A Renewable Energy-Aware Distributed Task Scheduler for Multi-sensor IoT Networks

Elizabeth Liri
University of California Riverside
Riverside, CA
eliri001@ucr.edu

K. K. Ramakrishnan
University of California Riverside
Riverside, CA
kk@cs.ucr.edu

Koushik Kar
Rensselaer Polytechnic Institute
Troy, NY
koushik@ecse.rpi.edu

ABSTRACT

IoT devices are becoming increasingly complex, support multiple sensors, and often rely on batteries and renewable energy. Scheduling algorithms can help to manage their energy usage. When multiple devices cooperatively monitor an environment, scheduling sensing tasks across a distributed set of IoT devices can be challenging because they have limited information about other devices, limited energy, and communication bandwidth. In addition, sharing information between devices can be costly in terms of energy.

Our Tier-based Task scheduling protocol (T2), is an energy-efficient distributed scheduler for a network of multi-sensor IoT devices. T2, adapting on an epoch-by-epoch basis distributes task executions throughout an epoch to minimize temporal sensing overlap without exceeding task deadlines. Our experiments show that T2 schedules an IoT device's sensing task start time before its deadline expires. When compared against a simple periodic scheduler, T2 schedules are closer to the optimal centralized EDF scheduler.

CCS CONCEPTS

• **Networks** → **Network protocol design**;

KEYWORDS

IoT, distributed scheduler, tier-based, dynamic, deadline-aware

ACM Reference Format:
Elizabeth Liri, K. K. Ramakrishnan, and Koushik Kar . 2022. A Renewable Energy-Aware Distributed Task Scheduler for Multi-sensor IoT Networks . In *ACM SIGCOMM 2022 Workshop on Networked Sensing Systems for a Sustainable Society (NET4us '22), August 22, 2022, Amsterdam, Netherlands.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3538393.3544936

This work is licensed under a Creative Commons Attribution-ShareAlike International 4.0 License.

NET4us '22, August 22, 2022, Amsterdam, Netherlands
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9392-8/22/08.
https://doi.org/10.1145/3538393.3544936

1 INTRODUCTION

Recently developed IoT devices have more processing and communications functionality. IoT applications are also becoming more complex. When deployed over large remote areas, IoT devices rely on batteries and renewable energy sources [15],[10],[1] and so need careful energy management. When deploying many IoT devices, it is worthwhile to use cooperative monitoring. This ensures sensing tasks are not duplicated between neighboring devices, especially in terms of ensuring minimal temporal overlap between IoT devices performing the same sensing task. This helps better utilize total available energy across all devices.

Setting up cooperative monitoring is a scheduling problem but, scheduling tasks in a distributed environment is challenging. IoT devices have limited information so do not know when the neighbor will be scheduled, inter sensor communication is costly in terms of energy and the network topology changes as IoT devices go to sleep or become inactive due to lack of energy. Therefore designing a distributed task scheduling algorithm is more complex than other centralized scheduling algorithms like Earliest Deadline First (EDF) where deadlines of all nodes are known in advance.

IoT task scheduling algorithms include Lazy Scheduling Algorithm (LSA) [12] designed for rechargeable IoT devices which have task deadlines and [14] a cooperative power minimization scheme for an IoT network that saves energy along the data path to the sink. Our algorithm saves energy at the device, is simple and considers task deadlines like [12].

Our Tier-based Task scheduling protocol (T2), is an energy efficient distributed scheduler for an IoT network. T2 works with multi-sensor IoT devices which can perform sensing activities (tasks) either independently or with some dependencies. The T2 algorithm distributes task executions throughout the epoch as much as possible to minimize temporal sensing overlap. T2 divides the monitoring epoch (e.g., 5 min.) into a set of non-overlapping periods called tiers. T2 finds the starting deadline for each task then schedules tasks with shorter deadlines in earlier tiers and tasks with longer deadlines in later tiers in a distributed manner. Only minimal information is communicated between the IoT devices.

The performance of T2 was compared against a simple periodic scheduler that uniformly distributes task start-times

Corrected Version of Record. V.1.1. Published October 14, 2022.

within the epoch and then allocates start times to IoT devices in a round robin manner. Experimental results show that T2 always schedules an IoT device's task start time before its deadline expires and the schedule order generated using T2 is generally closer to the optimal than the periodic scheduler.

