TrickleTree: A Gossiping Approach To Fast And Collision Free Staggered Scheduling

Wojciech Bober, Chris J. Bleakley, Xiaoyun Li
UCD Complex & Adaptive Systems Laboratory
UCD School of Computer Science and Informatics
University College Dublin
Belfield, Dublin 4, Ireland
{wojciech.bober, xiaoyun.li, chris.bleakley}@ucd.ie

Abstract*—In recent years, data gathering has received significant attention as an application of Wireless Sensor Networks (WSNs). Staggered data tree based protocols have been shown to be successful in reducing energy consumption in data gathering scenarios. An important part of staggered protocols is the process of schedule construction. In order to minimize energy consumption, this process must be fast. In this paper, we present TrickleTree, a fast distributed protocol for establishing staggered and collision free communication schedule. TrickleTree has three functions: to establish routes, i.e., construct a data gathering tree, to establish a staggered communication schedule, i.e, assign time slots to links, and to disseminate the maximal tree depth in the network. To minimize network setup time, TrickleTree combines neighborhood discovery and schedule construction into one step. To ensure that good neighbors are discovered before a node joins the network, TrickleTree uses a rating mechanism. Collisions during node association are reduced by using association slots. To increase the message delivery rate with small message overhead, TrickleTree uses adaptive gossiping. We provide a formal analysis of the protocol properties i.e., collision free scheduling and termination. The behavior of the proposed approach is evaluated in simulation. The results show up to 90% in a reduction in schedule setup time and a 50% reduction of duty cycle compared to a flooding approach.

***Index Terms*—Wireless sensor networks, staggered schedule, schedule construction, fast association, collisions reduction, association ranking**

I. INTRODUCTION

This paper is an extended version of [1]. It contains a modified version of the previously proposed algorithm. It contains analysis of the algorithm with proofs of termination and collision free slot assignment. Additional simulations were added for in-depth evaluation of the algorithm.

The promise of cheap sensors deployed at large scale is attractive for areas such us microclimate research [1], and habitat monitoring [2]. Precise observations produce large quantities of data, that must be transmitted via the network. In addition, these networks are often expected to operate for long periods of time. Although various methods of energy harvesting have been proposed [3], so far using a battery is the most common method of powering nodes. Dutta et al. [4] have shown that radio operation is the main cause of power consumption. Therefore, communication protocols which reduce radio on-time are crucial for achieving the goal of long network lifetime.

Data gathering networks are characterized by a many to one traffic pattern. A common approach to routing in this class of networks is tree based routing. In tree base routing, a node selects a node closer to the sink as its parent. All messages are forwarded only to this node. In order to improve energy-efficiency and data latency a staggered approach has been proposed. In this approach, communication between nodes is scheduled according to their level in the tree (i.e., hop count from the root). Only two consecutive levels off the tree are active at any given time. Hence all nodes in the network must be aware of the maximal tree depth in order to schedule their communication correctly. The quality of wireless links can change considerably in a short amount of time [5]. This means a new schedule must be established each time the network topology changes. Therefore it is important that a staggered schedule is established quickly, so that the cost of control does not exceed the cost of data transmission. We address this problem by proposing TrickleTree, a protocol designed to establish staggered schedule quickly, yet with a small message overhead.

TrickleTree differs from existing protocols, in that it combines neighborhood discovery and schedule construction into one step. This reduces the number of messages which must be exchanged. To ensure that a balanced schedule is constructed, TrickleTree carefully selects the time at which a node starts its association process. This is done by delaying the association process accordingly to a ranking function. The function takes into consideration link quality and the number of potential parents. To reduce the likelihood of a node becoming an orphan these factors are weighted accordingly to the number of messages received by the node.

TrickleTree is based on gossiping, which is a simple but robust and reliable technique of information dissemination. We show how this technique can be applied to establish a staggered schedule. Thanks to adaptive mechanisms derived from the gossiping approach, we are able to balance the delay and communication overhead (energy consumption) required to establish the network tree and communication schedule. TrickleTree is able to derive a collision free schedule, therefore energy is not wasted resolving collisions during the data gathering phase. To the authors' knowledge, this is the first protocol for establishing staggered communication schedules.

The contribution of the work is a novel algorithm which has three functions: 1) to establish communication routes, i.e, construct data gathering tree, 2) to establish a collision free staggered communication schedule, i.e., assign time slots to nodes, 3) and to disseminate the maximal tree depth in the network.

The rest of the paper is structured as follows. In section II we discuss related work. Section III, describes the proposed protocol. In Section IV we provide proof of collision free scheduling and algorithm termination. Section V presents simulation results. We conclude the paper with Section VI.

II. Related work

In this section, we discuss two categories of protocols related to TrickleTree. The first category are protocols which can be used to create staggered schedule. The second, is a category of protocols used to disseminate information in Wireless Sensor Networks.

A. Staggered scheduling protocols

Staggered scheduling was first introduced in D-MAC by Lu [7]. In D-MAC, transmission times are staggered in very short time slots. This reduces latency and end-to-end delivery time because the receiver is guaranteed to be awake at the time of the sender's transmission. Because nodes with at same network depth have the same transmission time offset they must to compete for the channel. A CSMA scheme is used to deal with collisions. The authors do not discuss many practical issues related to the protocol. For example, how nodes learn the maximal routing tree depth in order to calculate their wake-up offset.

A similar concept is used in Merlin [8], a cross-layer protocol integrating MAC and routing. In Merlin staggered scheduling is used for up- and down-link traffic.

TIGRA [9] is a protocol designed for periodic collection of raw data from the network. The authors focus on minimization of the time required to collect the data from the entire network. The collection time is reduced by two techniques. Firstly, data from separate packets is merged into one packet. Note, this is different from data aggregation, where a single value is calculated to represent data from a number of sources. Secondly, collisions are eliminated by ensuring interference-free transmission scheduling.

ASLEEP [10] is a data gathering protocol focusing on reducing message latency. ASLEEP, like TIGRA, uses staggered data gathering to reduce latency. The protocol is able to adapt to varying bandwidth requirements by run-time schedule adjustment. ASLEEP adjusts the active radio time at each level of the tree. Schedule modification is made based on previous traffic trends. If bandwidth requirements are increasing over a certain period of time, then the active period is extended, otherwise the active period is decreased.

In [6], staggered scheduling was combined with synchronous low power listening to reduce energy consumption in low rate data collection networks. Figure1 illustrates the concept of a staggered schedule implemented in Bailigh [6]. The network is organized as a tree. Each node has exactly one parent, which forwards data to the tree root (the sink). Nodes at the same distance from the sink (hop count) are at the same level. At any time only two consecutive levels of the tree are active. In the example, nodes A, B and C, at Level 2 transmit data to nodes D and E, at Level 1, using links 1, 2, and 4. This approach reduces delivery latency because messages are almost immediately forwarded. When the slots are guaranteed to be collision and contention free there is no delay due to backoff, as would occur in a CSMA protocol. However, in the case of distributed scheduling, links 1 and 4 sometimes might use the same time slot. If nodes C and D are in radio communication range this may lead to collision. This is an example of the hidden terminal problem in present in networks using staggered scheduling.

