Scheduling Sensor Data Collection with Dynamic Traffic Patterns

Wenbo Zhao and Xueyan Tang

Abstract—The network traffic pattern of continuous sensor data collection often changes constantly over time due to the exploitation of temporal and spatial data correlations as well as the nature of condition-based monitoring applications. In contrast to most existing TDMA schedules designed for a static network traffic pattern, this paper proposes a novel TDMA schedule that is capable of efficiently collecting sensor data for any network traffic pattern and is thus well suited to continuous data collection with dynamic traffic patterns. In the proposed schedule, the energy consumed by sensor nodes for any traffic pattern is very close to the minimum required by their workloads given in the traffic pattern. The schedule also allows the base station to conclude data collection as early as possible according to the traffic load, thereby reducing the latency of data collection. We present a distributed algorithm for constructing the proposed schedule. We develop a mathematical model to analyze the performance of the proposed schedule. We also conduct simulation experiments to evaluate the performance of different schedules using real-world data traces. Both the analytical and simulation results show that, compared with existing schedules that are targeted on a fixed traffic pattern, our proposed schedule significantly improves the energy efficiency and time efficiency of sensor data collection with dynamic traffic patterns.

Index Terms—Data collection, TDMA, scheduling, energy efficiency, latency, wireless sensor networks.

1 Introduction

Energy efficiency and time efficiency are two major considerations for sensor data collection in wireless sensor networks. Energy efficiency concerns the amount of energy spent in data collection. Since sensor nodes are normally powered by batteries, it is critical to conserve energy as much as possible to extend the lifetime of a sensor network [2]–[4]. Time efficiency, on the other hand, refers to the latency of collecting data from sensor nodes to a base station (or a sink node). Sensor data are often required to be quickly gathered after acquisition for timely processing [2].

TDMA is an attractive MAC protocol for efficient data collection in wireless sensor networks [5]–[16]. TDMA eliminates collisions by scheduling only non-interfering transmissions to proceed in the same time slot. It avoids the energy cost and latency overhead required by contention-based MAC protocols to compete for channel access and to perform retransmissions upon collisions.

Most existing TDMA schedules are constructed for a static network traffic pattern in which a fixed set of nodes report data to the base station [6]–[10]. In practice, however, continuous sensor data collection often exhibits dynamically changing traffic patterns over time due to energy conservation concerns and the nature of monitoring applications. For example:

- To save energy, temporal and spatial correlations among sensor measurements are usually exploited to reduce the amount of data that need to be collected [17]–[19]. In this approach, a sensor node updates new measurements with the base station only when the new measurements differ considerably from past ones or their correlations with other measurements in the vicinity change substantially. As a result, the set of reporting nodes normally changes from one sampling interval to another in a largely unpredictable manner.
- Condition-based monitoring is widely used in surveillance applications such as volcano monitoring [2], contour mapping [20], and structural health monitoring [21]. In these applications, only sensor measurements satisfying certain conditions need to be reported. For instance, to monitor volcanic activities, only the spikes of seismic and infrasonic signals need to be collected [2]. Thus, the set of sensor nodes that report to the base station usually varies over different sampling intervals.

One possible approach to cope with dynamic traffic patterns in continuous data collection is to construct and deploy a new TDMA schedule tailored to the new traffic pattern whenever the traffic pattern changes. However, identifying new traffic patterns and disseminating new schedules over the network both require sensor nodes to communicate with each other, which introduces extra energy and latency overheads. When the traffic pattern changes frequently, the overheads are very likely to cancel out or even outweigh the benefits of deploying new schedules [8]. *Therefore, it is highly desirable to consistently use a single TDMA schedule that is able to efficiently handle a wide variety of traffic patterns.*

This paper proposes a novel TDMA schedule that achieves high energy efficiency and time efficiency for any traffic pattern and is thus well suited to continuous data collection with dynamic traffic patterns. A salient feature of the schedule is that it enables each node to transmit all data in its successive transmission slots irrespective of the traffic pattern. This allows a receiving node to identify the end of transmission performed by a sending node and then to stop listening to the sending node without worrying about missing any data. In this way, the energy consumed by each node self-adapts to its

• *A preliminary report of this work was presented at the IEEE INFOCOM 2011 Mini-Conference [1].* *The authors are with the School of Computer Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798. Email: {zhao0101, asxytang}@ntu.edu.sg.*

required workload given in any traffic pattern. In addition, the proposed schedule allows the base station to conclude data collection as early as possible according to the traffic load, thereby reducing the latency of data collection. We present a distributed algorithm for constructing the proposed schedule. We develop a mathematical model to analyze the performance of the proposed schedule and compare it with a state-of-the-art schedule and a yardstick ideal schedule. We also conduct simulation experiments to evaluate the performance of different schedules using real-world data traces. Both the analytical and experimental results show that, compared with existing schedules that are targeted on a fixed traffic pattern, our proposed schedule significantly improves the energy efficiency and time efficiency of sensor data collection with dynamic traffic patterns.

The rest of the paper is organized as follows. Section 2 summarizes the related work. Section 3 describes the system model. Section 4 elaborates our proposed scheduling algorithm. Section 5 analyzes the performance of the proposed schedule. The experimental evaluation is presented in Section 6. Finally, Section 7 concludes the paper.

2 Related Work

TDMA scheduling for sensor data collection has attracted much research attention in recent years. Some early work aims to construct schedules for each node to communicate once with each of its neighbors [5]. In these schedules, one time slot is assigned to each communication link in the network. Some recent work targets at constructing schedules for *aggregate data collection* with in-network data aggregation [11]–[16]. In the in-network aggregation, each internal node in the routing tree aggregates all the data received from its children before forwarding them upstream. One transmission slot is normally assigned to each node in the schedule, because the partial aggregate results sent by different nodes to their parents have the same size that can often fit into one data packet. To allow for in-network aggregation, the transmission of an internal node must be scheduled after all of its children's transmissions.

The above schedules do not meet the communication demands of *non-aggregate data collection* that aims to collect the data acquired by individual sensor nodes [3], [4], [18], [22]. In non-aggregate data collection, an internal node needs more transmission slots than any of its children in the routing tree because it has to relay all the data received from its children. Meanwhile, due to the absence of in-network aggregation, the transmissions of an internal node do not have to be all scheduled after all the transmissions of its children. Several recent studies have investigated scheduling non-aggregate data collection [6]–[10]. Nevertheless, these studies all target at constructing schedules for a given static network traffic pattern in which each node generates a fixed amount of data to transport to the base station. These schedules are either inapplicable to or inefficient in dealing with dynamic traffic patterns. A schedule constructed for a light traffic pattern is unable to meet the demand of a heavy traffic pattern, whereas a schedule built for a heavy traffic pattern introduces much idle listening and unnecessary delay when handling a light traffic pattern as shall be shown in Section 4.1. To the best of our knowledge, there has been little work on designing efficient TDMA schedules for continuous non-aggregate data collection with dynamic traffic patterns.

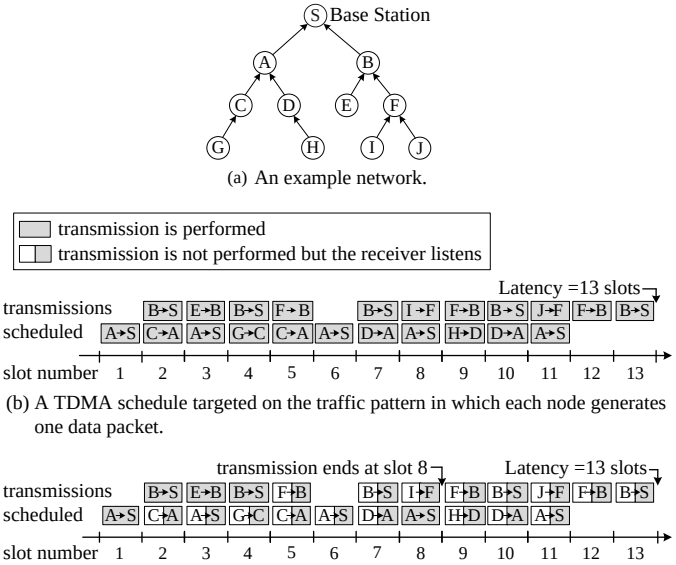
There are also a number of studies on the network capacity of sensor networks [23]–[30]. Many of these studies have proposed scheduling algorithms for maximizing the data delivering rate to the base station, which is important to applications where large amount of data needs to be continuously and quickly collected. However, a capacity-maximized schedule indicates neither the minimum delay of data collection nor the minimum amount of energy spent by sensor nodes. In addition, the above schedules are all constructed to deal with the fixed traffic pattern in which all sensor nodes generate equal amounts of data to send to the base station. They could not effectively handle continuous data collection with dynamic traffic patterns.

3 System Model

We consider a continuous data collection scenario in which data is collected from sensor nodes to the base station once per sampling interval. In each sampling interval, sensor nodes first sample local phenomena (like temperature and solar radiation) and then transmit the acquired data to the base station. Following common practices [6]–[9], [31], [32], we assume that the sensor nodes are organized into a tree structure rooted at the base station for data collection. In addition to reporting its own data, each internal node in the tree is also responsible for forwarding the data received from its children to its parent. Due to various reasons as described in the introduction, a sensor node may not report its acquired data to the base station at every sampling interval. Decisions of what data to report are driven by the data itself and are thus made only after data are acquired by the sensor node. As a result, the network traffic pattern of data collection changes over different sampling intervals in an unpredictable manner.

Our objective is to design a TDMA schedule to conserve energy at sensor nodes and reduce the latency of data collection as much as possible for any traffic pattern. Similar to other studies [5]–[9], clocks are assumed to be synchronized among sensor nodes [33]. Time is divided into slots and the duration of a time slot allows a sensor node to transmit one packet. To simplify presentation, we assume that the acquired data reported by each node in a sampling interval, if any, fit into one packet, and packets generated by different nodes are not aggregated on their ways toward the base station. Our algorithm can be easily extended to take into account packet aggregation in which an internal node may combine multiple packets received from its children into one packet for forwarding to its parent, subject to some packet size limit. Please refer to Appendix B in the supplementary file for detailed discussions.

To avoid collisions, only transmissions that do not conflict with each other are allowed to be scheduled in the same time slot. Our proposed scheduling strategy, as shall be elaborated in Section 4, is independent of the propagation and interference models. For simplicity of illustration, we shall assume the



(a) An example network.

(b) A TDMA schedule targeted on the traffic pattern in which each node generates one data packet.

Fig. 1: Inefficiency of a schedule targeted on a static network traffic pattern.

protocol interference model in the discussion of this paper, which considers pairwise conflict relationships only. But we do not make any specific assumption about the interference radius relative to the transmission radius and whether the transmission/interference range of a sensor node is regular. We simply assume that each node knows the other nodes conflicting with it. This information can be obtained by using practical methods such as RID [34].

