Energy-Efficient Distributed Task Scheduling for Multi-Sensor IoT Networks

Elizabeth Liri, *Student Member, IEEE,* K. K. Ramakrishnan, *Fellow, IEEE,* Koushik Kar, *Senior Member, IEEE,*

***Abstract*—Multi-sensor IoT devices can gather different types of data by executing different sensing activities or tasks. Therefore, IoT applications are also becoming more complex in order to process multiple data types and provide a targeted response to the monitored phenomena. However, IoT devices which are usually resource-constrained still face energy challenges since using each of these sensors has an energy cost. Therefore, energy-efficient solutions are needed to extend the device lifetime while balancing the sensing data requirements of the IoT application. Cooperative monitoring is one approach for managing energy and involves reducing the duplication of sensing tasks between neighboring IoT devices. Setting up cooperative monitoring is a scheduling problem and is challenging in a distributed environment with resource-constrained IoT devices.**

In this work, we present our Distributed Token and Tier-based task Scheduler (DTTS) for a multi-sensor IoT network. Our algorithm divides the monitoring period (5 min epochs) into a set of non-overlapping intervals called tiers and determines the start deadlines for the task at each IoT device. Then to minimize temporal sensing overlap, DTTS distributes task executions throughout the epoch and uses tokens to share minimal information between IoT devices. Tasks with earlier start deadlines are scheduled in earlier tiers while tasks with later start deadlines are scheduled in later tiers. Evaluating our algorithm against a simple round-robin scheduler shows that the DTTS algorithm always schedules tasks before their start deadline expires.

***Index Terms*—IoT, distributed scheduler, tier-based, dynamic, deadline-aware, task, multi-sensor.**

I. Introduction

Many IoT devices are resource-constrained in terms of energy but still need to regularly send sensing data to the relevant IoT applications. Therefore, to maximize IoT device operating lifetime and meet IoT application sensing data requirements, energy-efficient solutions like cooperative monitoring are required. Cooperative monitoring saves energy by minimizing redundant data sent from neighboring IoT devices with overlapping coverage areas. Our strategy to implement cooperative monitoring is to develop a scheduling mechanism minimizing temporal sensing overlap. Therefore, we present DTTS, an energy-efficient distributed task scheduler for multi-sensor IoT devices.

This paper was produced by the IEEE Publication Technology Group. They are in Piscataway, NJ.

Manuscript received October 31, 2022; revised Jan 09, 2023 2022.

Elizabeth Liri and K. K. Ramakrishnan are with the University of California Riverside, Riverside, CA 92521 USA (email: eliri001@ucr.edu; kk@cs.ucr.edu

Koushik Kar is with the Rensselaer Polytechnic Institute, Troy, NY 12180 USA (koushik@ecse.rpi.edu).

Multi-sensor IoT devices utilize multiple sensors to monitor different environmental phenomena in a variety of applications like agriculture, smart cities, and disaster management including forest fires and hurricanes. Analyzing multiple data types from multi-sensor IoT devices requires more complex IoT applications but improves situational awareness, helping provide targeted responses to the monitored phenomena. For example, in agriculture, multi-sensor IoT devices can measure soil moisture, temperature, and capture images to observe crop health. Precision agriculture applications use this data to provide a response that maximizes the yield e.g., by adjusting the irrigation cycle.

Deploying IoT devices over large areas poses several challenges. First, most IoT devices are resource-constrained with limited memory, energy, and computing power. They typically rely on batteries and renewable energy sources, since brown power is not always accessible or cost-effective to provide. Therefore, careful energy management strategies are needed to ensure the IoT devices operate as long as possible (preferably always) while providing the data required by the IoT applications [1]. Secondly, the IoT device deployment pattern may have overlapping coverage areas. Therefore, it is important to avoid data redundancy when IoT devices monitoring the same location simultaneously transmit their sensor data (i.e. temporal sensing overlap) and thus avoid wasting precious energy. It can be very worthwhile to use cooperative monitoring when deploying many IoT devices. Cooperative monitoring reduces duplication of sensing tasks between neighboring IoT devices and helps better utilize the total available energy across all the IoT devices, by minimizing temporal sensing overlap.

Cooperative sensing for energy management is used in [2] which presents a distributed multi-sensor cooperative scheduling model for target tracking based on the partially observable Markov decision process. Another example is [3], which presents an IoT network cooperative power minimization scheme. Nodes receive task requests with estimated task execution times and schedule the tasks by scaling the CPU core's operating frequency ensuring task completion within the estimated time.