B. Dissemination protocols

When all nodes in the network must share common information, message dissemination is necessary. Early systems used packet floods to disseminate common information such as parameters or commands. Flooding protocols rebroadcast packets they receive [11]. This is a very simple method, which has many disadvantages. Firstly, flooding is unreliable. Due to collisions, some nodes do not receive the information, so typically flooding is repeated. This leads to excessive rebroadcasting and is energy inefficient. To overcome this problem, adaptive protocols have been introduced. They modify node behavior depending on the information they receive. Dissemination protocols which have proven to work reliably in Wireless Sensor Networks are based on gossiping. Trickle [12] was the first proof of concept implementation designed for code dissemination. Trickle is an adaptive gossiping protocol. When nodes detect inconsistent information in the network, they broadcast new information quickly. When nodes agree, they slow down their communication rate exponentially, such that, when in a stable state, they transmit infrequently. Based on this concept, dissemination protocols have been proposed for various applications [13], [14]. TrickleTree is a modification of Trickle used to construct a staggered scheduling tree. We use adaptive beaconing for neighborhood discovery and tree depth dissemination.

TrickleTree is able to minimize collisions between nodes in the tree by scheduling individual transmission slots. We use 2-hop neighborhood information about scheduled time slots to reduce collisions. This technique is often used in TDMA MAC protocols [15]. Fang-Jing Wu [16] shows how collision-free scheduling can be taken advantage of tree based networks.

III. TrickleTree protocol

TrickleTree uses three types of packets Beacon (BCN), Join Request (JREQ) and Join Reply (JREP). Each packet contains a source and destination address. A BCN packet contains the sender's distance from the sink (hop count), parent address, slot number, and the maximal tree depth known to the node, and number of free slots. A JREQ packet contains the same fields. JREP contains the slot number assigned by the parent

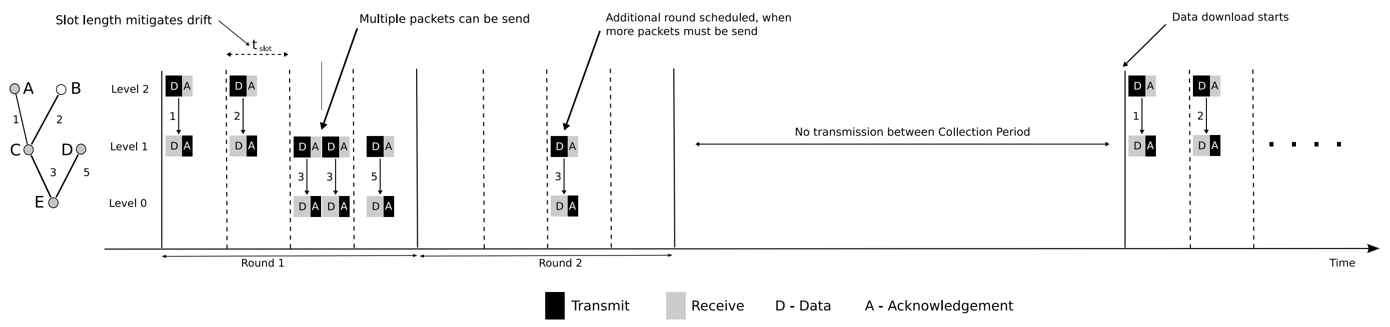

Slot length mitigates drift
t_{slot}
Multiple packets can be send
Additional round scheduled, when more packets must be send
Data download starts
A
B
C
D
E
Level 2
Level 1
Level 0
No transmission between Collection Period
Round 1
Round 2
Time
Transmit
Receive
D - Data
A - Acknowledgement

Figure 1: Staggered communication scheme used in [6].

to the child. Note that the purpose of TrickeTree is to quickly create a short living data gathering tree. We assume, that during the tree lifetime (tens of seconds), the network topology remains stable. Therefore, TrickleTree lacks functions typical for routing protocols, i.e., route maintenance. If required, this can be achieved by running TrickleTree periodically.

A node can be in one of six states: *Suspended*, *Listening*, *Joining*, *Collision*, *Gossiping or Connected*. Initially a node starts in a *Listening* state and waits for BCN packets. Upon receiving a BCN packet the node compute value of the ranking function R. After that it delays the start of its association process accordingly to the function value. In general, the lower the value of the function the delay is longer. Each time a BCN packet is received, the ranking function R is computed. If a BCN from a better candidate is received the association process delay is set accordingly to the new value. This means that a node waits for a better candidates for parent before starting association. This is necessary because the neighborhood is discovered during schedule construction.

TrickleTree uses a Shortest Path with threshold ($SP(x)$) metric to select the best parent. This metric selects a node with the smallest distance from the sink among neighbors with link quality exceeding x. The $SP(x)$ metric is used for simplicity. More complicated metrics like MintRoute [17] can be used.

A. Beacon Dissemination

Beacon dissemination based on gossiping is key to the proposed approach, serving multiple purposes. In principle, Trickle adjusts the frequency of information dissemination depending on consistency. When information in the network is consistent (e.g., a version of binary code) beacons are broadcasted infrequently. In contrast, when a node detects inconsistent information, the frequency of beaconing is increased. Consistency in the network is determined by overhearing beacons from other nodes. In Trickle dissemination, information is divided into metadata and the data itself. This allows separate transmission of large data (e.g., binary code) from version information. In TrickleTree only small beacons are broadcasted thus there is no separation between metadata and data.

Table I: SUMMARY OF SYMBOL DEFINITIONS

Symbol	Description	Symbol	Description
t	beacon timer	t_d	discovery timer
τ_l	low gossiping period	t_g	gossiping timer
τ_h	high gossiping period	s_a	# of available slots
j_s	join slot	t_{pkt}	packet duration
j_{max}	max # of join slots	t_{ack}	ack duration
l	node level	j_{dly}	join delay
d	tree depth		

Table II: TrickleTree dissemination algorithm pseudocode.

Event	Action
τ expires	If c $>$ 0 double τ, up to τ_h. Set $c = 0$, pick a new t.[1]
t expires	If $c < k$ or $c = 0$, transmit.
Receive BCN and $BCN(d) = d$	Increment c.
Join network; change level; Receive BCN and $BCN(d) \neq d$	Set τ to τ_l, Set $c = 0$, pick a new t.[1]

[1] t is a random value from the range $[\frac{\tau}{2}, \tau)$

The symbol definitions used in TrickleTree are described in Table I. TrickleTree uses a modified version of the Trickle algorithm. In the original algorithm, the gossiping period τ is doubled whenever the previous gossiping period expires. In TrickleTree τ is doubled only if in the previous gossiping period a beacon with consistent tree depth (d) was received ($c > 0$). This is a simple method of detecting collision due to a hidden terminal: if the node broadcasts a beacon, it should receive at least one from one of its neighbors. For the same reason, a beacon is transmitted whenever the beacon period t expires and no beacon with the same tree depth is received ($c = 0$). This is different from the original Trickle algorithm. We have also extended the list of events on which the gossiping period is set to its lowest value. Table II presents pseudocode for the TrickleTree algorithm.

Upon receiving a BCN packet, a node performs a set of actions. Information from the packet is used to construct a neighbor table. Each record of the table consists of a node address, distance from the sink, assigned slot number, received signal strength, and time synchronization data. Every time a node receives a BCN packet from a node which is already in the table, the information is updated.