2 RELATED WORK

Distributed task scheduling in IoT networks can improve energy efficiency. Jarvsis [5] is a distributed scheduler that executes multiple tasks on IoT and robotics applications. It uses a hierarchy of control tasks operating in the Cloud/Fog to control robots and IoT devices on the ground. [4] uses dynamic programming to find a task schedule that maximizes QoS with an energy neutrality constraint. Lazy Scheduling Algorithm (LSA) [12] is designed for rechargeable IoT devices. Depending on current battery capacity, task energy requirements and task deadlines it determines whether all task deadlines can be met or not. [11] manages time scheduling and power allocation for multiple modules.

Co-operative sensing mechanisms can also be used to manage energy and [17] presented a distributed multi-sensor cooperative scheduling model for target tracking based on the partially observable Markov decision process. [14] presents a cooperative power minimization scheme for an IoT network and proposed a distributed and a centralized algorithm.

In [5] device task execution is controlled by a device one level higher in hierarchy but the T2 algorithm only sets the task start time. Although [4] restricts the solution search space for their dynamic programming approach, their algorithm still has a pseudo-polyomial complexity which can have a significant impact on energy. LSA handles both energy and time domain constraints while the T2 algorithm handles time domain constraints, is simple and can easily run on an IoT device with limited compute resources.

T2 uses a cooperative sensing approach and can be used in any IoT environment unlike [17]. T2 saves energy at the IoT device itself not along the path from the IoT device to the sink like [14]. T2 also considers task deadlines unlike [11].

3 SYSTEM DESIGN

This section discusses the Tier-based Task scheduling protocol, (T2). The design principles include minimizing inter-IoT device communication costs by using tokens to share minimal data and limiting communication to neighbors, minimizing temporal overlap, distribution of tasks across the epoch and minimizing missed start time deadlines.

3.1 Task Characteristics and Deployment

For multi-sensor IoT devices, the operation of each sensor executing sensing activities is referred to as a task. In an agricultural environment for example, we may use multi-sensor IoT devices over a large deployment area and require all IoT devices to run all sensing tasks at least once in a given time period (epoch). If the IoT device uses batteries and a solar panel for renewable energy, then using solar prediction mechanisms e.g [3], [9] and other algorithms e.g [4], the optimum task parameter values to maximize energy efficiency can be determined. Due to the unique environmental conditions at each IoT device (e.g. shade or other obstacles), the renewable energy they receive may differ. Thus, the task values used at each IoT device may differ. Due to this, it is important to distribute task execution, across the set of devices, throughout the epoch, to minimize temporal overlap and maximize information utility. The T2 algorithm is designed to address this need. It works with multi-sensor IoT devices with tasks that are executed every epoch. This work assumes task parameter values per epoch have already been found.

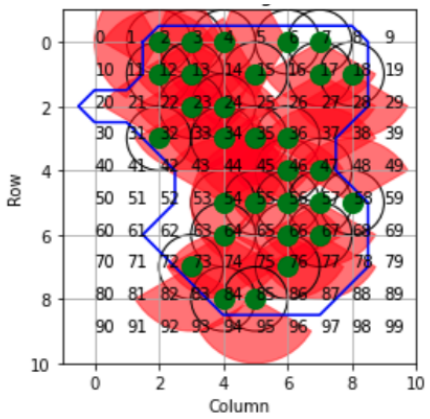

Figure 1: *Sample IoT device deployment in an area*

We use temperature and image tasks that vary the number of measurements/images taken per epoch (n_t and n_i respectively) and a video task that uploads 1 streaming video per epoch but varies its duration t_v. n_t, n_i and t_v range from 1-15, 1-30 and 5-30s respectively. Independent tasks e.g. temperature task can operate simultaneously with other tasks. Dependent tasks cannot operate simultaneously e.g. image and video tasks because they share the camera.

For inter sensor communication we assume a circular Distributed Hash Tables (DHT) like network where each IoT device knows its nearest live neighbor's address. Fig. 1 is an example of an IoT deployment area with numbered grid points, covered by 30 randomly deployed IoT devices. The sensing coverage areas for the temperature task (circular) and video/image tasks (conical) are shown. The T2 algorithm works in deployments with both limited and significant overlap as long as the IoT devices can communicate.

Table 1: Temperature task example.

