Distributed Real Time TDMA Scheduling Algorithm for Tree Topology WSNs

Aasem Ahmad * **Zdeněk Hanzálek.** **

* *DCE, FEE, CTU in Prague Prague, Czech Republic (e-mail: ahmadaas@fel.cvut.cz).*
** *CIIRC, DCE, FEE, CTU in Prague Prague, Czech Republic (e-mail: hanzalek@fel.cvut.cz).*

Abstract: In this paper, we address the problem of developing *TDMA* scheduling algorithm for tree topology *WSNs*. The data transmissions are organized into periodic data flows that may have opposite directions since they are carrying sensor and actuator values for feedback control. It is required to determine a periodic and collision-free allocation of the time-slots to the sensor nodes such that the *end-to-end deadline* of each data flow, as given in time units, is satisfied. The objective is to maximize the lifetime of the network by maximizing the time when the nodes are in the sleep mode. However, the longer the time at which the nodes stay in the sleep mode, the harder is to meet the timeliness requirements of the data flows. To solve the *TDMA* scheduling problem, we have found an elegant approach to express the *end-to-end deadline* as an integer number of the length of the schedule period. Moreover, since the distributed algorithms, in compassion with the centralized algorithms, well-suit the scarce resources of the *WSNs*, we focus on the distributed methods that allow each node in the network to come up with its allocated time-slots in the schedule. The proposed algorithm is based on the graph theory algorithms, namely the distributed shortest path and the distributed topological ordering. Furthermore, it falls into the category of the exact algorithms for tree topology with single-collision domain and in the category of the heuristic algorithms for multiple-collision domains tree topology.

Keywords: Wireless Sensor Networks, ZigBee cluster-tree topology, distributed algorithms, energy efficiency.

1. INTRODUCTION

With the rapid utilization of wireless sensor networks (WSNs) in various industrial applications including monitoring and control systems, and with the fact that sensor nodes are usually battery powered, energy-efficient *Medium Access Control* (MAC) protocols are becoming more required. Also, other requirements of industrial applications such as timeliness, robustness, and on-the-fly deployment and configurations are essential.

Concerning the above-mentioned pivotal demands, the *MAC* protocols based on *Time Division Multiple Access* (TDMA) outperform *MAC* protocols based on *Carrier Sense Multiple Access* (CSMA) as revealed by Ergen and Varaiya (2006). The *TDMA* strategy slices the time domain (i.e., the period of the schedule) into equal sized time-slots and nodes are assigned a proper number of time-slots relevant to the payload of the data to be transmitted by the node. Thus, the nodes are in sleep mode until their allocated time-slots in order to save energy. To avoid the collision occurrence, when the nodes are not far enough from each other, it is mandatory to assign a different set of time-slots to each node. On the other hand, the time-slots might be shared between the nodes that are not in the neighborhood (i.e., spatial reuse of the transmission medium which is not considered in this paper). Since *TDMA* scheduling algorithms eliminate the collision occurrence and seek for minimizing the number of time-slots assigned to each node, the energy consumption of the nodes is reduced. Furthermore, with the proper ordering of the allocated time-slots, the transmission delay can be significantly reduced as shown by Moriyama and Zhang (2015); Ahmad et al. (2014).

In this paper, we focus on the problem of determining a periodic collision-free allocation of the time-slots to the nodes, which are organized in a tree topology. All nodes may have sensing and/or actuating capabilities; therefore, they can be sources and/or sinks of the data transmissions organized as periodic data flows. Since *WSNs* for control and monitoring applications introduce critical constraints on the design of *TDMA* schedule to guarantee timely data delivery (Zheng et al. (2014)), each flow is constrained by *end-to-end deadline* and given in time units. Moreover, to support the control applications where the data goes in both directions simultaneously (i.e., sensed data and control data from and to the field devices), we consider the case when the data flows traverse the tree topology simultaneously in opposite directions. Since wireless nodes are usually battery-powered, the energy efficiency of the *TDMA* schedule is a problem of paramount importance in order to maximize the lifetime of the network. Hence,

⋆ Sponsor and financial support acknowledgment goes here. Paper titles should be written in uppercase and lowercase letters, not all uppercase.

the objective is to maximize the time when the nodes are in sleep mode. However, the longer the time at which the nodes stay in the sleep mode, the harder is to satisfy the *end-to-end deadline* requirements especially when the data flows have opposite directions.

To solve the *TDMA* scheduling problem, we have found an elegant approach to express the *end-to-end deadline* as an integer number of the length of the schedule period. Also, since the distributed methods well-suit the scarce properties of the nodes especially related to memory size, power consumption, and processor performance (Lenzen and Wattenhofer (2011)), we focus on the distributed algorithm to solve the scheduling problem.

The proposed scheduling algorithm falls into the category of the exact algorithms for *WSNs* with single-collision domain tree topology where the spatial reuse of the transmission medium is prohibited to avoid collisions. Since the feasible *TDMA* schedule with no consideration of the spatial reuse of the transmission medium is also a feasible schedule to the generalized problem (i.e., multiple-collision domains tree topology), then the proposed algorithm also falls into the category of the heuristic algorithms for the generalized problem. The worth of the heuristic in such case is its fast computation time due to less complexity and consequently, more energy-efficient. The shortcoming of the heuristic lies behind the less efficient utilization of the bandwidth since the time-slots are not shared between the nodes. Therefore, the possibility of missing the feasible schedule when the schedule period is so tight.

2. RELATED WORK

Many researchers tackled the *TDMA* scheduling problem with various requirements. R. Severino and Tovar (2014) presented a dynamic centralized *TDMA* scheduling algorithm for single-collision domain *cluster-tree WSN* where all flows are directed to the root node of the tree. Yu et al. (2009) presented a distributed *TDMA* scheduling algorithm for collision-free data aggregation in large scale *WSNs* based on the maximal independent sets. Ergen and Varaiya (2010) showed that the *TDMA* scheduling problem is an $\mathcal{NP}$-hard for multiple-collision domains *WSNs*. Moreover, the authors designed distributed graph coloring algorithms to minimize the total number of the time-slots assigned to the nodes. However, none of the previously mentioned papers assumed time-constrained data flows or data flows in opposite directions.