Implementing cooperative monitoring is a scheduling problem. A centralized scheduler may not be desirable in an IoT environment, from the point of view of resiliency and the potential need to frequently communicate every IoT device's state information to the central scheduler. However, setting up an energy-efficient distributed scheduler is challenging. First, IoT devices are resource-constrained and therefore can only store and process a limited amount of their neighbors' scheduling/state information. Next, since IoT devices are usually energy-constrained they typically limit communication to conserve energy. Therefore, schedulers requiring significant inter-device communication to share state may not be energy efficient. Lastly, as IoT devices go to sleep or become inactive, the network topology changes must be communicated to an IoT device, thus consuming energy. These challenges make designing a distributed task scheduling algorithm more complex than a centralized scheme like Earliest Deadline First, where all necessary information (deadlines) of all nodes is known in advance.

Our main contribution in this work is our Distributed Token and Tier-based task Scheduler (DTTS), a simple energy-efficient distributed scheduler for an IoT network. Our DTTS scheduler works with multi-sensor or single-sensor IoT devices and we refer to the monitoring period (duty cycle) as an epoch. Also, each IoT device sensor has a *start deadline*. This is the latest time (from the start of the epoch) that the sensor must begin its sensing activity (task) in order to have all measurements completed within the epoch. To minimize temporal sensing overlap, our algorithm divides each epoch into a set of non-overlapping intervals called tiers. Then, in a distributed manner and using tokens to share minimal information between the IoT devices, our DTTS algorithm schedules tasks with earlier start deadlines in the earlier tiers and tasks with later start deadlines in later tiers. Comparing DTTS against a simple periodic scheduler shows that DTTS always schedules each task before its start deadline expires.

Some examples of distributed task schedulers are in [2]–[6]. In [4], the authors use an energy neutrality constraint and dynamic programming to find a task schedule that maximizes the Quality of Service. The Lazy Scheduling Algorithm (LSA) [5] determines whether all task deadlines can be met before creating a schedule and uses task energy requirements, task deadlines, and current battery capacity of rechargeable IoT devices to make a scheduling decision. Jarvsis [6] is a distributed task scheduler that uses a hierarchy of control tasks operating in the Cloud/Fog to control robots and IoT devices on the ground.

The DTTS algorithm is based on [7] and is simple to execute unlike the dynamic programming approach in [4] or Markov decision process in [2]. DTTS does not require all task deadlines in advance like [5]. In DTTS each IoT device independently schedules its task execution. It is different from [6], where nodes higher in the hierarchy control task execution of IoT nodes lower in the hierarchy. In [3], task execution is triggered by requests from another node. While both DTTS and [3] are distributed and consider deadlines when scheduling, *Local* [3] requires each IoT device to also keep track of neighbor's deadlines. DTTS instead uses tokens for inter-device communication of minimal information, with independent decision-making at each IoT device.

II. System Design

A. Design Concepts

A task is when an IoT device performs a sensing activity using a particular sensor. For example, an IoT device with

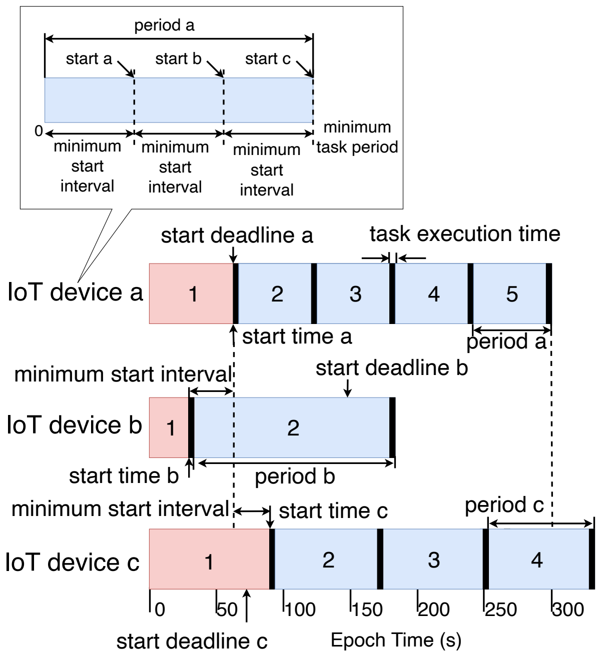

Fig. 1: *Figure illustrating design concepts.*

a camera and temperature sensors has image, video, and temperature tasks. A task can be executed multiple times per epoch. This number is referred to as the task parameter value and can change every epoch. For example, if the temperature task parameter value is 5, the temperature task is executed 5 times in the epoch.

Assuming task executions at a device are uniformly distributed in the epoch, the time interval between two consecutive task executions is the task *period*. Therefore, there is a time limit (from the start of the epoch) to start the first task execution so the last task execution can still be completed within the epoch. This time limit is the task's *start deadline* (i.e. task maximum start time) and its value is determined by subtracting the task *execution time* from the *period*.

Our algorithm sets the task *start time*, i.e., when the IoT device first executes the task in the epoch, to any value from 0 up to the task *start deadline*. Once the IoT device knows its *start time*, it automatically executes the task every *period* during the epoch until it reaches the required task parameter value (number of executions). For each task, we also record the *minimum* and *maximum task periods* within the IoT network.

