IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 22, 2018, accepted October 23, 2018, date of publication November 9, 2018,
date of current version December 7, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2879811

An Efficient On-Demand Latency Guaranteed Interactive Model for Sensor-Cloud

NGOC-THANH DINH AND YOUNGHAN KIM, (Member, IEEE)
Department of Electronics and Telecommunication, Soongsil University, Seoul 06978, South Korea

Corresponding author: Younghan Kim (younghak@ssu.ac.kr)

This work was supported in part by the Korea Government through the Institute for Information & Communications Technology Promotion (IITP) Grant (Development of Content-Oriented Delay Tolerant Networking in Multi-Access Edge Computing Environment) under Grant 2017-0-00613 and in part by the Ministry of Science and ICT, South Korea, through the Information Technology Research Center Support Program, supervised by the IITP under Grant IITP-2018-2017-0-01633.

ABSTRACT Motivated by the Internet-of-Things (IoT) cloudification as the trend to implement IoT applications, an efficient design for interactions between sensors and cloud is necessary. In this paper, we propose an efficient interactive model that is designed for the sensor-cloud integration to enable the sensor-cloud to simultaneously provide sensing services on-demand to multiple applications with various latency requirements. In the proposed model, complicated functions are offloaded to the cloud, and only the light-weight processes are executed at resource constrained sensor nodes. We design an aggregation mechanism for the sensor-cloud to aggregate the application requests so that the workloads that are required for sensors are minimized, thereby saving energy. The latency of sensing packets from sensor-to-cloud is controlled by the sensor-cloud based feedback control theory. Based on the feedback from the sensor-cloud, physical sensor nodes optimize their scheduling accordingly to save energy while maintaining the latency of all sensing flows routed through them satisfied the requirements of applications. Extensive experimental and analysis results show that the proposed model effectively controls the latency of sensing flows with a low signaling overhead and a high energy efficiency compared with the state-of-the-art scheme.

INDEX TERMS Sensor cloud, IoT cloud, interactive model, latency guarantee, sensor cloud interactions.

I. INTRODUCTION

Recently, the sensor-cloud integration [2]–[11] has been receiving a great extent of interest among researchers and is being considered as a promising solution to solve existing limitations of traditional wireless sensor network (WSN) models such as the sensor system management and sensing data usage model. Sensor-cloud infrastructure constituting WSN and cloud is an expanded form of cloud computing for sensing services. By integrating WSNs with the cloud [11], [12], the sensor-cloud is able to provide sensing-as-a-service (SSaaS) for multiple applications simultaneously, unlike the traditional sensing model that is built for a dedicated application. In particular, the application model in the sensor-cloud architecture [7], [8], [10] is presented as follows:

- Physical sensor nodes collect sensing information from the real world and forward them to the sensor-cloud.
- The sensor-cloud virtualizes physical nodes into virtual sensors [13] which provide sensing-as-a-services for applications or users.
- Applications or users request the sensor-cloud for sensing services on demand.

One of the main objectives of the sensor-cloud is the enabling of a single WSN to produce sensing services for multiple applications at the same time while allowing applications or users to specify their requirements of sensing services on-demand based on their budget and expectation [14], for example, allowing applications to request sensing services with their packet latency requirement [7], [8], [10]. For many vertical Internet of Things (IoT) applications (i.e., mission-critical IoT or industrial monitoring), the latency guarantee is very important. Although the existing studies indicate that enabling on-demand latency SSaS is one of main features of sensor -cloud [7], [8], [10], [11], a study wherein an efficient interactive model to enable such a feature in the sensor-cloud has not yet been conducted.

This paper proposes an efficient interactive model for sensor-cloud integration that enables the sensor-cloud to serve periodic sensing services on-demand to multiple applications with different latency requirements. The model automatically adjusts the wakeup schedule of sensors to satisfy applications' latency requirements. In the proposed model, a request aggregator is designed for the sensor-cloud to aggregate requests of applications so that the required workloads

for physical sensors are minimized to save energy. The aggregator enables a sensor to run only a single schedule for serving multiple applications at the same time. A QoS controller is implemented on the sensor-cloud using the control theory [15], [17] to control the end-to-end (e2e) latency of sensing flows, gather feedback information from sensing flows, compute control parameters, and guide sensor nodes for adaptation. A lightweight scheduling controller implemented at local physical sensors is used to adapt their scheduling following guidance from the QoS controller. The scheduling adaptation have two objectives as follows. Firstly, the latency of data flows routed through a node is maintained within a pre-defined range that meets requirements of applications. Secondly, the energy efficiency of physical sensors is optimized.

In our design, complicated functions are offloaded to the cloud. With rich resources of the cloud, more complicated functions can be designed and executed for more complicated scenarios, which cannot run within the resource-constrained WSNs. Our proposed scheme can support dynamic latency requirements and multi-application scenarios while only light-weight processes are required at sensor nodes. The proposed scheme is compared to the state-of-the-art approaches, Delar [16] and DutyCon [15] in which each resource-constrained sensor is required to run multiple controllers as well as multiple schedules for cases of multi-flows, thereby limiting the scalability and increasing the complexity. In addition, each sensor node in DutyCon is required to collect feedbacks for scheduling adaptation, thereby incurring high energy consumption and overhead as shown in section IV. The interactive sensor-cloud model in this paper is proposed to address the limitations of DutyCon. Compared to the preliminary version [1], this paper provides more insight into the detailed design of the model with new mechanisms like scheduling mechanism and saving modes for the QoS controller. We also analyze behaviors and features of the scheduling controller and how it interacts with the QoS controller under various scenarios. We add stability analysis for the system and simulation-based validation. We conduct more experiments and present new significant results in comparison with the state-of-the-art scheme.

In summary, we make the following contributions in this paper.

- We propose and implement an efficient interactive model for the sensor-cloud that supports the provision of on-demand sensing services to multiple applications with different QoS requirements.
- We design an efficient aggregation mechanism to minimize the number of application requests that are sent to WSNs, and a distributed feedback control mechanism for interactions between the QoS controller on the sensor-cloud and the scheduling controller at the sensor nodes. The mechanisms guarantee that the latency requirements of the applications are satisfied while energy consumption of sensors is optimized. (section III)
- Through our comprehensive experimental and analysis results, we present that the proposed model effectively guarantees latency of sensing flows satisfied application requirements and is more energy efficient compared with the state-of-the-art scheme [15], [16]. (section IV)

II. RELATED WORK

Over the last few years, a number of researches have indicated the limitations of conventional WSN models [7], [8], [10], [11], [18], [19] such as the sensing management models, sensing pricing models, and the data usage models. Recently, the sensor-cloud model [2]–[8], [10] has been proposed as a potential approach and is attracting growing interest from both academia and industry. The sensor-cloud architecture is motivated by combining the ubiquitous data gathering features of conventional WSNs with the powerful storage, processing, and distributing of cloud computing for sensing data. The integration between WSNs and the cloud enables us to build sensor data collections as sensing-as-a-service where sensing data can be distributed and shared by multiple applications at the same time, which differs from the conventional sensing data usage model for a single dedicated application. As a result, the utility of WSNs could be greatly enhanced. The sensor cloud can also enhance the sensor resource utilization, sensor management, and play a role as a middle interface between physical WSNs and the cyber world.

Although several sensor-cloud architectures have been proposed [7], [8], [10], [14], [20]–[22], they share similar basic designs as follows. Physical WSNs are responsible for sensing data that are then collected by the sensor cloud. The sensor cloud is responsible for sensing data processing, storage, and providing sensing services to customers (users and applications). Physical sensors are normally virtualized into virtual sensors that are emulations of physical sensors on the sensor cloud. Based on virtual sensors, the sensor cloud can provide customized view sensing services to users/applications. In the sensor cloud architecture, sensing services are expected to be customizable and on-demand based on the users/applications's demands.

A number of initial studies have been done for detailed designs for the sensor cloud based on different approaches for different applications. In the previous works, we implement a sensor-cloud architecture for smart cities [23], for mobile computing applications [18], and distributed interactive digital signage systems [24]. Giancarlo *et al.* [25], [26] exploit the sensor cloud for body sensor network (BSN) for body monitoring. Fortino *et al.* [21] and Hassanalieragh *et al.* [27] raise opportunities and challenges of the sensor cloud, in other words, IoT cloud (i.e., Internet of Things' cloud) for e-healthcare. Misra *et al.* [20] integrate a sensor cloud for military services where virtual sensors are used to provide utility services for a battlefield scenario. Neto *et al.* [28] design a smart component for industrial sensor cloud which will act in the factory shop-floor, creating digital machines by means of sensing services. In another

work, Lyu *et al.* [29] investigate a scheduling problem of the sensor cloud for smart-living.

Several preliminary real implementations [24], [30]–[32] of the sensor cloud have been carried out for testbed. Most of the designs use NFV (networking function virtualization) [33], [34] for the implementation of cloud and virtual sensors due to their advantages. The detailed implementation of a sensor-cloud with NFV is presented in our previous work [24] in which the sensor-cloud is designed to manage sensors and sensing data used in our interactive digital signage system.

For many vertical IoT applications like industrial monitoring and mission-critical IoT, latency guarantee is very important. In addition, the main objectives of the sensor cloud are as follows 1) enabling a single WSN to provide sensing information for multiple applications 2) allowing users and applications to specifying their own QoS requirements on demand based on their budget and expectation [14] (i.e., applications can request sensing services with their own packet latency requirement [7], [8], [10]). Although existing studies show that enabling on-demand latency SSaS is one of the main features of the sensor cloud [7], [8], [10], [11], a study of an efficient interactive model between WSNs and the cloud to enable such a feature has not been conducted.

The conventional latency control and optimization mechanisms [15], [19], [35], [36] are not designed to support multiple applications and shared WSNs as well as on-demand requirements, so they may be not applicable to sensor-cloud. The detailed review of delay-aware mechanisms is presented in [35].

For the latency enhancement, a delay aware flooding algorithm (DEF) [37] is proposed as an effective flooding scheme which uses a delay-ware tree adaptive mechanism to enhance the packet latency. The algorithm is designed especially for broadcast traffic. DASF [38] is another scheduling and forwarding scheme designed for packet latency optimization. In the algorithm, the distribution of delay with given parameters and network model is estimated to enable each node to determine the maximum duty cycle interval. Packets are then transmitted to a selected node that satisfies the expected delay constrained success ratio. For the same objective, feedback control for adaptive duty cycle is used in [39] for adapting the sleep period based on traffic changes dynamically for reducing the packet latency. In [40], Least square error (LSE) rule is exploited with mean distortion level to enhance the delay and transmission power at each node.

In [41], an adaptive routing and scheduling mechanism for delay guarantee is proposed based on exploratory search. The main idea is that a sensor node searches all neighbor sensors to find nodes that can offer better delay and potentially shorten the end-to-end latency. The sensor node then changes to better routes. If there is no router available to satisfy requirements, the sensor node adjusts the scheduling. DGRAM [42] and TDGEE [43] are similar protocols which are based on coordinated sleep and time slot reuse to optimize latency of a node in accessing the channel. However, DGRAM is designed for WSNs with uniform node density only. In EDGE [44], delay minimization is achieved by exploiting path diversity of an opportunistic forwarding technique. In another approach, delay priority based scheduling policies are used in [45] to jointly optimize the end-to-end delay constraints and throughput requirements of a flow.