The work proposed by Hanzálek and Jurčík (2010) is the most related to ours. The authors considered multiple-collision domains ZigBee *cluster-tree* topology and data flows constrained by the precise *end-to-end deadline* given in time units. The *TDMA* scheduling algorithm is centralized and based on *Integer Linear Programming* (ILP) which can solve small size instances. Ahmad et al. (2014) expressed the precise *end-to-end deadline*, as considered by Hanzálek and Jurčík (2010), into the maximum number of periods crossed by each data flow till its delivery. The authors also proposed an exact centralized algorithm that is able to solve instantly large size deployments without the consideration of the spatial reuse of the transmission medium. The algorithm is based on the shortest path tree and the topological ordering algorithms. Int this paper, we also adopt the scenario proposed by Ahmad et al. (2014) with the aim of developing the distributed counterpart of the centralized algorithm.

Table 1. User-defined data flows parameters.

flow ID q	*source* (α_{f_q})	*sink* (β_{f_q})	*sampleSize* [bit]	*reqPeriod* [s]	*e2eDeadline* [s]	*sampleACK*
1	1	9	64	1	2	1
2	6	10	16	2	6	0
3	11	12	16	1	2	1
4	12	1	64	2	4	0

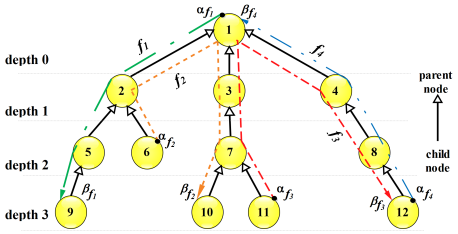

Fig. 1. Tree *WSN* with four time-constrained data flows.

The rest of the paper is organized as follows. In Sec. 3, the scheduling problem is described in details. The overview of the centralized *TDMA* scheduling algorithm is highlighted in Sec. 4. The counterpart distributed *TDMA* algorithm is explained in Sec. 5. The experimental results are presented in Sec. 6. Finally, we draw the conclusion in Sec. 7.

3. SYSTEM MODEL

3.1 Tree Topology

We consider a tree topology with n nodes (Fig. 1). Each pair of connected nodes use the shared bidirectional wireless link for data transmission. Each node has its depth where the root node, node 1, has depth 0. The logical topology is expressed by the parent-child relation between the connected nodes in a similar manner to the ZigBee *cluster-tree* topology where each cluster is composed of the parent node with its child nodes Org. (2006).

3.2 Data flow model

The transmissions are organized into data flows (see Fig. 1 where 4 data flows are illustrated as dashed directed lines). Each data flow has one source node α_{f_q} and one sink node β_{f_q}. The user defined parameters for each flow are shown in Tab. 1. A source node periodically measures a sensed value with a given size and required period denoted by $sampleSize_{f_q}$ and $reqPeriod_{f_q}$ respectively and reports it to the sink. The $reqPeriod_{f_q}$ defines the minimal interarrival time between two consecutive measurements such that the particular interarrival time for flow f_q is greater or equal to the given $reqPeriod_{f_q}$. Also, Each flow is constrained by *end-to-end deadline* denoted by *e2eDeadline* and given in time units. The *e2eDeadline* specifies the maximum allowed elapsing time between the instant when the source sends the packet to the instant when the sink receives the packet. The acknowledged transmission can be requested by setting $sampleACK_{f_q} = 1$.

3.3 Node life cycle

The life cycle of each node, concerning the node period P, is divided into two parts, the active and inactive parts

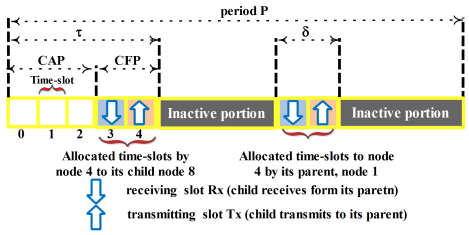

Fig. 2. The life cycle of node 4.

(see Fig. 2 where the life cycle of node 4 is illustrated). The active part is subdivided into time-slots of equal size and grouped into two portions, namely τ and δ. The time-slots of the τ portion are utilized by the node to exchange the data with its child nodes while the time-slots of the δ portion are utilized by the node to exchange the data with its parent node. The τ portion is partitioned into a *Contention Access Period* (CAP) and optional *Contention Free Period* (CFP). During the *CAP*, a slotted *CSMA/CA* protocol is used for the best-effort data delivery while during the *CFP*, the node periodically allocates time-slots to its child nodes that can be exploited for transmitting real-time traffic. The number of time-slots allocated to a given child node is relevant to the amount of data to be exchanged between the parent and that child. The time-slots used by child node to send data to its parent are denoted by Tx slots while Rx slots denote the time-slots for receiving the data from the parent node. During the inactive portion, the node is in a sleep mode to save energy. P and τ are defined by two parameters as follows:

$$P = A \cdot 2^{PO}, \quad \tau = A \cdot 2^{TO} \tag{1}$$

where $0 \leq TO \leq PO \leq 14$ and A denotes the minimum duration of τ when $TO = 0$. For ZigBee *cluster-tree* topology, $A = 15.36$ ms (assuming a 2.4 GHz frequency band and 250 kbps of bit rate). We assume that all nodes have an equal P, but various τ for better bandwidth utilization. In this paper, we skip the details related to the τ_i calculation given by the payload of the flows. Please refer to Ahmad et al. (2014) for more information. However, the required parameters for calculating the τ_i are taken into consideration while designing the distributed algorithm as will be explained in Sec. 5.2.