4 Traffic Pattern Oblivious Schedules

4.1 A Motivating Example

We start with an example illustrating how a conventional TDMA schedule constructed for a static traffic pattern introduces idle listening and unnecessary delay under dynamic traffic patterns. Fig. 1(a) shows a network of 10 sensor nodes organized into a tree structure for data collection. For simplicity, in this example, we assume that the transmission from each node to its parent conflicts with only the transmissions of its siblings, parent and grandparent in the tree. Fig. 1(b) shows a typical schedule [6], [8] targeted on the full traffic pattern in which each node generates one packet to send to the base station. In the full traffic pattern, the base station receives the last packet in slot 13. So, the latency of data collection is 13 time slots. In the data collection process, node B listens to transmissions in four time slots (slots 3, 5, 9 and 12).

Now suppose the traffic pattern changes such that only nodes A, B, D and E generate packets to send, and the change is not known a priori to any node as well as the base station. Using the same schedule, actual transmissions occur only in the fully shaded rectangles shown in Fig. 1(c). Although the base station has received all the four packets by the end of slot 8, it has to continue listening for possible transmissions in slots 10, 11 and 13. This is because packets generated by nodes I, H and J, if any, are scheduled to be forwarded to the base station in slots 10, 11 and 13 respectively by nodes B, A and

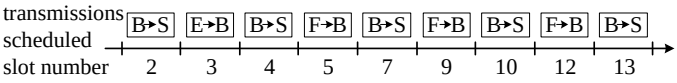


Fig. 2: Node B's transmission and listening slots in the schedule of Fig. 1(c).

B. Since the base station does not know beforehand whether nodes I, H and J generate any packet or not, it has to listen in these slots in order not to miss any potential transmission. The base station cannot make certain that all data have been received until the end of slot 13. Therefore, the latency of data collection remains 13 time slots. For node B, it receives only one packet from node E in slot 3. Similar to the base station, since B is not aware of which descendants generate packets, B still has to listen for possible transmissions in all the four slots 3, 5, 9 and 12, the last three of which result in idle listening as shown by the half shaded rectangles in Fig. 1(c). Similar observations can also be made for other nodes in the network.

To investigate the cause, Fig. 2 shows all the transmission and listening slots of node B in the schedule of Fig. 1(c). It can be seen that B's transmission slots alternate with its listening slots in the temporal order. As a result, each transmission slot of B is used to forward the packet received in a *unique* listening slot (i.e., the immediate preceding listening slot). If node B does not receive any packet in a listening slot, the following transmission slot would be left idle. However, since node B's parent does not know the traffic pattern in advance, B's parent has to listen in all of B's transmission slots in order not to miss any possible transmission. If B's parent is the base station (as in the example of Fig. 1), this prevents the base station from concluding data collection early when the traffic load is light. If B's parent is another sensor node, this leads to a lot of idle listening at B's parent when the traffic load is light. The unnecessary delay in data collection and the energy waste due to idle listening are more significant for traffic patterns of lighter loads.

4.2 Reducing Idle Listening and Latency

Our key strategy for reducing idle listening and latency is to allow each node to transmit all data in its *successive* transmission slots starting from its first transmission slot, irrespective of the traffic pattern. In this way, if a node does not send out any packet in a scheduled transmission slot, the node will leave all of its subsequent transmission slots idle as well. Thus, on observing an idle transmission slot of the node, its parent does not need to listen in any subsequent transmission slot of that node, thereby reducing idle listening. By applying the above strategy to all nodes in the network, a parent node listens to each child for at most one more slot than the actual number of transmission slots used by the child to send packets. As a result, the energy consumed by each node self-adapts to its required workload given in any traffic pattern. Moreover, when the base station observes an idle transmission slot of a child node, it can infer that the child node has finished transmitting all data. Therefore, instead of listening till the end of the schedule, the base station concludes data collection once it infers that all of its children have finished transmission.

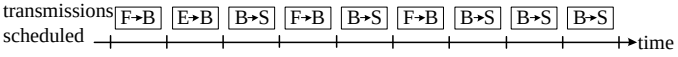


Fig. 3: A temporal order of transmission slots that allows node B to transmit all data in its successive transmission slots.

To illustrate our strategy, consider again the example network of Fig. 1(a). Suppose that node B and its children's transmission slots are arranged in the temporal order shown in Fig. 3, where B's first transmission slot is scheduled after E and F's first transmission slots, and B's second transmission slot is scheduled after F's second transmission slot. Then, node B is able to transmit all data in its successive transmission slots irrespective of the traffic pattern, provided that both nodes E and F do so. This is because based on whether E and F's first transmission slots are idle or not, B would have learned whether E and F have any packet to forward to it. In addition, B certainly knows whether itself generates any packet to send to the base station. Thus, prior to B's first transmission slot, B is already aware of whether it needs to send at least one packet to its parent or not (but B may not know the total number of packets to send to its parent). If at least one packet needs to be sent, B can send out a packet in its first transmission slot, which may be any packet available at B (i.e., either the packet generated by B or the packet received from E or F in their first transmission slots). Otherwise, B leaves its first transmission slot idle and will not have anything to send in all its subsequent transmission slots either. Similar arguments apply to B's second transmission slot as well. Therefore, node B is able to transmit all data in its successive transmission slots starting from its first transmission slot.

Let T_v be the subtree rooted at a node v, and $|T_v|$ be the size of T_v. In general, the relative transmission order of a node v and its children must satisfy the following condition $\mathcal{S}$ to enable v's transmission in its successive transmission slots.

Condition $\mathcal{S}$: for each index $1 \leq i \leq |T_v|$ and each child c of node v, v's ith transmission slot should be scheduled after c's ith transmission slot if $|T_c| > i$, and after all of c's transmission slots if $|T_c| \leq i$.

Theorem 4.1. *Condition $\mathcal{S}$ is the **necessary and sufficient condition** for enabling a node v to transmit all data in its successive transmission slots starting from its first transmission slot, assuming that all of its children do so.*

Proof: To prove the sufficiency, consider any schedule that satisfies condition $\mathcal{S}$. For each index $1 \leq i \leq |T_v|$, we show that if node v leaves its ith transmission slot idle, it would not have anything to send in all its subsequent transmission slots either. In fact, leaving its ith transmission slot idle implies that node v has less than i packets available for transmission by that time. This means that v has received at most $i-1$ packets from its children. For each child c of v where $|T_c| > i$, based on condition $\mathcal{S}$, c has i transmission slots scheduled before v's ith transmission slot. If v receives at most $i-1$ packets in total by its ith transmission slot, at least one of the first i transmission slots of c must be idle. This implies that c has finished transmitting all data. On the other hand, for each child c of v where $|T_c| \leq i$, all of c's transmission slots are scheduled before v's ith transmission slot according to condition $\mathcal{S}$. So, c certainly finishes transmission by v's ith transmission slot. Therefore, all of v's children must have finished transmission before v's ith transmission slot. As a result, v would not have anything to send after its ith transmission slot.

To prove the necessity, for each index $1 \leq i \leq |T_v|$, consider a traffic pattern in which node v needs to send a total of i packets to its parent. For each child c of node v, if $|T_c| > i$, it is possible that all these i packets are generated in the subtree T_c and hence would be forwarded by c to v. Thus, it is essential to schedule i transmission slots of c before v's ith transmission slot to guarantee successive transmission by v. Similarly, for each child c of node v where $|T_c| \leq i$, it is possible that $|T_c|$ out of the i packets that node v needs to send to its parent are contributed by the subtree T_c. So, it is necessary to schedule all of c's transmission slots before v's ith transmission slot to guarantee successive transmission by v.

Hence, the theorem is proven. □

Theorem 4.1 is generic and is applicable to any network topology, radio propagation model and interference model. By induction, we have the following corollary:

Corollary 4.2. *If condition $\mathcal{S}$ is fulfilled at each node, all the nodes in the network can transmit data in their successive transmission slots starting from their first transmission slots.*

4.3 Our Proposed Scheduling Algorithm

Our proposed scheduling algorithm is called TPO (Traffic Pattern Oblivious) in that the constructed schedule effectively deals with any network traffic pattern. The TPO scheduling algorithm works in rounds. In each round, the algorithm assigns one new transmission slot to each node that has not been assigned its required number of transmission slots. Note that a node rooted at a subtree of size x needs a total of x transmission slots. Thus, each round i involves all the nodes rooted at subtrees of sizes at least i. Fig. 4 shows the sets of nodes involved in different rounds for the network of Fig. 1(a). As can be seen, the nodes involved in each round induce a subtree together with the base station.

Algorithm 1 presents the pseudo code of the TPO scheduling algorithm, where $C(v)$ denotes the set of node v's children in the routing tree, $I(v)$ represents the set of nodes that conflict with v, and $TS(v)$ records all the transmission slots assigned to node v. In each round (steps 5 to 9), the algorithm performs scheduling by a post-order traversal of the subtree induced by the nodes involved. As given in step 6 of Algorithm 1, to fulfill condition $\mathcal{S}$, each node v in the subtree is assigned a new transmission slot that is after all the transmission slots that have been assigned to itself and its children at that time, i.e., $TS(v) \cup \bigcup_{c \in C(v)} TS(c)$. In addition, to avoid conflicts, the new transmission slot is different from all the transmission slots that have been assigned to the nodes in $I(v)$, i.e., $\bigcup_{u \in I(v)} TS(u)$.

Theorem 4.3. *Algorithm 1 generates a conflict-free schedule that fulfills condition $\mathcal{S}$ at each node in the network.*

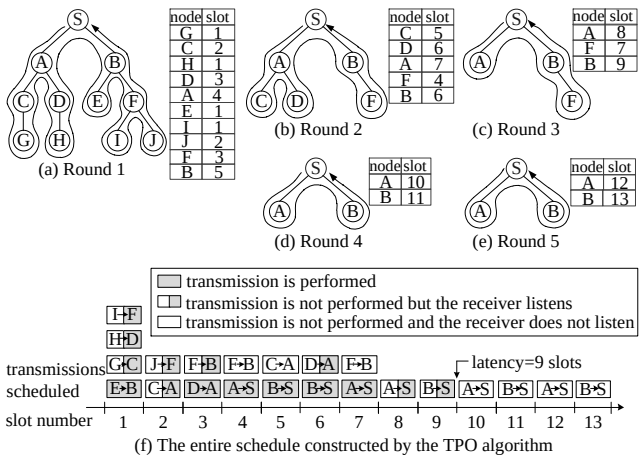
Algorithm 1: TPO Scheduling Algorithm

```
1  let N be the list of all nodes in a post-order traversal;
2  for each node v in N do
3  |   TS(v) ← ∅;
4  while N ≠ ∅ do
5  |   for each node v in N do
6  |   |   t ← min{s | s ∉ ⋃_{u∈I(v)} TS(u) and s > j,
   |   |                ∀j ∈ TS(v) ∪ (⋃_{c∈C(v)} TS(c))};
7  |   |   TS(v) ← TS(v) ∪ {t};
8  |   |   if |TS(v)| = |T_v| then
9  |   |   |   remove v from N;
```

Proof: Please refer to Appendix A in the supplementary file for the detailed proof. □

Figs. 4(a) to (e) show an example execution of the TPO algorithm for the network of Fig. 1(a) round by round. The transmission slots assigned to the nodes involved in each round are given by a table, in which the nodes are listed from the top to the bottom based on the order in which they are assigned new transmission slots. For example, in round 1, node G is the first node to be assigned a new transmission slot and node B is the last node to be assigned a new transmission slot. Node G is first assigned slot 1. Node C must be assigned a transmission slot after that of G, so C is assigned slot 2. Then, the algorithm assigns a transmission slot to node H. Since H does not conflict with G, H is also assigned slot 1. Node D must be assigned a transmission slot after that of H. In addition, D conflicts with C as they are both node A's children. Therefore, the earliest conflict-free slot for D is slot 3. Node A's transmission slot must be scheduled after those of C and D. Thus, A is assigned slot 4, and so on and so forth.

