stochastic nature of energy harvesting process. In case a sensor fails (due to lack of sufficient energy), there is no actuation event that can be expected. The computation capabilities offered by MEC can be of great use to EH-IoTs as the computation can be offloaded to MECs. Thus, end-to-end frameworks are needed that integrate EH-IoTs into MEC environment. In this paper we propose a framework for the EH-IoT nodes to offload computations to edge nodes.
2) *MEC Resource Utilization*: Although MEC can provide the needed computation resources for IoT applications, it is important to optimize the MEC resources as: (i) unlike traditional cloud data centers, MEC servers cannot have unlimited resources as they are located at the mobile edge where over-provisioning can only be facilitated to a certain extent; and (ii) increased deployment of EH-IoTs choosing MEC will eventually deplete the MEC resources. Let us consider an EH-IoT application, that's executed at the edge. Let's say a sequentially executed scenario with an application divided into 5 tasks is available. It should be noted that there is a dependence of tasks on the CPU cycles [8]. Assuming each task finds an edge node for execution, there will be total 5 edge nodes serving 5 tasks considering the state-of-the art resource allocation mechanism [5]. On the other hand, we propose a selective resource assignment scheme where edge nodes are selected based on the task dependencies on the CPU cycles. For example, consider different CPU cycles required for execution of two of these tasks, such that only 3 edge nodes are required for the computation of the complete set of tasks. In this case, we save 2 edge nodes for another service. By doing this, we not only reduce computation units but also save energy consumption for computation.

B. Contributions

- In order to address Problem 1, we propose a preliminary architecture for unifying the EH IoT devices with the mobile edge service nodes. We essentially look at real-time actuation scenarios where the software component trigger sensor network formation and acts in a way that the latency critical traffic (for real-time actuation) is routed towards the edge resources.
- As defined in Problem 2, the edge service node capacities are limited and this limitation can be addressed by following an optimal resource provisioning scheme. Thus, we mathematically model a threshold policy based service provisioning scheme that considers the energy, the resource capacities (CPU, RAM and Storage) and the stability of the edge nodes while deploying a service. We believe these three factors impact the profit of edge cloud service providers.
- Finally, we develop a heuristic algorithm to check the usefulness of the last-mile computation and share the insights we gathered by evaluating the threshold-policy.

The rest of this paper is structured to delineate the novelty of the framework with a brief discussion about the state of the art models in the related work in Section II, followed by Section III, which discusses the system overview. In Section IV, the system architecture is elaborated, and in Section V, we evaluate the performance of our algorithm. Section VI concludes this work and proposes potential future extensions.

II. Related Work

Zhu et al. address the problem of sensor network integration in Mobile Cloud Computing [9]. The authors define an algorithm for intermittent data-sharing and connectivity to cloud. However, this model ignores the possibility of facilitating edge computing services. Sun and Ansari [10] consider an SDN-assisted mobile edge computing framework called Edge-IoT that allows the fog nodes to utilize the cloud or the edge computing services depending on the application requirements. Further, the authors outline a hierarchical fog computing architecture and describe a method using VM service migration to achieve a flexible application service deployment. However, this work does not consider the challenges of an energy-aware ecosystem.

Efforts in the past such as [5], [6] have looked at computation offloading in an MEC system while making decisions based on CPU cycle, transmit power etc. However, the former considers mobile device equipped with an energy harvesting component with no real impact on actuation. The latter focuses only on resource allocation of a single edge cloud. On the other hand we consider optimizing the edge resource usage based on a set requirement strategy coupled with the knowledge of the energy signatures of the end-users and the actuator candidates in real-time scenarios.

In [11], the authors propose a model for providing edge services to latency-critical applications in an IoT environment. The paper primarily proposes providing service nodes at the network edges where the users directly connect. As this follows a split cloud architecture by interconnecting edge networks with data-center networks, it is called Edge Cloud. Authors monitor traffic and based on the latency requirements offload data to the cloud. This work ignores the challenges put forth by Mobile Edge Computing in the IoT domain. Therefore, by considering the relationship with the EH aware

IoT devices and the Mobile Edge Cloud Computation we propose a unified architecture and move towards a "Green-IoT" ecosystem.