3.4 Cyclic schedule nature

The life cycle of each node, as explained in Sec. 3.3, leads to the so-called cyclic behavior of the periodic schedule (i.e., there is a flow whose delay is longer than P) when there are flows in opposite directions in a *WSN*. In such a case, the delay minimization of the flow f_i is in contradiction with the delay minimization of flow f_j when $\{f_i, f_j\}$ are in opposite directions. Let us denote a complete data communication from the source to the sink by a wave where the notation $f_{q,l}$ is used for the wave l of flow q. Consider nodes $\{1, 4, 8, 12\}$ in Fig. 1 and assume without loss of generality that $\tau_1 = \tau_4 = \tau_8 = \tau_{12} = 4$ *time-slots*. Then, ordering the τ portions as shown in Fig. 3(a) ensures that each wave of f_3 starts and completes in the same period (0 crossed period) while each wave of f_4 (the part from node 12 to node 1), requires three periods to reach the destination (i.e., 2 crossed periods). The reverse order of the τ portions, as shown in Fig. 3(b), leads to 2 crossed

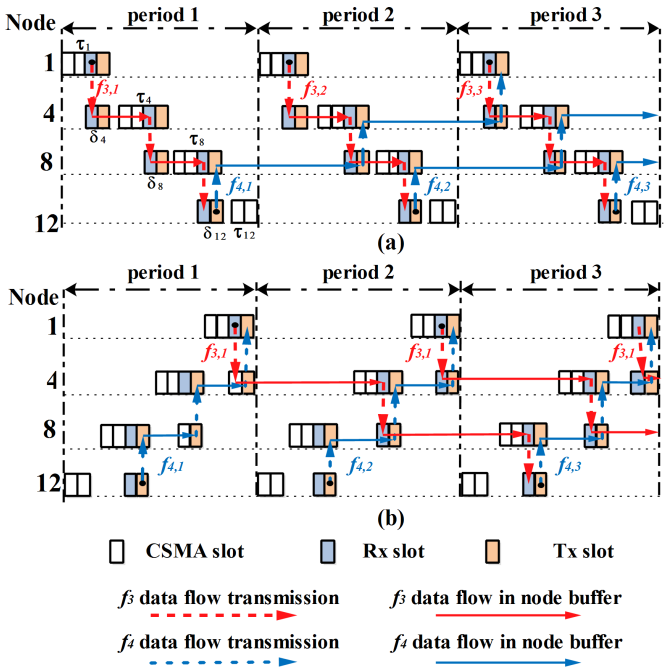

Fig. 3. (a) and (b): Cyclic nature of the nodes schedule

periods for each wave of f_3 and 0 crossed period for the considered wave of f_4. This observation enables one to express the *e2eDeadline* in terms of the maximum number of crossed periods h_{f_q}, where $e2eDeadline \geq P$ as follows:

$$h_{f_q} = \left\lfloor \frac{e2eDeadline_{f_q}}{P} \right\rfloor - 1 \tag{2}$$

Considering $P = 1$ s, then and the *e2eDeadline* as given in Tab. 1, we get: $h_{f_1} = 1$, $h_{f_2} = 2$, $h_{f_3} = 1$ and $h_{f_4} = 1$.

4. CENTRALIZED TDMA SCHEDULING ALGORITHM

For the sake of an easy understanding of our distributed algorithm, we present by example the core idea of the centralized algorithm. The *TDMA* scheduling problem is constrained by: the h_{f_q} of each flow f_q, P_{max} the upper bound of P and P_{min} the lower bound of P. PO_{max} is given by the shortest $reqPeriod_{f_q}$ among all of the flows as shown in Eq. (3). PO_{min} is rounded up to the nearest PO such that the resulting period P is large enough to accommodate the active portion for all the nodes when the spatial reuse of the transmission medium is not considered as shown in Eq. (4). Since each node is periodically activated for a fixed amount of time, then maximizing the period P, increases the inactive portion within P (i.e., the sleeping time of each node). Hence, to maximize the lifetime of the network, the objective is to maximize P, given by $PO \in \{PO_{min}, \ldots, PO_{max}\}$, provided that the h_{f_q} is met for each flow f_q. However, Since h_{f_q} is inversely proportional to the P (see Eq. (2)), then the longer the P, the harder is to satisfy the h_{f_q} of each f_q.

$$PO_{max} = \left\lfloor \log_2 \left(\frac{\min_q(reqPeriod_{f_q})}{A} \right) \right\rfloor \tag{3}$$

$$PO_{min} = \left\lceil \log_2 \left(\frac{\sum_{i=1}^{n} \tau_i}{A} \right) \right\rceil \tag{4}$$

$\forall\, i, j = 1 \ldots n : i = parent(j)$
$0 \leq D_j - D_i \leq 1$ (a)

$\forall f_q$ is a decreasing flow : $i = \beta_{f_q}$, $j = parent(\alpha_{f_q})$
$D_j - D_i \leq h_{f_q}$ (b)

$\forall f_q$ is an increasing flow : $i = parent(\beta_{f_q})$, $j = \alpha_{f_q}$
$D_j - D_i \leq h_{f_q} + depth(j) - depth(i)$ (c)

$\forall f_q$ is a bidirectional flow : $i = parent(\beta_{f_q})$, $j = parent(\alpha_{f_q})$
$D_j - D_i \leq h_{f_q} + depth(z_{f_q}) - depth(i)$ (d)

$\forall\, j = 1 \ldots n$
$D_j \geq 0$ (e)

Fig. 4. The constraint model.

$0 \leq D_j - D_i \leq 1$	topological constraints
$D_1 - D_5 \leq -1$	for flow f_1
$D_2 - D_7 \leq 0$	for flow f_2
$D_7 - D_8 \leq -1$	for flow f_3
$D_8 - D_1 \leq 1$	for flow f_4
$D_j \geq 0$	$\forall j = 1 \ldots n$

Fig. 5. The constraint model for the example in Fig. 1.

4.1 Modeling the deadline constraints of the data flows

Three types of flows may cross the tree: decreasing, increasing or bidirectional flow. f_q is a decreasing flow if for every two consecutive nodes i and j, crossed by the flow from its source α_{f_q} to its sink β_{f_q}, $depth(i) > depth(j)$. f_q is an increasing flow if for every two consecutive nodes i and j, crossed by the flow from its source α_{f_q} to its sink β_{f_q}, $depth(i) < depth(j)$. The flow f_q is a bidirectional flow where a node z_{f_q} crossed by f_q exists such that the part from α_{f_q} to z_{f_q} is decreasing and the part from z_{f_q} to β_{f_q} is increasing. For the flows in Fig. 1, f_1 is an increasing, f_4 is a decreasing while f_2, f_3 are bidirectional.

