TrickleTree: A Gossiping Approach To Fast Staggered Scheduling For Data Gathering Wireless Sensor Networks

Wojciech Bober, Xiaoyun Li, Chris Bleakley
UCD Complex & Adaptive Systems Laboratory
UCD School of Computer Science and Informatics
University College Dublin
Belfield, Dublin 4, Ireland
wojciech.bober@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 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 the neighborhood discovery and scheduling steps. To reduce message overhead, TrickleTree uses adaptive gossiping. The behavior of the proposed approach is evaluated in simulation. The results show up to 90% reduction in schedule setup time and a 50% reduction of duty cycle compared to a flooding approach.**

I. Introduction

The promise of cheap sensors deployed at large scale is attractive for areas such us microclimate research [1], habitat monitoring [2], and precision agriculture [3]. 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 [4], [5], so far using a battery is the most common method of powering nodes. Dutta et al [6] have shown that radio operation is the main cause of power consumption. Therefore, communication protocols which reduce radio on-time are crucial to 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 was 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 tree are active at any given time. All nodes in the network must be aware of the maximal tree depth in order to schedule their communication correctly. You should note that quality of wireless links can change considerably in a short amount of time [7]. 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 preponderate the cost of data transmission. We address this by proposing TrickleTree, a protocol designed to establish staggered schedule fast, yet with a small message overhead.

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 to resolve 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 staggered communication schedule i.e. assign time slots to nodes, 3) and to disseminate the maximal tree depth in the network.

II. Related work

Staggered scheduling was first introduced in D-MAC by Lu et al [9]. D-MAC staggers transmission in very small time slots thus reduces latency. A similar concept is used in Merlin [10], a cross-layer protocol integrating MAC and routing. In Merlin staggered scheduling is used for up- and down-link traffic. In ASLEEP [11] staggered scheduling is used for adaptive sleeping to reduce network duty cycle and save energy. In [8] staggered scheduling was combined with a synchronous low power listening to reduce energy consumption in low rate data collection networks. Fig. 1 illustrates the concept of a staggered schedule implemented in Bailigh [8]. The network is organized in a tree. Each node has exactly one parent, which

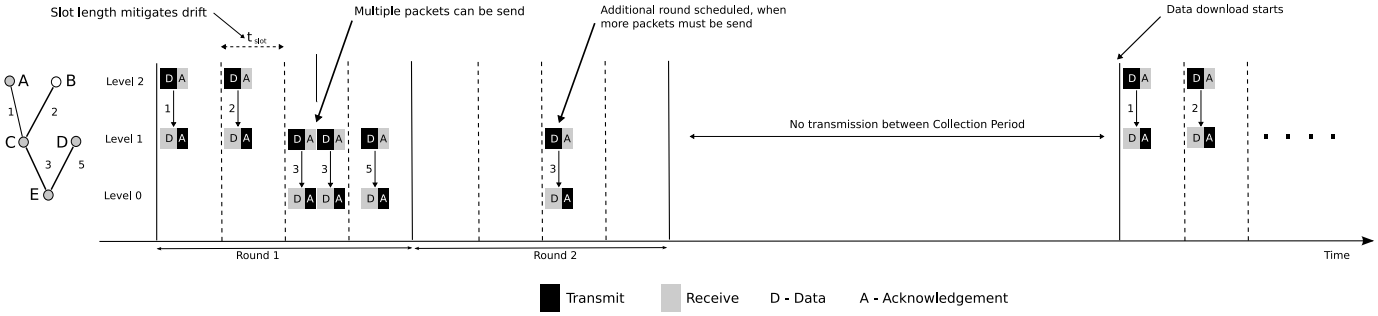

Fig. 1. Staggered communication scheme used in [8].

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 given 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, it may happen that links 1 and 4 sometimes 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 hidden terminal problem in present in staggered scheduling.

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 [12]. 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 [13] 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 [14], [15]. TrickleTree is a modification of Trickle used to construct a staggered scheduling tree. We use the 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 is technique is often used in TDMA MAC protocols [16], [17]. Fang-Jing Wu [18] shows how collision free scheduling can be taken advantage of tree based networks.

III. Proposed 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. A JREQ packet contains the same fields. JREP contains the slot number assigned by the parent to the child.

A. Beacon Dissemination

Beacon dissemination based on gossiping is key to the proposed approach serving multiple purposes. In principle, Trickle adjusts frequency of information dissemination depending on its consistency. When information in the network is consistent (eg. 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 (eg. binary code) from version information. In TrickleTree only small beacons are broadcasted thus there is no separation between metadata and data. 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

TABLE I
Symbol definitions

Symbol	Description	Unit
t	beacon timer	s
τ_l	low gossiping period	s
τ_h	high gossiping period	s
$\tau(j)$	join slot	s
c	beacon counter	-
l	node level	-
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] t is a random value from the range $[\frac{\tau}{2}, \tau)$

with consistent tree depth (d) was received ($c > 0$). This is a simple method of detecting collision due to a hidden terminal: since the node broadcast 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 was received ($c = 0$). This is different from the original 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 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.