III. System Overview

In this section the foundations, concepts and the integral components on which the architecture is built is discussed.

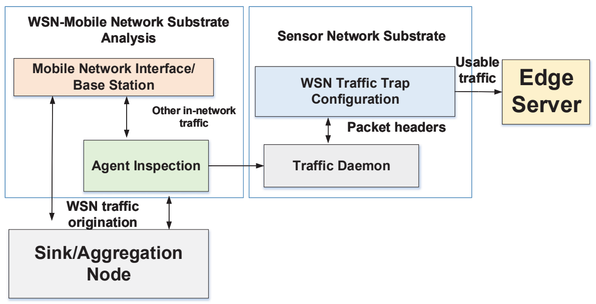

Fig. 2. Component Interaction

A. Foundations

Protocols like Low-power Wireless Bus [12] and Choco [13] address the problem of routing data on sensor nodes that have limited energy in a highly energy-efficient manner and with very low latencies. We consider one such protocol can be used to select a sink dynamically. However, these protocols assume the computation to be already taken care of at the application layer on the sensor nodes. Additionally, these protocols are agnostic to the edge entities in the use cases mentioned in [7], such as computing the context of a user considering several ambient sensing technologies. The sensors may be using different radio technologies as well. In such use cases, a quick context computation is required despite a large scale deployment of sensor nodes. Furthermore, computing the context considering several sensor data make a perfect example use case for using mobile edge cloud. In such cases, communication and computation should be jointly considered as both energy and latency are of prime importance. This makes resource allocation in such environments challenging.

In the use case of computing context of the user, we mentioned that the sensors may use different radio technologies. Framework such as EdgeIoT [10] propose to empower the base-stations with all these technologies, which can be easily integrated. However, one aspect that is not yet considered is that the eNodeBs and the mobile edge cloud (MEC) will be serving not only IoT but many other types of applications, hence traffic, such as calls, video on-demand, and online gaming. Therefore, it is required to consider a mobile network where multiple classes of traffic flow, and hence prioritization by QoS Class Identifiers (QCI) becomes essential. In mobile-networks trying to disentangle one form of traffic from another can intuitively provide positive results, a similar performance gain can be achieved if the computational entity is aware of the incoming requests. Motivated by this, we build the $2EA$ framework to make the edge entities aware of the IoT traffic and a traffic daemon that reduces response time by avoiding needless traversal of packets through the core network.

B. Concepts

To design the $2EA$ framework, we continue the example of computing the context of a user. We use the architecture from [7] as a source of inspiration. In this work, the authors divide the architecture into several layers including ambient context providers, context interpreters and reasoners, above which is the application layer. The computed context and actuation information must then be disseminated to the actuators and output devices.

In order to realize such a system that contains a large number of context providers (namely, wireless sensor networks, LoRaWAN/NB-IoT based sensor devices, social networks, etc.), it is impossible to compute context on any one sensor node. To this end, the sensors must 'offload' their data to their nearest MEC. Naturally, we consider the context interpreter (not needed for energy-harvesting context-event triggered systems), and the context reasoner to be in the MEC.

A high-level view of the processes involved in obtaining the information and the dependencies that need to be satisfied is illustrated here. Context-event triggers, which can be either based on event-based reporting or energy-harvesting triggered [7], initiate the discovery module on the sensor OS (for IEEE 802.15.4 and IEEE 802.15.1 devices) that finds other sensors in the vicinity and forms a sensor network. Other cellular IoT technologies will connect directly. We focus on the WSNs here. Traditionally, in a sensor network, a sink acts as a data aggregation point and other nodes report periodically to the sink. **Step 1** is the selection of the sink after which other nodes submit the details along with their node ids. **Step 2** is the energy-aware routing and management with the energy profiles of the sink. Concurrently, the energy database is updated. **Step 3** involves the topology controller (Manager Module) managing the nodes in the sensor network. Once a data packet is received the global buffer in the Contiki core sends a reminder to the TCP/IP stack which in turn notifies the software layer for scheduling the traffic. **Step 4** involves the aggregated data being sent to the computational entity with an impregnated agent that makes the system edge aware on one side and energy-aware on the other end. Once a sink is selected, the sink node becomes the gateway for the WSN traffic to send the data to the mobile network. Typically, in an energy harvesting sensor network, the ambient energy is used as an input for context determination. Similarly, in our architecture, the harvested energy triggers the sensor network formation module after which the sink is chosen.