Because staggered data gathering requires maximal tree depth for timing offset calculation, all nodes in the network must agree on a common value. Each node connected to the network disseminates the maximal tree depth it currently holds in the BCN packet. If a node overhears a BCN packet which has a maximal tree depth field value greater than its current tree depth value, the node updates its current tree depth to the received value. It then resets the counter c, sets τ to τ_l and picks up a new t value. This allows for fast dissemination of a new tree depth in the network. TrickleTree does not have explicit mechanisms to check the consistency of the tree depth value. We rely on the general convergence property of dissemination protocols based on gossiping.

Note that a node is allowed to participate in gossiping only when it is in the *Gossiping* or *Connected* state, i.e., after joining the network. Initially only the sink node is able to disseminate beacons.

B. Node association

Most schedule based data gathering protocols build schedules based on data gathering tree. To build such a schedule, a parent - child relationship must be established between nodes. To establish this relationship a node associates with a node closer to the sink. This node becomes the node's parent. In the simplest case, a node starts the association procedure immediately or shortly after receiving a beacon from a potential parent. This method, used in [9], [18], is shown in Figure 2b. Although simple, there are two main drawbacks of this method. Firstly, it relies on the MAC to resolve collisions. These collisions are caused by multiple nodes requesting association to the same node. Secondly, it does not take into consideration the quality of the link to a selected node. To address this issue, a threshold T_h on the link quality is often introduced. In this case, the association process is started only when the link quality is above minimal threshold. To address the former issue a method based on association slots was proposed in [19]. In this method, nodes use time slots to perform the association process (Figure2b). Whenever a node wants to join the network, it selects a random slot and starts the association process. Because the length of the slot is sufficient for the whole association process, i.e., exchange of request and reply packets, collisions at the MAC layer are reduced. As in the previous methods, a threshold on link quality is used to ensure that only good links are used. The method proposed herein improves on the method proposed in [19]. Instead of using a random slot, a node selects a slot according to a ranking function (1). The value of the function is computed each time a BCN packet from a neighboring node is received. The function takes into consideration the signal quality σ, the degree of a node δ (defined as a number of potential parents), and the total number of beacons received β. The aim of the function is to assign a higher rank i.e., earlier join slot, to nodes which have a better link to the node broadcasting a beacon. It also takes into consideration how many potential parents a node has, e.g., if a node has only one potential parent in its neighborhood its priority will be higher than a node with more potential parents, even if the link quality is worse. The function uses the number of beacons received from neighboring nodes as a weight for both factors. As the number of beacons increases, degree of a node δ gains on significance. This is to reduce the number of orphan nodes and forced associations.

$$q_\sigma = \frac{(\sigma - \sigma_L)}{\sigma_H - \sigma_L},\ q_\delta = \frac{(\delta - \delta_h)}{\delta_L - \delta_H}\ q_\beta = \frac{(\beta - \beta_L)}{\beta_H - \beta_L}$$

$$R(\sigma, \delta, \beta) = (1 - q_\beta)q_\sigma + q_\beta q_\delta \tag{1}$$

Based on the value of the ranking function, a node calculates association delay (2) and (3).

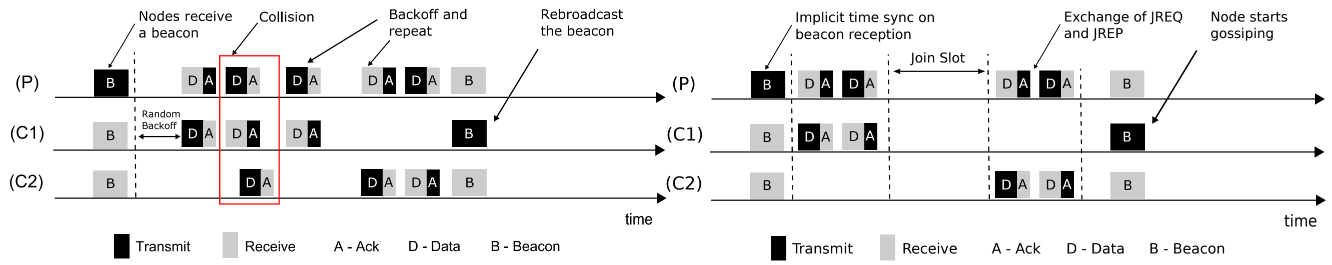
$$j_s = \lfloor j_{max}(1 - R) \rfloor \tag{2}$$

$$j_d = 2(t_{pkt} + t_{ack})j_s \tag{3}$$

Each BCN received by a node is an implicit synchronization point. Each join slot is long enough to allow for JREQ and JREP exchange as well to mitigate for clock drift. If the JREQ packet is delivered successfully, the node will set up a JREP timeout. Setting a timeout on the JREP prevents the node from infinite waiting in the case that the selected node does not reply. This might happen when the potential parent fails before it manages to send a JREP. In the case of delivery failure, JREP timeout, or join rejection, the node will return to the *Listening* state. It might happen, however, that two nodes will request association in the same slot. This is more probable when node density is high and the value of j_{max} is low. If both nodes are in the radio range, a node will detect an on going transmission. In this case the node will repeat the association procedure after receiving the next beacon. If both nodes are not in the radio range, most likely their JREQ packet will collide. This is reported by the MAC layer. A node switches to Random mode and repeats the association procedure after receiving the next beacon.

A node accepts JREQ packets only when it is in the *Gossiping* state. A parent node which receives a JREQ packet from a node selects an slot using Algorithm 1. The slot is marked as used by the node which requests the join and a JREP packet is sent back. In the case that there are no free slots left, the node will refuse to accept the join request and send a JREP packet with the REFUSED flag set.

Upon receiving the JREP packet, the node cancels the JREP timeout. If Collision Free (CF) mode is enabled, the node compares the assigned slot number with the slot numbers assigned to nodes stored in its neighbor table. If a node with the same slot number at the same level exists, it indicates a slot collision. In this case, the node enters the *Collision* state and initiates a collision resolution procedure (see Section III-E). If a node with the same slot number does not exist or CF mode is disabled, the node stores the slot number and enters the *Gossiping* state. Once connected, the node enables BCN packet dissemination as described in Section III-A. It also sets a gossiping timer t_g. This timer determines how long a node maintains in the *Gossiping* state. After the timer expires the node enters the *Connected* state.



(a) Association procedure based on MAC layer.

(b) TrickleTree association procedure.

Figure 2: Comparison of association procedures. (P) denotes a potential parent node, whereas (C1) and (C2) denote child nodes.

C. Forcing association

Occasionally a node has only one potential parent in its neighborhood. In this case, the node can force association with the selected node. The node sends a JREQ with the FORCE flag set. Upon receiving this packet, the parent will select the child with the highest degree d and remove it by sending JREP with the REFUSED flag set. The slot released in this way will be assigned to the node forcing join by sending a JREP packet. The node which was removed will attempt to join a different parent, as described in the previous section. Forced association ensures that the whole network is connected as long as good quality links are available.