Stick ID	Instances n_t	Period p^{temp} (s)	Token t^{temp}_{min}	Token t^{temp}_{max}
a	20	15.0	15.0	15.0
b	16	18.8	15.0	18.8
c	15	20.0	15.0	20.0
d	21	14.3	14.3	20.0

3.2 Scheduler Design

The T2 scheduler uses tokens to schedule a set of cooperating IoT devices in sequence to perform complementary monitoring across the IoT network, using two rounds. In the first round, *status round*, a single status token is generated by a predetermined IoT device (acting as a 'leader' or s_0). The status round gathers key information from all the IoT devices in the IoT network before the token returns to s_0. This information is used during the second round, *scheduling round* where a scheduling token generated by s_0, sets the start time for the different tasks as it circulates through the IoT network. In this work we assume the leader and last device s_N are known and challenges like handling lost or duplicate tokens are addressed using existing techniques such as those for token ring networks (IEEE 802.5) [6], [2] and FDDI [8], [13]. The primary need is to elect a leader node that is responsible for token generation, and ensuring failure recovery by use of timers (e.g., a 'target token rotation time'). The leader node detects lost and duplicate tokens and initiates token recovery after a failure [16], [7].

3.2.1 Scheduling Definitions. If task k in an IoT device is to be executed multiple times per epoch, each task execution is a *task instance*. Task instances are distributed in the epoch, separated from each other by a *task period* p^k determined by dividing the epoch in seconds by the number of task instances required by k. The number of task instances per task and per IoT device depends on factors such as IoT device's battery state, solar prediction and energy goals. These differ across IoT devices so the resulting individual periods also differ. The minimum and maximum task periods across the IoT network are t^k_{min} and t^k_{max} and are used to determine the tier boundaries. Regarding tiers, the epoch time is split into one or more non-overlapping time periods called tiers but the boundaries for Tier 1 are always from 0 to t^k_{min}. The parameter *task start time* t^k_s for task k is the time in seconds measured from the start of the current epoch to when the first task instance is executed. The *maximum task start time* t^k_{smax} is the maximum possible value of t^k_s to ensure all task instances complete execution before the end of the epoch and is calculated from Eqn.1.

$$t^k_{smax} = p^k - t^k_e \tag{1}$$

Where t^k_e is the task execution time. To illustrate these terms, Table 1 shows four IoT devices $a-d$ with various task instances n_t for the temperature task i.e. $k = temp$. If the epoch duration is 300s (5mins) then the task periods (given by $p^{temp} = epoch/n_t$) are in column 3. Columns 4 and 5 show how the final t^k_{min} and t^k_{max} are determined by comparing the current values to the current IoT device period value p^k. In this example $t^k_{min} = 14.3s$ and $t^k_{max} = 20s$.

Consider task c with 15 instances and task period $p^{temp} = 20s$. This period is the interval between instances after the first instance, see width of the blue rectangles in Fig. 2 which shows three possible scheduling options ($c_1 - c_3$) for task c. The thick black lines show the task execution time t^{temp}_e. The task start time t^{temp}_s is shown by the red rectangles. If the start time is greater than the max. task start time i.e. $t^{temp}_s > t^{temp}_{smax}$ (see c_3), then the last task instance (15) is executed after the epoch ends which is unacceptable since that data is needed at the sink to calculate the next epoch task values. For the last instance to complete execution within the epoch, the first task instance must be completed by p^{temp} (20s). If $t^{temp}_s \leq t^{temp}_{smax}$ it ensures all instances are completed within the epoch (see cases c_1 and c_2). To maximize temporal separation, a task's *minimum start interval* t^k_b is defined. It is the minimum time interval between the start times of task k on two IoT devices s_n and s_m when the scheduling token moves directly from s_n to s_m. Eq. (2) calculates t^k_b for independent tasks where n_{live} is the total number of live IoT devices during that epoch.

$$t^k_b = \frac{t^k_{min}}{n_{live}} \tag{2}$$

Our algorithm addresses both intra- and inter-device task dependency by first using task execution time as the minimum start interval i.e., $t^k_b = t^k_e$ to ensure tasks do not run simultaneously. Thus, the dependent tasks share a tier structure. The T2 algorithm schedules the dependent tasks individually. That is, on a single device, the T2 algorithm would first schedule the video task and then the image task, for example.