Figure 2 shows the component interactions that occur during this process. The **Agent Inspection** module determines the sensor traffic (from both WSNs and other sensors) that goes to a sensor network substrate after it separates the in-network traffic based on the defined packet header classes. Our approach resembles in part to [14] where the embedded agent usage has been defined for virtual sensor networks (VSN).

In **Sensor Network Substrate**, packet headers are sifted and WSN traffic that is trapped is forwarded to the compute location. This will be elaborated further in the following section.

IV. System Architecture

In this section the details of the system architecture are discussed. There are two integral parts to this system: (a) Context providers where the sensor data generation happens and (b) the MEC where the traffic and the resources must be managed as to provide the required computational services as required.

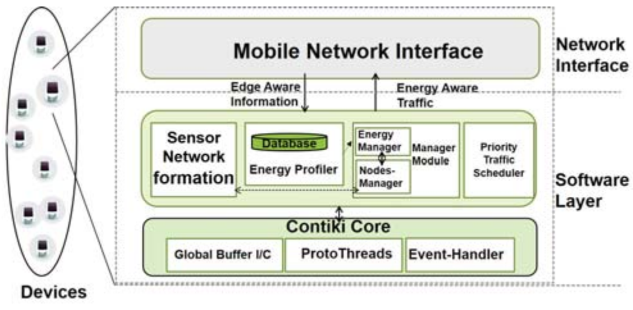

Fig. 3. 2(EA) Context Provider Architecture

A. Context Providers

We specifically focus on the energy-harvesting context-event triggered systems. Figure 3 shows the system architecture for the context providers. The architecture consists of the following:

1) Device Layer- This layer has the context providers which are essentially sensors that sense and collect the raw data from the environment. These devices rely on energy harvesting techniques which act as input trigger for software components.
2) Software Layer- This is the operating system in the sensor over which the architecture components are built. We build our components on top of the Contiki operating system [15]. These components are:
   Sensor Network formation module: Once a context-event has been triggered, the network formation module is initiated. After this, nearby sensors are discovered and the most capable sensor is selected as the sink. The capability is dependent on the power usage effectiveness (PUE) of the sensor. PUE for a sensor is the value of how much energy is used by the computing element in the sensor. These values are determined primarily to decide the aggregation node (sink) that establishes edge to sensing layer communication.
   Energy Profiler: Energy profiler maintains the database with power metrics, constants and information associated with the WSN formed and the sink selected. Even among energy harvesting nodes the energy profiles of different nodes in a region will be different based on its location. This profiler uses a couple of these relevant information (sensor type, sensor port numbers, IDs, operational report of on/off) which is then given to the Management module.
   Management Module: The Management module comprises of the **Node Manager** and the **Energy Manager**. It correlates information obtained from the first two modules, remembers the sink as the head and finds the power consumed by the sink. A threshold is set below which the sink status is demoted and another sink is chosen based on the suggestions given by the management module. The cases of constructive interference are beyond the scope of this paper and will be considered in the future. Energy profiles are gathered (for one hop neighbours) by the energy manager and stored in the database. Node Manager checks unused sensors and switches to Sleep mode. The Node Management threads are also running on the VMs such that the actuator nodes are remotely controlled by the MEC.
   Priority Traffic Scheduler: The prioritization algorithms are run and a schedule is determined. For instance, consider an MEC server as the edge service providing node, a part of the Management algorithm is run to manage the two ends of the system. The collocation of the MEC with the Mobile Network infrastructure ensures the management module to figure out the topology of the actuators and make appropriate decisions for the actuator. Rightly so, the schedules are prepared with the link usages from the memory of the MEC. Hereby, a large part of the computation tasks are taken off from the sensors. Further, if certain links fail a back-up path is computed towards the actuators to realize a task at the appropriate time. All of these tasks are determined by the network substrate that remotely gathers information from the WSN infrastructure as shown in Figure 4. It should be noted that the threads running on the energy harvesters are the ones that need sensing-event triggers and acknowledgement of the edge aware traffic.