DutyCon [15] is a well-known dynamic duty cycle control scheme in conventional WSNs for end-to-end delay guaranteeing. DutyCon addresses limitations of the static uniform duty cycle approach [15] which is inefficient and works only for static traffic conditions with fixed network parameters. For applicability with resource-constrained nodes, the authors simplify the e2e packet latency guarantee problem by decomposing the e2e problem into a problem set of single-hop delay guarantee for each data flow. At each hop, a node collects feedback locally to adapt its duty cycle so that the single hop latency requirement is achieved. The advantages of DutyCon are as follows. Firstly, DutyCon supports dynamic traffic rates. Secondly, DutyCon supports multi-flow scenarios. Therefore, we select DutyCon as the state-of-the-art technique in this paper. However, DutyCon has the following disadvantages. Firstly, DutyCon does not support multi-application scenarios and dynamic latency requirement. Secondly, each sensor is required to run multiple schedules and multiple controllers in cases of multi-flows, thereby limiting the scalability and increasing the complexity. Thirdly, each node has to gather feedback information by itself for scheduling adaptation, thereby incurring high overhead. The interactive sensor cloud model in this paper is proposed to address the limitations of DutyCon.

III. THE PROPOSED INTERACTIVE MODEL

We now describe the proposed sensor-cloud interactive model in detail using on-demand packet latency requirements of the applications. Note that the model can also be used generally for other factors like sensing frequency, packet reliability, etc. Table 1 and Table 2 present the list of symbols and the list of acronyms used in this paper.

A. SENSOR-CLOUD MODEL

In this subsection, we present the model for sensor-cloud integration, as illustrated in figure 1.

1) PHYSICAL SENSORS

physical sensors form a physical WSN. We use the following attributes for each sensor: ID (i.e., a unique integer label); type i_τ with $i_\tau \in \tau = \{\tau_1, \tau_2, \ldots, \tau_N\}$, a set of sensor types (i.e., temperature, humidity, light,...) registered with the sensor-cloud; and S, which contains scheduling parameters (i.e., wakeup and sleep interval) to determine how long a node sleeps and wakes up every cycle. We model a sensor node i as follows.

$$i = (i_{ID}, i_\tau, i_S), i_\tau \in \tau.$$

TABLE 1. List of symbols.

Symbol	meaning
τ	a set of sensor types (i.e., temperature, humidity)
τ_α^*	a of sensor types requested by application α
L_r^α	dedicated latency requirement of an application α
A	a set of applications $A = (\alpha_1, \alpha_2, ..., \alpha_N)$
L_r^A	a set of latency requirements of applications in A
L_r^c	the consolidated latency requirement of applications
f_i	a sensing flow routed through node i
F_i	a set F of sensing flows routed through node i
$L_m^{f_i}$	actual packet latency of a sensing flow f_i
γ	a set of virtual sensors
γ_α^*	a set of virtual sensors for application α
SI	a set of sensing data types of interest
RI	a region of interest of an application
qTAS	queue threshold adjusting step
ζ	a set of physical sensors
D_i	duty cycle of node i
ϕ_i	the queue length value of i
φ_i	the queue threshold value of i
T_s	sleep period of a sensor in a cycle
T_w	wakeup period of a sensor

TABLE 2. List of acronyms.

Acronym	meaning
SC	scheduling controller
PSM	Physical Sensor Manager
VSM	virtual sensor manager
QTU	queue threshold update
LPL	low-power listening protocol
CCA	Clear Channel Assessment
CoAP	Constrained Application Protocol
TOSSIM	TOSSIM simulator in TinyOS
CC2420	a RF transceiver type for sensors

2) SENSOR-CLOUD

sensor-cloud consists of virtual sensors that are software images of the physical sensors in cloud environment and responsible for providing sensing-as-a-service to applications [7]. We use the following attributes for a cloud c: ID, resources, a set of τ sensor types, and price options P (i.e., price P_1 for sensing services with the QoS Q_1 (i.e., packet latency) provided by a sensor, price P_2 for sensing services with the QoS Q_2 provided by the sensor). According to sensor-cloud's pricing models [5], [7], [14], the price which sensing service consumers are charged is corresponding to the QoS (i.e., packet latency) that they request (i.e., $P_1 > P_2$ with $Q_1 < Q_2$). We model sensor-cloud c as follows.

$$c = (c_{ID}, c_\tau, c_P).$$

Definition: a virtual sensor can be considered as a software emulation of a physical sensor, which receives its data from the underlying physical sensor and provides sensing data distribution transparently to users/ applications [7], [8], [10]. Virtual sensors inherit all attributes from their corresponding physical sensors and contain metadata for mapping purposes.

3) APPLICATION

We use the following attributes to characterize an application α: ID, a set of sensing data types of interest α_{SI} (i.e., temperature, humidity, or light,...), a region of interest α_{RI} (i.e., sensors in the region RE_1 or sensors in the region RE_2), and quality of service requirements (i.e., packet latency requirement L_r^α, sensing interval, or data accuracy). As sensor cloud is proposed to make applications transparent from the underlying physical WSNs [7], we specify only α_{SI} for an application [23]. We later define a mapping function to map α_{SI} to τ^*, a set of sensor types. Each application α may request a different latency requirement L_r^α, namely a **dedicated sensing latency requirement**. We model an application α as follows.

$$\alpha = (\alpha_{ID}, \alpha_{SI}, \alpha_{RI}, L_r^\alpha)$$

B. THE INTERACTIVE OPERATIONS

As shown in figure 1, the model consists of a down-stream from cloud-to-sensors (C2S) for application requests and an up-stream from sensors-to-cloud (S2C) for sensing traffic.

The sensor-cloud is considered as a middleware between physical sensors and applications, which virtualizes physical sensors into virtual sensors [23]. The virtual sensor manager (VSM) manages all information of virtual sensors such as metadata and applications which are using their sensing services, and their current queue threshold, etc.

A mapping function to map a set of physical sensor nodes ζ (i.e., $\zeta = \{p_1, p_2, p_3, \ldots\}$, with p_i is the i^{th} physical sensor) to a set of virtual sensors γ (i.e., $\gamma = \{v_1, v_2, v_3, \ldots\}$, with v_i is the i^{th} virtual sensor) is modeled as follows

$$f_{phy->vir}(\zeta) = \gamma$$

Based on the virtual sensors, the sensor-cloud distributes sensing services to applications. Sensing requests of applications are processed in the model as follows.

1) A new application α requests a sensing service by subscribing its request to SSaaS of the sensor-cloud. Based on the quality of service demand and the cost limitation [7], the application sends a request with the following specifications (1) α_{SI}, (2) region of interest α_{RI}, and (3) packet latency requirement L_r^α.

2) First, the SSaaS translates α_{SI} to $\tau_\alpha^* \subset \tau$, a set of actual sensor types (ST), using a mapping function that is modeled as follows.

$$f_{SI->ST}(\alpha_{SI}) = \tau_\alpha^* = (\tau_j : \tau_j \in \tau)$$

With known τ_α^* and α_{RI}, the VSM is called to allocate γ_α^*, a set of virtual sensors which will be responsible for providing sensing services to the application. We model the allocation function as follows.

$$f_{viralloc}(\alpha_{RI}, \tau_\alpha^*) = \gamma_\alpha^* = (\gamma_j : \gamma_{j->type} \in \tau_\alpha^*) \text{ and } \gamma_{j->location} \in \alpha_{RI}$$

3) The application requests are then forwarded and aggregated at the Request Aggregator.

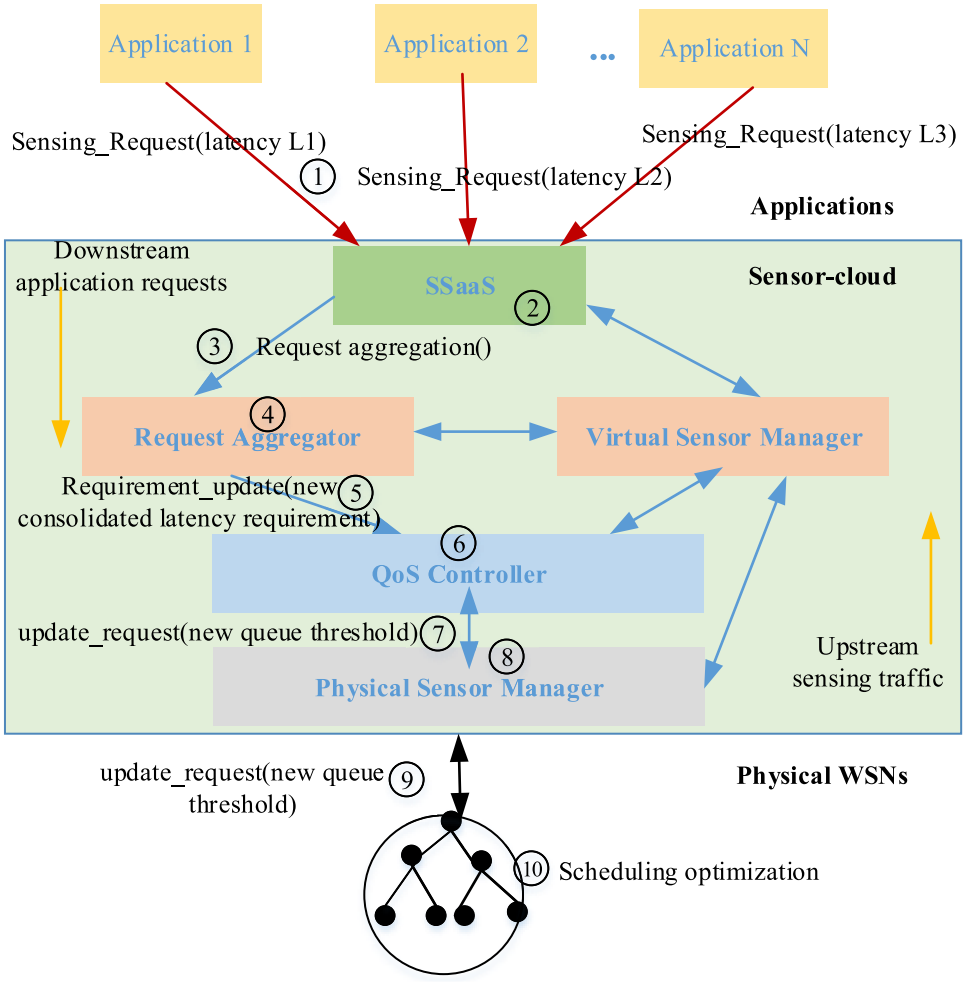

FIGURE 1. The proposed interactive model for the sensor-cloud.

4) In the Request Aggregator, the application requests are processed. The aggregator determines whether or not a new consolidated latency requirement is found for the physical sensors. We define the consolidated latency requirement of a set of applications as a combined representative requirement for the application set. When sensing data meet the consolidated latency requirement, dedicated latency requirements of each application are also satisfied. If the aggregator does not detect any change in the consolidated latency requirement (current sensing data already satisfies the requirement of the new application), the application request is hidden from sensor nodes.