Lastly, to minimize the temporal overlap we set a *minimum start interval* between the *start time* of the same task on any two IoT devices e.g. if the *minimum start interval* is 15s and the *start time* for an IoT temperature task is 60s, then the *start time* for the temperature task at the next IoT device must be 75s or later. Fig. 1 illustrates these key concepts using a temperature task at three IoT devices a, b, and c. The task execution time is the same in all cases (shown by the thick black vertical lines) and the epoch duration is five minutes. The task parameter values for a, b, and c are 5, 2, and 4 giving periods 60s, 150s, and 75s, respectively. The task *start time* is shown by the width of the red rectangle and takes any value from 0 to *start deadline*. Note that IoT devices with different *periods* have different *start deadlines*. In each case, the interval between consecutive task executions at a device is the task *period* (shown by the width of the blue rectangles). The *minimum task period* is $period\ a$ and the *maximum task period* is $period\ b$.

With a, the *start time* is equal to the *start deadline* and the task is executed five times within the epoch. With b, the *start time* is less than the *start deadline* so the last time the task is executed is much earlier in the epoch. With c, the *start time* exceeds the *start deadline* so when the task is executed for the last time, it is not within the epoch which is unacceptable.

The call-out above a in the figure shows how the *minimum start interval* is determined. Since the *minimum task period* is 60s we assume the temperature task on all IoT devices must be scheduled from 0-60s. Evenly distributing the task *start times* from all 3 IoT devices within 60s means the interval between each *start time* is 20s i.e. the *minimum start interval*. Therefore, the interval between the task *start time* at two different IoT devices must be at least the *minimum start interval* (compare start times at a, b, and c).

B. IoT Monitoring Environment

Our design is considered in the context of an IoT monitoring solution we developed, and the important characteristics in terms of tasks and monitoring environment requirements.

1) SEMA: An Energy Efficient Multi-Sensor IoT Monitoring Solution: One key aspect required for our DTTS scheduling algorithm is knowing how many times a task must be executed per epoch i.e. the task parameter value. DTTS relies on existing algorithms to determine these values per epoch at the IoT device and one example algorithm is SEMA.

We use SEMA [8] to provide the task parameter values for several reasons. First, it is a solution we have built that includes a practical low-cost and multi-sensor IoT device, therefore we can perform on-device experiments to test the scheduler's performance. Next, SEMA includes two task adaptation algorithms of which one is distributed and can be executed quickly on the IoT device itself. Therefore, together with our DTTS scheduler, this provides a distributed energy management solution. Finally, it adapts the sensor task parameter values every epoch based on the available energy and thus responds quickly to energy changes in the device/environment.

SEMA is a Smart Energy Management Solution for IoT Applications and runs on the SEMA Stick hardware unit which is a prototype IoT device. The key hardware components include a solar panel, lithium-ion battery, a Raspberry Pi Zero W with a camera, and an embedded microcontroller with multiple sensors. The SEMA Stick can run 5 tasks: temperature, humidity, soil moisture, image, and video and the energy cost for each task was modeled based on a single variable parameter e.g. the temperature task uses number of task executions per epoch.

For each SEMA Stick, the SEMA algorithm uses a heuristic optimization, total available energy (from the battery and solar source), and system models to determine the appropriate parameter values for each task every epoch. SEMA maximizes the information utility but also ensures that there is sufficient

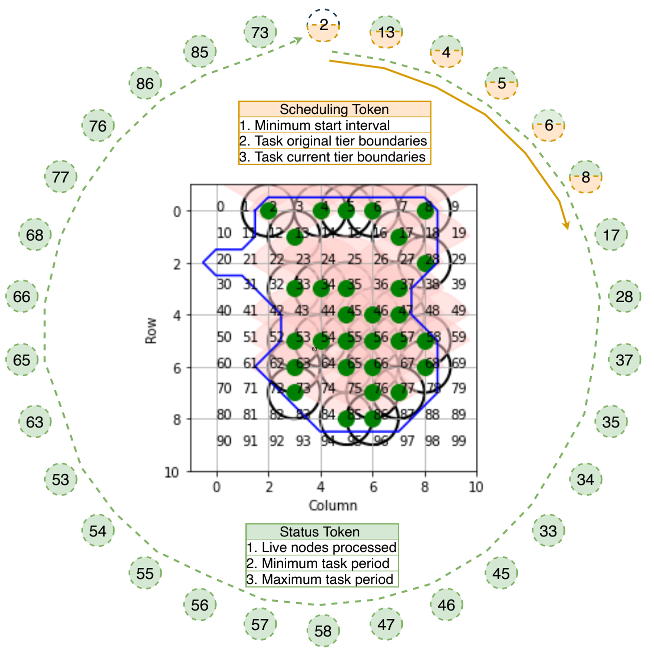

Fig. 2: *Example deployment (physical and logical network).*

battery energy for the SEMA Stick to operate overnight till the battery can be recharged the next day. In deployments over large areas with multiple SEMA Sticks, the devices may experience different local conditions that affect the amount of solar energy their solar panel receives, for example, there may be obstructions like clouds, trees, or buildings. The SEMA algorithm caters for such scenarios and therefore in the same epoch, may generate different task parameter values for each SEMA Stick. In this work, we focus on the independent temperature task which has a fixed execution time, and can be executed independently of any other task on the same SEMA Stick. We also assume SEMA determines the task parameter values.