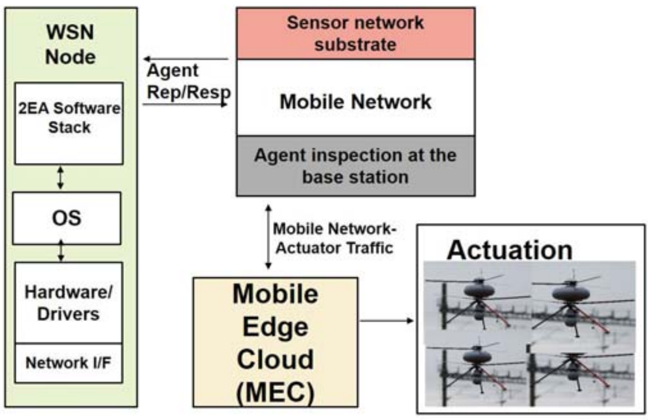

Fig. 4. 2EA Network Management Architecture

B. *Traffic Management*

Network Interface Layer - This layer consists of the Wi-Fi or 3G/4G/5G interface modules that are used to communicate with the eNodeBs (or Access Points). When the sensors are deployed around a cell site, the mobile network has a sensor-network composition substrate module close at the eNodeBs which inspects the incoming traffic. To address this, an agent code inspects the multi-network traffic for usable sensor traffic and forwards it to the Traffic Daemon. At the WSN Traffic Trap Configuration module the headers are modified and the traffic is moved to the MEC where the processing takes place. As the actuating entities are already subscribed to the compute services, the results are directly sent to those without any delay.

C. *Edge Resource Utilization*

In this section we define threshold L based on which the edge resources are labelled and made available to the end user. The parameters on which L is dependent are normalized and used for quantifying the edge nodes at the time of service provisioning. Let T_r^k = Total resource (CPU, RAM, storage) and Q_r^k = Resource Required (CPU, RAM, storage). Here 'k' represents the k^{th} node. Let Ω_r^k = Energy consumption of the k^{th} edge node. Post execution of the task the new energy value is defined as Ω_o^k. Let Node Stability of the k^{th} node is defined by S^k which is the ratio of the number of requests that were served by the node in the past to the total requests received by the node. To this end we define the parameter $L(k)$ given by,

$$L(k) = \alpha\{(T_r^k, Q_r^k), \Omega_r^k, S^k\}. \qquad (1)$$

Here α is the weight given to a particular form of resource at the time of subscription. As the value of L for a node increases, the execution time decreases. As the number of these nodes $\{1, 2, 3..n\}$ goes on increasing, the choice of nodes has to be made based on proper evaluation of tasks (j from $\{j_1, j_2, ...j_n\}$) to residual resources mapping. The resources are modelled based on modified QoS class definitions in [16]. Example, when a request comes to the service provider the QoS requirement for resources belonging to class r with r_1 for CPU, r_2 for RAM, r_3 for storage must be distinctly defined. Total resource provided by that class is r_t, with r_{t_1}, r_{t_2}, r_{t_3}. Now for an $L - value$ we have $\{(r/r_t), \Omega_{r_n}, S_i\}$. We restrict the range between $(1, 4) \leftrightarrow L(k)$ for the experiments. However, based on the service providers needs the range may change. The objective is subject to the following constraint:

$$y_j^k = \begin{cases} 1, & \text{if } \; node\ k\ is\ used\ for\ execution\ of\ task\ j \\ 0, & \text{otherwise} \end{cases} \qquad (2)$$

For reducing the number of edge resources used

$$min \sum_{k \in n} \sum_{j \in J} y_j^n \frac{\psi_j^k}{L(k)} \qquad (3)$$

ψ_j^k represents the execution time for a task j being executed on node k. Similarly, L represents the threshold value for the node k. As the value of L begins to increase, the overall execution time reduces. We need to figure out the dependence of CPU utilization on computing energy at the edge node. It is clear that lower the CPU utilization, lower would be the energy consumption. This is because the computation energy is directly proportional to the CPU utilization [5]. The challenge here is to not only reduce the number of compute cores inside one edge node but also to find out if the task acceptance and reduction in the number of edge nodes can have a positive impact on the system when viewed holistically.

