Efficient Scheduling, Caching, and Merging of Notifications to Save Message Costs in IoT Networks Using CoAP

Wei-Kuang Lai, You-Chiun Wang, and Sih-Yu Lin

Abstract—IoT devices are widely deployed and many of them have limited capabilities. CoAP (constrained application protocol) is developed to integrate such devices into the web environment. Specifically, client devices register their interested data with IoT devices, which will keep notifying them of the status of sensing data. However, client devices have different demands in terms of minimum and maximum periods to receive notifications, whereas IoT devices usually offer notifications at regular intervals. Some IoT devices would send many more notifications than necessary, which wastes bandwidth and energy. To conquer this problem, the paper proposes a *group-based message management (GMM) framework* for CoAP proxies to efficiently coordinate the sending of notifications from IoT devices and forward them to client devices. GMM curtails superfluous notifications by three modules. The scheduling module groups client devices based on their demands and finds an optimal observation period for each group. The caching module adjusts the Max-Age value to allow a proxy reusing its cached notifications to answer requests. The merging module combines the notifications originated from different IoT devices queried by the same client device, so as to save bandwidth. Simulation results show that GMM not only reduces unnecessary notifications but also conserves the energy of devices.

Index Terms—Caching, CoAP, merging, proxy, scheduling.

1 Introduction

Thanks to the rapid development of wireless technology and microelectronic systems, the Internet of Things (IoT) is ushering in a new era. The aim of IoT is to enable various devices such as appliances, vehicles, and electronic products to form a network for sensing their surroundings and transferring information to one another, which is carried out by embedding wireless sensors in these devices [1].

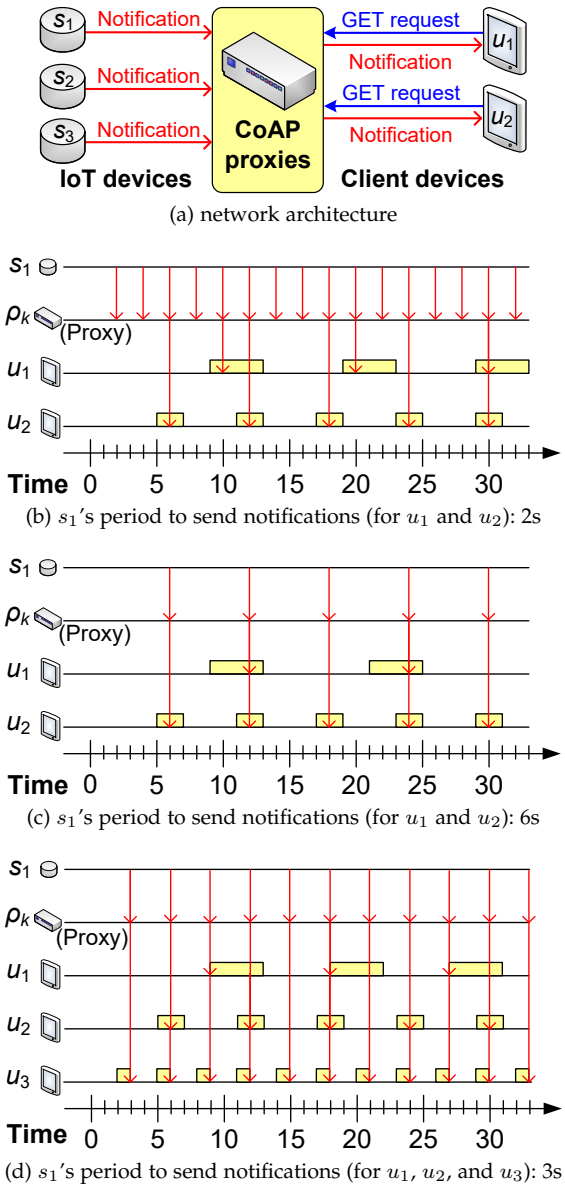
Due to the nature of sensors, many IoT devices are regarded as *constrained devices*, which means that they possess limited computing power, memory storage, and energy [2]. In contrast to powerful devices like laptops and mobile phones, the popular hypertext transfer protocol (HTTP) would be complicated and not efficient for constrained devices. Thus, the *constrained application protocol (CoAP)* is proposed as a good substitute for HTTP to help constrained devices handily access the web and communicate with other devices that use HTTP [3].

CoAP uses a client-server architecture and offers a request-response interaction model between two application endpoints. Specifically, a client device (e.g., a mobile phone) first registers with an IoT device (i.e., a server) the data that it has an interest. The IoT device then keeps sending *notifications* to the client device for updating the status of sensing data, until the client device does not require the data anymore (e.g., by canceling its registration). To facilitate this procedure, proxies are placed in the middle of IoT devices and client devices to relay messages between them, as shown in Fig. 1(a). Moreover, the extension of CoAP [4] allows client devices specifying their demands (in minimum and maximum periods, denoted by $p_j^{\min}$ and $p_j^{\max}$, respectively) to get notifications for adding flexibility.

When multiple client devices request data from one IoT device, a proxy can establish one single observation relationship with that IoT device on behalf of these client devices. In this case, the proxy can ask the IoT device to transmit notifications in a fixed period (called the *observation period*), which should satisfy the demand of every client device. Let us consider two examples in Fig. 1(b) and (c), where two client devices u_1 and u_2 request data from an IoT device s_1. Suppose that u_1 has a demand of $p_1^{\min} = 9$ and $p_1^{\max} = 13$, and u_2 has a demand of $p_2^{\min} = 5$ and $p_2^{\max} = 7$. Without caching notifications, the observation period may be set to 2 or 6 seconds, so the proxy can always forward s_1's notifications to u_1 and u_2 and satisfy their demands, as shown in Fig. 1(b) and (c), respectively. In fact, Fig. 1(c) gives a better solution, since s_1 can send fewer notifications and save its energy accordingly.

As sending and receiving messages are both energy-costly operations [5], it is critical to reduce the number of messages exchanged among devices for energy conservation and congestion avoidance in IoT networks. In view of this, the paper proposes a *group-based message management (GMM) framework* by using proxies to regulate the transmission of notifications in a CoAP-based IoT network, which consists of three modules. Specifically, the *scheduling module* divides client devices into groups and finds an optimal observation period for each group, such that the IoT device generates the minimum notifications and the number of notifications relayed to client devices is also reduced. To help a proxy better reuse its cached notifications for answering client devices, the *caching module* computes an adequate Max-Age value for each IoT device. When a client device queries multiple IoT devices, the *merging module* will combine their notifications efficiently. Thus, the client device need not receive duplicate information such

The authors are with the Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung City, 80424, Taiwan. Email: wklai@cse.nsysu.edu.tw; ycwang@cse.nsysu.edu.tw; m033040038@student.nsysu.edu.tw.



(a) network architecture

(b) s_1's period to send notifications (for u_1 and u_2): 2s

(c) s_1's period to send notifications (for u_1 and u_2): 6s

(d) s_1's period to send notifications (for u_1, u_2, and u_3): 3s

Fig. 1: Examples of scheduling the transmission of notifications in a CoAP-based IoT network.

as identical packet headers, thereby conserving bandwidth. Through simulations, we validate that GMM can slash the number of notifications sent in the network. Our contribution is to develop an efficient framework to save the message cost for IoT networks using CoAP, which economizes on the energy of constrained devices and mitigates network congestion.

This paper is organized as follows: Section 2 briefly introduces CoAP. Section 3 surveys related work and Section 4 gives the system model. Then, the GMM framework is proposed in Section 5, followed by the discussion of its issues in Section 6. After that, the performance evaluation is presented in Section 7. Lastly, Section 8 concludes this paper.

2 Overview of CoAP

CoAP is a specialized web transfer protocol used to support machine-to-machine (M2M) communications in IoT networks. It is built on the *representational state transfer (REST) model* so as to be HTTP-compatible [6]. More concretely, IoT devices make sensing data available through uniform resource locators

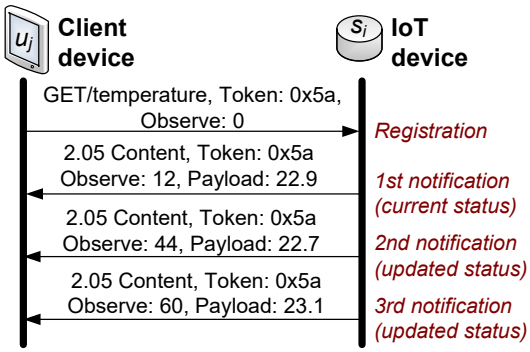


Fig. 2: An example of the observing mechanism.

(URLs). Then, client devices access these data by using HTTP-like messages (e.g., GET, PUT, POST, DELETE, and so on). In CoAP, the *observing mechanism* makes IoT devices obediently update client devices on the status of sensing data. Moreover, CoAP provides *proxy* and *cache* mechanisms to add flexibility to message transmissions. Below, we discuss each mechanism.

2.1 Observing Mechanism

This mechanism is proposed in [7] as an extension of CoAP to allow client devices monitoring the changes in sensing data. A client device u_j can get a representation of data and keep this representation updated by an IoT device s_i over a period of time. Specifically, u_j registers its interested data with s_i by sending a GET request with an *observe flag*. Then, s_i replies a notification regarding the current status of sensing data. After that, whenever something changes in the sensing data, s_i will send a notification with the updated status to u_j.

Fig. 2 shows an example, where u_j queries s_i about temperatures. Specifically, the token helps u_j associate its registration with all notifications replied by s_i. The code of "2.05 Content" is similar to "HTTP 200 OK". The "Observe" field indicates the sequence of messages. When s_i receives the registration, it replies a notification with the current temperature (given in the payload) to u_j. After that, subsequent notifications will be transmitted by s_i whenever the temperature changes. In case that u_j no longer has an interest in the sensing data, u_j can send an RST message to s_i. Then, s_i removes u_j from the list of observers and stops sending notifications to u_j.

2.2 Proxy Mechanism

In CoAP, proxies are used to mediate between client devices and IoT devices, as shown in Fig. 1(a). To do so, a client device indicates the appointed proxy in the GET request. Suppose that a client device u_j appoints a proxy ρ_k as its deputy, whose IP address and port are "140.117.169.65" and 5683, respectively. Then, u_j sends a GET request to ρ_k by adding an instruction of "Proxy-URI: coaps://140.117.169.65:5683/S1," where URI means *uniform resource identifier*. After that, ρ_k performs the registration with the queried IoT device on behalf of u_j. Thus, the IoT device sends notifications to ρ_k for updating the status of sensing data, which will be further relayed to u_j.

Using proxies brings two benefits [8]. First, client devices simply obtain sensing data from a proxy without knowing the sensitive information of IoT devices (e.g., their identifications). Since IoT devices are *hidden* by the proxy, malicious devices are thus hard to attack IoT devices. Second, many IoT devices are resource-constrained, so each of them is capable of serving

very few client devices. This problem can be solved by using a proxy to establish one single observation relationship between many client devices and an IoT device. In view of this, we will exploit proxies to save the message cost in an IoT network and improve its performance.