As shown in Fig. 3, for every tow consecutive nodes i and j crossed by flow f_q, the precedence decision between τ_i and τ_j is the key factor for satisfying h_{f_q}. Let the triple $(i, \rightarrow, j)$ denote that τ_i is followed by τ_j in the schedule while the triple $(i, \leftarrow, j)$ denote that τ_j is followed by τ_i in the schedule such that $i = parent(j)$. Then for a given flow f_q and the set of precedence decisions between every two consecutive nodes on the path of f_q, the number of crossed periods of f_q is given by the number of precedence decisions (i.e., arcs in the triples) that are directed in opposite direction to the flow direction. However, if the first hop of f_q is a child-parent hop, e.g. f_4, the precedence decision between the nodes α_{f_q} and $parent(\alpha_{f_q})$ is not contributing to the resulting number of crossed periods of f_q. Also, if the final hop of f_q is a parent-child hop, e.g. f_1, the precedence decision between the nodes $parent(\beta_{f_q})$ and β_{f_q} is also not contributing to the resulting number of crossed periods of f_q.

The above-mentioned observation is the essence of the proposed constraint model as shown in Fig. 4 where D_j is an integer decision variable associated with each node j and represents the number of the precedence decisions with a forward direction ($\rightarrow$) on the unique path from node 1 to node j. The constraint (4a) is the topological constraint which ensures that for each two nodes i and j such that $i = parent(j)$, the value of D_j is equal to or greater than the value of D_i by one. The constraint (4b) bounds the number of precedence decisions ($\rightarrow$) from

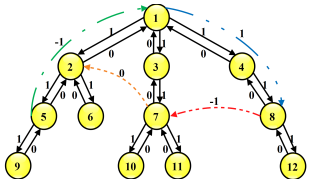

Fig. 6. The inequality graph Q for the constraint model shown in Fig. 5.

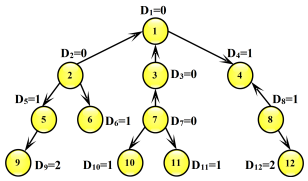

Fig. 7. *PONA* graph after applying Bellman-Ford on the Q graph shown in Fig. 6.

$j = parent(\alpha_{f_q})$ to node $i = \beta_{f_q}$ to be, at most, h_{f_q} when f_q is a decreasing flow. The constraint (4c) bounds the number of the precedence decisions with a backward direction ($\leftarrow$) from node $j = \alpha_{f_q}$ to node $i = parent(\beta_{f_q})$ to be, at most, h_{f_q} when f_q is an increasing flow. The constraint (4d) is is for the bidirectional flow which can be derived from the combination of the increasing and decreasing flow constraints. The constraint (4e) ensures non negative value of D_i. For the example in Fig. 1 and based on the values of h_{f_q} that are calculated in Sec. 3.4, the resulting inequality constraints are illustrated in Fig. 5.

4.2 Inequality graph Q and PONA garph

The constraints in Fig. 4 have the form of $D_j - D_i \leq const$ and can be sketched as an inequality graph $Q(V, E)$. Where V represents the nodes while for each constraint, an edge is added from node i to node j and weighted by *const*. The inequality graph, for the constraints in Fig. 5, is depicted in Fig. 6. The set of solid edges represents the topological constraints while the set of dashed ones represents the data flows constraints. The constraint model can be solved in polynomial time such that D_j is the length of the shortest path from node 1 to node j in the inequality graph Q shown in Fig. 6. Since some edges in Fig. 6 may have negative weights, a negative cycle may exist and consequently, the constraint model is infeasible. Therefore, the nonexistence of a negative cycle is a necessary condition for the feasibility of the scheduling problem. Using the Bellman-Ford shortest path algorithm, we get $D = (0, 0, 0, 1, 1, 1, 0, 1, 2, 1, 1, 2)$ and the resulting chosen precedence decisions are depicted by *Partial Order of Node Activation* (PONA) graph as shown in Fig. 7. $E(PONA)$ represents the precedence decisions between each node and its child nodes. Obviously, the *PONA* graph is a *Directed Acyclic Graph* (DAC). Hence, one topological ordering, at least, exists for $V(PONA)$ and consequently for τ_i portion for each node $i \in V(PONA)$. The following order of the nodes $(2, 5, 9, 6, 7, 10, 11, 3, 1, 8, 12, 4)$ represents a feasible schedule where τ_2 are forwarded by τ_5 and so on.

5. DISTRIBUTED TDMA SCHEDULING ALGORITHM

The distributed *TDMA* scheduling algorithm consists of multiple stages. During each stage, the *CSMA/CA* method is utilized by the nodes for data communications. We assume that the network topology has been set up and that each node i maintains the following input parameters:

(1) *id*: a unique identifier assigned to the node; $id = i$.
(2) $depth(i)$: the depth of the node i in the tree.

(3) $child(i)$: the set of child nodes of node i.
(4) $parent(i)$: the parent node of the node i.
(5) t_s: the duration of the fixed-size time-slot within τ_i.
(6) N_{csma}: the number of the time-slots within the CAP portion of τ_i.

Likewise in ZigBee, the $depth(i)$, $child(i)$, and $parent(i)$ parameters can be determined in distributed manner during the tree construction phase. During the stages of the algorithm, each node i collaborates with its parent and child nodes to determine the following parameters:

(1) n: the total number of nodes in the tree.
(2) P: the length of the schedule period.
(3) $subT(i)$: the set of nodes in the subtree rooted by node i, as given in Fig. 1, e.g. $subT(3) = \{7, 10, 11\}$.
(4) st_i: the current state of the node i. A node can be either in *not-ready*, *pre-ready*, or *ready* state.
(5) F_i^l (resp. F_i^e): the set of nodes that are the heads (resp. tails) of the dashed edges in Fig. 6 that represent the flows constraints and linked with node i. (e.g. $F_1^l = \{8\}$ and $F_1^e = \{5\}$).
(6) c_{ji}: the cost assigned to each edge entering node i in the inequality graph and maintained by node i such that $c_{ji} = 0$ if $j \in child(i)$, $c_{ji} = 1$ if $j = parent(i)$ and $c_{ji} = c_{f_q}$ if $j \in F_i^e$ where c_{f_q} represents the cost of flow f_q and calculated based on the type of the flow as shown in Fig. 4.
(7) D_i: the length of the shortest path from node 1 to node i in Fig. 6.
(8) τ_i: the length of the active portion.
(9) N_{Tx_j}, I_{Tx_j} (resp. N_{Rx_j}, I_{Rx_j}): the number and indices of Tx (resp. Rx) time-slots assigned by node i to each $j \in child(i)$.
(10) s_i: the index of the first slot within the τ portion that is allocated to the node in the schedule.