2) Design of a Multi-Sensor IoT Monitoring Environment: **Leader Node** We assume the network has a leader IoT device which may change between epochs and is determined using existing leader selection algorithms.

Inter-device Communication We assume that IoT devices communicate with each other using a circular Distributed Hash Table (DHT)-like network whereby all IoT devices know the address of their nearest live neighbor [9]. Fig. 2 shows an example deployment with 30 randomly placed IoT devices in a deployment area covered by numbered grid points. The central inner figure represents the physical network and the sensing coverage areas for the temperature task (circular) and the video/image tasks (conical) are illustrated by the blue circles and red sectors respectively. The outer circle of nodes represents the logical network where the leader node (2) is white and we assume the IoT device ID is the grid point number where it is located. In this figure, the tokens travel in a clockwise direction and we show the contents of the Status and Scheduling tokens. The figure shows the protocol progress, with the Status token having been processed completely by all the nodes (shown by the dashed node borders). The Scheduling token has been processed by the first 6 nodes (shown by the orange color) and is sent to the next node (17) in the deployment.

SEMA currently uses WiFi for communication. However, since we are parsimonious in using the communication for only uploading sensor results and strive to minimize the communication between devices, we also explore the use of LoRaWAN (Long Range WAN), a Low Power Wide Area Network (LPWAN) technology. LoRa consumes less power [10] compared to WiFi, albeit having lower bandwidth (250 Kbps vs. 54 Mbps for 802.11 b/g), but can be suitable for the DTTS system e.g. in [11] for data transmission at 10Kbps over 50 meters distance between transmitter and receiver, WiFi uses 100mW while LoRaWAN uses 20mW. Another potential advantage for LoRAWAN is its longer range (few Kms for LoRA compared to a few 100 meters for WiFi), which permits larger-scale deployments.

Tokens With DTTS, each IoT device does not need to know the sensing schedule of other IoT devices. DTTS strives to transmit a minimal amount of data between IoT devices, allowing each IoT device to schedule its tasks in a completely distributed manner. However, generating the data to be shared requires gathering some information across the entire IoT network. An efficient way to first gather this information and then share minimal data between IoT devices is through tokens. Token passing is a well-understood technology (e.g., IEEE 802.5 [12], FDDI [13]). Challenges like handling lost/duplicate tokens are easily resolved using existing techniques where leader nodes handle token generation and initiate token recovery after failure by using timers [14], [15].

III. Scheduler Design

Given the task parameter value for an individual sensor on a particular IoT device, the key design goal of DTTS's scheduler is to distribute the task execution across the epoch and to minimize the temporal overlap from execution of the same sensing task across multiple IoT devices in the vicinity of each other.

We ensure that on each IoT device, the first time the task is executed per epoch i.e. the task *start time*, is before the *start deadline*.

This is accomplished by using tiers in DTTS. Our algorithm divides the epoch time into a given number of non-overlapping intervals called tiers. Tasks are then scheduled within tiers based on their *start deadline*. This ensures that tasks with earlier start deadlines are scheduled before tasks with later deadlines. The number of tiers required is provided in advance and is at least one. Each tier has a lower and upper boundary and the first tier (Tier 1) is always from 0-*minimum task period*. The remaining time interval from *minimum task period* to *maximum task period* is then divided equally into the remaining tiers. In this work, we use only two tiers so Tier 2 runs from *minimum task period* to *maximum task period.*

Our algorithm also minimizes inter-IoT device communication costs by piggy-backing minimal meta-data in tokens that are passed around between the IoT devices arranged in a logical ring.