Fig. 3 and 4 show the relationship between start time t^k_s, task execution time t^k_e and min start interval t^k_b for independent and dependent tasks respectively. For independent tasks after the first task instance runs, the next instances are executed after t^k_bs giving a uniform task distribution in the epoch. We assume $t^k_e \leq t^k_b$ and for dependent tasks, and after the first task instance runs, start time of the next instance depends on the last set task time and its execution time.

If all the epoch timeslots have been scheduled but there are still IoT devices left to be scheduled then the T2 algorithm resets the tier boundaries back to the original values and continues scheduling the remaining tasks.

3.2.2 Status Round. The token carries data shown in Table 2. At each IoT device, t^k_{min} and t^k_{max} per task are updated as well as the total live nodes processed n_{live}. On receiving the token each IoT device also checks whether it is the last IoT device by comparing its ID s_{id} to the last IoT device ID,

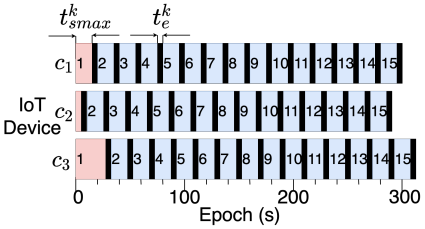

Figure 2: *Max. start time, t^k_{smax}, execution time t^k_e, and period p^k.*

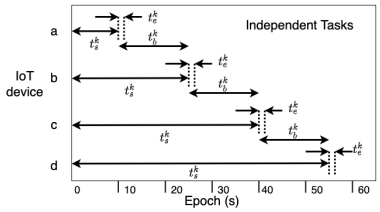

Figure 3: *Start time t^k_s and min. start interval t^k_b between independent tasks k.*

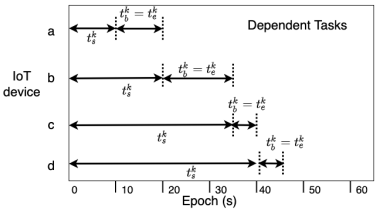

Figure 4: *Start time t^k_s and min. start interval t^k_b between dependent tasks k.*

Table 2: Token information.

Parameter type	Description
Status token	
s_0	ID/IP of first stick that generates tokens
s_N	ID/IP of last stick that will update token
t^k_{min} and t^k_{max}	Minimum and maximum period seen so far for task k, k=[1,K]
n_{live}	Total number of live sticks the token has traversed so far
Scheduling token	
s_N	ID/IP of last IoT device that will update token
olb^k_1:oub^k_1, ..., $olb^k_{n_{tiers}}$:$oub^k_{n_{tiers}}$	Task k tier structure with original tier boundaries ($k \in K$)
clb^k_1:cub^k_1, ..., $clb^k_{n_{tiers}}$:$cub^k_{n_{tiers}}$	Task k tier structure with current tier boundaries ($k \in K$)

s_N. If the ID is not equal, the token is forwarded to the next IoT device otherwise it returns to s_0. When s_0 receives the returned token it determines the minimum start interval t^k_b for the independent tasks using Eq. (2). This allows T2 to uniformly distribute the start time of task k on all n_{live} IoT devices within t_{min} (i.e. in Tier 1) in the worst case.

Once t^k_b has been found the tiers are generated. The total number of tiers $n^k_{tiers} \geq 1$ is given in advance. Tier 1 always runs from 0 to t^k_{min}.The remaining epoch time is split equally into the remaining $n_{live} - 1$ tiers. Dependent tasks share a tier structure where $t^{k'}_{min}$ and $t^{k'}_{max}$ are the min. and max. of the dependent task t^k_{min} and t^k_{max} values. For example if tasks $k1$ and $k2$ are dependent, then $t^{k'}_{min}$ used for the shared tier structure is $t^{k'}_{min} = min(t^{k1}_{min}, t^{k2}_{min})$ and $t^{k'}_{max} = max(t^{k1}_{max}, t^{k2}_{max})$. Once the tier boundaries have been determined then the scheduling token is generated.

3.2.3 Scheduling Round. Information carried in the scheduling token is in Table 2. The last IoT device ID s_N is needed so the token is not forwarded beyond it. Data for every tier includes its ID, current tier $ctier$, original lower and upper boundary values (olb^k_{ctier} and oub^k_{ctier}) and current lower and upper boundary values (clb^k_{ctier} and cub^k_{ctier}), where $ctier$ is the tier we are trying to schedule the task in. Tier structures for tasks includes all tier data.