5.1 Distributed calculation of n and PO_{max}

Two phases encompass this stage of the algorithm. The first phase is triggered as the bottom-up pattern (i.e., from leaf nodes of the tree to the root node). Each node i sends to node $j = parent(i)$ one packet denoted by $RPER\text{-}N(i, j, |subT(i)|, PO^i_{max})$. The $|subT(i)|$ parameter indicates the number of nodes that belong to $subT(i)$. PO^i_{max} is given by Eq. (5) such that $PO^{i=\alpha_{f_q}}_{reqPeriod_{f_q}}$ is calculated by Eq. (3) considering only the flows with $\alpha_{f_q} = i$. If a node is not a source of any flow, then $PO^{i=\alpha_{f_q}}_{reqPeriod_{f_q}} = 14$. When the root node, node 1, receives $RPER\text{-}N$ from each node $j \in child(1)$, it also applies Eq. (5) and the second phase of this stage is commenced. In contrast to the first phase, the second phase is triggered as the up-bottom pattern. Each node i sends to each node $j \in child(i)$ one packet denoted by $PER\text{-}N(i, j, n, PO_{max})$ such that $n = |subT(1)| + 1$ is the total number of nodes in the given tree topology and $PO_{max} = PO^1_{max}$.

$$PO^i_{max} = \min\{\min_{k \in child(i)} PO^k_{max}, PO^{i=\alpha_{f_q}}_{reqPeriod_{f_q}}\} \quad (5)$$

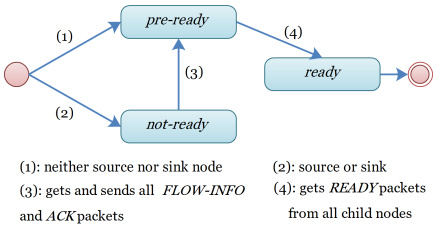

Fig. 8. Node state diagram for the first stage.

5.2 Distributed construction of the inequality graph

At this stage, each node i computes F_i^l and F_i^e in addition to c_{ji} for each node $j \in child(i) \cup parent(i) \cup F_i^e$. Initially, as given by transitions (1) and (2) in the node's state diagram shown in Fig. 8, $st_i = not\text{-}ready$ if node i is a source or a sink of a flow, otherwise $st_i = pre\text{-}ready$. When node i sends and receives all required *FLOW-INFO* and *ACK* packets, as will be explained later in this section, st_i is updated from *not-ready* to *pre-ready* as depicted by transition (3). Transition (4) implies updating st_i from *pre-ready* to *ready* when $st_k = ready \; \forall \, k \in subT(i)$. This stage lasts until st_1 is set to *ready*.

Each source node α_{f_q} sends a *FLOW-INFO* packet to the sink node β_{f_q}. The *FLOW-INFO* packet includes the following fields, which are set by node α_{f_q} following the observations in Sec. 4.1 and the constraint model in Fig. 4:

(1) *src_id*: it is set to the *id* of the source node α_{f_q}.
(2) *dest_id*: it is set to the *id* of the sink node β_{f_q}.
(3) *parent_id*: it is set to $parent(\alpha_{f_q})$. However, this field is set to -1 in case the first hop of the *FLOW-INFO* is a parent-child hop.
(4) *sample_size*: it is set to $sampleSize_{f_q}$. It is required for the calculation of τ_j for each node j crossed by f_q.
(5) *sample_ACK*: it is set to $sampleACK_{f_q}$. It is required for the calculation of τ_j for each node j crossed by f_q
(6) *deadline*: it is set to h_{f_q} as calculated by Eq. (2). However, this field is set to -1 if the first hop of *FLOW-INFO* is a parent-child hop. Hence, this filed is used for the c_{f_q} calculation by the node $parent(\alpha_{f_q})$ when the first hop of the *FLOW-INFO* is a child-parent hop.

The calculation of the F_i^l takes place at the last hop of the *FLOW-INFO*, such that $i = \beta_{f_q}$ if the last hop is a child-parent hope and $i = parent(\beta_{f_q})$ otherwise. The node i checks the *parent_id* field of the *FLOW-INFO* packet. If it is set to -1, then it adds the node $j = \alpha_{f_q}$ to F_i^l, otherwise it adds the node $j = parent(\alpha_{f_q})$ to F_i^l.

When the sink node β_{f_q} receives the *FLOW-INFO* packet, it sends an acknowledgment packet (ACK) to the source node α_{f_q} that consists of the following fields:

(1) *src_id*: the *id* of the *ACK* sender β_{f_q}.
(2) *dest_id*: the *id* of the *ACK* receiver α_{f_q}.
(3) *parent_id*: it is set to $parent(\beta_{f_q})$. However, this filed is set to -1 in case the first hop of the *ACK* is a parent-child hop.
(4) *flow_type*: it is equal to 1 for the increasing flow, -1 for the decreasing flow and 0 for the bidirectional flow. Since the flow f_q has an opposite direction to the

direction of the ACK packet, then if the first hop of the ACK packet is a child-parent hop, the *flow_type* is initialized to 1; otherwise to -1. This field is altered from -1 to 0 by the node z_{f_q} if the ACK reverses its direction from decreasing to increasing.

(5) *depth_parent*: it is set to $depth(parent(\beta_{f_q}))$ and used for c_{f_q} calculation when *flow_type* $= 1$ or *flow_type* $= 0$ (see constraints (4c) and (4d)).