Fig. 4(f) shows the entire schedule constructed by the TPO algorithm and how the schedule handles the traffic pattern in which only nodes A, B, D and E generate packets to send to the base station. The fully shaded rectangles are the actual transmissions that take place; the half shaded rectangles represent that the sending node does not perform transmission, but the receiving node listens for transmission in the time slot; the non-shaded rectangles mean that the sending node does not transmit and the receiving node does not listen either. As seen from Fig. 4(f), the base station infers that nodes A and B have finished transmission respectively when it observes that slots 8 and 9 are idle. So, the base station concludes data collection at the end of slot 9. Therefore, the latency of data collection is four slots shorter than that in the schedule of Fig. 1(c). In addition, since nodes F, I and J do not generate packets, node F leaves all its transmission slots (slots 3, 4 and 7) idle. Node B, on observing that slot 3 is idle, does not need to listen in slots 4 and 7. Thus, including listening to node E in slot 1, B listens for a total of two slots only. As a result, the number of time slots in which node B listens is reduced by half (from four slots down to two) compared to the schedule of Fig. 1(c).



(f) The entire schedule constructed by the TPO algorithm

Fig. 4: An example execution of the TPO algorithm for the network of Fig. 1(a).

4.4 Distributed Implementation of TPO

In this section, we describe how to implement TPO scheduling in a distributed manner. We assume that each sensor node v knows its parent p_v, its children $C(v)$, and the size of its subtree, i.e., $|T_v|$. Normally, such information is readily available at each node after the routing tree is constructed.[1] In addition, each node v also knows the other nodes that conflict with it, i.e., the set $I(v)$. This can be obtained by having sensor nodes broadcast to their neighbors [15] or using methods such as RID [34].

In the distributed TPO algorithm, each node v maintains the following information:

- $TS(v)$, which records all the transmission slots assigned to v. Initially, $TS(v) = \emptyset$.
- $C_N(v)$, which records v's children that have not been assigned the required numbers of transmission slots. Initially, $C_N(v) = C(v)$.
- $CTS(c)$, which records the transmission slots assigned to each child $c \in C(v)$. Initially, $CTS(c) = \emptyset$.
- $I_N(v)$, which records the nodes that conflict with v and have not been assigned the required numbers of transmission slots. Initially, $I_N(v) = I(v)$.
- $CS(v)$, which records the time slots that cannot be assigned to v due to conflicts. Initially, $CS(v) = \emptyset$.

Algorithm 2 presents the pseudo code of the distributed TPO algorithm. In the scheduling process, each node chooses its own transmission slots. Each round of scheduling is conducted by circulating a `TOKEN` message to visit all the nodes that have not chosen the required numbers of transmission slots in the post order (see the example of Figs. 4(a) to (e)). Initially, the base station generates a `TOKEN` and passes it to one of its children (steps 1–3). When a sensor node v receives the `TOKEN` from its parent p_v, it indicates the beginning of a new round (steps 23–24). Thus, v forwards the `TOKEN` to its children in $C_N(v)$ one at a time (steps 25–27). After the `TOKEN` has traversed all of v's children in $C_N(v)$ and come back to v, v chooses a new transmission slot t (steps 28–30). The selection of t follows the same strategy as step 6 of Algorithm 1. Then, v sends a notification message $\mathcal{M}_N$ to all nodes in the set $I_N(v)$ (step 33), so that these nodes conflicting with v can record slot t in their $CS(\cdot)$ and avoid choosing t as their transmission

1. The nodes can also acquire the information of subtree sizes by a post-order traversal of the routing tree initiated by the base station [6].

Algorithm 2: Distributed TPO Algorithm

```
   // Algorithm executed by the base station s
1  C_tovisit ← C_N(s);                    // C_tovisit: the children to visit
2  s sends TOKEN to a node w ∈ C_tovisit;
3  C_tovisit ← C_tovisit − {w};
4  while C_N(s) ≠ ∅ do
5      wait for a TOKEN message to arrive;
6      if TOKEN⟨t, tag⟩ is received from a child u then
7          CTS(u) ← CTS(u) ∪ {t};
8          if tag = finished then C_N(s) ← C_N(s) − {u};
9          if C_tovisit ≠ ∅ then
10             s sends TOKEN to a node w ∈ C_tovisit;
11             C_tovisit ← C_tovisit − {w};
12         else C_tovisit ← C_N(s);
   // Algorithm executed by a sensor node v
13 while |TS(v)| ≠ |T_v| do
14     wait for a message to arrive;
15     if M_N⟨t, tag⟩ is received from a node u then
16         CS(v) ← CS(v) ∪ {t};
17         if tag = finished then I_N(v) ← I_N(v) − {u};
18     else if a TOKEN message is received then
19         if TOKEN⟨t, tag⟩ is received from a child u then
20             CTS(u) ← CTS(u) ∪ {t};
21             if tag = finished then
22                 C_N(v) ← C_N(v) − {u};
23         if TOKEN is received from parent p_v then
24             C_tovisit ← C_N(v); // C_tovisit: the children to visit
25         if C_tovisit ≠ ∅ then
26             v sends TOKEN to a node w ∈ C_tovisit;
27             C_tovisit ← C_tovisit − {w};
28         else
29             t ← min{ s | s ∉ CS(v) and s > j,
                        ∀j ∈ TS(v) ∪ (⋃_{c∈C(v)} CTS(c)) };
30             TS(v) ← TS(v) ∪ {t};
31             if |TS(v)| < |T_v| then tag = unfinished;
32             else tag = finished;
33             v sends a notification message M_N⟨t, tag⟩ to all
               nodes in I_N(v);
34             v sends TOKEN⟨t, tag⟩ back to parent p_v;
```

slots. Acknowledgements from the nodes in $I_N(v)$ can be used to guarantee successful transmission of message $\mathcal{M}_N$.[2] Next, v sends the TOKEN message back to its parent p_v (step 34).

The notification message $\mathcal{M}_N$ sent by v has the form of $\langle t, \texttt{tag}\rangle$, where t is v's newly selected transmission slot and the boolean tag indicates whether v has finished scheduling. tag is computed by comparing $|TS(v)|$ against $|T_v|$ (steps 31–32). On receiving $\mathcal{M}_N$, each node $x \in I_N(v)$ inserts t into its $CS(x)$ (steps 15–16). Node x also removes v from its $I_N(x)$ if $\mathcal{M}_N$ indicates that v has finished scheduling (step 17). The TOKEN sent by node v to its parent p_v also bears the form $\langle t, \texttt{tag}\rangle$. On receiving the TOKEN, p_v inserts t into $CTS(v)$ (steps 19–20), and removes v from its $C_N(p_v)$ if the TOKEN indicates that v has finished scheduling (steps 21–22).

Similar to sensor nodes, the base station s forwards the TOKEN to its children in $C_N(s)$ one at a time (steps 9–11). On receiving TOKEN$\langle t, \texttt{tag}\rangle$ from a child u, s inserts t into $CTS(u)$ (steps 6–7), and removes u from its $C_N(s)$ if the TOKEN indicates that u has finished scheduling (step 8). After the TOKEN has traversed all the children in $C_N(s)$ and come back to s, s starts a new round of scheduling (step 12). This process continues until all the children of the base station have finished scheduling (step 4).

On completing the scheduling process, each node v would know its transmission slots $TS(v)$ and its listening slots recorded in $CTS(c)$ for each child c. A schedule constructed by the TPO algorithm is to be consistently used throughout the duration of continuous data collection, irrespective of the traffic pattern. Thus, the scheduling overhead, amortized over the duration of continuous data collection, would be minimal.

In essence, Algorithm 2 is a distributed implementation of Algorithm 1 presented in Section 4.3. Thus, the schedules constructed by these two versions of TPO are the same. Unless stated otherwise, we shall not distinguish between these two versions of TPO in the rest of this paper where we study the performance of their constructed schedules.

5 Performance Analysis

In this section, we develop a mathematical model to analyze the performance of the proposed TPO scheduling algorithm.

As shown in Fig. 5, we consider a complete k-ary routing tree of d levels, where each internal node has k child nodes. The base station is located at the tree root and all the other nodes in the tree are sensor nodes. Suppose that data collection has random network traffic patterns. At each sampling interval, each sensor node generates a packet to report its acquired data to the base station with a probability of p. For simplicity, packet generation is assumed to be independent across different sensor nodes and over different sampling intervals. We again assume that the transmission from each node to its parent conflicts with the transmissions of its siblings, parent and grandparent in the tree.

We compare our TPO algorithm with a state-of-the-art scheduling algorithm called TIGRA [9] and a yardstick ideal schedule denoted by IDEAL. TIGRA constructs a conflict-free schedule targeted on the full traffic pattern in which each sensor node generates one packet to send to the base station. IDEAL, on the other hand, builds a conflict-free schedule for the actual traffic pattern of each sampling interval. The TIGRA and IDEAL algorithms shall be elaborated later. We analyze the schedules built by different algorithms for the k-ary routing tree and compare their expected latency and energy performance in one sampling interval under the aforementioned random traffic patterns.

To simplify presentation, we shall denote by $\triangle_i$ the size of the subtree rooted at a node located at level i of the k-ary tree. For each $1 \le i \le d$, $\triangle_i = \sum_{j=0}^{d-i} k^j$.

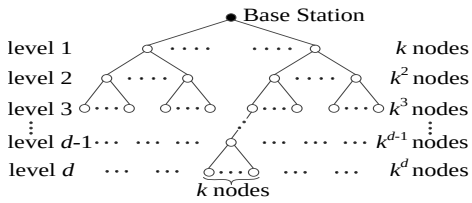


Fig. 5: A complete k-ary routing tree of d levels.