2.3 Cache Mechanism

When a proxy ρ_k gets a new notification from an IoT device s_i, it will be stored in ρ_k's cache. In this way, ρ_k can *reuse* s_i's prior notifications to satisfy current requests (if feasible), so as to reduce the response time and save s_i's energy. There are two models proposed for the cache mechanism in CoAP [3]: *freshness* and *validation*.

In the freshness model, when a cached notification has not expired yet, ρ_k is allowed to use it for answering subsequent requests without contacting s_i. To do so, s_i can negotiate with ρ_k about the expiration time of its sensing data (in seconds) via the *Max-Age option*. Once the age of a notification cached by ρ_k is greater than that specified in the Max-Age option, it is considered to be not fresh. In case that the Max-Age option is not specified, the default value is set to 60 seconds. Therefore, if s_i prefers not to use the cache mechanism, it must indicate a zero-value Max-Age option.

For the validation model, if ρ_k has some cached notifications that fit for a request but cannot use any of them (e.g., they are not fresh), ρ_k gives s_i an opportunity to update its freshness for these cached notifications. In particular, ρ_k adds an *ETag option* in the request that specifies the entity-tag of the cached notification. After getting a valid response from s_i, ρ_k updates the cached notification with the new Max-Age option indicated in the response.

Our work also aims to adaptively adjust the value of Max-Age for the freshness model, so as to efficiently diminish the message overhead incurred by IoT devices.

3 Related Work

Since CoAP is one of popular data transfer protocols for IoT networks, a variety of relevant topics are addressed. Choi and Koh [9] develop two schemes to support mobility management for CoAP with the help of Proxy Mobile IPv6. In [10], a CoAP extension is proposed to support the context-aware discovery service of smart objects (e.g., TVs and appliances). The study [11] designs a rate-based congestion control method for CoAP, so as to improve fairness and also reduce data retransmissions. The work [12] evaluates the performance of video streaming applications over CoAP in IoT networks. As can be seen, none of them considers leveraging the proxy mechanism in CoAP.

Some studies analyze the efficiency of using CoAP proxies. The work [13] evaluates the effect of arrival rates of requests and the number of IoT devices on the cache mechanism (e.g., the hit ratio and response time). With random inter-observation time, [14] measures both data lifetime and round-trip time at a proxy by kernel estimation of probability density distribution. How to use proxies to offer various services is also discussed. The study [15] facilitates the bootstrapping process by using a proxy, which helps smart objects join an IoT network more securely. In [16], a cross-protocol proxy is developed to broker among HTTP, CoAP, and MQTT (message queuing telemetry transport). Obviously, they discuss different issues from ours.

A few studies consider scheduling messages in CoAP-based IoT networks. In [17], client devices express the priority with which they want to be notified. Notifications are then classified according to their expressions. High-priority notifications (e.g., urgent ones) will be given precedence to be transmitted first. Mingozzi et al. [8] propose a proxy virtualization framework to provide scalability in a large IoT network, which is built on a Linux container. The framework also differentiates between notifications based on their priorities for QoS consideration. However, neither [17] nor [8] permits client devices to specify their preferred periods for getting notifications.

TABLE 1: Summary of notations.

notation	definition
$\hat{\mathcal{U}}_i^k$	set of client devices which query IoT device s_i via proxy ρ_k
$\hat{\mathcal{S}}_j^k$	set of IoT devices that client device u_j queries via proxy ρ_k
$p_j^{\min}$	u_j's minimum period to get notifications
$p_j^{\max}$	u_j's maximum period to get notifications
$\hat{P}_{i,a}$	s_i's observation period for a group $\hat{\mathcal{G}}_a \subseteq \hat{\mathcal{U}}_i^k$
T_i	data generating interval of s_i
$\tilde{A}_i$	Max-Age value assigned to s_i

The work [18] takes account of fairness on message transmissions, which asks a proxy to schedule the registrations of client devices to make their requests possess equal conditions on timeliness. Thus, IoT devices could have similar depletion degrees of energy. Considering that there are multiple proxies in a CoAP-based IoT network, [19] selects the best proxy for each request such that the amount of energy spent on the steps of registration and notification can be minimized. Afterward, a heuristic algorithm is proposed to decrease the average hop count between client devices and IoT devices. Apparently, both [18] and [19] aim at different objectives from this paper.

Given a set of client devices that request data from one IoT device, [20] uses a proxy to build an observation relationship with the IoT device and receive its periodic notifications (about sensing data) on behalf of client devices. The proxy then relays notifications to client devices based on their demands, which are specified in terms of minimum and maximum periods. The objective is to find the optimal observation period to minimize the notifications sent by the IoT device, such that the demand of each client device is satisfied. However, [20] does not adopt the cache mechanism to reduce the message overhead. If some requests have short periods, the IoT device has to send many notifications, thereby consuming more energy.

As compared with the prior work, our GMM framework is not only the first to schedule the transmission of notifications for client devices in a group-based manner for more flexibility, but also exploits the cache mechanism in CoAP by adjusting the Max-Age option. Hence, GMM prevents IoT devices from sending unnecessary notifications and also reduces the number of notifications forwarded to client devices. Moreover, when a client device queries multiple IoT devices, their notifications can be efficiently merged to save bandwidth and energy. These designs distinguish GMM from existing solutions and greatly decrease the message cost in an IoT network.

4 System Model

We are given an IoT network which comprises IoT devices, client devices, and proxies, as shown in Fig. 1(a). Every IoT device offers sensing data, in which some client devices have an interest. A client device can also request multiple IoT devices for their data. A number of proxies sit between IoT devices and client devices for managing message transmissions. They can

cache and reuse previous notifications to answer some requests of client devices without overly disturbing IoT devices, as discussed in Section 2.3. Recall that in Section 2.2, each client device needs to specify its appointed proxy in the GET request. Therefore, let us denote by $\hat{\mathcal{U}}_i^k$ the set of client devices that request data from an IoT device s_i via a proxy ρ_k. Besides, we also denote by $\hat{\mathcal{S}}_j^k$ the set of IoT devices that a client device u_j asks for their data through ρ_k.

According to the implementation guidance for CoAP [21], IoT devices can provide *periodic* resources (i.e., sensing data). After ρ_k establishes the observation relationship between one IoT device s_i and the client devices in $\hat{\mathcal{U}}_i^k$, s_i keeps sending notifications in a constant period. By the means of *dynamic resource linking* [4], client devices can specify their demands to obtain notifications. To do so, before a client device $u_j \in \hat{\mathcal{U}}_i^k$ performs registration, u_j sends to ρ_k a PUT message with two parameters $p_j^{\min}$ and $p_j^{\max}$, which indicate the minimum and maximum intervals between two successive notifications that u_j wants to get, respectively. Both $p_j^{\min}$ and $p_j^{\max}$ are measured in seconds. Fig. 1 presents some examples.

For each IoT device s_i, given the set $\hat{\mathcal{U}}_i^k$ of client devices that query it for data through proxy ρ_k and their $(p_j^{\min}, p_j^{\max})$ demands, our problem asks how to determine the observation period for s_i and also the forwarding of notifications to client devices, such that the overall *message cost* in the IoT network is minimized. More concretely, the message cost is defined to be the summation of notifications passed through each proxy, which include those sent from IoT devices and those relayed to client devices. A lower message cost means that IoT devices save more energy on sending notifications and client devices also reduce more energy consumption on getting notifications. Table 1 summarizes the notations used in this paper.

5 The Proposed GMM Framework

When a client device u_j submits its request, the responsible proxy, say, ρ_k first *screens* the request to filter out improper ones, for example, requesting some type of sensing data not supported by any IoT device or unreasonable demands (e.g., $p_j^{\min} > p_j^{\max}$ or $p_j^{\max} \leq 0$). If the request is passed by the screening process, it will be forwarded to the corresponding IoT device, say s_i. Afterward, s_i sends a notification to ρ_k if it accepts u_j's request, as discussed in Section 2.1. In case that u_j's request is screened out or rejected by s_i, ρ_k replies a rejection message (i.e., 4.00 Bad Request) to s_i. Then, ρ_k removes u_j from the set $\hat{\mathcal{U}}_i^k$.

After that, ρ_k calculates not only the length of observation period (denoted by $\tilde{P}_i$) but also the value of Max-Age option (denoted by $\tilde{A}_i$) for each queried IoT device s_i, and notifies s_i accordingly. To consider the demands of client devices as well as the freshness of sensing data, s_i should send its notifications to ρ_k in an interval of $\max\{\tilde{P}_i, \tilde{A}_i\}$. Whenever ρ_k receives a notification from s_i, ρ_k keeps a backup of the notification in its cache and replaces the old one if necessary. Then, for each client device u_j in $\hat{\mathcal{U}}_i^k$, ρ_k will forward s_i's notification[1] to u_j when $\tilde{P}_i$ is due and u_j's minimum period (i.e., $p_j^{\min}$) has already elapsed since the last received notification.

To efficiently save the message cost, each proxy ρ_k applies the GMM framework to carry out the above work, which is made up of three modules. Based on the $(p_j^{\min}, p_j^{\max})$ demands of client devices in $\hat{\mathcal{U}}_i^k$, the *scheduling module* clusters them into groups and computes a maximum observation period for each group, such that IoT device s_i can send the minimum notifications and the number of notifications relayed to client devices in $\hat{\mathcal{U}}_i^k$ is also reduced. Then, the *caching module* finds an adequate Max-Age value for each IoT device, so as to help a proxy exploit its cached notifications for answering client devices. If a client device u_j queries a set $\hat{\mathcal{S}}_j^k$ of IoT devices, the *merging module* well combines the notifications originated from $\hat{\mathcal{S}}_j^k$ into one single notification. In this way, the amount of data sent to u_j could decrease. Below, we elaborate on each module, followed by the design rationale of GMM.

1. The notification will be the cached one when $\tilde{P}_i > \tilde{A}_i$.

Algorithm 1: Observation Period Selection (OPS)

Data: Demand $(p_j^{\min}, p_j^{\max})$ of each u_j in $\hat{\mathcal{U}}_i^k$
Result: Observation period $\tilde{P}_i$ for IoT device s_i
1 $\tilde{P}_i \leftarrow \min_{\forall u_j \in \hat{\mathcal{U}}_i^k}\{p_j^{\max}\}$;
2 **while** $\exists u_x \in \hat{\mathcal{U}}_i^k$ *such that* $\lceil p_x^{\min}/\tilde{P}_i \rceil \tilde{P}_i > p_x^{\max}$ **do**
3 $\quad \tilde{P}_i \leftarrow \min_{\forall u_j \in \hat{\mathcal{U}}_i^k} \left\{ \left\lfloor \frac{p_j^{\max}}{\lceil p_j^{\min}/\tilde{P}_i \rceil} \right\rfloor \right\}$;

5.1 The Scheduling Module

In [20], an *observation period selection (OPS)* algorithm is proposed to minimize the number of notifications replied from an IoT device s_i based on demands of all client devices in $\hat{\mathcal{U}}_i^k$. Its objective is to find the maximum observation period $\tilde{P}_i$ for s_i such that the following condition is fulfilled:

$$p_j^{\max} \geq p_j^{\min} + \tilde{P}_i, \ \forall u_j \in \hat{\mathcal{U}}_i^k. \tag{1}$$

In other words, after $p_j^{\min}$ expires, s_i should send at least one notification to proxy ρ_k before $p_j^{\max}$ is due, so ρ_k can forward a *timely* notification to each client device $u_j \in \hat{\mathcal{U}}_i^k$ and satisfy its demand. By doing some algebra, we can derive that

$$p_j^{\max} \geq (p_j^{\min}/\tilde{P}_i + 1)\tilde{P}_i \geq \lceil p_j^{\min}/\tilde{P}_i \rceil \tilde{P}_i, \ \forall u_j \in \hat{\mathcal{U}}_i^k \tag{2}$$

$$\Rightarrow \tilde{P}_i \leq \frac{p_j^{\max}}{\lceil p_j^{\min}/\tilde{P}_i \rceil}, \ \forall u_j \in \hat{\mathcal{U}}_i^k \tag{3}$$

Algo. 1 presents the pseudocode of OPS. Because the solution to satisfy Eq. (1) cannot be larger than the maximum demand of any u_j in $\hat{\mathcal{U}}_i^k$, the initial value of $\tilde{P}_i$ is set to the minimum $p_j^{\max}$ value (i.e., the upper bound on the solution) in line 1. Then, the while-loop checks if there exist some client devices in $\hat{\mathcal{U}}_i^k$ which violate the condition of Eq. (2). If so, we update the value of $\tilde{P}_i$ based on Eq. (3), as shown in line 3.