(6) *depth_z_f*: it is initialized to -1. Node z_{f_q} sets this field to $depth(z_{f_q})$ for bidirectional flow f_q. It is used for c_{f_q} calculation when $flow\_type = 0$ (see Eq. (4d)).

The calculation of the F_j^e takes place at the last hop of the ACK packet such that $j = \alpha_{f_q}$ if the last hop is a child-parent hope and $j = parent(\alpha_{f_q})$ otherwise. The node j checks the *parent_id* field of the ACK packet. If it is set to -1, then it adds the node $i = \beta_{f_q}$ to F_j^e, otherwise it adds the node $i = parent(\beta_{f_q})$ to F_j^e. Moreover, node j calculates and maintains the c_{f_q} using the corresponding constraint in Fig. 4.

5.3 Calculation of D_i

The *Distance Vector* (DV) is one approach for the distributed calculation of the shortest path tree in a given graph (Bonaventure (2011)). However, the DV has the following limitations concerning the specification of the shortest path tree problem in the inequality graph shown in Fig. 6 : (i) it is an asynchronous algorithm, and since our proposed distributed $TDMA$ scheduling algorithm consists of multiple stages, then the synchronization between stages is required. (ii) it considers only communication between neighboring nodes. Since edges, that connect nodes which are not in neighboring relation, exist in Fig. 6, e.g. the edge (8,7), then considering those edges is crucial for the correctness of the algorithm. (iii) the existence of the negative cycles is not detected.

To cope with DV limitations, we propose a modified version of the DV as shown in the pseudo code in Alg. 1. Each node initializes PO to PO_{max} and the algorithm iterates till a feasible solution is found or PO becomes less than 0. The *feasible* variable denotes whether this stage terminates with a feasible solution or a negative cycle has been detected. Each node i initializes D_i to $depth(i)$. Then, the algorithm, for given PO, iterates for at most $n+1$ iterations ($n-1$ iterations, at most, for calculating D_i, one iteration for detecting the existence of negative cycles and one additional iteration to inform the nodes whether a feasible solution has been found). At each iteration, the *initialize*() function sets st_i as shown in the state diagram in Fig. 9 (transitions (1) and (2)). Since this stage is entirely not overlapping with the previous stage, then identical node states labels have been used.

D_i calculation utilizes mainly two types of packets. The *Single Hop Distance Value* (SHDV) and the *Multiple Hops Distance Value* (MHDV). Both packets have the same format that includes the *id* of the sender *src_id*, the *id* of the receiver *dest_id* and the estimated shortest distance $D_{i=src\_id}$. To accomplish the synchronization purposes, sending the $SHDV$ packets, at each iteration $r \leq n$, is firstly commenced by node 1. Notice that node 1 skips the statement at line 7 since $parent(1) = \emptyset$. As depicited at lines 9 and 10, whenever a node i receives $SHDV$ packet from node $j = parent(i)$, it applies $D_i = \min\{D_j + c_{ji}, D_i\}$. Then, node i sends $SHDV$ to node $j = parent(i)$ and to each node $k \in child(i)$. When node i receives $SHDV$ from every node $k \in child(i)$, it applies $D_i = \min\{D_k + c_{ki}, D_i\}$.

Furthermore, Each node i sends $MHDV$ packet to each node $j \in F_i^l$ (line 19). The $MHDV$ packet traverses the tree topology node by node until reaching node j (e.g. in Fig. 6, when node 1 sends $MHDV$ to node 5, the route will be $1 \rightarrow 2 \rightarrow 5$ such that node 2 forwards the packet to node 5 without updating its D_2 value). When the node j (e.g. node 5) receives the $MHDV$, it applies $D_j = \min\{D_i + c_{ij}, D_j\}\ \forall i \in F_j^e$.

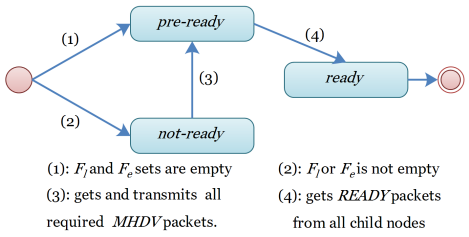

Fig. 9. Node state diagram during the second stage.

Algorithm 1: D_i calculation

```
1  PO ← PO_max
2  while PO ≥ 0 do
3      feasible ← true, D_i ← depth_i, r ← 1
4      while r ≤ n + 1 do
5          quit ← false, st_i ← initialize()
6          if r ≤ n then
7              pck ← receive packet form my parent node
8              if pck is SHDV then
9                  send SHDV to my parent and my child nodes
10                 receive SHDV from my child nodes
11             else if pck is STOP then
12                 send STOP to my child nodes
13                 quit ← true, r ← n + 1
14         else
15             pck ← receive STOP from my parent node
16             send STOP to my child nodes
17             quit ← true
18         if F_i^l ≠ ∅ and ¬quit then
19             send MHDV to each node j ∈ F_i^l
20         while ¬quit do
21             Route the packet in case receiving MHDV packet
22             st_i ← update_my_state()
23             if st_i is ready then
24                 send READY packet to my parent node
25                 if i is the root node in the tree then
26                     if none of the nodes has updated its D_i
                       then
27                         r ← n /* feasible remains true      */
28                     else if negative cycle is detected then
29                         feasible ← false, r ← n
30                 quit ← true
31         r ← r + 1
32     if feasible then
33         break
34     else
35         PO ← PO − 1
36         if PO ≥ 0 and F_i^e ≠ ∅ then
37             update the value of c_fq for each each edge (j, i) such
               that j ∈ F_i^e
```

Lines 20 to 30 determines when the node becomes *ready*. The node, that sends and receives all required *SHDV* and *MHDV* packets, updates its state to *pre-ready*. The *pre-ready* node becomes *ready* when it receives *READY* packets from all its child nodes. The *update_my_state*() function updates the state of the node as depicted in Fig. 9 (transitions (3), and (4)). Hence, each iteration, at each node, lasts until the node informs its parent about its *ready* state. Due to the Bottom-up pattern of sending the *READY* packets, node 1 is the last node that becomes *ready* at each iteration.