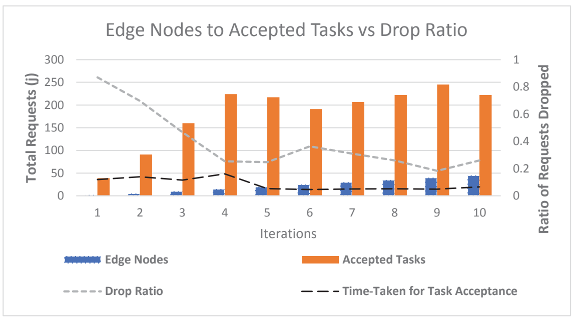

Fig. 5. Edge Service Provisioning and Task Execution based on Resource Labels

V. Performance Analysis

We have used a core 2 Duo CPU, with a 4GB RAM for this experiment. We ignore CPU-induced delays, as the objective is to find the conceptual feasibility of this model, using synthetic workloads. As, the resources at the edge are limited compared to the cloud, an optimal edge resource utilization is required. Furthermore, as our focus is to evaluate the compute-resource utilization parameters through this simulation, the interplay of EH-devices with the Mobile Edge Cloud would be evaluated as our future work. We use the Gurobi optimizer [17] to solve this problem. As this problem formulation is NP-hard, we use a greedy bin-packing algorithm to evaluate the following metrics: (a) Drop Ratio - ratio between the number of total dropped tasks to accepted tasks. (b) Total Tasks Accepted - total scheduled-and-accepted tasks. (c) Utilization Efficiency of the edge resources - $(Q_r/\ T_r)$ (d) Time taken for task acceptance - Time taken for Edge resource to process a set of tasks.

A. *Discussion*

As it is observed, with any increase in the number of scheduled tasks, the performance of the resource selection improves. We synthetically introduce a set of randomly distributed task arrivals at the Edge. However, as the resources are gradually increased, more tasks are accepted, and the drop ratio decreases. Further, an irregularity was observed in the form of a sudden peak in Figure 5, that is mainly attributed to a drastic change

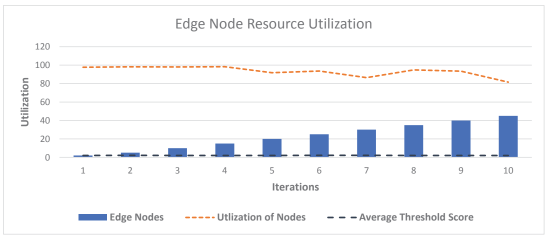

Fig. 6. Edge Resource Utilization

in data traffic, from 220 tasks to 180. Syntactically the bin model is as follows: *bins[nodes[k].nodeID] = Bin(nodes[k].**Lvalue**,int(nodes[k].cpu),int(nodes[k].storage), int(nodes[k].ram))* low =0.1 high =4.0 size =300. For every value in this range float("0:.1f".format(random.uniform(low,high))) is evaluated. In Figure 6, the average threshold score determines the edge resources that will be chosen by utilizing the limited amount of resources efficiently. The resources below the average threshold line are rejected. The execution time reduces as the number of resources increases. For a variety of tasks requiring a specific number of resources, some nodes are used irrespective of their requirements. The utilization remains close to 89-91%, with the initial peak close to approximately 98%, owing to low traffic rates. Over 300 tasks and around 100 nodes were synthetically modelled, but only a part of the results are shown for brevity. A major limitation of following a random task scheme is that a real world data-trace would probably behave in a very different manner. This would be considered in the future along-with the inclusion of the task deadlines.