2. This is similar to employing the α-synchronizer at sensor nodes to ensure that they stay in step with all their interferers [35].

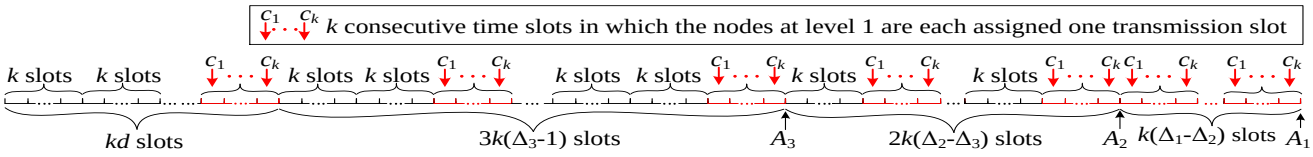


Fig. 6: The TPO schedule constructed for the k-ary routing tree shown in Fig. 5.

5.1 Performance of the TPO Algorithm

5.1.1 *Latency performance*

To compute the expected latency of data collection, we need to find out the time slots at which the base station may conclude data collection and the corresponding probabilities of concluding data collection at these time slots. We start by examining the structure of the TPO schedule built for the k-ary routing tree shown in Fig. 5. Since the k leaf nodes having the same parent at level $d-1$ conflict with each other, they must be assigned different time slots for transmission in round 1 of scheduling. The leaf nodes under different parents at level $d-1$, on the other hand, can transmit together in the same time slot. Thus, as shown in Fig. 6, k consecutive time slots (i.e., slots 1 to k) are needed for all leaf nodes at level d to complete transmission to their parents. Then, similar to the leaf nodes, k consecutive time slots (i.e., slots $k+1$ to $2k$) are needed for all nodes at level $d-1$ to be assigned one transmission slot each. Following this schedule pattern, in every k consecutive time slots, the nodes at a new level closer to the base station are each assigned one transmission slot. As a result, the nodes at level 1 are each assigned their first transmission slots in the group of slots $(d-1)k+1$ to dk.

In round 2 of scheduling, the nodes at level $d-1$ are each assigned their second transmission slots. To simplify the analysis, we make an additional assumption that the nodes that are one or two levels apart in the k-ary routing tree cannot be scheduled to transmit concurrently. That is, when a node at level i is transmitting, all nodes at levels $i-1$ and $i-2$ are not allowed to transmit at the same time. Under the above constraint, level-$(d-1)$ nodes cannot transmit in slots $2k+1$ to $4k$ in which level-$(d-2)$ and level-$(d-3)$ nodes are already scheduled to transmit. Level-$(d-1)$ nodes are assigned their second transmission slots in the group of slots $4k+1$ to $5k$, in which the first transmission slots of level-$(d-4)$ nodes are also scheduled. Similarly, level-$(d-2)$ nodes are assigned their second transmission slots in the group of slots $5k+1$ to $6k$, and so on. Thus, level-1 nodes are assigned their second transmission slots in the group of slots $(d+2)k+1$ to $(d+3)k$. In subsequent rounds of scheduling, each involved node is assigned a new transmission slot that is $3k$ time slots after its previous transmission slot. This schedule pattern continues until level-3 nodes are assigned their required numbers of transmission slots. Note that each level-3 node needs $\triangle_3$ transmission slots. Therefore, as shown in Fig. 6, the first $\triangle_3$ transmission slots of level-1 nodes are assigned in the groups of slots $(d-1)k+1$ to dk, $(d+2)k+1$ to $(d+3)k$, $(d+5)k+1$ to $(d+6)k$, $\cdots$, and A_3-k+1 to A_3, where $A_3=(d+3\triangle_3-3)k$.

By the end of slot A_3, all the nodes at levels 3 and below have finished scheduling. Starting from slot A_3+1, only the nodes at levels 1 and 2 need to be assigned more transmission slots. Thus, the cycle of the schedule pattern reduces to $2k$ time slots, in which slot assignment alternates between level-1 nodes and level-2 nodes every k consecutive time slots. This schedule pattern repeats until level-2 nodes are assigned their required numbers of transmission slots. Note that each level-2 node needs $\triangle_2$ transmission slots and has already been assigned $\triangle_3$ transmission slots by slot A_3. Therefore, as shown in Fig. 6, the $(\triangle_3+1)$-th to $\triangle_2$-th transmission slots of level-1 nodes are assigned in the groups of slots A_3+k+1 to A_3+2k, A_3+3k+1 to A_3+4k, $\cdots$, and A_2-k+1 to A_2, where $A_2=A_3+2(\triangle_2-\triangle_3)k=(d+\triangle_3+2\triangle_2-3)k$.

Starting from slot A_2+1, only level-1 nodes need to be assigned more transmission slots. Thus, each level-1 node is assigned one transmission slot in every k consecutive time slots. Since each level-1 node needs $\triangle_1$ transmission slots and has been assigned $\triangle_2$ transmission slots by slot A_2, the TPO schedule ends at slot $A_1=A_2+(\triangle_1-\triangle_2)k=(d+\triangle_3+\triangle_2+\triangle_1-3)k$.

As summarized in Fig. 6, level-1 nodes perform transmission in $\triangle_1$ slot groups each consisting of k consecutive time slots. Each slot group z begins at slot $B(z)-k+1$ and ends at slot $B(z)$, where

$$B(z)=\begin{cases} dk+3k(z-1), & \text{if } 1\le z\le \triangle_3,\\ A_3+2k(z-\triangle_3), & \text{if } \triangle_3<z\le\triangle_2,\\ A_2+k(z-\triangle_2), & \text{if } \triangle_2<z\le\triangle_1. \end{cases} \quad (1)$$

Let c_i denote the ith node at level 1 ($1\le i\le k$). The zth transmission slot of c_i is at slot $B(z)-k+i$.

In the TPO schedule, the base station can conclude data collection only at the time slots in which level-1 nodes transmit. Now, we analyze the probability for the base station to conclude data collection at the ith slot of slot group z. For ease of presentation, we shall denote by $P(x,y)$ the probability that among a set of x nodes, exactly y nodes generate packets in a sampling interval. Given the packet generation probability p, we have

$$P(x,y)=\binom{x}{y}p^y(1-p)^{x-y}. \quad (2)$$

Consider the following cases:

- $z=1$, i.e., data collection concludes in the first slot group. Since the base station needs to listen to each level-1 node for at least one time slot, data collection can only be concluded at the last slot in the first slot group. The corresponding latency of data collection is $B(1)$ slots. This happens when no packet is generated by any node in the network, which has a probability $P(k\triangle_1,0)$ to occur, where $k\triangle_1$ is the total number of sensor nodes in the k-ary tree.

- $2 \leq z \leq \triangle_1 - 1$. To conclude data collection at the ith slot of slot group z, the base station must be inferring that level-1 node c_i has finished transmission at this time slot. Since this slot is the zth transmission slot of c_i, the subtree rooted at node c_i must generate *exactly* $z-1$ packets. The probability for this to occur is $P(\triangle_1, z-1)$. For level-1 nodes $c_1, c_2, \cdots, c_{i-1}$, the base station must be assured that they have finished transmission by the ith slot of slot group z. Thus, each of the subtrees rooted at $c_1, c_2, \cdots, c_{i-1}$ can generate *at most* $z-1$ packets. The probability for this to occur is $\left(\sum_{j=0}^{z-1} P(\triangle_1, j)\right)^{i-1}$. For level-1 nodes $c_{i+1}, c_{i+2}, \cdots, c_k$, the base station must be assured that they have finished transmission by the end of slot group $z-1$. So, the subtrees rooted at $c_{i+1}, c_{i+2}, \cdots, c_k$ each can generate *at most* $z-2$ packets. The probability for this to occur is $\left(\sum_{j=0}^{z-2} P(\triangle_1, j)\right)^{k-i}$. Therefore, the overall probability of concluding data collection at the ith slot of slot group z is $\left(\sum_{j=0}^{z-1} P(\triangle_1, j)\right)^{i-1} \cdot \left(\sum_{j=0}^{z-2} P(\triangle_1, j)\right)^{k-i} \cdot P(\triangle_1, z-1)$. The corresponding latency of data collection is $B(z) - k + i$ slots.
- $z = \triangle_1$, i.e., data collection concludes in the last slot group. To conclude data collection at the ith slot of the last slot group, the base station must be inferring that c_i has finished transmission at this time slot. Since this is the last transmission slot of c_i, the subtree rooted at c_i can generate either $\triangle_1 - 1$ or $\triangle_1$ packets.[3] The probability for this to occur is $P(\triangle_1, \triangle_1 - 1) + P(\triangle_1, \triangle_1)$. Moreover, since the base station has listened to all the transmission slots of nodes $c_1, c_2, \cdots, c_{i-1}$, each of the subtrees rooted at these nodes can generate whatever number of packets from 0 to $\triangle_1$. On the other hand, the base station must be assured that nodes $c_{i+1}, c_{i+2}, \cdots, c_k$ have all finished transmission by the end of slot group $\triangle_1 - 1$. Thus, the subtrees rooted at $c_{i+1}, c_{i+2}, \cdots, c_k$ each can generate *at most* $\triangle_1 - 2$ packets. The probability for this to occur is $\left(\sum_{j=0}^{\triangle_1 - 2} P(\triangle_1, j)\right)^{k-i}$. Therefore, the overall probability of concluding data collection at the ith slot of the last slot group is $\left(\sum_{j=0}^{\triangle_1 - 2} P(\triangle_1, j)\right)^{k-i} \cdot \left(P(\triangle_1, \triangle_1 - 1) + P(\triangle_1, \triangle_1)\right)$. The corresponding latency of data collection is $B(\triangle_1) - k + i$ slots.

Thus, the expected latency of the TPO schedule is given by

$$
\begin{aligned}
& B(1)P(k\triangle_1, 0) + \sum_{z=2}^{\triangle_1 - 1} \sum_{i=1}^{k} \Bigg((B(z) - k + i) \\
& \left(\sum_{j=0}^{z-1} P(\triangle_1, j)\right)^{i-1} \left(\sum_{j=0}^{z-2} P(\triangle_1, j)\right)^{k-i} P(\triangle_1, z-1) \Bigg) \\
& + \sum_{i=1}^{k} \Bigg((B(\triangle_1) - k + i) \left(\sum_{j=0}^{\triangle_1 - 2} P(\triangle_1, j)\right)^{k-i} \\
& \left(P(\triangle_1, \triangle_1) + P(\triangle_1, \triangle_1 - 1)\right) \Bigg), \qquad (3)
\end{aligned}
$$

3. If the subtree rooted at c_i generates $\triangle_1$ packets, the ith slot of the last slot group is not idle. However, since it is the last transmission slot of c_i, the base station is still able to conclude data collection if it had inferred that all the other level-1 nodes have already finished transmission.

where $B(\cdot)$ and $P(\cdot, \cdot)$ are given by formulas (1) and (2) respectively.