B. Joining Scheme

The purpose of the slot assignment scheme is to assign individual transmission slots to each node in the network.

1) Joining network: A node can be in one of four states: Disconnected, Listening, Joining, Collision or Gossiping. The finite state machine of TrickleTree is shown in Fig. 2. Initially a node starts in a Listening state and waits for BCN packets. Upon receiving the first BCN packet the node will set its network beacon timeout to a random value. This allows the node to gather more information about its neighborhood before performing a join attempt. The information from the received BCN packet is stored in the neighbor table as described. After the join timer expires or a sufficient number of BCN packets from distinctive nodes is received, the node selects the best candidate for a parent. TrickleTree uses Shortest Path

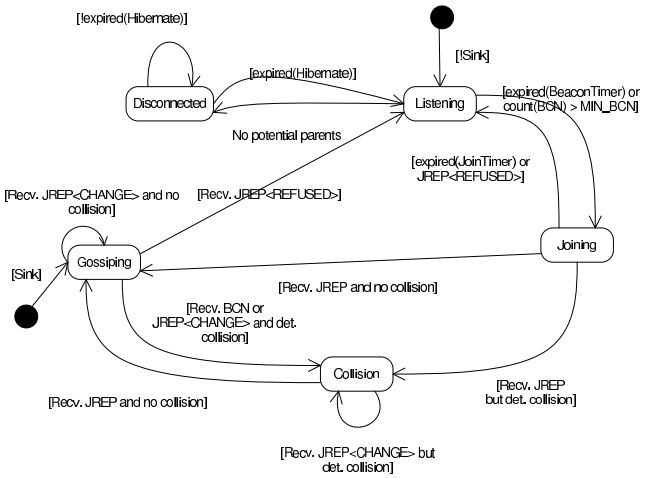

Fig. 2. TrickleTree finite state machine diagram. PACKET<FLAG> notation is used to denote a flag which must be set in a packet for a transition to occur. Expired() and count() notations are used for timers and counters respectively.

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 and more complicated metrics like MintRoute [19] can be used.

Once the node is ready to send JREQ to the selected parent, it calculates a join slot. Fig. 3 shows the TrickleTree join procedure. Each BCN received by a node is a implicit synchronization point. Nodes which decide to join the same parent, upon receiving a BCN, calculate a random join slot. Each join slot is long enough to allow for JREQ and JREP exchange as well to mitigate for clock drift. The TrickleTree join sequence reduces the number of collisions which have to be handled at the MAC layer, thus reducing network setup time. If the JREQ packet is delivered successfully, the node will set up a JREP timer. 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 replies. In the case of delivery failure, JREP timeout, or join rejection, the node will return to Listening state.

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 slot numbers assigned

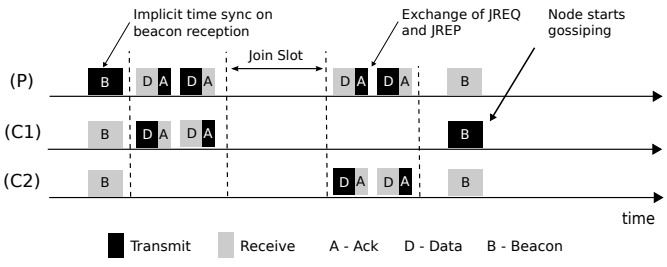

Fig. 3. TrickleTree Join Procedure. (P) denotes a potential parent node, whereas (C1) and (C2) denote child nodes.

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-B4). 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.

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 a new random slot from slots which are not marked as used. 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.

In order to create a data gathering tree the scheme starts with the sink node, which broadcasts a BCN packet. Gradually all nodes in the network join the tree and the topology is established.

2) Forcing join: It might happen that a node has only one potential parent in its neighborhood. In this case, the node can force join with the selected node. The node sends a JREQ with FORCE flag set. Upon receiving this packet, the parent, will select one of its children and remove it by sending JREP with 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 join ensures that the whole network is connected as long as good quality links are available.

3) Detecting collisions: As mentioned previously TrickleTree can work in Collision Free (CF) mode. 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. The same 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-B4. If a node is a parent to any node, 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 a two hop neighborhood information to assign individual slots. This method is often used by TDMA MAC protocols [16], [17], [20] to handle collisions from hidden terminals. In TrickleTree collisions from hidden terminals are detected and resolved by either parent or child, what is described in detail in the next section.