Let us consider the example in Fig. 1(c). By line 1, $\tilde{P}_1$ is set to $\min\{13, 7\} = 7$. Since $\lceil 9/7 \rceil \times 7 = 14 > p_2^{\max} = 13$, $\tilde{P}_1$ is updated to $\min\{\lfloor 13/(\lceil 9/7 \rceil) \rfloor, \lfloor 7/(\lceil 5/7 \rceil) \rfloor\} = 6$ by line 3. Thus, the solution is 6 seconds. Theorem 1 proves that OPS can minimize the number of notifications that s_i sends to ρ_k.

Theorem 1. *OPS in Algo. 1 finds an optimal observation period to let IoT device s_i send the minimum notifications to proxy ρ_k such that ρ_k can always forward a timely notification to each client device in $\hat{\mathcal{U}}_i^k$ to meet its demand.*

Proof: Please refer to Theorem 1 in [20]. □

Although OPS minimizes the number of notifications sent by an IoT device, it may not reduce the number of notifications that client devices have to receive. Two examples are given in Fig. 1(c) and (d), where the total time is 45 seconds. Observing

Algorithm 2: The Scheduling Module

Data: Demand $(p_j^{\min}, p_j^{\max})$ of each u_j in $\hat{\mathcal{U}}_i^k$
Result: Disjoint groups $\hat{\mathcal{G}}_1, \hat{\mathcal{G}}_2, \cdots, \hat{\mathcal{G}}_m$ and their observation periods $\tilde{P}_{i,1}, \tilde{P}_{i,2}, \cdots, \tilde{P}_{i,m}$, where $\hat{\mathcal{G}}_1 \cup \hat{\mathcal{G}}_2 \cup \cdots \cup \hat{\mathcal{G}}_m = \hat{\mathcal{U}}_i^k$

1 Sort all devices in $\hat{\mathcal{U}}_i^k$ increasingly by the $p_j^{\max}$ value;
2 **foreach** $u_j \in \hat{\mathcal{U}}_i^k$ **do**
3 Create a new group $\hat{\mathcal{G}}_a = \{u_j\}$;
4 $\hat{\mathcal{U}}_i^k \leftarrow \hat{\mathcal{U}}_i^k \setminus \{u_j\}$;
5 **foreach** $u_x \in \hat{\mathcal{U}}_i^k$ **do**
6 $\hat{\mathcal{G}}_{\text{tmp}} \leftarrow \hat{\mathcal{G}}_a \cup \{u_x\}$ and $\tilde{P}_{\text{tmp}} \leftarrow \text{OPS}(\hat{\mathcal{G}}_{\text{tmp}})$;
7 canAdd $\leftarrow$ true;
8 **while** $\exists u_y \in \hat{\mathcal{G}}_{\text{tmp}}$ *such that* $\text{RCF}(u_y, \tilde{P}_{\text{tmp}}) =$ *false* **do**
9 canAdd $\leftarrow$ false;
10 break;
11 **if** *canAdd = true* **then**
12 $\hat{\mathcal{G}}_a \leftarrow \hat{\mathcal{G}}_a \cup \{u_x\}$ and $\hat{\mathcal{U}}_i^k \leftarrow \hat{\mathcal{U}}_i^k \setminus \{u_x\}$;
13 $\tilde{P}_{i,a} \leftarrow \text{OPS}(\hat{\mathcal{G}}_a)$;

from Fig. 1(c), u_1 gets notifications in the 12th, 24th, and 36th seconds. Suppose that a client device u_3 is added to $\hat{\mathcal{U}}_1^k$, whose demand is $(p_3^{\min}, p_3^{\max}) = (2, 3)$. By Algo. 1, the new value of $\tilde{P}_1$ is 3. Thus, proxy ρ_k sends notifications to u_1 in the 9th, 18th, 27th, 36th, and 45th seconds (since $p_1^{\min} = 9$), as shown in Fig. 1(d). Evidently, u_1 is compelled to receive more notifications when u_3 is added, even though its demand does not change.

To address this issue, our scheduling module partitions $\hat{\mathcal{U}}_i^k$ into disjointed groups. In this way, long-period client devices (e.g., u_1) need not humor short-period ones (e.g., u_3) and will not receive too many notifications due to following the same observation period. For convenience, we define a *notification accepted interval (NAI)* of client device u_j as the time interval since u_j begins its $p_j^{\min}$ period (to get notifications) until the $p_j^{\max}$ period is due. Fig. 1(b) presents an example, where u_1 has three NAIs: 9th–13th, 19th–23rd, and 29th–33rd seconds. Note that NAIs of a client device may not be fixed but depend on the observation period. This argument can be validated by three examples in Fig. 1(b)–(d), where u_1 has different NAIs when the observation period changes. Then, the aim of our scheduling module is to find out as few groups of client devices in $\hat{\mathcal{U}}_i^k$ as possible (along with their observation periods), such that for each group $\hat{\mathcal{G}}_a \subseteq \hat{\mathcal{U}}_i^k$, IoT device s_i will send exactly one notification during every NAI of each client device in $\hat{\mathcal{G}}_a$ based on the observation period $\tilde{P}_{i,a}$. Fig. 1(c) shows an example, where s_1 sends just one notification within each NAI of both u_1 and u_2 when the observation period is set to 6. Thus, u_1 and u_2 can be put in the same group with $\tilde{P}_{i,a} = 6$.

Algo. 2 gives the pseudocode of the scheduling module. Observing from OPS, the observation period $\tilde{P}_i$ will be bound by the smallest $p_j^{\max}$ value (referring to line 1 of Algo. 1). In view of this, all client devices in $\hat{\mathcal{U}}_i^k$ are sorted in ascending order of their $p_j^{\max}$ values. After that, we create a new group $\hat{\mathcal{G}}_a$ and move the first client device (which has the minimum $p_j^{\max}$ value) from $\hat{\mathcal{U}}_i^k$ to $\hat{\mathcal{G}}_a$, as shown in lines 3 and 4. When $\hat{\mathcal{U}}_i^k$ is not

Algorithm 3: Redundancy Checking Function (RCF)

Data: Client device u_x and observation period $\tilde{P}$
Result: True (i.e., passed) or false (i.e., not passed)

1 **if** $\texttt{MOD}(p_x^{\min}, \tilde{P}) = 0$ **then**
2 **if** $\tilde{P} > p_x^{\max} - p_x^{\min}$ **then**
3 **return** true;
4 **else**
5 **if** $\tilde{P} > (p_x^{\max} - p_x^{\min} + \texttt{MOD}(p_x^{\min}, \tilde{P}))/2$ **then**
6 **return** true;
7 **return** false;

empty, we iteratively check if each client device u_x in $\hat{\mathcal{U}}_i^k$ can be added to $\hat{\mathcal{G}}_a$, on the premise that the objective of the scheduling module is still satisfied. The code is presented in lines 5–12. Specifically, we compute the observation period (denoted by $\tilde{P}_{\text{tmp}}$) for a temporary group $\hat{\mathcal{G}}_{\text{tmp}} = \hat{\mathcal{G}}_a \cup \{u_x\}$ by OPS, and check if there will be no more than one notification sent from s_i during each NAI of every client device in $\hat{\mathcal{G}}_{\text{tmp}}$. This can be done by using the *redundancy checking function (RCF)*. If the above check is passed (i.e., the Boolean variable "canAdd" is still true), it is feasible to add u_x to $\hat{\mathcal{G}}_a$. After adding all possible client devices to group $\hat{\mathcal{G}}_a$, we decide its observation period $\tilde{P}_{i,a}$ by OPS in line 13.

Algo. 3 then exhibits the pseudocode of RCF. In line 1, $\texttt{MOD}(p_x^{\min}, \tilde{P})$ takes the remainder after dividing $p_x^{\min}$ by $\tilde{P}$. Theorem 2 proves the correctness of Algo. 3 (and also gives the rationale of RCF).

Theorem 2. *If RCF in Algo. 3 returns true, IoT device s_i must send exact one notification during each NAI of client device u_x based on the observation period $\tilde{P}$.*

Proof: Algo. 3 considers two cases: $p_x^{\min}$ is dividable by $\tilde{P}$ (i.e., line 1) and otherwise (i.e., line 4). Suppose that u_x gets the last notification (denoted by N_{LR}) at time t, which is within the last NAI. Thus, the current NAI of u_x starts at time $t + p_x^{\min}$ and ends at time $t + p_x^{\max}$. In Fig. 3, we also denote by N_{CR}, N_{LS}, and N_{NS} the notification received by u_x in the current NAI, the last notification sent by IoT device s_i before the current NAI, and the next notification sent by s_i after the current NAI, respectively.

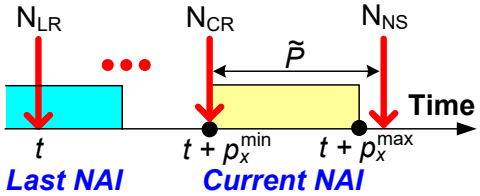
Fig. 3(a) shows the first case, where N_{CR} must be received by u_x at time $t + p_x^{\min}$ (i.e., the beginning of u_x's current NAI since $\texttt{MOD}(p_x^{\min}, \tilde{P}) = 0$). To ensure that s_i will not send N_{NS} within the current NAI (or two notifications, N_{CR} and N_{NS}, are sent in this NAI, which breaches the goal of the scheduling module), the following condition should be satisfied:

$$t + p_x^{\min} + \tilde{P} > t + p_x^{\max} \Rightarrow \tilde{P} > p_x^{\max} - p_x^{\min}. \quad (4)$$

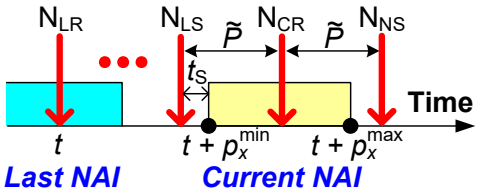
Eq. (4) is identical to the checking condition in line 2, so the first part is verified.