5.1.2 *Energy performance*

For energy performance, we focus on the energy spent by sensor nodes in transmitting data, receiving data and idle listening. It is widely known that receiving data consumes the same order of energy as transmitting data, and idle listening consumes as much energy as receiving data [10], [36], [37]. In our analysis, we assume that each sensor node consumes e_t energy unit for transmitting a packet in a time slot, and e_r energy unit for listening for transmission in a time slot, be it idle listening or receiving a packet.

The leaf nodes in the k-ary routing tree do not listen to any other node. They simply transmit their generated packets (if any) to their parents. So, the expected energy consumption of each leaf node is $e_t \cdot p$.

Consider an internal node v located at level i ($i = 1, 2, \cdots, d-1$) of the k-ary tree. All the packets generated in the subtree rooted at v need to be transmitted by v to its parent. Since the size of the subtree rooted at v is $\triangle_i$, the expected number of packets generated in this subtree is $p \cdot \triangle_i$. Thus, the expected energy consumption of v for transmitting data is $e_t \cdot p \cdot \triangle_i$. The packets received by node v are those generated in the subtrees rooted at v's children. Since the total size of these subtrees is $k \cdot \triangle_{i+1} = \triangle_i - 1$, the expected number of packets generated in these subtrees is $p \cdot (\triangle_i - 1)$. So, the expected energy consumption of v for receiving data is $e_r \cdot p \cdot (\triangle_i - 1)$. In the TPO schedule, node v listens to each child for one idle slot unless the child sends packets in all of its transmission slots which happens when every node in the subtree rooted at the child generates a packet. Thus, the probability for node v to perform one slot of idle listening to a child is $1 - p^{\triangle_{i+1}}$. Since node v has k children, the expected energy consumption of v for idle listening is $e_r \cdot k \cdot (1 - p^{\triangle_{i+1}})$. Therefore, the expected total energy consumption of node v is $e_t p \triangle_i + e_r p(\triangle_i - 1) + e_r k(1 - p^{\triangle_{i+1}})$.

Since the k-ary tree has d levels and there are k^i nodes at each level i, the expected total energy consumption of all nodes in the network is given by $k^d e_t p + \sum_{i=1}^{d-1} k^i \left(e_t p \triangle_i + e_r p(\triangle_i - 1) + e_r k(1 - p^{\triangle_{i+1}})\right)$.

The network lifetime is largely determined by the most energy-consuming node in the network [31]. From the above analysis, the nodes with the highest expected energy consumption are those located at level 1 of the k-ary tree. Each of them has the expected energy consumption of $e_t p \triangle_1 + e_r p(\triangle_1 - 1) + e_r k(1 - p^{\triangle_2})$.

We remark here that as shown by the above analysis, TPO does not eliminate the problem that the nodes closer to the base station generally suffer from heavier workload and higher energy consumption than the other nodes in non-aggregate data collection. The burden of these heavily loaded nodes could be alleviated by enabling packet aggregation [9], or constructing more balanced routing trees for data collection [31], [38]. These techniques are orthogonal to our TPO scheduling strategy and can be used together with TPO to further improve the energy efficiency of data collection.

5.2 Performance of the TIGRA Algorithm

5.2.1 *Latency performance*

The TIGRA algorithm builds a conflict-free schedule for the full traffic pattern where each sensor node generates one packet to send to the base station [9]. The algorithm schedules one time slot at a time and keeps track of the number of packets held by each node in the scheduling process. Each time slot is scheduled in a greedy manner by examining all the nodes level-by-level down the routing tree. A node is scheduled to transmit in the current time slot if the node holds at least one packet and it does not conflict with any other node already scheduled for transmission in the current slot. In the full traffic pattern, each subtree rooted at a level-3 node generates $\triangle_3$ packets. Thus, each level-3 node needs $\triangle_3$ transmission slots to forward these packets to its parent. The k level-3 nodes under the same parent at level 2 cannot transmit concurrently. So, the transmission slots of all level-3 nodes would occupy at least $k \cdot \triangle_3$ time slots. Similarly, the transmission slots of all level-2 nodes would occupy at least $k \cdot \triangle_2$ time slots, and those of all level-1 nodes would occupy $k \cdot \triangle_1$ time slots. Under the additional constraint that the nodes one or two levels apart cannot transmit concurrently, all the above time slots must be disjoint. Therefore, the total length of the TIGRA schedule is at least $k\triangle_1 + k\triangle_2 + k\triangle_3$ time slots. Since the base station cannot conclude data collection until the end of the schedule, no matter how the traffic pattern changes, the latency of data collection is at least $k\triangle_1 + k\triangle_2 + k\triangle_3$ slots.

5.2.2 *Energy performance*

The leaf nodes simply transmit their generated packets (if any) to their parents. So, the expected energy consumption of each leaf node is $e_t \cdot p$.

Consider an internal node v located at level i ($i = 1, 2, \cdots, d-1$) of the k-ary tree. Similar to the analysis of the TPO schedule, the expected energy consumption of v for transmitting data is $e_t \cdot p \cdot \triangle_i$. In addition, node v has to listen in all the transmission slots of its children, irrespective of the network traffic pattern. Note that node v has k children and each child has $\triangle_{i+1}$ transmission slots. Thus, the energy consumption of v for receiving data and idle listening together is $e_r \cdot k \cdot \triangle_{i+1} = e_r(\triangle_i - 1)$. Hence, the expected total energy consumption of node v is $e_t p\triangle_i + e_r(\triangle_i - 1)$. Therefore, the expected total energy consumption of all nodes in the network is given by $k^d e_t p + \sum_{i=1}^{d-1} k^i \left(e_t p\triangle_i + e_r\left(\triangle_i - 1\right)\right)$.

From the above analysis, the level-1 nodes in the k-ary tree have the highest expected energy consumption. Each of them has the expected energy consumption of $e_t p\triangle_1 + e_r(\triangle_1 - 1)$.

5.3 Performance of the IDEAL Schedule

5.3.1 *Latency performance*

Finally, we analyze the performance of an IDEAL schedule. The IDEAL schedule assumes a priori knowledge of the network traffic pattern at each sampling interval. A conflict-free schedule tailored to the traffic pattern of the sampling interval is built for data collection at each sampling interval. Since we do not account for the latency and energy overheads of identifying the traffic pattern and constructing the schedule, the IDEAL schedule is used as a yardstick on the performance of data collection.

Since the IDEAL schedule is constructed for the actual traffic pattern, the base station always concludes data collection at the end of the schedule. Thus, the expected latency is equal to the expected length of the schedule.

We apply similar analysis to that of TIGRA for deriving a lower bound on the expected length of the IDEAL schedule. Each level-3 node needs to forward all the packets generated in its subtree to its parent, and each group of k level-3 nodes under the same parent cannot transmit concurrently. So, the transmission slots of all level-3 nodes would occupy x time slots, where x is the maximum number of packets generated by all the groups of k level-3 nodes under the same parent. Each group of k level-3 nodes generates i packets with probability $P(k\triangle_3, i)$. Since there are k^2 groups of level-3 nodes, the probability for all of them to generate at most x packets is $\left(\sum_{i=0}^{x} P(k\triangle_3, i)\right)^{k^2}$, and that for all of them to generate at most $x-1$ packets is $\left(\sum_{i=0}^{x-1} P(k\triangle_3, i)\right)^{k^2}$. Therefore, the probability for the transmission slots of all level-3 nodes to occupy x time slots is $\left(\sum_{i=0}^{x} P(k\triangle_3, i)\right)^{k^2} - \left(\sum_{i=0}^{x-1} P(k\triangle_3, i)\right)^{k^2}$. Similarly, the probability for the transmission slots of all level-2 nodes to occupy x time slots is $\left(\sum_{i=0}^{x} P(k\triangle_2, i)\right)^{k} - \left(\sum_{i=0}^{x-1} P(k\triangle_2, i)\right)^{k}$. On the other hand, the expected total number of packets generated by all nodes is $pk\triangle_1$. Thus, the transmission slots of all level-1 nodes occupy an expected number of $pk\triangle_1$ time slots. Under the additional constraint that the nodes one or two levels apart cannot transmit concurrently, all the above time slots must be disjoint. Therefore, the expected length of the IDEAL schedule is at least $pk\triangle_1 + \sum_{x=0}^{k\triangle_2} x\Big(\left(\sum_{i=0}^{x} P(k\triangle_2, i)\right)^{k} - \left(\sum_{i=0}^{x-1} P(k\triangle_2, i)\right)^{k}\Big) + \sum_{x=0}^{k\triangle_3} x\Big(\left(\sum_{i=0}^{x} P(k\triangle_3, i)\right)^{k^2} - \left(\sum_{i=0}^{x-1} P(k\triangle_3, i)\right)^{k^2}\Big)$ time slots.

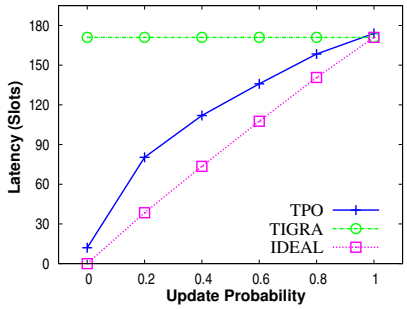
5.3.2 *Energy performance*

Since the IDEAL schedule well fits the actual network traffic pattern, there is no idle listening in executing the schedule for data collection. Similar to the analysis of the TPO schedule, the expected energy consumption of each leaf node in the IDEAL schedule is simply $e_t \cdot p$. Each internal node in the k-ary tree needs to receive all the packets generated in the subtrees rooted at its children and forward them together with its own generated packet (if any) to its parent. Thus, the expected energy consumption of each internal node at level i ($i = 1, 2, \cdots, d-1$) is $e_t \cdot p \cdot \triangle_i + e_r \cdot p(\triangle_i - 1)$. As a result, the expected total energy consumption of all nodes in the network is given by $k^d e_t p + \sum_{i=1}^{d-1} k^i (e_t p\triangle_i + e_r p(\triangle_i - 1))$.

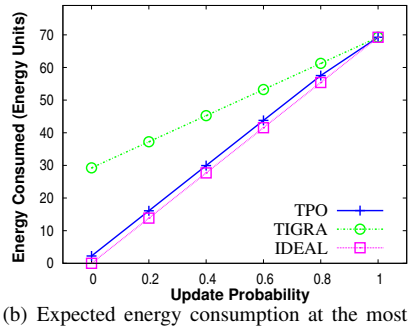
From the above analysis, the nodes with the highest expected energy consumption are those located at level 1 of the k-ary tree. Each of them has the expected energy consumption of $e_t p\triangle_1 + e_r p(\triangle_1 - 1)$.