If an iteration terminates with no further changes of $D_{i=1...n}$, the D_i calculation can be terminated, as subsequent iterations will not make any more changes. The termination in such case avoids sending and receiving the redundant *SHDV* and *MHDV* packets. Hence, the *READY* packet, sent by a node i to its parent j at r^{th} iteration, also indicates whether D_k has been updated $\forall k \in \{i\} \cup subT(i)$. Hence, when node 1 becomes *ready* at r^{th} iteration, it is fully aware whether to terminate the D_i calculation. If termination is the case, it sends *STOP* packet that traverses the network from the root node to the leaf nodes to terminate the D_i calculation and to start the subsequent stage of the *TDMA* scheduling algorithm.

If a node i updates its D_i at the n^{th} iteration or D_i becomes negative at any iteration $r \leq n$, node i informs its parent node through the *READY* packet. Consequently, when the root receives the *READY* packets, it sends *STOP* packet to all nodes indicating to stop the scheduling algorithm for the given length of the period. Then each node i decreases the value of *PO* by 1 and updated the value of the c_{f_q} in case $F_i^e \neq \emptyset$ and the algorithm iterates again for the new given length of the period.

5.4 Nodes schedule and Rx, Tx allocations

Distributed topological ordering: Two phases encompass this part. During the first phase, each node i calculates the sum of τ_j of all nodes $j \in subT(i)$ in addition to τ_i. This sum is denoted by T_i (see Eq. (6)). To accomplish this task, each node i receives *Start Topological Ordering* (STO) from its child nodes, calculates its T_i value and sends $STO(i, j, T_i)$ packet to node $j = parent(i)$. This phase lasts until node 1 receives *STO* packets from its child nodes as illustrated in Fig. 10. The dashed directed edges represent the direction of the transmissions while the numbers in the squares represent the value of T_i. The τ duration for each node is depicted next to the node and the $N_{csma} = 2$ *time-slots*. Notice that in case $T_1 > P$, then there is no schedule fits into the period, and the *TDMA* scheduling algorithm terminates with no feasible schedule.

$$T_i = \sum_{j \in child(i)} T_j + \tau_i \tag{6}$$

The second phase is triggered in the opposite direction to first phase (i.e., from node 1 to the leaf nodes) and illustrated in Fig. 11. Each parent node partitions its child nodes into two sets, namely the *pred* set and the *succ* set. The *pred* set includes the child nodes which are predecessors to the parent node while the *succ* set includes the child nodes which are successors to the parent node. The partitioning procedure is accomplished depending on the value of D_i and D_j where $j \in child(i)$ (i.e., $j \in pred(i)$ if $D_i = D_j$; otherwise $j \in succ(i)$). Nodes in both sets are sorted in ascending order based on the *id*. The parent node then calculates its start slot in the schedule denoted by s_i as shown in Eq. (7). Also, it calculates the release slot denoted by r_j for each $j \in child(i)$ as shown in Eq. (8a), (8b) and (8c). Then, the parent node i sends the *End Topological Ordering* $ETO(s_i, r_j, I_{Tx_j}, N_{Tx_j}, I_{Rx_j}, N_{Rx_j})$ packet to each child node $j \in child(i)$.

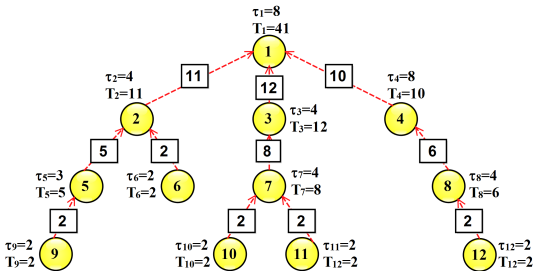

Fig. 10. Distributed topological ordering (first phase).

Each child node j utilizes the received *ETO* packet to synchronize its assigned Tx, Rx with its parent (i.e., the child node sends the data to its parent during the time-slots interval given by $[s_i + I_{Tx_j}, s_i + I_{Tx_j} + N_{Tx_j}]$ and receives data from its parent node during the time-slots interval given by $[s_i + I_{Rx_j}, s_i + I_{Rx_j} + N_{Rx_j}]$. This phase lasts until each node receives the *ETO* packet and calculates its start time in the schedule.

$$s_i = \sum_{j \in pred(i)} T_j + r_i \tag{7}$$

$$r_j = \begin{cases} 0 & \text{if } j = root \quad (8a) \\ r_i + \sum_{\substack{k \in pred(i) \\ k<j}} T_k & \text{if } D_i = D_j : j \in child(i) \quad (8b) \\ s_i + \tau_i + \sum_{\substack{k \in succ(i) \\ k<j}} T_k & \text{if } D_i = D_j - 1 : j \in child(i) \quad (8c) \end{cases}$$

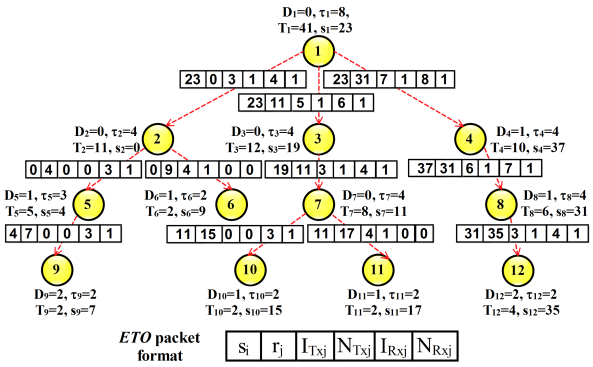

Fig. 11. Distributed topological ordering (second phase).

6. EXPERIMENTAL RESULTS

The distributed algorithm was implemented in Java such that each node is represented as a thread. Testing the algorithm was performed on a PC with an Intel Core i7 3520M Dual-core (2.9 GH) CPU and 8 GB RAM.

Each problem instance consists of a tree topology with a given set of nodes as shown in the first column in Tab. 2.