5) If a new consolidated requirement for latency L_r^c is found, the aggregator pushes a request piggybacked with the new information of L_r^c to the QoS controller for latency requirement update.

6) The QoS controller executes processing the received request. It first checks if L_r^c can be satisfied with current sensing data or not. If current sensing data satisfy L_r^c, the request is then hidden from physical sensor nodes. Otherwise, the QoS controller prepares a queue threshold update (QTU) by calculating a new queue threshold for physical sensors. The detailed procedures for the request aggregator and queue threshold calculation are presented in the next sections.

7) If a QTU is required, the QoS controller sends QTU requests to the PSM (Physical Sensor Manager) with the new queue threshold.

8) The PSM executes a mapping from the virtual sensor set γ_α^* to a physical sensor set ζ. The mapping function is modeled as follows.

$$f_{vir->phy}(\gamma) = \zeta = f^{-1}_{phy->vir}(\zeta)$$

9) The PSM then sends the QTU request to corresponding physical sensors.

10) Upon receiving a QTU request, the physical nodes update their queue threshold and adapt their scheduling accordingly.

An aggregation scheme for the Request Aggregator is presented in the next subsection.

C. THE REQUEST AGGREGATOR

When the request aggregator receives a new application request for sensing services targeting for a set of virtual sensors, it queries the VSM about detailed information for the virtual sensors (i.e., current consolidated latency, applications). It then classifies virtual sensors into a set (i.e, γ_A^*), that are used by the same application set (i.e., A) including the new application.

We assume a virtual sensor set γ_A^* which is requested by an application set A including the new application. Their current consolidated requirement for latency is L_r^c. A new application α has a dedicated latency requirement L_r^α. The request Aggregator is responsible for aggregating the application requests to determine a single consolidated latency requirement of sensing flows, which meets latency requirements of the applications in A.

Definition: A sensing flow f_i meets a dedicated latency requirement L_r^α of an application α if its packet latency $L_m^{f_i}$ is equal or lower than L_r^α.

The aggregation procedures for application requests performed at the aggregator are shown in Algorithm 1.

Algorithm 1 Aggregation Procedures for New Application Requests

INPUT: $\gamma_A^*, A, L_r^c, L_r^\alpha$
OUTPUT: updating-flag, new L_r^c if updating-flag $= 1$
Initialize: updating-flag = 0
Repeat
  if $L_r^\alpha < L_r^c$ **then**
    $L_r^c = L_r^\alpha$;
    updating-flag $= 1$
    return updating-flag;
  end if
  return updating-flag;
  UNTIL all application requests are processed.

According to the definition above, the consolidated latency requirement is determined as the minimum dedicated latency requirement of applications.

Given a set with N dedicated latency requirements $L_r^A = (L_r^{\alpha_1}, L_r^{\alpha_2}, \ldots, L_r^{\alpha_N})$ requested by a set of N applications $A = (\alpha_1, \alpha_2, \ldots, \alpha_N)$ for a virtual sensor set γ_A^*. We denote L_r^{min} as the minimum latency requirement, $\min(L_r^{\alpha_1}, L_r^{\alpha_2}, \ldots, L_r^{\alpha_N})$, of N dedicated latency requirements. According to the algorithm, we have

$$L_r^c = L_r^{min} \leq L_r^{\alpha_i}, \quad \forall \alpha_i \in A$$

Therefore, if sensing data from γ_A^* satisfies L_r^c, the data also meet dedicated latency requirements of all applications in A. We thus consider $L_r^c = L_r^{min}$ as the optimal consolidated latency requirement to minimize the number of update requests transmitted to physical sensors for saving energy. As the current value of L_r^c is also the existing applications' minimum latency requirement, in the aggregation algorithm we only compare the requirement of the new application L_r^α with L_r^c. However, we specify a general aggregation function as presented below. The purpose of the general aggregation function is to find the updated minimum latency requirement of applications when there are changes (receiving new requirements, updating requirements, or service un-subscription).

$$aggregation(L_r^A) = \min(L_r^{\alpha_1}, L_r^{\alpha_2}, \ldots, L_r^{\alpha_N})$$

If a new consolidated latency requirement is determined (updating-flag $= 1$), the aggregator sends a requirement update request to the QoS controller. If the current L_r^c still meets requirements of the applications including the new one (updating-flag $= 0$), no update is required. The request of the new application is hidden from physical sensor nodes. In this case, both the new client and the network service provider are beneficial. In particular, the new client may receive better quality sensing traffic while the network service provider can get more revenue without extra overhead consumption in the WSNs. This is also an advantage of the proposed framework.

D. THE SCHEDULING CONTROLLER AND QOS CONTROLLER

The QoS controller on the sensor-cloud is used to control the e2e packet latency of sensing flows. The QoS controller interacts with the scheduling controller at sensor nodes and enforces sensor nodes to adapt their operations for guaranteeing the latency requirement or optimizing energy consumption. The interaction between QoS controller and scheduling controller happens in the following cases: 1) the aggregator observes a new consolidated latency requirement for a sensing flow, which is not satisfied by current sensing data 2) the QoS controller detects the latency of a sensing flow is higher or significantly lower than the current consolidated latency requirement.

Sleep latency [36] is considered as the main source of the e2e packet latency in duty-cycled WSNs. Queuing delay is popularly used to indicate sleep latency as well as other sources of delay like congestion due to high incoming packet rate or low link quality.

In the literature of traditional WSNs, several studies propose to adapt duty cycle of a node to guarantee a certain latency requirement based on queue managements [15]. However, it still lacks a study for the duty-cycle control based delay guarantee in dynamic environments where a node may concurrently serve multiple applications and delay requirements may dynamically change overtime. Dealing with such a dynamic characteristic is desirable in sensor-cloud.

We propose to enable the sensor-cloud to control [17] the queue threshold of sensor nodes to guarantee on-demand e2e latency requirements of applications. The queue threshold of a node is considered as the maximum size allowed for messages on its queue. The queue threshold is not the queue capacity and the value of the queue threshold can be adjusted depending on the latency requirement. The general idea of designing the scheduling controller based on the

queue threshold is that the scheduling controller of node i tends to adapt its sleep period so that its queue length value ϕ_i is equal to or lower than the queue threshold. For example, if a lower queue threshold is assigned and the packet generation rate is a constant, the node needs to transmit more packets in a control interval [17]. As a result, the node has to shorten its sleep period and wake up more frequently to send data packets. The packet delivery latency is thus reduced. In our design, the queue threshold at a node i is inversely proportional with its duty cycle value and directly proportional with the latency of its sensing flows. For that reason, by controlling the queue threshold, the QoS controller in the sensor-cloud can control the duty cycle of sensors and the packet latency of their sensing flows. Note that the end-to-end packet latency mentioned in this paper is the sensor-to-cloud packet latency which consists of sensor-to-sink latency and sink-to-cloud latency. With a wired connection, the sink-to-cloud latency is low and stable compared with the sensor-to-sink latency. Therefore, we focus on adapting sensor-to-sink latency to on-demand latency guarantee.

The QoS controller is responsible for checking average latency of sensing data flows in each control interval I_c, to detect flows with longer latency or significant shorter latency compared with the latency requirement. The QoS controller also gathers feedback from different flows, calculates new queue thresholds for related sensor nodes, and provides guidance to enable physical sensor nodes to adapt their scheduling locally so that 1) packet latency of all flows is controlled to meet their latency requirements and 2) energy consumption of a sensor is optimized.

In our design, the scheduling controller running locally at a physical sensor node is responsible for adapting the node's wakeup scheduling based on the queue threshold value under guidance provided by the QoS controller. In duty cycled WSNs, a node periodically sleeps for a sleep period T_s and then wakes up with a wakeup period T_w which is long enough for transmitting a data packet and receiving an acknowledgment. The cycle length of the node is equal to $T_w + T_s$. We assume T_w as a constant. The queue length in a control interval k (the length of control interval is I_c) of node i is computed as follows.

$$\phi_i^k = max(0, \phi_i^{k-1} + P_{self} + P_{in} - P_{out}) \quad (1)$$

where P_{out}, P_{in}, and P_{self} are the number of transmitted packets, incoming packets, and self-generated packets (sensing packets produced at the sensor node) from the control interval $k-1$ to the control interval k, respectively. We denote P_i as the number of input packets at the node i, with $P_i = P_{self} + P_{in}$. We calculate $P_{out} = RT_wI_c/(T_w + T_s)$, with R as the packet transmission success ratio.

The QoS controller calculates queue threshold for a node as follows.

Given a WSN modeled as a directed graph $G = N, L$ rooted at sink node which consists of a set of sensors N, a set L of links, and a set F of sensing flows. Each sensing flow $f_i \in F$ originated from the source node i is forwarded through i's intermediate nodes, then to the sink and the sensor-cloud. We assume f_i has a latency requirement of $L_r^{f_i}$ and denote $L_m^{f_i}$, φ_i, and φ_i^{max} as the current actual latency value of the flow, the current queue threshold value, and the maximum queue threshold value of i, respectively. We set φ_i^{max} equal to queue capacity of i. We use average duty cycle as an indicator for energy efficiency of a sensor node, and denote D_i as the average duty cycle of node i. The objective of the controllers is to improve energy efficiency of the sensor node i (i.e., by minimizing D_i) while maintaining the latency of all sensing flows routed through i satisfied the latency requirement of applications. We summarize the objective function as follows.

Objective: minimize D_i subject to $L_m^{f_i} < L_r^{f_i} \forall f_i$.

We first define an expected range for the latency of a flow f_i as $[L_r^{f_i} - \epsilon, L_r^{f_i}]$. The guard time ϵ helps detect incipiently excessive latency. The greater the value of ϵ the quicker the sensor-cloud can react to avoid the potentiality that the actual latency of the flow does not meet the latency requirement (for example, when the measured latency falls into the range, QoS controller shouldn't increase the threshold value further to stop a growing trend of the latency, that may potentially result in an excessive latency). If $L_m^{f_i} < L_r^{f_i} - \epsilon$, energy efficiency of sensors in the flow is not optimized. The reason is that the sensors can still reduce their duty cycle for saving energy so that $L_m^{f_i}$ falls into the expected latency range and the latency requirement is still satisfied. We name this as a false positive case. If $L_m^{f_i} > L_r^{f_i}$, this sensing flow does not satisfy its latency requirement. We name this as a false negative case.

If the QoS controller detects an occurrence of a false positive case or a false negative case at any flow f_i, the controller will adjust queue threshold value of corresponding nodes to force the nodes for optimizing their energy consumption locally and adjust latency of f_i into the expected range. We define the queue threshold adjusting step ($qTAS_i$) of a node i as the gap value of its queue threshold in each adjustment (for example, if $qTAS_i = 2$, the queue threshold of i is increased by 2 units in each adjustment).