D. Detecting collisions

As mentioned previously, TrickleTree can work in Collision Free (CF) mode. If CF mode is enabled, TrickleTree ensures that all nodes have individual transmission slots. To achieve this, once in *Gossiping* state, a node constantly monitors received BCN packets in order to detect potential collisions. This is done by comparing its level and slot number with the received values. Equal values indicate slot collision. In this case, the node will change its state to *Collision* and request a new slot from its parent as described in Section III-E. If a node is a parent and the level of the node in the received BCN packet is equal to the level of its children then the node will check if there is a child node assigned to the same slot number. If so, the node will use a collision resolution procedure to assign a new slot to its child. A node will always invalidate a given slot, so that it cannot be used to schedule children. It is possible that a BCN packet indicating collision is received by the parent and child at the same time. In order to prevent both nodes from requesting a new slot at the same time, join requests sent by children are delayed. This allows the parent to solve the collision first, which is preferred since it requires transmission of only one packet. The method used in TrickleTree to detect collisions uses two hop neighborhood information to assign individual slots. This method is often used by TDMA MAC protocols [15] to handle collisions from hidden terminals. In TrickleTree collisions from hidden terminals are detected and resolved by either the parent or child, as described in detail in the next section.

Algorithm 1 Slot selection algorithm

```
if $s_l > 0$ then
    $s_r \leftarrow random(s_c)$
    if $s_r < s_c - 1$ then
        $s_i \leftarrow s_r + 1$
    else
        $s_i \leftarrow 0$
    end if
    while $s_i <> s_r$ AND ($s_i$ is Free OR $s_i$ is Valid) do
        if $s_i < s_c - 1$ then
            $s_i \leftarrow s_i + 1$
        else
            $s_i \leftarrow 0$
        end if
    end while
    $s_l \leftarrow s_l - 1$
    return $s_i$
else
    return $NONE$
end if
```

E. Resolving collisions

The method of resolving a detected collision depends on the node's role. A node can solve a collision as a parent, child, or intermediate node.

1) Parent collision resolving procedure:

- A beacon from an unrelated child node is received: the collision is solved by assigning a new slot to the child node. The parent invalidates the collided slot and selects a different slot. Next, the parent sends a JREP packet to its child node with the new slot. If slot cannot be assigned, the child node is disconnected from the parent and a JREP packet with REFUSED flag is send back. This forces the child node to connect to a other parent. If a node which is a parent for different nodes cannot connect to the network it will send a JREP to all its children with the REFUSED flag set.
- A beacon from an unrelated parent node is received: in this case, a node compares s_a with $BCN(s_a)$. If $s_a > BCN(s_a)$ then the node changes its own child slot, as

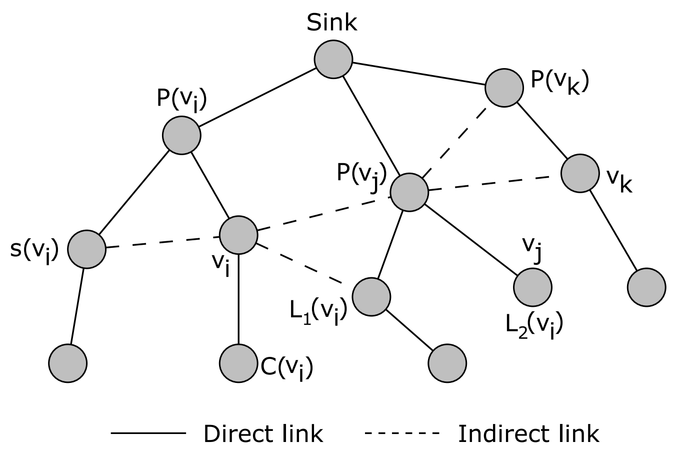


Figure 3: Classification of nodes with respect to v_i

described previously. If $s_a < BCN(s_a)$ then the node requests the other node to change slot.

2) Child collision resolving procedure: As a child node, the collision is resolved by requesting a new slot from its parent. This is done by sending a JREQ packet to the current parent.

IV. Analysis of the TrickleTree algorithm

In this section we provide formal analysis of TrickleTree algorithm, which is the core of the TrickleTree protocol presented in Section III. Formally we consider a network modeled as an unidirected graph $G = (V, E)$, where V contains all nodes and E contains all communication links. The goal is to construct a tree T and a communication schedule S from G rooted at the sink. The communication schedule uses a time-division model, where time is divided into fixed length slots. The slots are grouped in a frame, which has a fixed length. The communication schedule is constructed by assigning a slot s_i to each node $v_i \in V$.

Definition 1. *Given a node v_i and a data collection tree T in G, we define $level(v_i)$ as the distance in hops of v_i from the sink, $P(v_i)$ as $v_i's$ parent, $N_n(v_i)$ as the set of v_i's n-hop neighbors, $L_n(v_i)$ as the set of $v_i's$ n-hop neighbors where $\forall\, v_j \in L_n(v_i) : level(v_j) = level(v_i)$. We define v_i's interference set as $I(v_i) = L_1(v_i) \cup L_2(v_i)$.*

Note, that TrickleTree uses a dynamic interference set, i.e. as nodes join the data gathering tree, new nodes whose slot assignment might collide will appear in the interference set.

Theorem 1. TrickleTree ensures collision free collection schedule.

Proof: By definition for each pair of v_i and v_j where $level(v_i) \neq level(v_j)$ the schedule is interference free even if $s_i = s_j$ as the transmission is separated temporally. Therefore collision is possible only if a node $v_j \in I(v_i)$. We prove Theorem 1, by showing that TrickleTree provides a collision free assignment for all cases where $v_j \in I(v_i)$.

1) A node $v_j \in L_1(v_i) \wedge P(v_j) = P(v_i)$, in this case a node v_j is $v_i's$ sibling and collision is not possible as the parent assigns different slots to its children.
2) A node $v_j \in L_1(v_i) \wedge P(v_j) \notin N_1(v_i)$, in this case assignment $s_i = s_j$ is detected by overhearing BCN packets by either node v_j or v_i. The collision is then resolved using the child collision resolution procedure.
3) A node $v_j \in L_1(v_i) \wedge P(v_j) \in N_1(v_i)$, in this case assignment $s_i = s_j$ might be detected by overhearing by node v_j and $P(v_j)$ simultaneously. The procedure parent collision resolution procedure takes precedence over the child collision resolution procedure. Thus, the collision is resolved.
4) A node $v_j \in L_2(v_i) \wedge P(v_j) \in N_1(v_i)$, in this case assignment $s_i = s_j$ is detected by overhearing by node $P(v_j)$. The collision is resolved using the parent collision resolution procedure.
5) A node $v_j \in L_2(v_i) \wedge P(v_j) \notin N_1(v_i)$, in this case assignment $s_i = s_j$ is detected by intermediate node m_{ij}. The node m_{ij} sends a unicast message either to node v_i or v_j accordingly to the intermediate node collision resolution procedure.

■

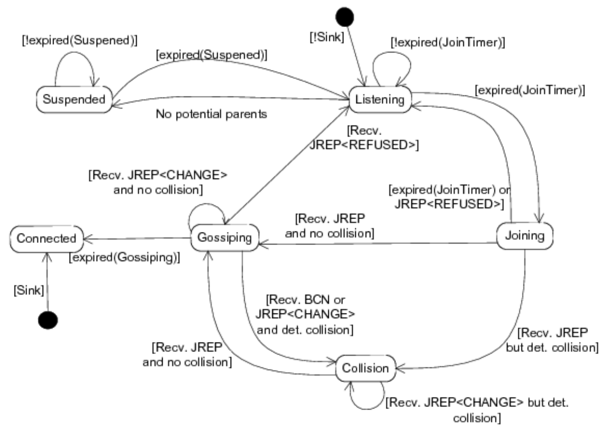

[!expired(Suspened)]
[expired(Suspened)]
[!Sink]
[!expired(JoinTimer)]
Suspended
Listening
[expired(JoinTimer)]
No potential parents
[Recv. JREP<REFUSED>]
[Recv. JREP<CHANGE> and no collision]
[expired(JoinTimer) or JREP<REFUSED>]
[Recv. JREP and no collision]
Connected
Gossiping
Joining
[expired(Gossiping)]
[Sink]
[Recv. BCN or JREP<CHANGE> and det. collision]
[Recv. JREP and no collision]
[Recv. JREP but det. collision]
Collision
[Recv. JREP<CHANGE> but det. collision]

Figure 4: TrickleTree finite state machine diagram. PACKET<FLAG> notation is used to denote a flag which must be set in a packet for a transition to occur. Expired() notation is used for timers.