The maximum number of child nodes of every node is shown in the second column. For each tree topology, we generate a various number of data flows denoted by #flows where each flow has one source and one sink. The deadline of each flow is given in the number of crossed periods and shown in deadline column. The $sampleSize = 64$ bits and $sampleACK = 0$. For each tree topology and each combination of #flows and given deadline, we generate a set of 30 instances. Due to different kinds of disturbances, the transmission over the channel might be lost in real case *WSNs*. Thus, and for the sake of simplicity, we assume that the channel error rate is fixed to 30%. Hence, the node drops the received packet with the probability of 0.3 (Jurk and Hanzlek (2010)).

The average number of packets transmitted by the nodes, for the given 30 instances, is shown in avg(#pck) column. Furthermore, Due to the tree topology of the network, and the utilization of the 3 types of multiple hops packets, namely *FLOW-INFO*, *ACK* and *MHDV*, then the lower the depth value of the node, the more packets are forwarded by the node. Thus, we also show the average number of the maximum numbers of the transmitted packets by the nodes per each instance as shown in avgmax(#pck) column. The results consider only the feasible instances and no aggregation of the packets is considered. The results indicate that increasing the number of flows in given tree topology increases the number of transmitted packets. However, having less time-constrained data flows by increasing the value of the flow's deadline, decreases the number of iterations required by the D_i calculation stage which leads to a smaller number of transmitted packets.

The energy consumption metrics of the nodes are shown in columns avg(E) and avgmax(E) and given in millijoule $[mJ]$. E is calculated by the formula: $E = U \cdot I \cdot t$ where U is the voltage, I is the current drawn and t is the execution time (i.e., transmitting and receiving times). The particular voltage is 3 volts and the current drawn were given as follows: the current drawn in *receive mode* = 18.2 mA, *transmit mode* = 19.2 mA at 0 dBm (Hanzálek and Jurčík (2010)). Hence, we consider only the energy consumption due to transmitting and receiving the packets. We assume a 2.4 GHz frequency band and 250 kbps of bit rate. The results prove the energy efficiency of our proposed distributed algorithm. Even though the number of transmitted packets might be high, Nevertheless, the packets are of a small size and consequently the energy consumption is low.

Table 2. The number of packets metrics and the energy consumption of the nodes.

Problem Instance				Number of Messages		Energy Consumption	
#nodes	#child	#flows	deadline [#P]	avg(#pck)	avgmax(#pck)	avg(E) [mJ]	avgmax(E) [mJ]
20	3	4	1	21	53	0.7	8.5
			2	16	41	0.4	5.1
		6	1	23	63	1.1	12.4
			2	17	43	0.2	2.1
40	3	8	2	24	76	0.5	11.1
			3	18	85	0.3	4.2
		12	2	32	114	0.7	12.5
			3	23	80	0.5	6.2
60	4	15	3	23	102	0.4	10
			5	17	78	0.3	4.5
		20	3	23	120	0.4	6.7
			5	18	92	0.3	3.4
100	5	25	3	33	189	0.7	10.4
			5	17	102	0.3	3.2
		35	3	30	207	0.6	11.3
			5	20	134	0.4	4.5
150	5	40	3	22	198	0.4	11.4
			5	15	134	0.3	4.7
		60	4	23	252	0.6	10.5
			6	21	215	0.4	7.5
200	6	70	4	22	265	0.5	11.5
			6	22	265	0.42	8.8
		90	4	24	315	0.56	13.62
			6	22	295	0.53	10.52

7. CONCLUSION

In this paper, we tackled a challenging scheduling problem which is highly appealing in control applications. We considered a tree topology with time-constrained data flows that traverse the network simultaneously in opposite directions. Since the distributed algorithms in the field of the design of *WSNs* scheduling algorithms well-suit the scarce resources of the sensor nodes, we developed a distributed *TDMA* scheduling algorithm that falls into the category of the exact algorithms for single-collision domains *WSNs* and in the category of heuristic algorithms for multiple-collision domains. We proved by the experimental results the energy efficiency of our proposed approach.

For our future work, the spatial reuse of the transmission medium will be considered to overcome the algorithm shortcoming, mainly related to the bandwidth utilization, when the multiple-collision domain tree topology is considered.

REFERENCES

Ahmad, A., Hanzalek, Z., and Hanen, C. (2014). A polynomial scheduling algorithm for IEEE 802.15.4/ ZigBee cluster tree WSN with one collision domain and period crossing constraint. In *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, 1–8.

Bonaventure, O. (2011). *Computer Networking : Principles, Protocols and Practice*. The Saylor Foundation.

Ergen, S.C. and Varaiya, P. (2006). PEDAMACS: power efficient and delay aware medium access protocol for sensor networks. *IEEE Transactions on Mobile Computing*, 5(7), 920–930.

Ergen, S.C. and Varaiya, P. (2010). TDMA scheduling algorithms for wireless sensor networks. *Wirel. Netw.*, 16(4), 985–997.

Hanzálek, Z. and Jurčík, P. (2010). Energy efficient scheduling for cluster-tree Wireless Sensor Networks with time-bounded data flows: Application to IEEE 802.15.4/ZigBee. *IEEE Transaction on Industrial Informatics*, 6(3).

Jurk, P. and Hanzlek, Z. (2010). Simulation study of energy efficient scheduling for IEEE 802.15.4/zigbee cluster-tree wireless sensor networks with time-bounded data flows. In *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*.

Lenzen, C. and Wattenhofer, R. (2011). Distributed algorithms for sensor networks. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 370(1958), 11–26.

Moriyama, K. and Zhang, Y. (2015). An efficient distributed TDMA MAC protocol for large-scale and high-data-rate wireless sensor networks. In *Advanced Information Networking and Applications (AINA), 2015 IEEE 29th International Conference on*, 84–91.

Org., Z.S. (2006). ZigBee specification, std. 053 474r13.

R. Severino, N.P. and Tovar, E. (2014). Dynamic cluster scheduling for cluster-tree wsns. *SpringerPlus Commun. Netw*, 1–17.

Yu, B., Li, J., and Li, Y. (2009). Distributed data aggregation scheduling in wireless sensor networks. In *INFOCOM 2009, IEEE*, 2159–2167.

Zheng, T., Gidlund, M., and kerberg, J. (2014). Medium access protocol design for time-critical applications in wireless sensor networks. In *Factory Communication Systems (WFCS), 2014 10th IEEE Workshop on*, 1–7.