The queue threshold adjusting step ($qTAS_i$) of node i in f_i is proportional to the gap δ_{f_i} between $L_m^{f_i}$ and $L_r^{f_i}$.

$$\delta_{f_i} = L_m^{f_i} - L_r^{f_i}$$

We assume node i (i.e., a relay node) has a set F_i of m sensing flows $F_i = f_i^1, \ldots, f_i^j, \ldots, f_i^m$ routed through i. Each flow f_i^j may have a different gap $\delta_{f_i^j}$. In our implementation, we use the maximum value among values of $\delta_{f_i^j}$ to calculate $qTAS_i$ as follows.

$$\delta_{f_i} = max(\delta_{f_i^1}, \ldots, \delta_{f_i^m})$$

The purpose is to achieve a target that all sensing flows satisfy their latency requirement. According to this policy, satisfying the latency requirement of sensing flows at a node is set a higher priority compared to saving energy. For instance, if node i has flow f_i^x with an excessive delay ($\delta_{f_i^x} > 0$) and another flow f_i^y having latency lower than the requirement

($\delta_{f_i^y} < 0$), our mechanism adjusts the node queue threshold according to f_i^x.

The $qTAS_i$ (if required) is calculated as follows.

$$qTAS_i = \begin{cases} -min(\varphi_i, \varphi_i * \delta_{f_i}/L_r^{f_i}) & \text{if false negative} \\ min(\varphi_i^{max} - \varphi_i, \varphi_i * |\delta_{f_i}|/L_r^{f_i}) & \text{if false positive} \end{cases} \tag{2}$$

After calculating $qTAS$ for update required nodes (i.e, nodes having sensing flows' latency out of the expected range or having a new consolidated requirement), the QoS controller sends QTU requests to the required nodes through the PSM, and updates the queue threshold value of corresponding physical and virtual sensors. Upon the reception of a QTU request, node i updates its queue threshold $\varphi_i = \varphi_i + qTAS_i$. The scheduling controller (SC) of i then computes its new sleep period T_s^k for a new control interval k [17] to enhance its scheduling as follows.

$$T_s^k = T_s^{k-1} + \lambda(\varphi_i^k - \phi_i^k) - \theta(\phi_i^k - \phi_i^{k-1}) \tag{3}$$

where θ and λ are the two control parameters [17] of SC, which are simply real numbers reflecting the weights of the remaining queue capacity ($\varphi_i^k - \phi_i^k$) and the queue changing velocity ($\phi_i^k - \phi_i^{k-1}$) in calculating the sleep period of a node. The selection of θ and λ is presented in section IV. The scheduling controller adapts T_s based on the value of queue threshold φ_i^k, queue changing velocity ($\phi_i^k - \phi_i^{k-1}$), and remaining queue capacity ($\varphi_i^k - \phi_i^k$) of a node. The remaining queue capacity reflects the current traffic condition of a node in comparison to the queue threshold. The queue changing velocity indicates changes in incoming traffic rate as well as the successful packet transmission ratio.

We also formulate the second version of SC (SC2) by directly considering R as follows.

$$T_s^k = T_s^{k-1} + \lambda R(\varphi_i^k - \phi_i^k) - \theta(\phi_i^k - \phi_i^{k-1}) \tag{4}$$

The performance comparison of SC2 and SC is presented in section IV to verify the necessity of considering R directly as used in DutyCon [15].

It is easy to prove the proposed controlled system is stable according to standard procedures of control theory [17]. The stability analysis is presented in section IV.

1) FEATURES OF THE SCHEDULING CONTROLLER

We now analyze behaviors and features of the scheduling controller regarding its parameters separately as follows. The scheduling controller at a node operates based on guidance of the QoS controller as well as local queue conditions.

For φ_i^k, when the QoS controller detects that latency of a sensing flow through the node i does not meet the requirement, the QoS controller requests node i to reduce its queue threshold. As the queue threshold value is decreased, its local scheduling controller also reduces the sleep period for quickly adjusting latency of the flow to an expected range. On the contrary, once the QoS controller detects that latency of a sensing flow through i falls below the expected range which indicates an energy wasted scenario, the QoS controller requests the node to increase its queue threshold. As a result, the local scheduling controller at the node increases its sleep period so that its energy consumption is reduced while latency of the flow is adjusted to the expected range. In this way, the QoS controller controls the queue length and latency of flows through the node.

For ($\varphi_i^k - \phi_i^k$), if the value of remaining queue capacity is large, the scheduling controller allows the node to sleep more. The smaller the remaining queue capacity the earlier the node must wake up to transmit packets. A queue threshold protection mechanism is also implemented as follows. Whenever the queue threshold is violated or there is a packet drop due to queue overflow (i.e., the incoming traffic rate increases significantly), the node reduces its sleep period to enable it to forward packets faster to 1) protect the queue threshold 2) ensure the delay requirement under dynamic traffic rates. Note that the queue threshold value is normally smaller than the queue capacity.

For ($\phi_i^k - \phi_i^{k-1}$), when incoming traffic rate at the node increases, the scheduling controller tends to reduce the sleep period of the node for accelerating packet forwarding and avoiding excessive queuing delay. Oppositely, when the incoming traffic rate decreases, the controller tends to allow the node sleeping more if latency of flows through the node allows. In other cases, assuming other parameters (i.e., traffic rate) as constant, if the successful transmission ratio is decreased (i.e., in highly interference scenarios), the node's queue length will be increased up according to (1). According to (3), the node has a tendency to reduce its sleep period and wake up more frequently to transmit packets. Combining (3) and (1), we find that the scheduling controller at a node indirectly controls the latency of all sensing flows routed through the node.

By exploiting powerful resources of the cloud, our model enables the QoS controller to gather latency feedback information of sensing flows, aggregate, and calculate queue threshold for sensor nodes. Each related sensor node receives and processes only one request. In other hands, if sensor nodes calculate the queue threshold by themselves, they are required to gather all latency feedback information from the sensor-cloud. This may create flooding in the constrained WSNs which is obviously inefficient. In addition, if a node has multiple flows routed through it, constrained sensor nodes may not have enough resources for gathering, containing feedback information of all flows, and for queue threshold computation.

2) LAZY MODE AND ACTIVE MODE OF QOS CONTROLLER

False cases may happen to a flow when 1) it has a new consolidated latency requirement, 2) network conditions are degraded severely, or 3) when burst packet transmission from some nodes may affect the latency of several sensing flows. However, burst packet transmission rarely occurs in periodic sensing services. After detecting false cases

(i.e., both positive and negative), the controllers force latency of sensing flows into the expected range. After that, latency of the sensing flows may be kept stable within the expected range.

To save computing resource for the QoS controller, we propose two operation modes for the QoS controller including lazy mode and active mode. Whenever a false case occurs to a flow, the active mode is enabled for the QoS controller. Under the active mode, the QoS controller runs with a short control interval, I_c^{active}, to quickly detect and force latency of the sensing flow into the predefined expected latency range. If the QoS controller finds that latency of the sensing flow achieves a stable state within the expected range during a period equal to ψ consecutive control intervals and there is no additional false cases, the QoS controller switches its operations to lazy mode. In lazy mode, the QoS controller runs with a longer control interval I_c^{lazy} for saving computing resources. In adaptive duty-cycled WSNs, each sensor may operate with a different duty cycle, thus duty-cycled information of a sender may need to be notified to its receiver. For this feature, we reuse the technique [39], [46], [47] implemented in our previous study for adaptive duty-cycled schemes.

IV. PERFORMANCE EVALUATION

We conduct analysis and extensive experiments with a network [48] consisting of one sink node, 54 temperature sensors and 54 humidity sensors, one sensor-cloud, with 7 different applications. As implemented in our previous work [49], HTTP-CoAP converter [49] are used to convert HTTP requests of applications to CoAP requests. Each application is assumed to request for one of the sensing data types above. Application requests are encoded with XML templates, which are then decoded using SensorML interpreter [23]. LPL and CTP [36], [46] are used as a duty cycled MAC protocol and tree-based data collection protocol, respectively. The QoS controller measures the latency of a flow based on packets' timestamps. We use the radio noise model with closest-fit-pattern matching (CPM) [36]. To measure the costs like duty cycle, we record state changes in the radio and use counters to accumulate time period in each state. We use the default CCA checks setting (up to 400 times) of the TinyOS LPL. Table 3 shows detailed parameters used in the simulations. For other parameters, we use default values of TOSSIM radio model for CC2420. The sensor-cloud with virtual sensors can be implemented prototypes with Network Function Virtualization (NFV) [33] approach available in OpenStack [50] as presented in our previous work [24]. Results are obtained with the scheduling controller SC if the controller SC2 is not mentioned explicitly. The performance of the proposed model is compared with [15] in terms of scalability and the number controllers for the cases of multi-flows, and with [15], [16] in terms of energy efficiency and packet latency for the case of a single latency requirement because Delar [16] considers the case of a single latency requirement only.

TABLE 3. Parameters.

parameter	value	parameter	value
T_w	10 ms	Data payload	50 bytes
Platform	CC2420	CCA checks	400 times
Tx energy rate	17.4 mA	Rx energy rate	18.8 mA
Lx energy rate	18.8 mA	λ	0.0005
θ	0.001	I_c^{active}	1-5 s
I_c^{lazy}	1 min	Tx range	20 m
ϵ	$5-10\%L_r$	Sensing interval	5 s

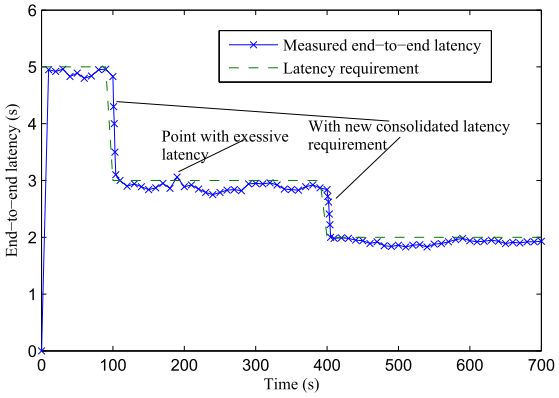

FIGURE 2. End-to-end latency over time with different number applications.

A. DYNAMIC NUMBER OF APPLICATIONS

Figure 2 presents the end-to-end latency over time of a sensing flow compared to its consolidated latency requirements once seven applications are deployed. Each application requests for sensing services (in an ascending order) with a request interval of 100 s. Latency requirement requested by applications 1^{st} to 7^{th} are 5.0 s, 3.0 s, 4.0 s, 6.0 s, 2.0 s, 3.5 s, 2.5 s, respectively. The purpose of selecting a variation of latency requirements is to expose performance behaviors of the model under a different number of applications.

The figure shows that the e2e latency values fluctuate around the line of latency requirement in all cases. In normal cases, the latency graph fluctuates only slightly due to a small latency variation at each hop. Significant changes in the end-to-end latency occur once there are new application requests with a new consolidated latency requirement. Whenever a new consolidated latency requirement (i.e., at 400 s and 100 s) or sudden excessive latency (i.e., at 180 s and 590 s) is detected as presented in the figure, the controllers force the packet latency quickly into the expected latency range. The e2e latency of sensing flows is maintained and controlled well closely under the consolidated latency requirement. The results prove that the proposed model guarantees end-to-end latency effectively.