As the token travels between IoT devices each device executes the scheduling algorithm (Algorithm 1) and only the current tier boundaries in the token are updated. The algorithm uses the task max. start time t^k_{smax} and checks all the original tier ($otier$) boundaries to determine which tier the task should be scheduled in. The task current tier $ctier$ is then set to $otier$ and the algorithm then checks if $ctier$ is full by comparing its current boundaries i.e., if current tier width (difference between upper and lower tier boundary) is less than the minimum start interval between consecutive IoT devices (t^k_b). As more tasks are scheduled in a tier, the tier width reduces. For independent tasks t^k_b is fixed so the tier width is 0 when the tier is full. Dependent tasks set t^k_b to be the task execution time and this may vary, e.g., video task varies its duration so the task boundaries are not updated by a fixed value at every node. If the tier's current width is less than the execution time for the task then from the point of view of that task, the tier is 'full'.

If $ctier$ is full then the algorithm checks the next lower tiers, one at a time, till it finds a tier that is not full. That tier then becomes the new $ctier$. If $ctier = 1$ (i.e. the first tier) is full, then the algorithm resets all tier boundaries in the token to their original values and then tries to schedule this task again. Previously scheduled tasks keep their assigned slots however from the point of view of this IoT device, the tiers are now 'empty' and it schedules its task in its original tier. The remaining unassigned IoT devices also use the algorithm with the reset tier boundaries to schedule their tasks. This means that there may be some start time overlap, i.e., some task start times scheduled with the reset boundaries and those scheduled before the reset may be the same. In this work we assume that in such cases, the spatial distance between such IoT devices with start time overlap is large enough that they can be scheduled at the same time. Future work involves checking the spatial distance between such devices to confirm the distance is large enough.

If $ctier$ is not full, then the task is scheduled at the lower boundary of that tier if the current lower boundary is less than t^k_{smax} or if the original tier is Tier 1 i.e., $otier = ctier = 1$. In both these cases, scheduling the task at the lower boundary will not violate the task t^k_{smax}. Otherwise the task is scheduled at the upper boundary of the next lower tier. Once a task is scheduled, the current tier boundaries in the token are updated i.e., lower boundary clb^k_{ctier} and upper boundary cub^k_{ctier}, and the token is forwarded to the next IoT device. By scheduling tasks at the tier boundaries and updating the current boundaries, tasks are scheduled entirely in a distributed

Algo 1: Task scheduling of task k in epoch-final

Result: Task k scheduled at $startTime \leq maxStartTime$.

Initialize: $maxStartTime = Period - executionTime$;
$done$=FALSE; $taskType = taskType$
if $taskType$="Dependent" **then**
    $minStartInterval = executionTime$
$originalTier$=getOriginalTier($maxStartTime$)
$currentTier = originalTier$
while !done
    if isFull($currentTier$)=True
        if $currentTier$=1 **then**
            resetTierBoundaries()
            $currentTier = originalTier$
        else $currentTier = currentTier - 1$
    else
        if $currentTier$! = $originalTier$
            scheduleTask($upperBoundary$)
            $done$=TRUE
        else
            if $currentTier = 1$
                scheduleTask($lowerBoundary$)
                $done$=TRUE
            else
                if getLowerBoundary($ctier$)≤ $maxStartTime$
                scheduleTask($lowerBoundary$)
                $done$=TRUE
            else $currentTier = currentTier - 1$

way. The final schedule graph looks like multiple tiers being placed side by side. As each additional IoT device is scheduled, it moves a tier's boundaries progressively inwards (see white area in Fig. 5 in evaluation section).

The T2 algorithm schedules tasks with no overlapping start times and within each task's maximum start deadline t^k_{smax} if $n_{sched} t^k_b < t^k_{min}$ for independent tasks or $\sum_{n=1}^{n_{sched}} t^k_b < t^k_{min}$ for dependent tasks. Where n_{sched} is the total IoT devices scheduled so far, and t^k_b is a fixed value for independent tasks.

3.3 Proof of Correctness

Using independent tasks with a fixed t^k_b as an example, we prove by induction that as long as the total time scheduled so far is less than the minimum network period, Algorithm 1 can schedule all tasks without overlapping start times and without exceeding any task's start time deadline.