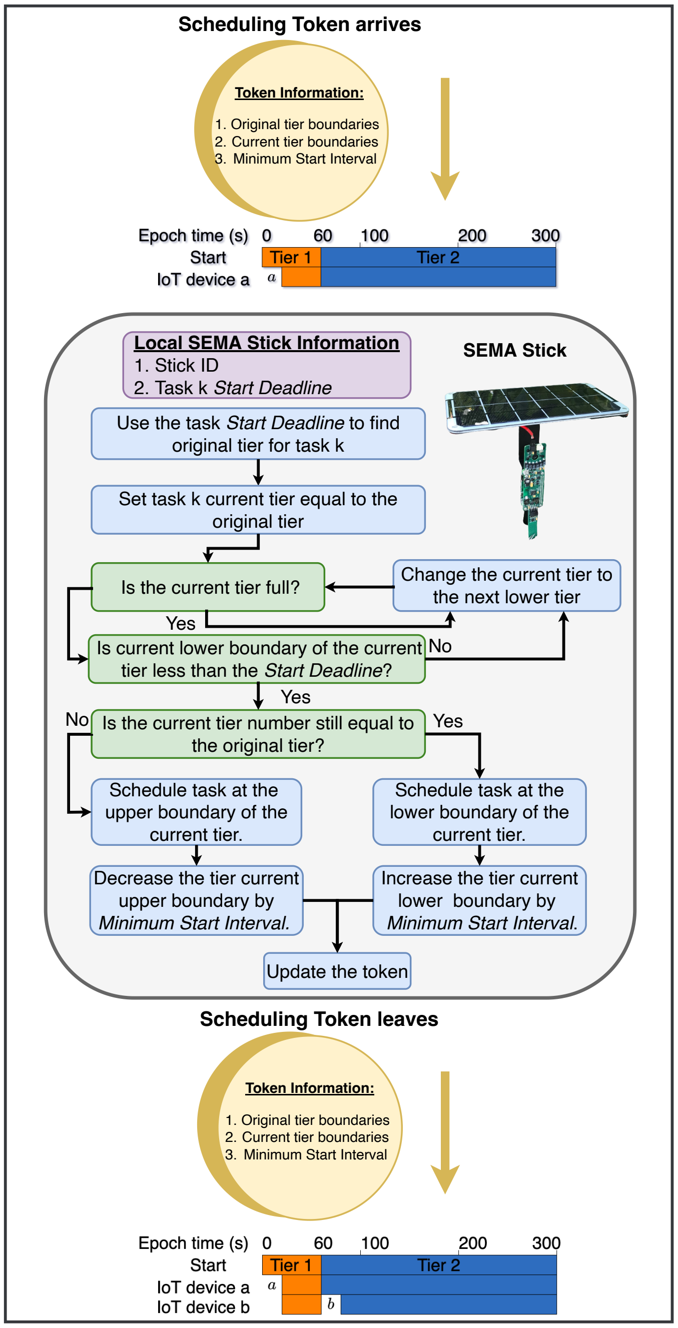
Scheduling Token arrives
Token Information:
1. Original tier boundaries
2. Current tier boundaries
3. Minimum Start Interval
Epoch time (s) 0 60 100 200 300
Start
IoT device a
Tier 1
Tier 2
a
Local SEMA Stick Information
1. Stick ID
2. Task k Start Deadline
SEMA Stick
Use the task Start Deadline to find original tier for task k
Set task k current tier equal to the original tier
Is the current tier full?
Change the current tier to the next lower tier
Yes
Is current lower boundary of the current tier less than the Start Deadline?
No
Yes
No
Is the current tier number still equal to the original tier?
Yes
Schedule task at the upper boundary of the current tier.
Schedule task at the lower boundary of the current tier.
Decrease the tier current upper boundary by Minimum Start Interval.
Increase the tier current lower boundary by Minimum Start Interval.
Update the token
Scheduling Token leaves
IoT device b
b

Fig. 3: *DTTS algorithm for scheduling round*

In terms of operation, the DTTS algorithm relies on two rounds of token passing each epoch traversing through the network of IoT devices to schedule the start time for each sensing task on each IoT device. The leader IoT node first generates a *status token* that traverses the network and gathers key information like the *minimum and maximum task periods* from all the IoT devices before returning to the leader node. The leader node first processes the information to determine key parameters such as the tier boundaries. It then generates a *scheduling token* which traverses the network and sets the task *start time* for all the tasks at each IoT device it passes through.

A. Status round

The first status round token is generated by the leader node to carry three key pieces of critical information that are updated as the token travels between IoT devices. The first is the *number of live nodes* that have processed the token so far in the Status round and the next two are the *minimum task period* and *maximum task period* for each task seen so far. When the token returns to the leader node, its information is used to determine the tier boundaries and the *minimum start interval*.

The tier boundaries are determined as described earlier by using the *minimum task period*, *maximum task period*, and the specified number of tiers. Given that we have an IoT device in the network reporting a *minimum task period*, we need to set the *minimum start interval* to be small enough that all the devices start their tasks within this *minimum task period*. This guarantees that no (*start deadline*) will be violated. The *minimum start interval* is calculated by dividing the *minimum task period* by the total number of active IoT devices (see Fig. 1). Once the *minimum start interval* and the tier boundaries have been determined, the scheduling token can then be generated.