Fig. 3(b) gives another case. Specifically, N_{NS} will be sent at time $t + p_x^{\min} - t_{\text{S}} + 2\tilde{P}$. To prevent s_i from sending N_{NS} in u_x's current NAI, we have to guarantee that

$$\begin{aligned} &t + p_x^{\min} - t_{\text{S}} + 2\tilde{P} > t + p_x^{\max} \\ &\Rightarrow \tilde{P} > (p_x^{\max} - p_x^{\min} + t_{\text{S}})/2. \end{aligned} \quad (5)$$



(a) Case of $\texttt{MOD}(p_x^{\min}, \tilde{P}) = 0$



(b) Case of $\texttt{MOD}(p_x^{\min}, \tilde{P}) \neq 0$

Fig. 3: Two cases checked in Algo. 3.

Suppose that s_i sends α notifications between $\mathrm{N_{LR}}$ and $\mathrm{N_{CR}}$ (excluding $\mathrm{N_{LR}}$ and $\mathrm{N_{CR}}$). Then, we can derive that

$$\alpha\tilde{P} + t_{\mathrm{S}} = p_x^{\min}, \quad \alpha \in \mathbb{N}^0. \tag{6}$$

Evidently, Eq. (6) implies that t_{S} will be the remainder after dividing $p_x^{\min}$ by $\tilde{P}$. In other words, we have

$$t_{\mathrm{S}} = \texttt{MOD}(p_x^{\min}, \tilde{P}). \tag{7}$$

By combining Eqs. (5) and (7), we can obtain the checking condition in line 5, so the other part is also proven. □

Let us review the example in Fig. 1(d), where client devices u_1, u_2, and u_3 have demands of (9, 13), (5, 7), and (2, 3), respectively. Based on Algo. 2, we sort them and get $\hat{\mathcal{U}}_1^k = \{u_3, u_2, u_1\}$. In the beginning, we create a new group $\hat{\mathcal{G}}_1 = \{u_3\}$ and have $\hat{\mathcal{U}}_1^k = \{u_2, u_1\}$. By using OPS, we get $\tilde{P}_{\mathrm{tmp}} = 3$ for group $\hat{\mathcal{G}}_{\mathrm{tmp}} = \hat{\mathcal{G}}_1 \cup \{u_2\}$. Then, we check if u_2 can be added to $\hat{\mathcal{G}}_1$. By Algo. 3, we do the check as follows:

For u_3: Since $\texttt{MOD}(p_3^{\min}, \tilde{P}_{\mathrm{tmp}}) = \texttt{MOD}(2, 3) = 2$, we then do the check by line 5. As $(p_3^{\max} - p_3^{\min} + \texttt{MOD}(p_3^{\min}, \tilde{P}_{\mathrm{tmp}}))/2 = (3 - 2 + 2)/2 = 1.5 < \tilde{P}_{\mathrm{tmp}} = 3$, u_3 passes the check.

For u_2: Since $\texttt{MOD}(p_2^{\min}, \tilde{P}_{\mathrm{tmp}}) = \texttt{MOD}(5, 3) = 2$, we then do the check by line 5. As $(p_2^{\max} - p_2^{\min} + \texttt{MOD}(p_2^{\min}, \tilde{P}_{\mathrm{tmp}}))/2 = (7 - 5 + 2)/2 = 2 < \tilde{P}_{\mathrm{tmp}} = 3$, u_2 passes the check.

Thus, it is safe to add u_2 to $\hat{\mathcal{G}}_1$, so we have $\hat{\mathcal{G}}_1 = \{u_3, u_2\}$ and $\hat{\mathcal{U}}_1^k = \{u_1\}$. Next, we check if u_1 can be added to $\hat{\mathcal{G}}_1$. Again, we obtain that $\tilde{P}_{\mathrm{tmp}} = 3$ for group $\hat{\mathcal{G}}_{\mathrm{tmp}} = \hat{\mathcal{G}}_1 \cup \{u_1\}$ by OPS. Since u_3 and u_2 have been checked for the case of $\tilde{P}_{\mathrm{tmp}} = 3$, we only check u_1 below:

For u_1: Since $\texttt{MOD}(p_1^{\min}, \tilde{P}_{\mathrm{tmp}}) = \texttt{MOD}(9, 3) = 0$, we do the check by line 2. However, as $\tilde{P}_{\mathrm{tmp}} = 3 < p_1^{\max} - p_1^{\min} = 4$, u_1 cannot be added to $\hat{\mathcal{G}}_1$.

After that, we create another group $\hat{\mathcal{G}}_2$ which contains only u_1. In summary, two groups $\hat{\mathcal{G}}_1 = \{u_2, u_3\}$ and $\hat{\mathcal{G}}_2 = \{u_1\}$ are found by the scheduling module, whose observation periods are $\tilde{P}_{1,1} = 3$ and $\tilde{P}_{1,2} = 13$, respectively.

We then analyze some properties of the scheduling module. Theorem 3 shows that every group found by the scheduling module in a set $\hat{\mathcal{U}}_i^k$ of client devices must have an observation period no shorter than that found by the OPS algorithm for $\hat{\mathcal{U}}_i^k$. Theorem 4 proves that by dividing $\hat{\mathcal{U}}_i^k$ into multiple groups, the scheduling module can reduce the number of notifications received by the client devices in $\hat{\mathcal{U}}_i^k$, as compared with OPS.

Algorithm 4: The Caching Module

Data: Data generating interval T_i of IoT device s_i and each group $\hat{\mathcal{G}}_a$ found in $\hat{\mathcal{U}}_i^k$ by Algo. 2
Result: Max-Age value $\tilde{A}_i$ for s_i

1 $\tilde{A}_i \leftarrow \min_{\forall \hat{\mathcal{G}}_a \subseteq \hat{\mathcal{U}}_i^k} \{\tilde{P}_{i,a}\}$;
2 **if** $\tilde{A}_i < T_i$ **then**
3 $\quad \tilde{A}_i \leftarrow T_i$;
4 **else**
5 $\quad$ Proxy ρ_k asks s_i to send only the last notification in each $\tilde{A}_i$ period;

Theorem 3. *Let $\tilde{P}_i$ be the observation period found by OPS for a set $\hat{\mathcal{U}}_i^k$ of client devices. For each group $\hat{\mathcal{G}}_a \subseteq \hat{\mathcal{U}}_i^k$ found by the scheduling module in Algo. 2, it is guaranteed that $\tilde{P}_{i,a} \geq \tilde{P}_i$.*

Proof: We prove this theorem by contradiction. Suppose that Algo. 2 finds a group $\hat{\mathcal{G}}_b$ from $\hat{\mathcal{U}}_i^k$ such that $\tilde{P}_{i,b} < \tilde{P}_i$. In line 13 of Algo. 2, OPS in Algo. 1 is directly used to calculate $\tilde{P}_{i,b}$. According to Eq. (2), there must exist a client device, say, u_x in $\hat{\mathcal{G}}_b$ that meets the following two conditions:

$$p_x^{\max} < \lceil p_x^{\min}/\tilde{P}_i \rceil \tilde{P}_i, \tag{8}$$

$$p_x^{\max} \geq \lceil p_x^{\min}/\tilde{P}_{i,b} \rceil \tilde{P}_{i,b}, \tag{9}$$

Since u_x belongs to $\hat{\mathcal{U}}_i^k$, based on lines 2 and 3 in Algo. 1, OPS must find an observation period smaller than or equal to $\tilde{P}_{i,b}$ for $\hat{\mathcal{U}}_i^k$ to satisfy the conditions in Eqs. (8) and (9). This obviously causes a contradiction, so the theorem is proven. □

Theorem 4. *Let M_{OPS} and M_{SM} be the number of notifications received by all client devices of $\hat{\mathcal{U}}_i^k$ in OPS and the scheduling module during a constant time T, respectively. The condition of $M_{\mathrm{SM}} \leq M_{\mathrm{OPS}}$ must hold.*

Proof: Observing from Fig. 3(b), the interval between the reception of two successive notifications by a client device u_j (i.e., $\mathrm{N_{LR}}$ and $\mathrm{N_{CR}}$) is $\lceil p_j^{\min}/\tilde{P} \rceil \times \tilde{P}$, where $\tilde{P}$ is the observation period. Since OPS computes one single observation period $\tilde{P}_i$ for a set $\hat{\mathcal{U}}_i^k$ of client devices, we can derive that

$$M_{\mathrm{OPS}} = \sum_{u_j \in \hat{\mathcal{U}}_i^k} \left\lfloor \frac{T}{\lceil p_j^{\min}/\tilde{P}_i \rceil \times \tilde{P}_i} \right\rfloor. \tag{10}$$

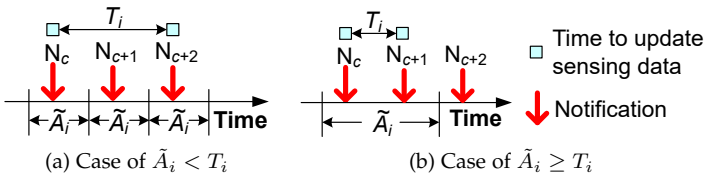
The scheduling module partitions $\hat{\mathcal{U}}_i^k$ into disjoint groups and then assigns each group $\hat{\mathcal{G}}_a$ an observation period $\tilde{P}_{i,a}$. Thus, we can calculate that

$$M_{\mathrm{SM}} = \sum_{\hat{\mathcal{G}}_a \subseteq \hat{\mathcal{U}}_i^k} \sum_{u_j \in \hat{\mathcal{G}}_a} \left\lfloor \frac{T}{\lceil p_j^{\min}/\tilde{P}_{i,a} \rceil \times \tilde{P}_{i,a}} \right\rfloor. \tag{11}$$

According to Theorem 3, each group in $\hat{\mathcal{U}}_i^k$ must satisfy $\tilde{P}_{i,a} \geq \tilde{P}_i$. In other words, the following condition will be met:

$$\left\lfloor \frac{T}{\lceil p_j^{\min}/\tilde{P}_{i,a} \rceil \times \tilde{P}_{i,a}} \right\rfloor \leq \left\lfloor \frac{T}{\lceil p_j^{\min}/\tilde{P}_i \rceil \times \tilde{P}_i} \right\rfloor. \tag{12}$$

By combining Eqs. (10), (11), and (12), we derive that $M_{\mathrm{SM}} \leq M_{\mathrm{OPS}}$, thereby validating this proof. □



(a) Case of $\tilde{A}_i < T_i$ (b) Case of $\tilde{A}_i \geq T_i$

Fig. 4: Two cases considered in Algo. 4.

5.2 The Caching Module

The OPS algorithm asks a proxy to always forward "non-cached" notifications to client devices (referring to the examples in Fig. 1), which may force IoT devices to frequently send notifications to the proxy, thereby consuming more energy. In fact, CoAP permits proxies to cache notifications for cutting down the energy expense of IoT devices. This can be realized by adopting the Max-Age option, as mentioned in Section 2.3. Therefore, the objective of our caching module is to calculate an adequate Max-Age value $\tilde{A}_i$ for each IoT device s_i.