Definition 2. *We define $C(v_i)$ as the set of potential parents where $\forall\, v_j \in C(v_i) : v_j \in N_1(v_i) \wedge q(v_j) > q_0$ and $q(v_i)$ is a function which allows for assessing quality of the link to a node v_i. We define $A \rightarrow B$ as transition from state A to state B, and $A \xrightarrow{t} B$ as timeout transition from state A to state B i.e., transition which happens after time t unless other condition occurs earlier.*

Theorem 2. TrickleTree terminates.

Proof: The finite state machine of TrickleTree is shown in Figure 4. We prove Theorem 2 by showing that, for each node v_i participating in schedule, a transition to the $Connected$ or $Suspend$ states occurs.

1) All nodes, apart from the sink, start in the $Listen$ state.

Table III: SIMULATION PARAMETERS.

Parameter	TelosB	Unit
P^*_{tx}	58.5	mW
P^*_{rx}	65.4	mW
P^*_{sleep}	0.015	mW
P^*_{poll}	14.1	mW
t_{poll}	2.5	ms
t_{cca}	2	ms
Data Rate	250	kbps
Voltage	3	V

Parameter	TT	Tigra
Retransmission #	3	10
Slot Count	10	
Buffer Size	20 packets	
Clock drift	100 ppm	
Initial Backoff	-	16
Packet Size	48 bytes	

Each node v_i sets a discover timer t_d. During time t_d a node v_i constructs set $C(v_i)$.

2) If $|C(v_i)| = 0$ after t_d, transition $Listen \overset{t_d}{\rightarrow} Suspend$ occurs. Hence, the algorithm terminates because there are no potential parents.
3) If $|C(v_i)| \neq 0$ after t_d, transition $Listen \overset{t_d}{\rightarrow} Joining$ occurs. A node v_i selects the first node $v_j \in C(v_i)$ as $P(v_i)$. The join procedure is started.
4) A node may go through a number of transitions $Joining \rightarrow Collision$, $Collision \rightarrow Gossiping$, and $Gossiping \rightarrow Collision$.
5) Each time a node v_i enters the $Collision$ state, v_i is assigned a new slot s_i by $P(v_i)$. If $P(v_i)$ cannot assign a new slot to v_i the node v_j is moved from $C(v_i)$ to $\overline{C}(v_i)$. The next node $v_j \in C(v_i)$ is selected as $P(v_i)$. The join procedure is repeated.
6) Since $C(v_i)$ is finite, a node v_i can enter the $Collision$ state a limited number of times. In this case, a node v_i selects a node v_j from $\overline{C}(v_i)$ and forces join. Due to this, the transition $Collision \rightarrow Connected$ is made and the algorithm terminates.
7) Each time a node v_i enters the $Gossiping$ state a timer t_g is set. Transition $Gossiping \overset{t_g}{\rightarrow} Connected$ occurs after t_g. The algorithm terminates.

■

V. EVALUATION

A. Simulation Model

In order to verify the behavior of the proposed algorithm, a set of simulations were performed using the OMNeT++ [20] simulator. The simulations were performed using 10-50 randomly deployed nodes. In the two first scenarios the simulation area was 35×35 m^2, whereas in the last the area was 65×65 m^2. In each case, the sink was placed in the center of the simulated area.

The simulation uses the Log-Normal Shadowing Model [21] for wireless signal propagation. The model takes into consideration the effects of wireless signal fading and shadowing. These effects, common in wireless transmissions, are modeled by adding a perturbation factor to the reception power. This factor follows a normal distribution, with a standard deviation σ which can be defined for each simulation run. In addition, the asymmetry of links is modeled. To capture the effects of wireless interference, the simulation uses an physical (additive) interference model [22]. In this model, reception probability is determined by signal-to-noise ratio. The sum of power from multiple concurrent transmissions may cause interference at a given node, even though separately each is below the receiver sensitivity.

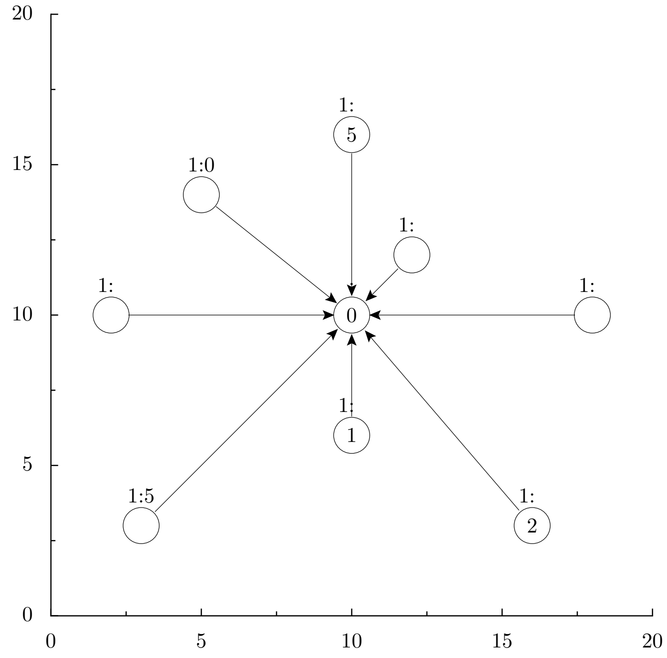


Figure 5: Example of node placement in association experiment.

A model of the Chipcon CC2420 [23] transceiver, which conforms to the IEEE 802.15.4 specification, was used in the simulation. The transceiver is modeled as a finite state machine consisting of tree states: sleep, receive, and transmit. Delays in transition between respective states were modeled, which allows for precise calculation of the duty cycle and energy consumption. In addition, the radio model includes properties such as, modulation type (PSK), sensitivity, and bit error rate. These properties influence the signal propagation of the model.

Each simulation was repeated 100 times and a 0.05 confidence level was used to calculate respective confidence intervals. For each simulation, nodes were uniformly distributed in the simulation field. Motes boot up with start-up times randomized according to a uniform distribution. In all simulations we measured the time required to establish a network schedule. We considered the network topology to be established when all nodes were connected, have the same maximal tree depth value, and no schedule collisions exist. TrickleTree can use any MAC protocol. Herein, the protocol is evaluated with B-MAC [24].