Theorem 3.1. *If $nt^k_b \leq t^k_{min}$ and t^k_b is a fixed value, Algorithm 1 will schedule task k for n IoT devices such that every t^k_s value is unique, the difference between any two t^k_s values is $\geq t^k_b$ and $t^k_s \leq t^k_{smax}$ for all n IoT devices.*

Proof. We prove by induction over n from n_0 to N.

Base step: At $n = n_0$ we schedule task k for only 1 node thus $n_{tier} = 1$ with boundaries $clb^k_1 = 0$ and $cub^k_1 = t^k_{min}$ since $t^k_{min} = t^k_{max} = p^k_{n_0}$. $ctier = 1$ and is not full so $t^k_s = clb^k_1 = 0$ and $t^k_s < t^k_{smax} = t^k_{min} - t^k_e$, satisfying all three conditions.

Induction step: For the induction step we assume the proof holds for n IoT devices and show that it holds for $n+1$ IoT devices. Initially at IoT device $n+1$, for task k, $ctier = otier$ and we check if $ctier$ is full or not. If $cub^k_{ctier} - clb^k_{ctier} < t^k_b$, then $ctier$ is full and the algorithm checks the next lower tier till it finds one $ctier'$ that is not full. If $ctier' = 1$ and is still full then all n_{tier} tier boundaries are reset, $ctier' = otier$ and task k is scheduled similar to the base step. If $ctier$ is not full then are two cases, either $ctier! = otier$ or $ctier = otier$:

Case $ctier! = otier$**:** In this case either $ctier''$ is full where $ctier'' \in [ctier + 1, otier]$ or $clb_{ctier''} > t^k_{smax}$. Therefore $t^k_s = cub^k_{ctier}$. Since $ctier \leq otier - 1$ and t^k_{smax} lies in $otier$, the third condition $t^k_s < t^k_{smax}$ is satisfied.

Case $ctier = otier$**:** There are two options in this case.

Case $ctier = 1$: In this case $t^k_s = clb^k_{ctier=1}$. For any task in $otier = 1$, $t^k_{smax} = oub^k_1 - t^k_e$ because $oub^k_1 = t^k_{min}$. Also $t^k_s \leq t^k_{smax}$ because if this was not true then the full tier condition, $cub^k_{ctier} - clb^k_{ctier} < t^k_b$ would have been true earlier. Therefore condition 3 is satisfied.

Case $ctier > 1$: If $clb^k_{ctier} \leq t^k_{smax}$ and $t^k_s = clb^k_{ctier}$ then clearly $t^k_s \leq t^k_{smax}$. If $clb^k_{ctier} > t^k_{smax}$, then $t^k_s = cub^k_{ctier-1}$, and again clearly $t^k_s \leq t^k_{smax}$ since t_{smax} is in the higher $ctier$. Therefore condition 3 is also satisfied. Finally, by adjusting the boundaries for $ctier$ by t^k_b after setting t^k_s, conditions 1 and 2 are always satisfied in every case. □

4 EVALUATION

4.1 Metrics

The metrics we use for comparison include similarity measures to test schedule orders, and the number of missed start deadlines. We compare how close to the optimal order (generated using EDF) our T2 generated schedule is and use Jaro-Winkler similarity and Damerau-Levenshtein distance. The Jaro-Winkler similarity measures similarities between two strings and outputs a value in the range 0 (no similarity) to 1 (perfect match). Damerau-Levenshtein distance counts how many edits are required to transform one string to another. A missed start deadline occurs when the scheduled start time exceeds the max start time i.e. $t^k_s > t^k_{smax}$.

4.2 Experimental Setup

For evaluation we used 5 and 15 minute epochs and 3 epoch types (cloudy, sunny and hybrid) representing scenarios with

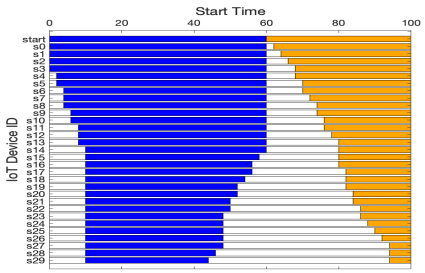

Figure 5: *T2 generated schedule for temperature task, sunny 5 min. epoch*

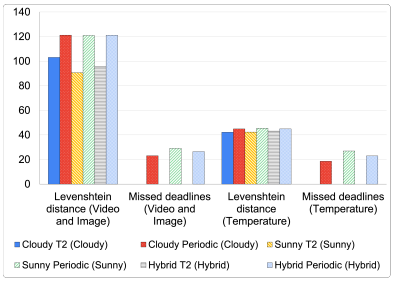

Figure 6: *Levenshtein distance & start deadline misses (T2 & periodic schedulers).*