Let T_i be the *data generating interval* of s_i, which is the time interval between two successive sensing data produced by s_i. It is worth noting that T_i genuinely reflects the *heterogeneity* of IoT devices. More concretely, different types of IoT devices would generate sensing data at different speeds, which results in different T_i values. Furthermore, IoT devices could change their T_i values in different situations. For example, they may accelerate the generation of sensing data due to event detection [22], thereby shortening T_i. On the other hand, IoT devices could slow down the speed to produce data when they go to longer sleeping cycles [23]. In this case, their T_i values will increase accordingly. Therefore, we should take account of this heterogeneity by introducing T_i to our caching module.

Algo. 4 gives the pseudocode of the caching module. To prevent some client devices from getting notifications with stale sensing data, proxy ρ_k expects to obtain a fresh enough notification (sent from s_i) to satisfy the demand of each client device in $\hat{\mathcal{U}}_i^k$. That is why $\tilde{A}_i$ is initially set to the minimum observation period of each group in $\hat{\mathcal{U}}_i^k$ by line 1.

Then, we check if $\tilde{A}_i$ can be enlarged (so s_i can send fewer notifications to save more energy) by considering the effect of T_i. There are two cases to be addressed, as shown in Fig. 4. For the case of $\tilde{A}_i < T_i$ in Fig. 4(a), if ρ_k asks s_i to send notifications following the $\tilde{A}_i$ period set in line 1, the value of sensing data in notification N_{c+1} must be identical with that in notification N_c, because s_i will update its sensing data only after time T_i (e.g., notification N_{c+2}). Apparently, N_{c+1} is redundant, so we change $\tilde{A}_i$ to a larger T_i value to remove unnecessary notifications. The code is given in lines 2 and 3.

For the case of $\tilde{A}_i \geq T_i$ in Fig. 4(b), since $\tilde{A}_i$ set by line 1 meets the demand of each client device in $\hat{\mathcal{U}}_i^k$ and s_i must update its sensing data during every $\tilde{A}_i$ period, any notification sent by s_i within $\tilde{A}_i$ will not be overdue from the viewpoint of client devices in $\hat{\mathcal{U}}_i^k$. Thus, there is no need to change the value of $\tilde{A}_i$. However, s_i may send multiple notifications to ρ_k in each $\tilde{A}_i$ period. In fact, s_i can send just the last notification within $\tilde{A}_i$ (i.e., N_{c+1}) with the latest updated sensing data to ρ_k, so as to save its energy. The code is presented in lines 4 and 5. Theorem 5 proves that the caching module can reduce notifications sent from IoT devices, as compared with OPS.

Theorem 5. *By using Algo. 4, any IoT device s_i will not send notifications to proxy ρ_k more than that using OPS.*

Algorithm 5: The Merging Module

Data: Period $\tilde{P}_{i(j)}$ and notification N_i of each IoT device $s_i \in \hat{\mathcal{S}}_j^k$ (queried by client device u_j)
Result: Notifications N_b that proxy ρ_k sends to u_j

1 **while** $\hat{\mathcal{S}}_j^k \neq \emptyset$ **do**
2 $\quad s_x \leftarrow \arg\min_{\forall s_i \in \hat{\mathcal{S}}_j^k}\{\tilde{P}_{i(j)}\}$;
3 $\quad$ Create a basic notification N_b initially set to N_x;
4 $\quad \hat{\mathcal{S}}_j^k \leftarrow \hat{\mathcal{S}}_j^k \setminus \{s_x\}$;
5 $\quad$ **foreach** $s_y \in \hat{\mathcal{S}}_j^k$ **do**
6 $\quad\quad$ **if** $\tilde{P}_{y(j)} = \tilde{P}_{x(j)}$ *or* $\tilde{A}_y \geq \tilde{P}_{x(j)}$ **then**
7 $\quad\quad\quad \mathrm{N}_b \leftarrow \mathrm{N}_b \parallel \mathrm{N}_y$;
8 $\quad\quad\quad \hat{\mathcal{S}}_j^k \leftarrow \hat{\mathcal{S}}_j^k \setminus \{s_y\}$;

Proof: In line 1 of Algo. 4, $\tilde{A}_i$ is set to the minimum observation period of every group in $\hat{\mathcal{U}}_i^k$. According to Theorem 3, $\tilde{A}_i \geq \tilde{P}_i$ must hold, where $\tilde{P}_i$ is the observation period computed by OPS for s_i. Let us review the two cases in Fig. 4. When $\tilde{A}_i < T_i$, since the Max-Age value is set to T_i in line 3, which is larger than $\tilde{P}_i$ found in OPS, we guarantee that s_i must elongate the interval to send notifications by using Algo. 4, thereby saving more notifications than OPS. If $\tilde{A}_i \geq T_i$, the Max-Age value is set to the minimum observation period, which is no smaller than $\tilde{P}_i$ by applying Algo. 4. Thus, s_i sends at most the same number of notifications with OPS. Consequently, the above two cases verify this proof. □

5.3 The Merging Module

A client device u_j could request data from a set $\hat{\mathcal{S}}_j^k$ of IoT devices in practice. If proxy ρ_k simply forwards notifications to u_j, u_j will receive one notification originated from every device in $\hat{\mathcal{S}}_j^k$, where each notification has the identical header and some similar content (e.g., "2.05 Content, Token: XXX", as shown in Fig. 2). Apparently, sending these duplicate data wastes both bandwidth and energy. In view of this, our merging module helps each proxy integrate multiple notifications, so it can relay as few notifications as possible to client devices.

Algo. 5 gives the pseudocode of the merging module. Let $\tilde{P}_{i(j)}$ be the observation period for each IoT device $s_i \in \hat{\mathcal{S}}_j^k$ to meet the demand of u_j (found by Algo. 2) and N_i be its notification. In lines 2 and 3, we first pick the notification of an IoT device, say, s_x from $\hat{\mathcal{S}}_j^k$ with the minimum observation period to be the *base* (denoted by N_b), so other notifications can be combined with N_b. The reason is that the notification of s_x is the most urgent one, so ρ_k should deal with it first. Then, s_x is removed from $\hat{\mathcal{S}}_j^k$ by line 4.

In lines 5–8, we check if any notification of residual devices in $\hat{\mathcal{S}}_j^k$ can be combined with N_b. Line 6 gives two checking conditions. First, if a device $s_y \in \hat{\mathcal{S}}_j^k$ has the same observation period with s_x (i.e., $\tilde{P}_{y(j)} = \tilde{P}_{x(j)}$), their notifications can be merged. Second, if the Max-Age value of s_y is no smaller than the observation period of s_x (i.e., $\tilde{A}_y \geq \tilde{P}_{x(j)}$), which means that ρ_k has a non-overdue cached notification for s_y, ρ_k can also consolidate this cached notification with N_b. In line 7, the notation '$\parallel$' indicates the operation of merging notifications. If so, s_y is removed from $\hat{\mathcal{S}}_j^k$. The above procedure is repeated until $\hat{\mathcal{S}}_j^k$ becomes empty (i.e., each notification given to u_j is either combined with others or sent alone).

Recall in Section 2.2, with the proxy mechanism in CoAP, client devices usually get data from a proxy without knowing the identifications of IoT devices for safety reasons. Therefore, we can merge notifications from multiple IoT devices by just combining their payloads. There are two cases to be discussed.

[Case 1] All devices in $\hat{\mathcal{S}}_j^k$ are homogeneous: Since these IoT devices produce the same type of sensing data, we can thus apply some data aggregation or compression schemes [24] to merge and condense their data. The merged data will be put in the payload of a notification (i.e., N_b).

[Case 2] Devices in $\hat{\mathcal{S}}_j^k$ are heterogeneous: In this case, we will concatenate their sensing data in the payload. Since the payload's size is fixed but the length of sensing data produced by each device in $\hat{\mathcal{S}}_j^k$ could be different, our objective is to use the minimum notifications to pack all data. Interestingly, the *bin packing problem*, where items of different volumes must be packed into identical bins in a way that minimizes the number of bins used, is reducible to our problem. By simple substitutions, the problem becomes data (items) of different sizes (volumes) must be loaded (packed) into payloads of the same size (identical bins) in a way that minimizes the number of payloads (bins) used. It is an NP-hard problem, and we can employ some approximation algorithms [25] to solve the problem. An alternative solution is to further partition devices in $\hat{\mathcal{S}}_j^k$ into several subgroups with homogeneous devices. Then, for devices within each subgroup, we merge their data into one payload as in Case 1. This approach is peculiarly suited for devices in $\hat{\mathcal{S}}_j^k$ consisting of limited types.

5.4 Design Rationale

Given an IoT device s_i and a set $\hat{\mathcal{U}}_i^k$ of client devices that ask for its data via proxy ρ_k, the OPS algorithm in [20] finds the optimal observation period to minimize notifications that s_i will send to ρ_k, as mentioned earlier in Theorem 1. However, OPS has three deficiencies. Thus, our GMM framework adopts three modules to conquer each of them. First, since OPS uses a common period for every client device in $\hat{\mathcal{U}}_i^k$, long-period devices unavoidably accommodate themselves to short-period devices, which forces them to get more notifications than they need. To overcome this problem, the scheduling module (i.e., Algo. 2) divides $\hat{\mathcal{U}}_i^k$ into groups and finds an optimal period for each group, so as to reduce the number of notifications received by client devices. Second, OPS minimizes the number of notifications that s_i will send on the premise that ρ_k always forwards its notifications to every client device *immediately*. In fact, CoAP proxies are allowed to cache notifications and reuse them according to RFC 7252 [3]. To exploit this mechanism, the caching module in Algo. 4 calculates an appropriate value of Max-Age for each IoT device based on its minimum observation period and data generating interval. Thus, GMM further diminishes notifications transmitted from IoT devices to proxies. Third, OPS considers only the relationship between one single IoT device (i.e., s_i) and multiple client devices (i.e., $\hat{\mathcal{U}}_i^k$). In addition to this relationship, GMM also addresses the practical case where each client device u_j queries data from a set $\hat{\mathcal{S}}_j^k$ of IoT devices. The merging module in Algo. 5 combines the notifications originated from IoT devices in $\hat{\mathcal{S}}_j^k$, so ρ_k can send as few notifications to u_j as possible to save both energy and bandwidth. The above designs distinguish our GMM framework from the OPS solution and help greatly save the message cost in a CoAP-based IoT network.

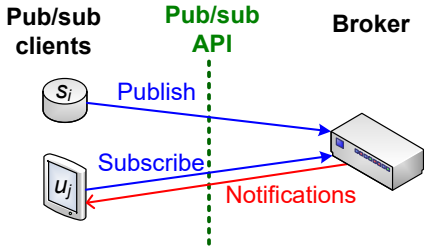


Fig. 5: CoAP pub/sub architecture.