B. Results

1) Association time: The first set of experiments was performed to assess the impact of the association scheme on the number of collisions and on association time. In the experiment, the sink was placed in the center of the field. A number of nodes within radio range were placed around the

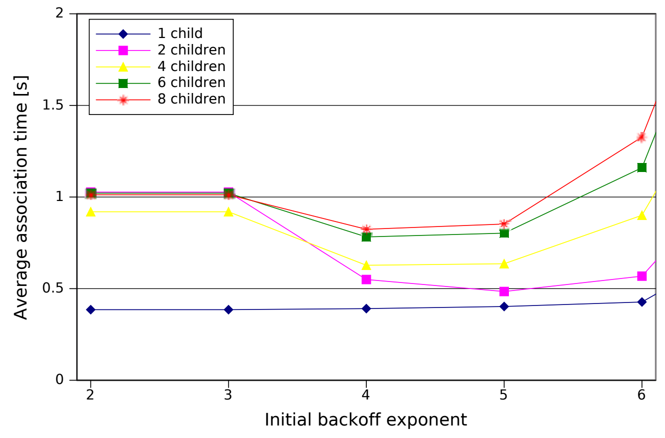


Figure 6: Avg. association time for *MAC-Exponential* scheme.

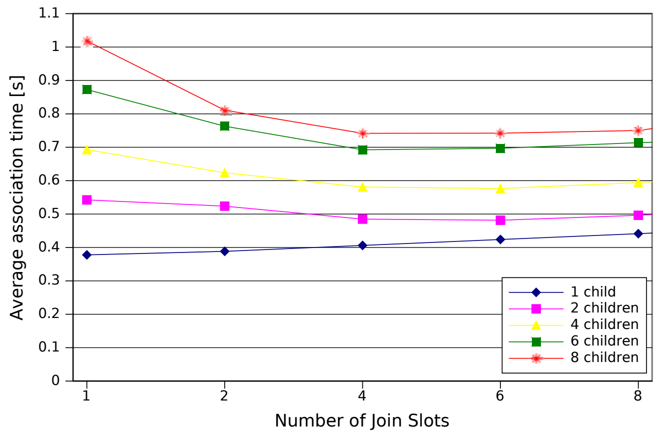


Figure 8: Avg. association time for *TrickleTree-Rank* scheme.

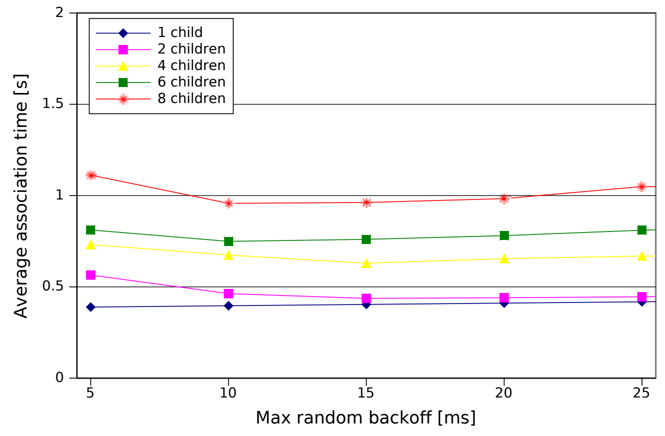


Figure 7: Avg. association time for *MAC-Random* scheme.

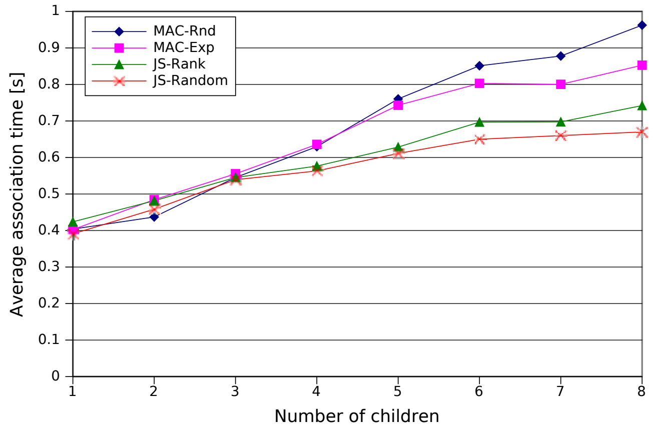


Figure 9: Comparison of avg. association time.

sink. The distance between each child and the sink was varied, so that a different RSSI value was achieved for each child. An example scenario is shown in Figure 5. Note that, association time is different from schedule construction time. The later includes the time required to resolve schedule collisions and disseminate the tree value, whereas the former denotes how fast a node associates with its parent. We compare four different association schemes. Two schemes are based on Medium Access Control: random backoff offset *(MAC-Rnd)* and exponential backoff offset *(MAC-Exp)*. We compare these MAC based schemes with two schemes based on join slots: slot calculation based on rank *(JS-Rank)* and randomly chosen slot *(JS-Random)*. In all cases, the protocols use TrickleTree's beacon dissemination method and the low gossiping period was $\tau_l = 0.5$s. The initial beacon was broadcasted at time randomly chosen between 0.25s-0.5s. For this reason, the average association time is greater than 0.35s. We disabled LPL in order to eliminate additional delay. Figure 6 shows the average association time for *the MAC-Exp* scheme. It can be seen that association based on exponential backoff has an optimum near backoff exponent 4-5. This corresponds to a backoff limit of 16-32ms. A similar observation, although the effect is less pronounced, can be made with *MAC-Rnd* (Figure 7). The optimal value for the *MAC-Rnd* scheme is close to 15ms. The results can be interpreted as follows: too small backoff value can cause high contention and a large number of collisions. This increases association time as nodes have to backoff multiple times (in the case of contention) or repeat the association process (in the case of collision). When a large backoff duration is used, delay is not increased due to contention or collision, but due to the duration of the initial backoff.

Figure 8 shows an average association time for the *JS-Rank* scheme. For this scheme, we varied the maximal number of join slots j_{max}. The optimal value of j_{max} should be close to the average neighborhood size of a node. This ensures that association requests are spread equally in time. When j_{max} is too low, a higher association time is needed due to the increased number of collisions. With high j_{max}, the ranking function has more impact on association delay. This can be seen clearly in the case of a single child. Since no collisions or contentions are involved, the association time depends solely on the outcome of the ranking function. Hence it increases with j_{max}. In Figure 9, we compare the association time of all

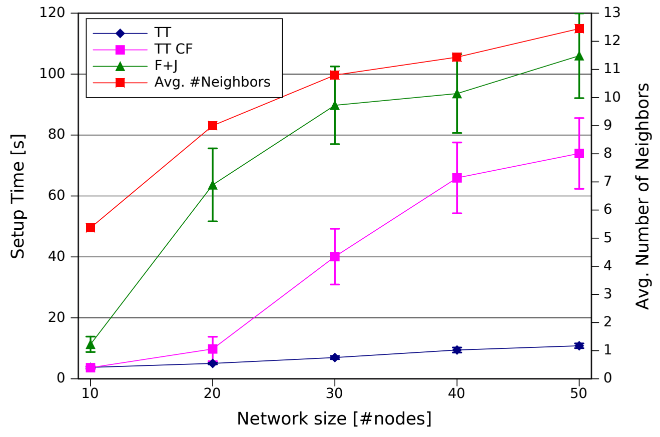


Figure 10: Tree setup time as function of network size. TT denotes TrickleTree algorithm, whereas F-J denotes the flooding join algorithm.