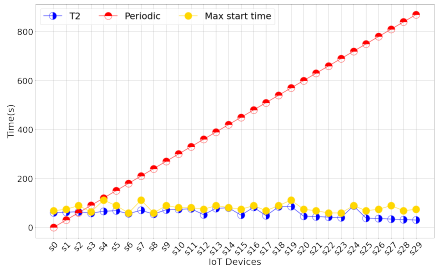

Figure 7: *T2 & periodic schedulers start times and max. start time (temp. task).*

Table 3: Comparing Jaro similarity.

	Cloudy Epoch		Sunny Epoch		Hybrid Epoch	
Epoch type	T2	Periodic	T2	Periodic	T2	Periodic
Video & Image	0.8890	0.8745	0.8897	0.8742	0.8915	0.8756
Temp.	0.9081	0.8828	0.9041	0.8762	0.9017	0.8736

different solar energy availability i.e. cloudy day, sunny day and hybrid day with both sunny and cloudy periods. We used 30 IoT devices and generated task values for each device according to the epoch type. For the cloudy epoch task values used were limited to the lower half of the task range, e.g for temperature task, values ranged from 1-7. For the sunny epoch task values were limited to the upper half of the task range, e.g for temperature task, values ranged from 8-15. Hybrid epochs use the full range of task values.

We performed multiple experiments and averaged the results over all experiments. We also compare the T2 algorithm against a simple round robin periodic scheduler. The periodic scheduler divides the total epoch time by the number of live IoT devices n_{live} to determine the minimum start interval. If the token schedules IoT device s at time t_s then the next IoT device has it's start time set to t_s+ minimum start interval. Thus scheduling all IoT devices in a round robin manner.

4.3 Results and Discussion

4.3.1 Schedule Order. Fig. 5 shows how tasks are scheduled in tiers for a 5 min sunny epoch as the token moves to all IoT devices. By scheduling in tiers and at boundaries, T2 leaves room for tasks at waiting IoT devices with shorter deadlines to be scheduled. The schedule order generated by the T2 and periodic schedulers are different from the optimal EDF schedule order. From Levenshtein distance and Jaro similarity measures for both task types the T2 algorithm schedule order is closer to the optimal (Fig. 6 and Table 3).

4.3.2 Start Deadlines. We compare start deadlines between T2 and the periodic scheduler using a 15 min. epoch and results show the T2 scheduler always schedules the start time within the max. start time deadline. The periodic scheduler on the other hand exceeds the deadline most of the time. Fig. 7 shows results for the temperature task for a 15 minute sunny epoch. If we reduce the epoch duration to 5 mins (but maintain the same task values) the T2 algorithm is still able to schedule all the tasks before their max. start time deadlines. There is no change for the periodic scheduler. Although a shorter epoch means its interval between task start times $(epoch\ duration)/(live\ nodes)$ is now shorter, if the task values are the same, task periods also decrease and so missed deadlines remains the same.

The T2 algorithm can successfully schedule all dependent and independent tasks within their start deadlines, ensuring each IoT device is able to perform its sensing tasks fully. The T2 algorithm can meet task dependency constraints and by distributing the tasks within the epoch reduces temporal overlap. Therefore the T2 scheduler is a feasible option for distributed task scheduling in IoT networks.

5 CONCLUSIONS

Cooperative sensing among multiple IoT devices can help manage energy consumption. But, scheduling tasks in a distributed environment is challenging. Each IoT device has limited data about other devices and limited energy, especially to share information between devices. The Tier-based Task scheduling protocol (T2) is an energy efficient distributed scheduler for an IoT network. T2 works with multi-sensor IoT devices and distributes task execution throughout each epoch to minimize temporal overlap within a device and between devices, without exceeding deadlines. T2 finds a task's start time deadline and schedules tasks with shorter deadlines in earlier tiers in a distributed manner and with minimal information shared between IoT devices. Experiments showed T2 always schedules an IoT device's start time before its deadline expires and compared to a simple periodic scheduler, the T2 schedule order is closer to an optimal EDF schedule.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Carlo Bergonzini, Davide Brunelli, and Luca Benini. 2009. Algorithms for harvested energy prediction in batteryless wireless sensor networks. In *2009 3rd International workshop on advances in sensors and interfaces*. IEEE, Trani,Italy, 144–149. https://doi.org/10.1109/IWASI.2009.5184785

[2] Werner Bux. 1989. Token-ring local-area networks and their performance. *Proc. IEEE* 77, 2 (1989), 238–256.