5.4 Numerical Results and Comparison

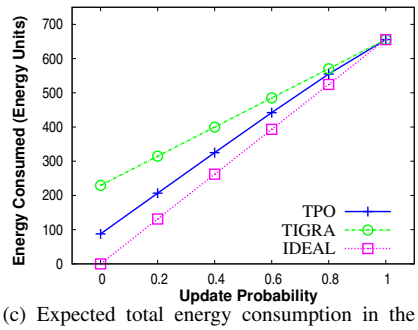
Now, we show the numerical results obtained from the analysis in the above sections. The power consumption of a Mica2



(a) Expected latency of data collection.

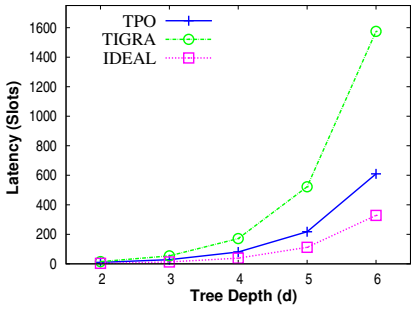


(b) Expected energy consumption at the most consuming node.

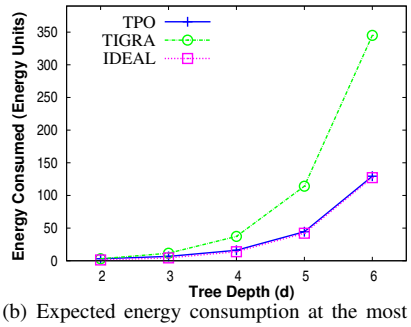


(c) Expected total energy consumption in the whole network.

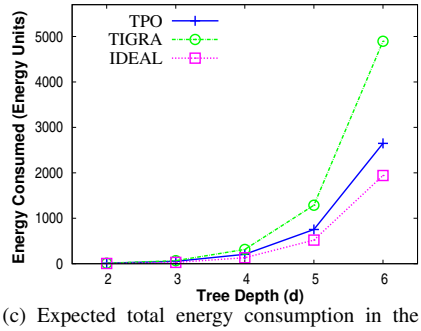
Fig. 7: Performance for different packet generation probabilities for a 3-ary tree of 4 levels.



(a) Expected latency of data collection.

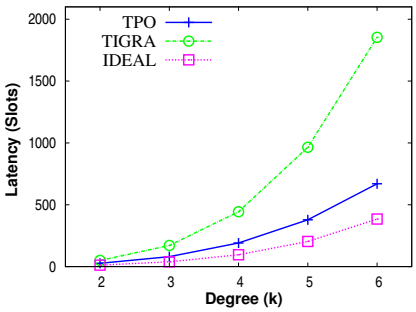


(b) Expected energy consumption at the most consuming node.

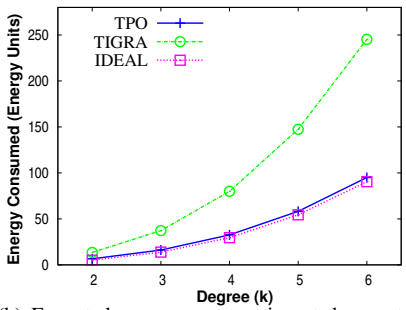


(c) Expected total energy consumption in the whole network.

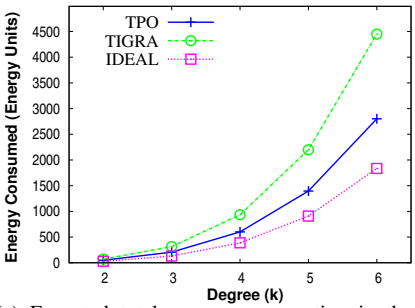
Fig. 8: Performance for 3-ary tree of different depths ($p = 0.2$).



(a) Expected latency of data collection.



(b) Expected energy consumption at the most consuming node.



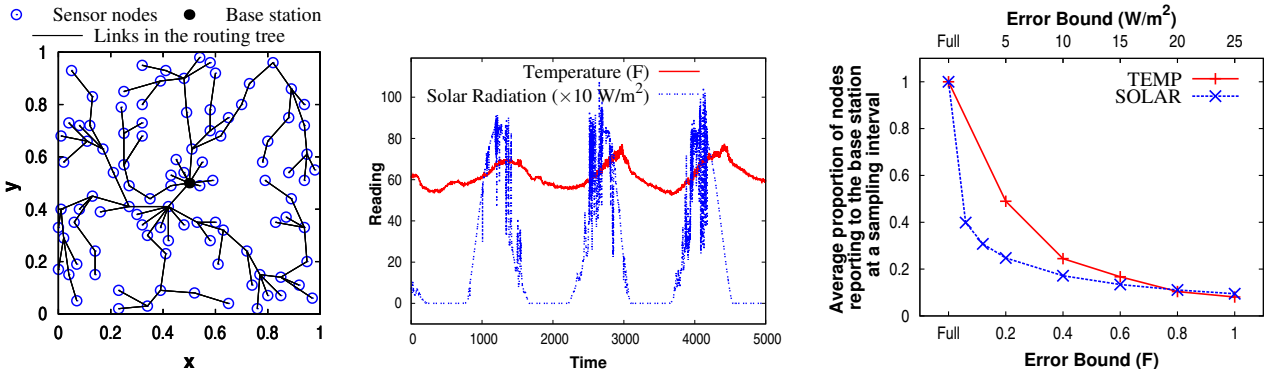
(c) Expected total energy consumption in the whole network.

Fig. 9: Performance for 4-level k-ary trees of different degrees ($p = 0.2$).

mote with a CC1000 transceiver is 60mW and 45mW for transmitting and receiving (listening) respectively [10], [36]. Following this ratio, in computing the numerical results, we set the energy costs for a sensor node to perform transmission and listen for transmission in a time slot at $e_t = 1$ and $e_r = 0.75$ energy units respectively.

First, we consider a complete 3-ary tree of 4 levels. We vary the probability p for each node to generate a packet from 0 to 1. Fig. 7(a) shows the expected latency of data collection for different scheduling algorithms as a function of p. It can be seen that TIGRA has the same latency for different p values, because the base station cannot conclude data collection until the end of the TIGRA schedule, irrespective of the network traffic pattern. In contrast, the latency of both TPO and IDEAL reduces with decreasing traffic load and is always shorter than that of TIGRA when $p < 1$. In general, the latency of TPO is quite close to that of IDEAL (i.e., the lower bound). Figs. 7(b) and (c) show the energy performance of the three algorithms for different p values. As can be seen, our TPO algorithm consistently outperforms the TIGRA algorithm in terms of both the total energy consumption in the whole network and the energy consumption at the most energy consuming node. The energy performance of our TPO algorithm is quite similar to that of IDEAL (i.e., the lower bound). In contrast, since TIGRA cannot avoid any idle listening, its energy waste becomes more significant when the traffic load is lighter. Thus, the performance difference between TIGRA and TPO increases with decreasing traffic load.

Next, we study the impacts of tree depth and degree. We first increase the depth d from 2 to 6 while keeping the degree k at 3 and the packet generation probability p at 0.2. Fig. 8 shows that the performance improvement of our TPO



(a) A sample network topology. (b) Segments of data traces. (c) Traffic load vs. error bound.

Fig. 10: Experimental settings.

algorithm over TIGRA generally increases with the tree depth. Owing to idle listening, the energy consumption at the most energy consuming node of TIGRA grows rapidly with the tree depth. We then increase the degree k from 2 to 6 while keeping the tree depth d at 4 and the packet generation probability p at 0.2. Fig. 9 shows similar performance trends to those observed in Fig. 8. Our TPO algorithm consistently outperforms the TIGRA algorithm and the performance improvement generally increases with the degree. It can also be seen from Figs. 8 and 9 that the performance of TPO is fairly close to that of IDEAL across various tree depths and degrees.

6 Experimental Evaluation

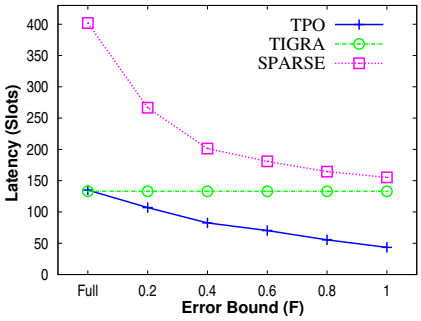
6.1 Experimental Setup

We developed a simulator to evaluate the performance of the proposed TPO scheduling algorithm using real-world data traces. We simulated a network of sensor nodes randomly placed over a square field with the base station located at the center of the field. The sensor nodes were assumed to have equal radio transmission ranges. A breadth first search tree rooted at the base station was constructed and used as the routing tree for data collection [6], [8], [10], [32]. Following other works [5]–[9], the following conflict constraints were placed on scheduling: when a node is scheduled to receive data from another node, no other neighbor of the receiving node is allowed to be scheduled for transmission in the same time slot. We have experimented with many randomly generated network topologies and observed similar performance trends. Due to space limitations, we shall focus on presenting the results for a sample network topology of 100 nodes placed over a 1×1 field as shown in Fig. 10(a), in which the transmission range of each node is set at 0.15 to ensure network connectivity.

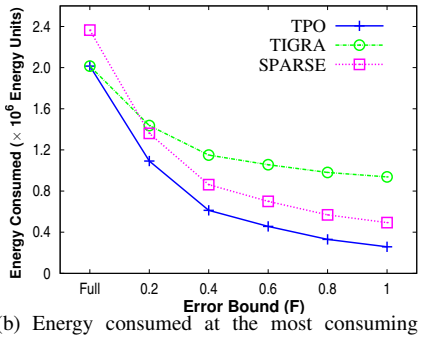
We made use of the weather data provided by the LEM project [39] at the University of Washington to simulate the physical phenomena in the immediate surroundings of sensor nodes. We used the temperature (TEMP) and solar radiation (SOLAR) traces logged by the station at the University of Washington from January 2003 to January 2009 in our experiments. Each trace consisted of more than 3,000,000 readings acquired at one-minute intervals. Fig. 10(b) shows some representative segments of these traces. For each of the TEMP and SOLAR traces, we extracted a large number of subtraces starting at randomly selected days and associated different subtraces with different sensor nodes in the simulated network. Each subtrace contained 20,000 readings.

To simulate dynamic network traffic patterns, we implemented error-bounded approximate data collection that exploits temporal data correlations to trade data accuracy for less network traffic [19]. Specifically, the base station would like to be assured that its knowledge of sensor readings is always kept within a required error bound e of the exact readings. To suppress unnecessary updates with the base station, each sensor node maintains a filter window $[u-e, u+e]$ centered at the reading u that it last updated with the base station. At each sampling interval, if the new reading acquired by the node is within the filter window, no update is sent to the base station. Otherwise, the node reports the new reading to the base station and updates its filter window. It is intuitive that the traffic load in the network decreases with an increasing error bound. Fig. 10(c) shows the average proportion of sensor nodes reporting data to the base station at a sampling interval as a function of error bound for the TEMP and SOLAR traces. In addition to approximate data collection, we also simulated exact data collection with a full traffic pattern in which all sensor nodes report their readings to the base station at every sampling interval (denoted by "Full" in Fig. 10(c)).