Average duty cycle of sensors along the flow under a various number of applications is presented in figure 3. Figure 3 shows that duty cycle of sensors using our proposed model does not rely on the number of application requests, but the consolidated latency requirement. In particular, duty cycle of a node changes when it obtains a new consolidated latency

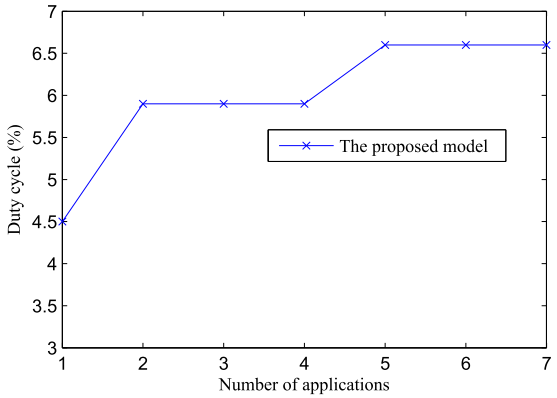

FIGURE 3. Average duty cycle vs. number of applications.

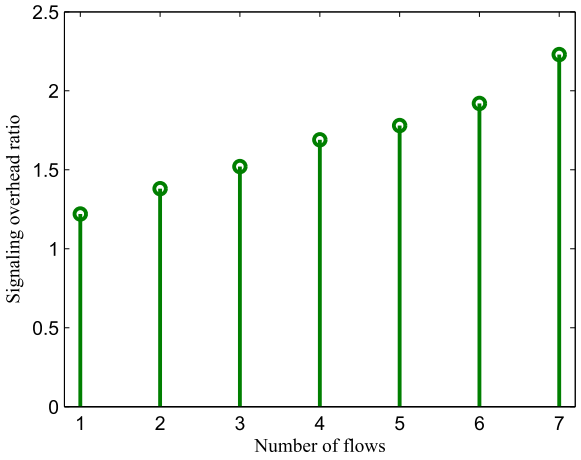

FIGURE 4. Signaling overhead ratio between DutyCon and the proposed model vs. number of flows.

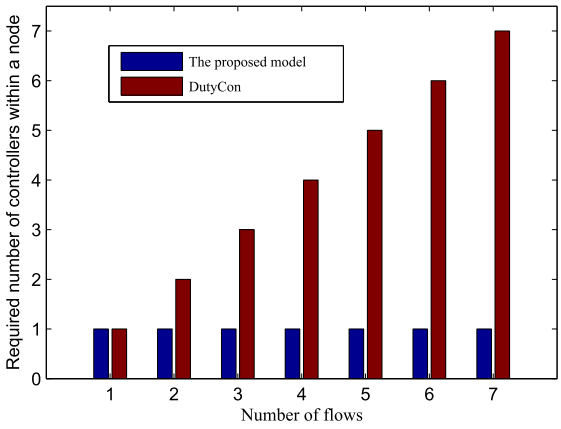

FIGURE 5. Required number of controllers vs. number of flows.

requirement (for example, under the request of 5^{th} and 2^{nd} applications). The framework actually hides sensing requests by the 3^{rd}, 4^{th}, 6^{th}, and 7^{th} applications from sensors. Adding those applications does not affect duty cycle of sensors since their dedicated latency requirement is already satisfied based on the current consolidated latency requirement. Although wireless sensor networks are shared among multiple applications, each sensor node is required to operate with only a single schedule. This indicates that the model facilitates sensing data reusability among applications and minimizes the number of requests sent to physical sensors to conserve energy. In this way, the model can achieve a good scalability.

As the current duty-cycled control schemes (i.e., DutyCon) for the e2e latency guarantee do not support multiple applications with latency requirement changes, we compare scalability between the proposed model with DutyCon in experiments with a different number of flows.

B. DYNAMIC NUMBER OF FLOWS

In this section, we conduct experiments with a different number of flows (from 1 to 7). Latency requirements for 7 flows are as same as those for the 7 applications described above. We compare the proposed model to DutyCon [15], the state-of-the-art latency guaranteed mechanism for traditional WSNs using control theory. Note that DutyCon also supports multi-flow scenarios. In data collection traffic patterns, each intermediate node may serve more than one sender and sensing flows generated by different source nodes can share the same intermediate node for packet relaying. Sensing flows may have the different or same requirement for packet latency. In DutyCon, a sensor node maintains a sleep schedule and a controller for each flow. Therefore, each sensor may be required to have several sleep schedules simultaneously for multi-flow scenarios. As a result, a sensor may have to waking up several times within a cycle when wakeup timers of any of its sleep schedules fire. An intermediate node goes to sleep only when all of its sleep schedules are in a sleep state.

We do experiments with DutyCon and the proposed scheme using a fixed latency requirement for sensing flows through a node. We record behaviors of the sensor node when we add a number of additional flows routed through the sensor.

Figure 4 illustrates the signaling overhead ratio of the node operating with DutyCon and proposed model. In all of the cases, the proposed model shows a lower overhead even though overhead for QTU is taken into account. The ratio increases considerably in proportional to the number of sensing flows. The reason is that the sensor-cloud in the proposed model gathers feedback information from flows and computes the queue threshold value for the node. In DutyCon, each sensor needs to operate multiple controllers for different sensing flows and each sensor node is required to gather feedback from all flows separately. The results can also be explained using Figure 5 and Figure 6.

Figure 5 and figure 6 present the required number of controllers and sleep schedules for a node when we increase the number of flows through it. In the proposed model, all flows' feedback information is aggregated by the sensor-cloud, thus only one sleep schedule and one controller are required for each sensor. In DutyCon, the number of sleep schedules and controllers is proportional with the number of flows. This leads to high overhead, increases the number of wakeups

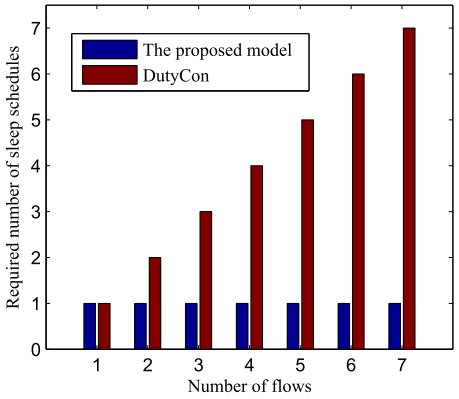

FIGURE 6. Required number of sleep schedules vs. number of flows.

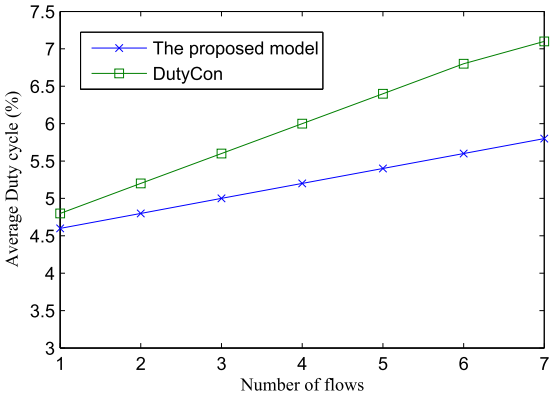

FIGURE 7. Duty cycle vs. number of flows.

for each sensor node, increase the complexity, and limits the scalability of the overall system.

High signaling overhead and more frequent wakeups are the main reasons which result in a high energy consumption at nodes running with DutyCon compared to nodes running with the proposed model, as illustrated in figure 7. Energy consumption of DutyCon increases quickly when we increase the number of flows. We find that the energy efficiency improvement of the proposed model compared to DutyCon is proportional to the number of flows.

C. SCALABILITY TEST

We now evaluate the scalability of the systems. We use a stress testing method as follows. We conduct simulations with an increasing number of sensing flows routed through a node until it cannot accommodate a new flow due to out of resources (processing, duty cycle, or storage). We use default configurations of Skype motes in TOSSIM. At the end of simulations, we compute the average number of flows that each scheme can accommodate. We report the average results of 10 runs.

Figure 8 shows the average number of sensing flows that a sensor can accommodate in each scheme. The results indicate that the proposed scheme achieves a significant scalability improvement compared to DutyCon. While the proposed

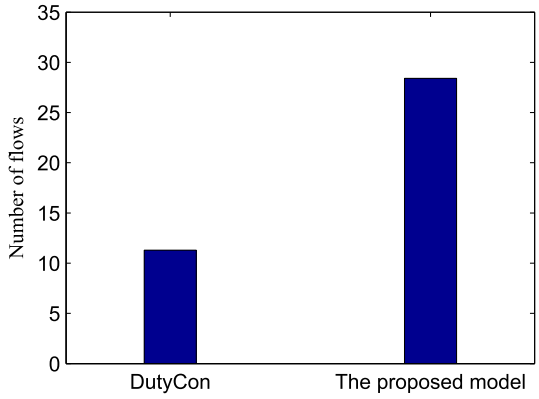

FIGURE 8. Scalability test: average number of sensing flows a sensor node can accommodate.

TABLE 4. Controlling overhead ratios.

Parameter	SC2/SC	DutyCon/SC
Average controlling overhead	1.26	1.64
Average duty cycle	1.18	1.42

model can accommodate on average of 28.4 flows, that of DutyCon is only 12.1 flows. This is due to the high overhead characteristic of DutyCon as shown in figures 4, 5, and 6.

D. SC/SC2 AND DUTYCON'S OVERHEAD COMPARISON IN NOISY ENVIRONMENTS

As SC2 directly considers the packet transmission success ratio, similar to DutyCon, we are interested in evaluating the control overhead of DutyCon, with SC and SC2 in a noisy indoor environment (i.e., with dynamic interference) as described in our previous work [46]. Controlling overhead ratios are presented in Table 4.

Under stable network conditions, we witness that SC and SC2 operate similarly while the controlling cost of SC2 is only slightly greater than that of SC. However, in noisy environments, the average controlling overhead of SC2 is significantly higher than SC. The control overhead of DutyCon is the most expensive. The reason is that SC2 and DutyCon are very sensitive to changes in the channels. Changes in the channels immediately affect the estimated value of R which triggers adaptations and updates frequently. Considering R directly may enable quick adaptation, but incurs high controlling overhead. As a result, the average duty cycle of nodes running DutyCon and SC2 are higher than nodes running SC. In SC, small and temporary changes in the environment are hidden from the controller. Environmental changes trigger updates the SC controller only once they result in changes in the queue output and input volumes.

E. DYNAMIC TRANSMISSION RATE

Through the previous experiments, we show clearly the benefits of the proposed model in comparison with DutyCon under a various number of applications and flows. We are now

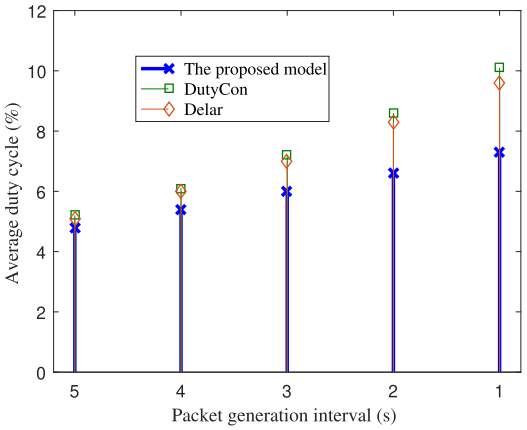

FIGURE 9. Average duty cycle under various packet generation intervals.