4) Resolving collisions: The method of resolving a detected collision depends on the node's role. If the node is a child node, the collision is solved by requesting a new slot from its parent. This is done by sending a JREQ packet to the current parent. If the node is a parent node, 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. In the case that a new slot could not be assigned, the child node is disconnected from a given parent and a JREP packet with REFUSED flag is send back. This forces the child node to connect to a different parent. During this process, a node might connect to a parent which has a different distance from the sink than the former parent node. Since all children must know their exact distance from the sink in order to calculate their communication offset properly, the information must be updated. This is done during BCN packet dissemination after the node successfully connects to a new parent. Each node which receives a BCN packet from its parent compares its current level with the level stored by the BCN packet and updates it if the values are different. 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.

IV. Evaluation

A. Simulation Model

In order to verify the behavior of the proposed algorithm, a set of simulations were performed using the OMNeT++ [22] simulator. Wireless signal propagation is modeled using a Log-Normal Shadowing Model [23]. To capture the effect of interference, we used an additive interference model [24]. The radio is modeled after the CC2420 transceiver [21]. The model uses a finite state machine with idle, receive and transmit states to model transition delays and the power consumption in respective states.

Each simulation was repeated 200 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 booted with randomized startup times, with 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, and no schedule collisions exist. TrickleTree can use any MAC protocol. Herein, the protocol is evaluated with B-MAC [25], which uses Low Power Listening to reduce power consumption. Due to its low control overhead, B-MAC is often selected as a deployed phase protocol where network topology is not yet established [26].

B. Results

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

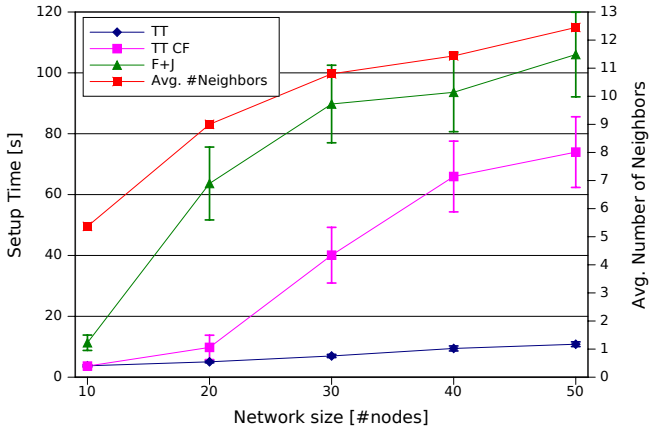

Fig. 4. Tree setup time as function of network size. TT denotes TrickleTree algorithm, whereas F-J denotes the flooding join algorithm.

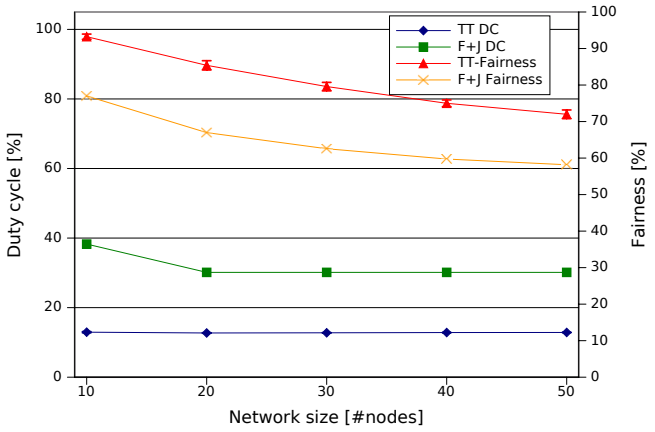

Fig. 6. Duty cycle as function of network size. TT denotes TrickleTree algorithm, whereas F-J denotes the flooding join algorithm.

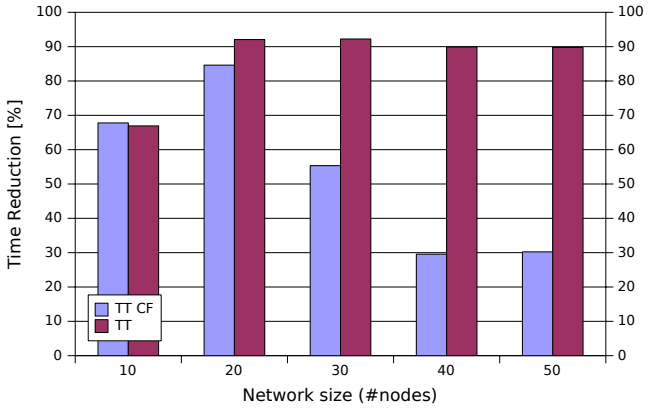

Fig. 5. Network setup time reduction in relation to flooding join.