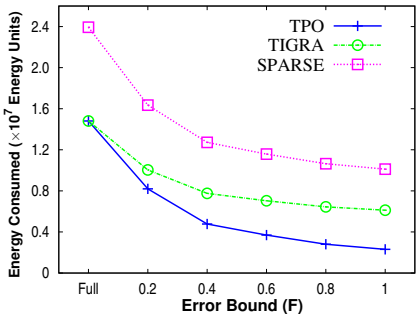
Besides our proposed TPO algorithm, we also implemented two state-of-the-art scheduling algorithms TIGRA [9] and SPARSE [8]. TIGRA, as described in Section 5, constructs a conflict-free schedule targeted on the full traffic pattern. SPARSE, on the other hand, builds a conflict-free schedule for a given traffic pattern. Both TIGRA and SPARSE aim to reduce the length of the constructed schedule. In SPARSE, to identify the traffic pattern and construct the schedule, messages must flow over the entire routing tree to complete a depth-first tree traversal and one top-down pass from the base station to all sensor nodes [8]. In our simulation of SPARSE, at each sampling interval, a new schedule tailored to the traffic pattern of the sampling interval was first constructed by conducting the tree traversal and top-down pass. Then, data collection proceeded with the constructed schedule. We conservatively assumed that all messages transmitted in the traversal and top-



(a) Average latency of data collection.

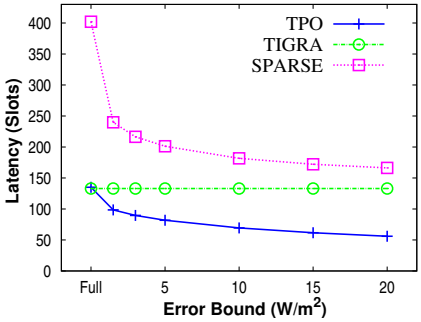


(b) Energy consumed at the most consuming node.

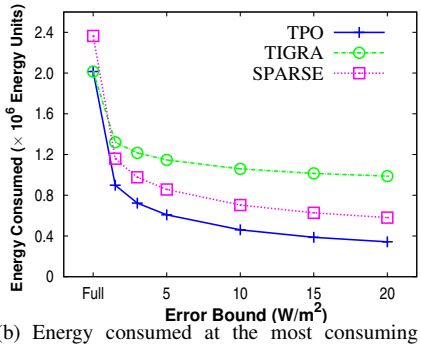


(c) Total energy consumed in the whole network.

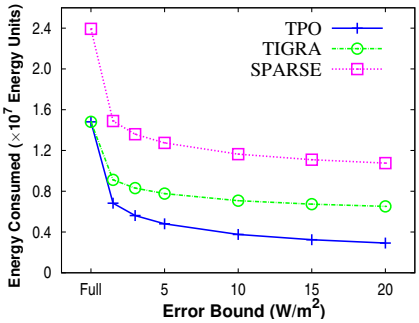
Fig. 11: Performance for different error bounds of approximate data collection (TEMP trace).



(a) Average latency of data collection.



(b) Energy consumed at the most consuming node.



(c) Total energy consumed in the whole network.

Fig. 12: Performance for different error bounds of approximate data collection (SOLAR trace).

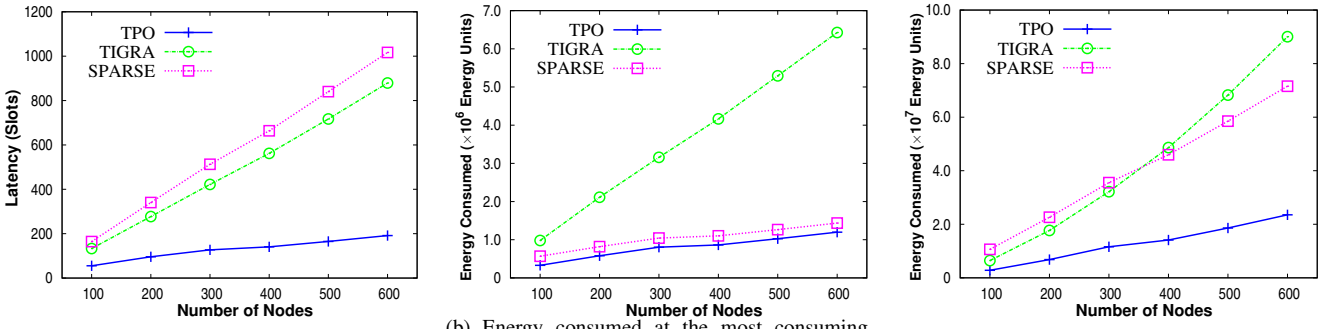
down pass can fit into one packet and accounted for the energy and latency overheads due to these messages in SPARSE. In contrast, TIGRA and our TPO algorithm consistently use a single schedule throughout all sampling intervals.

Each simulation run was performed for 20,000 sampling intervals. We compared different scheduling algorithms in terms of time efficiency and energy efficiency. For time efficiency, we recorded the latency of data collection at each sampling interval and calculated the average latency over the 20,000 sampling intervals simulated. For energy efficiency, as discussed in Section 5.4, the energy costs for a sensor node to perform transmission and listen for transmission in a time slot were set at 1 and 0.75 energy units respectively. We recorded the total amount of energy consumed by each node for transmitting data, receiving data and idle listening over the 20,000 sampling intervals simulated. Then, we added up these amounts to obtain the total energy consumed in the whole network. Since the network lifetime is largely determined by the most energy-consuming node [31], we also found out the node that consumed the largest amount of energy and plotted this amount for performance comparison.

6.2 Performance for Different Traffic Patterns

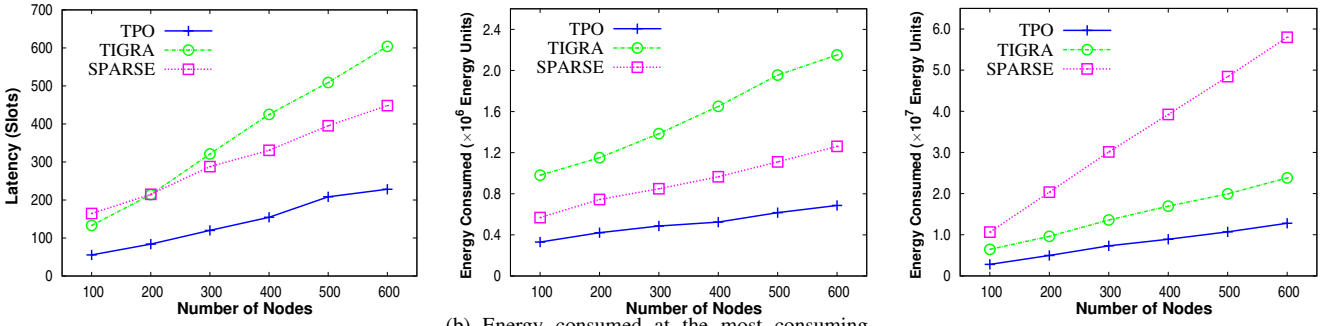
Figs. 11 and 12 show the performance of different scheduling algorithms as a function of the error bound of approximate data collection for the TEMP and SOLAR traces respectively. Figs. 11(a) and 12(a) show that the TIGRA schedule produces the same latency of data collection for all error bounds. This is because TIGRA targets at the full traffic pattern. But in approximate data collection, the traffic pattern changes over time in an unpredictable manner. Since the base station does not know a priori which nodes would report data in a sampling interval, it has to always listen until the end of the TIGRA schedule to make sure that all data have been received. In contrast, TPO and SPARSE are both able to take advantage of lighter traffic load at larger error bounds to reduce the latency of data collection. However, for the sample network topology shown in Fig. 10(a), the latency overhead of SPARSE for constructing a new schedule at each sampling interval is 136 time slots, which far outweighs the benefit of deploying the new schedule for data collection. Therefore, as seen from Figs. 11(a) and 12(a), the latency of SPARSE is much higher than that of TIGRA, even when the error bound is large. Our proposed TPO schedule considerably reduces the latency of data collection compared to TIGRA and SPARSE over a wide range of error bounds. Note that TPO also has decent latency performance under the full traffic pattern (the leftmost points in Fig. 11(a) and 12(a)) in that TPO has a similar latency to TIGRA, which is a latency-optimized schedule designed specifically for the full traffic pattern.

Figs. 11(b), (c) and 12(b), (c) show that our proposed TPO schedule leads to significantly lower energy consumption than TIGRA and SPARSE. In the TIGRA schedule, irrespective of the traffic pattern, the number of time slots each node has to listen always equals the number of its descendants in the routing tree. Therefore, the nodes close to the base station spend much energy in idle listening when the traffic load is light. This explains why the energy consumption of the



(a) Average latency of data collection.
(b) Energy consumed at the most consuming node.
(c) Total energy consumed in the whole network.

Fig. 13: Performance for different network sizes where the node density is kept constant (TEMP trace, error bound = 0.8 F).



(a) Average latency of data collection.
(b) Energy consumed at the most consuming node.
(c) Total energy consumed in the whole network.

Fig. 14: Performance for different network sizes where the field size is kept constant (TEMP trace, error bound = 0.8 F).

most consuming node is generally much higher in TIGRA than in the other schedules (see Figs. 11(b) and 12(b)). On the other hand, while the SPARSE schedule is able to avoid idle listening, the energy overhead for constructing a new schedule at each sampling interval is substantial. Thus, as shown in Figs. 11(c) and 12(c), SPARSE results in even higher total energy consumption in the whole network than TIGRA. Our proposed TPO schedule allows each node to transmit all data in its successive transmission slots regardless of the traffic pattern. Therefore, a receiving node can stop listening to a sending node when it identifies the end of transmissions performed by the sending node, thereby conserving energy. Figs. 11(b) and 12(b) show that the most consuming node in TPO spends much less energy compared to that in TIGRA and SPARSE. This implies our TPO schedule can substantially prolong network lifetime over the other two algorithms.

6.3 Impact of Network Size

To investigate the impact of network size, we performed two experiments that increased the number of sensor nodes in different ways. In the first experiment, we expanded the field size and increased the number of nodes concurrently to maintain the same node density. This makes the routing tree become deeper when the network size grows. In the second experiment, we maintained a constant field size as more nodes were added into the network. This results in increasing node density and hence makes the routing tree become fatter. Since the experimental results for the TEMP and SOLAR traces with different error bounds of data collection have similar performance trends, we report only the results for the TEMP trace with an error bound of 0.8 F in Figs. 13 and 14. As can be seen, the latency of and the energy consumed for data collection both increase with the network size. Our proposed TPO schedule consistently results in significantly lower latency and energy consumption than SPARSE and TIGRA for different network sizes. The performance improvement of the TPO schedule generally increases with the network size.