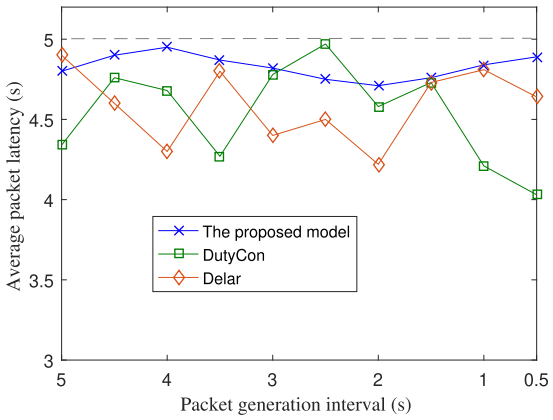

FIGURE 10. Average packet latency under different packet generation intervals (the latency requirement is fixed at 5 s).

interested in studying the benefits and performance behaviors of the proposed model compared to DutyCon and Delar [16] when the traffic rate is changed. We conduct experiments by changing the traffic rate in runtime while the latency requirement is fixed at 5 s as follows. The packet generation interval (I_g) of each sensor starts with a setting of 5 s and is then decreased.

Figure 9 reports the average duty cycle of sensors running with DutyCon, Delar and the proposed model when I_g is decreased from 5 s to 1 s. When the packet generation interval of source nodes decreases (i.e., the incoming traffic rate increases), duty cycle of all nodes along the route increases. However, DutyCon and Delar witnesses a faster increase in duty cycle. Nodes running the proposed model consume less energy than nodes running with DutyCon and Delar. The gap between the proposed model and the two schemes is increasingly greater when the incoming traffic rate increases. The reasons discussed above can be used to explain the energy efficiency improvement witnessed in the proposed model. In addition, the improvement is achieved as the proposed model solves a limitation of DutyCon which is interestingly presented in figure 10.

Figure 10 shows the average e2e packet latency of a flow under different packet generation intervals. When the incoming traffic rate changes, controllers of Dutycon and the proposed model adjust the scheduling parameters to meeting the latency requirement. Because Delar doesn't support the automatic adjustment, separate experiments have been done for each packet generation interval. In all schemes, average packet latency is maintained below 5 s (the latency requirement). However, variation in the chart of DutyCon and Delar is more significant than that of the proposed model. In some experiments, we find that the latency is below 3.5 s. The average packet latency value fluctuates from 3.9 s to 5 s. While latency values of the proposed scheme are controlled within the expected range closely below the latency requirement, latency values of DutyCon and Delar are considerably lower than the requirement and not stable. This leads to wasted energy consumption at sensor nodes because they may have to wake-up more frequent than necessary to transmit packets.

The results of DutyCon can be explained using the following reasons. First, the end-to-end packet latency of DutyCon is controlled indirectly by multiple single hop controllers. In each hope, DutyCon selects a sleep interval value for nodes to ensure that the single hop latency is not greater than the requirement for a single hop determined by DutyCon. Variations in real latency values at every hop contribute to a large variation of the end-to-end latency. Moreover, we also observe that the adaptation of e2e latency in sensing flows running DutyCon is slower than the proposed model because all sensors are required to gather feedbacks. In addition, nodes may not receive feedbacks for executing adaptation at the same time. There is no control and validation on the e2e latency directly in DutyCon. Second, DutyCon does not consider to define lower bound for the e2e latency and to control the packet latency of sensing flows within an expected range. In other words, DutyCon is not optimized for energy consumption of constrained sensors. In our model, the end-to-end latency is controlled directly. An adjustment mechanism is also designed to control the latency of sensing flows within an expected range satisfied the requirement of applications while energy consumption of sensors is optimized.

F. STABILITY ANALYSIS

In this subsection, we summarize the stability analysis of the proposed system and verify conditions of the system stability, following standard procedures of control theory [17].

In the steady state, $qTAS \rightarrow 0$ and the measured end-to-end latency should be within the expected range $(L_r - \epsilon, L_r)$. Therefore, we have the equilibrium for the latency at the steady state, denoted as L^{ss}, as follows.

$$L_r - \epsilon < L^{ss} < L_r \tag{5}$$

In steady state, the queue length of a node converges to its queue threshold. We denote P_i^{ss} of node i as its average incoming packet rate. Following standard procedures of control theory [17], we exploit the first-moment approximation method to find the average steady state solutions of the proposed system [51], [52] as presented in the appendix and obtain the following results.

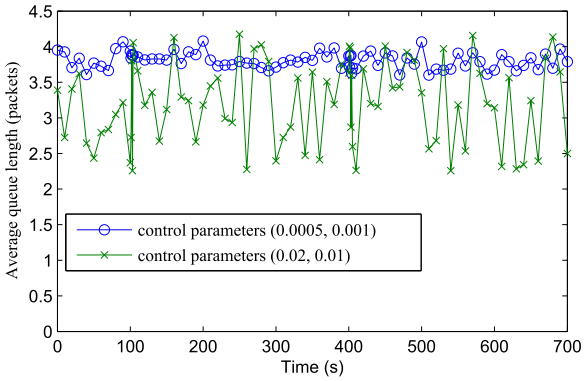

FIGURE 11. Average queue length over time with the two different groups of the control parameters ($\lambda = 0.0005$; $\theta = 0.001$) and ($\lambda = 0.02$; $\theta = 0.01$).

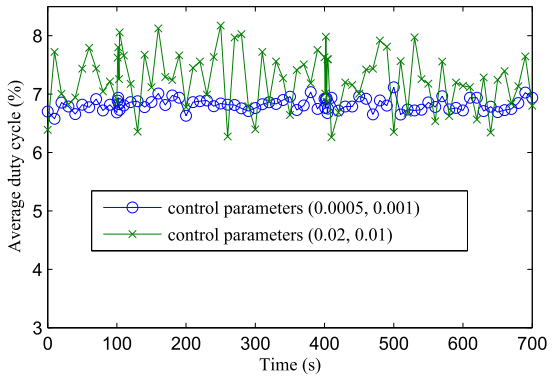

FIGURE 12. Average duty cycle over time with the two different groups of the control parameters ($\lambda = 0.0005$; $\theta = 0.001$) and ($\lambda = 0.02$; $\theta = 0.01$).

The scheduling controller of node i is asymptotically stable if the two parameters, λ_i and θ_i, satisfy:

$$0 < \lambda + 2\theta < \frac{4RT_wI_c}{P_i^{ss^2}} \tag{6}$$

$$0 < \theta < \frac{RT_wI_c}{P_i^{ss^2}} \tag{7}$$

G. STABILITY ANALYSIS'S VALIDATION

To validate the stability analysis, we conduct simulations with two different value pairs of the control parameters: ($\lambda = 0.0005, \theta = 0.001$) and ($\lambda = 0.02, \theta = 0.01$). $\lambda = 0.0005$ and $\theta = 0.001$ are default values used in this paper, which satisfy the stability condition in (29) and (30). The second pair of the control parameters doesn't satisfy (29) and (30).

Figure 11 presents the time evolution of average queue length. The average queue threshold in this case is set to four packets. The first chart with ($\lambda = 0.0005, \theta = 0.001$) shows a stable pattern of queue length. In particular, the average queue length slightly fluctuates below the queue threshold value. The second graph with the parameters $\lambda = 0.02$ and $\theta = 0.01$) shows larger oscillation in queue length. The large oscillation in queue length indicates a great oscillation in average duty cycle, as illustrated in figure 12. With greater values of control parameters which don't satisfy the stable conditions, the sleep period is adapted with a larger interval when the network has changes. This enables a quick adaptation, but also requires updates aggressively and high energy consumption for updates since the queue threshold as well as expected latency range are often violated. As a result, the queue length convergence is not observed in the second graph during experimental periods while average queue length in the first graph converges to its queue threshold.

V. DISCUSSION AND CONCLUSIONS

This paper proposes an efficient interactive model which enables the sensor cloud to guarantee latency requirements of multiple applications simultaneously. Experimental results show that the proposed model effectively controls the latency of sensing flows closely under the bound by the latency requirement. Compared to the state-of-the-art protocol, the proposed model not only supports multiple flows and multiple applications simultaneously with high scalability, but also requires a considerable lower signaling overhead. By saving energy resources for sensors, the proposed scheme helps reduce the cost for service providers. In addition, by achieving a high scalability, the proposed scheme allows a service provider providing the services within a WSN to more applications/clients to make more profits as well as reduce the price of sensing services for clients. For those reasons, the proposed scheme leads to a win-win model for both service providers and clients.

APPENDIX

A. FULL STABILITY ANALYSIS

In this appendix, we present the detailed analysis for the stability of the proposed system and verify conditions of the system stability, following standard procedures of control theory [17]. We exploit the first-moment approximation method to find the average steady state solutions of the proposed system [51], [52].

We assume ϕ_i^{ss}, T_s^{ss}, and φ_i^{ss} as the average stead-state value of ϕ_i, T_s, and φ_i, respectively.

In the steady state, $qTAS \rightarrow 0$ and the measured end-to-end latency should be within the expected range $(L_r - \epsilon, L_r)$. Therefore, we have the equilibrium for the latency at the steady state L^{ss} as follows.

$$L_r - \epsilon < L^{ss} < L_r \tag{8}$$

From (3), we can also find the correlation between the equilibriums of ϕ and φ as follows.

$$\phi_i^{ss} = \varphi_i^{ss} \tag{9}$$

In steady state, queue length of a node converges to its queue threshold. We denote P_i^{ss} of node i as its average incoming packet rate and $d_i(k)$ as its queue deviation ratio in the interval k.

$$d_i(k) = \frac{\phi_i(k) - \phi_i^{ss}}{\varphi_{max}} \tag{10}$$

From (10) and (1), we have

$$d_i(k) = d_i(k-1) - \frac{1}{\varphi_{max}}[\frac{RI_cT_w}{(T_s+T_w)} - P_i] \quad (11)$$

Let $\mu = 1/\varphi_{max}$. By using the asymptotic theory [53]–[55], we obtain asymptotic approximation of (11) as follows.

$$f_i(k) = f_i(k-1) + \mu(P_i^{ss} - \frac{RI_cT_w}{(T_s+T_w)}) \quad (12)$$

From (12), clearly that if $\mu \rightarrow 0$, we have $\mid d_i(k) - f_i(k) \mid \rightarrow 0$. Then from (9), we observe the equilibrium of T_s^{ss} as below.

$$T_s^{i-ss} = (\frac{R_iI_c}{P_i^{ss}} - 1)T_w \quad (13)$$

As presented in section 2, PSM pushes queue threshold update requests to sensors when the latency of sensing flows is greater than the latency requirement, under the expected range, or with a probability when the measured latency is in the range $(L_r - \epsilon, L_r)$. We specify the queue threshold updated function of node i as follows.