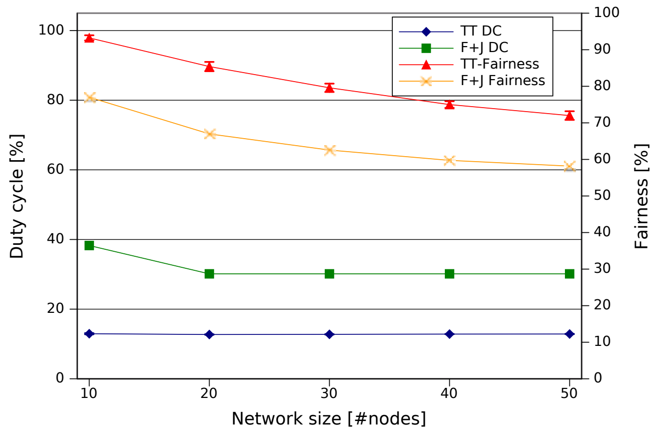


Figure 12: Duty cycle as function of network size. TT denotes the TrickleTree algorithm, whereas F-J denotes the flooding join algorithm.

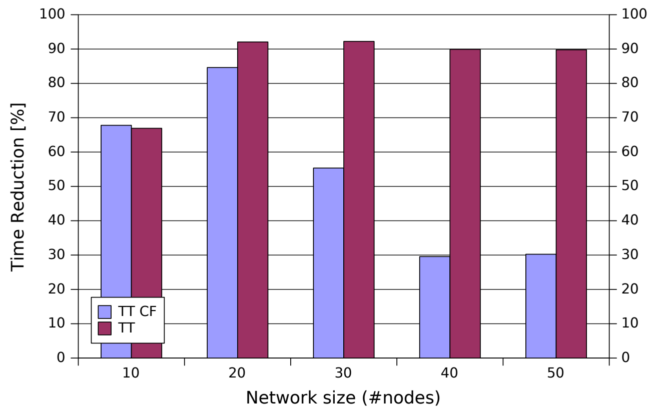


Figure 11: Network setup time reduction in relation to flooding join.

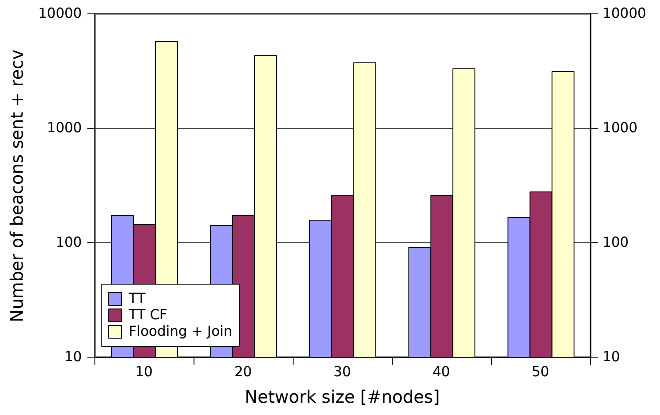


Figure 13: Sum of beacons sent and received for various network sizes.

schemes. For each scheme, we selected the lowest association time for a given number of children i.e., the one with optimal backoff duration or number of slots. It can be seen that for a small number of children (less than 3), all schemes perform similarly. As the number of contending nodes increases, the time required to complete association differs. For number of children greater than 3, the *JS-Rank* scheme performs better than both *MAC-Rnd* and *MAC-Exp*. On average, JS-Rank is faster than *MAC-Rnd* and *MAC-Exp* by 17% and 12%, respectively. The *JS-Random* scheme achieves slightly faster association time. This is because there is no delay introduced by the ranking function. On average, the JS-Random scheme is faster than *MAC-Rnd* and *MAC-Exp* by 21% and 17%, respectively.

2) Schedule construction time: The main goal of TrickleTree is to quickly establish a staggered data gathering tree with minimal energy overhead. TrickleTree uses join slots to reduce collisions and improve setup time. Adaptive gossiping is used to reduce message overhead and ensure that changes in tree depth are disseminated quickly. To verify TrickleTree performance it was compared with a flooding approach, similar to that used in [25], and Tigra [9].

In the flooding approach, the sink periodically broadcasts a tree setup beacon. Upon receiving the beacon, unconnected nodes attempt to join a selected parent. Nodes which are connected to the tree rebroadcast received beacons. The flooding approach relies on the MAC protocol to resolve collisions. If not all nodes can join the network in given simulation run, the results were rejected and not included in calculations.

Tigra is a state-of-the art data gathering protocol. It is one of a few protocols which discuss the process of staggered schedule construction. We implemented a slightly modified version of the algorithm described in [9]. The first modification was necessary because the original algorithm is designed to assign a round to a node based on the number of its descendants. In TrickleTree the round is the same as the level of a node in the tree. The modification does not change the number of exchanged messages nor the exchange procedure. We simply changed the contents of the packets. Instead of number of descendants the *RESPONSE* packet contains level

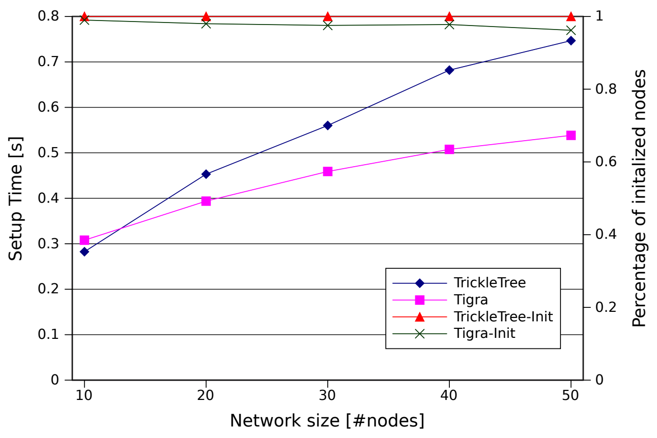


Figure 14: Comparision of Initialization time of Tigra and TrickleTree

of a leaf node. When all response packets get to the sink, the sink is able to determine the maximal tree depth. After that the *INIT2* packet is sent down the tree, to inform all nodes about the maximal tree depth.

Figure 10 shows the setup time and average neighborhood size. It can be seen that as the average number of neighbors grows, the performance of the flooding approach (F+J) deteriorates considerably. This is due to the high number of collisions which must be resolved by the MAC layer. Moreover, nodes ignore information contained in the received beacons and rebroadcast them, even though the network state is consistent. This causes additional collisions due to the hidden terminal problem. TrickleTree (TT) on the other hand, performs more consistently. Collisions are reduced, due to the fact that nodes calculate distinctive join slots, therefore the number of collisions which must be resolved at the MAC layer is reduced. When collision free scheduling is enabled (TT CF), Trickle Tree requires more time to establish the schedule. This is clearly seen in Figure 13, which shows the average number of beacons sent and received. It can be seen that the collision free version of TrickleTree requires up to 65% more beacons to be sent. This is because each change in a schedule (e.g., slot change) resets the gossiping period to the lowest value, therefore beacons are sent more frequently. Figure 13 shows the advantage of adaptive gossiping as used in TrickleTree, over the flooding approach. Adaptive gossiping reduces the average number of beacons from 4000 in the flooding approach to 145 and 223 for regular and collision free TrickleTree, respectively. Figure 11 shows the time reduction achieved using the regular and collision free versions of TrickleTree with respect to the flooding approach. The advantage of TrickleTree is increases with the network size. In the flooding approach large numbers of nodes produce more of collisions slowing down schedule setup. As indicated previously, the collision free version of TrickleTree requires more time to setup the network. There is a tradeoff between fast schedule setup time and possible collisions during data gathering and slower schedule setup and collision free transmissions. The decision as to which version of TrickleTree should be used depends on the application requirement and network stability. For unstable networks, low setup time is important as the network might be rescheduled frequently. Therefore control overhead should be reduced. In stable networks, higher control overhead for setting up collision free schedule will be balanced in the long run by a collision free data gathering phase.