B. Scheduling round

The key steps of the scheduling round algorithm for independent tasks are shown in Fig. 3. The leader node generates a scheduling token, that carries three key pieces of information per task: the *minimum start interval*, the *original tier boundaries*, and the *current tier boundaries*.

When the scheduling token arrives at an IoT device, DTTS checks the device's task *start deadline* and by looking at the task's *original tier boundaries* it determines which tier this *start deadline* is in.

If the tier is full i.e., no available slots for scheduling tasks, then DTTS finds the next lowest tier that is not full. When it finds a tier that is not full, DTTS checks whether the *current lower boundary* of the tier is lower than the device's *start deadline*. If so, then we set the task *start time* to the *current lower boundary* value because this means the task can be scheduled at this time without violating its *start deadline*. DTTS then updates the tier's *current lower boundary* value in the token, increasing it by the *minimum start interval*. If however, the tier's *current lower boundary* is higher than the IoT device's *start deadline* then the task cannot be scheduled in that tier without violating the *start deadline*. In this case, DTTS moves to the next lower tier and schedules the task at the *upper boundary* of this lower tier. DTTS then updates the tier's *current upper boundary* value in the token, decreasing it by *minimum start interval*.

DTTS repeats this scheduling process for all device tasks before forwarding the token to the next IoT device. By scheduling tasks at tier boundaries and updating the current boundaries in the token, DTTS schedules tasks in an entirely distributed manner with minimal information being communicated between IoT devices. More information is in [7]. Given T tiers and K tasks, DTTS complexity is $O(TK)$.

IV. Example

We now describe the operation of DTTS using an example with four IoT devices. Additionally, the evaluation in Section V is with 30 devices. Consider a network with four IoT devices a, b, c, and d running a temperature task with a 1 second task execution time. The table in Fig. 4 shows the task parameter values, *periods*, and *start deadlines* for a 5-minute epoch. For example, the task parameter value (number of measurements per epoch) for a is 5 and therefore has a 60s *period* with a *start deadline* of 59s. By comparing all the periods, the *minimum task period* is 60s, the *maximum task period* is 300s, and since there are four devices, the *minimum task period* is 60s. If we have 2 tiers, then Tier 1 runs from 0-60s, and Tier 2 runs from 60-300s.

Fig. 4 illustrates how the tasks are scheduled on the four IoT devices. The blue and orange areas indicated slots in Tier 1 and Tier 2, respectively, that are available for scheduling while the white slots already have tasks scheduled. As DTTS schedules the temperature task at each additional device, the tier boundaries are moved progressively inwards.

If a is the leader node it is the first to be scheduled. Since its *start deadline* is 59s, we see that from the original tier boundaries this is in Tier 1 which is not full. The *current lower boundary* of Tier 1 is 0 (less than 59s), so we schedule the task at the beginning of the epoch (i.e. at 0 seconds). We then move the Tier 1 *current lower boundary* value up by 15s (the *minimum start interval*) in the token before forwarding it to b.

At b, its *start deadline* is 149s which is in Tier 2 (from checking the task *original boundaries*). The *current lower boundary* of Tier 2 is 60s (less than 149s) so we can schedule the task at 60s without violating the *start time* deadline. The Tier 2 *current lower boundary* is increased by 15s to 75s in the token, before forwarding it to c.

At c, its *start deadline* is 299s which is in Tier 2 (from the *original boundaries*). Similar to the case with b, we can schedule this task at the *current lower boundary* of Tier 2. The *current lower boundary* of Tier 2 is updated from 75s to 90s in the token before it is forwarded to d.

At d, its *start deadline* is 74s which according to the *original tier boundaries* is in Tier 2. However, the *current lower boundary* of Tier 2 is 90s, which is beyond this task *start deadline*. Therefore, in this case, we fall back to the next lower tier (Tier 1) and schedule the task at the *current upper boundary* of Tier 1. The *current upper boundary* of Tier 1 is then reduced by 15s to 45s. Since d is the last IoT device the token returns to the leader node.

V. Results

The results from experiments done via simulations show that our DTTS algorithm schedules tasks with earlier *start deadlines* in earlier tiers and tasks with later *start deadlines* in later tiers. The results also show that DTTS always schedules the task *start time*, before the *start deadline* expires.