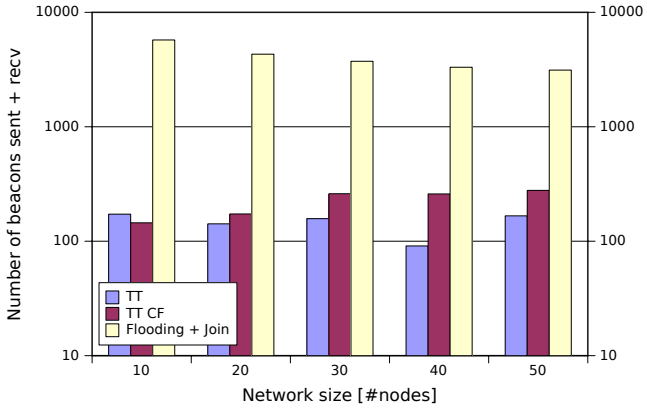

Fig. 7. Sum of beacons sent and received for various network sizes.

setup time. Adaptive gossiping is used to reduce message overhead and ensure that changes in tree depth are disseminated quickly. To verify TrickleTree performance we have compared it with a flooding approach, similar to that used in [27]. In the flooding approach, the sink periodically broadcasts a tree setup beacon. We have performed a number of experiments to establish what broadcasting period achieves the best results for the flooding approach. The results showed that period of 0.7s provides the fastest networks setup in most cases. If not all nodes could join the network in given simulation run, the results were rejected and not included in calculations. Upon receiving the beacon, unconnected nodes attempt to join selected parent. The flooding approach relies on the MAC protocol to resolve collisions. Nodes which are connected to the tree are allowed to rebroadcast beacons. Fig. 4 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 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 Fig. 7 which shows average number of beacons sent and received. It can be seen that collision free version of TrickleTree requires up to 65% more beacons to be sent. This is beacuse each change in a schedule (eg. slot change) resets the gossiping period to the lowest value, therefore beacons are sent more frequently. Fig. 7 shows an advantage of adaptive gossiping of 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. Fig. 5 shows time reduction of regular and collision free version of TrickleTree with respect to the flooding approach. The advantage of TrickleTree is more distinct as the network size is increased. In the flooding approach large number of nodes produces high degree of collisions what slows down schedule setup time. As indicated previously, the collision free

version of TrickleTree requires more time to setup the network. It is a tradeoff between fast schedule setup time and possible collisions during data gathering and slower schedule setup but collision free transmissions. The decision which version of TrickleTree should be used depends on network stability. For instable 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 of setting up collision free schedule will be balanced in long run by collision free data gathering phase.

In Fig. 6 the duty cycle of both protocols is shown. The average duty cycle of TrickleTree over different network sizes is 12%, where as the duty cycle of the 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 longer network setup time. The fairness for both protocols is also shown in Fig. 6 confirms that the main reason for TrickleTree performance is reduction of collisions. In general, the improvment of network setup time depends on a network size. TrickleTree reduces setup time by 68% for network of 10 nodes, to nearly 90% for a network of 50 nodes.

V. Conclusions

Data gathering is one of the most interesting applications for wireless sensor networks. As the 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 in 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 of duty cycle compared to flooding approach

Acknowledgment

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*, 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*, 2004, pp. 214–226.

[3] R. Musaloiu-E, A. Terzis, K. Szlavecz, A. Szalay, J. Cogan, and J. Gray, “Life under your feet: A wireless soil ecology sensor network,” in *Proc. 3rd Workshop on Embedded Networked Sensors (EmNets 2006). May*, 2006.

[4] C. Alippi and C. Galperti, “An adaptive system for optimal solar energy harvesting in wireless sensor network nodes,” *IEEE Trans. Circuits Syst. I*, vol. 55, no. 6, pp. 1742–1750, July 2008.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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 Netw.*, vol. 6, no. 8, pp. 1238–1257, 2008.

[11] 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.

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

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

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

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

[16] 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.

[17] I. Rhee, A. Warrier, J. Min, and L. Xu, “DRAND: distributed randomized TDMA scheduling for wireless ad-hoc networks,” in *Proc. of ACM MobiHoc*. New York, NY, USA: ACM, 2006, pp. 190–201.

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

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

[20] L. F. W. van Hoesel and P. J. M. Havinga, “A lightweight medium access protocol (lmac) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches,” in *1st International Workshop on Networked Sensing Systems (INSS, Tokio, Japan*, Tokio, Japan, 2004, pp. 205–208.

[21] CC2420 Data Sheet. Texas Instruments.

[22] A. Varga. Omnet++.

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

[24] 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.

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

[26] A. Meier, M. Woehrle, M. Weise, J. Beutel, and L. Thiele, “NoSE: Efficient Maintenance and Initialization of Wireless Sensor Networks,” in *Proc. of IEEE SECON*, June 2009, pp. 1–9.

[27] 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.