Figure 14 shows a comparision of initialization time of TrickleTree and Tigra. As it can be seen Tigra, on average, is 15% faster than TrickleTree. This is because TrickleTree introduces a delay in the association in order to find good links. In Tigra nodes connect to a node from which the first beacon is received. However, Tigra was not able to complete schedule construction in most of the cases i.e., connect 100% of the nodes and agree on a common depth of the tree. This is because the Tigra assumes lossless links and reliable packet delivery by the MAC layer. In this regards results obtained in herein confirm results obtained by the authors in a testbed [26]. TrickleTree completed schedule construction in all simulated cases.

3) Duty cycle: In Figure 12, the duty cycle of both protocols is shown. The average duty cycle of TrickleTree over different network sizes is 12%, whereas the duty cycle of flooding is 32%. The duty cycle is reduced by adaptive gossiping. Although, the duty cycle of the flooding approach can be reduced by increasing the sink broadcast period it results in a longer network setup time. Fairness of both protocols is also shown in Figure 12 confirms that the main reason for TrickleTree's performance is reduction in collisions. In general, the improvement in network setup time depends on network size. TrickleTree reduces setup time by 68% for networks of 10 nodes, to nearly 90% for networks of 50 nodes.

VI. Conclusions

Data gathering is one of the most recognized applications of wireless sensor networks. As available network energy is a very limited resource and radio communication exploits this resource the most, developing efficient communication protocols is an important task. In this work, we proposed a practical approach to scheduling staggered networks based on gossiping. To validate the behavior of the proposed approach we performed a number of simulations and the results show up to 90% reduction of schedule setup time and 50% reduction duty cycle compared to the conventional flooding approach.

Acknowledgments

This work is supported by Enterprise Ireland grant number CFTD/07/IT/303, "Network-level power management for wireless sensor networks".

References

[1] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong, "A macroscope in the redwoods," in *Proc. of ACM SenSys*. ACM, 2005, pp. 51–63.

[2] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler, "An analysis of a large scale habitat monitoring application," in *Proc. of ACM SenSys*. ACM, 2004, pp. 214–226.

[3] S. Roundy, P. K. Wright, and J. M. Rabaey, *Energy Scavenging for Wireless Sensor Networks: With Special Focus on Vibrations*, Norwell, MA, USA, 2004.

[4] P. Dutta, D. Culler, and S. Shenker, “Procrastination might lead to a longer and more useful life,” in *Proceedings of HotNets-VI, Atlanta, GA, November*, 2007, pp. 1–7.

[5] K. Srinivasan, M. A. Kazandjieva, S. Agarwal, and P. Levis, “The B-factor: measuring wireless link burstiness,” in *Proc. of ACM SenSys*. New York, NY, USA: ACM, 2008, pp. 29–42.

[6] W. Bober and C. Bleakley, “Bailigh: Low power cross-layer data gathering protocol for Wireless Sensor Networks,” in *Proc. of ICUMT*, Oct. 2009, pp. 1–7.

[7] G. Lu, B. Krishnamachari, and C. S. Raghavendra, “An adaptive energy-efficient and low-latency mac for tree-based data gathering in sensor networks: Research articles,” *Wirel. Commun. Mob. Comput.*, vol. 7, no. 7, pp. 863–875, 2007.

[8] A. G. Ruzzelli, G. M. P. O’Hare, and R. Jurdak, “Merlin: Cross-layer integration of mac and routing for low duty-cycle sensor networks,” *Ad Hoc Networks*, vol. 6, no. 8, pp. 1238–1257, 2008.

[9] L. Paradis and Q. Han, “A data collection protocol for real-time sensor applications,” *Pervasive and Mobile Computing*, vol. 5, no. 4, pp. 369 – 384, 2009.

[10] G. Anastasi, M. Conti, and M. Di Francesco, “Extending the lifetime of wireless sensor networks through adaptive sleep,” *Industrial Informatics, IEEE Transactions on*, vol. 5, no. 3, pp. 351 –365, Aug. 2009.

[11] J. Lu and K. Whitehouse, “Flash flooding: Exploiting the capture effect for rapid flooding in wireless sensor networks,” in *Proc. of INFOCOM*, 2009, pp. 2491 – 2499.

[12] P. Levis, N. Patel, D. Culler, and S. Shenker, “Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networkstr,” in *Proc. of USENIX NSDI*, Berkeley, CA, USA, 2004, pp. 2–2.

[13] G. Tolle and D. Culler, “Design of an application-cooperative management system for wireless sensor networks,” in *Proc. of EWSN*, 2005, pp. 121–132.

[14] K. Lin and P. Levis, “Data Discovery and Dissemination with DIP,” in *Proc. of IEEE IPSN*, 2008, pp. 433–444.

[15] M. Nunes, A. Grilo, and M. Macedo, “Interference-Free TDMA Slot Allocation in Wireless Sensor Networks,” in *LCN ’07: Proceedings of the 32nd IEEE Conference on Local Computer Networks*, 2007, pp. 239–241.

[16] Y.-C. T. Fang-Jing Wu, “Distributed wake-up scheduling for data collection in tree-based wireless sensor networks,” vol. 13, no. 11, pp. 850 –852, November 2009.

[17] A. Woo, T. Tong, and D. Culler, “Taming the underlying challenges of reliable multihop routing in sensor networks,” in *Proc. of ACM SenSys*. ACM, 2003, pp. 14–27.

[18] N. Burri, P. von Rickenbach, and R. Wattenhofer, “Dozer: ultra-low power data gathering in sensor networks,” in *Proc. of IEEE IPSN*. IEEE Computer Society, 2007, pp. 450–459.

[19] W. Bober, C. Bleakley, and X. Li, “TrickleTree: A Gossiping Approach To Fast Staggered Scheduling For Data Gathering Wireless Sensor Networks,” in *4th Int. Conf. on Sensor Technologies and Applications (SENSORCOMM)*. IEEE Computer Society, 2010, pp. 214–219.

[20] A. Varga. (Last accessed: 04/2010) OMNeT++. http://www.omnetpp.org.

[21] T. Rappaport, *Wireless Communications: Principles and Practice*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[22] A. Iyer, C. Rosenberg, and A. Karnik, “What is the right model for wireless channel interference?” in *QShine ’06: Proceedings of the 3rd international conference on Quality of service in heterogeneous wired/wireless networks*. New York, NY, USA: ACM, 2006, p. 2.

[23] (Last accessed: 04/2010) CC2420 Data Sheet. http://www.ti.com. Texas Instruments.

[24] J. Polastre, J. Hill, and D. Culler, “Versatile low power media access for wireless sensor networks,” in *Proc. of ACM SenSys*. ACM, 2004, pp. 95–107.

[25] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, “TAG: a Tiny AGgregation service for ad-hoc sensor networks,” *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 131–146, 2002.

[26] L. Paradis, “Tigra: Timely sensor data collection using distributed graph coloring,” Master’s thesis, Colorado School of Mines, 2007.