TABLE 2: Comparison between our GMM framework and the CoAP pub/sub extension.

item	GMM framework	pub/sub extension
architecture	Fig. 1(a)	Fig. 5
model	request-response	publish-subscribe
application	real-time or short sleeping	limited reachability
intermediary	proxy	broker
association	observe flag	topic
user demand	$(p_j^{\min}, p_j^{\max})$	not support

6 Discussion

6.1 CoAP Publish/Subscribe Extension

In [26], a publish/subscribe (abbreviated to *pub/sub*) extension for CoAP is proposed to support those devices with long breaks in connectivity (e.g., due to deep sleep). Fig. 5 gives its architecture, where pub/sub clients interact with a special node called *broker* via the CoAP pub/sub REST application program interface (API) hosted by the broker. The state information is updated between clients and the broker by a store-and-forward method. As shown in Fig. 5, some devices (e.g., u_j) subscribe to the sensing data which are published by other devices (e.g., s_i). Afterward, the broker forwards data (by notifications) to the subscribing clients.

The pub/sub extension can reduce the number of messages sent by IoT devices, which need not depend on the behavior of client devices. Nevertheless, this extension and our GMM framework have different objectives and application domains. Table 2 presents a comparison between them. First, the GMM framework adopts the common CoAP architecture defined in RFC 7252 [3], so it could perform well on most of CoAP-based devices. On the contrary, the pub/sub extension does not conform to the client-server architecture specified in CoAP and just views all devices as clients, which may have backward compatibility problems with old devices.

Second, GMM and the pub/sub extension implement different communication models. More concretely, GMM uses the popular request-response model, where client devices make requests to IoT devices if they are interested in sensing data. After that, IoT devices *actively* reply notifications to update the status of their sensing data. In the pub/sub extension, IoT devices *occasionally* publish their sensing data to a broker, regardless of whether other devices have interest in these data. Thus, the GMM framework can be applied to more general scenarios, where client devices can ask for real-time or fresh data from IoT devices, or IoT devices can also doze off to save energy when situations allow. On the other hand, the pub/sub extension is suitable for a special scenario where IoT devices have very limited reachability because they spend most of the time in a sleeping state with almost no network connectivity.

Third, proxies serve as "coordinators" in the GMM framework, where they help regulate the observation period for IoT devices to send out notifications and forward the notifications

to client devices to satisfy their demands. By contrast, brokers function more like "containers" to store the sensing data of IoT devices beforehand, and then give these data to subscribing client devices later in the pub/sub extension. This difference lets GMM and the pub/sub extension take different strategies to make associations with the messages of devices. As discussed in Section 2.1, GMM uses the observing mechanism in CoAP by associating the notifications of IoT devices with the requests of client devices through *observe flags*. On the other hand, the pub/sub extension employs *topics* to establish the association relationship, which are unique identifiers for particular items being published and subscribed to.

Lastly, the GMM framework adds flexibility to client devices by allowing them to indicate minimum and maximum periods that they desire to receive notifications (i.e., $p_j^{\min}$ and $p_j^{\max}$). Based on the demands, GMM not only finds optimal observation periods for IoT devices to transmit notifications but also makes good use of both cache and proxy mechanisms provided by CoAP to save the message cost and reduce the energy consumption of devices. Although very useful, this feature is not supported by the pub/sub extension.

6.2 Heterogeneous IoT Networks

Our GMM framework can work well in heterogeneous IoT networks, and this section explains how GMM accommodates itself to these networks. In particular, a heterogeneous IoT network may be composed of heterogeneous client devices or heterogeneous IoT devices. According to [4], the profiles of (heterogeneous) client devices can be reflected by their demands. For example, high-priority client devices are set with smaller $p_j^{\min}$ and $p_j^{\max}$ values (i.e., short-period demands) for the preferential reception of notifications, while low-priority ones are assigned with larger $p_j^{\min}$ and $p_j^{\max}$ values (i.e., long-period demands) to save bandwidth[2]. Even for an individual client device, it can differentiate between the importance of different sensing data from different IoT devices by giving different demands of $p_j^{\min}$ and $p_j^{\max}$. Based on Theorems 1 and 2, we show that the demand of each client device must be satisfied (i.e., it will get notifications during the interval between $p_j^{\min}$ and $p_j^{\max}$) in GMM. This implies that GMM can cope with the heterogeneity in client devices.

On the other hand, depending on the monitoring missions or their functions, different characteristics could be exhibited on IoT devices, such as battery consumption, the quantity of data being sensed, and specific properties of the acquired data. This heterogeneity can be taken on by their data generating intervals (i.e., T_i). Specifically, if an IoT device wants to conserve more energy (e.g., by sleeping for a longer time), it would has a larger T_i value. Moreover, the data generating interval decides the amount of data being sensed. The shorter the interval is, the more data will be sensed, and vice versa. Furthermore, IoT devices equipped with different types of sensing devices may have different T_i values due to hardware designs or properties of the data [27], [28]. By taking account of the effect of T_i values in Algo. 4, the caching module can help GMM deal with heterogeneous IoT devices. In addition, the merging module in Algo. 5 handles the case where the lengths of sensing data produced by IoT devices may be different. This also addresses the heterogeneity in IoT devices.

2. In this way, IoT devices and proxies can decrease their frequencies of sending/forwarding notifications to these low-priority client devices, thereby saving bandwidth.

TABLE 3: Mean and standard deviation of data generating intervals of IoT devices contacted by each proxy.

proxy	mean	standard deviation
ρ_A	10 seconds	2 seconds
ρ_B	30 seconds	6 seconds
ρ_C	50 seconds	10 seconds

TABLE 4: Four simulation scenarios applied to a single proxy.

parameter	I	II	III	IV
client devices	10–50	50	10	50
LPP $L_{\max}$	50	10–50	50	50
total requests	50	50	10–50	50
IoT devices	10	10	10	10–50

7 Performance Evaluation

We adopt the Californium simulator [29] to imitate a CoAP-based IoT network and implement our GMM framework on it for performance evaluation. The OPS method [20] is used for comparison. Because OPS asks each proxy to directly forward notifications generated by an IoT device to each client device, we apply our caching and merging modules in Algos. 4 and 5 to OPS, so as to let it use the cache mechanism in CoAP and also combine notifications from multiple IoT devices. This method is named as *enhanced OPS (E-OPS)*. By comparing E-OPS with GMM, we can measure the amount of performance improvement gained by our scheduling module in Algo. 2.

In our simulations, there are three proxies ρ_A, ρ_B, and ρ_C. Each of them builds observation relationships with some IoT devices on behalf of a number of client devices and relays their notifications to client devices. Let us consider a *heterogeneous* IoT network composed of three types of IoT devices, where the devices of the same type have similar data generating intervals (i.e., T_i). To observe the effect of heterogeneous IoT devices, each proxy contacts with only one type of IoT devices. More concretely, the T_i values of the IoT devices contacted by each proxy obey a uniform distribution whose mean and standard deviation are given in Table 3, where the standard deviation is taken as 20% of the mean. Based on the above setting, proxies ρ_A, ρ_B, and ρ_C will relay the notifications of IoT devices that generate sensing data across a wide range of frequencies.

For each proxy, we consider the four scenarios in Table 4. The minimum period $p_j^{\min}$ of a client device is picked from the range of $[1, L_{\max}]$ (arbitrarily by the uniform distribution), where $L_{\max}$ is the *longest possible period (LPP)*. Then, its maximum period $p_j^{\max}$ is selected between $p_j^{\min}$ and $L_{\max}$. In each scenario, only one parameter is adjusted (while others are kept constant) to observe its effect:

- **I.** Every proxy serves 10 to 50 client devices (i.e., there are 30 to 150 client devices totally) and contacts 10 IoT devices (i.e., there are 30 IoT devices in total). A proxy gets 50 requests from its served client devices (i.e., there are 150 requests in all). The LPP (i.e., $L_{\max}$) is set to 50 seconds.
- **II.** We increase $L_{\max}$ from 10 to 50 seconds. A proxy serves 50 client devices and contacts 10 IoT devices. There are 50 requests sent to each proxy. This scenario considers the heterogeneity in client devices.
- **III.** There are 10 to 50 requests sent to a proxy. Each proxy serves 10 client devices and also contacts 10 IoT devices. The $L_{\max}$ value is set to 50 seconds.
- **IV.** Each proxy contacts 10 to 50 IoT devices. It also serves 50 client devices, which generate 50 requests. The LPP is kept to 50 seconds.

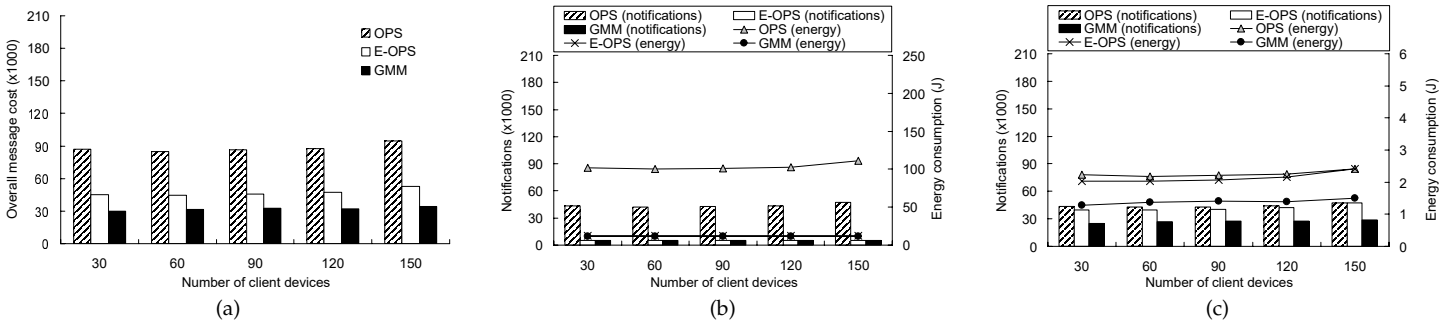


Fig. 6: Performance evaluation in scenario I by varying the number of client devices: (a) the overall message cost, (b) the number of notifications sent by IoT devices and their energy consumption, and (c) the number of notifications forwarded to client devices and their energy consumption.

We measure the *message cost*, which is defined by the total number of notifications sent in the network. This cost includes the notifications sent from IoT devices and those forwarded to client devices by proxies. Moreover, we evaluate the amount of energy consumed by devices in the IoT network. In particular, suppose that a sensor s_i transmits a notification with l bits to a proxy ρ_k, and their Euclidean distance is denoted by $D(s_i, \rho_k)$. Then, the amount of energy that s_i spends on data transmission can be estimated as follows [30]:

$$E_{\mathrm{Tx}}(s_i, \rho_k) = (\psi_1 + \psi_2[D(s_i, \rho_k)]^{\varepsilon}) \times l, \tag{13}$$

where ψ_1 and ψ_2 indicate the amount of power taken by s_i's transmitter and amplifier circuits to deliver a bit, respectively, and ε is an exponent used to model path loss. On the other hand, a client device u_j also depletes some energy to receive a notification relayed from ρ_k, which is measured by

$$E_{\mathrm{Rx}}(u_j, \rho_k) = \psi_3 \times l, \tag{14}$$

where ψ_3 gives the amount of power required by u_j's receiver circuit to get one bit. Based on [30], we set $\psi_1 = 50\,\mathrm{nJ/bit}$, $\psi_2 = 100\,\mathrm{pJ/bit}$ per m^2, $\psi_3 = 50\,\mathrm{nJ/bit}$, and $\varepsilon = 2$. The distance $D(s_i, \rho_k)$ is set to 150 meters. Besides, we consider using ZigBee as the underlying protocol, so the length of a notification will be 128 bytes. The total simulation time is set to 3600 seconds (i.e., one hour).