$$U_p = \begin{cases} 1, & \text{if } L_i > L_r \\ 0, & \text{if } L_i < L_r - \epsilon, \\ 1 - (L_r - L_i)/\epsilon, & \text{otherwise} \end{cases} \quad (14)$$

Based on queue threshold updated function, a node may need to update the queue threshold accordingly.

$$\varphi_i(k) = \begin{cases} \varphi_i(k-1) + qTAS_i, & \text{if } U_p = 1 \\ \varphi_i(k-1) - qTAS_i, & \text{if } U_p = 0, \\ \varphi_i(k-1), & \text{otherwise} \end{cases} \quad (15)$$

By applying the saturation non-linearity theory, we obtain the approximation of the function as follows.

$$U_p(k) = Sat\{\varphi_1(e^{-\varphi_2 \wedge(\phi_i, T_s^i)} + e^{\varphi_2 \wedge(\phi_i, T_s^i)})\} \quad (16)$$

where the saturation function $Sat(Z) = min(Z, 1)$, and

$$\wedge(\phi_i, T_s^i) = max_{j \in F_i} \sum_{i \in I_j} (T_s + T_w)\phi_i(k) \quad (17)$$

φ_2 is the minimum value which satisfies the following condition.

$$\frac{e^{-\varphi_2(L_r-\epsilon)} + e^{\varphi_2(L_r-\epsilon)}}{e^{-\varphi_2 L_r} + e^{\varphi_2 L_r}} \leq \psi \quad (18)$$

with ψ ($\ll 1$) is a positive coefficient and $\varphi_1 = \frac{1}{e^{-\varphi_2 L_r} + e^{\varphi_2 L_r}}$. The probability value of a point near $L_r - \epsilon$ is close to zero which leads to a smaller ψ. As in the stead state, the maximum delay should be within the expected range, we have

$$L_r - \epsilon < \wedge(\phi_i^{ss}, T_s^{i-ss}) < L_r \quad (19)$$

From (13), (17), and (19), we have

$$\frac{L_r - \epsilon}{R_iI_cT_w} < max_{j \in F_i} \sum_{i \in I_j} \frac{\phi_i^{ss}(k)}{P_i^{ss}} < \frac{L_r}{R_iI_cT_w} \quad (20)$$

With $i = 1$, and $j = 1$ (i.e., single hop, single flow), we have

$$\frac{P_i^{ss}(L_r - \epsilon)}{R_iI_cT_w} < \varphi_i^{ss}(k) < \frac{P_i^{ss}L_r}{R_iI_cT_w} \quad (21)$$

With the obtained saturation nonlinearity approximate function of U_p above, we acquire (15) as the limit of a continuous function as follows.

$$\varphi_i(k) = (1-\psi)\varphi_i(k-1) + qTAS\gamma(U_p^i(k-1)) \quad (22)$$

where

$$\gamma(U_p^i(k-1)) = \frac{tan(3.(0.5 - U_p^i(k-1)))}{tan(1.5)} \quad (23)$$

To analyze the stability of the system around the steady point, we define the following parameters:

$$\begin{aligned} \vartheta T_s &= T_s - T_s^{ss} \\ \vartheta \phi_i &= \phi_i - \phi_i^{ss} \\ \vartheta \varphi_i &= \varphi_i - \varphi_i^{ss} \\ \vartheta P_i &= = P_i - P_i^{ss} \end{aligned}$$

and we rewrite (3), (1), and (22) as follows.

$$\begin{aligned} \vartheta T_s^i(k) = g_1(\vartheta T_s, \vartheta \phi_i, \vartheta \varphi_i) &= \vartheta T_s^i(k-1) \\ &- (\lambda + \theta)[\vartheta P_i(k-1) \\ &+ \frac{R_iT_wI_c\vartheta T_s^i(k-1)}{(T_w + T_s^{ss})(\vartheta T_s^i(k-1) + T_w + T_s^{i-ss})} \\ &+ (1-\psi)\lambda\vartheta\varphi_i(k-1) \\ &- \lambda\vartheta\phi_i(k-1) + \lambda qTAS\gamma(U_p^i(k-1)) - \psi\varphi_i^{ss} \end{aligned} \quad (24)$$

$$\begin{aligned} \vartheta \phi^i(k) = g_2(\vartheta T_s, \vartheta \phi_i) &= \vartheta\phi_i(k-1) + \vartheta P_i(k-1) \\ &+ \frac{R_iT_wI_c\vartheta T_s^i(k-1)}{(T_w + T_s^{ss})(\vartheta T_s^i(k-1) + T_w + T_s^{i-ss})} \end{aligned} \quad (25)$$

$$\begin{aligned} \vartheta \varphi^i(k) = g_3(\vartheta T_s, \vartheta \phi_i, \vartheta \varphi_i) &= (1-\psi)\vartheta\varphi_i(k-1) \\ &+ qTAS\gamma(U_p^i(k-1)) - \psi\varphi_i^{ss} \end{aligned} \quad (26)$$

We assume

$$y(k) = \begin{bmatrix} \vartheta T_s(k) \\ \vartheta \phi_i(k) \\ \vartheta \varphi_i(k) \end{bmatrix}$$

According to (24), (25), (26), this is a nonlinear system. We then find an approximation for the system based on linearization with respect to average stead points at (13), (21) and (9). We obtain the linear systems as follows.

$$\vartheta T_s^i(k) = \sigma_{11}\vartheta T_s^i(k-1) + \sigma_{12}\vartheta\phi_i(k-1) + \sigma_{13}\vartheta\varphi_i(k-1) \quad (27)$$

$$\vartheta \phi_i(k) = \sigma_{21}\vartheta T_s^i(k-1) + \sigma_{22}\vartheta\phi_i(k-1) + \sigma_{23}\vartheta\varphi_i(k-1) \quad (28)$$

$$\vartheta \varphi_i(k) = \sigma_{31}\vartheta T_s^i(k-1) + \sigma_{32}\vartheta\phi_i(k-1) + \sigma_{33}\vartheta\varphi_i(k-1) \quad (29)$$

We rewrite the linear systems (27), (28), and (29) in a vector form as follows.

$$y(k) = H.y(k-1) \tag{30}$$

With

$$H = \begin{bmatrix} \frac{\partial g_1}{\partial \vartheta T_s^i} & \frac{\partial g_1}{\partial \vartheta \phi_i} & \frac{\partial g_1}{\partial \vartheta \varphi_i} \\ \frac{\partial g_2}{\partial \vartheta T_s^i} & \frac{\partial g_2}{\partial \vartheta \phi_i} & \frac{\partial g_2}{\partial \vartheta \varphi_i} \\ \frac{\partial g_3}{\partial \vartheta T_s^i} & \frac{\partial g_3}{\partial \vartheta \phi_i} & \frac{\partial g_3}{\partial \vartheta \varphi_i} \end{bmatrix}_{y=0}$$
$$= \begin{bmatrix} 1 - \frac{P_i^{ss^2}(\lambda+\theta)}{RT_wI_c} & -\lambda & (1-\psi)\lambda \\ \frac{P_i^{ss^2}}{RT_wI_c} & 1 & 0 \\ 0 & 0 & 1-\psi \end{bmatrix}$$

Then we obtain the characteristic polynomial [51] of (30) as follows.

$$P_3(x) = x^3 + [\frac{(\lambda+\theta)P_i^{ss^2}}{RT_wI_c} + \psi - 3]x^2 + (\frac{[(\lambda+\theta)\psi - \lambda - 2\theta]P_i^{ss^2}}{RT_wI_c} + 3 - 2\psi)x + (\psi - 1)\frac{RT_wI_c - \theta P_i^{ss^2}}{RT_wI_c} \tag{31}$$

To enable the controller to run stably, the characteristic polynomial of (30) should be all zeros within a unit circle. From (30) and (31), the scheduling controller of node i is asymptotically stable if the two parameters, λ_i and θ_i, satisfy:

$$0 < \lambda + 2\theta < \frac{4RT_wI_c}{P_i^{ss^2}} \tag{32}$$

$$0 < \theta < \frac{RT_wI_c}{P_i^{ss^2}} \tag{33}$$

As the source of the linearized state equation achieves the stable state in small neighborhood of average steady state points, the trajectories of the nonlinear state equation will operate like a stable node. Therefore, the system stability can be achieved.

ACKNOWLEDGMENT

This paper was presented at the Proceedings of the 2017 IEEE International Conference on Communications [1].