[3] Alessandro Cammarano, Chiara Petrioli, and Dora Spenza. 2012. Pro-Energy: A novel energy prediction model for solar and wind energy-harvesting wireless sensor networks. In *2012 IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2012)*. IEEE, 75–83.

[4] Antonio Caruso, Stefano Chessa, Soledad Escolar, Xavier Del Toro, and Juan Carlos López. 2018. A dynamic programming algorithm for high-level task scheduling in energy harvesting IoT. *IEEE Internet of Things Journal* 5, 3 (2018), 2234–2248.

[5] Massimiliano De Benedetti, Fabrizio Messina, Giuseppe Pappalardo, and Corrado Santoro. 2017. JarvSis: a distributed scheduler for IoT applications. *Cluster Computing* 20, 2 (2017), 1775–1790.

[6] American National Standards Institute. 1985. *Local Area Networks: Token Ring Access Method and Physical Layer Specifications–802.5*. John Wiley & Sons, Inc., USA.

[7] Marjory J. Johnson. 1986. Reliability mechanisms of the FDDI high bandwidth token ring protocol. *Computer Networks and ISDN Systems* 11, 2 (1986), 121–131. https://doi.org/10.1016/0169-7552(86)90012-7

[8] M. Scott Kingley. 1996. *ANSI Fiber Distributed Data Interface (FDDI) Standards*. Retrieved June 26, 2022 from http://www.bitsavers.org/pdf/datapro/communications_standards/2715_FDDI.pdf

[9] Selahattin Kosunalp. 2016. A new energy prediction algorithm for energy-harvesting wireless sensor networks with Q-learning. *IEEE Access* 4 (2016), 5755–5763.

[10] Meng-Lin Ku, Wei Li, Yan Chen, and K. J. Ray Liu. 2016. Advances in Energy Harvesting Communications: Past, Present, and Future Challenges. *IEEE Communications Surveys Tutorials* 18, 2 (2016), 1384–1412. https://doi.org/10.1109/COMST.2015.2497324

[11] Xiaolan Liu, Yue Gao, and Fengye Hu. 2019. Optimal Time Scheduling Scheme for Wireless Powered Ambient Backscatter Communications in IoT Networks. *IEEE Internet of Things Journal* 6, 2 (2019), 2264–2272. https://doi.org/10.1109/JIOT.2018.2889700

[12] Clemens Moser, Jian-Jia Chen, and Lothar Thiele. 2010. Dynamic power management in environmentally powered systems. In *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 81–88.

[13] F.E. Ross. 1989. An overview of FDDI: the fiber distributed data interface. *IEEE Journal on Selected Areas in Communications* 7, 7 (1989), 1043–1051. https://doi.org/10.1109/49.44552

[14] Smruti R. Sarangi, Sakshi Goel, and Bhumika Singh. 2018. Energy Efficient Scheduling in IoT Networks. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing* (Pau, France) *(SAC '18)*. Association for Computing Machinery, New York, NY, USA, 733–740. https://doi.org/10.1145/3167132.3167213

[15] Faisal Karim Shaikh and Sherali Zeadally. 2016. Energy harvesting in wireless sensor networks: A comprehensive review. *Renewable and Sustainable Energy Reviews* 55 (2016), 1041–1054. https://doi.org/10.1016/j.rser.2015.11.010

[16] Henry Yang and KK Ramakrishnan. 1991. A Ring Purger for the FDDI token ring. In *[1991] Proceedings 16th Conference on Local Computer Networks*. IEEE Computer Society, 503–504.

[17] Zhen Zhang, Jianfeng Wu, Yan Zhao, and Ruining Luo. 2022. Research on Distributed Multi-Sensor Cooperative Scheduling Model Based on Partially Observable Markov Decision Process. *Sensors* 22, 8 (2022). https://doi.org/10.3390/s22083001