Using 5 and 15-minute epochs we evaluated the performance of our algorithm using simulations and 30 IoT devices. We assume the tokens travel between IoT devices based on

IoT ID	Task Parameter Values	Period (s)	Start Deadline (s)
a	5	60	59
b	2	150	149
c	1	300	299
d	4	75	74

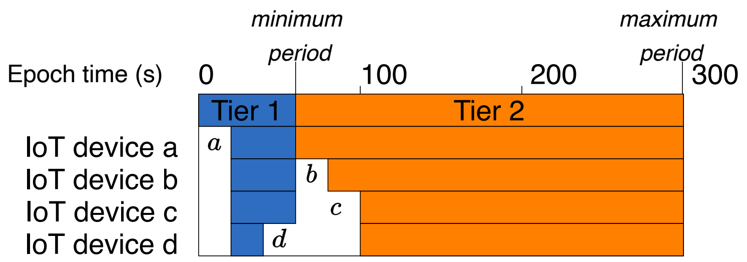

Fig. 4: *Temperature task scheduling example.*

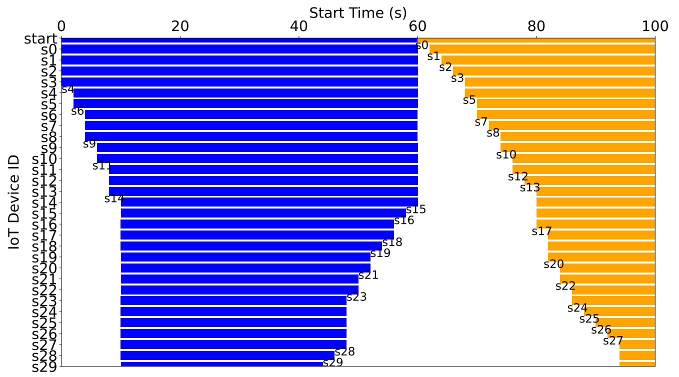

Fig. 5: *DTTS temperature task schedule for a sunny day*

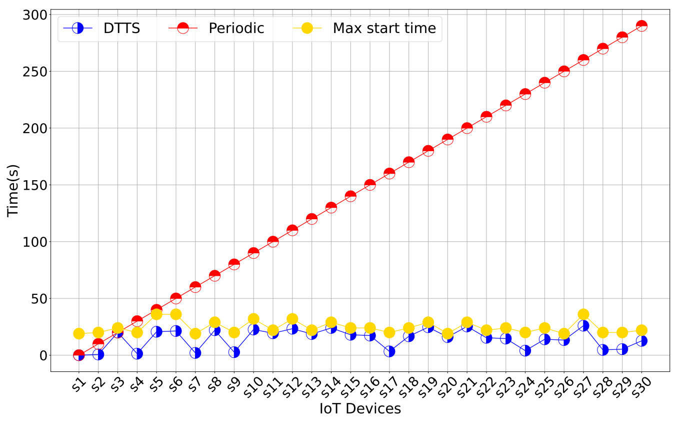

Fig. 6: *Comparing start deadline and start time with DTTS and the periodic schedulers*

increasing device IDs. Here we show the results for the temperature task only, while in [7] we considered tasks like image and video with different parameters e.g. duration, not just the number of executions per epoch. We used 3 different types of epochs to represent cloudy, sunny, and hybrid days. Based on the SEMA work, the maximum task parameter value for a temperature sensing task during a 5 min epoch is 5. Therefore on a sunny day, task parameter values range from 3-5 due to sufficient energy to charge the IoT device and execute more sensing activities. On a cloudy day, task parameter values range from 1-2 since the IoT device is conserving energy due to low solar to recharge the battery. On a hybrid day, task parameter values range from 1-5 since the solar energy varies.

For each day type, we generated the task parameter values for all the IoT devices and then ran the DTTS scheduler to schedule all tasks. This was done multiple times and the results from the 5-minute epoch are presented in Fig. 5 for one sample schedule on a sunny day.

On a sunny day, higher task parameter values correspond to a shorter task *period* and smaller tiers for the temperature sensing task. From Fig. 5, we see Tier 1 is from 0-60s and Tier 2 is from 60-100s. Task *start times* are set to earlier in the epoch, i.e., all task *start times* are set within the first 100s even though the epoch duration is 300s.

On a cloudy day, however, lower task parameter values correspond to a longer task *period* and larger tiers so tasks are spread throughout the entire epoch.

We also compare our DTTS scheduler with a simple round-robin periodic scheduler which calculates its *minimum start interval* by dividing the total epoch time by the number of live IoT devices. All the IoT devices are then scheduled one after another with the *start times* separated by an interval of *minimum start interval* seconds.

Fig. 6 compares the scheduled *start times* and *start deadlines* between DTTS and the periodic scheduler at the IoT devices during a sunny epoch. The results show that DTTS always sets the *start time* before the *start deadline* expires while the periodic scheduler frequently exceeds that deadline.

If multiple devices in a deployment area simultaneously send their data to the sink (due to independent scheduling), there is a high risk of congestion loss at the sink, resulting in retransmissions by IoT devices, potentially wasting energy. In addition, if multiple devices have overlapping coverage areas, simultaneous sensing results in duplicate data, wasting device energy and unnecessary processing upstream for deduplication. DTTS mitigates this by coordinating the scheduling across all devices, reducing redundant data, reducing simultaneous transmission, and minimizing the need for retransmissions. DTTS is thus more energy efficient compared with independent scheduling.