7 Conclusions

We have presented a TDMA schedule that is suited to continuous data collection with dynamic traffic patterns. Our proposed schedule is traffic pattern oblivious in that it achieves high energy efficiency and time efficiency of data collection irrespective of the traffic pattern. In this schedule, the energy consumed by sensor nodes for any traffic pattern is very close to the minimum required by their workloads given in the traffic pattern. The schedule also allows the base station to conclude data collection as early as possible according to the traffic load. We have theoretically analyzed the performance of our proposed schedule and compared it with a state-of-the-art TDMA schedule and a yardstick ideal schedule. We have also conducted simulation experiments of approximate data collection using real-world data traces. The analytical and simulation results show that: (1) existing schedules built for a static traffic pattern lead to a lot of idle listening and unnecessary delay when handling traffic patterns of lighter loads; (2) in continuous data collection with dynamic traffic patterns, constructing and deploying a new schedule tailored to the new

traffic pattern whenever the traffic pattern changes introduce too much energy and latency overheads; (3) compared with existing schedules, our proposed schedule considerably reduces the latency of data collection and achieves significant energy savings for continuous data collection with dynamic traffic patterns. For future work, we plan to show the effectiveness of the proposed schedule on a real sensor network testbed.

REFERENCES

[1] W. Zhao and X. Tang, "Scheduling Data Collection with Dynamic Traffic Patterns in Wireless Sensor Networks," *Proc. IEEE INFOCOM Mini-Conference '11*, pp. 286–290, Apr. 2011.

[2] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh, "Deploying a Wireless Sensor Network on an Active Volcano," *IEEE Internet Computing*, vol. 10, no. 2, Mar.-Apr. 2006.

[3] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin, "Habitat Monitoring with Sensor Networks," *Communications of the ACM*, vol. 47, no. 6, pp. 34–40, Jun. 2004.

[4] J. Gehrke and S. Madden, "Query Processing in Sensor Networks," *IEEE Pervasive Computing*, vol. 3, no. 1, pp. 46–55, Jan.-Mar. 2004.

[5] S. Gandham, M. Dawande, and R. Prakash, "Link Scheduling in Sensor Networks: Distributed Edge Coloring Revisited," *Proc. IEEE INFOCOM '05*, pp. 2492–2501, Mar. 2005.

[6] S. Gandham, Y. Zhang, and Q. Huang, "Distributed Minimal Time Convergecast Scheduling in Wireless Sensor Networks," *Proc. IEEE ICDCS '06*, Jul. 2006.

[7] W. Song, F. Yuan, and R. LaHusen, "Time-Optimum Packet Scheduling for Many-to-One Routing in Wireless Sensor Networks," *Proc. IEEE MASS '06*, pp. 81–90, Oct. 2006.

[8] Y. Zhang, S. Gandham, and Q. Huang, "Distributed Minimal Time Convergecast Scheduling for Small or Sparse Data Sources," *Proc. IEEE RTSS '07*, pp. 301–310, Dec. 2007.

[9] L. Paradis and Q. Han, "TIGRA: Timely Sensor Data Collection Using Distributed Graph Coloring," *Proc. IEEE PerCom '08*, Mar. 2008.

[10] J. Ma, W. Lou, Y. Wu, X. Li, and G. Chen, "Energy Efficient TDMA Sleep Scheduling in Wireless Sensor Networks," *Proc. IEEE INFOCOM '09*, pp. 630–638, Apr. 2009.

[11] O. Chipara, C. Lu, and J. Stankovic, "Dynamic Conflict-free Query Scheduling for Wireless Sensor Networks," *Proc. IEEE ICNP '06*, pp. 321–331, Nov. 2006.

[12] S. Gobriel, D. Mosse, and R. Cleric, "TDMA-ASAP: Sensor Network TDMA Scheduling with Adaptive Slot-Stealing and Parallelism," *Proc. IEEE ICDCS '09*, pp. 458–465, Jun. 2009.

[13] H. Wu, Q.Luo, and W. Xue, "Distributed Cross-Layer Scheduling for In-Network Sensor Query Processing," *Proc. IEEE PerCom '06*, pp. 180–189, Mar. 2006.

[14] P. Wan, S. Huang, L. Wang, Z. Wan, and X. Jia, "Minimum-Latency Aggregation Scheduling in Multihop Wireless Networks," *Proc. ACM MobiHoc '09*, pp. 185–194, May 2009.

[15] B. Yu, J. Li, and Y. Li, "Distributed Data Aggregation Scheduling in Wireless Sensor Networks," *Proc. IEEE INFOCOM '09*, pp. 2159–2167, Apr. 2009.

[16] Y. Li, L. Guo, and S. Prasad, "An Energy-Efficient Distributed Algorithm for Minimum-Latency Aggregation Scheduling in Wireless Sensor Networks," *Proc. IEEE ICDCS '10*, Jun. 2010.

[17] A. Silberstein, R. Braynard, and J. Yang, "Constraint Chaining: On Energy-Efficient Continuous Monitoring in Sensor Networks," *Proc. ACM SIGMOD '06*, pp. 157–168, Jun. 2006.

[18] D. Chu, A. Deshpande, J. Hellerstein, and W. Hong, "Approximate Data Collection in Sensor Networks using Probabilistic Models," *Proc. IEEE ICDE '06*, Apr. 2006.

[19] D. Wang, J. Xu, J. Liu, and F. Wang, "Mobile Filtering for Error-Bounded Data Collection in Sensor Networks," *Proc. IEEE ICDCS '08*, pp. 530–537, Jun. 2008.

[20] Y. Liu and M. Li, "Iso-Map: Energy-Efficient Contour Mapping in Wireless Sensor Networks," *Proc. IEEE ICDCS '07*, Jun. 2007.

[21] N. Xu, S. Rangwala, K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin, "A Wireless Sensor Network for Structural Monitoring," *Proc. ACM SenSys '04*, pp. 13–24, Nov. 2004.

[22] T. Wark, P. Corke, P. Sikka, L. Klingbeil, Y. Guo, C. Crossman, P. Valencia, D. Swain, and G. Bishop-Hurley, "Transforming Agriculture through Pervasive Wireless Sensor Networks," *IEEE Pervasive Computing*, vol. 6, no. 2, pp. 50–57, Apr.-Jun. 2007.

[23] C. Luo, F. Wu, J. Sun, and C. Chen, "Compressive Data Gathering for Large-Scale Wireless Wensor Networks," *Proc. ACM MobiCom '09*, pp. 145–156, Sept. 2009.

[24] S. Chen, S. Tang, M. Huang, and Y. Wang, "Capacity of Data Collection in Arbitrary Wireless Sensor Networks," *Proc. IEEE INFOCOM '10*, pp. 1–5, Mar. 2010.

[25] S. Ji, Y. Li, and X. Jia, "Capacity of Dual-Radio Multi-Channel Wireless Sensor Networks for Continuous Data Collection," *Proc. IEEE INFOCOM '11*, pp. 1062–1070, Apr. 2011.

[26] S. Chen, Y. Wang, X. Li, and X. Shi, "Data Collection Capacity of Random-Deployed Wireless Sensor Networks," *Proc. IEEE GLOBECOM '09*, pp. 1–6, Nov. 2009.

[27] S. Ji, R. Beyah, and Y. Li, "Continuous Data Collection Capacity of Wireless Sensor Networks under Physical Interference Model," *Proc. IEEE MASS '11*, pp. 222–231, Oct. 2011.

[28] Y. Xu and W. Wang, "Scheduling Partition for Order Optimal Capacity in Large-Scale Wireless Networks," *Proc. ACM MobiCom '09*, pp. 109–120, Sept. 2009.

[29] B. Liu, D. Towsley, and A. Swami, "Data Gathering Capacity of Large Scale Multihop Wireless Networks," *Proc. IEEE MobiHoc '08*, pp. 124–132, May 2008.

[30] E. Duarte-Melo and M. Liu, "Data-Gathering Wireless Sensor Networks: Organization and Capacity," *Computer Networks*, vol. 43, no. 4, pp. 519–537, Nov. 2003.

[31] C. Buragohain, D. Agrawal, and S. Suri, "Power Aware Routing for Sensor Databases," *Proc. IEEE INFOCOM '05*, Mar. 2005.

[32] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "TinyDB: An Acquisitional Query Processing System for Sensor Networks," *ACM Trans. on Database Systems*, vol. 30, no. 1, pp. 122–173, Mar. 2005.

[33] B. Sundararaman, U. Buy, and A. Kshemkalyani, "Clock Synchronization for Wireless Sensor Networks: A Survey," *Ad Hoc Networks*, vol. 3, no. 3, pp. 281–323, Jan. 2005.

[34] G. Zhou, T. He, J. Stankovic, and T. Abdelzaher, "RID: Radio Interference Detection in Wireless Sensor Networks," *Proc. IEEE INFOCOM '05*, pp. 891–901, Mar. 2005.

[35] B. Awerbuch, "Complexity of Network Synchronization," *Journal of the ACM*, vol. 32, no. 4, pp. 804–823, Oct. 1985.

[36] J. Polastre, J. Hill, and D. Culler, "Versatile Low Power Media Access for Wireless Sensor Networks," *Proc. ACM SenSys '04*, Nov. 2004.

[37] W. Ye, J. Heidemann, and D. Estrin, "An Energy-Efficient MAC Protocol for Wireless Sensor Networks," *Proc. IEEE INFOCOM '02*, pp. 1567–1576, Jun. 2002.

[38] J. Liang, J. Wang, J. Cao, J. Chen, and M. Lu, "An Efficient Algorithm for Constructing Maximum Lifetime Tree for Data Gathering without Aggregation in Wireless Sensor Networks," *Proc. IEEE INFOCOM Mini-Conference '10*, pp. 1–5, Mar. 2010.

[39] Live From Earth And Mars (LEM) Project, http://www-k12.atmos.washington.edu/k12/, 2009.

PLACE PHOTO HERE

Wenbo Zhao received the BSc degree from the Computer Science Department, Nanjing University of Astronautics and Aeronautics, China in 2006. She is currently pursuing the PhD degree in the Department of Computer Science at the Nanyang Technological University. Her research interests include the design of energy efficient MAC and routing protocols for wireless, ad hoc and sensor networks.

PLACE PHOTO HERE

Xueyan Tang received the BEng degree in computer science and engineering from Shanghai Jiao Tong University, Shanghai, China, in 1998 and the PhD degree in computer science from the Hong Kong University of Science and Technology, in 2003. He is currently an Associate Professor in the School of Computer Engineering, Nanyang Technological University, Singapore. His research interests include distributed systems, mobile and pervasive computing, and wireless sensor networks.