7.1 Scenario I: Varying Client Devices

In scenario I, the number of total client devices is increased from 30 to 150. Fig. 6(a) compares the overall message cost of each method. As can be seen, increasing client devices has little effect on the message cost, because the number of requests does not change and LPP $L_{\max}$ of client devices is also fixed. By applying the caching and merging modules, the E-OPS method significantly diminishes the message cost compared to the original OPS method. Our GMM framework always keeps the lowest message cost among all methods. On average, GMM can save 63.5% and 31.8% of the message cost, as compared with OPS and E-OPS, respectively.

Fig. 6(b) presents the number of notifications sent by IoT devices and their energy consumption. Since proxies cannot cache notifications in OPS, IoT devices have to send notifications more frequently. That is why OPS has distinctly more notifications than others. By comparing E-OPS with OPS, we show that the caching module in Algo. 4 helps lower 88.7% of notifications and energy consumption of IoT devices in OPS, which verifies its high efficiency. Since GMM also uses the same caching module, its result is similar to that of E-OPS. Despite this, the scheduling module in Algo. 2 still helps GMM further save 5.6% of notifications and energy consumption of IoT devices, as compared with E-OPS.

Then, Fig. 6(c) gives the number of notifications forwarded to client devices and their energy consumption. By applying the merging module in Algo. 5 to OPS, the E-OPS method can decrease notifications forwarded to client devices, but the reduction is not large (in particular, less than 5.2%). The reason is that E-OPS finds the same observation period with OPS for all client devices served by each proxy. Thanks to the scheduling module, GMM can flexibly group client devices and assign an optimal period for each group. In this way, there is a good possibility for the merging module to combine more notifications. To sum up, GMM reduces 38.5% and 35.2% of notifications forwarded to client devices (and their energy consumption), as compared with OPS and E-OPS, respectively.

7.2 Scenario II: Varying The Longest Possible Period

We raise LPP $L_{\max}$ from 10 to 50 in this scenario to observe the effect of the heterogeneity in client devices. Specifically, a larger $L_{\max}$ value implies that most client devices would have larger $p_j^{\min}$ and $p_j^{\max}$ periods. In this situation, the number of notifications that client devices expect to receive will decrease when $L_{\max}$ increases. That is why the overall message cost of each method decreases as $L_{\max}$ grows in Fig. 7(a). Since the E-OPS method uses the cache mechanism in CoAP, it results in a lower message cost than OPS. As compared with OPS and E-OPS, our GMM framework can averagely reduce 54.5% and 17.5% of the message cost, respectively.

Let us analyze the numbers of notifications sent from IoT devices and forwarded to client devices, as shown in Fig. 7(b) and (c), respectively. These two figures also give the amount of energy consumption of devices. As can be seen, the caching module helps the E-OPS method reduce more notifications and energy consumption than OPS (referring to Fig. 7(b)) whereas the merging module does not (referring to Fig. 7(c)). There are two reasons. First, client devices are many more than IoT devices (i.e., there are 150 client devices but only 30 IoT devices). Since the number of requests is equal to the number of client devices, a client device may seldom request multiple IoT devices. Second, the E-OPS method has the same observation period as OPS for client devices. In this case, the merging module is not easy to aggregate notifications in E-OPS. By properly grouping client devices, GMM can conquer the above difficulty, so the merging module performs better in GMM to combine notifications.

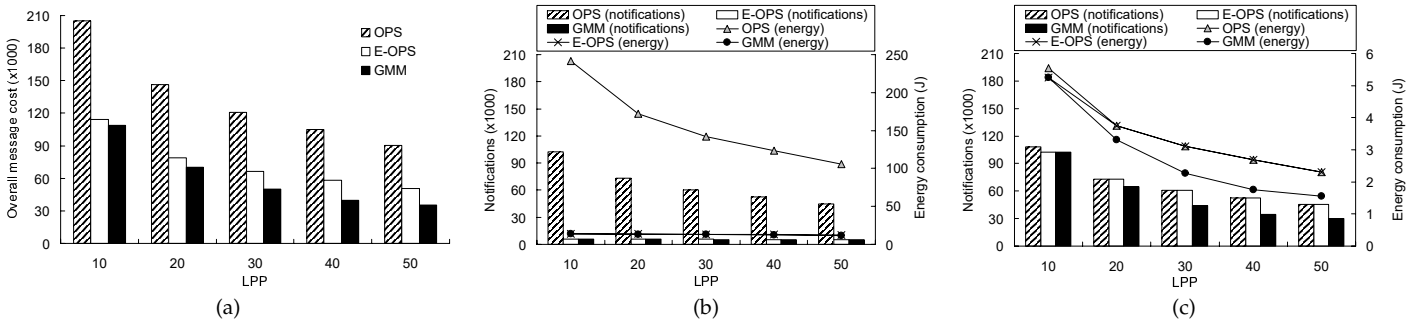


Fig. 7: Performance evaluation in scenario II by varying LPP $L_{\max}$: (a) the overall message cost, (b) the number of notifications sent by IoT devices and their energy consumption, and (c) the number of notifications forwarded to client devices and their energy consumption.

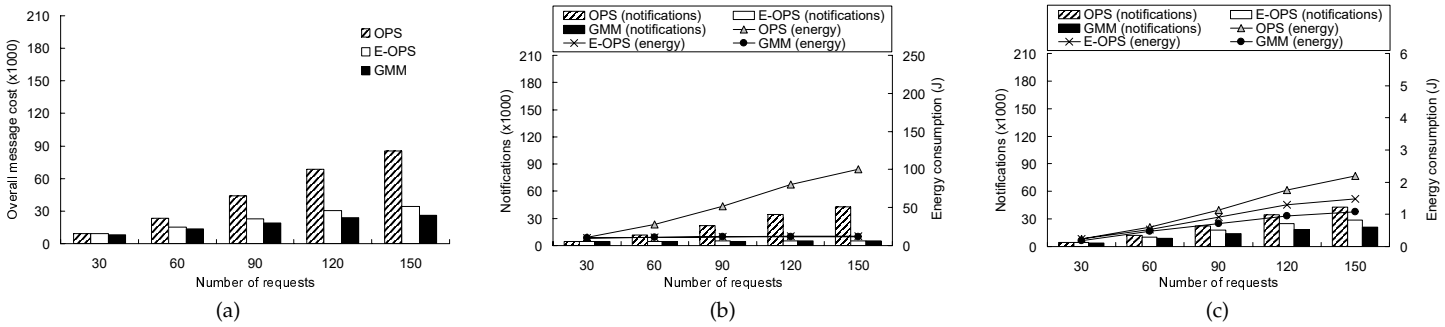


Fig. 8: Performance evaluation in scenario III by varying the number of total requests: (a) the overall message cost, (b) the number of notifications sent by IoT devices and their energy consumption, and (c) the number of notifications forwarded to client devices and their energy consumption.

Based on Fig. 7(b), GMM reduces 91.8% and 2.7% of notifications and energy consumption of IoT devices, as compared with OPS and E-OPS, respectively. Furthermore, GMM saves 18.7% and 17.3% of notifications and energy consumption of client devices than OPS and E-OPS, respectively, in Fig. 7(c). These experimental results validate the effectiveness of GMM under different demands of client devices in scenario II.

7.3 Scenario III: Varying Total Requests

Then, we evaluate the effect of different numbers of requests on the overall message cost, as shown in Fig. 8(a). Plainly, the message cost increases as there are more requests. Such a phenomenon is more obvious in OPS. On average, our GMM framework can decrease 60.9% and 19.0% of the message cost as compared with OPS and E-OPS, respectively.

Fig. 8(b) shows both notifications and energy consumption of IoT devices. Without considering the cache mechanism, IoT devices need to send more notifications when there are more requests in OPS, which forces them to consume more energy. Thanks to the caching module, the growth in both notifications and energy consumption for E-OPS and GMM is pretty small. To sum up, the GMM framework saves 79.3% and 2.4% of notifications and energy consumption of IoT devices over OPS and E-OPS, respectively.

The number of notifications sent to client devices and their energy consumption is presented in Fig. 8(c). Similar to the trend in Fig. 8(a), more notifications will be forwarded to client devices when more requests are generated. Through efficiently combining notifications in Algo. 5, GMM can curtail 42.6% and 23.7% of notifications forwarded to client devices and also their energy consumption, as compared with OPS and E-OPS, respectively.

7.4 Scenario IV: Varying IoT Devices

In this scenario, we vary the number of IoT devices. Since client devices and requests are constant, increasing IoT devices implies that the requests of client devices can be distributed among more IoT devices. This causes an effect of "grouping" client devices. Thus, the overall message cost of each method decreases when the number of IoT devices increases, as shown in Fig. 9(a). Even in this case, our GMM framework can still cut down 27.9% and 16.9% of the message cost over OPS and E-OPS, respectively, which shows its adaptability.

Fig. 9(b) compares notifications and energy consumption of IoT devices. For OPS, there will be fewer notifications by increasing IoT devices. On the other hand, since the constant requests are processed by more IoT devices, the efficiency of the cache mechanism may decline. That is why E-OPS and GMM have slightly more notifications when the number of IoT devices increases. On average, GMM performs as good as E-OPS and saves 56.4% of notifications and energy consumption of IoT devices than OPS.

In Fig. 9(c), we compare the number of notifications relayed to client devices and their energy consumption. Interestingly, OPS and E-OPS have almost equal notifications and energy consumption. This means that the merging module can hardly work in E-OPS. Since the number of requests is equal to the number of client devices, each client device almost requests just one IoT device. Besides, the observation periods found by both OPS and E-OPS are the same. Thus, the merging module cannot find notifications to be amalgamated. Thanks to the scheduling module, GMM can group client devices based on their demands and reduce the number of notifications sent to them. In particular, GMM saves 23.9% of notifications forwarded to client devices and also their energy consumption,

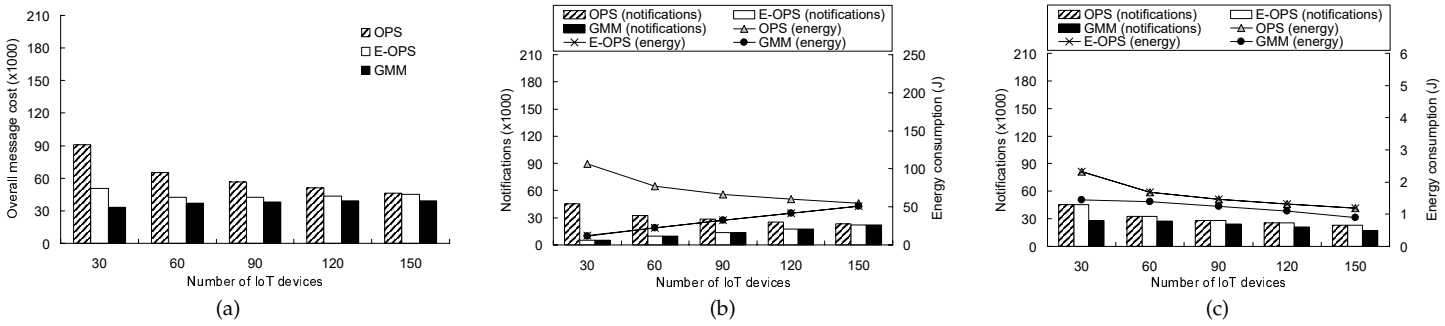


Fig. 9: Performance evaluation in scenario IV by varying the number of IoT devices: (a) the overall message cost, (b) the number of notifications sent by IoT devices and their energy consumption, and (c) the number of notifications forwarded to client devices and their energy consumption.