REFERENCES

[1] T. Dinh and Y. Kim, "An efficient sensor-cloud interactive model for on-demand latency requirement guarantee," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Paris, France, May 2017, pp. 1–6.
[2] C. Zhu, V. C. M. Leung, K. Wang, L. T. Yang, and Y. Zhang, "Multi-method data delivery for green sensor-cloud," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 176–182, May 2017.
[3] A. G. Neiat, A. Bouguettaya, T. Sellis, and S. Mistry, "Crowdsourced coverage as a service: Two-level composition of sensor cloud services," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 7, pp. 1384–1397, Jul. 2017.
[4] S. Misra, S. Chatterjee, and M. S. Obaidat, "On theoretical modeling of sensor cloud: A paradigm shift from wireless sensor network," *IEEE Syst. J.*, vol. 11, no. 2, pp. 1084–1093, Jun. 2017.
[5] C. Zhu, X. Li, V. C. M. Leung, L. T. Yang, E. C.-H. Ngai, and L. Shu, "Towards pricing for sensor-cloud," *IEEE Trans. Cloud Comput.*, to be published. [Online]. Available: https://ieeexplore.ieee.org/document/7809146, doi: 10.1109/TCC.2017.2649525.
[6] A. Sen and S. Madria, "Risk assessment in a sensor cloud framework using attack graphs," *IEEE Trans. Serv. Comput.*, vol. 10, no. 6, pp. 942–955, Nov./Dec. 2017.
[7] S. Madria, V. Kumar, and R. Dalvi, "Sensor cloud: A cloud of virtual sensors," *IEEE Softw.*, vol. 31, no. 2, pp. 70–77, Mar. 2014.
[8] M. Fazio and A. Puliafito, "Cloud4sens: A cloud-based architecture for sensor controlling and monitoring," *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 41–47, Mar. 2015.
[9] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, "Cloud of things for sensing-as-a-service: Architecture, algorithms, and use case," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 1099–1112, Dec. 2016.
[10] I. L. Santos, L. Pirmez, F. C. Delicato, S. U. Khan, and A. Y. Zomaya, "Olympus: The cloud of sensors," *IEEE Cloud Comput.*, vol. 2, no. 2, pp. 48–56, Mar./Apr. 2015.
[11] T. Dinh and Y. Kim, "An efficient interactive model for on-demand sensing-as-a-services of sensor-cloud," *Sensors*, vol. 16, no. 7, pp. 1–18 2016.
[12] J. Barbarán, M. Díaz, and B. Rubio, "A virtual channel-based framework for the integration of wireless sensor networks in the cloud," in *Proc. Int. Conf. Future Internet Things Cloud (FiCloud)*, Aug. 2014, pp. 334–339.
[13] T. Ojha, S. Bera, S. Misra, and N. S. Raghuwanshi, "Dynamic duty scheduling for green sensor-cloud applications," in *Proc. IEEE 6th Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Dec. 2014, pp. 841–846.
[14] S. Chatterjee, R. Ladia, and S. Misra, "Dynamic optimal pricing for heterogeneous service-oriented architecture of sensor-cloud infrastructure," *IEEE Trans. Serv. Comput.*, vol. 10, no. 2, pp. 203–216, Mar. 2017.
[15] X. Wang, X. Wang, L. Liu, and G. Xing, "DutyCon: A dynamic duty-cycle control approach to end-to-end delay guarantees in wireless sensor networks," *ACM Trans. Sensor Netw.*, vol. 9, no. 4, pp. 42:1-42:33, Jul. 2013.
[16] T.-T. Huynh, T.-N. Tran, C.-H. Tran, and A.-V. Dinh-Duc, "Delay constraint energy-efficient routing based on Lagrange relaxation in wireless sensor networks," *IET Wireless Sensor Syst.*, vol. 7, no. 5, pp. 138–145, Oct. 2017.
[17] W. M. L. Franklin, J. D. Powell, and M. L. Workman, *Digital Control of Dynamic Systems*. Boston, MA, USA: Addison-Wesley, 1997.
[18] T. Dinh, Y. Kim, and H. Lee, "A location-based interactive model of Internet of Things and cloud (IoT-cloud) for mobile cloud computing applications," *Sensors*, vol. 17, no. 3, pp. 1–14, 2017.
[19] T. Dinh, Y. Kim, T. Gu, and A. V. Vasilakos, "L-MAC: A wake-up time self-learning MAC protocol for wireless sensor networks," *Comput. Netw.*, vol. 105, pp. 33–46, Aug. 2016.
[20] S. Misra, A. Singh, S. Chatterjee, and M. S. Obaidat, "Mils-cloud: A sensor-cloud-based architecture for the integration of military tri-services operations and decision making," *IEEE Syst. J.*, vol. 10, no. 2, pp. 628–636, Jun. 2016.
[21] S. Ghanavati, J. Abawajy, and D. Izadi, "An alternative sensor cloud architecture for vital signs monitoring," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2016, pp. 2827–2830.
[22] W. Lee, K. Nam, H.-G. Roh, and S.-H. Kim, "A gateway based fog computing architecture for wireless sensors and actuator networks," in *Proc. 18th Int. Conf. Adv. Commun. Technol. (ICACT)*, Jan./Feb. 2016, pp. 210–213.
[23] T. Dinh and Y. Kim, "A novel location-centric IoT-cloud based on-street car parking violation management system in smart cities," *Sensors*, vol. 16, no. 6, pp. 1–24, Jun. 2016.
[24] Y. Park, H. Yang, T. Dinh, and Y. Kim, "Design and implementation of a container-based virtual client architecture for interactive digital signage systems," *Int. J. Distrib. Sensor Netw.*, vol. 13, no. 7, pp. 1–13, 2017.
[25] G. Fortino, D. Parisi, V. Pirrone, and G. Di Fatta, "BodyCloud: A SaaS approach for community body sensor networks," *Future Gener. Comput. Syst.*, vol. 35, pp. 62–79, Jun. 2014.
[26] G. Fortino, G. Di Fatta, M. Pathan, and A. V. Vasilakos, "Cloud-assisted body area networks: State-of-the-art and future challenges," *Wireless Netw.*, vol. 20, no. 7, pp. 1925–1938, Oct. 2014.

[27] M. Hassanalieragh, A. Page, T. Soyata, G. Sharma, and M. Aktas, "Health monitoring and management using Internet-of-Things (IoT) sensing with cloud-based processing: Opportunities and challenges," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jul. 2015, pp. 285–292.

[28] L. Neto, J. Reis, R. Silva, and G. Gonçalves, "Sensor SelComp, a smart component for the industrial sensor cloud of the future," in *Proc. IEEE Int. Conf. Ind. Technol. (ICIT)*, Mar. 2017, pp. 1256–1261.

[29] Y. Lyu, F. Yan, Y. Chen, D. Wang, Y. Shi, and N. Agoulmine, "High-performance scheduling model for multisensor gateway of cloud sensor system-based smart-living," *Inf. Fusion*, vol. 21, pp. 42–56, Jan. 2015.

[30] A. Javed, H. Larijani, A. Ahmadinia, R. Emmanuel, M. Mannion, and D. Gibson, "Design and implementation of a cloud enabled random neural network-based decentralized smart controller with intelligent sensor nodes for HVAC," *IEEE Internet Things J.*, vol. 4, no. 2, pp. 393–403, Apr. 2017.

[31] A. Sen, V. P. Modekurthy, R. Dalvi, and S. Madria, "A sensor cloud test-bed for multi-model and multi-user sensor applications," in *Proc. IEEE Wireless Commun. Netw. Conf.*, Apr. 2016, pp. 1–7.

[32] A. Gupta and N. Mukherjee, "Implementation of virtual sensors for building a sensor-cloud environment," in *Proc. 8th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2016, pp. 1–8.

[33] *Network Functions Virtualisation (NFV); Architectural Framework*, document ETSI GS NFV 002 V1.1.1., ETSI, 2013.

[34] V. Anh-Vu, T. Dinh, and Y. Kim, "Modeling of service function chaining in network function virtualization environment," in *Proc. KICS Summer Conf.* Jeju, South Korea: KICS, 2016, pp. 1–2.

[35] I. Al-Anbagi, M. Erol-Kantarci, and H. T. Mouftah, "A survey on cross-layer quality-of-service approaches in WSNs for delay and reliability-aware applications," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 525–552, 1st Quart., 2016.

[36] T. Dinh and T. Gu, "A novel metric for opportunistic routing in heterogenous duty-cycled wireless sensor networks," in *Proc. IEEE 23rd Int. Conf. Netw. Protocols (ICNP)*, Nov. 2015, pp. 224–234.

[37] S. Wu, J. Niu, W. Chou, and M. Guizani, "Delay-aware energy optimization for flooding in duty-cycled wireless sensor networks," *IEEE Trans. Wireless Commun.*, vol. 15, no. 12, pp. 8449–8462, Dec. 2016.

[38] T.-N. Dao, S. Yoon, and J. Kim, "A deadline-aware scheduling and forwarding scheme in wireless sensor networks," *Sensors*, vol. 16, no. 1, pp. 1–12, 2016.

[39] H. Byun and J. Yu, "Adaptive duty cycle control with queue management in wireless sensor networks," *IEEE Trans. Mobile Comput.*, vol. 12, no. 6, pp. 1214–1224, Jun. 2013.

[40] M. Xu and H. Leung, "A joint fusion, power allocation and delay optimization approach for wireless sensor networks," *IEEE Sensors J.*, vol. 11, no. 3, pp. 737–744, Mar. 2011.

[41] Y. Sun, C. Chen, and H. Luo, "Adaptive scheduling and routing scheme for delay guarantee in wireless sensor networks," *Int. J. Distrib. Sensor Netw.*, vol. 9, no. 8, pp. 1–14, 2013.

[42] C. Shanti and A. Sahoo, "DGRAM: A delay guaranteed routing and MAC protocol for wireless sensor networks," *IEEE Commun. Mobile Comput.*, vol. 9, no. 10, pp. 1407–1423, Oct. 2010.

[43] L. Chang, B. Zhang, L. Cui, Q. Li, and Z. Miao, "Tree-based delay guaranteed and energy efficient MAC protocol for wireless sensor networks," in *Proc. Int. Conf. Ind. Control Electron. Eng.*, Aug. 2012, pp. 893–897.

[44] L. Cheng, J. Niu, Y. Gu, T. He, and Q. Zhang, "Energy-efficient statistical delay guarantee for duty-cycled wireless sensor networks," in *Proc. 12th Annu. IEEE Int. Conf. Sens., Commun., Netw. (SECON)*, Jun. 2015, pp. 46–54.

[45] I.-H. Hou, "Providing end-to-end delay guarantees for multi-hop wireless sensor networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2013, pp. 414–419.

[46] T. Dinh, Y. Kim, T. Gu, and A. V. Vasilakos, "An adaptive low-power listening protocol for wireless sensor networks in noisy environments," *IEEE Syst. J.*, vol. 12, no. 3, pp. 2162–2173, Sep. 2018.

[47] J. Wang, Z. Cao, X. Mao, X.-Y. Li, and Y. Liu, "Towards energy efficient duty-cycled networks: Analysis, implications and improvement," *IEEE Trans. Comput.*, vol. 65, no. 1, pp. 270–280, Jan. 2016.

[48] S. Madden. *Intel Berkeley Research Lab Sensor Networks*. Accessed: Nov. 1. [Online]. Available: http://db.csail.mit.edu/labdata/labdata.html

[49] Y. Park, N.-T. Dinh, and Y. Kim, "A network monitoring system in 6LoWPAN networks," in *Proc. 4th Int. Conf. Commun. Electron. (ICCE)*, Aug. 2012, pp. 69–73.

[50] OpenStack. *OpenSource*. Accessed: Nov. 1. [Online]. Available:https://www.openstack.org/

[51] A. Haldar and S. Mahadevan, *Probability, Reliability, and Statistical Methods in Engineering Design*. New York, NY, USA: Wiley, 2000.

[52] T. u. S. Janson and A. Rucinski, *Random Graphs* (Discrete Mathematics and Optimization). New York, NY, USA: Wiley, 2000.

[53] E. T. Whittaker and G. N. Watson, *A Course of Modern Analysis*, 4th ed. Cambridge, U.K.: Cambridge Univ. Press, 1963.

[54] K. H. Shim and J. T. Lim, "Extreme-point robust stability of a class of discrete-time polynomials," *Electron. Lett.*, vol. 32, no. 15, pp. 1421–1422, Jul. 1996.

[55] J.-T. Lim and G.-H. Shim, "Asymptotic performance evaluation of token-passing networks," *IEEE Trans. Image Process.*, vol. 40, no. 3, pp. 384–385, Jun. 1993.

NGOC-THANH DINH received the M.Sc. and Ph.D. degrees in electronic and telecommunication from Soongsil University. He was a Ph.D. student with the School of Computer Science and IT, Royal Melbourne Institute of Technology University. He is currently an Assistant Professor with the Department of IT Convergence, Soongsil University. His current research interests include the Internet of Things and cloud computing, 5G networking, and next-generation networks. His publications appear in top journals, such as the IEEE Internet of Things, the IEEE Transactions on Big Data, and the IEEE Systems Journal, and top conferences, such as the IEEE ICNP and ICC. He serves as a TPC member for many leading conferences and members for journals, such as the IEEE Transactions on Mobile Computing, the IEEE Sensors, and *Computer Networks*.

YOUNGHAN KIM received the B.S. degree from Seoul National University and the M.Sc. and Ph.D. degrees in electrical engineering from KAIST. He is currently a Full Professor with the Department of IT Convergence, Soongsil University. He is also an Executive Director of the Korea Information and Communications Society and the President of the Open Standards and Internet Association.