VI. Conclusion

Cooperative monitoring among multiple IoT devices helps manage their energy consumption while monitoring large physical areas. Our Distributed Token and Tier-based task Scheduler scheduling protocol (DTTS) presented here is an energy-efficient distributed scheduler suitable for an IoT network. Using a simple protocol with minimal information sharing between IoT devices, DTTS works with multi-sensor IoT devices utilizing *start deadlines* to distribute task *start* *times* every epoch to minimize temporal overlap. Experiments show DTTS always schedules the IoT device's task start time before its start deadline expires.

References

[1] F. K. Shaikh and S. Zeadally, "Energy Harvesting in Wireless Sensor Networks: A Comprehensive Review.", *Renewable and Sustainable Energy Reviews*, vol. 55, 2016, pp. 1041-1054.

[2] Z. Zhang et al., "Research on Distributed Multi-Sensor Cooperative Scheduling Model Based on Partially Observable Markov Decision Process." *Sensors*, vol. 22, 2022, no. 8, pp. 3001.

[3] S. R. Sarangi, S. Goel, and B. Singh, "Energy Efficient Scheduling in IoT Networks", in *Proc. ACM SAC'18*, 2018, pp. 733-740.

[4] A. Caruso, et al., "A Dynamic Programming Algorithm for High-Level Task Scheduling in Energy Harvesting IoT", *IEEE IoT Journal*, vol. 5, no. 3, 2018, pp. 2234-2248.

[5] C. Moser, J. J. Chen, and L. Thiele, "Dynamic Power Management in Environmentally Powered Systems", *IEEE ASP-DAC*, IEEE, 2010, pp. 81-88.

[6] M. De Benedetti, et al., "JarvSis: A Distributed Scheduler for iIoT Applications.", *Cluster Computing*, vol. 20, no. 2, 2017, pp. 1775-1790.

[7] E. Liri, K. K. Ramakrishnan, and K. Kar, "A Renewable Energy-Aware Distributed Task Scheduler for Multi-Sensor IoT Networks.", *Proc. ACM SIGCOMM Workshop on NET4us'22*, 2022, pp. 26-32.

[8] E. Liri, et al, "An Efficient Energy Management Solution for Renewable Energy Based IoT Devices.", *Proc. ICDCN'23*, IEEE, 2023, pp. 20-27.

[9] I. Stoica, et al, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications.", *IEEE/ACM ToN*, vol. 11, no. 1, 2003, pp. 17-32.

[10] M. Afaneh, "Wireless Connectivity Options for IoT Applications - Technology Comparison.", 2020, https://www.bluetooth.com/blog/wireless-connectivity-options-for-iot-applications-technology-comparison/, accessed Sept, 23 2022.

[11] Voler Systems, "Which wireless standard makes sense for your application?", 2022, https://www.volersystems.com/guide-wireless-technology, accessed Feb 01, 2023.

[12] ANSI, "Local Area Networks: Token Ring Access Method and Physical Layer Specifications–802.5", John Wiley & Sons, Inc., 1985.

[13] M. S. Kingley, "ANSI Fiber Distributed Data Interface (FDDI) Standards", Feb 1996, https://tinyurl.com/2yu9x25r, accessed June 26, 2022.

[14] H. Yang, and K. K. Ramakrishnan. "A Ring Purger for the FDDI Token Ring.", *[1991] Proc. 16th Conf. on Local Computer Networks*, IEEE Comput. Soc., 1991, pp. 503-504.

[15] M. J. Johnson, "Reliability Mechanisms of the FDDI High Bandwidth Token Ring Protocol.", *Computer Networks and ISDN Systems*, vol. 11, no. 2, 1986, pp. 121-131, 1986.

VII. Acknowledgement

This work was supported by the US National Science Foundation grant CNS-1818971 and the US Dept. of Commerce, NIST PSIAP award 70NANB17H188.

VIII. Biography Section

Elizabeth Liri [Student Member, IEEE](eliri001@ucr.edu) is currently working towards a Ph.D degree in Computer Science at the University of California, Riverside, USA. Her research interests include developing communication protocols for the Internet of Things to improve energy efficiency and network resiliency and integration of IoT with edge computing.

K. K. Ramakrishnan [Life Fellow, IEEE](kk@cs.ucr.edu) is a Distinguished Professor at the University of California, Riverside. Earlier, he was at Digital Equipment Corporation, and then AT&T Labs-Research. He is an ACM and AT&T Fellow. He has an MTech. from IISc, India, and MS, Ph.D. from University of Maryland, College Park.

Koushik Kar (Senior Member, IEEE)(kk@cs.ucr.edu) received his Ph.D. in electrical and computer engineering from the University of Maryland at College Park in 2002 and has been a faculty member at Rensselaer Polytechnic Institute since then. His primary research expertise is in developing and analyzing low-complexity optimization algorithms for computer networks.