VI. Conclusion & Future Work

In this work, we developed a preliminary Energy-Aware-Edge-Aware architecture ($2EA$) that considers sensors that rely on energy harvesting in IoT ecosystems. In this paper we capitalize on the limited edge resources by providing a threshold-based scheme that not only saves computational energy consumption but also ensures optimal usage of resources based on the task arrival process. Further, such a system enables an energy aware IoT ecosystem. At the device layer, where the context providers are placed, the computation capabilities are limited. Therefore, to offer computation service within one hop we consider the edge cloud. At the computation points, we have evaluated the resource utilization in the Edge by following an approach based on threshold policies. There were some important insights gathered through this work, for instance, the drop ratio goes on decreasing as the workload fetches for resources in an incremental manner. The edge resource utilization gives clarity about the importance of managing the resources at the resource-limited-edge. A detailed inspection of the random tasks arrival gave us some valuable insights on the anomalies of following a synthetic stochastic process using a greedy-algorithm. Next, we will focus on studying the effect of degradation in the link quality in critical scenarios that may lead to link failures.

Acknowledgement

This work has been undertaken as part of the SCOTT project. SCOTT has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No. 737422. This joint undertaking receives support from the European Union's Horizon 2020 research and innovation program.

We also thank Dr. ir. Vijay S. Rao for his useful comments during the preparation of this manuscript.

References

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A Survey on Mobile Edge Computing : The Communication Perspective," pp. 1–37.

[2] M. Patel, Y. Hu, P. Hédé, J. Joubert, C. Thornton, B. Naughton, J. R. Ramos, C. Chan, V. Young, S. J. Tan, D. Lynch, N. Sprecher, T. Musiol, C. Manzanares, U. Rauschenbach, S. Abeta, L. Chen, K. Shimizu, A. Neal, P. Cosimini, A. Pollard, and G. Klas, "Mobile-Edge Computing," *ETSI White Paper*, vol. 11, no. 1, pp. 1–36, 2014.

[3] G. Brown, "Mobile Edge Computing Use Cases & Deployment Options Prepared by," no. July, 2016.

[4] M. Satyanarayanan, P. Bahl, R. Carceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[5] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic Computation Offloading for Mobile-Edge Computing with Energy Harvesting Devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[6] C. You and K. Huang, "Multiuser Resource Allocation for Mobile-Edge Computation Offloading," pp. 1–31, 2016.

[7] V. S. Rao, S. N. Akshay Uttama Nambi, R. Venkatesha Prasad, and I. Niemegeers, "On systems generating context triggers through energy harvesting," *IEEE Communications Magazine*, vol. 52, no. 6, pp. 70–77, 2014.

[8] A. P. Miettinen, "Energy efficiency of mobile clients in cloud computing," *Energy*, p. 4, 2010.

[9] C. Zhu, V. C. M. Leung, L. T. Yang, and L. Shu, "Collaborative Location-Based Sleep Scheduling for Wireless Sensor Networks Integratedwith Mobile Cloud Computing," *IEEE Transactions on Computers*, vol. 64, no. 7, pp. 1844–1856, 2015.

[10] N. E. Sun, X., & Ansari, "Mobile edge computing for the internet of things." *IEEE Communications Magazine, 54(12)*, no. December, pp. 22–29, 2016.

[11] H. Chang, A. Hari, S. Mukherjee, and T. V. Lakshman, "Bringing the cloud to the edge," *Proceedings - IEEE INFOCOM*, pp. 346–351, 2014.

[12] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Low-power wireless bus," *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems - SenSys '12*, p. 1, 2012.

[13] Y. Katsumata, M. Suzuki, and H. Morikawa, "Demo abstract: Choco - A versatile communication protocol in Wireless Sensor Networks," *13th ACM Conference on Embedded Networked Sensor Systems, SenSys 2015*, pp. 475–476, 2015.

[14] R. Tynan, G. M. O'Hare, M. J. O'Grady, and C. Muldoon, "Virtual sensor networks: An embedded agent approach," *Proceedings of the 2008 International Symposium on Parallel and Distributed Processing with Applications, ISPA 2008*, pp. 926–932, 2008.

[15] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki – A Lightweight and Flexible Operating System for Tiny Networked Sensors," *LCN '04: Proceedings of the 29{th} Annual {IEEE} International Conference on Local Computer Networks*, pp. 455–462, 2004.

[16] A. Jarray, J. Salazar, A. Karmouch, J. Elias, and A. Mehaoua, "Qos-based cloud resources partitioning aware networked edge datacenters," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 313–320.

[17] I. Gurobi Optimization, "Gurobi optimizer reference manual," 2016. [Online]. Available: http://www.gurobi.com