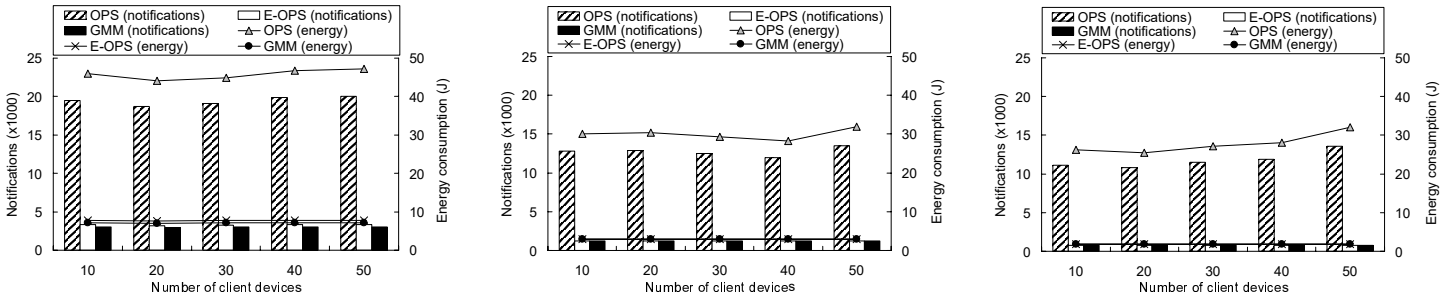


Fig. 10: Comparison on the number of notifications and the amount of energy consumption of IoT devices contacted by each proxy in scenario I.

as compared with OPS and E-OPS.

7.5 Effect of Heterogeneous IoT Devices

Finally, we study the effect of heterogeneous IoT devices in terms of data generating intervals (i.e., T_i). Since the effects of different parameters have been discussed in previous sections and T_i is used in the caching module for a proxy to reuse its cached notifications to answer requests, we pick scenario I as a representative and evaluate the number of notifications sent by IoT devices and their energy consumption.

Fig. 10 compares notifications and energy consumption of IoT devices contacted by proxies ρ_A, ρ_B, and ρ_C. Recall that each proxy contacts with one type of IoT devices whose T_i values follow a uniform distribution. According to Table 3, the sets of IoT devices contacted by ρ_A, ρ_B, and ρ_C would produce sensing data in a relatively high, medium, and low frequency, respectively. This heterogeneity of IoT devices decides the number of notifications sent by IoT devices and also their energy consumption. For OPS, although it does not use the cache mechanism in CoAP, OPS still has to ensure that client devices can always get the *freshest* notifications from IoT devices. Thus, when IoT devices update the status of sensing data more frequently (e.g., the case of proxy ρ_A in Fig. 10(a)), there will be more notifications sent from them in OPS. On the other hand, both E-OPS and GMM adopt the caching module in Algo. 4, which takes account of both T_i and the observation period in deciding the Max-Age value. Thus, when IoT devices produce sensing data more slowly (e.g., the cases of proxy ρ_B in Fig. 10(b) and most notably proxy ρ_C in Fig. 10(c)), each proxy is allowed to reuse more cached notifications for answering requests from client devices. Therefore, IoT devices can send fewer notifications and save more energy accordingly.

8 Conclusion

CoAP helps constrained IoT devices sensibly access the web and uses proxies to mediate between them and client devices. In practice, client devices possess different demands on getting notifications, but each IoT device may generate notifications in a fixed period. It is a challenge to manage the generation and forwarding of notifications to reduce the message cost in the network. Thus, we propose the GMM framework containing three modules. The scheduling module groups client devices by their demands and computes an optimal observation period for every group. To help proxies better reuse notifications, the caching module finds an appropriate Max-Age value for each IoT device. When a client device queries multiple IoT devices, the merging module combines notifications generated by these IoT devices, so a proxy can forward fewer notifications to the client device. By considering the four scenarios in simulations, we show that GMM greatly saves the message cost and energy of devices, as compared with OPS and its enhanced version.

References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: a survey on enabling technologies, protocols, and applications," *IEEE Comm. Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[2] Y.C. Wang, "Mobile sensor networks: system hardware and dispatch software," *ACM Computing Surveys*, vol. 47, no. 1, pp. 12:1–12:36, 2014.

[3] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (CoAP)," Internet Engineering Task Force, RFC 7252, June 2014. [Online]. Available: https://tools.ietf.org/html/rfc7252

[4] Z. Shelby, M. Koster, C. Groves, J. Zhu, and B. Silverajan, "Dynamic resource linking for constrained RESTful environments," Internet Engineering Task Force, Internet-Draft, July 2019. [Online]. Available: https://tools.ietf.org/html/draft-ietf-core-dynlink-10

[5] Y.C. Wang and G.W. Chen, "Efficient data gathering and estimation for metropolitan air quality monitoring by using vehicular sensor networks," *IEEE Trans. Vehicular Technology*, vol. 66, no. 8, pp. 7234–7248, 2017.
[6] L. Richardson and S. Ruby, *RESTful Web Services*. Sebastopol: O'Reilly, 2007.
[7] K. Hartke, "Observing resources in the constrained application protocol (CoAP)," Internet Engineering Task Force, RFC 7641, September 2015. [Online]. Available: https://tools.ietf.org/html/rfc7641
[8] E. Mingozzi, G. Tanganelli, and C. Vallati, "CoAP proxy virtualization for the Web of Things," *Proc. IEEE Int'l Conf. Cloud Computing Technology and Science*, 2014, pp. 577–582.
[9] S.I. Choi and S.J. Koh, "Use of proxy mobile IPv6 for mobility management in CoAP-based Internet-of-Things networks," *IEEE Comm. Letters*, vol. 20, no. 11, pp. 2284–2287, 2016.
[10] F.M. Barreto, P.A.S. Duarte, M.E.F. Maia, R.M.C. Andrade, and W. Viana, "CoAP-CTX: a context-aware CoAP extension for smart objects discovery in Internet of Things," *Proc. IEEE Annual Computer Software and Applications Conf.*, 2017, pp. 575–584.
[11] E. Ancillotti and R. Bruno, "BDP-CoAP: leveraging bandwidth-delay product for congestion control in CoAP," *Proc. IEEE World Forum on Internet of Things*, 2019, pp. 656–661.
[12] W.U. Rahman, Y.S. Choi, and K. Chung, "Performance evaluation of video streaming application over CoAP in IoT," *IEEE Access*, vol. 7, pp. 39852–39861, 2019.
[13] V.B. Misic and J. Misic, "Performance of caching in a layered CoAP proxy," *Proc. IEEE Int'l Wireless Comm. & Mobile Computing Conf.*, 2018, pp. 89–94.
[14] J. Misic, V.B. Misic, and X. Chang, "Kernel based estimation of domain parameters at IoT proxy," *Proc. IEEE Global Comm. Conf.*, 2018, pp. 1–6.
[15] D. Garcia-Carrillo and R. Marin-Lopez, "Multihop bootstrapping with EAP through CoAP intermediaries for IoT," *IEEE Internet of Things J.*, vol. 5, no. 5, pp. 4003–4017, 2018.
[16] P. Pierleoni, A. Belli, L. Palma, L. Incipini, S. Raggiunto, M. Mercuri, R. Concetti, and L. Sabbatini, "A cross-protocol proxy for sensor networks based on CoAP," *Proc. IEEE Int'l Symp. Consumer Technologies*, 2019, pp. 251–255.
[17] A. Ludovici, E. Garcia, X. Gimeno, and A.C. Auge, "Adding QoS support for timeliness to the observe extension of CoAP," *Proc. IEEE Int'l Conf. Wireless and Mobile Computing, Networking and Comm.*, 2012, pp. 195–202.
[18] N. Correia, G. Schutz, and A. Barradas, "Fairness for CoAP/observe based wireless sensor networks with aggregation deployment," *Proc. IEEE World Forum on Internet of Things*, 2015, pp. 1–6.
[19] N. Correia, D. Sacramento, and G. Schutz, "Dynamic aggregation and scheduling in CoAP/observe-based wireless sensor networks," *IEEE Internet of Things J.*, vol. 3, no. 6, pp. 923–936, 2016.
[20] G. Tanganelli, C. Vallati, E. Mingozzi, and M. Kovatsch, "Efficient proxying of CoAP observe with quality of service support," *Proc. IEEE World Forum on Internet of Things*, 2016, pp. 401–406.
[21] M. Kovatsch, O. Bergmann, and C. Bormann, "CoAP implementation guidance," Internet Engineering Task Force, Internet-Draft, July 2018. [Online]. Available: https://tools.ietf.org/html/draft-ietf-lwig-coap-06
[22] Y.C. Wang and C.T. Wei, "Lightweight, latency-aware routing for data compression in wireless sensor networks with heterogeneous traffics," *Wireless Comm. and Mobile Computing*, vol. 16, no. 9, pp. 1035–1049, 2016.
[23] V.L. Quintero, C. Estevez, M.E. Orchard, and A. Perez, "Improvements of energy-efficient techniques in WSNs: a MAC-protocol approach," *IEEE Comm. Surveys & Tutorials*, vol. 21, no. 2, pp. 1188–1208, 2019.
[24] Y.C. Wang, "Data compression techniques in wireless sensor networks," in *Pervasive Computing*. Hauppauge: Nova Science Publishers, 2012.
[25] D.P. Williamson and D.B. Shmoys, *The Design of Approximation Algorithms*. New York: Cambridge University Press, 2011.
[26] M. Koster, A. Keranen, and J. Jimenez, "Publish-subscribe broker for the constrained application protocol (CoAP)," Internet Engineering Task Force, Internet-Draft, September 2019. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-core-coap-pubsub/
[27] Y.C. Wang, "A two-phase dispatch heuristic to schedule the movement of multi-attribute mobile sensors in a hybrid wireless sensor network," *IEEE Trans. Mobile Computing*, vol. 13, no. 4, pp. 709–722, 2014.
[28] K. Kinoshita, N. Inoue, Y. Tanigawa, H. Tode, and T. Watanabe, "Fair routing for overlapped cooperative heterogeneous wireless sensor networks," *IEEE Sensors J.*, vol. 16, no. 10, pp. 3981–3988, 2016.
[29] M. Kovatsch, M. Lanter, and Z. Shelby, "Californium: scalable cloud services for the Internet of Things with CoAP," *Proc. IEEE Int'l Conf. the Internet of Things*, 2014, pp. 1–6.
[30] Y.C. Wang and K.C. Chen, "Efficient path planning for a mobile sink to reliably gather data from sensors with diverse sensing rates and limited buffers," *IEEE Trans. Mobile Computing*, vol. 18, no. 7, pp. 1527–1540, 2019.